People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

Abderrahmane Mira University, Bejaia

Faculty of Exact Sciences

Department of Computer Science

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Practical Work Booklet

Module: Software Engineering

For use by third-year undergraduate students in Mathematics and Computer Science (MCS), National Recruitment (NR) and higher education establishments.

Established by Dr. Mohamed Mohammedi

Version 1

2024/2025

Preamble

This software engineering lab booklet has been carefully designed to enrich the learning experience of third-year undergraduate students. By combining theory and practice, it aims to transform the knowledge learned in class into concrete, applicable skills.

Based on in-depth reflection and significant teaching experience, this document is perfectly aligned with the official curriculum. It aims to act as a true bridge between academic concepts and the realities of the professional world, accessible to the entire university community.

The labs are organized into carefully chosen thematic series, offering a diverse range of activities that explore the multiple facets of software engineering. Each activity is accompanied by sample answer keys, allowing you not only to assess your progress but also to encourage an independent and thoughtful approach to learning. In addition, practice exercises are included, with detailed solutions for the most significant cases, to stimulate your intellectual curiosity and engagement.

I sincerely hope that this booklet will be a source of inspiration and practical tools in your academic journey, guiding you towards excellence in the field of software engineering. May this learning adventure be rich in discoveries and success!

Table of Contents

Foreword

Knowledge is a treasure, but practice is the key.

This practical workbook (PW) was developed to support Software Engineering (SE) students in their learning of Java programming. It is structured into several sections, each aimed at developing specific skills and strengthening understanding of fundamental concepts.

Lab 1 focuses on installing and configuring the NetBeans development environment, providing a solid foundation for the following assignments: Lab 2 provides refresher exercises on the Java language, allowing students to review essential concepts and ensure they are comfortable with the syntax and structures of this language. Lab 3 covers object-oriented programming (OOP), with an introduction to key concepts followed by practical exercises, allowing students to master the principles of encapsulation, inheritance, and polymorphism. Lab 4 focuses on GUI programming, offering exercises for developing interactive and user-friendly applications. Lab 5 introduces the Three Types of Software Architecture MVC (Model-View-Controller), Three Tiers and Service-Oriented, essential for the development of well-structured applications. Practical exercises complete this section to reinforce the understanding of the roles of each component in the architecture. Lab 6 covers the transition from domain model to relational model, a crucial skill for database design. Lab 7 focuses on connecting to a MySQL database via Java with NetBeans, illustrating how to integrate data into Java applications. Finally, a mini-project allows students to implement all the skills acquired throughout the labs.

The general conclusion summarizes the booklet's learnings and highlights the importance of these skills in modern software development. A bibliography is also provided for those wishing to deepen their knowledge.

This booklet is a comprehensive and structured tool for anyone wishing to familiarize themselves with Java development and prepare for practical projects.

Lab 1 : Netbeans development environment

Outline	1.1 Introduction 1.2 Configuration of the Java environment: JDK, JRE and NetBeans 1.3 Initiation to NetBeans 1.4 Project notion 1.4.1 Creation of a project 1.4.2 Structure of a project 1.4.3 Creation of a class 1.5 Compilation 1.6 Launch of a program 1.7 Debugger 1.8 Packages 1.9 Exercises 1.10 Exercise Corrections
Goals	➢ Understand the importance of NetBeans IDE for Java development; ➢ Familiarize yourself with the interface of NetBeans and learn to create and manage projects; ➢ Learn to use the beginning to identify and correct errors in the code; ➢ Acquire basic knowledge on Java packages and their management in NetBeans; ➢ Understand the structure of a project and the files necessary for the development of Java applications; ➢ Apply this knowledge through practical exercises to consolidate learning.

1.1 Introduction

The NetBeans integrated development environment (EDI abridged in French or IDE in English, for Integrated Development Environment) is a fundamental tool for developers, in particular those working with Java. Designed to simplify the development process, NetBeans offers a user -friendly interface and a rich set of features that facilitate the creation, debugging and management of software projects.

This IDE is particularly appreciated for its ability to integrate several essential tasks in a single environment, allowing developers to focus on the logic of the code rather than the management of tools. In this initiation, we will explore the different facets of NetBeans, in particular how it helps to structure the code and improve productivity.

1.2 Java environment configuration: JDK, JRE and NetBeans

This section will guide you through the steps to start using Java. We will focus on the download and installation of NetBeans. Before proceeding, make sure that the JRE and the JDK are installed on your computer. The JRE allows Java programs outside the IDE, while JDK is used to compile your code. The steps to follow are as follows:

Step 1: Download the JDK

Download the Java SE Development Kit (JDK) by choosing the appropriate version for your system:

- 64-bit JDK: [Télécharger JDK 64 bits](https://www.numerama.com/telecharger/java-se-development-kit-jdk-28668-telechargement.html)
- 32-bit JDK: [Télécharger JDK 32 bits](https://www.01net.com/telecharger/windows/Programmation/javaapplet/fiches/270.html)

Step 2: Download the JRE

Download the Java Runtime Environment (JRE) according to your system:

- 64-bit JRE: [Télécharger JRE 64 bits](https://filehippo.com/download_jre_64/)
- 32-bit JRE: [Télécharger JRE 32 bits](https://www.01net.com/telecharger/windows/Programmation/java/fiches/8138.html)

Step 3: Download NetBeans

To download NetBeans, go to the official Apache NetBeans website and choose the version adapted to your operating system. It is important to note that NetBeans is not portable software and generally requires installation. If you have opted for a compressed version, you will have to extract the folder and run the **netbeans.exe** file in the directory created. During your first use, a window will open to allow you to define your workspace, or a default workspace will be generated automatically. Make sure to follow the instructions specific to your version for a successful installation.

- 64-bit NetBeans: [Download NetBeans 64 Bits] (https://telecharger.tomsguide.fr/netbeans-ide,0305-45313-2132.html)
- 32-bit NetBeans: [Download NetBeans 32 Bits] (https://en.freedownloadManager.org/users-choice/netbeans_8.2_32_bit_download.html)

During our exploration of NetBeans, we will start by familiarizing ourselves with its interface and basic features. You will discover how to create a new project, navigate the IDE and use the tools available to write code. This recovery is essential to get the most out of the environment.

One of the major assets of NetBeans is its beginning, which allows developers to identify and correct errors in their code effectively. Thanks to features such as monitoring the execution of the code and the analysis of the values of the variables, the beginning offers a precious overview of the behavior of the application. It makes it possible to define stop points and explore the state of the application at different times of its execution, which is crucial to ensure the robustness of the code.

To maximize the use of NetBeans, a basic understanding of Java is necessary, in particular with regard to packages. These groups of classes and interfaces facilitate the organization and modularity of the code. We will approach how to create and manage packages, which is essential to effectively structure development projects, especially when they become more complex.

1.3 Initiation to NetBeans

NetBeans is an IDE that centralizes all the tasks necessary for the development of applications. This not only includes code writing, but also file management, compilation, debugging and deployment. A good FDI must offer tools that improve productivity and code quality.

There are several competitors on the market, each with its own characteristics and advantages.

- Eclipse: very popular for Java development, it is expandable thanks to many plugins.
- Intellij IDEA: appreciated for its advanced functionalities of coding assistance and its management of several languages.
- VSCODE: A light and expandable code editor that supports many languages and frameworks thanks to its extensions.

NetBeans allows most operations related to software production. In other words, it allows you to carry out a multitude of operations in an integrated manner. This means that you can easily go from one task to another without having to leave the development environment. This integration makes the development process more fluid and more consistent.

Writing, compilation

In NetBeans, code writing is done in a rich text editor which offers features such as syntactic coloring, self-completion and error management. Once the code is written, the compilation is carried out directly in the IDE, with clear error messages which allow the problems quickly.

Note:

In NetBeans, red underlines indicate compilation errors (for example: incorrect syntax, unknown type, missing method) — the code will not compile until these errors are corrected. Orange (or yellow) underlines indicate warnings: the code compiles, but there is a potential problem or bad practice (unused code, deprecation, unnecessary casting, violated conventions) that should be corrected or investigated.

Test

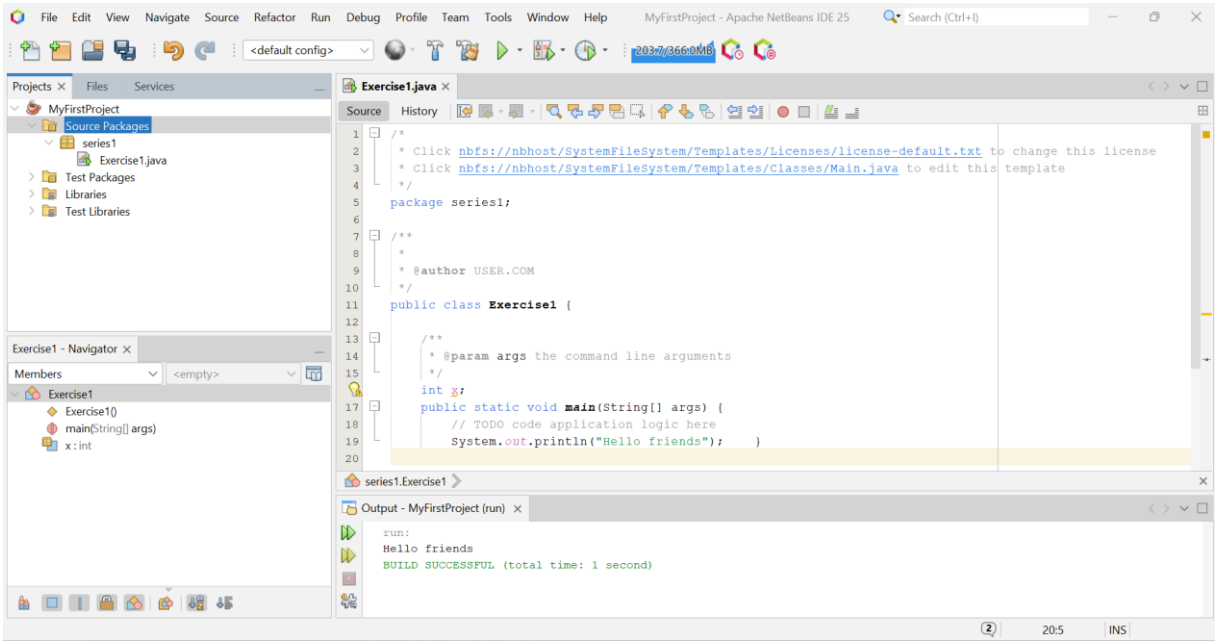
NetBeans also facilitates the test process thanks to integrated tools to perform unit and integration tests. You can configure test frameworks like Junit to ensure that your code works as expected before its publication.

Publication

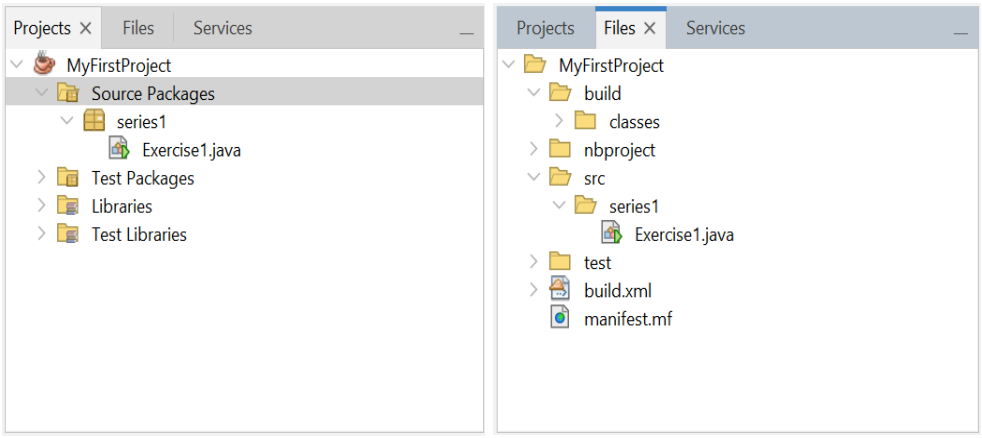
The IDE offers features for the publication of your application, whether on a local server or on the cloud. You can easily manage deployment configurations and generate executable packages, thus making the production of your apps simpler and faster.

Presentation of the NetBeans interface

NetBeans allows, thanks to different tabs, the simultaneous visualization of the project and its tree structure, and the content of a class.

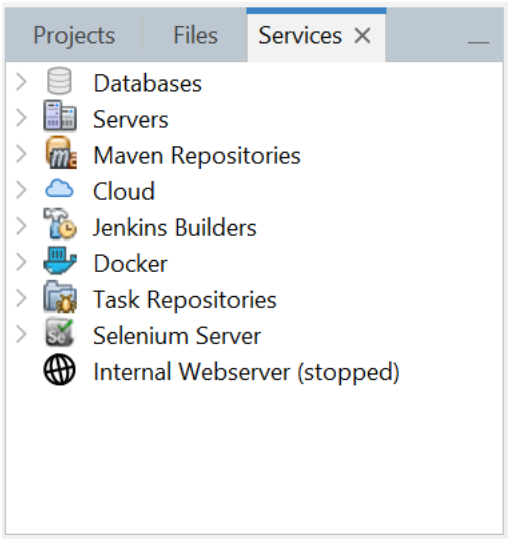


- The **Projects** and **Files** tabs, located at the top left of NetBeans, present the different repertoires of the project. The Projects tab brings together the different files by packages while the Files tab presents the folders and files in a hierarchical way, that is to say as they are saved on the user position.



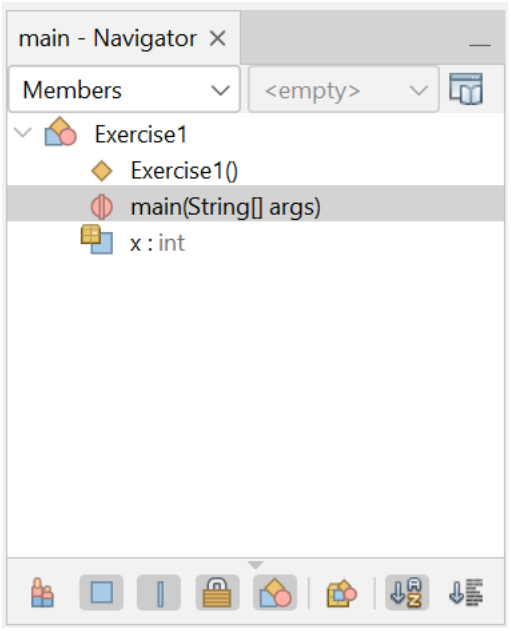
- The **Services tab**, located at the top left of NetBeans, has an essential functionality which allows developers to manage and interact with various services and resources of their project. Here is a small description of its main characteristics:
    1. Database management: You can connect, explore and manage databases. The tab displays active connections and allows you to run SQL queries.
    2. Application servers: It allows you to manage application servers, such as Apache Tomcat or Glassfish, by facilitating the deployment and configuration of web applications.
    3. Web services: You can explore and test web services, including REST and SOAP APIs, directly from IDE.
    4. Version management: Integration with version control systems, such as Git, to manage code deposits.
    5. Configuration of projects: Access to the specific configurations of projects, which simplifies the management of dependencies and libraries.

    This tab centralizes useful features for development, making the process more fluid and effective.

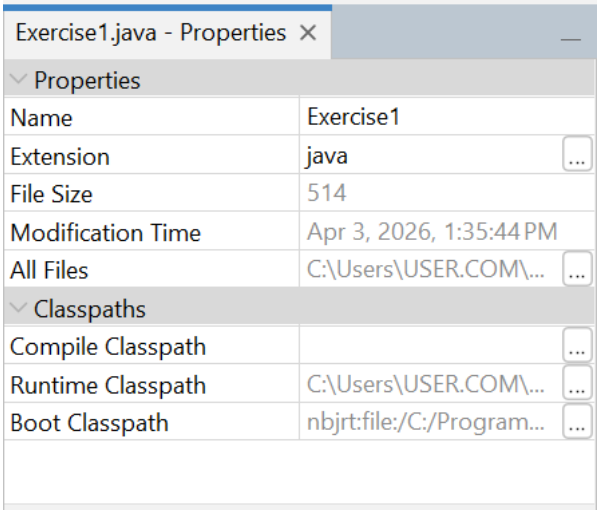


- The **Navigator tab**, located at the bottom left of the IDE, lists the various attributes and methods created in the open class in NetBeans. A double click on an attribute or on a method on the list of this tab positions the cursor on the first line of this attribute or

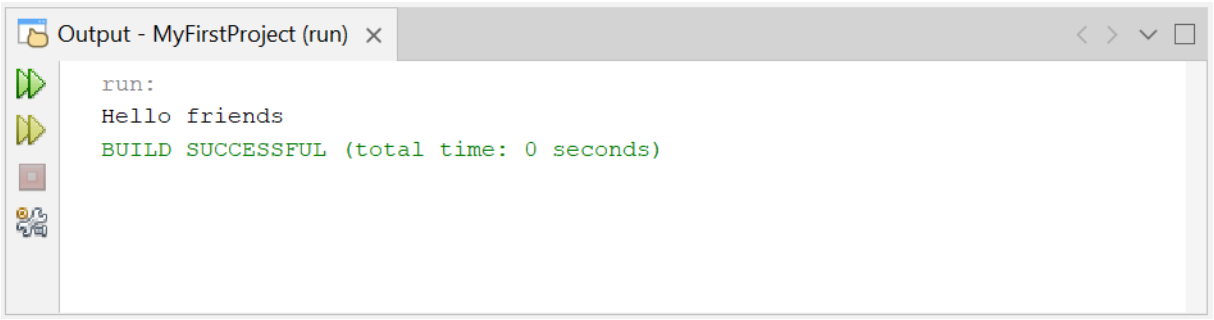
method in the Java class. This can be very useful for accessing code in a class containing a large number of elements.



- The **Properties tab** presents the technical properties of the open class in NetBeans, for example its date of modification, size and location on the developer's position.



- The box located at the bottom of the interface contains: an **Output** tab commonly called **console**, which displays the **project logs**, a **Search tab**.

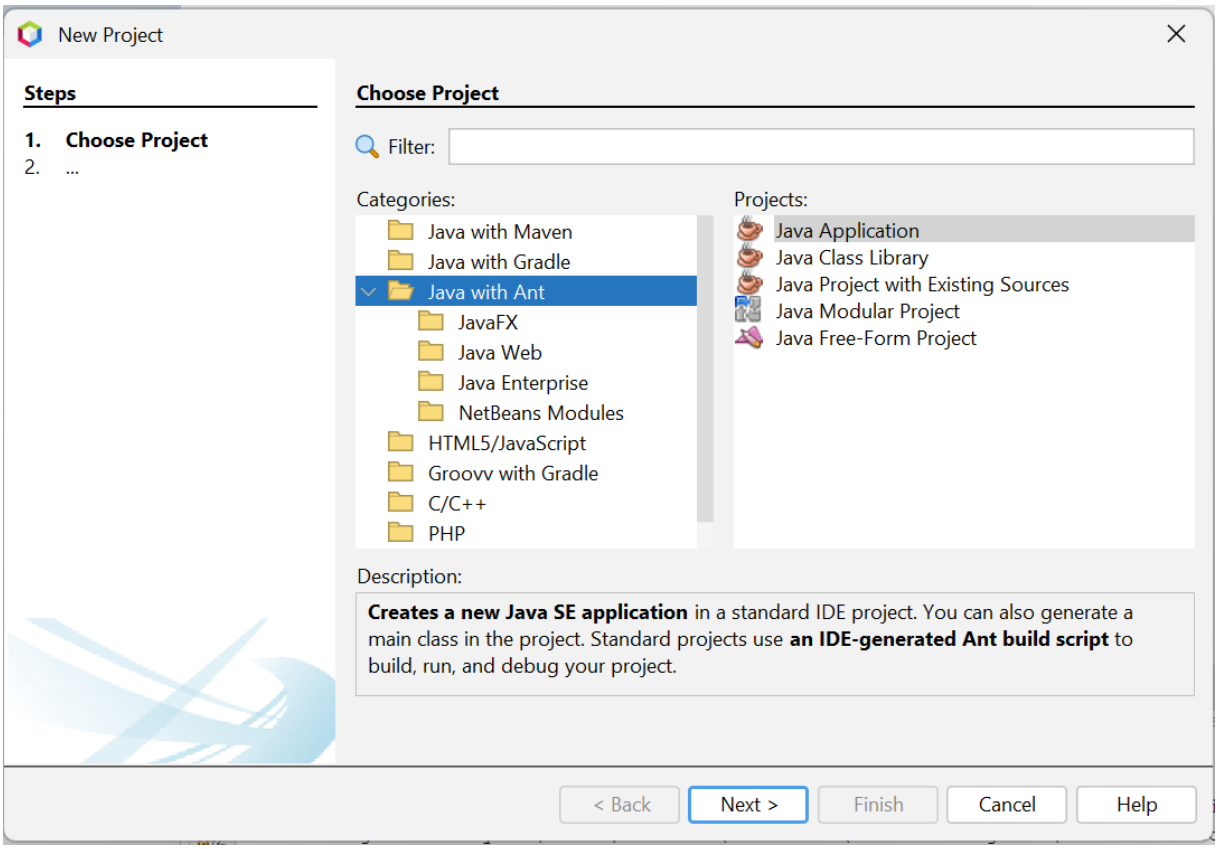


1.4 Project notion

In NetBeans, the concept of project is essential, because it is within projects that all the files necessary for the development of an application are created and organized. A real Java program, by nature, generally contains a large number of files with the ''.java'' extension, each corresponding to a class or an interface. To facilitate the management of these files and maintain a coherent structure, they are grouped within projects. This allows developers to organize themselves better, to facilitate navigation between the different files and to structure their code logically, which is crucial for the effective development of complex software.

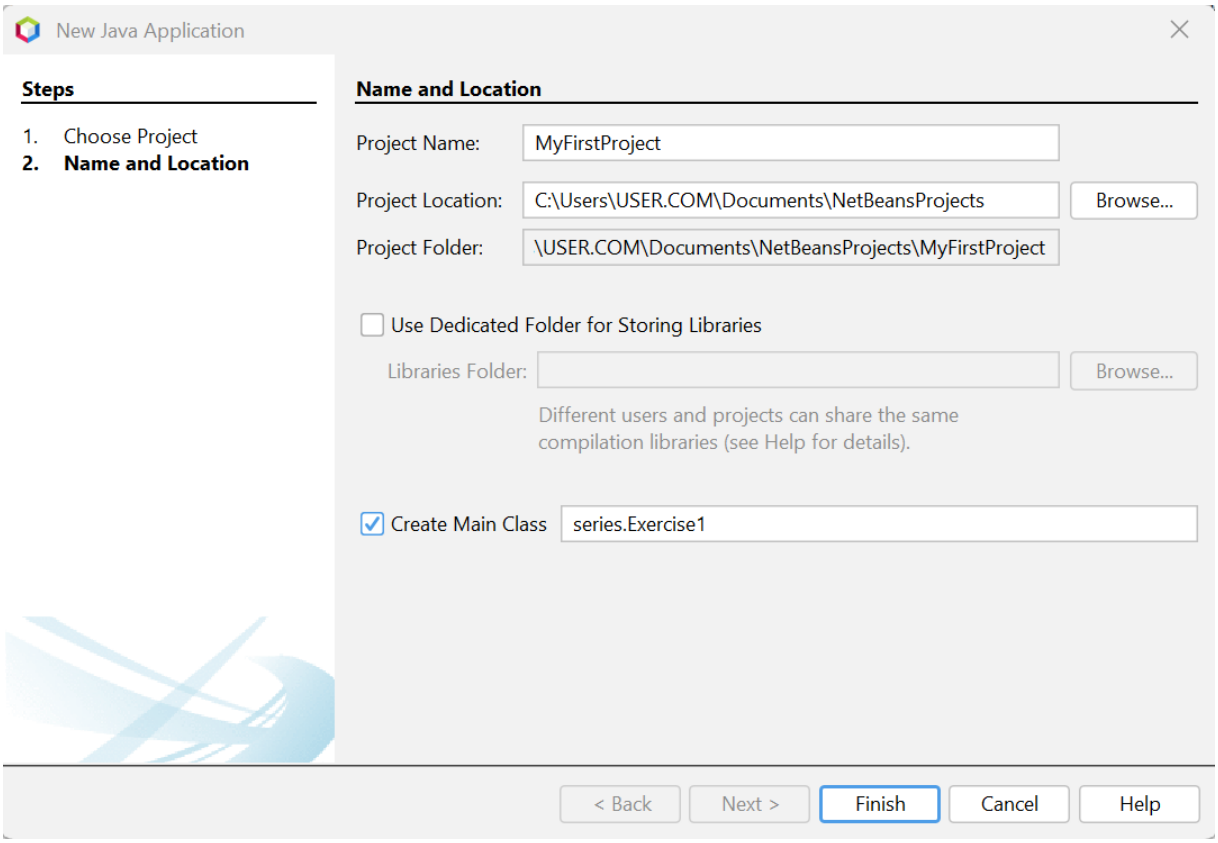
1.4.1 Creation of a project

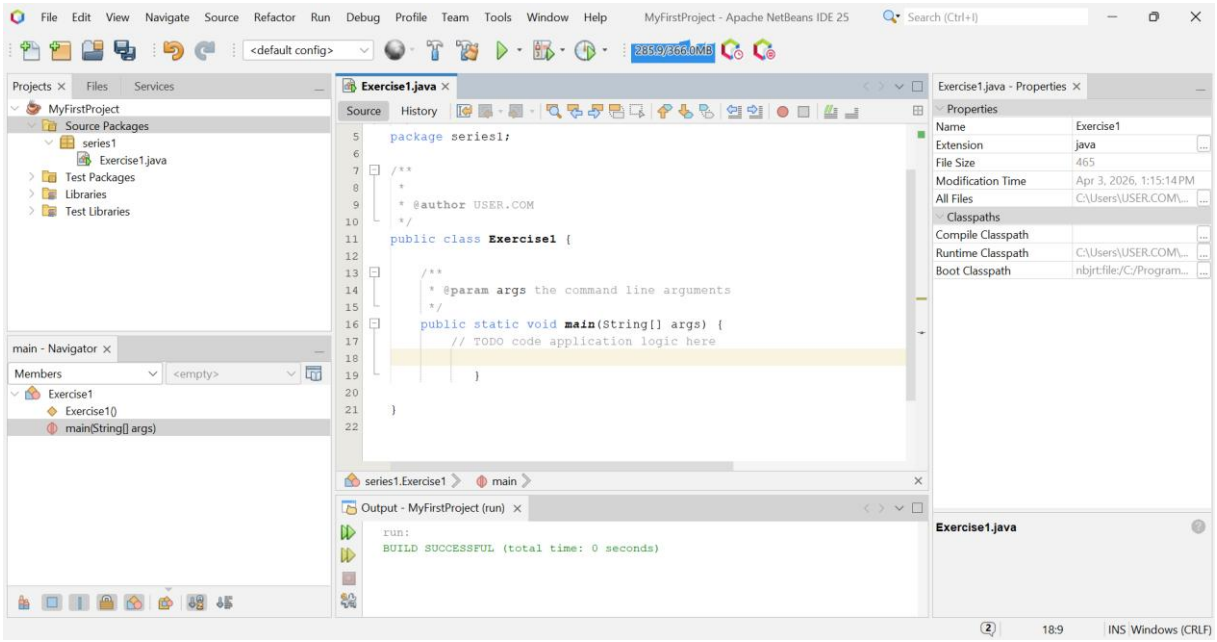
First you have to launch NetBeans, then create a new Java project in which your classes will be. To do this, go to the file menu, new project (or file, New Project). Then, a window is displayed, in this window specify the type of the project to be created. You therefore you in Categories (on the left) the standard type (**Java with Ant**), then in Projects (right) Choose **Java Application**.



Then, click on the Next button, another window is displayed, in which it is necessary to enter the name of the project in the Project Name part, and bring in the name ''**MyFirstProject**'' and unlike Eclipse, in NetBeans the hand class can be created at the same time as the project if the box ''**Create Main Class**'' is checked. Consequently, check this box, generally, in NetBeans, the ''Main Class'' box automatically ticks and the hand -class can also be automatically named

MyFirstProject, if not, you can rename it (series1.Exercise1). Then, click on the finish button. The project is created as well as the hand exercise. Window then on projects.





In NetBeans, the main difference between a project created with ''Java with Maven'' and a project created with ''Java with Ant'' resides in the management of dependencies and the structure of the project.

1. Java with Maven:
    a. Use a ‘’pom.xml ’’ file to manage project outbuildings and configurations.
    b. Allows simplified management of third -party libraries thanks to a system of repositories.
    c. Promotes a standardized project structure, facilitating collaboration between developers.

2. Java with Ant:
    a. Use a ‘’build.xml’’ file for the configuration and management of construction tasks.
    b. Offers more flexibility but often requires more manual configuration for dependencies.
    c. Does not necessarily follow a standard project structure, which can vary from one project to another.

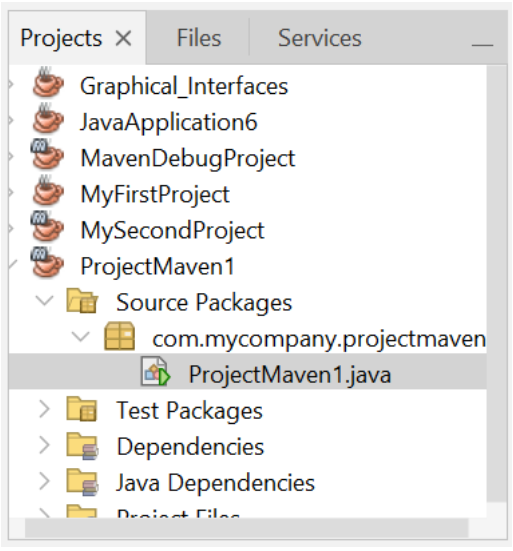
Note:

Maven is often preferred for its simplicity and automation, while it can be suitable for projects requiring further personalization.

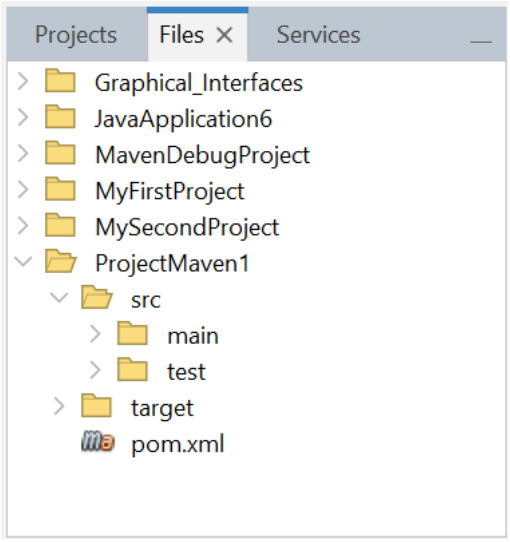
1.4.2 Structure of a project

The structure of a project is organized around a configuration file named ‘’Pom.xml’’, which plays a central role in the management of dependencies and configurations. Inside this structure, a ‘’src’’ folder is dedicated to the storage of project sources, while the files compiled with the ‘’.class’’ extension are automatically generated and placed in the ‘’Target’’ folder. In addition, the NetBeans development environment uses Maven software to orchestrate the compilation process, thus facilitating the management and execution of the project.

Vue logique : onglet Projects

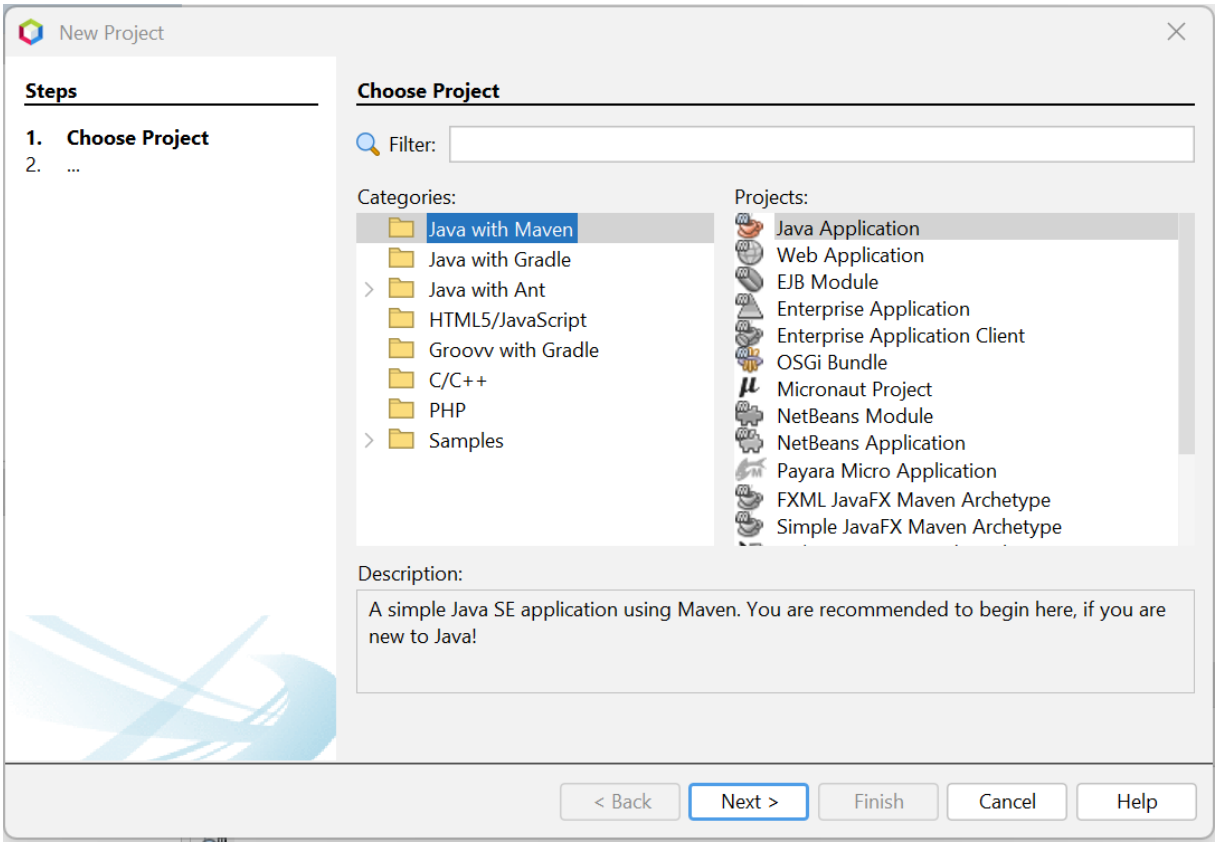


Vue réelle : onglet Files

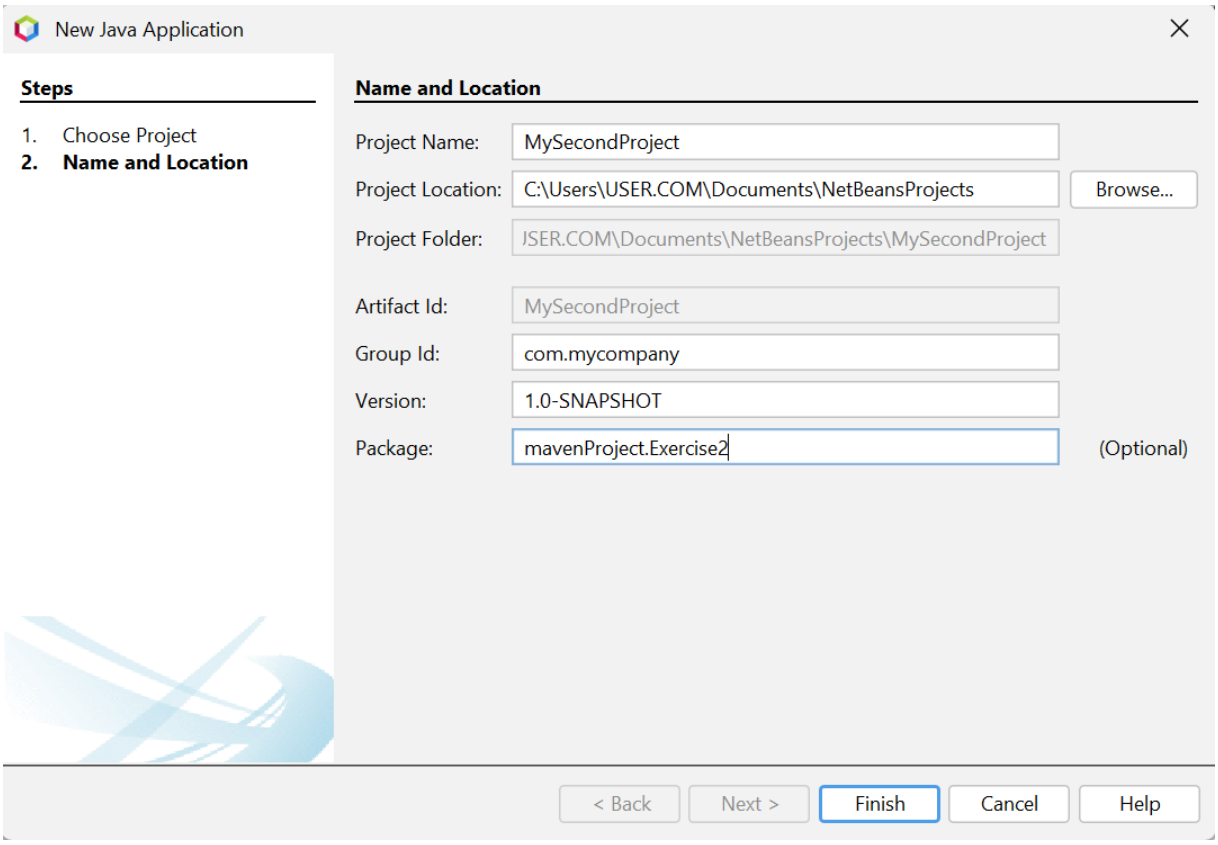


1.4.3 Creation of a class

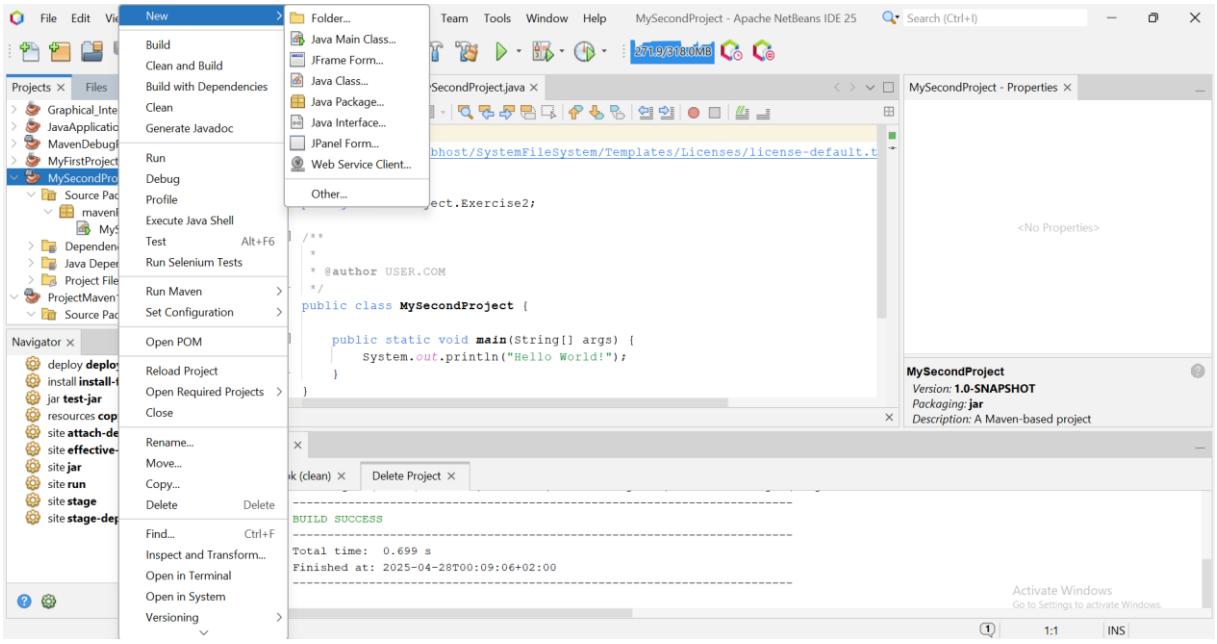
To create a Java class in NetBeans, start by opening the NetBeans application on your computer. Then you need to create a new project by going to the menu ''File'' and selecting ''New Project''. Choose ''Java with Maven'' in the categories, then select ''Java application'' in projects.



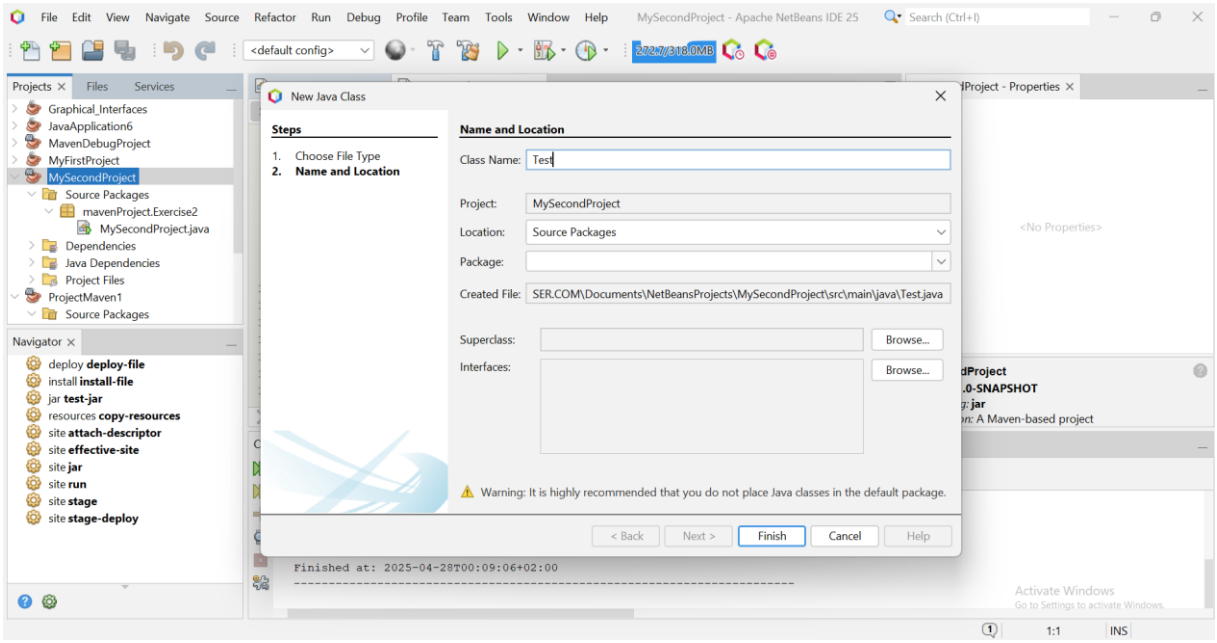
Before clicking on ''Next''. Give your "**MySecondProject**" project a name and choose the location, then click on **Finish**.



With Maven, the main class "**MySecondProject**" is created at the same time as the project. If you want to create another class in the same project once it is created, right click on the ''Source Packages'' file in the tree structure of your "**MySecondProject**". Select ''New'', then ''Java Class''.



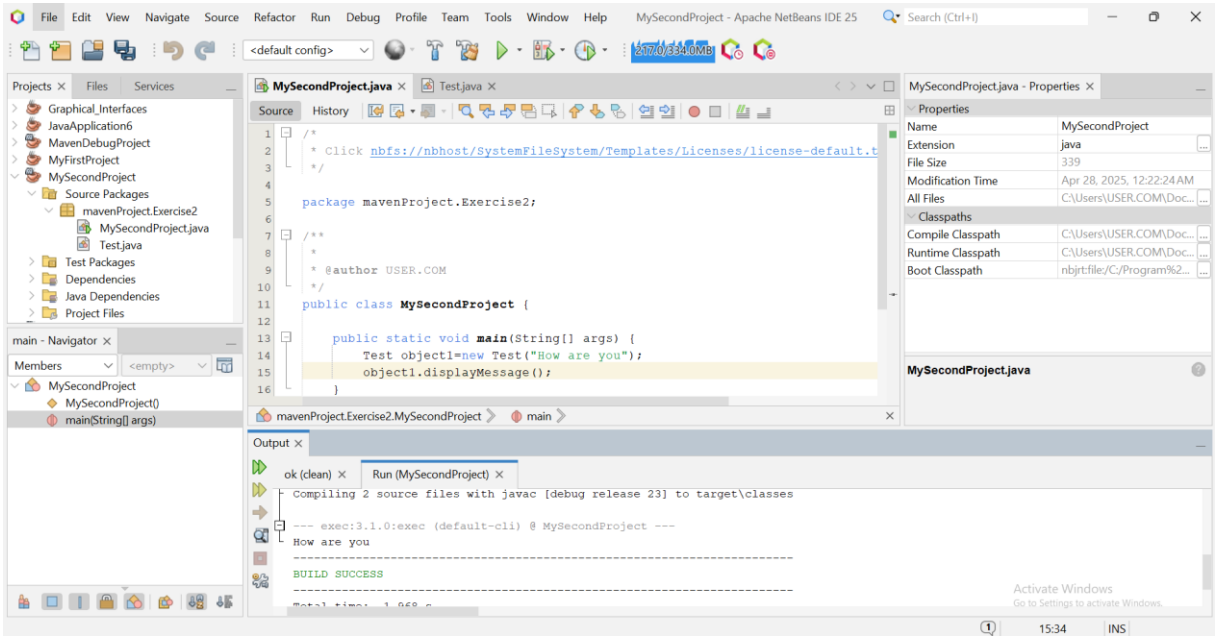
In the dialog box that is displayed, enter the name of your class ''Test'' as well as the package if necessary, then click on ''Finish''. NetBeans will open a code editor with your new class, where you can add attributes, methods and the necessary code.



To illustrate, here is a simple example of a Java class:

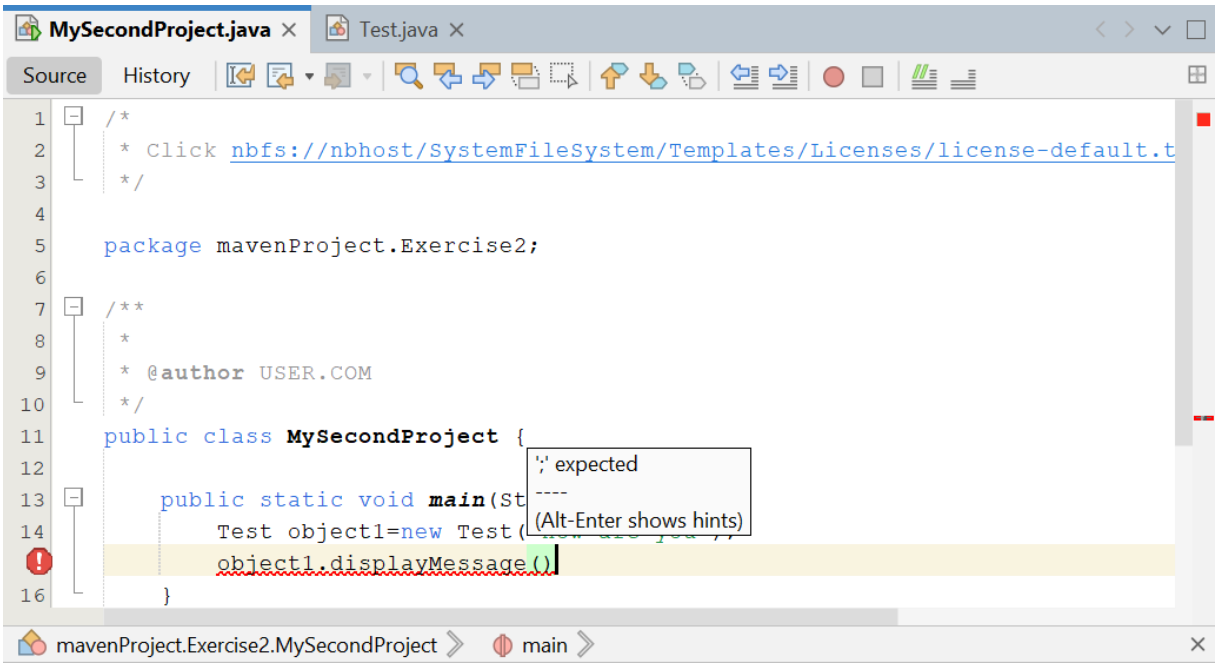
```
package maven.project.exercise2;
public class Test{
 private String message;

 public Test(String message) {
 this.message = message;
 }

 public void displayMessage() {
 System.out.println(message);
 }
}
```

To run your code, you will need a hand method. You can either exploit the main class ''MySecondProject.java'' which already has the Main method, or add it to your existing class. Finally, click the Run button or press ''F6'' to launch your project. Thus, you will have managed to create and execute a Java class under NetBeans with Maven.



1.5 Compilation

In NetBeans, code is compiled **instantly**, allowing for immediate responsiveness. Syntax errors are reported in **real time**, facilitating the development process by allowing developers to correct errors as they occur.



1.6 Launching a program

Once the coding stage is complete, compile your program to translate it into special binary code that Java understands, and then run it. To run a program, simply select the file containing the ''main'' method, then right-click and choose the ''Run File'' option. This launches the program's execution simply and efficiently. Alternatively, you can also click the green arrow at the top. Or, in the context menu, select ''run'' and then ''run file''.

1.7 Debugger

The debugger is an essential tool for viewing variable values during program execution. Basic principles include the ability to set breakpoints. When the program reaches a breakpoint, it pauses, allowing you to examine variables and execute the program in single-step mode, which helps identify and correct problems in a more targeted manner.

Here's a simple example of Java code you can use to understand debugging in NetBeans. This program asks you to enter two numbers and displays their sum. You can add breakpoints to examine variables during execution.

Java Code Example

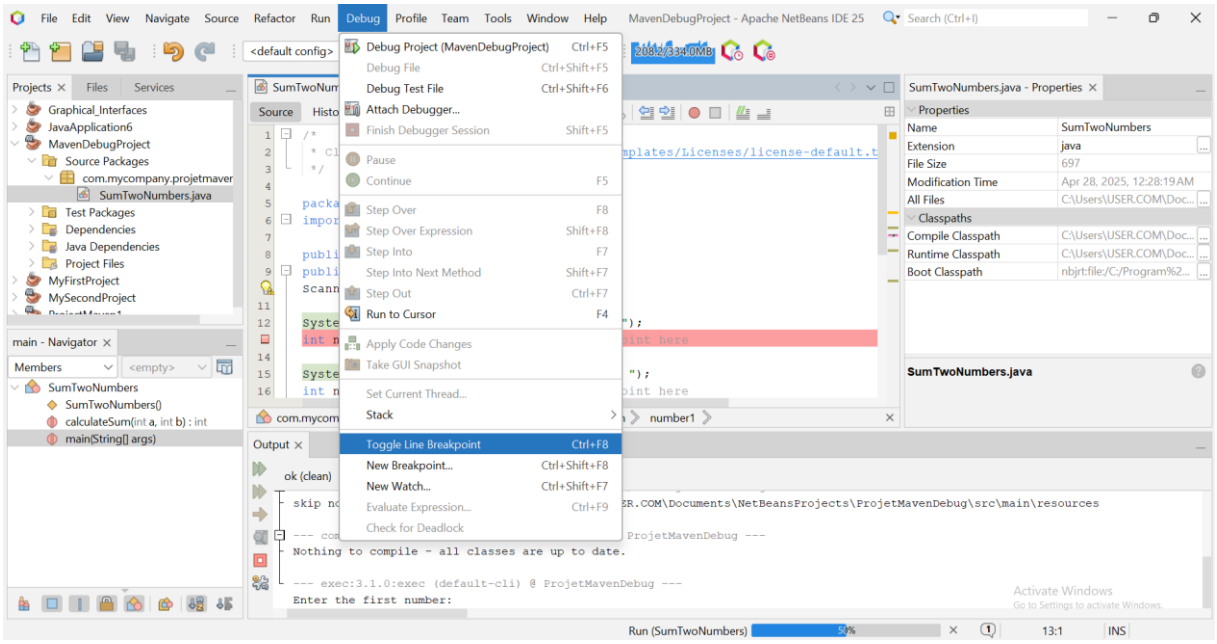
```
package com.mycompany.projetmavendebug;
import java.util.Scanner;

public class SumTwoNumbers {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

System.out.print("Enter the first number: ");
int number1 = scanner.nextInt(); // Breakpoint here

System.out.print("Enter the second number: ");
int number2 = scanner.nextInt(); // Breakpoint here

int sum = calculateSum(number1, number2); // Breakpoint here
System.out.println("The sum is: " + sum);

scanner.close();
}

public static int calculateSum(int a, int b) {
return a + b; // Breakpoint here
}
}
```

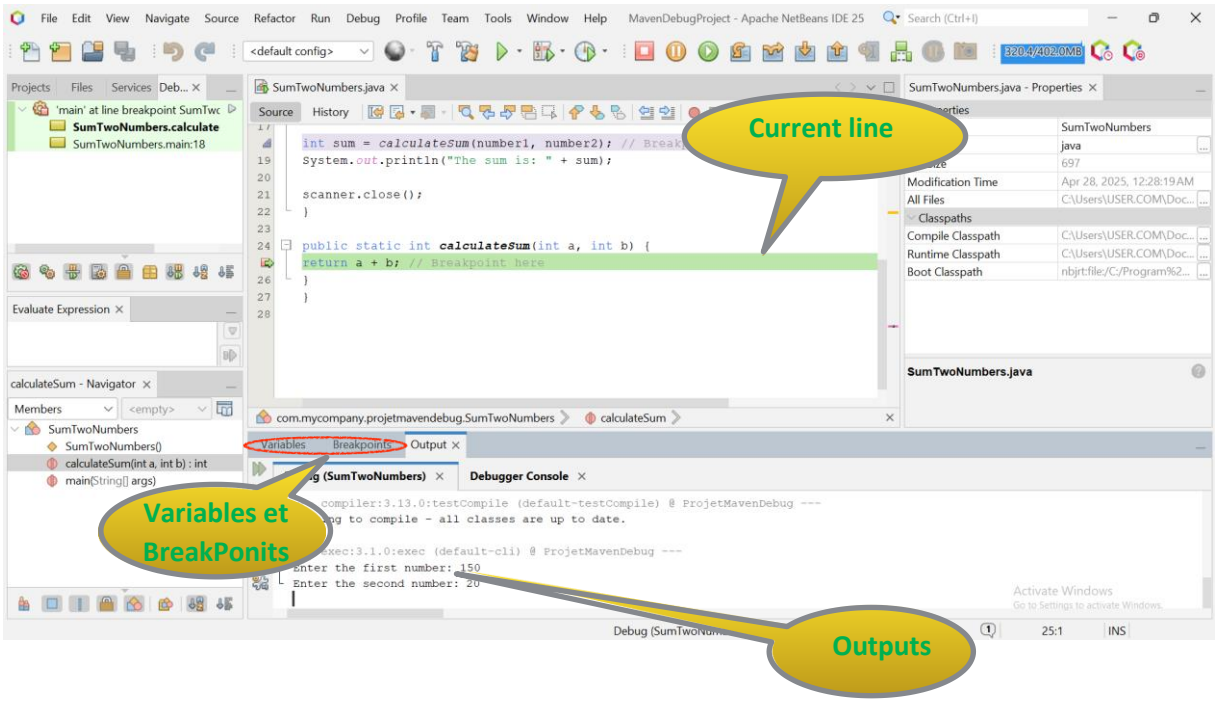
Enter the first number: 12
Enter the second number: 23
The sum is: 35

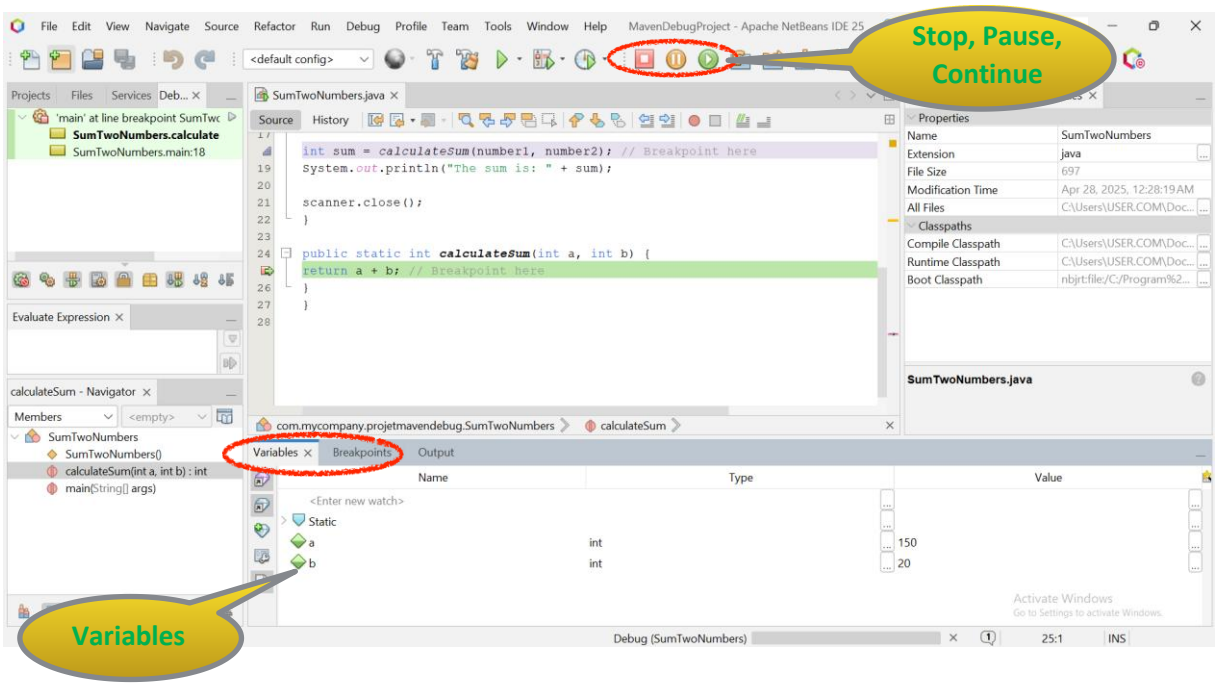
Steps to Debug in NetBeans

1. Open NetBeans and create a new Java project.
2. Copy and paste the above code into the ''SumTwoNumbers.java'' file.
3. Add breakpoints:
   - Click the margin to the left of the line number in the code editor to add a breakpoint (a small red square will appear).
4. Run the debugger:
   - Click the ''Debug'' button (or press ''F5'').
5. Trace execution:
   - When execution reaches a breakpoint, you can examine the variable values in the ''Variables'' or ''Show Memory'' pane.

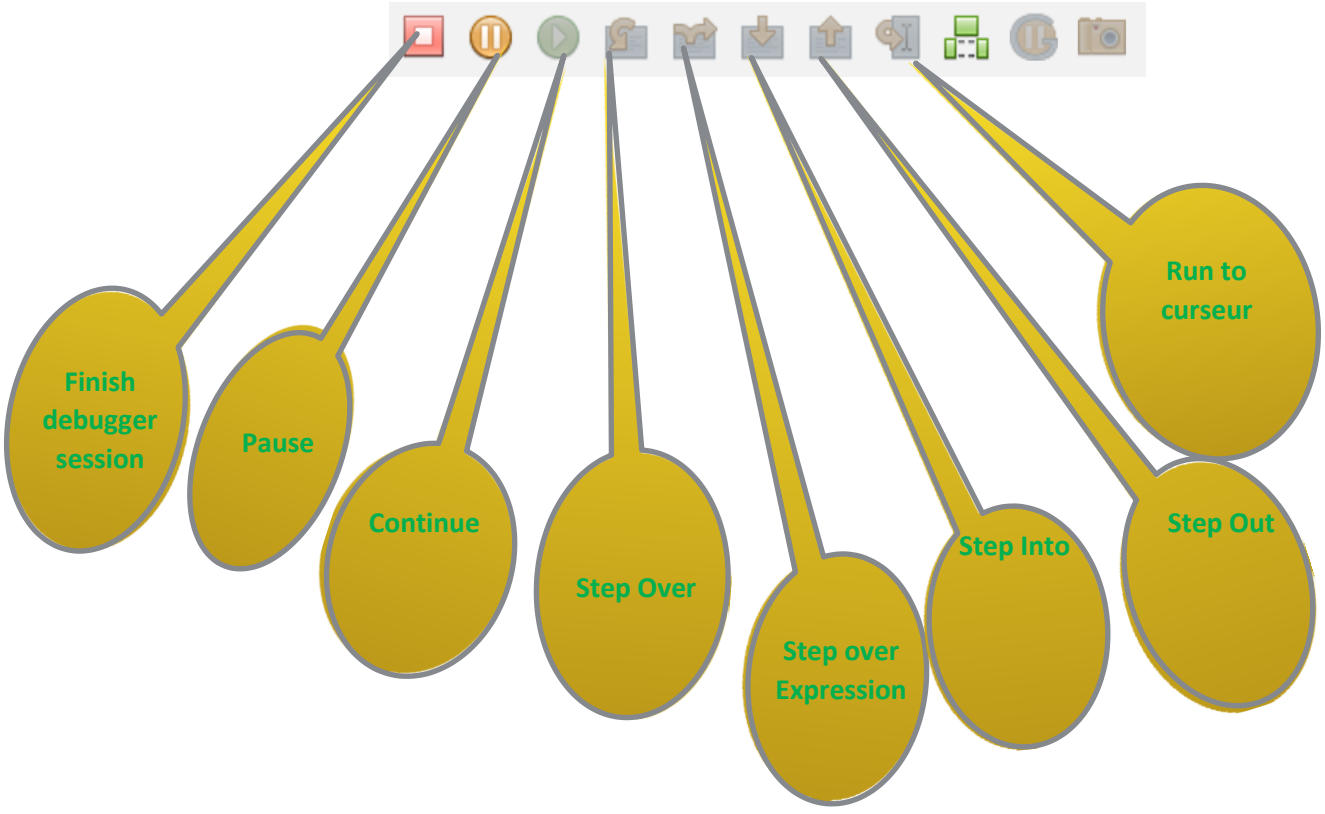
This will allow you to see how the values change and better understand the flow of your program.







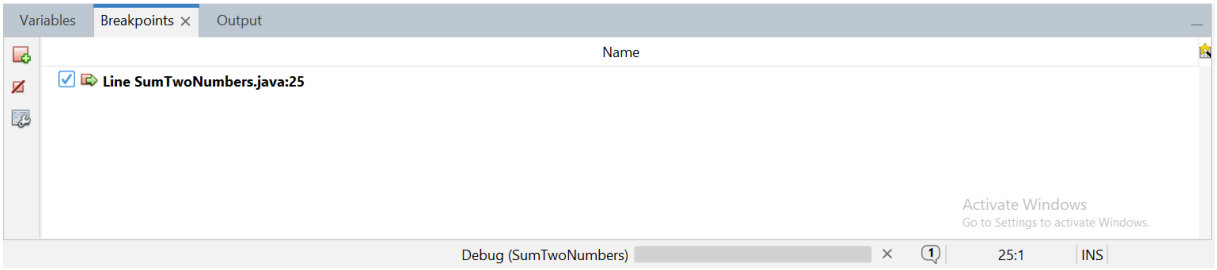
Débugger : barre de commande

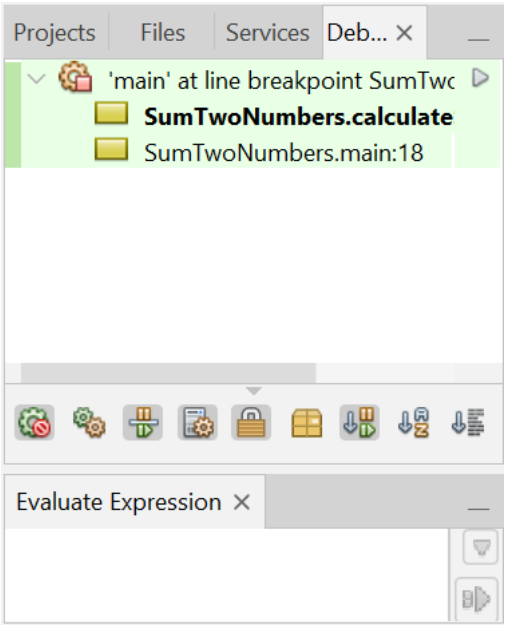


Debugger: breakpoint management

The NetBeans IDE offers advanced debugging functionality, including breakpoint management. Users can remove or disable these breakpoints as needed, allowing them to control the

execution flow of their code. Additionally, breakpoints can be added specifically for exceptions, a particularly useful option for quickly identifying and resolving errors in the program.



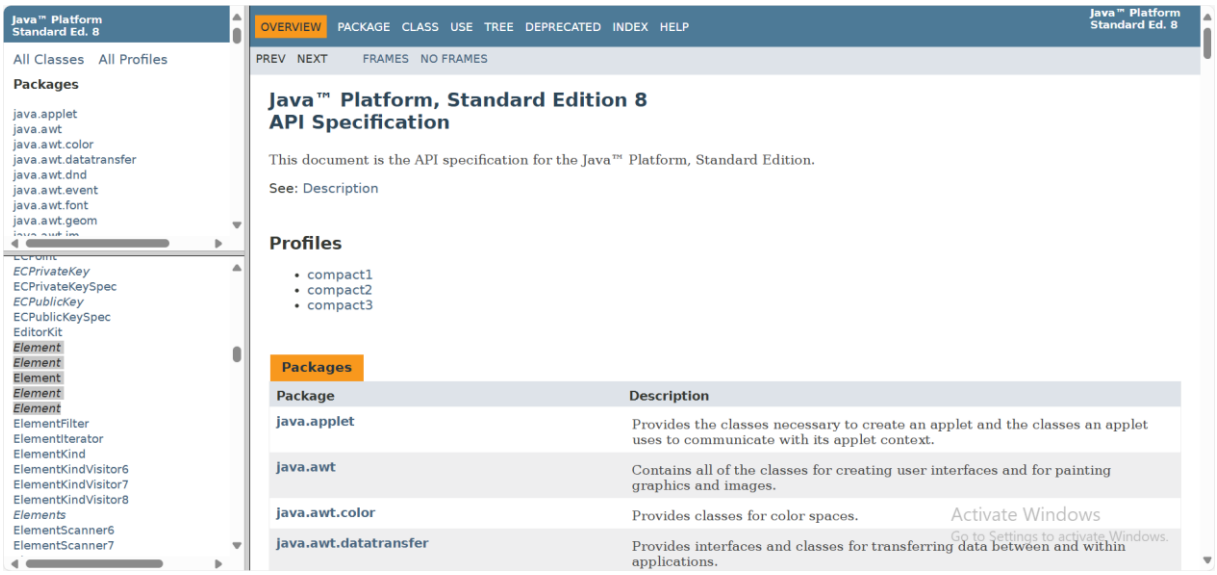


1.8 Packages

In the NetBeans IDE, package management is crucial, especially when developing complex programs. A real program typically uses a large number of classes, and it's common for it to integrate several libraries downloaded from the web, each of which also contains classes. This inevitably leads to the risk, if not the certainty, of class name collisions. It's not uncommon to end up with multiple classes with the same name, which can complicate code management and understanding.

Example

Let's take a concrete example: it is possible to have five different classes named ‘’Element’’ and two others named ‘’List’’. This multiplicity of identical names poses a significant challenge: how can these classes be distinguished from one another? This situation underlines the importance of good package organization to avoid ambiguity and facilitate code maintenance.



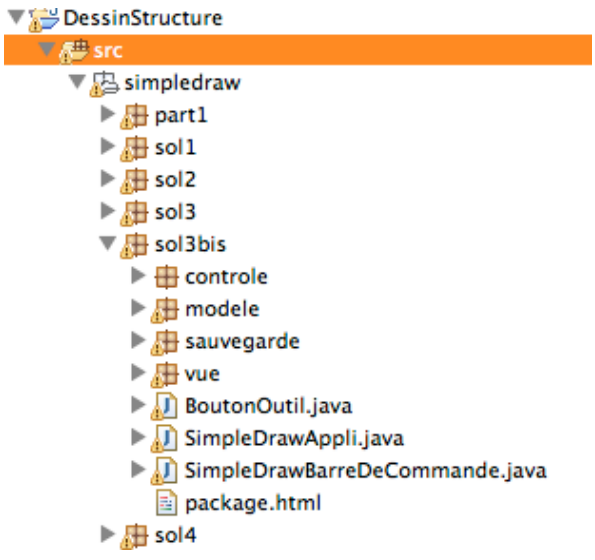
To view the API documentation online, go to:
https://docs.oracle.com/javase/8/docs/api/index.html

Solution: Packages

In the NetBeans IDE, class management is primarily done through the use of packages. These packages allow classes to be grouped thematically, facilitating code organization and structure. By convention, a package name begins with a lowercase letter, which helps distinguish it in the code. This approach is very similar to using folders on a file system, where files are grouped according to their theme. A package can contain not only classes, but also other packages, providing hierarchy and modularity in code organization.

Let's take a few examples to illustrate this structure: the ''java.text'' package includes classes dedicated to text manipulation, while ''javax.swing'' is the base package for user interface classes. Additionally, ''javax.swing.border'' groups together all the classes representing window ''edges'', allowing for advanced customization of graphical interfaces.

In the context of a drawing program, one could consider several distinct packages. For example, one package could be dedicated to the "model," representing the structure of the drawing in memory, another for the ''view'', which handles display, and a third for the save code, managing the saving of drawings to files.

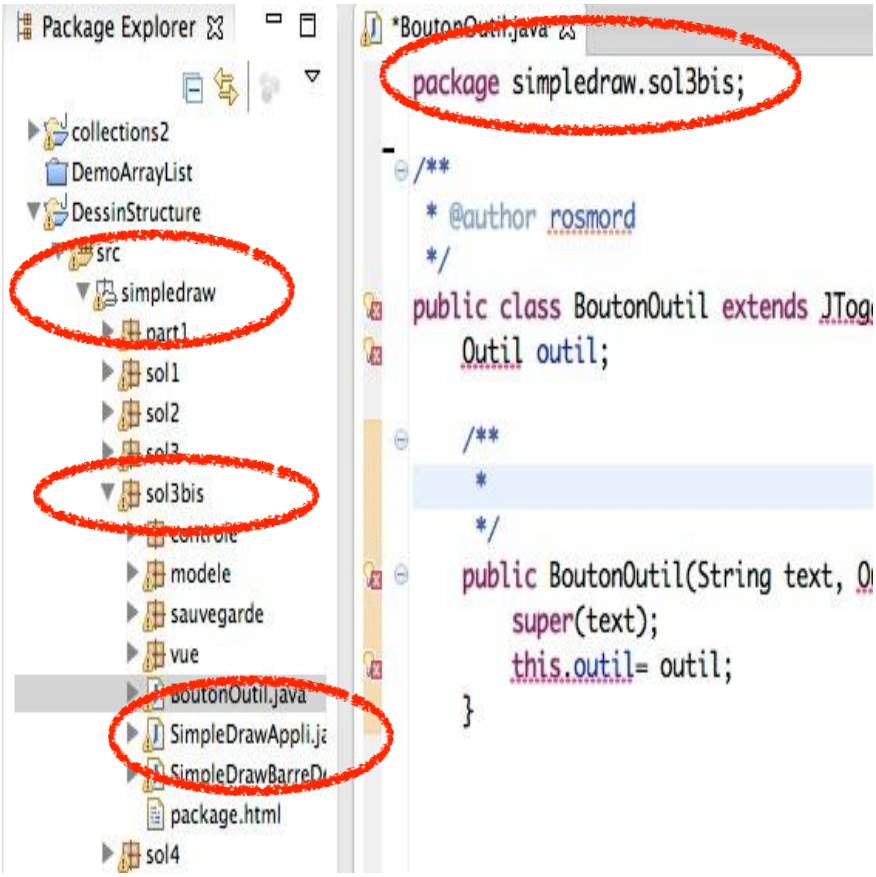


Organizational methods

Packages can be organized in two main ways: by theme or by layer. In a grade-tracking application, for example, it would be wise to create a package for student management and another for subjects. This thematic approach allows for grouping related features, making the code more readable and easier to maintain.

On the other hand, organizing by layer is also an effective method. In this case, you could have a separate package for the user interface, another for the program logic, and a final one for data access. This separation of responsibilities helps maintain a clear and modular architecture, thus facilitating the development and evolution of the application.

Creating packages



Creating packages

Creating packages in a Java project first requires establishing a folder for each package, while respecting the defined hierarchy. This folder structure allows for a clear organization of the project's various components. Within each class, it is essential to declare the package to which it belongs, making it easier to identify its location within the overall code architecture. Fortunately, integrated development environments like NetBeans automate this process, making package creation easier and faster for developers.

Using a class in a package other than its own

When using a class from another package, it is always possible to specify the full class name. For example, you can write ''public static int sum(java.util.List myList) {...}'' to refer to the ''List'' class in the ''java.util'' package. However, this method can quickly become cumbersome, especially if you need to reference multiple classes from different packages. A more convenient alternative is to use import statements.

Import

Import statements are placed between the line that declares the package and the beginning of the class. This allows classes to be imported and makes them easier to use without having to specify their full name each time. For example, in the class BoutonOutil, importing a class, such as ''Outil'', is done simply by its name, making the code more readable. In addition, it is possible to use the ''*'' character to import all the classes in a package, although this does not include classes in subpackages. For example, the statement ''import java.util.*;'' imports all the classes in the ''java.util'' package, such as ''List'' and ''Set'', thus simplifying the writing of the code.

```
package simpledraw.sol3bis;

import javax.swing.JToggleButton;

import simpledraw.sol3bis.controle.Outil;


/**
 * @author rosmord
 */
public class BoutonOutil extends JToggleButton {
    Outil outil;
```

The default package

A class that doesn't explicitly have a package belongs to the default package, which, unfortunately, doesn't have a name. This is the package you've been using so far, but it's important to note that you can't import a class from it. Therefore, it's advisable to avoid using it to facilitate code management and organization.

Note:

In Java, package names must be in lowercase and follow the reverse naming convention of the domain (e.g., com.example.mypackage). Avoid special characters, spaces, and uppercase letters; use periods to separate sublevels (e.g., fr.company.project.util).

''Real'' names of classes

The full name of a class consists of the name of the class, preceded by the name of the package that contains it. A subpackage is named by taking the name of its parent, followed by a period and the name of the package. For example, the class ''java.awt.List'' represents a list in an AWT GUI, where ''java.awt'' is a package located within the "java" package, which contains the standard libraries. Similarly, the class ''java.util.List'' denotes a list of items in memory, illustrating the importance of package structure for identifying classes in specific contexts.

public, private and nothing

When it comes to visibility, access modifiers play a crucial role. The "public" modifier indicates that the class and method are accessible to everyone. Conversely, an element declared "private" is only visible from the class where it is defined. There is also a "protected" modifier, an attribute or method declared protected in a class becomes accessible in the methods of its own class and in all subclasses. It also becomes accessible in all classes (and types) defined in the same package as the one that defines the protected member. Finally, if a method is neither public nor private, it is considered "public in its package, private elsewhere," which limits its accessibility within the same package while making it invisible outside.

1.9. Exercises

Exercise 1: Creating a Java Project with Debugging

Objective: Create a simple Java project in NetBeans, write a class with intentional errors, and then use the debugger to identify and fix these errors.

Steps to follow:

1. Create a new project:
   - Open NetBeans.
   - Go to ''File'' > ''New Project''.
   - Select Java under ''Categories'', then ''Java Application'' under ''Projects''. Click ''Next''.
   - Name your project ''MyFirstProject'' and click Finish.

2. Create a class:
   - In the ''Projects'' tab, right-click the ''Source Packages'' folder.
   - Select ''New'' > ''Java Class''. Name the class ''Calculator''.

3. Write the code:
   - In the Calculator class, write the following code:

```
public class Calculator {
public static void main(String[] args) {
int a = 10;
int b = 0;
int result = a / b; // Error: division by zero
System.out.println("The result is: " + result);
}
```

```
}
```

4. Compile and run the program:

- Attempt to run the program by clicking the ''Run'' button or by right-clicking the class and selecting ''Run File''.
- Note the error that appears in the console.

5. Use the debugger:

- Place a breakpoint on the line ''`int result = a/b`''; by clicking in the margin to the left of this line.
- Run the program in debug mode (right-click the class > ''Debug File'').
- When execution stops at the breakpoint, examine the values of "a" and "b" in the variables window.

6. Fix the error:

- Modify the code to avoid division by zero, for example:

```
if (b != 0) {
int result = a / b;
System.out.println("The result is: " + result);
} else {
System.out.println("Error: Division by zero!");
}
```

7. Recompile and run again:

- Compile and run the program again to verify that the error has been corrected.

Expected Outcome:

At the end of this exercise, you should be able to create a Java project in NetBeans, write code, identify errors using the debugger, and effectively fix them.

Exercise 2: Managing Packages in a Java Project

This exercise will help you understand how to organize your code using packages in NetBeans, as well as the importance of importing classes between different packages for better modularity and code organization.

Objective: Create a Java project in NetBeans, organize the code using packages, and demonstrate the import and use of classes between these packages.

Étapes à suivre :

1. Create a new project:

- Open NetBeans.
- Go to ''File''> ''New Project''.
- Select ''Java'' under ''Categories'', then ''Java Application'' under ''Projects''. Click ''Next''.
- Name your project ''PackageManagement'' and click ''Finish''.

2. Create the packages:
   - In the ''Projects'' tab, right-click the "Source Packages" folder.
   - Select ''New'' > ''Java Package''. Name the first package ''calculations''.
   - Repeat this process to create a second package named ''display''.

3. Create classes:
   - In the ''calculations'' package, create a new class named ''Calculator''. - In the ''Display'' package, create a new class named ''Display''.

4. Write the code in ''Calculator'':
   - Add the following code to the Calculator class:

```
package calculations;

public class Calculator {
public int addition(int a, int b) {
return a + b;
}
}
```

5. Write the code in the Display class:
   - Add the following code to the Display class:

```
package display;

import calculations.Calculator;

public class Displayer {
public void displayResult(int result) {
System.out.println("The result is: " + result);
}

public static void main(String[] args) {
Calculator calc = new Calculator();
int result = calc.addition(5, 7);
Display display = new Display();
display.displayResult(result);
}
}
```

6. Compile and run the program:
   - Run the Display class by right-clicking on the file and selecting ''Run File''.

Expected result:

At the end of this exercise, you should see the following output in the console:
The result is: 12

2.10. Exercise corrections

Answers to exercise 1: Création d'un Projet Java avec Débogage

Step 1: Creating a New Project

- Action Completed: A new project named ''MyFirstProject'' has been successfully created.

Step 2: Creating a Class

- Action Completed: The ''Calculator'' class has been created in the ''Source Packages'' folder.

Step 3: Writing the Code

- Initial Code:

```
public class Calculator {
public static void main(String[] args) {
int a = 10;
int b = 0;
int result = a / b; // Error: division by zero
System.out.println("The result is: " + result);
}
}
```

- Problem: Running the program results in an ArithmeticException due to division by zero.

Step 4: Compile and run the program

- Result: An error appears in the console stating ''Exception in thread'' main ''java.lang.ArithmeticException: / by zero''.

Step 5: Use the debugger

- Action taken: A breakpoint was placed on the division line.
- Observation: In debug mode, the variable values were:
- a = 10
- b = 0

Step 6: Fix the error

- Code corrected:

```
public class Calculator {
public static void main(String[] args) {
int a = 10;
int b = 0;
if (b != 0) {
int result = a / b;
System.out.println("The result is: " + result);
} else {
System.out.println("Error: Division by zero!");
}
}
}
```

Step 7: Recompile and Run Again

- **Expected Result:** When running the program after the fix, the output should be:

Error: Division by zero!

The exercise covered how to create a Java project, identify a common error (division by zero), and use the debugger to analyze and correct the code. You also learned how to handle runtime errors with appropriate conditions.

Answers to exercise 2: Gestion des Packages dans un Projet Java

Step 1: Creating a New Project

- Action Completed: A new project named ''PackageManagement'' has been successfully created.

Step 2: Creating Packages

- Packages created:
  a. ''calculations''
  b. ''display''

Step 3: Creating Classes

- Classes created:
  a. ''Calculator'' in the ''calculations'' package.
  b. ''Display'' in the ''display'' package.

Step 4: Writing Code in Calculator

- Code for the Calculator class:

```
package calculs;
  public class Calculatrice {
      public int addition(int a, int b) {
          return a + b;
      }
  }
```

- Functionality: The Calculator class contains an add() method that takes two integers as parameters and returns their sum.

Step 5: Write the code in the ''Display''

- Code for the Display class:

```
package display;
import calculations.Calculator;
public class Displayer {
public void displayResult(int result) {
System.out.println("The result is: " + result);
}

public static void main(String[] args) {
Calculator calc = new Calculator();
int result = calc.addition(5, 7);
Display display = new Display();
display.displayResult(result);
}
}
```

- Functionality:
  - The Display class imports the Calculator class.
  - It contains a displayResult method that displays the result of an addition.
  - In the main method, a Calculator instance is created, the addition of 5 and 7 is performed, and the result is displayed.

Step 6: Compile and Run the Program

- Action Performed: Execute the Display class.
- Expected Result: The console output is:
  - The result is: 12

Lab 2 : Refresher exercises on the Java language

Outlin

2.1. Introduction
2.2. Exercises
2.3. Exercise Corrections

Goals

- Strengthen understanding of fundamental Java language concepts;
- Practice basic syntax and structures through applied exercises;
- Review key concepts of object-oriented programming (OOP) already covered in the previous year.

2.1. Introduction

The following exercises will cover various aspects of the Java language and help you strengthen your skills. They focus particularly on object-oriented programming (OOP) concepts you have already studied, such as classes, objects, inheritance, and interfaces. Through these exercises, you will have the opportunity to practice your knowledge and consolidate your understanding. Each exercise is designed to be accessible while providing challenges appropriate to your learning level, facilitating effective recall of important concepts.

2.2. Exercises

Exercise 1: Variables and Data Types

- Write a Java program that declares variables of different types (int, double, char, String) and displays their values.

Exercise 2: Control structures

- Create a program that asks the user to enter an integer and displays whether that number is even or odd.

Exercise 3: Loops

- Write a program that uses a for loop to display the numbers from 1 to 100, indicating whether each number is a multiple of 3, 5, or both.

Exercise 4: Arrays

- Write a program that declares an array of integers, fills it with values of your choice, and then calculates and displays the sum and average of the array elements.

Exercise 5: Methods

- Create a method that takes two integers as parameters and returns the larger of the two. Test this method in the main.

Exercise 6: Classes and Objects

- Define a class ''Car'' with attributes like ''make'', ''model'', and ''year''. Create an object of this class in the ''main'' and display its attributes.

Exercise 7: Inheritance

- Create an Animal class with a makeNoise() method. Create a Dog subclass that implements this method. Test for inheritance by creating an object of the Dog class.

Exercise 8: Interfaces

- Define a ''Playable'' interface with a ''play()'' method. Create a ''Game'' class that implements this interface and displays a message when ''play()'' is called.

Exercise 9: Exception Management

- Write a program that prompts the user for a number and handles exceptions if the user enters a non-numeric value.

Exercise 10: Collections

- Create a list of integers, add some values, and use a loop to display each value in the list. Then, sort the list and display it again.

2.3. Exercise Corrections

Answers to exercise 1: Variables and data types

```
package lab2;
public class Variables {
    public static void main(String[] args) {
        int number = 10;
        double decimal = 5.5;
        char letter = 'A';
        String sentence = "Hello, Java!";

      System.out.println("Number: " + number);
      System.out.println("Decimal: " + decimal);
      System.out.println("Letter: " + letter);
      System.out.println("Sentence: " + sentence);
    }
}
```

The execution result is:

```
Number: 10
Decimal: 5.5
Letter: A
Sentence: Hello, Java!
```

Answers to exercise 2: Control structures

```
package lab2;
import java.util.Scanner;

public class EvenOdd {
public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);
System.out.print("Enter an integer: ");
int number = scanner.nextInt();

if (number % 2 == 0) {
System.out.println(number + " is even.");
} else {
System.out.println(number + " is odd.");
}
scanner.close();
}
}
```

The execution result is:

```
Enter an integer: 179
179 is odd.
```

Answers to exercise 3: *Loops*

```
package lab2;
public class Multiples {
       public static void main(String[] args) {
       for (int i = 1; i <= 100; i++) {
       String message = i + ": ";
       if (i % 3 == 0) message += "multiple of 3 ";
       if (i % 5 == 0) message += "multiple of 5 ";
       System.out.println(message);
       }
       }
      }
```

The execution result is:

```
1:
2:
3: multiple of 3
4:
5: multiple of 5
6: multiple of 3
7:
8:
9: multiple of 3
10: multiple of 5
11:
12: multiple of 3
13:
14:
15: multiple of 3 multiple of 5
16:
17:
18: multiple of 3
19:
20: multiple of 5
21: multiple of 3
22:
23:
24: multiple of 3
25: multiple of 5
26:
27: multiple of 3
28:
29:
30: multiple of 3 multiple of 5
```

```
31:
32:
33: multiple of 3
34:
35: multiple of 5
36: multiple of 3
37:
38:
39: multiple of 3
40: multiple of 5
41:
42: multiple of 3
43:
44:
45: multiple of 3 multiple of 5
46:
47:
48: multiple of 3
49:
50: multiple of 5
51: multiple of 3
52:
53:
54: multiple of 3
55: multiple of 5
56:
57: multiple of 3
58:
59:
60: multiple of 3 multiple of 5
61:
62:
63: multiple of 3
64:
65: multiple of 5
66: multiple of 3
67:
68:
69: multiple of 3
70: multiple of 5
71:
72: multiple of 3
73:
74:
75: multiple of 3 multiple of 5
76:
77:
78: multiple of 3
79:
80: multiple of 5
81: multiple of 3
82:
83:
84: multiple of 3
85: multiple of 5
86:
87: multiple of 3
88:
89:
90: multiple of 3 multiple of 5
91:
```

```
92:
93: multiple of 3
94:
95: multiple of 5
96: multiple of 3
97:
98:
99: multiple of 3
100: multiple of 5
```

Answers to exercise 4: Arrays

```
package lab2;
public class SumAverage {
public static void main(String[] args) {
int[] array = {10, 20, 30, 40, 50};
int sum = 0;

for (int number : array) {
sum += number;
}
double average = (double) sum / array.length;

System.out.println("Sum: " + sum);
System.out.println("Average: " + average);
}
}
```

The execution result is:

```
Sum: 150
Average: 30.0
```

Answers to exercise 5: Methods

```
package lab2;
public class Larger {
       public static void main(String[] args) {
       int a = 5;
       int b = 10;
       System.out.println("The largest is: " + largest(a, b));
       }

       public static int largest(int x, int y) {
       return (x > y) ? x:y;
       }
      }
```

The execution result is:

```
The largest is: 10
```

Answers to exercise 6: Classes and objects

```
package lab2;
class Car {
String brand;
String model;
int year;

Car(String brand, String model, int year) {
this.brand = brand;
this.model = model;
this.year = year;
}
```

```
}

public class TestCar {
public static void main(String[] args) {
Car myCar = new Car("Toyota", "Corolla", 2020);
System.out.println("Make: " + myCar.brand);
System.out.println("Model: " + myCar.model);
System.out.println("Year: " + myCar.year);
}
}
```

The execution result is:

```
Make: Toyota
Model: Corolla
Year: 2020
```

Answers to exercise 7: Inheritance

```
package lab2;
class Animal {
void makeNoise() {
System.out.println("The animal is making noise");
}
}

class Dog extends Animal {
@Override
void makeNoise() {
System.out.println("The dog is barking");
}
}

public class TestAnimal {
public static void main(String[] args) {
Dog myDog = new Dog();
myDog.makeNoise();
}
}
```

The execution result is:

```
The dog is barking
```

Answers to exercise 8: Interfaces

```
 package lab2;
interface Playable {
    void play();
}

class Game implements Playable {
    @Override
    public void play() {
        System.out.println("The game begins !");
    }
}

public class GameTest {
    public static void main(String[] args) {
      Game myGame = new Game();
      myGame.play();
    }
}
```

The execution result is:

```
The game begins !
```

Answers to exercise 9: Exception management

```
package lab2;
import java.util.Scanner;

public class ExceptionManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");

        try {
            int nombre = scanner.nextInt();
            System.out.println("You entered: " + nombre);
        } catch (Exception e) {
            System.err.println("Error: Please enter a valid number.");
        } finally {
            scanner.close();
        }
    }
}
```

The execution result is:

Example 1:

```
Enter a number: ggg
Error: Please enter a valid number.
```

Example 2:

```
Enter a number: 150
You entered: 150
```

Answers to exercise 10: Collections

```
package lab2;
import java.util.ArrayList;
import java.util.Collections;

public class IntegersList {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(5);
        list.add(3);
        list.add(8);
        list.add(1);

        System.out.println("Unsorted list: " + list);
        Collections.sort(liste);
        System.out.println("Sorted list: " + list);
    }
}
```

The execution result is:

```
Unsorted list: [5, 3, 8, 1]
Sorted list: [1, 3, 5, 8]
```

Lab 3: Reminder on Object-Oriented Programming

Outline	3.1. Introduction 3.2. Basic OOP Concepts 3.3. Class Diagram 3.4. Implementation with Java 3.5. Exercise 3.6. Exercise correction
Goals	➢ Understand the principles of object-oriented programming (OOP); ➢ Learn to model objects using Java classes; ➢ Implement and test classes and objects in a development environment.

3.1. Introduction

Object-oriented programming (OOP) is a programming paradigm that uses objects to represent data and actions. It differs from traditional programming methods, notably through the use of classes and objects, which facilitate modularity and code reuse. In this lab, we will explore the fundamental concepts of OOP and put them into practice using a real-world example related to employee management in a company.

3.2. Basic Concepts of OOP

OOP is based on four main concepts:

1. Encapsulation: Protecting data by grouping it into classes and controlling access to attributes via methods.
2. Inheritance: A mechanism for creating new classes from existing ones, promoting code reuse.
3. Polymorphism: The ability to treat objects of different classes in a uniform manner, often achieved through overriding or refining methods.
4. Abstraction: Simplifying complex systems by modeling only the essential elements.

3.3. Class diagram

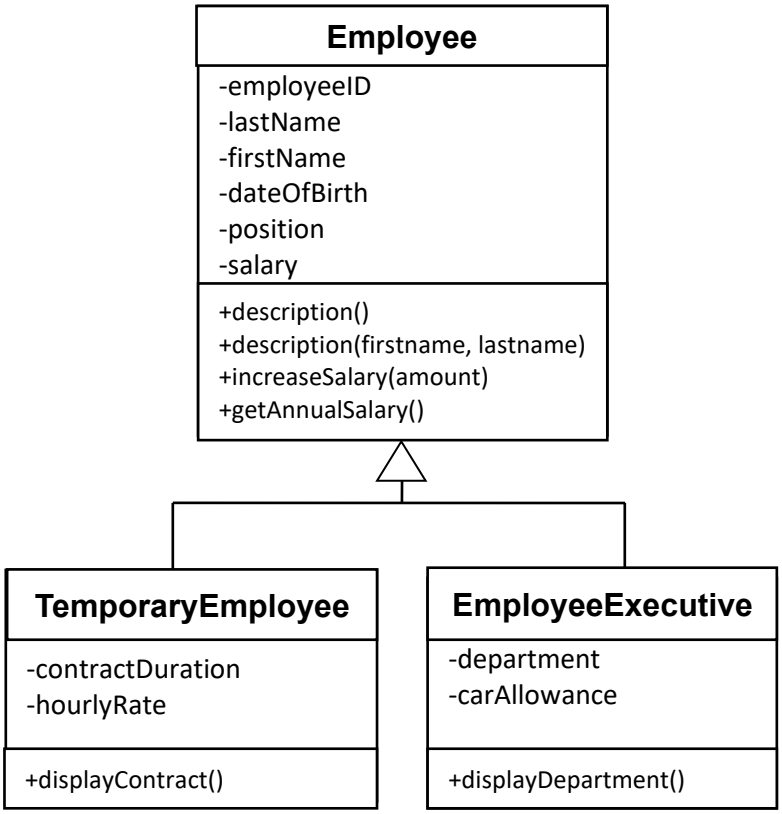
The class diagram is a visual tool for representing a system's classes and their relationships. In our case, we have the following classes:

- Employee
  - Attributes: employeeID, lastName, firstName, dateOfBirth, position, salary
  - Methods: description(), increaseSalary(), getAnnualSalary()
- TemporaryEmployee (inherits from Employee)
  - Attributes: contractDuration, hourlyRate

  - Methods: displayContract()
- ExecutiveEmployee (inherits from Employee)
  - Attributes: department, carAllowance
  - Methods: displayDepartment()

3.4. Implementation with Java

Firstpack, a company specializing in the distribution and manufacturing of disposable tableware and food packaging, wants to implement an employee management system. This system must support regular employees, temporary employees, and managers. An employee can be either a temporary employee or a manager. To develop this system, Firstpack hired a private development company. After a thorough study and agreement on the specifications, the company hired developers who proposed a design. An excerpt from the proposed class diagram for the system is shown below:



Required tasks:

1. Create a new Java project using the NetBeans IDE.
2. Define the ''Employee'' class with a default constructor and a constructor that takes parameters. The default attributes are: ''0'' for the employee ID, ''null'' for the employee's first and last name, date of birth, and position, and ''0.0'' for the salary. It's worth noting that the idEmployee attribute is initialized to 0 and automatically increments by 1 each time a new instance of the "Employee" class is created.

3. Implement the necessary methods in the ''Employee'' class according to the following specifications:

a) The description() method should display the employee's details, such as their first and last name, date of birth, position, and salary, as follows:
b) LANGLOIS Alain is an employee, Developer, born on 02/18/1978, and has a monthly salary of 6,000 $.
c) The second method, description(firstname, lastname), assigns an employee number as follows: LANGLOIS Alain's employee ID is: idEmploye.
d) The increaseSalary(amount) method should increase the employee's salary by the value specified in the ''amount'' parameter.
e) The getAnnualSalary() method should return the employee's annual salary, calculated by multiplying the monthly salary by 12.

4. Write the main class, ManageEmployees, to test the methods of the ''Employee'' class. Instantiate two objects of the ''Employee'' class: ''employee1'' with the attributes (''HUBERT'', ''Christophe'', ''12/03/1988'', ''Developer'', 4800.0); and ''employee2'' with the attributes (''RENARD'', ''Nathalie'', ''02/01/1975'', ''Manager'', 7000.0);

5. Implement the derived classes TemporaryEmployee and Executive by adding the attributes and methods specific to each class:

a) The TemporaryEmployee class must have the following additional attributes: ContractDuration (int) and HourlyRate (double). Implement the necessary constructors and methods specific to this class.
b) The ExecutiveEmployee class must have the following additional attributes: Department (String) and carAllowance (double). Implement the necessary constructors and methods specific to this class.

6. Instantiate an object of the ''TemporaryEmployee'' class and another of the ''ExecutiveEmployee'' class and test their methods.

3.6. Exercise correction:

1. Create a new Java project using the NetBeans IDE

First, you need to launch NetBeans and then create a new Java project to house your classes. To do this, go to the File, New Project menu. A window will appear. In this window, specify the type of project to create. Choose the standard type (Java) from Categories (on the left), then choose Java Application from Projects (on the right). Then, click the Next button. Another window will appear. Enter the project name in the Project Name section. Enter the name ''Firstpack''. Unlike Eclipse, in NetBeans, the main class can be created at the same time as the project if the Create Main Class box is checked. Check this box. Generally, in NetBeans, the "Create Main Class" box is automatically checked, and the main class can also be automatically named firstpack. Firstpack. Then, click the Finish button. The project is created along with the Firstpack main class. If you don't see the projects, click on Window in the menu and then on Projects.

The answers to the remaining questions in this exercise are embedded in the following complete code:

```
package gestionemployes;

/**
 *
 * @author Dr Mohamed Mohammedi
 */

class Employee {
private static long EmployeeId;
private String lastName;
private String firstName;
private String dateOfBirth;
private String position;
private double salary;
private static long num_Employee=0;

// Default constructor
public Employee() {
this.EmployeeId = 0;
this.lastName = null;
this.firstName = null;
this.dateOfBirth=null;
this.position = null;
this.salary = 0.0;
}

// Constructor with parameters
public Employee(String lastName, String firstName, String dateOfBirth,
String position, double salary) {
this.num_Employee++;
EmployeeId = num_Employee;
this.lastName = lastName;
this.firstName = firstName;
this.dateOfBirth=dateOfBirth;
this.position = position;
this.salary = salary;
}

public String getLastName() {
return lastName;
}

public String getFirstName() {
return firstName;
}

public String getDateOfBirth() {
return dateOfBirth;
}

public void description() {
System.out.println(this.lastName + " " + this.firstName + " is an employee,
" + this.position +
", and their monthly salary is: " + this.salary + " $.");
}

public void description(String firstName, String lastName) {
```

```
System.out.println("The employee ID " + this.lastName + " " +
this.firstName + " is: " + this.idEmployee);
}

public void increaseSalary(double amount) {
this.salary += amount;
}

public double getAnnualSalary() {
return this.salary * 12;
}
}

class TemporaryEmployee extends Employee {
private int ContractDuration;
private double HourlyRate;

public TemporaryEmployee(String lastName, String firstName, String
dateOfBirth, String position, double salary,
int ContractDuration, double hourlyRate) {
super(lastName, firstName, dateOfBirth, position, salary);
this. ContractDuration = ContractDuration;
this. HourlyRate = HourlyRate;
}

// Method specific to the TemporaryEmployee class
public void displayContract() {
System.out.println(this.getLastName() + " " + this.getFirstName() + " born
on " + this.getDateOfBirth() + " has a " + " temporary contract lasting " +
this.ContractDuration + " months");
}
}

class ExecutiveEmployee extends Employee {
private String department;
private double carAllowance;

public ExecutiveEmployee(String lastName, String firstName, String
dateOfBirth, String position, double salary,
String department, double carAllowance) {
super(lastName, firstName, dateOfBirth, position, salary);
this.department = department;
this.carAllowance = carAllowance;
}

// Method specific to the Cadre class
public void displayDepartment() {
System.out.println("The department: " + this.getLastName() + " " +
this.getFirstName() + " born on " + this.getDateOfBirth() + " is " +
this.department);
}
}

public class GestionEmployes {
public static void main(String[] args) {
Employee employee1 = new Employee("HUBERT", "Christophe", "12/03/1988",
"Developer", 4800.0);
employee1.description("HUBERT", "Christophe");
employee1.description();
employee1.increaseSalary(500.0);
```

```
Employee employee2 = new Employee("RENARD", "Nathalie", "02/01/1975",
"Manager", 7000.0);
employee2.description("RENARD", "Nathalie");
employee2.description();

double AnnualSalary = employee2.getAnnualSalary();
System.out.println("The annual salary of "+employee2.getLastName()+"
"+employee2.getFirstName()+ " is: " + AnnualSalary + " $.");

TemporaryEmployee TemporaryEmployee = new TemporaryEmployee("DUPUIS",
"Pierre", "22/09/1980", "Assistant", 2500.0,
6, 15.0);
TemporaryEmployee.description("DUPUIS", "Pierre");
TemporaryEmployee.description();
TemporaryEmployee.displayContract();

ExecutiveEmployee executiv = new ExecutiveEmployee("LEBLANC", "Sophie",
"07/04/1968", "Director", 10000.0,
"Human Resources", 1500.0);
executiv.description("LEBLANC", "Sophie");
executiv.description();
executiv.displayDepartment();
}
}
```

The execution result is:

```
The employee ID HUBERT Christophe is: 1
HUBERT Christophe is an employee, Developer, and their monthly salary is:
4800.0 $.
The employee ID RENARD Nathalie is: 2
RENARD Nathalie is an employee, Manager, and their monthly salary is: 7000.0
$.
The annual salary of RENARD Nathalie is: 84000.0 $.
The employee ID DUPUIS Pierre is: 3
DUPUIS Pierre is an employee, Assistant, and their monthly salary is: 2500.0
$.
DUPUIS Pierre born on 22/09/1980 has a temporary contract lasting 6 months
The employee ID LEBLANC Sophie is: 4
LEBLANC Sophie is an employee, Director, and their monthly salary is: 10000.0
$.
The department: LEBLANC Sophie born on 07/04/1968 is Human Resources
```

Lab 4: Programming graphical interfaces

Outline

Goals

- Understand the fundamentals of graphical interfaces in Java,
- Learn how to create a graphical interface,
- Master common controls,
- Explore event-driven programming,
- Practice event handling,
- Apply knowledge through practical exercises,
- Develop interface design skills.

4.1. Introduction

In computing, a GUI component is a basic element with which a user can interact with our application. These components are generally grouped into graphical toolkits. Once assembled by a programmer, these components form a complete graphical interface. In this chapter, we will see how to create these components and their uses.

As you may have noticed in previous chapters, running a Java program automatically creates a console window. However, nothing comparable is provided for a graphical window intended to support event-driven programming. The program must therefore explicitly create it. Here, we will see how to achieve this. For the sake of simplicity, we will limit ourselves to creating this window. Later, we will see how to make it support the user interface by introducing the desired components (menus, buttons, dialog boxes, etc.) and properly handling the corresponding events.

4.2. Creating a Graphical Interface in Java

The Human-Computer Interface (HCI) refers to the set of elements that allow users to interact with a computer system. In the context of Java, several libraries and frameworks facilitate the creation of user-friendly and dynamic graphical interfaces. Here is an overview of the key concepts related to the HCI in Java.

Main Libraries

- **Swing**: One of the most widely used libraries for creating graphical interfaces in Java. Swing offers a wide range of components such as windows, buttons, lists, etc. The components are lightweight and can be customized.

- **JavaFX**: A newer technology than Swing, designed to replace it. JavaFX allows you to create modern interfaces with advanced visual effects, animations, and better support for web and mobile applications.

To create a graphical window, the package named **javax.swing** includes a standard class named JFrame, which has a constructor without arguments.

Example:

```
JFrame fen = new JFrame();
```

Several methods can be used to create windows:

setTitle: Allows you to assign a title to your window.

setBounds: Allows you to specify the window's size and position on the screen.

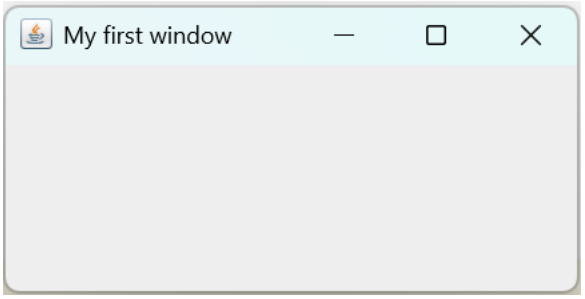
setVisible: Allows you to make the window visible.

Example:

Here's a simple example that allows you to create a graphical window:

```
package lab4;
import javax.swing.JFrame;
public class MyFirstWindow {
public MyFirstWindow () {
// TODO Auto-generated constructor stub
}
public static void main(String[] args) {
// TODO Auto-generated method stub
JFrame fen = new JFrame() ;
fen.setSize (300, 150) ;
fen.setTitle ("My first window") ;
fen.setVisible (true) ;
}

}
```

The result of executing the previous program gives the following window:



We've seen that closing the graphical window makes it disappear. To be precise, we should say that it simply makes it invisible (as if we called setVisible(false)). In a pinch, it would be possible to make it reappear. We can impose a different behavior when closing the window by calling the JFrame's setDefaultCloseOperation method with one of the following arguments:

- DO_NOTHING_ON_CLOSE: Do nothing.
- HIDE_ON_CLOSE: Hide the window (default behavior).
- DISPOSE_ON_CLOSE: Destroy the window object.

However, closing the window would never terminate the application.

4.3. Common Controls of Graphical Interface Components

The graphical interface consists of various components, each with specific functions. Some of these components are containers, designed to house other elements. For example, windows (JFrames) and panels (JPanels) serve as supports for other components. Others, however, cannot contain other elements, such as buttons, often called controls or atomic components.

This section examines the main controls available in the Swing package, including:

- Check boxes;
- Radio buttons and the associated notion of grouping;
- Labels;
- Text fields;
- List boxes;
- Combo boxes.

4.3.1. Creating a button

To create a button, we have, in the package named **javax.swing**, a standard class named **JButton**, which has a constructor without arguments:

```
JButton b1=new JButton() ;
```

Several methods can be used to manipulate buttons:

- **setText**: Allows you to assign text to your button.
- **setBounds**: Allows you to specify the button's size and position in the window.

- **setVisible**: Allows you to make the button visible.
- **setBackground**: Allows you to assign a specific color to your button.

As we've already mentioned, windows are containers, while buttons are components that we need to position within the window. To do this, we use a container, which is found in the **java.awt** package.
Containers allow us to integrate objects into windows and organize them. There are several types of containers, each with a different way of managing components within the window. Containers include:

- FlowLayout: Allows you to position components one after the other relative to the first component.
- Null: Allows you to position components at the locations desired by the programmer (defined in the **setBounds** method).

4.3.2. Creating components and placing them on the window

First, we need to create a container using the Container class, which is located in the **java.awt** package. It allows us to integrate other visual objects and organize them on the screen.

All components will be created like the window, and then we'll add them to the window using the container.

Example: Creating a Button

```
package lab4;
import javax.swing.*;
import java.awt.Container;

public class Window extends JFrame {
public Window() {
// Create the container
Container c = getContentPane();

// Choose the layout to use
c.setLayout(null);

// Create the window
this.setTitle("Window 2");
this.setVisible(true);
this.setBounds(100, 100, 200, 300);

// Create a button
JButton b1 = new JButton();
b1.setText("Button");
b1.setVisible(true);
b1.setSize(150, 30);
b1.setLocation(10, 10);

// Place the button in the window using the container using the add
method
c.add(b1);
}
}
```

Here is a table that shows some classes for creating other components:

Graphic component	Class
Checkbox allows the user to make multiple choices in a graphical interface. It displays a checkbox that can be enabled or disabled.	JCheckBox
Radio Button allows the user to select one option from a group of exclusive options. Unlike **JCheckBoxes**, where multiple options can be selected, a **JRadioButton** only allows one selection at a time from a given group.	JRadioButton
Label allows you to display text or an image	JLabel
Text field for user input	JTextField
Text space allows you to display or edit multi-line text.	JTextArea
Container that can hold other components	Jpanel
A **separator** is used to create a dividing line in a graphical interface.	JSeparator

4.3.3. Creating menus

Let's now move on to creating menus and positioning them on windows, which are important elements in our graphical interface.

These common drop-down menus involve three types of objects:

- A menu bar object (JMenuBar);
- Various menu objects (JMenu), which will be visible in the menu bar;
- For each menu, the various options, of type JMenuItem, that constitute it.

a) Creating a menu bar object:
Creating a menu bar object is done as follows:

```
JMenuBar menubar = new JMenuBar(); // This bar will be attached to a window
wind by:
wind.setJMenuBar(menubar); // Attaches the barreMenus object to the window wind.
```

The various menu objects are created by calling a JMenu constructor, which is given the name of the menu as it will appear in the bar. Each menu object is added to the bar by add (it appears in the order in which it was added):

Example :

```
JMenu File = new JMenu("File"); // Creates a menu named File
barMenus.add(File); // Adds it to the File menu's barMenus.
```

b) Finally, the different options of a menu are created by calling a JMenuItem constructor, to which we provide, again, the name of the option as it will appear when the user displays the contents of the menu. Each option is added to a menu by add.

Example :

```
J JMenuItem New = new JMenuItem("New"); // Creates an option named New
File.add(New); // Adds it to the File menu
```

4.4. Event-Driven Programming in Java

For this type of programming, we must first define what an interface is:

4.4.1. Interface

An interface contains a set of methods without content. To create an interface, simply use the interface keyword and list the methods it must contain.

Creation and Implementation:

```
//Create an interface
interface InterfaceName {
//Methods of the interface;
}

//Implementation in a class
class ClassName implements InterfaceName {
//Define the instructions for the different methods;
```

```
}
```

4.4.2 Managing a click in a window

Implementing the MouseListener interface

The MouseListener interface has five (05) methods, each corresponding to a specific event:

- mousePressed: each time the mouse is pressed.
- mouseReleased: each time the mouse is released.
- mouseEntered: each time the mouse moves from outside to inside a component.
- mouseExited: each time the mouse moves from inside to outside a component.
- mouseClicked: each time the mouse is clicked.

Note:

addMouseListener(object_name): This method allows you to add a listener to process the interface methods.

The following example displays a message in the console for each click in the graphics window.

Example :

```
package lab4;
import javax.swing.*; // For the JFrame
import java.awt.event.*; // For MouseListener and MouseEvent

class Window extends JFrame implements MouseListener {
public Window() { // Create a constructor
setTitle("Window name");
addMouseListener(this); // The window will be its own event listener
}

// Declare the interface methods
public void mouseClicked(MouseEvent ev) {
System.out.println("Click in window");
}

public void mousePressed(MouseEvent ev) { }

public void mouseReleased(MouseEvent ev) { }

public void mouseEntered(MouseEvent ev) { }

public void mouseExited(MouseEvent ev) { }
}

public class Clic1 {
public static void main(String[] args) {
Window fen = new Window();
fen.setSize(400, 300); // Set the window size
fen.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Close the application
when the window closes
fen.setVisible(true); // Make the window visible
```

```
}
}
```

Consequently, depending on the needs, we will implement the appropriate methods.

4.4.3. Managing Events in general

We've just shown you how an event triggered by an object named source can be processed by another object named listener, previously associated with the source. Everything explained here in a simple example will generalize to other events, whatever they may be and whatever their source.

In particular, a given category Xxx will always be associated with an event listener object (of type XxxEvent), using a method named addXxxListener. Whenever a given category has multiple methods, we can:

- Either redefine all methods of the corresponding XxxListener interface (the implements XxxListener clause must appear in the listener's class header), as some methods may have an empty body.
- Or call a class derived from an XxxAdapter adapter class and provide only the methods we're interested in (when the category only has a single method, Java hasn't provided an adapter class because it would be useless).

4.4.4. Component Layout

Component layout is managed by layout managers such as:

- FlowLayout: Arranges components in a row, wrapping them to the next row if necessary.
- BorderLayout: Divides the window into five zones (north, south, east, west, center).
- GridLayout: Organizes components in a grid.

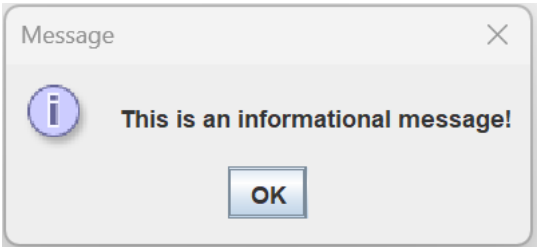
4.4.5. Creating Dialog Boxes with JOptionPane

To create a JOptionPane in Java, you can use this class to display simple dialog boxes, such as informational messages, confirmations, or user input. Here are some basic examples:

- Display an information message

```
package lab4;
import javax.swing.JOptionPane;
public class MessageDialogExample {
 public static void main(String[] args) {
 JOptionPane.showMessageDialog(null, "This is an informational
message!");
 }
}
```

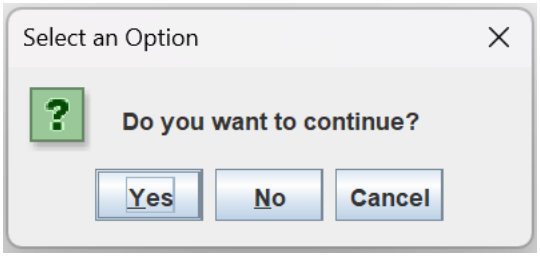
The execution result is:



- Request confirmation

```
package lab4;
import javax.swing.JOptionPane;
public class ConfirmationDialogExample {
public static void main(String[] args) {
int response = JOptionPane.showConfirmDialog(null, "Do you want to
continue?");
if (response == JOptionPane.YES_OPTION) {
System.out.println("You have chosen to continue.");
} else {
System.out.println("You have chosen not to continue.");
}
}
}
```

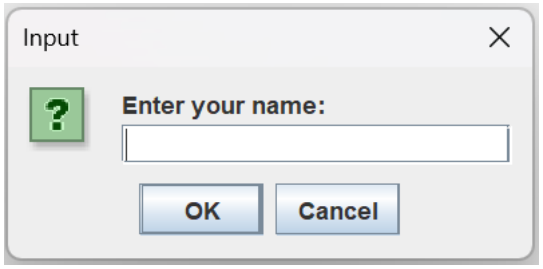
The execution result is:



- Request user input

```
package lab4;
import javax.swing.JOptionPane;
public class InputDialogExample {
 public static void main(String[] args) {
 String name = JOptionPane.showInputDialog("Enter your name:");
 JOptionPane.showMessageDialog(null, "Hello, " + name + "!");
 }
}
```

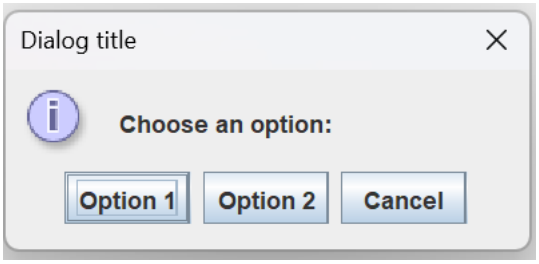
The execution result is:



- Display a dialog box with multiple options

```
package lab4;
import javax.swing.JOptionPane;

public class OptionDialogExample {
public static void main(String[] args) {
String[] options = {"Option 1", "Option 2", "Cancel"};
int choice = JOptionPane.showOptionDialog(null,
"Choose an option:",
"Dialog title",
JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE,
null,
options,
options[0]);

if (choice != -1) {
JOptionPane.showMessageDialog(null, "You have chosen: " + options[choice]);
}
}
}
```

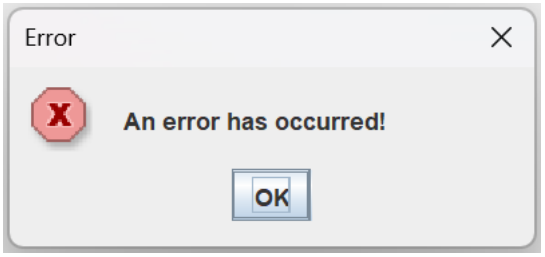
The execution result is:



- Display a simple error message

```
package lab4;
import javax.swing.JOptionPane;

public class ErrorMessageExample {
public static void main(String[] args) {
JOptionPane.showMessageDialog(null, "An error has occurred!", "Error",
JOptionPane.ERROR_MESSAGE);
}
}
```

The execution result is:



4.4.6. Design Practices

For an effective Human-Computer Interface (HCI), it is essential to follow certain best practices:

- **Simplicity**: Keep the interface clear and intuitive.
- **Consistency**: Use similar design elements for similar actions.
- **Accessibility**: Make the HMI accessible to all users, including those with disabilities.

4.5. Exercises

Exercise 2: Creating a Simple Calculator

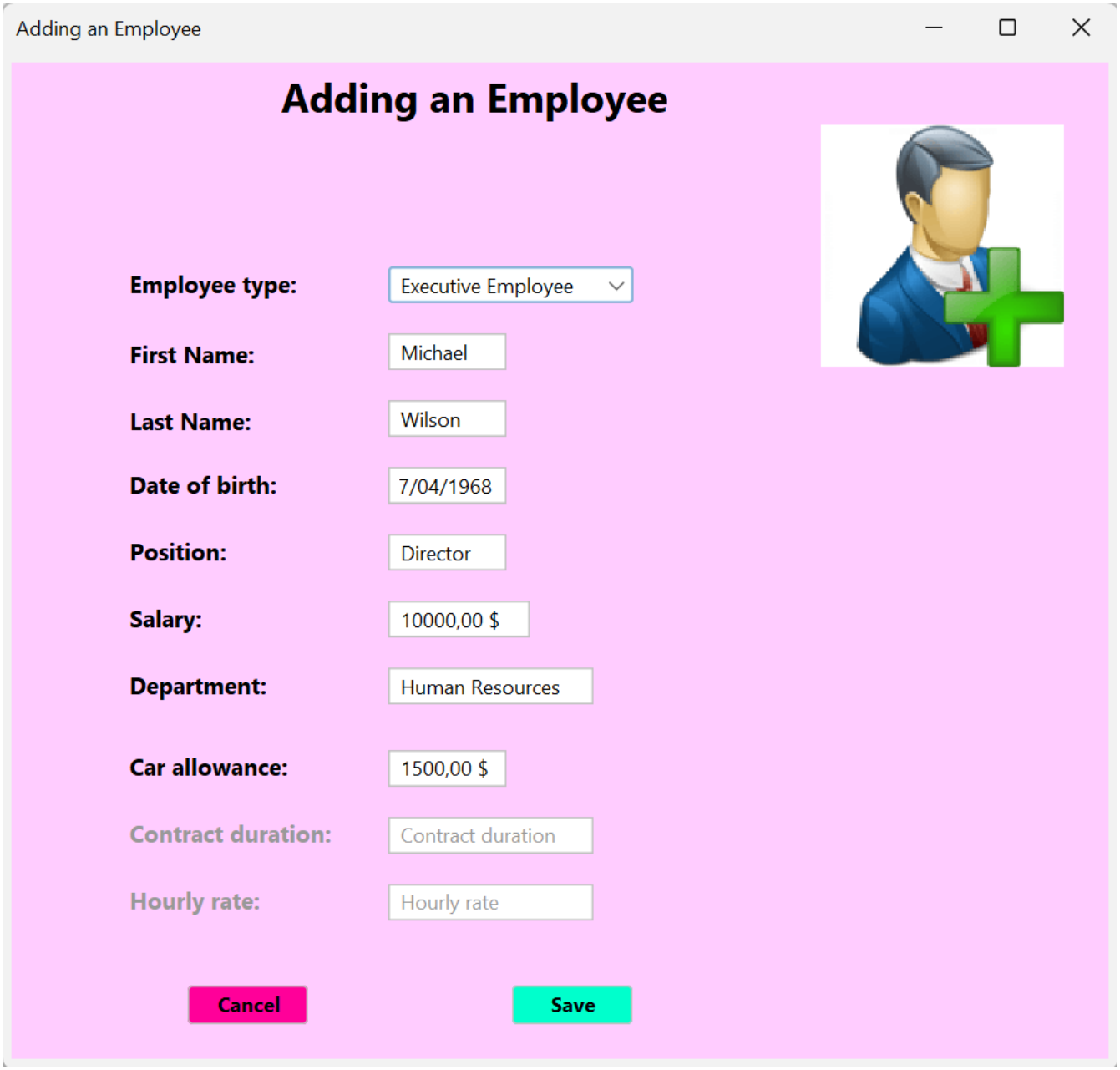
Goal: Create a graphical interface for a basic calculator that can perform addition, subtraction, multiplication, and division operations.

Steps:

1. Window Creation:
- Create a main window (JFrame) with the title ''Calculator''.
2. Adding Components:
- Add two text fields (JTextFields) to enter numbers.
- Add buttons (JButtons) for the operations: +, -, *, /, and = to calculate the result.
- Add a text field (JTextField) to display the result.
3. Event Handling:
- Implement event listeners (ActionListeners) for each button to retrieve the field values, perform the appropriate calculation, and display the result.
4. Layout:
- Use a layout manager (e.g., GridLayout) to organize components intuitively.

Exercise 2:

In this exercise, we'll delve deeper into using classes from the Java Swing package, using the NetBeans integrated development environment (IDE) for the user interface design phase. To illustrate this, we'll use the employee management project presented in **Lab 3** as a starting point. Your task will be to create a single graphical interface using Java Swing, allowing you to add a manager or temporary employee. The interface should be based on the example below:



Exercise 3 :

Creating a graphical interface for an application is essential in the development process. Using an Integrated Development Environment (IDE) allows this task to be divided into two distinct phases:

a. Design Phase

This stage involves designing the visual representation of the interface. No code is required at this stage. The developer selects components from palettes and arranges them in the window. They can also adjust the properties of these components, such as their position, size, or color.

b. Event-Driven Programming Phase

This phase requires writing event handlers, also known as event procedures. This is where the interface becomes interactive. Object-oriented programming principles are applied, with windows and their components represented by classes.

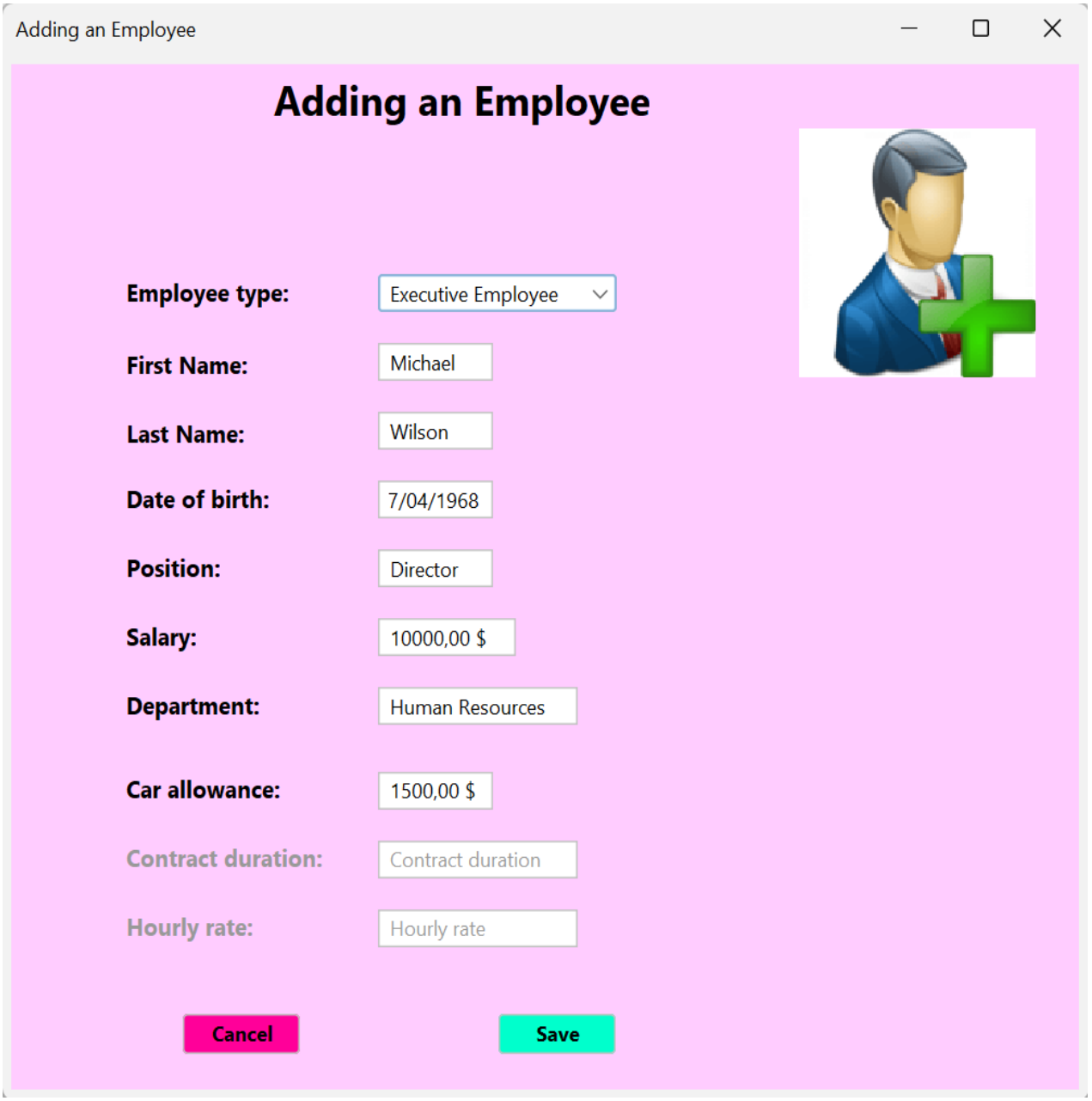
In this exercise, we will delve deeper into the use of classes from the Java Swing package through the NetBeans IDE for the design phase. Let's take an employee management project as an example, as discussed in Lab 3.

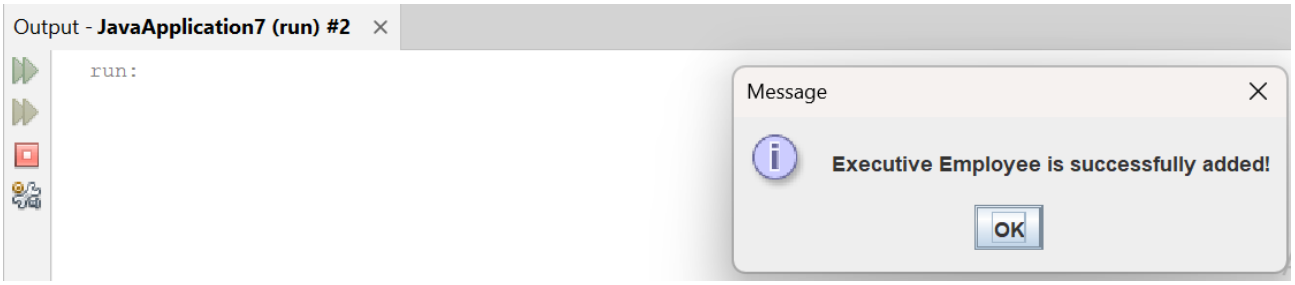
Tasks to be Completed

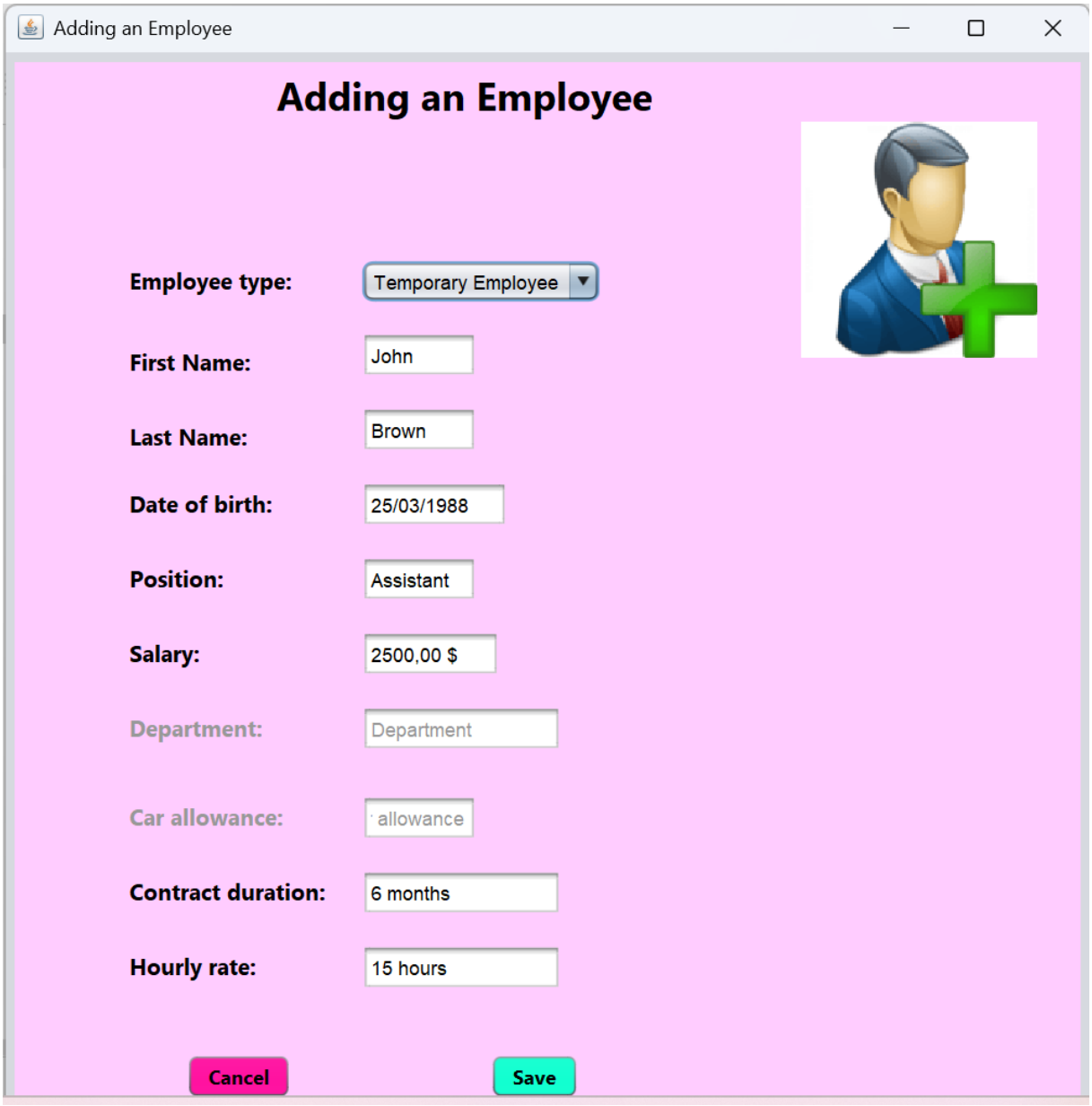
Using the graphical interface created previously with Java Swing, which allows you to add a Executive employee or Temporary employee:

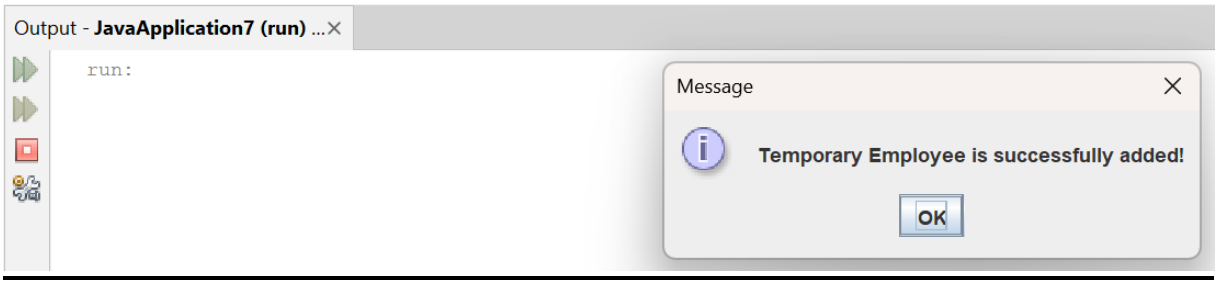
a) Implement the JButtonActionPerformed method for the ''Cancel'' button to reset the values of the text fields (JTextField) when the user clicks on it.

b) Integrate JOptionPane dialogs as shown in the interfaces below:


Adding an Employee
Adding an Employee
Employee type:
Executive Employee
First Name:
Michael
Last Name:
Wilson
Date of birth:
7/04/1968
Position:
Director
Salary:
10000,00 $
Department:
Human Resources
Car allowance:
1500,00 $
Contract duration:
Contract duration
Hourly rate:
Hourly rate
Cancel
Save


Output - JavaApplication7 (run) #2
run:
Message
Executive Employee is successfully added!
OK





4.6. Exercise corrections

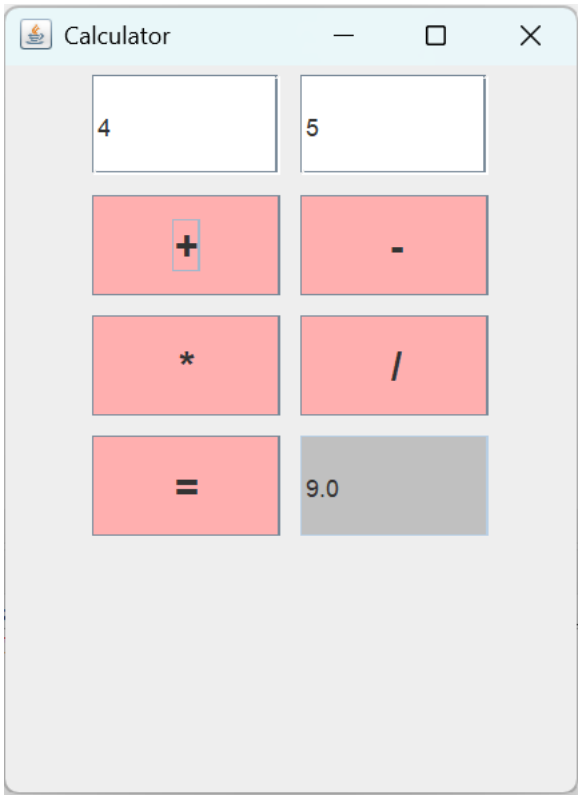
Answers to exercise 1:

```
package lab4;

/**
*
* @author USER.COM
```

```
*/
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Calculator {
public static void main(String[] args) {
JFrame frame = new JFrame("Calculator");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(300, 400);
frame.setLayout(new FlowLayout());

JTextField num1 = new JTextField(10);
JTextField num2 = new JTextField(10);
JTextField result = new JTextField(10);
result.setEditable(false);
result.setBackground(Color.LIGHT_GRAY);

JButton addButton = new JButton("+");
JButton subButton = new JButton("-");
JButton mulButton = new JButton("*");
JButton divButton = new JButton("/");
JButton equalButton = new JButton("=");

// Button style
for (JButton button: new JButton[]{addButton, subButton, mulButton,
                                   divButton,
                                   equalButton}) {
button.setFont(new Font("Arial", Font.BOLD, 20));
button.setBackground(Color.PINK);
button.setPreferredSize(new Dimension(50, 50));
}

JPanel panel = new JPanel();
panel.setLayout(new GridLayout(5, 2, 10, 10));
panel.add(num1);
panel.add(num2);
panel.add(addButton);
panel.add(subButton);
panel.add(mulButton);
panel.add(divButton);
panel.add(equalButton);
panel.add(result);

frame.add(panel);
frame.setVisible(true);
frame.setLocationRelativeTo(null); // Center the window

addButton.addActionListener(e -> {
double n1 = Double.parseDouble(num1.getText());
double n2 = Double.parseDouble(num2.getText());
result.setText(String.valueOf(n1 + n2));
});

subButton.addActionListener(e -> {
double n1 = Double.parseDouble(num1.getText());
double n2 = Double.parseDouble(num2.getText());
result.setText(String.valueOf(n1 - n2));
});
```

```
mulButton.addActionListener(e -> {
double n1 = Double.parseDouble(num1.getText());
double n2 = Double.parseDouble(num2.getText());
result.setText(String.valueOf(n1 * n2));
});

divButton.addActionListener(e -> {
double n1 = Double.parseDouble(num1.getText());
double n2 = Double.parseDouble(num2.getText());
if (n2 != 0) {
result.setText(String.valueOf(n1 / n2));
} else {
result.setText("Error");
}
});
}
}
```

The execution result is:



Answers to exercise 2:

```
package graphical.interfaces;

import javax.swing.JOptionPane;

/**
 *
 * @author Dr Mohamed Mohammedi
 */
public class EmployeeInterface extends javax.swing.JFrame {

    /**
```

```
     * Creates new form ExecutiveEmployee
     */
    public EmployeeInterface() {
        initComponents();// Method generated to initialize components

// Check the current selection and disable fields if necessary
    if (jComboBox1.getSelectedItem().toString().trim().equals("Executive
Employee")) {
        jLabel10.setEnabled(false);
        jTextField8.setEnabled(false);
        jLabel11.setEnabled(false);
        jTextField9.setEnabled(false);
    }
    }

    /**
     * This method is called from within the constructor to initialize the
form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jComboBox1 = new javax.swing.JComboBox<>();
        jTextField1 = new javax.swing.JTextField();
        jTextField2 = new javax.swing.JTextField();
        jTextField3 = new javax.swing.JTextField();
        jTextField4 = new javax.swing.JTextField();
        jTextField5 = new javax.swing.JTextField();
        jLabel8 = new javax.swing.JLabel();
        jTextField6 = new javax.swing.JTextField();
        jLabel9 = new javax.swing.JLabel();
        jTextField7 = new javax.swing.JTextField();
        jLabel10 = new javax.swing.JLabel();
        jTextField8 = new javax.swing.JTextField();
        jLabel11 = new javax.swing.JLabel();
        jTextField9 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jLabel13 = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Adding an Employee");

        jPanel1.setBackground(new java.awt.Color(255, 204, 255));
        jPanel1.setAlignmentX(0.1F);
        jPanel1.setAlignmentY(0.1F);
        jPanel1.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
```

```
        jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 24)); // NOI18N
        jLabel1.setText("Adding an Employee");

        jLabel2.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel2.setText("Employee type:");

        jLabel3.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel3.setText("First Name:");

        jLabel4.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel4.setText("Last Name:");

        jLabel5.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel5.setText("Date of birth:");

        jLabel6.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel6.setText("Position:");

        jLabel7.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel7.setText("Salary:");

        jComboBox1.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Executive Employee", "Temporary Employee" }));
        jComboBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jComboBox1ActionPerformed(evt);
            }
        });

        jTextField1.setText("Michael");

        jTextField2.setText("Wilson");

        jTextField3.setText("07/04/1968");

        jTextField4.setText("Director");

        jTextField5.setText("10000,00 $");

        jLabel8.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel8.setText("Department:");

        jTextField6.setText("Human Resources");

        jLabel9.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel9.setText("Car allowance:");

        jTextField7.setText("1500,00 $");

        jLabel10.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel10.setText("Contract duration:");

        jTextField8.setText("Contract duration");

        jLabel11.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel11.setText("Hourly rate:");

        jTextField9.setText("Hourly rate");

        jButton1.setBackground(new java.awt.Color(255, 0, 153));
```

```
        jButton1.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton1.setText("Cancel");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jButton2.setBackground(new java.awt.Color(0, 255, 204));
        jButton2.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton2.setText("Save");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jLabel13.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/Graphical_Interfaces/busines
sman_add (2).png"))); // NOI18N

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(161, 161, 161)
                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 92,
Short.MAX_VALUE)
                .addComponent(jLabel13)
                .addGap(27, 27, 27))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(70, 70, 70)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel11)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addComponent(jLabel10)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                    .addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                                    .addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField4,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField8,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField9,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING, false)
                                        .addComponent(jTextField3,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jTextField5,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 85, Short.MAX_VALUE))))

.addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                                    .addComponent(jLabel5,
javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel2,
javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel3,
javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel4,
javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel6,
javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel7,
javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel8,
javax.swing.GroupLayout.Alignment.LEADING)
                                    .addComponent(jLabel9,
javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                                        .addGap(35, 35, 35)
                                        .addComponent(jButton1)
                                        .addGap(122, 122, 122)
                                        .addComponent(jButton2)))
                                .addGap(0, 0, Short.MAX_VALUE)))
                        .addContainerGap(284, Short.MAX_VALUE))))
```

```
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(37, 37, 37)
                .addComponent(jLabel13)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(0, 0, Short.MAX_VALUE)
                        .addComponent(jLabel3))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 82,
Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                            .addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel2))
                        .addGap(18, 18, 18)
                        .addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                    .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel4))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jLabel5)
                    .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
```

```
                    .addComponent(jTextField4,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel6))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField5,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel8))
                .addGap(27, 27, 27)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel9))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField8,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel10))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jLabel11)
                    .addComponent(jTextField9,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(39, 39, 39)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jButton2)
                    .addComponent(jButton1))
                .addContainerGap(21, Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
```

```
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    }

    private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt)
{
        // TODO add your handling code here:
        //Always activate the fields
    jLabel10.setEnabled(true);
    jTextField8.setEnabled(true);
    jLabel11.setEnabled(true);
    jTextField9.setEnabled(true);

    jLabel8.setEnabled(true);
    jTextField6.setEnabled(true);
    jLabel9.setEnabled(true);
    jTextField7.setEnabled(true);

    String Rec = jComboBox1.getSelectedItem().toString().trim();

    // Disable fields based on selection
    if (Rec.equals("Executive Employee")) {
        jLabel10.setEnabled(false);
        jTextField8.setEnabled(false);
        jLabel11.setEnabled(false);
        jTextField9.setEnabled(false);
    } else {
        jLabel8.setEnabled(false);
        jTextField6.setEnabled(false);
        jLabel9.setEnabled(false);
        jTextField7.setEnabled(false);
    }
```

```
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel
setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with
the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new EmployeeInterface().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JComboBox<String> jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
```

```
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    private javax.swing.JTextField jTextField6;
    private javax.swing.JTextField jTextField7;
    private javax.swing.JTextField jTextField8;
    private javax.swing.JTextField jTextField9;
    // End of variables declaration
}
```

Note:

Using **trim()** is good practice to ensure you are working with clean strings free of unwanted spaces, thus preventing logical errors in the program.

Answers to exercise 3:

```
package graphical.interfaces;

import javax.swing.JOptionPane;

/**
 *
 * @author Dr Mohamed Mohammedi
 */
public class EmployeeInterface extends javax.swing.JFrame {

    /**
     * Creates new form ExecutiveEmployee
     */
    public EmployeeInterface() {
        initComponents();// Method generated to initialize components

// Check the current selection and disable fields if necessary
    if (jComboBox1.getSelectedItem().toString().trim().equals("Executive
Employee")) {
        jLabel10.setEnabled(false);
        jTextField8.setEnabled(false);
        jLabel11.setEnabled(false);
        jTextField9.setEnabled(false);
    }
    }

    /**
     * This method is called from within the constructor to initialize the
form.
```

```
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jComboBox1 = new javax.swing.JComboBox<>();
        jTextField1 = new javax.swing.JTextField();
        jTextField2 = new javax.swing.JTextField();
        jTextField3 = new javax.swing.JTextField();
        jTextField4 = new javax.swing.JTextField();
        jTextField5 = new javax.swing.JTextField();
        jLabel8 = new javax.swing.JLabel();
        jTextField6 = new javax.swing.JTextField();
        jLabel9 = new javax.swing.JLabel();
        jTextField7 = new javax.swing.JTextField();
        jLabel10 = new javax.swing.JLabel();
        jTextField8 = new javax.swing.JTextField();
        jLabel11 = new javax.swing.JLabel();
        jTextField9 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jLabel13 = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Adding an Employee");

        jPanel1.setBackground(new java.awt.Color(255, 204, 255));
        jPanel1.setAlignmentX(0.1F);
        jPanel1.setAlignmentY(0.1F);
        jPanel1.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));

        jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 24)); // NOI18N
        jLabel1.setText("Adding an Employee");

        jLabel2.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel2.setText("Employee type:");

        jLabel3.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel3.setText("First Name:");

        jLabel4.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel4.setText("Last Name:");

        jLabel5.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel5.setText("Date of birth:");

        jLabel6.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel6.setText("Position:");
```

```
        jLabel7.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel7.setText("Salary:");

        jComboBox1.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Executive Employee", "Temporary Employee" }));
        jComboBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jComboBox1ActionPerformed(evt);
            }
        });

        jTextField1.setText("Michael");

        jTextField2.setText("Wilson");

        jTextField3.setText("07/04/1968");

        jTextField4.setText("Director");

        jTextField5.setText("10000,00 $");

        jLabel8.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel8.setText("Department:");

        jTextField6.setText("Human Resources");

        jLabel9.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel9.setText("Car allowance:");

        jTextField7.setText("1500,00 $");

        jLabel10.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel10.setText("Contract duration:");

        jTextField8.setText("Contract duration");

        jLabel11.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel11.setText("Hourly rate:");

        jTextField9.setText("Hourly rate");

        jButton1.setBackground(new java.awt.Color(255, 0, 153));
        jButton1.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton1.setText("Cancel");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jButton2.setBackground(new java.awt.Color(0, 255, 204));
        jButton2.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton2.setText("Save");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });
```

```
        jLabel13.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/Graphical_Interfaces/busines
sman_add (2).png"))); // NOI18N

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(161, 161, 161)
                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 92,
Short.MAX_VALUE)
                .addComponent(jLabel13)
                .addGap(27, 27, 27))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(70, 70, 70)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel11)

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addComponent(jLabel10)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                    .addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField4,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                    .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                                        .addComponent(jTextField8,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                        .addComponent(jTextField9,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING, false)
                                            .addComponent(jTextField3,
javax.swing.GroupLayout.Alignment.LEADING)
                                            .addComponent(jTextField5,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 85, Short.MAX_VALUE))))

.addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                                        .addComponent(jLabel5,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel2,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel3,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel4,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel6,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel7,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel8,
javax.swing.GroupLayout.Alignment.LEADING)
                                        .addComponent(jLabel9,
javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                                            .addGap(35, 35, 35)
                                            .addComponent(jButton1)
                                            .addGap(122, 122, 122)
                                            .addComponent(jButton2)))
                                    .addGap(0, 0, Short.MAX_VALUE)))
                            .addContainerGap(284, Short.MAX_VALUE))))
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(37, 37, 37)
                .addComponent(jLabel13)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
```

```
                        .addGap(0, 0, Short.MAX_VALUE)
                        .addComponent(jLabel3))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 82,
Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                            .addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel2))
                        .addGap(18, 18, 18)
                        .addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)
                    .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel4))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jLabel5)
                    .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField4,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel6))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField5,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7))
                .addGap(18, 18, 18)
```

```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel8))
                .addGap(27, 27, 27)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel9))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField8,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel10))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jLabel11)
                    .addComponent(jTextField9,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(39, 39, 39)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jButton2)
                    .addComponent(jButton1))
                .addContainerGap(21, Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Display a confirmation message
    String Rec = jComboBox1.getSelectedItem().toString().trim();

    if(Rec.equals("Executive Employee")) {
    JOptionPane.showMessageDialog(null, "Executive Employee has been
successfully added!");
    } else {
    JOptionPane.showMessageDialog(null, "Temporary Employee has been
successfully added!");
    }
    }

    private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt)
{
        // TODO add your handling code here:
        //Always activate the fields
    jLabel10.setEnabled(true);
    jTextField8.setEnabled(true);
    jLabel11.setEnabled(true);
    jTextField9.setEnabled(true);

    jLabel8.setEnabled(true);
    jTextField6.setEnabled(true);
    jLabel9.setEnabled(true);
    jTextField7.setEnabled(true);

    String Rec = jComboBox1.getSelectedItem().toString().trim();

    // Disable fields based on selection
    if (Rec.equals("Executive Employee")) {
        jLabel10.setEnabled(false);
        jTextField8.setEnabled(false);
        jLabel11.setEnabled(false);
        jTextField9.setEnabled(false);
    } else {
        jLabel8.setEnabled(false);
        jTextField6.setEnabled(false);
        jLabel9.setEnabled(false);
        jTextField7.setEnabled(false);
    }
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
       //Reset the fields
```

```
        jTextField1.setText("");
        jTextField2.setText("");
        jTextField3.setText("");
        jTextField4.setText("");
        jTextField5.setText("");
        jTextField6.setText("");
        jTextField7.setText("");
        jTextField8.setText("");
        jTextField9.setText("");
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel
setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with
the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new EmployeeInterface().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
```

```
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JComboBox<String> jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    private javax.swing.JTextField jTextField6;
    private javax.swing.JTextField jTextField7;
    private javax.swing.JTextField jTextField8;
    private javax.swing.JTextField jTextField9;
    // End of variables declaration
}
```

Lab 5: The Three Types of Software Architecture: MVC, Three Tiers and Service-Oriented

Outline	5.1 Introduction 5.2 Part 1: Three-Tier Architecture 5.3. Part 2: Service-Oriented Architecture 5.4. Part 3: MVC (Model-View-Controller) Architecture 5.5 Exercises 5.6 Exercise Corrections
Goals	➢ Understand the key concepts and components of each architecture (MVC, three-tier, service-oriented). ➢ Analyze the advantages and disadvantages to facilitate architectural selection. ➢ Explore data flows to understand how each architecture works. ➢ Apply knowledge to practical examples to illustrate their implementation. ➢ Encourage critical thinking about architectural choices tailored to a project's needs. ➢ Promote application modularity and maintainability.

5.1. Introduction

In this chapter, we will explore the three types of software architecture: MVC, 3-tier architecture, and service-oriented architecture. Each of these architectures has unique characteristics and advantages that make them suitable for different types of applications.

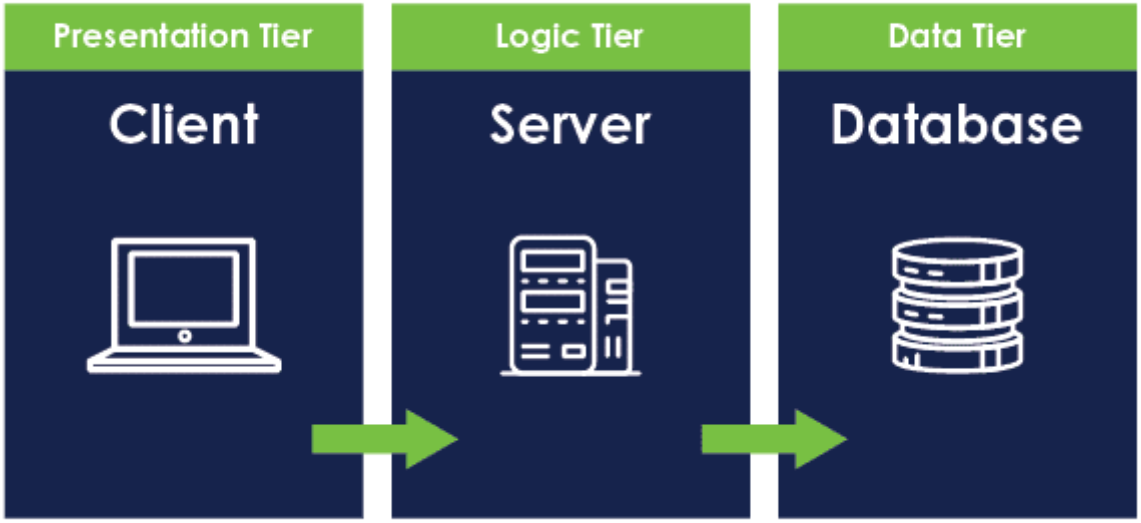
In this chapter, we will explore the three types of software architecture: MVC, 3-tier architecture, and service-oriented architecture. Each of these architectures has unique characteristics and advantages that make them suitable for different types of applications.

5.2. Part 1: Three-Tier Architecture

This part guides you through the implementation of a simple application using the three-tier architecture, which will help you better understand the concepts.

1. Introduction to Three-Tier Architecture

Three-tier architecture is a design pattern that divides an application into three distinct layers: presentation, business logic, and data access. This separation helps improve code management, maintenance, and scalability.



2. Components of Three-Tier Architecture

- **Presentation Layer:**

a. **Definition:** The presentation layer is responsible for the user interface. It displays data and collects user input.

b. **Functions:**
   - Display data in an understandable form.
   - Receive user input and pass it to business logic.
   - Update in response to data changes.

- **Business Logic Layer:**

a. **Definition :** The business logic layer contains the rules and logic of the application. It processes the data and performs the required operations.

b. **Functions :**
   - Interact with the data access layer to retrieve or modify data.
   - Apply business rules.
   - Provide methods to manipulate data.

- **Data Access Layer:**

a. **Definition :** The data access layer manages interaction with the database. It is responsible for data persistence.

b. **Functions :**
   - Perform CRUD (Create, Read, Update, Delete) operations.
   - Provide an interface to access data securely and efficiently.

3. Data Flow in Three-Tier Architecture

The data flow in a three-tier system generally follows the following pattern:

1. The user interacts with the presentation layer (e.g., clicking a button).
2. The presentation layer sends the event to the business logic layer.
3. The business logic processes the event and interacts with the data access layer if necessary.
4. The data access layer performs the required operations.
5. The business logic returns the results to the presentation layer.
6. The presentation layer updates to reflect the changes.

4. Mise en œuvre d'une Application Simple avec l'Architecture Trois Tiers

To integrate the three-tier architecture into a sample note management system, we will clearly define the roles of the classes. Here's how we will structure the code:

- Data Access Layer: Represents the notes and uses an array to store them (e.g., a NoteDAO class).
- Business Logic Layer: Manages the addition of new notes (e.g., a NoteService class).
- Presentation Layer: Displays the notes and a prompt to add a new note (e.g., a NoteView class).

Example: Note Management Application

```
package example.three.tiers.architecture;

import java.util.Scanner;

//Data Access Layer:

//Note Class
class Note {
private String title;
private String content;

public Note(String title, String content) {
this.title = title;
this.content = content;
}

public String getTitle() {
return title;
}

public String getContent() {
return content;
}
}

//Data Access Class
class NoteDAO {
private Note[] notes;
```

```
private int counter;

public NoteDAO(int capacity) {
notes = new Note[capacity]; // Create an array of notes
counter = 0;
}

public void addNote(Note note) {
if (counter < notes.length) {
notes[counter] = note;
counter++;
} else {
System.out.println("Maximum capacity reached.");
}
}

public Note[] getNotes() {
return notes;
}

public int getCounter() {
return counter;
}
}

//Business Logic Layer:

//Service class for notes
class NoteService {
private NoteDAO noteDAO;

public NoteService(NoteDAO noteDAO) {
this.noteDAO = noteDAO;
}

public void addNote(String title, String content) {
Note newNote = new Note(title, content);
noteDAO.addNote(newNote);
}

public Note[] getNotes() {
return noteDAO.getNotes();
}

public int getCounter() {
return noteDAO.getCounter();
}
}

//Presentation Layer:
//Class to display notes
class NoteView {
public void displayNotes(Note[] notes, int counter) {
System.out.println("List of Notes:");
for (int i = 0; i < counter; i++) {
System.out.println("- " + notes[i].getTitle() + ": " +
notes[i].getContent());
}
}
```

```
public void displayPrompt() {
System.out.println("Enter the title and content of the note:");
}
}

//Using the Application
public class Main {
public static void main(String[] args) {
NoteDAO noteDAO = new NoteDAO(5); // Maximum capacity: 5 notes
NoteService noteService = new NoteService(noteDAO);
NoteView noteView = new NoteView();

Scanner scanner = new Scanner(System.in);

for (int i = 0; i < 2; i++) { // Add two notes as an example
noteView.displayPrompt();
System.out.print("Title: ");
String title = scanner.nextLine();
System.out.print("Content: ");
String content = scanner.nextLine();
noteService.addNote(title, content);
}

noteView.displayNotes(noteService.getNotes(), noteService.getCounter());
}
}
```

The execution result is:

```
Enter the title and content of the note:
Title:  Software Engineering
Content: 17
Enter the title and content of the note:
Title: Object Oriented Programming
Content: 15
List of Notes:
-  Software Engineering: 17
- Object Oriented Programming: 15
```

5. Advantages of the Three-Tier Architecture

- Separation of Concerns: Each layer has a distinct responsibility, facilitating code understanding and evolution.
- Testability: Layers can be tested independently, simplifying the unit testing process.
- Reusability: Components can be reused in other applications or contexts.
- Maintainability: Changes to one layer do not directly affect other layers, reducing the risk of introducing bugs during updates.

Note:

Three-tier architecture is a powerful model for building modular and maintainable applications. By applying this model, developers can create more robust and scalable applications, facilitating collaboration and continuous development.

5.3. Part 2: Service-Oriented Architecture

This part guides you through implementing a simple application using service-oriented architecture, which will help you better understand the concepts.

1. Introduction to Service-Oriented Architecture

Service-oriented architecture (SOA) is a design pattern that structures an application as independent services. Each service is responsible for a specific functionality and communicates with other services through well-defined interfaces. This allows for more flexible, scalable, and reusable systems.

2. Components of Service-Oriented Architecture

- **Service:**
  a. **Definition**: A service is a self-contained functional unit that performs specific operations, often exposed through an API.

  b. **Functions**:
  - Manage business logic.
  - Interact with data.
  - Expose APIs for communication with other services or the user interface.

- **Interface**:
  a. **Definition**: The interface defines how services communicate with each other and with other components, often in the form of REST or SOAP[1] APIs.

  b. **Functions**:
  - Define the methods available for each service.
  - Manage requests and responses between services.

- **Client**:
  a. Definition: The client is the user interface or other service that consumes the exposed services.

  b. **Functions**:
  - Send requests to services.
  - Receive and display data.

[1] REST and SOAP are different approaches to transmitting data online. More specifically, both define how to develop application programming interfaces (APIs) that enable data exchange between multiple web applications. REST (Representational State Transfer) is a set of architectural principles. SOAP (Simple Object Access Protocol) is an official protocol managed by the W3C (World Wide Web Consortium). The main difference between the two is that SOAP is a protocol, REST is not. In general, APIs follow either the REST or SOAP approach depending on their use and the developer's preferences.

3. Data Flow in Service-Oriented Architecture

The data flow in a service-oriented system generally follows the following pattern:

1. The client sends a request to a service via an API.
2. The service processes the request, interacting with other services or the data layer if necessary.
3. The service returns a response to the client.
4. The client displays the results to the user.

4. Implementing a Simple Application with Service-Oriented Architecture

To integrate service-oriented architecture into a sample Notes management system, we'll clearly define the roles of the classes. Here's how we'll structure the code:

- **Service**: Service: Handles adding and retrieving notes (e.g., a **NoteService** class).
- **Interface**: Exposes application methods (e.g., an **INoteService** interface).
- **Client**: Displays notes and a prompt to add a new note (e.g., a **NoteClient** class).

Example: Note Management Application

```
package example.architecture.service;

import java.util.Scanner;

//Service:
//Note Class
class Note {
private String title;
private String content;

public Note(String title, String content) {
this.title = title;
this.content = content;
}

public String getTitle() {
return title;
}

public String getContent() {
return content;
}
}

//Note Service Interface
interface INoteService {
void addNote(String title, String content);
Note[] getNotes();
}

//Note Service Class
class NoteService implements INoteService {
private Note[] notes;
private int counter;
```

```
public NoteService(int capacity) {
notes = new Note[capacity]; // Create an array to store the notes
counter = 0;
}

@Override
public void addNote(String title, String content) {
if (counter < notes.length) {
Note newNote = new Note(title, content);
notes[counter] = newNote;
counter++;
} else {
System.out.println("Maximum capacity reached.");
}
}

@Override
public Note[] getNotes() {
return notes;
}

public int getCounter() {
return counter;
}
}

//Client:
//Class to manage the user interface
class NoteClient {
private INoteService noteService;

public NoteClient(INoteService noteService) {
this.noteService = noteService;
}

public void addNote() {
Scanner scanner = new Scanner(System.in);
System.out.println("Enter the title and content of the note:");
System.out.print("Title: ");
String title = scanner.nextLine();
System.out.print("Content: ");
String content = scanner.nextLine();
noteService.addNote(title, content);
}

public void displayNotes() {
Note[] notes = noteService.getNotes();
System.out.println("List of Notes:");
for (int i = 0; i < ((NoteService) noteService).getCounter(); i++) {
System.out.println("- " + notes[i].getTitle() + ": " +
notes[i].getContent());
}
}
}

//Using the Application
public class Main {
public static void main(String[] args) {
NoteService noteService = new NoteService(5); // Maximum capacity: 5 notes
NoteClient noteClient = new NoteClient(noteService);
```

```
// Add notes
noteClient.addNote(); // Call to add a note
noteClient.addNote(); // You can call it multiple times to add more notes

// Display notes
noteClient.displayNotes();
}
}
```

The execution result is:

```
Enter the title and content of the note:
Title: Software Engineering
Content: 17
Enter the title and content of the note:
Title: Object Oriented Programming
Content: 15
List of Notes:
- Software Engineering: 17
- Object Oriented Programming: 15
```

5. Advantages of Service-Oriented Architecture

- **Flexibility:** Services can be modified or replaced without affecting the entire system.
- **Scalability:** Allows services to be deployed independently, facilitating scalability.
- **Reusability:** Services can be used in different applications or contexts.
- **Maintenance:** Changes to services are easier to manage, reducing the risk of introducing bugs during updates.

Note:

Service-oriented architecture is a powerful model for building modular and maintainable applications. By applying this model, developers can create more robust and scalable applications, facilitating collaboration and continuous development.

5.4. Part 3: MVC (Model-View-Controller) Pattern

1. Introduction to MVC Pattern

This chapter part guide you through the implementation of a simple application based on the MVC (Model-View-Controller) pattern. By adopting this approach, you'll discover how to structure your code in a way that separates concerns, making it easier to maintain and evolve your application. You'll also learn how to integrate object-oriented programming concepts into a Java project, giving you a better understanding of creating modular and extensible software.

2. MVC Pattern

The MVC pattern is a design pattern that separates an application into three main components, making code management and maintenance easier. This separation makes applications more modular and testable.

3. Components of the MVC Pattern

❖ **Model**:

a. **Definition:** The model represents the application logic and data. It is responsible for managing business rules and data access.

b. **Functions**:

- Store and manage data.
- Notify the view of data changes.
- Provide methods for manipulating data (adding, deleting, updating).

❖ **View**:

a. **Definition**: The view is the visual representation of the application. It displays the data provided by the model and allows the user to interact with the application.

b. **Functions**:

- Display data in an understandable form.
- Receive user input and pass it to the controller.
- Update itself in response to notifications from the model.

❖ **Controller**:

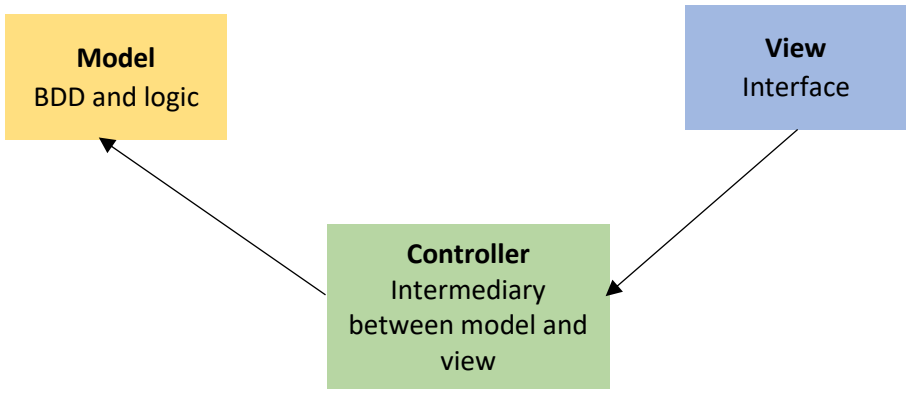
a. Definition: The controller acts as an intermediary between the model and the view. It handles user input, processes data, and updates the view accordingly.

b. Functions:

- Receives events from the view (clicks, taps, etc.).
- Interacts with the model to retrieve or modify data.
- Refreshes the view with new data.

As illustrated in the figure below, the MVC pattern is an architecture that divides an application into three components:

- **Model:** Manages data access, retrieval, updating, and validation.
- **View:** Represents the user interface and everything visible to the user.
- **Controller:** Serves as an intermediary between the model and the view.



4. Data Flow in MVC

Data flow in an MVC system generally follows the following pattern:

1. The user interacts with the view (e.g., click a button).
2. The view sends the event to the controller.
3. The controller processes the event and, if necessary, interacts with the model.
4. The model performs the required operations (data modification).
5. The model notifies the view of changes.
6. The view updates to reflect the changes in the model.

5. Implementing a Simple Application with MVC

To integrate the MVC architecture into a sample note management system, we'll clearly define the roles of the classes. Here's how we'll structure the code:

- Model: Represents a note with a title and content, and uses an array to store the notes (e.g., a Note class, a NoteModel class).
- View: Displays the notes and a prompt to add a new note (e.g., a NoteView class).
- Controller: Manages the addition of new notes and updates the view (e.g., a NoteController class).

Example: Note Management Application

```
package example.architecture.MVC;

import java.util.Scanner;

//**Model:**
//Note Class
class Note {
private String title;
private String content;

public Note(String title, String content) {
this.title = title;
this.content = content;
}

public String getTitle() {
return title;
}

public String getContent() {
return content;
}
}

//Notes Model
class NoteModel {
private Note[] notes;
private int counter;

public NoteModel(int capacity) {
```

```
notes = new Note[capacity]; // Create an array of notes
counter = 0;
}

public void addNote(Note note) {
if (counter < notes.length) {
notes[counter] = note;
counter++;
} else {
System.out.println("Maximum capacity reached.");
}
}
public Note[] getNotes() {
return notes;
}

public int getCounter() {
return counter;
}
}

//**View:**
//Class to display notes
class NoteView {
public void displayNotes(Note[] notes, int counter) {
System.out.println("List of Notes:");
for (int i = 0; i < counter; i++) {
System.out.println("- " + notes[i].getTitle() + ": " +
notes[i].getContent());
}
}

public void displayPrompt() {
System.out.println("Enter the title and content of the note:");
}
}

//**Controller:**
//Controller class
class NoteController {
private NoteModel model;
private NoteView view;

public NoteController(NoteModel model, NoteView view) {
this.model = model;
this.view = view;
}

public void addNote() {
view.displayPrompt();
Scanner scanner = new Scanner(System.in);

System.out.print("Title: ");
String title = scanner.nextLine();

System.out.print("Content: ");
String content = scanner.nextLine();

Note newNote = new Note(title, content);
model.addNote(newNote);
```

```
view.displayNotes(model.getNotes(), model.getCounter());
}
}

//Using the Application
public class Main {
public static void main(String[] args) {
//TODO Auto-generated method stub
NoteModel model = new NoteModel(5); // Maximum capacity: 5 notes
NoteView view = new NoteView();
NoteController controller = new NoteController(model, view);

controller.addNote(); // Call to add a note
controller.addNote(); // You can call it multiple times to add more notes
}
}
```

The execution result is:

```
Enter the title and content of the note:
Title: Software Engineering
Content: 17
List of Notes:
- Software Engineering: 17
Enter the title and content of the note:
Title: Object Oriented Programming
Content: 15
List of Notes:
- Software Engineering: 17
- Object Oriented Programming: 15
```

6. Advantages of MVC Pattern

- Separation of Concerns: Each component has a distinct responsibility, making it easier to understand and evolve the code.
- Testability: Components can be tested independently, simplifying the unit testing process.
- Reusability: Components can be reused in other applications or contexts.
- Maintenance: Changes to one component (e.g., the view) do not directly affect other components, reducing the risk of introducing bugs during updates.

Note:

MVC pattern is a powerful model for building modular and maintainable applications. By applying the MVC pattern, developers can create more robust and scalable applications, facilitating collaboration and continuous development.

5.5. Exercises

Exercise 1:

In this exercise, change the visibility of the **Employee** class's attributes from **Lab 3**. All attributes except **idEmployee** will now be **protected**, allowing direct access from derived classes, such as **TemporaryEmployee** and **ExecutiveEmployee**. This change makes it easier to

inherit and access employee properties in derived classes while maintaining an adequate level of encapsulation.

1. Integrate the Three Tiers architecture into the employee management system of **Lab 3**, clearly defining the role of each class. Here's how to structure the code:
   - **Data Access Layer:** Contains classes representing the application's data (e.g., **Employee**, **TemporaryEmployee**, **ExecutiveEmployee**, and an **EmployeeDAO** class for data management).
   - **Business Logic Layer:** Manages interactions between data and application operations (e.g., an **EmployeeManagement** class).
   - **Presentation Layer:** Responsible for displaying information to the user (e.g., an **EmployeeInterface** class).

2. Implement the same integration as in the previous question, but this time adopt the MVC architecture for the employee management system described in **Lab 3**. Clearly define the role of each class in this structure. Here's how you can organize the code:
   - **Model:** Will contain classes representing the application's data (e.g., **Employee**, **TemporaryEmployee**, **ExecutiveEmployee**).
   - **View:** Will be responsible for displaying information to the user (e.g., an **EmployeeView** class).
   - **Controller:** Will manage interactions between the model and the view, orchestrating updates and displays (e.g., an **EmployeeController** class).

3. Be sure to use the employee interface you carefully designed in Lab 4. When you click the "Save" button, carefully retrieve the information entered in the various fields, taking into account the employee category (Executive or temporary employee). Next, add a new employee to a table, using the methods provided by the employee management system code, implemented according to the Three-Tier architecture from the previous question. Finally, display all registered employees using a loop after clicking the "Displays" button.

Note:

In the Three-Tier architecture, it is recommended to use three distinct packages to organize the code and clearly manage responsibilities. The first layer, called the presentation layer or front-end, acts as the user interface, allowing users to interact with the application by displaying data and collecting input. The second, the business layer, often considered the middle tier, contains the application logic; it processes data from the presentation layer and applies the necessary business rules while interacting with the data layer. Finally, the data layer, or back-end, is responsible for storing and managing data, handling interactions with the database. While it is possible to organize these layers into a single package for simple projects, separating them into three distinct packages remains a good practice to ensure maintainability and clarity, especially in more complex applications.

5.6. Exercise corrections

Answers to exercise 1

1. Integrate the Three Tiers architecture into the employee management system of **Lab 3**, clearly defining the role of each class. Here's how to structure the code:

The following structure allows for better separation of concerns, thus facilitating code maintenance and evolution:

1. **Data Access Layer**:
   - Contains the **Employee**, **TemporaryEmployee**, **ExecutiveEmployee**, and **EmployeeDAO** classes to manage employee data.

2. **Business Logic Layer**:
   - The **EmployeeManagement** class handles employee-related operations, such as adding employees.

3. **Presentation Layer**:
   - The **EmployeeInterface** class is responsible for displaying information to the user.
4. **Main Class**:
   - The **EmployeeManagement** class orchestrates interactions between the different layers and executes the program.
   -

```
package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Basic employee.

*/
public class Employee {

private static long idEmployeCounter = 0; // ID counter

private long employeeID; // Remains private as instructed

protected String lastName;

protected String firstName;

protected String dateOfBirth;

protected String position;

protected double salary;

// Default constructor

      public Employee() {
      this.employeeID = 0;
```

```
        this.lastName = null;
        this.firstName = null;
        this.dateOfBirth = null;
        this.position = null;
        this.salary = 0.0;
        }

// Constructor with parameters

        public Employee(String lastname, String firstName, String
dateOfBirth, String position, double salary) {
        idEmployeeCounter++;

        employeeID = idEmployeeCounter;
        this.lastName = lastname;
        this.firstName = firstName;
        this.dateOfBirth = dateOfBirth;
        this.position = position;
        this.salary = salary;

        }

// Getter/Setter if needed
public String getLastName() {
return this.lastName;
}

public String getFirstName() {
return this.firstName;

}

        public String getDateOfBirth() {
        return this.dateOfBirth;

        }

// We expose the ID via a method (employeeID is private)`
        public long getEmployeeID() {
        return this.employeeID;

        }

        public void description() {
        System.out.println(this.lastName + " " + this.firstName + " is an
employee, " + this.position +", and he/she has a monthly salary of: " +
this.salary + " €.");
        }

        public void description(String prenom, String nom) {

        System.out.println("The employee ID " + this.lastName + " " +
this.firstName + " is: " + this.getEmployeeID());

        }

        public void increaseSalary(double amount) {
        this.salary += amount;

        }
```

```
	public double getAnnualSalary() {
	return this.salary * 12;

	}
}

package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Manager (inherits from Employee).

*/
public class ExecutiveEmployee extends Employee {

protected String department;

protected double carAllowance;

	public ExecutiveEmployee(String lastName, String firstName, String
dateofbirth, String position, double salary, String department, double
carAllowance) {
	super(lastName, firstName, dateofbirth, position, salary);

	this.department = department;
	this.carAllowance = carAllowance;

	}

// Method specific to the ExecutiveEmployee class

	public void displayDepartment() {

	System.out.println("The department of " + this.lastName + " " +
this.firstName + " born on " + this.dateOfBirth + " is: " +
this.department);

	}

}

package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Temporary Employee (inherits from Employee).

*/
public class TemporaryEmployee extends Employee {
```

```
protected int contractDuration;

protected double hourlyRate;

public TemporaryEmployee(String name, String firstName, String dateOfBirth,
String position, double salary,

int contractDuration, double hourlyRate) {
super(name, firstName, dateOfBirth, position, salary);

this.contractDuration = contractDuration;

this.hourlyRate = hourlyRate;

}

// Method specific to the TemporaryEmployee class

public void displayContract() {

System.out.println(this.lastName + " " + this.firstName + " born on " +
this.dateOfBirth + " has a temporary contract for a duration of " +
this.contractDuration + " months");

}
}

package data.access;

/**
 *
 * @author USER.COM
 */

/**

* Simple DAO that stores employees in an array.

* Basic operations: add, retrieve by index, search by first/last name,
list.

*/
public class DAOEmployee {

private Employee[] tableEmployees;

private int capacity;

private int size; // current number of employees

      public DAOEmployee(int capacity) {
      this.capacity = capacity;

      this.tableEmployees = new Employee[capacity];

      this.size = 0;

      }

// Add an employee (returns true if added, false if full)
```

```
	public boolean addEmployee(Employee e) {

	if (size >= capacity || e == null) {

	return false;

	}
	tableEmployees[size] = e;

	size++;

	return true;

	}

// Retrieve an employee by index
	public Employee getEmployee(int index) {
	if (index < 0 || index >= size) {
	return null;

	}
	return tableEmployees[index];

	}

// Search by last name/first name (returns first match or null)
	public Employee searchByLastNameFirstName(String lastName, String
firstName) {
	for (int i = 0; i < size; i++) {
	Employee e = tableEmployees[i];

	if (e != null && e.getLastName().equalsIgnoreCase(lastName) &&
e.getFirstName().equalsIgnoreCase(firstName)) {
	return e;

	}

	}
	return null;

	}

// List (returns the internal array and size via getters)
	public Employee[] getTableEmployees() {

	return tableEmployees;

	}

	public int getSize() {

	return size;

	}
}

package business.logic;

/**
```

```
 *
 * @author USER.COM
 */
import DataAccess.Employee;
import DataAccess.DAOEmployee;
import DataAccess.TemporaryEmployee;
import DataAccess.ExecutiveEmployee;

/**

* Business logic layer — employee management operation.

*/
public class EmployeeManagement {

private DAOEmployee dao;

      public EmployeeManagement(DAOEmployee dao) {
      this.dao = dao;

      }

// Adds an employee via the DAO

public boolean addEmployee(Employee e) {

return dao.addEmployee(e);

}

// Increases an employee's salary (search by last name/first name)

      public boolean increaseSalary(String lastName, String firstName,
double amount) {
      Employee e = dao.searchByLastNameFirstName(lastName, firstName);

      if (e == null) return false;

      e.increaseSalary(amount);

      return true;

      }

// Get annual salary
      public Double getAnnualSalary(String name, String firstName) {
      Employee e = dao.searchByLastNameFirstName(name, firstName);

      if (e == null) return null;

      return e.getAnnualSalary();

      }

// List all employees
public void listEmployees() {
Employee[] tab = dao.getTableEmployees();

int size = dao.getSize();
```

```
for (int i = 0; i < size; i++) {

if (tab[i] != null) {
tab[i].description();
}

}
}
// Examples of methods for specific types

public void displayTemporaryContract(String lastName, String firstName) {
Employee e = dao.searchByLastNameFirstName(lastName, firstName);

if (e instanceof TemporaryEmployee) {

((TemporaryEmployee) e).displayContract();

} else {

System.out.println("The employee is not a temporary employee.");

}

}

public void displayDepartmentExecutiveEmployee(String lastName, String
firstName) {
Employee e = dao.searchByLastNameFirstName(lastName, firstName);

if (e instanceof ExecutiveEmployee) {
((ExecutiveEmployee) e).displayDepartment();

} else {

System.out.println("The employee is not a manager.");

}

}
}

package presentation;
/**

* @author USER.COM

*/
import DataAccess.Employee;
import DataAccess.TemporaryEmployee;
import DataAccess.ExecutiveEmployee;
import DataAccess.DAOEmployee;
import BusinessLogic.EmployeeManagement;

/**

* Presentation layer: simple console interface.

*/
public class EmployeeInterface {
```

```
public static void main(String[] args) {

	// DAO initialization (capacity 10)
	DAOEmployee dao = new DAOEmployee(10);

	// business layer
	EmployeeManagement management = new EmployeeManagement(dao);

	// Create and add employees

	Employee employee1 = new Employee("HUBERT", "Christophe",
"12/03/1988", "Developer", 4800.0);

	management.addEmployee(employee1);

	Employee employee2 = new Employee("RENARD", "Nathalie", "02/01/1975",
"Manager", 7000.0);

	management.addEmployee(employee2);

	TemporaryEmployee temporaryEmployee = new TemporaryEmployee("DUPUIS",
"Pierre", "22/09/1980", "Assistant", 2500.0, 6, 15.0);

	management.addEmployee(temporaryEmployee);

	ExecutiveEmployee executiveEmployee = new
ExecutiveEmployee("LEBLANC", "Sophie", "07/04/1968", "Directeur", 10000.0,
"Ressources Humaines", 1500.0);
	management.addEmployee(executiveEmployee);

	// Displays via the presentation layer (uses the business layer)

	System.out.println("=== List of employees ===");

	management.listEmployees();

	System.out.println("\n=== IDs and annual salaries ===");

	employee1.description("HUBERT", "Christophe");

	employee1.description();

	management.increaseSalary("HUBERT", "Christophe", 500.0);

	System.out.println("Annual salary after increase: " +
management.getAnnualSalary("HUBERT", "Christophe") + " €.");

	System.out.println();

	employee2.description("RENARD", "Nathalie");

	employee2.description();

	System.out.println("The annual salary of " + employee2.getLastName()
+ " " + employee2.getFirstName() + " is: " + employee2.getAnnualSalary() +
" €.");

	System.out.println();

	temporaryEmployee.description("DUPUIS", "Pierre");
```

```
        temporaryEmployee.description();

        management.displayTemporaryContract("DUPUIS", "Pierre");

        System.out.println();

        executiveEmployee.description("LEBLANC", "Sophie");

        executiveEmployee.description();

        management.displayDepartmentExecutiveEmployee("LEBLANC", "Sophie");

        System.out.println("\n");

        // Display all registered employees using a loop

        System.out.println("=== Displaying registered employees (loop) ===");

        Employee[] all = dao.getTableEmployees();

        int size = dao.getSize();

        for (int i = 0; i < size; i++) {

        if (all[i] != null) {

        System.out.print((i + 1) + ". ");

        all[i].description();

        }

        }

}
}
```

The execution result is:

```
=== List of employees ===
HUBERT Christophe is an employee, Developer, and he/she has a monthly salary
of: 4800.0 €.
RENARD Nathalie is an employee, Manager, and he/she has a monthly salary of:
7000.0 €.
DUPUIS Pierre is an employee, Assistant, and he/she has a monthly salary of:
2500.0 €.
LEBLANC Sophie is an employee, Directeur, and he/she has a monthly salary of:
10000.0 €.

=== IDs and annual salaries ===
The employee ID HUBERT Christophe is: 1
HUBERT Christophe is an employee, Developer, and he/she has a monthly salary
of: 4800.0 €.
Annual salary after increase: 63600.0 €.

The employee ID RENARD Nathalie is: 2
RENARD Nathalie is an employee, Manager, and he/she has a monthly salary of:
7000.0 €.
The annual salary of RENARD Nathalie is: 84000.0 €.
```

```
The employee ID DUPUIS Pierre is: 3
DUPUIS Pierre is an employee, Assistant, and he/she has a monthly salary of:
2500.0 €.
DUPUIS Pierre born on 22/09/1980 has a temporary contract for a duration of
6 months

The employee ID LEBLANC Sophie is: 4
LEBLANC Sophie is an employee, Directeur, and he/she has a monthly salary of:
10000.0 €.
The department of LEBLANC Sophie born on 07/04/1968 is: Ressources Humaines


=== Displaying registered employees (loop) ===
1. HUBERT Christophe is an employee, Developer, and he/she has a monthly
salary of: 5300.0 €.
2. RENARD Nathalie is an employee, Manager, and he/she has a monthly salary
of: 7000.0 €.
3. DUPUIS Pierre is an employee, Assistant, and he/she has a monthly salary
of: 2500.0 €.
4. LEBLANC Sophie is an employee, Directeur, and he/she has a monthly salary
of: 10000.0 €.
```

2. Carry out the same integration as in the previous question, but this time adopting the MVC architecture for the employee management system described in Lab 3, clearly defining the role of each class:

```java
package model;

/**

* Basic employee.

*/
public class Employee {

private static long counterId = 0; // global counter

private long employeeId; // unique identifier for each instance

protected String lastName;

protected String firstName;

protected String dateOfBirth;

protected String position;

protected double salary;

// Default constructor

public Employee() {

this.employeeId = 0;

this.lastName = null;

this.firstName = null;
```

```
this.dateOfBirth = null;

this.position = null;

this.salary = 0.0;

}

// Constructor with parameters

public Employee(String lastName, String firstName, String dateOfBirth,
String position, double salary) {

this.employeeId = ++counterId; // unique ID per instance

this.lastName = lastName;

this.firstName = firstName;

this.dateOfBirth = dateOfBirth;

this.position = position;

this.salary = salary;

}

// Getters

public long getEmployeeId() {

return this.employeeId;

}

public String getLastName() {

return this.lastName;

}

public String getFirstName() {

return this.firstName;

}

public String getBirthDate() {

return this.dateOfBirth;

}

public double getSalary() {

return this.salary;

}
```

```
// Simple Business Methods

public void description() {

System.out.println(this.lastName + " " + this.firstName + " (" +
this.getEmployeeId() + ") is a " + this.position +

", monthly salary: " + this.salary + " €.");

}

public void identifiantDescription() {
System.out.println("The employee ID " + this.lastName + " " +
this.firstName + " is: " + this.getEmployeeId());

}

public void increaseSalary(double amount) {
this.salary += amount;

}

public double getAnnualSalary() {
return this.salary * 12;

}
}
```

```
package model;

/**

* ExecutiveEmployee (inherits from Employee).

*/
public class ExecutiveEmployee extends Employee {

protected String department;

protected double carAllowance;

public ExecutiveEmployee(String name, String firstName, String dateOfBirth,
String position, double salary,

String department, double carAllowance) {
super(name, firstName, dateOfBirth, position, salary);

this.department = department;

this.carAllowance = carAllowance;

}

// Method specific to the ExecutiveEmployee class

public void displayDepartment() {

System.out.println("The department of " + this.lastName + " " +
this.firstName + " (ID: " + this.getEmployeeId() + ") is: " +
this.department);
```

```
}

public String getDepartment() {

return this.department;

}

public double getCarAllowance() {
return this.carAllowance;
}
}

package model;

/**

* Temporary Employee (inherits from Employee).

*/
public class TemporaryEmployee extends Employee {

protected int contractDuration;

protected double hourlyRate;

public TemporaryEmployee(String name, String firstName, String dateOfBirth,
String position, double salary,

int contractDuration, double hourlyRate) {
super(name, firstName, dateOfBirth, position, salary);

this.contractDuration = contractDuration;

this.hourlyRate = hourlyRate;

}

// Method specific to the TemporaryEmployee class
public void displayContract() {

System.out.println(this.lastName + " " + this.firstName + " (ID: " +
this.getEmployeeId() + ") born on " + this.dateOfBirth +

" has a temporary contract for a duration of " + this.contractDuration + "
months");

}

public int getContractDuration() {

return this.contractDuration;

}

public double getHourlyRate() {

return this.hourlyRate;
```

```
}
}
package model;

/**

* Simple DAO that stores employees in an array.

*/
public class DAOEmployee {

private Employee[] tableEmployees;

private int capacity;

private int size; // current number of employees

public DAOEmployee (int capacity) {
this.capacity = capacity;

this.tableEmployees = new Employee[capacity];

this.size = 0;

}

// Add an employee (returns true if added, false if full)

public boolean addEmployee(Employee e) {

if (size >= capacity || e == null) {

return false;

}
tableEmployees[size] = e;

size++;

return true;

}

// Retrieve an employee by index
public Employee getEmployee(int index) {
if (index < 0 || index >= size) {
return null;

}
return tableEmployees[index];

}

// Search by last name/first name (returns first match or null)
public Employee searchByLastNameFirstName(String lastName, String
firstName) {
if (lastName == null || firstName == null) return null;

for (int i = 0; i < size; i++) {
Employee e = tableEmployees[i];
```

```
if (e != null && lastName.equalsIgnoreCase(e.getLastName()) &&
firstName.equalsIgnoreCase(e.getFirstName())) {
return e;

}

}
return null;

}

// Delete by ID (optional)
public boolean deleteById(long id) {
for (int i = 0; i < size; i++) {
if (tableEmployees[i] != null && tableEmployees[i].getEmployeeId() == id) {

// Shift left
for (int j = i; j < size - 1; j++) {
tableEmployees[j] = tableEmployees[j + 1];

}
tableEmployees[size - 1] = null;

size--;

return true;

}

}

return false;

}

public Employee[] getTableEmployees() {
return tableEmployees;

}

public int getSize() {
return size;

}
}
```

```
package controller;

import model.ExecutiveEmployee;
import model.Employee;
import model.DAOEmployee;
import model.TemporaryEmployee;

/**

* Controller — coordinates Model (DAO, entities) and View.

* Contains simple business logic and transforms the results for the view.
```

```
*/
public class ControllerEmployee {

private DAOEmployee dao;

public ControllerEmployee(DAOEmployee dao) {
this.dao = dao;

}

public boolean addEmployee(Employee e) {

return dao.addEmployee(e);

}

public boolean increaseSalaryByLastNameFirstName(String lastName, String
firstName, double amount) {
Employee e = dao.searchByLastNameFirstName(lastName, firstName);

if (e == null) return false;

e.increaseSalary(amount);

return true;

}

public Double getAnnualSalaryByLastFirstName(String lastName, String
firstName) {
Employee e = dao.searchByLastNameFirstName(lastName, firstName);

if (e == null) return null;

return e.getAnnualSalary();

}

public Employee[] listAllEmployees() {

return dao.getTableEmployees();

}

public int getNumberOfEmployees() {

return dao.getSize();

}

public String getTemporaryContractInfo(String lastName, String firstName) {
Employee e = dao.searchByLastNameFirstName(lastName, firstName);

if (e instanceof TemporaryEmployee) {
TemporaryEmployee temEmp = (TemporaryEmployee) e;

return temEmp.getLastName() + " " + temEmp.getFirstName() + " - contract "
+ temEmp.getContractDuration() + " month, rate " + temEmp.getHourlyRate() +
"€/h";
```

```
}

return "The employee is not a temporary employee.";

}

public String displayDepartmentExecutiveEmployee(String lastName, String
firstName) {
Employee e = dao.searchByLastNameFirstName(lastName, firstName);

if (e instanceof ExecutiveEmployee) {
      ExecutiveEmployee c = (ExecutiveEmployee) e;

return c.getLastName() + " " + c.getFirstName() + " - department: " +
c.getDepartment();

}

return "The employee is not a ExecutiveEmployee.";

}

public boolean deleteById(long id) {

return dao.deleteById(id);

}
}

package view;

import Controller.ControllerEmployee;
import Model.ExecutiveEmployee;
import Model.Employee;
import Model.DAOEmployee;
import Model.TemporaryEmployee;

/**

* View (console) – displays information and prompts the user for actions.

* Here, we simulate calls (without keyboard input) as in your initial main
function.

*/
public class InterfaceConsole {

public static void main(String[] args) {

// DAO initialization (capacity 10)
DAOEmployee dao = new DAOEmployee(10);

// controller
ControllerEmployee controller = new ControllerEmployee(dao);

// create and add employees via the controller
Employee employee1 = new Employee("HUBERT", "Christophe", "12/03/1988",
"Developer", 4800.0);
controller.addEmployee(employee1);
```

```
Employee employee2 = new Employee("RENARD", "Nathalie", "02/01/1975",
"Manager", 7000.0);

controller.addEmployee(employee2);

TemporaryEmployee TemporaryEmployee = new TemporaryEmployee("DUPUIS",
"Pierre", "22/09/1980", "Assistant", 2500.0,

6, 15.0);

controller.addEmployee(TemporaryEmployee);

ExecutiveEmployee executive = new ExecutiveEmployee("LEBLANC", "Sophie",
"07/04/1968", "Director", 10000.0,

"Human Resources", 1500.0);

controller.addEmployee(executive);

// Displays via the view (uses the controller)

System.out.println("=== List of employees (via Controller) ===");

Employee[] all = controller.listAllEmployees();

int size = controller.getNumberOfEmployees();

for (int i = 0; i < size; i++) {

if (all[i] != null) {

System.out.print((i + 1) + ". ");

all[i].description();

}

}

System.out.println("\n=== IDs and annual salaries ===");

employee1.identifiantDescription();

employee1.description();

controller.increaseSalaryByLastNameFirstName("HUBERT", "Christophe",
500.0);

System.out.println("Annual salary after increase: " +
controller.getAnnualSalaryByLastFirstName("HUBERT", "Christophe") + " €.");

System.out.println();

employee2.identifiantDescription();

employee2.description();

System.out.println("The annual salary of " + employee2.getLastName() + " "
+ employee2.getFirstName() + " is: " + employee2.getAnnualSalary() + "
€.");
```

```
System.out.println();

System.out.println(controller.getTemporaryContractInfo("DUPUIS",
"Pierre"));

System.out.println();

System.out.println(controller.displayDepartmentExecutiveEmployee("LEBLANC",
"Sophie"));

System.out.println("\nDeleting employee ID 2 (example):");

boolean delete = controller.deleteById(2);

System.out.println("Deletion successful? " + delete);

System.out.println("\n=== List after deletion ===");
all = controller.listAllEmployees();

size = controller.getNumberOfEmployees();

for (int i = 0; i < size; i++) {

if (all[i] != null) {
System.out.print((i + 1) + ". ");
all[i].description();

}

}

}
}
```

The execution result is:

```
=== List of employees (via Controller) ===
1. HUBERT Christophe (1) is a Developer, monthly salary: 4800.0 €.
2. RENARD Nathalie (2) is a Manager, monthly salary: 7000.0 €.
3. DUPUIS Pierre (3) is a Assistant, monthly salary: 2500.0 €.
4. LEBLANC Sophie (4) is a Director, monthly salary: 10000.0 €.

=== IDs and annual salaries ===
The employee ID HUBERT Christophe is: 1
HUBERT Christophe (1) is a Developer, monthly salary: 4800.0 €.
Annual salary after increase: 63600.0 €.

The employee ID RENARD Nathalie is: 2
RENARD Nathalie (2) is a Manager, monthly salary: 7000.0 €.
The annual salary of RENARD Nathalie is: 84000.0 €.

DUPUIS Pierre - contract 6 month, rate 15.0€/h

LEBLANC Sophie - department: Human Resources

Deleting employee ID 2 (example):
Deletion successful? true

=== List after deletion ===
```

```
1. HUBERT Christophe (1) is a Developer, monthly salary: 5300.0 €.
2. DUPUIS Pierre (3) is a Assistant, monthly salary: 2500.0 €.
3. LEBLANC Sophie (4) is a Director, monthly salary: 10000.0 €.
```

3.Be sure to use the employee interface you carefully designed in Lab 4. When you click the "Save" button, carefully retrieve the information entered in the various fields, taking into account the employee category (Executive or temporary employee). Next, add a new employee to a table, using the methods provided by the employee management system code, implemented according to the Three-Tier architecture from the previous question. Finally, display all registered employees using a loop after clicking the "Displays" button.

```
package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Basic employee.

*/
public class Employee {

private static long idEmployeCounter = 0; // ID counter

private long employeeID; // Remains private as instructed

protected String lastName;

protected String firstName;

protected String dateOfBirth;

protected String position;

protected double salary;

// Default constructor

      public Employee() {
      this.employeeID = 0;
      this.lastName = null;
      this.firstName = null;
      this.dateOfBirth = null;
      this.position = null;
      this.salary = 0.0;
      }

// Constructor with parameters

      public Employee(String lastname, String firstName, String
dateOfBirth, String position, double salary) {
      idEmployeCounter++;

      employeeID = idEmployeCounter;
      this.lastName = lastname;
```

```
        this.firstName = firstName;
        this.dateOfBirth = dateOfBirth;
        this.position = position;
        this.salary = salary;

        }

// Getter/Setter if needed
public String getLastName() {
return this.lastName;
}

public String getFirstName() {
return this.firstName;

}

        public String getDateOfBirth() {
        return this.dateOfBirth;

        }

// We expose the ID via a method (employeeID is private)`
        public long getEmployeeID() {
        return this.employeeID;

        }

        public void description() {
        System.out.println(this.lastName + " " + this.firstName + " is an
employee, " + this.position +", and he/she has a monthly salary of: " +
this.salary + " €.");
        }

        public void description(String prenom, String nom) {

        System.out.println("The employee ID " + this.lastName + " " +
this.firstName + " is: " + this.getEmployeeID());

        }

        public void increaseSalary(double montant) {
        this.salary += montant;

        }

        public double getAnnualSalary() {
        return this.salary * 12;

        }
}


package data.access;

/**
 *
 * @author USER.COM
 */
/**
```

```
* Temporary Employee (inherits from Employee).

*/
public class TemporaryEmployee extends Employee {

protected int ContractDuration;

protected double HourlyRate;

public TemporaryEmployee(String name, String firstName, String dateOfBirth,
String position, double salary,

int ContractDuration, double HourlyRate) {
super(name, firstName, dateOfBirth, position, salary);

this.ContractDuration = ContractDuration;

this.HourlyRate = HourlyRate;

}

// Method specific to the TemporaryEmployee class

public void displayContract() {

System.out.println(this.lastName + " " + this.firstName + " born on " +
this.dateOfBirth + " has a temporary contract for a duration of " +
this.ContractDuration + " months");

}
}
```

```
package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Manager (inherits from Employee).

*/
public class ExecutiveEmployee extends Employee {

protected String department;

protected double carAllowance;

      public ExecutiveEmployee(String lastName, String firstName, String
dateofbirth, String position, double salary, String department, double
carAllowance) {
      super(lastName, firstName, dateofbirth, position, salary);

      this.department = department;
      this.carAllowance = carAllowance;

      }

// Method specific to the ExecutiveEmployee class
```

```
	public void displayDepartment() {

	System.out.println("The department of " + this.lastName + " " +
this.firstName + " born on " + this.dateOfBirth + " is: " +
this.department);

	}

}
```

```
package data.access;

/**
 *
 * @author USER.COM
 */

/**

* Simple DAO that stores employees in an array.

* Basic operations: add, retrieve by index, search by first/last name,
list.

*/
public class DAOEmployee {

private Employee[] tableEmployees;

private int capacity;

private int size; // current number of employees

	public DAOEmployee(int capacity) {
	this.capacity = capacity;

	this.tableEmployees = new Employee[capacity];

	this.size = 0;

	}

// Add an employee (returns true if added, false if full)

	public boolean addEmployee(Employee e) {

	if (size >= capacity || e == null) {

	return false;

	}
	tableEmployees[size] = e;

	size++;

	return true;

	}
```

```
// Retrieve an employee by index
      public Employee getEmployee(int index) {
      if (index < 0 || index >= size) {
      return null;

      }
      return tableEmployees[index];

      }

// Search by last name/first name (returns first match or null)
      public Employee searchByLastNameFirstName(String lastName, String
firstName) {
      for (int i = 0; i < size; i++) {
      Employee e = tableEmployees[i];

      if (e != null && e.getLastName().equalsIgnoreCase(lastName) &&
e.getFirstName().equalsIgnoreCase(firstName)) {
      return e;

      }

      }
      return null;

      }

// List (returns the internal array and size via getters)
      public Employee[] getTableEmployees() {

      return tableEmployees;

      }

      public int getSize() {

      return size;

      }
}
```

```
package business.logic;

/**
 *
 * @author USER.COM
 */
import data.access.Employee;
import data.access.DAOEmployee;
import data.access.TemporaryEmployee;
import data.access.ExecutiveEmployee;

/**

* Business logic layer — employee management operation.

*/
public class EmployeeManagement {

private DAOEmployee dao;
```

```
	public EmployeeManagement(DAOEmployee dao) {
	this.dao = dao;

	}

// Adds an employee via the DAO

public boolean addEmployee(Employee e) {

return dao.addEmployee(e);

}

// Increases an employee's salary (search by last name/first name)

	public boolean increaseSalary(String lastName, String firstName,
double amount) {
	Employee e = dao.searchByLastNameFirstName(lastName, firstName);

	if (e == null) return false;

	e.increaseSalary(amount);

	return true;

	}

// Get annual salary
	public Double getAnnualSalary(String name, String firstName) {
	Employee e = dao.searchByLastNameFirstName(name, firstName);

	if (e == null) return null;

	return e.getAnnualSalary();

	}

// List all employees
public void listEmployees() {
Employee[] tab = dao.getTableEmployees();

int size = dao.getSize();

for (int i = 0; i < size; i++) {

if (tab[i] != null) {
tab[i].description();
}

}
}
// Examples of methods for specific types

public void displayTemporaryContract(String lastName, String firstName) {
Employee e = dao.searchByLastNameFirstName(lastName, firstName);

if (e instanceof TemporaryEmployee) {

((TemporaryEmployee) e).displayContract();
```

```
} else {

System.out.println("The employee is not a temporary employee.");

}

}

public void displayDepartmentExecutiveEmployee(String lastName, String
firstName) {
Employee e = dao.searchByLastNameFirstName(lastName, firstName);

if (e instanceof ExecutiveEmployee) {
((ExecutiveEmployee) e).displayDepartment();

} else {

System.out.println("The employee is not a manager.");

}

}
}
```

```
package presentation;

import javax.swing.JOptionPane;
import data.access.Employee;
import data.access.TemporaryEmployee;
import data.access.ExecutiveEmployee;
import data.access.DAOEmployee;
import business.logic.EmployeeManagement;

/**
 *
 * @author Mohamed Mohammedi
 */
public class EmployeeInterface extends javax.swing.JFrame {

    /**
     * Creates new form ExecutiveEmployee
     */

    // DAO initialization (capacity 100)
        DAOEmployee dao = new DAOEmployee(100);

        // Business layer
        EmployeeManagement management = new EmployeeManagement(dao);
        Employee[] all = dao.getTableEmployees();

    public EmployeeInterface() {
        initComponents();// Method generated to initialize components

// Check the current selection and disable fields if necessary
    if (jComboBox1.getSelectedItem().toString().trim().equals("Executive
Employee")) {
        jLabel10.setEnabled(false);
        jTextField8.setEnabled(false);
        jLabel11.setEnabled(false);
```

```
            jTextField9.setEnabled(false);
        }
        }

        /**
         * This method is called from within the constructor to initialize the
form.
         * WARNING: Do NOT modify this code. The content of this method is
always
         * regenerated by the Form Editor.
         */
        @SuppressWarnings("unchecked")
        // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
        private void initComponents() {

            jPanel1 = new javax.swing.JPanel();
            jLabel1 = new javax.swing.JLabel();
            jLabel2 = new javax.swing.JLabel();
            jLabel3 = new javax.swing.JLabel();
            jLabel4 = new javax.swing.JLabel();
            jLabel5 = new javax.swing.JLabel();
            jLabel6 = new javax.swing.JLabel();
            jLabel7 = new javax.swing.JLabel();
            jComboBox1 = new javax.swing.JComboBox<>();
            jTextField1 = new javax.swing.JTextField();
            jTextField2 = new javax.swing.JTextField();
            jTextField3 = new javax.swing.JTextField();
            jTextField4 = new javax.swing.JTextField();
            jTextField5 = new javax.swing.JTextField();
            jLabel8 = new javax.swing.JLabel();
            jTextField6 = new javax.swing.JTextField();
            jLabel9 = new javax.swing.JLabel();
            jTextField7 = new javax.swing.JTextField();
            jLabel10 = new javax.swing.JLabel();
            jTextField8 = new javax.swing.JTextField();
            jLabel11 = new javax.swing.JLabel();
            jTextField9 = new javax.swing.JTextField();
            jButton1 = new javax.swing.JButton();
            jButton2 = new javax.swing.JButton();
            jLabel13 = new javax.swing.JLabel();
            jButton3 = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
            setTitle("Adding an Employee");

            jPanel1.setBackground(new java.awt.Color(255, 204, 255));
            jPanel1.setAlignmentX(0.1F);
            jPanel1.setAlignmentY(0.1F);
            jPanel1.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));

            jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 24)); // NOI18N
            jLabel1.setText("Adding an Employee");

            jLabel2.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
            jLabel2.setText("Employee type:");

            jLabel3.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
```

```
        jLabel3.setText("First Name:");

        jLabel4.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel4.setText("Last Name:");

        jLabel5.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel5.setText("Date of birth:");

        jLabel6.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel6.setText("Position:");

        jLabel7.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel7.setText("Salary:");

        jComboBox1.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Executive Employee", "Temporary Employee" }));
        jComboBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jComboBox1ActionPerformed(evt);
            }
        });

        jTextField1.setText("Michael");

        jTextField2.setText("Wilson");

        jTextField3.setText("07/04/1968");

        jTextField4.setText("Director");

        jTextField5.setText("10000.00");

        jLabel8.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel8.setText("Department:");

        jTextField6.setText("Human Resources");

        jLabel9.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel9.setText("Car allowance:");

        jTextField7.setText("1500.00");

        jLabel10.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel10.setText("Contract duration:");

        jTextField8.setText("Contract duration");

        jLabel11.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel11.setText("Hourly rate:");

        jTextField9.setText("Hourly rate");

        jButton1.setBackground(new java.awt.Color(255, 0, 153));
        jButton1.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton1.setText("Cancel");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });
```

```
        jButton2.setBackground(new java.awt.Color(0, 255, 204));
        jButton2.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton2.setText("Save");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jLabel13.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/Presentation/businessman_add
(2).png"))); // NOI18N

        jButton3.setBackground(new java.awt.Color(255, 204, 51));
        jButton3.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton3.setText("Displays");
        jButton3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton3ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(70, 70, 70)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(422, 422, 422)
                        .addComponent(jLabel13)
                        .addGap(0, 19, Short.MAX_VALUE))
                    .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)

.addGroup(jPanel1Layout.createSequentialGroup()
                                    .addComponent(jButton1)
                                    .addGap(61, 61, 61)
                                    .addComponent(jButton2)
                                    .addGap(61, 61, 61)
                                    .addComponent(jButton3))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                    .addComponent(jLabel8)

.addGroup(jPanel1Layout.createSequentialGroup()
```

```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                            .addComponent(jLabel9)
                                            .addComponent(jLabel7)
                                            .addComponent(jLabel6)
                                            .addComponent(jLabel5)
                                            .addComponent(jLabel4)
                                            .addComponent(jLabel3)
                                            .addComponent(jLabel10)
                                            .addComponent(jLabel11))
                                        .addGap(134, 134, 134)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                            .addComponent(jTextField9,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField8,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField5,
javax.swing.GroupLayout.PREFERRED_SIZE, 85,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField4,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE, 85,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))))
                                .addComponent(jLabel2))

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(205, 205, 205)
                .addComponent(jLabel1)
                .addGap(0, 0, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
```

```
                .addContainerGap()
                .addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                    .addComponent(jLabel13)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(0, 26, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                            .addComponent(jLabel2,
javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(jComboBox1,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                            .addComponent(jTextField1,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel3,
javax.swing.GroupLayout.Alignment.TRAILING))
                        .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                            .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel4))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                            .addComponent(jLabel5)
                            .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addGap(7, 7, 7)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField4,
javax.swing.GroupLayout.PREFERRED_SIZE,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel6))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField5,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7))
                .addGap(26, 26, 26)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jLabel8)
                    .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel9))
                .addGap(29, 29, 29)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField8,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel10))
                .addGap(26, 26, 26)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jLabel11)
                    .addComponent(jTextField9,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(47, 47, 47)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jButton1)
                    .addComponent(jButton2)
                    .addComponent(jButton3))
                .addGap(45, 45, 45))
        );

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
```

```
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(29, Short.MAX_VALUE)
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton2ActionPerformed
    // TODO add your handling code here:
    // Display a confirmation message
        String Rec = jComboBox1.getSelectedItem().toString().trim();
        String lastName = jTextField1.getText().trim();
        String firstName = jTextField2.getText().trim();
        String dateOfbirth = jTextField3.getText().trim();
        String position = jTextField4.getText().trim();

        try {
        double salary = Double.parseDouble(jTextField5.getText().trim());

        if (Rec.equals("Executive Employee")) {
        String department = jTextField6.getText().trim();
        double carAllowance =
Double.parseDouble(jTextField7.getText().trim());

        // Check field validity
        if (isValidExecutiveEmployee(lastName, firstName, dateOfbirth,
position, salary, department, carAllowance)) {

        // Create and add employees
        ExecutiveEmployee executiveEmpl = new ExecutiveEmployee(lastName,
firstName, dateOfbirth, position, salary, department, carAllowance);
        management.addEmployee(executiveEmpl);
        JOptionPane.showMessageDialog(this, "The information for
ExecutiveEmployee " + lastName + " " + firstName + " has been successfully
added!");

        // Displays via the presentation layer (uses the business layer)
        System.out.println("=== Cadre Employee ===");
```

```
        System.out.println("\n=== Cadre ID and annual salary "+lastName + "
" + firstName + "===");
        executiveEmpl.description(lastName, firstName);
        executiveEmpl.description();
        management.displayDepartmentExecutiveEmployee(lastName, firstName);
        } else {
        JOptionPane.showMessageDialog(this, "Failed to add
ExecutiveEmployee information. Please check the information.", "Error",
JOptionPane.ERROR_MESSAGE);
        }

        } else {
        int contractDuration =
Integer.parseInt(jTextField8.getText().trim());

        double hourlyRate =
Double.parseDouble(jTextField9.getText().trim());

        // Check field validity
        if (isValidTemporaryEmployee(lastName, firstName, dateOfbirth,
position, salary, contractDuration, hourlyRate)) {
        TemporaryEmployee temporaryEmployee = new
TemporaryEmployee(lastName, firstName, dateOfbirth, position, salary,
contractDuration, hourlyRate);
        management.addEmployee(temporaryEmployee);

        JOptionPane.showMessageDialog(this, "The information for Temporary
Employee " + lastName + " " + firstName + " has been successfully added!");

        // Displays via the presentation layer (uses the business layer)
        System.out.println("=== Temporary Employee ===");

        System.out.println("\n=== Temporary Employee IDs and Annual
Salaries " + lastName + " " + firstName + "===");
        temporaryEmployee.description(lastName, firstName);
        temporaryEmployee.description();
        management.displayTemporaryContract(lastName, firstName);
        } else {
        JOptionPane.showMessageDialog(this, "Failed to add temporary
employee information. Please check the information.", "Error",
JOptionPane.ERROR_MESSAGE);
        }
        }
        } catch (NumberFormatException e) {
        // Handle the conversion error if the salary or other numeric field
is not a valid number
        JOptionPane.showMessageDialog(this, "Please enter valid numeric
values.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }//GEN-LAST:event_jButton2ActionPerformed

    // Validation methods for the ExecutiveEmployee
    private boolean isValidExecutiveEmployee(String lastName, String
firstName, String dateOfbirth, String position, double salary, String
department, double carAllowance) {
    return !lastName.isEmpty() && !firstName.isEmpty() &&
!dateOfbirth.isEmpty() && !position.isEmpty() && salary >= 0 &&
!department.isEmpty() && carAllowance >= 0;
    }
```

```
    // Validation methods for the temporary employee
    private boolean isValidTemporaryEmployee(String lastName, String
firstName, String dateOfBirth, String position, double salary, int
contractDuration, double hourlyRate) {
    return !lastName.isEmpty() && !firstName.isEmpty() &&
!dateOfBirth.isEmpty() && !position.isEmpty() &&  salary >= 0 &&
contractDuration >= 0 && hourlyRate >= 0;

    }

    private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jComboBox1ActionPerformed
        // TODO add your handling code here:
        //Always activate the fields
    jLabel10.setEnabled(true);
    jTextField8.setEnabled(true);
    jLabel11.setEnabled(true);
    jTextField9.setEnabled(true);

    jLabel8.setEnabled(true);
    jTextField6.setEnabled(true);
    jLabel9.setEnabled(true);
    jTextField7.setEnabled(true);

    String Rec = jComboBox1.getSelectedItem().toString().trim();

    // Disable fields based on selection
    if (Rec.equals("Executive Employee")) {
        jLabel10.setEnabled(false);
        jTextField8.setEnabled(false);
        jLabel11.setEnabled(false);
        jTextField9.setEnabled(false);
    } else {
        jLabel8.setEnabled(false);
        jTextField6.setEnabled(false);
        jLabel9.setEnabled(false);
        jTextField7.setEnabled(false);
    }
    }//GEN-LAST:event_jComboBox1ActionPerformed

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
        // TODO add your handling code here:
        //Reset the fields
       jTextField1.setText("");
       jTextField2.setText("");
       jTextField3.setText("");
       jTextField4.setText("");
       jTextField5.setText("");
       jTextField6.setText("");
       jTextField7.setText("");
       jTextField8.setText("");
       jTextField9.setText("");
    }//GEN-LAST:event_jButton1ActionPerformed

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton3ActionPerformed
        // TODO add your handling code here:
        // Display all registered employees using a loop
```

```
        System.out.println("=== Displaying registered employees (loop)
===");

        int size = dao.getSize();
        for (int i = 0; i < size; i++) {
        if (all[i] != null) {
        System.out.print((i + 1) + ". ");
        all[i].description();
        }
        }
    }//GEN-LAST:event_jButton3ActionPerformed

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel
setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with
the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new EmployeeInterface().setVisible(true);
            }
        });
    }
```

```
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JComboBox<String> jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    private javax.swing.JTextField jTextField6;
    private javax.swing.JTextField jTextField7;
    private javax.swing.JTextField jTextField8;
    private javax.swing.JTextField jTextField9;
    // End of variables declaration//GEN-END:variables
}
```

The execution result is:

```
=== Cadre Employee ===

=== Cadre ID and annual salary Michael Wilson===
The employee ID Michael Wilson is: 1
Michael Wilson is an employee, Director, and he/she has a monthly salary of:
10000.0 $.
The department of Michael Wilson born on 07/04/1968 is: Human Resources
=== Temporary Employee ===

=== Temporary Employee IDs and Annual Salaries Michael Wilson===
The employee ID Michael Wilson is: 2
Michael Wilson is an employee, Director, and he/she has a monthly salary of:
10000.0 $.
The employee is not a temporary employee.
=== Displaying registered employees (loop) ===
1. Michael Wilson is an employee, Director, and he/she has a monthly salary
of: 10000.0 $.
2. Michael Wilson is an employee, Director, and he/she has a monthly salary
of: 10000.0 $.
```

Lab 6: Transition from the domain model to the relational model

Outline

Goals

- Understand the transformation of a UML model into a relational model,
- Apply the rules for converting class diagrams into database relationships,
- Analyze practical cases to illustrate theoretical concepts.

6.1. Introduction

In this chapter, we will discuss the transition from the UML modeling language to the relational model, essential in software engineering for database development. UML, or Unified Modeling Language, provides a graphical representation of systems, while the relational model describes the structure and relationships of data in a database management system.

The main objective is to learn how to analyze a domain model from a UML class diagram (domain model) and convert it into a relational model. This conversion is crucial for the design of efficient databases, as it allows data to be structured in a way that ensures its integrity and accessibility.

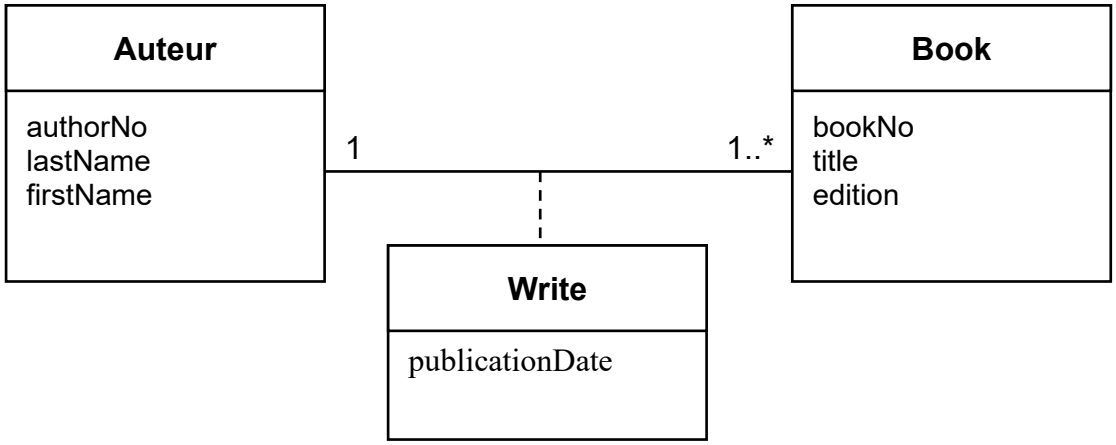
We will examine the rules for transforming classes, attributes, and associations into relationships, as well as the different types of multiplicity that influence this transformation. Through practical exercises, you will have the opportunity to apply these concepts and better understand the implications of each rule on data structure.

6.2. Domain model

In software engineering, a domain model is the conceptual model of a domain. In UML, a class diagram is used to represent the domain model.

Rule 1: Presence of the multiplicity ''*'' on one side of the association

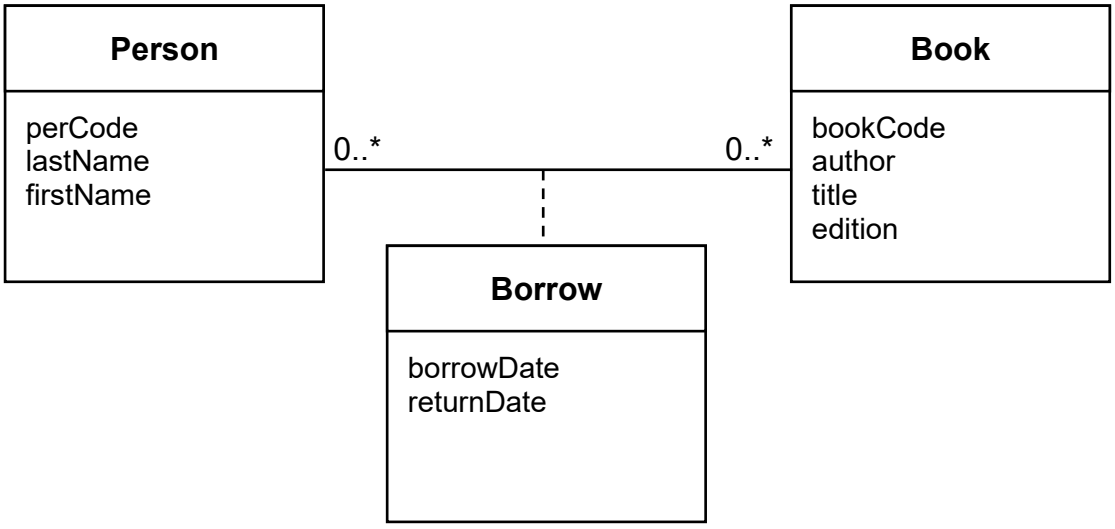
- Each class becomes a relationship.
- Each class attribute becomes a relationship field.
- The class ID associated with the multiplicity ''1'' becomes a foreign key in the other class.



Author (authorNo., lastName, firstName)
Book (bookNo., title, edition, authorNo., publicationDate)

Rule 2: Presence of the multiplicity ‘’*’’ on both sides

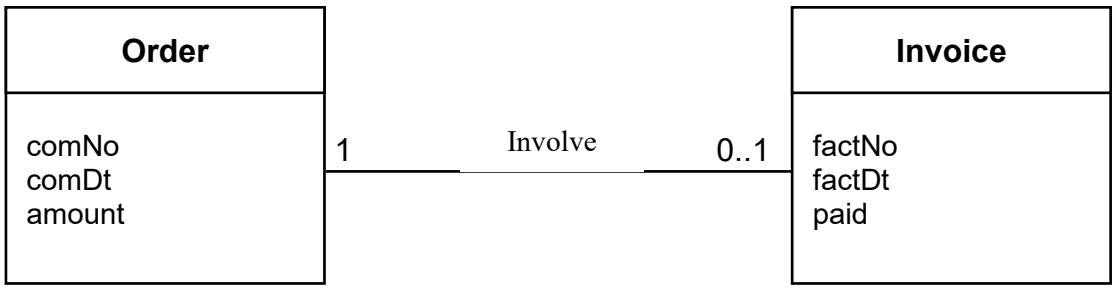
- Each class becomes a relationship.
- Each class attribute becomes a relationship field.
- The association becomes a relationship whose fields will contain the identifier of each of the two classes (plus any other attributes).



Person (PerCode, lastName, firstName)
Book (bookCode, author, title, edition)
Borrow (#PerCode, #bookCode, borrowDate, returnDate)

Rule 3: Presence of the multiplicity ‘’1..1’’ ‘’0..1’’ on the sides

- Each class becomes a relationship.
- Each class attribute becomes a relationship field.
- The class ID associated with the multiplicity ‘’1..1’’ becomes a foreign key in the other class (the one associated with the multiplicity ‘’0..1’’).



Order (comNo, comDt, amount)
Invoice (factNo, #comNo, factDt, paid)

Rule 4: Presence of a composition relationship

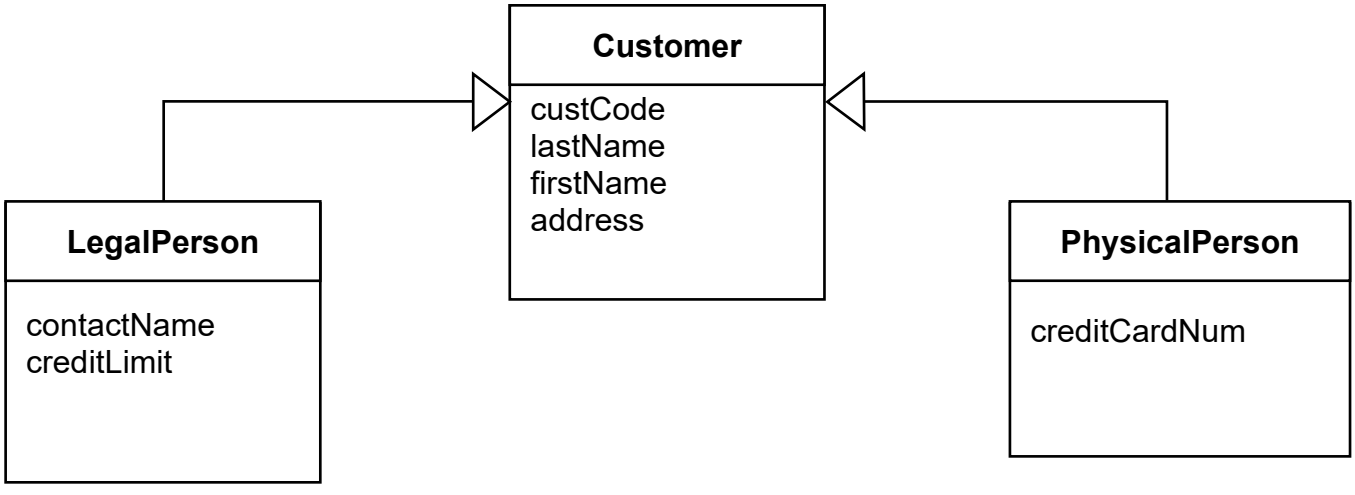
- Each class becomes a relationship.
- Each class attribute becomes a relationship field.
- The identifier of the class associated with the multiplicity ''1'' (the class representing the set level) becomes both a foreign key and part of the primary key in the other class.

Group (group_id, group_name)
Student (student_id, #group_id, lastName, firstName, date_of_birth, address)

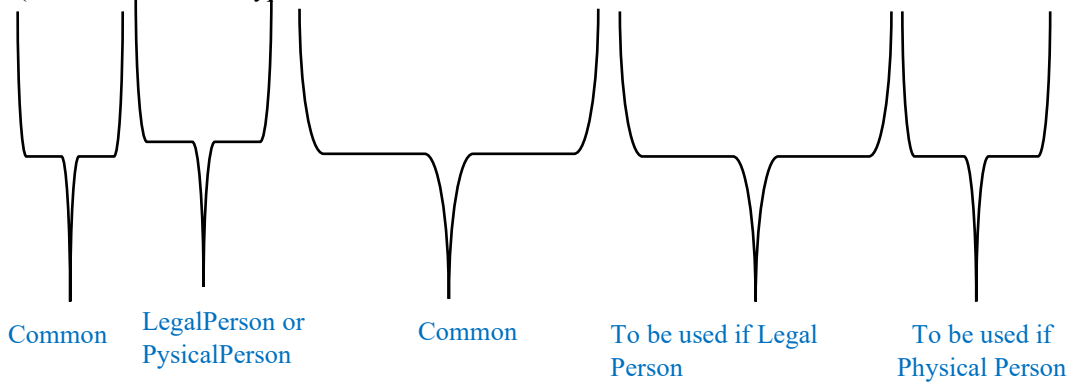
Rule 5: Presence of a generalization-specialization

Method 1: Push-up

- Create a relationship with all class attributes.
- Add an attribute to distinguish object types.

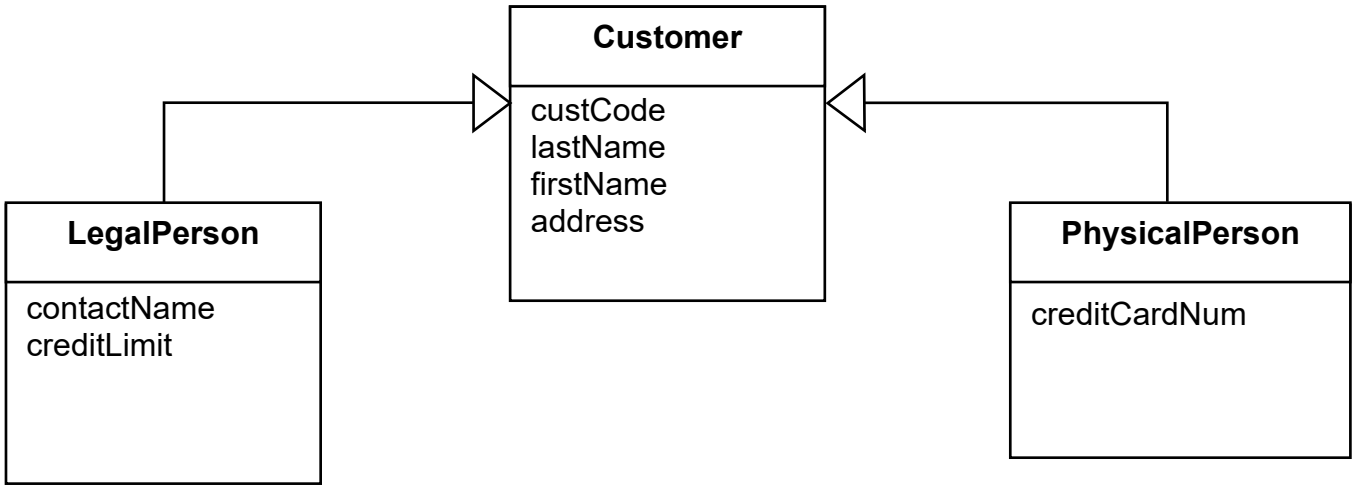


Customer (custCode, customerType, lastName, firsName, address, contactName, creditLimit, creditCardNum)



Method 2: Push-down

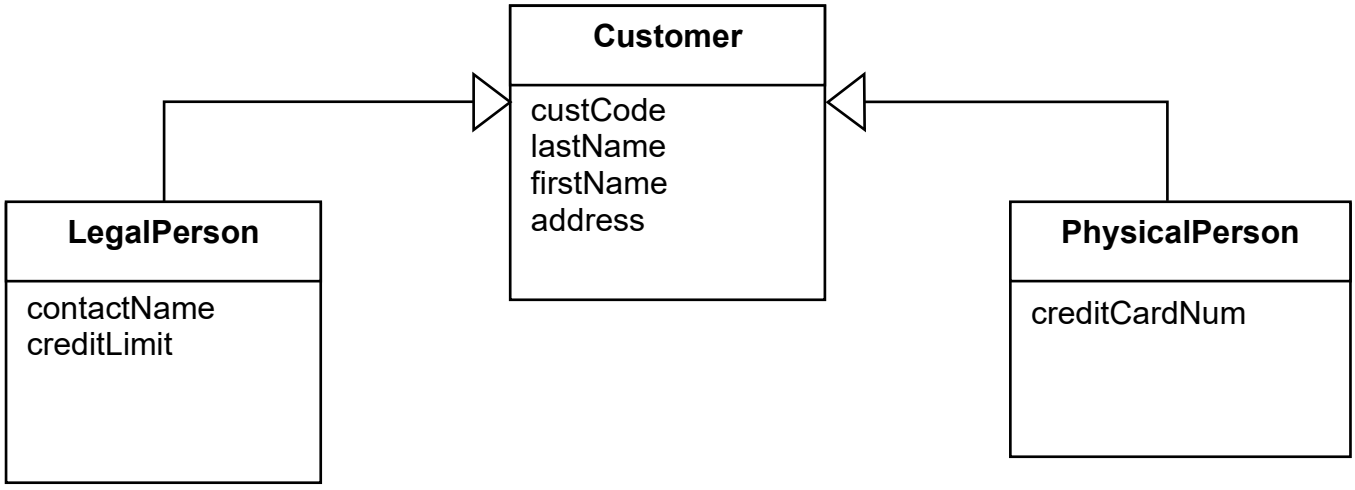
- Create a relationship for each subtype.
- Each relationship consists of generic and specific attributes.



LegalPerson (custCode, lastName, firstName, address, contactName, creditLimit)
PhysicalPerson (custCode, lastName, firstName, address, creditCardNum)

Method 3: Distinction

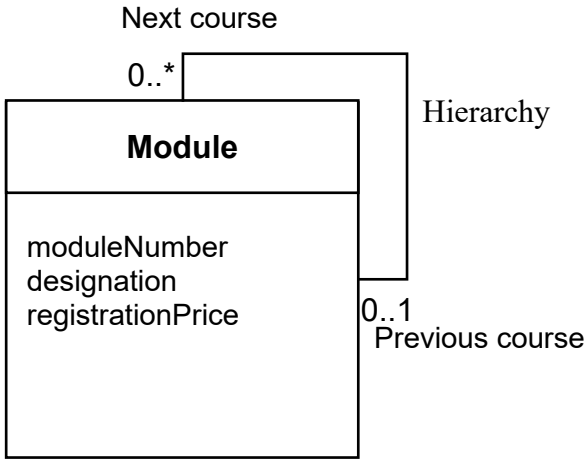
- Create a relationship per class and connect them using associations.



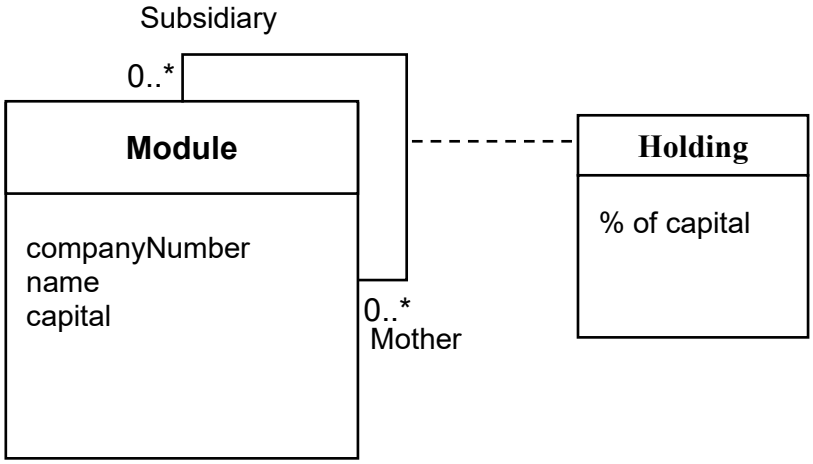
Customer (CustomerCode, lastName, firstName, Address)
LegalPerson (#CustomerCode, contactName, CreditLimit)
PhysicalPerson (#CustomerCode, creditCardNum)

Rule 6: Presence of a reflexive relationship

We apply the general rules, with the sole difference that the relationship is linked twice to the same entity.



Module (module_number, designation, registration price, #module_number_Previous course)



Company (companyNumber, name, capital)

Holding (#company_number, #subsidiary_company_number, % of capital)

6.3. Exercises

Exercise 1:

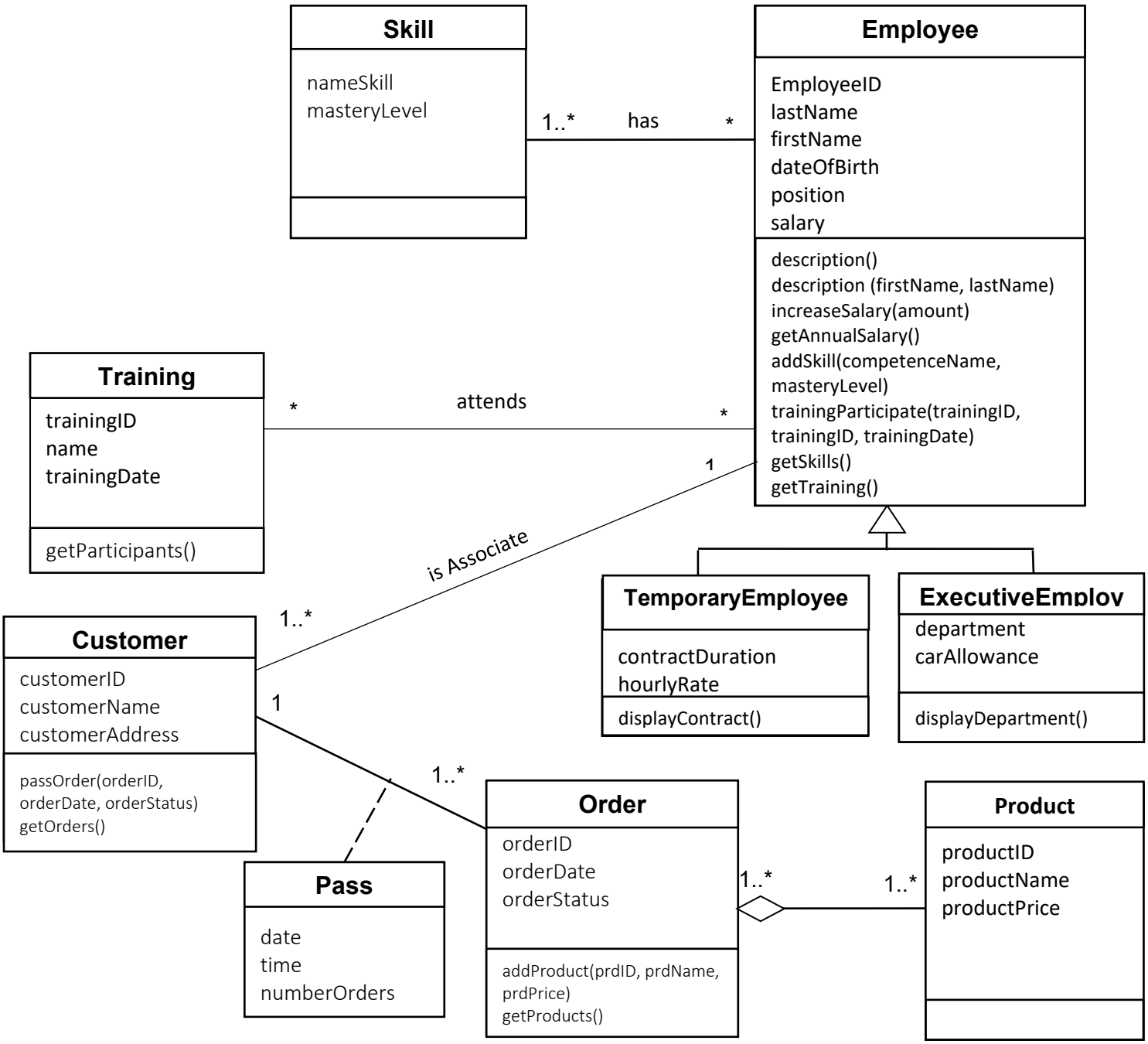
Firstpack wants to expand its employee management system by adding employee skills management. Each employee has a set of skills, represented by a skill name (SkillName) and a proficiency level (ProficiencyLevel) ranging from 1 to 5. An employee can have multiple skills, and a skill can be possessed by multiple employees. Additionally, the company wants to track the training courses employees have attended. Each training course is characterized by a

training ID (TrainingID), a training name (TrainingName), and a training date (TrainingDate). An employee can attend multiple training courses, and a training course can have multiple participants. In addition, the company wants to manage customers and their orders. Each customer is characterized by a customer ID (CustomerID), a customer name (CustomerName), and an address (CustomerAddress).

Each employee can be associated[1] with multiple customers, while a customer is associated with a single employee. A customer can place multiple orders, and an order is associated with a single customer. Each order is characterized by an order identifier (OrderID), an order date (OrderDate), and a status (OrderStatus). We want to be able to know at any time the dates, times, and number of orders placed by a customer. An order can contain multiple products, and a product can appear in multiple orders. Each product is characterized by a product identifier (ProductID), a product name (ProductName), and a price (ProductPrice).

[1] This association will allow you to reflect a specific employee's responsibility for customer and order management. For example, an employee can be designated as responsible for orders for a specific group of customers. This way, the employee management system will be able to access customer and order information related to that specific employee. Furthermore, this relationship will also facilitate access to the information needed to meet the specific request mentioned in the second part of the exercise: knowing the dates, times, and number of orders placed by a customer at any time.

From the information presented above, we will obtain a class diagram that will complement the one in Lab 3 as follows:

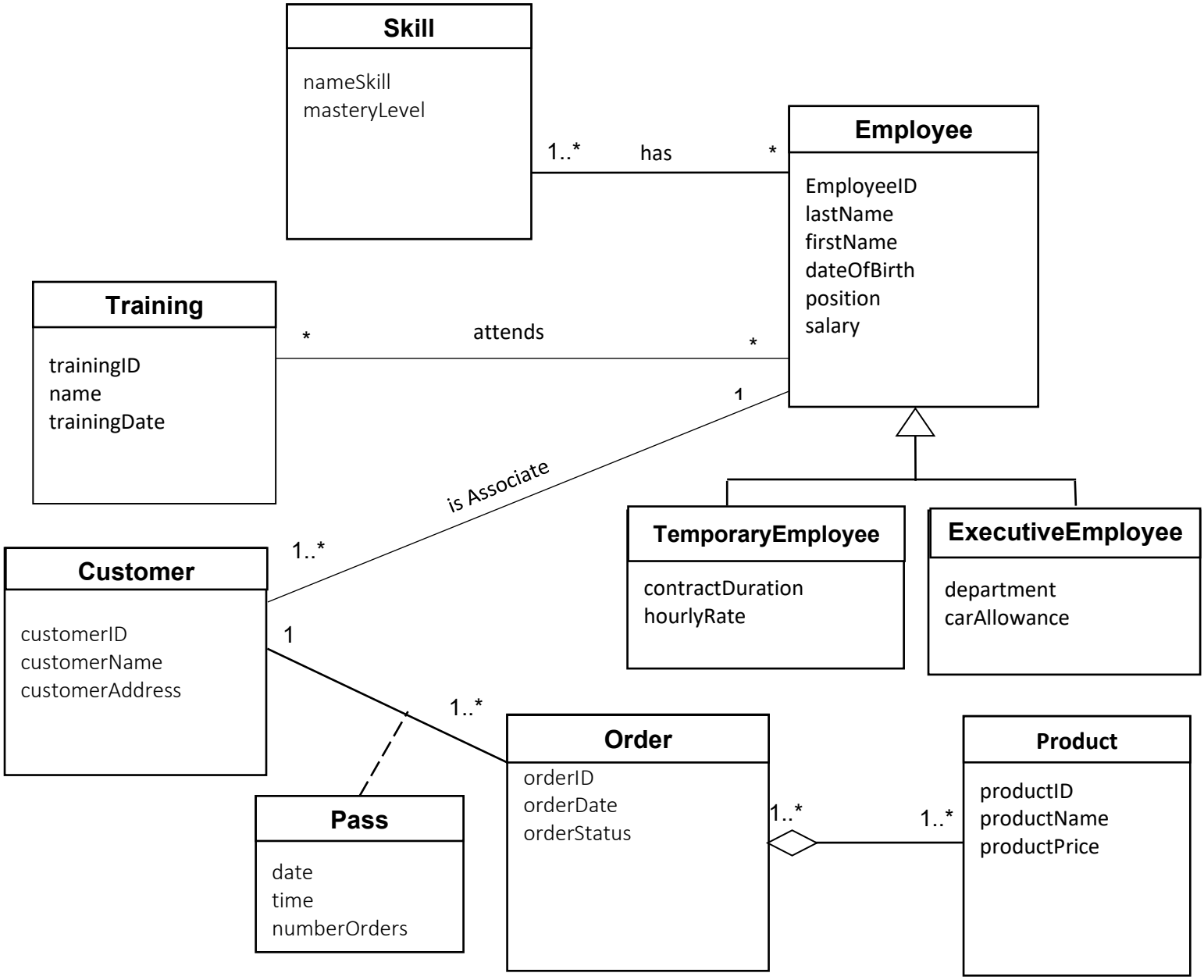


Note:

In order to implement a database, it is essential to translate the class diagram into a domain model. Then, this domain model must be transformed into a **relational model**[2].

[2] It is necessary for the student to read the appendix (entitled "**Transition from the domain model to the relational model**") which is associated with this series of practical exercises.

Consider the domain model shown below, which is derived from the previously mentioned class diagram:



1. Applying the appropriate transition rules, create the relational model corresponding to the domain model presented above. Use the ''push-up'' method when a generalized relationship exists.

2. Implement the above-mentioned domain model in the MySQL database management system and generate the SQL code corresponding to the database creation.

6.4. Exercise corrections

Answers to exercise 1:

The goal of implementing a domain model is to create a database that stores all the information from the employee management application. Based on the conceptual description provided by the developers, a relational model can be created using the rules for transitioning from UML to the relational model. Since the information system cannot manipulate it directly, these rules must be followed. Once the relational model is established, it must be implemented in a database management system (DBMS). By following the rules for transitioning from a domain model to a relational model, we obtain relationships with primary keys, simple fields, and foreign keys. When there is a generalization-specialization relationship (as in our case with the Employee, Executive Employee, and Temporary Employee classes), there are three possible methods for transitioning to the relational model: the Push Up method, the Push Down method, and the Distinction method. All three methods are valid.

Push UP

Employee (employeeID, employeeType, lastName, firstName, dateOfBirth, position, salary, contractDuration, hourlyRate, department, carAllowance)

Skill (skillName, masteryLevel)
Has(#employeeID, #skillName)

Training (trainingID, trainingName, trainingDate)

Attends(#employeeID, #trainingID)

Customer (customerID, customerName, customerAddress, #employeeID)

Order (orderID, orderDate, orderStatus, #customerID, date, time, numberOrders)

Product (productID, productName, productPrice)

Pass(#OrderID, #ProductID)

2. Implementation of the above-mentioned domain model in the MySQL database management system and generation of SQL code corresponding to the creation of the database:

```
CREATE DATABASE EmployeeManagement;
USE EmployeeManagement;

CREATE TABLE Employee (
employeeID INT AUTO_INCREMENT PRIMARY KEY,
employeeType VARCHAR(50),
```

```
lastName VARCHAR(100),
firstName VARCHAR(100),
birthDate DATE,
position VARCHAR(100),
salary DECIMAL(10, 2),
contractDuration INT NULL,
hourlyRate DECIMAL(10, 2) NULL,
department VARCHAR(100) NULL,
carAllowance DECIMAL(10, 2) NULL
);

CREATE TABLE Skill (
skillName VARCHAR(100) PRIMARY KEY,
masteryLevel VARCHAR(50)
);

CREATE TABLE Has (
 employeeID INT,
 skillName VARCHAR(100),
 PRIMARY KEY (employeeID, skillName),
 FOREIGN KEY (employeID) REFERENCES Employee(employeID),
 FOREIGN KEY (skillName) REFERENCES Skill(skillName)
);

CREATE TABLE Training (
 formationID INT AUTO_INCREMENT PRIMARY KEY,
 trainingName VARCHAR(100),
 dateDATE training
);

CREATE TABLE Attends (
 EmployeeID INT,
 trainingID INT,
 PRIMARY KEY (employeeId, trainingId),
 FOREIGN KEY (employeID) REFERENCES Employee(employeID),
 FOREIGN KEY (trainingID) REFERENCES Training (trainingID)
);

CREATE TABLE Customer (
 customerID INT AUTO_INCREMENT PRIMARY KEY,
 customerName VARCHAR(100),
```

```
 customerAddress VARCHAR(200),
 employeeID INT,
 FOREIGN KEY (employeID) REFERENCES Employee(employeID)
);

CREATE TABLE Order (
 orderID INT AUTO_INCREMENT PRIMARY KEY,
 orderDate DATE,
 orderStatus VARCHAR(50),
 customerID INT,
 date DATE,
 time TIME,
 numberOrders INT,
 FOREIGN KEY (customerID) REFERENCES Client (customerID)
);

CREATE TABLE Product (
 productID INT AUTO_INCREMENT PRIMARY KEY,
 productName VARCHAR(100),
 priceProduct DECIMAL(10, 2)
);

CREATE TABLE Pass (
orderID INT,
productID INT,
PRIMARY KEY (orderID, productID),
FOREIGN KEY (orderID) REFERENCES Order(orderID),
FOREIGN KEY (productID) REFERENCES Product(productID)
);
```

Execution Instructions

1. Open MySQL: Connect to your MySQL server.
2. Create the Database: Run the above SQL script in your database management environment (such as WAMP Server, XAMPP, MySQL Workbench, and others).
3. Verification: Verify that all tables have been created with the correct relationships and data types.

Lab 7 : Connecting to a MySQL database in Java with NetBeans.

Outline

Goals

- Understand the importance of project configuration,
- Configure the development environment (Ant and Maven),
- Establish a connection to the MySQL database,
- Perform database operations,
- Reinforce good programming practices.

7.1. Introduction

Connecting a Java project to a MySQL database is an essential step in developing robust and dynamic applications. In today's information technology landscape, data management is central to many applications, whether for management systems, web applications, or enterprise services.

MySQL, as an open-source relational database management system (RDBMS), is widely used for its reliability, performance, and ease of integration. Thanks to its Java Database Connectivity (JDBC) connector, Java allows developers to interact with MySQL databases smoothly and efficiently. This connection opens the door to a variety of operations, such as inserting, updating, deleting, and retrieving data.

This lab aims to provide a clear understanding of the steps required to set up your development environment, whether you are using Ant or Maven as your project management tool. We'll also cover creating a Java class dedicated to connection management, as well as practical examples illustrating how to perform database operations.

By following the steps outlined, you'll be able to configure your project to communicate effectively with a MySQL database, laying the foundation for developing more complex applications.

7.2 Step 1: Project Setup

Before connecting a Java project to a MySQL database, it is essential to configure your environment according to the project management tool you are using.

A. Configuration for Ant

a. Download the MySQL JDBC connector:
   - Go to the official MySQL website and download the JDBC connector.
   - Extract the **.zip** file to access the JAR file.

b. Add the JAR file to the project in NetBeans:
   - Right-click on your project in NetBeans and select **Properties**.
   - Go to the ''**Libraries**'' tab.
   - Click on ''**Manage platforms**'', then go to the ''**Sources**'' tab.
   - Click Add JAR/Folder, then select the extracted ''**mysql-connector-java.jar**'' file.
   - Confirm by clicking on ''**Close**'', then on ''**OK**''.

   Otherwise, if that doesn't work, go to **compile**, then **classpath**, then **+** and add ''**mysql-connector-java.jar**''

B. Configuration for Maven

1. Add the MySQL JDBC dependency to the pom.xml file:
   - Open the pom.xml file in your Maven project (available under Project Files).
   - Add the following dependency to the <dependencies> section:

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.33</version>
</dependency>
```

   - Save the pom.xml file. Maven will automatically download the library.

7.3 Step 2: Connect to a MySQL database

a. Create the Java **DatabaseConnection** class:
   - Create a class named **DatabaseConnection** in your project.
   - Add the following code to manage the connection:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
```

```
private static final String URL = "jdbc:mysql://localhost:3306/tp"; //
Replace 'tp' with your database name
private static final String USER = "root"; // Replace with your
username
private static final String PASSWORD = ""; // Replace with your
password

public static Connection getConnection() throws SQLException {
return DriverManager.getConnection(URL, USER, PASSWORD);
}

public static void main(String[] args) {
// Test the connection
try (Connection connection = getConnection()) {
System.out.println("Connection successful!");
} catch (SQLException e) {
System.err.println("Connection error: " + e.getMessage());
}
}
}
```

b. Using the DatabaseConnection class:

Once the DatabaseConnection class is created, you can use it in your model classes to perform database operations (such as inserting, reading, updating, or deleting data).

7.4 Practical example

Example of addEmployee method in a model to insert data into the database:

```
//Add an employee
public boolean addEmployee(String lastname, String firstname, String
position, double salary) {
String sql = "INSERT INTO employees (lastname, firstname, position,
salary) VALUES (?, ?, ?, ?)";
try (Connection connection = DatabaseConnection.getConnection();
PreparedStatement statement = connection.prepareStatement(sql)) {
statement.setString(1, lastname);
statement.setString(2, firstname);
statement.setString(3, position);
statement.setDouble(4, salary);
return statement.executeUpdate() > 0; // Returns true if the insert
is successful
} catch (SQLException e) {
System.err.println("Error adding employee: " + e.getMessage());
return false;
}
}
```

7.5. Exercises

Exercise 1:

In this lab, explore the steps required to establish a connection between your Java project and the MySQL database, as well as to execute SQL queries:

a. Add the necessary libraries to establish the connection between the project created in Lab 3 and the database created in Lab 6.

b. Execute SQL queries (insert, update, etc.) from your Java program and verify that the updates are correctly performed in your database.

Exercise 2:

1. Ensure you use the "employee" interface you designed in Lab 4. When you click the "Save" button, retrieve the data entered in the various fields, taking into account the employee category (Executive or temporary employee). Then, add a new employee to the employee table in your database, which you have already created with WAMP using the methods provided by the employee management system code (implemented according to the three-tier architecture, as in the previous question of Lab 5). Finally, after clicking the "Views" button, open a separate window (or view) dedicated to displaying employees—for example, a new window, a separate tab, or a non-blocking modal view—and display all the employees present in the ''employee'' table. This display view must be independent of the "employee" form (no embedded list below the form).

2. Implement a deletion feature: when the "Delete" button is clicked, display a dialog box asking for the ID of the employee to be deleted, request confirmation, and then delete the corresponding record in the database.

7.6. Exercise Corrections

Answers to exercise 1:

To establish a connection between your Java project and the MySQL database, here are the steps to follow:

Step 1: Add the MySQL Connector/J library

1. Download the JDBC Driver for MySQL:
   - Go to the [MySQL Connector/J](https://dev.mysql.com/downloads/connector/j/) website and download the JAR file corresponding to your MySQL version.

2. Add the library to the project in NetBeans:
   - Right-click the project in the NetBeans tree.
   - Select "Properties".
   - Go to "Libraries" and click "Add JAR/Folder".
   - Navigate to the location of the downloaded JAR file and select it.

Step 2: Establish the Database Connection

Create a class to manage the database connection. Here is an example of a "**DatabaseConnection**" class:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
public class DatabaseConnection {
private static final String URL =
"jdbc:mysql://localhost:3306/EmployeeManagement";// Replace
'EmployeeManagement' with the name of your database
private static final String USER = "your_user"; // Replace with your user
private static final String PASSWORD = "your_password"; // Replace with
your password

public static Connection getConnection() {
Connection connection = null;
try {
connection = DriverManager.getConnection(URL, USER, PASSWORD);
System.out.println("Successful connection to the database.");
} catch (SQLException e) {
System.out.println("Connection error: " + e.getMessage());
}
return connection;
}
}
```

Step 3: Execute SQL Queries

Create a class to handle database operations, such as inserting and updating. Here's an example:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class EmployeDAO {
 public void addEmployee(Employee employee) {
 String sql = "INSERT INTO Employee (employeeType, lastName, firstName,
dateOfBirth, position, salary) VALUES (?, ?, ?, ?, ?, ?)";

 try (Connection connection = DatabaseConnection.getConnection();
 PreparedStatement statement = connection.prepareStatement(sql)) {
 statement.setString(1, employee.getEmployeType());
 statement.setString(2, employee.getlastName());
 statement.setString(3, employee.getfirstName());
 statement.setDate(4, java.sql.Date.valueOf(employee.getBirthDate()));
 statement.setString(5, employee.getPosition());
statement.setDouble(6, employee.getSalary());

int rowsInserted = statement.executeUpdate();
if (rowsInserted > 0) {
System.out.println("A new employee has been successfully inserted!");
}
} catch (SQLException e) {
System.out.println("Error inserting: " + e.getMessage());
}
}

// Methods for updating, deleting, etc.
}
```

Step 4: Test Employee Insertion

Modify the main class to test the insertion of an employee:

```
public class Main {
public static void main(String[] args) {
```

```
Employee newEmployee = new Employee("EmployeeExecutive", "Smith", "John",
"1985-06-15", "Manager", 3500.00);

EmployeeDAO employeeDAO = new EmployeeDAO();
employeeDAO.addEmployee(newEmployee);
}
}
```

Step 5: Check for Updates in the Database

1. Run the program in NetBeans.
2. Check the database with a tool like WAMP Server, XAMPP, MySQL Workbench, and others to see if the employee has been added to the "Employee" table.

What is WAMP Server?

WAMP Server is a web development environment for Windows that allows you to create and manage web applications. Its name is an acronym that stands for:

- Windows
- Apache (the web server)
- MySQL (the database management system)
- PHP (the programming language)

WAMP Server combines all these components into a single application to facilitate local web application development. With this tool, you can test your projects before deploying them to an online server, which is especially useful for developers and students.

Creating a database with WAMP Server is as follows:

Step 1: Start WAMP

1. Launch WAMP: Double-click the WAMP icon on your desktop. Make sure the icon turns green, indicating that the server is running.

Step 2: Access phpMyAdmin

1. Open your browser (Chrome, Firefox, etc.).
2. Access phpMyAdmin by entering the following URL: "http://localhost/phpmyadmin".

Step 3: Log in to phpMyAdmin

1. Enter your credentials. By default, the username is "root" and the password is blank (leave the password field empty).
2. Click "Run" or press Enter.

Step 4: Create a new database

1. In phpMyAdmin, click the "Databases" tab at the top.
2. In the "Create a database" field, enter the name of your new database.
3. Click the "Create" button.

Step 5: Verify the creation

1. You should see a message confirming that the database was created successfully.

2. You can now see your new database in the list.

Step 6: Manage the database

1. Click on your database name to add tables and manage your data.

Finally, with these steps, we have established a connection to your MySQL database from your Java application. You can now execute SQL queries to manipulate your data. You can extend this logic to include other operations such as updating, deleting, and retrieving data.

Answers to exercise 2:

```
package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Basic employee.

*/
public class Employee {

private static long idEmployeCounter = 0; // ID counter

private long employeeID; // Remains private as instructed

protected String lastName;

protected String firstName;

protected String dateOfBirth;

protected String position;

protected double salary;

// Default constructor

      public Employee() {
      this.employeeID = 0;
      this.lastName = null;
      this.firstName = null;
      this.dateOfBirth = null;
      this.position = null;
      this.salary = 0.0;
      }

// Constructor with parameters

      public Employee(String lastname, String firstName, String
dateOfBirth, String position, double salary) {
      idEmployeCounter++;

      employeeID = idEmployeCounter;
      this.lastName = lastname;
      this.firstName = firstName;
      this.dateOfBirth = dateOfBirth;
```

```
        this.position = position;
        this.salary = salary;

        }

// Getter/Setter if needed
public String getLastName() {
return this.lastName;
}

public String getFirstName() {
return this.firstName;

}

        public String getDateOfBirth() {
        return this.dateOfBirth;

        }

// We expose the ID via a method (employeeID is private)`
        public long getEmployeeID() {
        return this.employeeID;

        }

        public void description() {
        System.out.println(this.lastName + " " + this.firstName + " is an
employee, " + this.position +", and he/she has a monthly salary of: " +
this.salary + " €.");
        }

        public void description(String prenom, String nom) {

        System.out.println("The employee ID " + this.lastName + " " +
this.firstName + " is: " + this.getEmployeeID());

        }

        public void increaseSalary(double montant) {
        this.salary += montant;

        }

        public double getAnnualSalary() {
        return this.salary * 12;

        }
}

package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Manager (inherits from Employee).

*/
```

```
public class ExecutiveEmployee extends Employee {

protected String department;

protected double carAllowance;

      public ExecutiveEmployee(String lastName, String firstName, String
dateOfBirth, String position, double salary, String department, double
carAllowance) {
      super(lastName, firstName, dateOfBirth, position, salary);

      this.department = department;
      this.carAllowance = carAllowance;

      }

// Method specific to the ExecutiveEmployee class

      public void displayDepartment() {

      System.out.println("The department of " + this.lastName + " " +
this.firstName + " born on " + this.dateOfBirth + " is: " +
this.department);

      }

}

package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Temporary Employee (inherits from Employee).

*/
public class TemporaryEmployee extends Employee {

protected int contractDuration;

protected double hourlyRate;

public TemporaryEmployee(String name, String firstName, String dateOfBirth,
String position, double salary,

int contractDuration, double hourlyRate) {
super(name, firstName, dateOfBirth, position, salary);

this.contractDuration = contractDuration;

this.hourlyRate = hourlyRate;

}

// Method specific to the TemporaryEmployee class

public void displayContract() {
```

```
System.out.println(this.lastName + " " + this.firstName + " born on " +
this.dateOfBirth + " has a temporary contract for a duration of " +
this.contractDuration + " months");

}
}
package data.access;

/**
 *
 * @author USER.COM
 */
/**

* Temporary Employee (inherits from Employee).

*/
public class TemporaryEmployee extends Employee {

protected int contractDuration;

protected double hourlyRate;

public TemporaryEmployee(String name, String firstName, String dateOfBirth,
String position, double salary,

int contractDuration, double hourlyRate) {
super(name, firstName, dateOfBirth, position, salary);

this.contractDuration = contractDuration;

this.hourlyRate = hourlyRate;

}

// Method specific to the TemporaryEmployee class

public void displayContract() {

System.out.println(this.lastName + " " + this.firstName + " born on " +
this.dateOfBirth + " has a temporary contract for a duration of " +
this.contractDuration + " months");

}
}


package data.access;
/**
 *
 * @author USER.COM
 */
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.sql.*;
import java.text.SimpleDateFormat;
```

```
public class DAOEmployee {
    public void addExecutiveEmployee(ExecutiveEmployee executiveEmployee)
throws ParseException {
        String sql = "INSERT INTO employee (employeeType, lastName,
firstName, dateOfBirth, position, salary, contractDuration, hourlyRate,
department, carAllowance) VALUES (?,?,?,?,?,?,?,?,?,?)";

        try (Connection connection = DatabaseConnection.getConnection();

        PreparedStatement statement = connection.prepareStatement(sql)) {
        //statement.setInt(1, 1);

        statement.setString(1, "Executive Employee");

        statement.setString(2, executiveEmployee.lastName);

        statement.setString(3, executiveEmployee.firstName);

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

        java.util.Date date = sdf.parse(executiveEmployee.dateOfBirth);

        statement.setDate(4, new java.sql.Date(date.getTime()));

        statement.setString(5, executiveEmployee.position);

        statement.setDouble(6, executiveEmployee.salary);

        // Set contractDuration and hourlyRate as NULL if no value

        statement.setNull(7, java.sql.Types.INTEGER); // contractDuration

        statement.setNull(8, java.sql.Types.DOUBLE); // hourlyRate

        statement.setString(9, executiveEmployee.department);

        statement.setDouble(10, executiveEmployee.carAllowance);

        int rowsInserted = statement.executeUpdate();

        if (rowsInserted > 0) {

        System.out.println("A new Executive employee has been successfully
inserted!");

        }

        } catch (SQLException e) {

        System.out.println("Error inserting: " + e.getMessage());

        }

        }

        public void addTemporaryEmployee(TemporaryEmployee
temporaryEmployee) throws ParseException {
```

```
        String sql = "INSERT INTO employee (employeeType, lastName,
firstName, dateOfBirth, position, salary, contractDuration, hourlyRate,
department, carAllowance) VALUES (?,?,?,?,?,?,?,?,?,?)";

        try (Connection connection = DatabaseConnection.getConnection();

        PreparedStatement statement = connection.prepareStatement(sql)) {
        //statement.setInt(1, 1);

        statement.setString(1, "Temporary Employee");
        statement.setString(2, temporaryEmployee.lastName);
        statement.setString(3, temporaryEmployee.firstName);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        java.util.Date date = sdf.parse(temporaryEmployee.dateOfBirth);
        statement.setDate(4, new java.sql.Date(date.getTime()));
        statement.setString(5, temporaryEmployee.position);
        statement.setDouble(6, temporaryEmployee.salary);
        statement.setInt(7, temporaryEmployee.contractDuration);
        statement.setDouble(8, temporaryEmployee.hourlyRate);
        // Set department and indemnityCar as NULL if no value
        statement.setNull(9, java.sql.Types.NULL);
        statement.setNull(10, java.sql.Types.DOUBLE);

        int rowsInserted = statement.executeUpdate();

        if (rowsInserted > 0) {

        System.out.println("A new Temporary employee has been successfully
inserted!");

        }

        } catch (SQLException e) {

        System.out.println("Error during insertion: " + e.getMessage());

        }

}

public void view() {
    // Create the display window
    JFrame frame = new JFrame("Displaying Employees");
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE); // Close this
window only
    frame.setSize(900, 500);
    frame.setLocationRelativeTo(null);
    frame.setLayout(new BorderLayout());

    // Table model with columns corresponding to the employee table
    String[] columnNames = {
    "employeeID", "employeeType", "lastName", "firstName", "dateOfBirth",
"position", "salary", "contractDuration","hourlyRate","department",
"carAllowance"
    };

    DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0) {
    // Prevent cell editing
    @Override
    public boolean isCellEditable(int row, int column) {
    return false;
```

```
    }
    };

    JTable table = new JTable(tableModel);
    table.setAutoCreateRowSorter(true);
    // Sort by columns
    JScrollPane scrollPane = new JScrollPane(table);

    // Top panel for message/optional filter
    JPanel topPanel = new JPanel(new BorderLayout());
    JLabel lbl = new JLabel("Loading employees...");
    lbl.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
    topPanel.add(lbl, BorderLayout.WEST);
    frame.add(topPanel, BorderLayout.NORTH);
    frame.add(scrollPane, BorderLayout.CENTER);

    // Refresh button

    JPanel bottomPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
    JButton refreshBtn = new JButton("Refresh");
    bottomPanel.add(refreshBtn);
    frame.add(bottomPanel, BorderLayout.SOUTH);

    // Display the window immediately (non-blocking)
    frame.setVisible(true);

    // Method to load data from the database
    Runnable loadData = () -> {

    // Load into SwingWorker to avoid blocking the schedule
    SwingWorker<Void, Void> worker = new SwingWorker<>() {

    @Override

    protected Void doInBackground() {

    String sql = "SELECT employeeID, employeeType, lastName, firstName,
dateOfBirth, position, salary, contractDuration, hourlyRate, department,
carAllowance FROM employee";

    try(Connection conn = DatabaseConnection.getConnection();
    PreparedStatement ps = conn.prepareStatement(sql);
    ResultSet rs = ps.executeQuery()) {

    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

    // Empty the model before adding
    SwingUtilities.invokeLater(() -> tableModel.setRowCount(0));

    while (rs.next()) {
    int employeeID = rs.getInt("employeeID");
    String type = rs.getString("Employeetype");
    String lastName = rs.getString("lastName");
    String firstName = rs.getString("firstName");
    Date dateOfBirth = rs.getDate("dateOfBirth");
    String dateStr = (dateOfBirth != null) ? sdf.format(dateOfBirth) : "";

    String position = rs.getString("position");

    // Potentially null numeric values
```

```
    Double salary = rs.getObject("salary") != null ? rs.getDouble("salary")
: null;
    Integer contractDuration = rs.getObject("contractDuration") != null ?
rs.getInt("contractDuration") : null;
    Double hourlyRate = rs.getObject("hourlyRate") != null ?
rs.getDouble("hourlyRate") : null;

    String department = rs.getString("department");
    Double carAllowance = rs.getObject("carAllowance") != null ?
rs.getDouble("carAllowance") : null;

    Object[] row = new Object[] {
    employeeID,

    type != null ? type : "",
    lastName != null ? lastName : "",
    firstName != null ? firstName : "",

    dateStr,
    position != null ? position : "",
    salary != null ? salary : "",
    contractDuration != null ? contractDuration : "",
    hourlyRate != null ? hourlyRate : "",
    department != null ? department : "",
    carAllowance != null ? carAllowance : ""
    };

    // Add to the timetable
    SwingUtilities.invokeLater(() -> tableModel.addRow(row));
    }
    } catch (SQLException ex) {
    SwingUtilities.invokeLater(() -> {
    JOptionPane.showMessageDialog(frame,
    "Error reading employees: " + ex.getMessage(),
    "Error",
    JOptionPane.ERROR_MESSAGE);
    });

    }
    return null;
    }

    @Override
    protected void done() {
    lbl.setText("Displaying employees (" + tableModel.getRowCount() + ")");
    }

    };

    worker.execute();

    };
    // Load data immediately
    loadData.run();
    // Refresh on click
    refreshBtn.addActionListener(e -> {
    lbl.setText("Refreshing...");
    loadData.run();
    });

    }
```

```
}

package business.logic;

/**
 *
 * @author USER.COM
 */
import data.access.Employee;
import data.access.DAOEmployee;
import data.access.TemporaryEmployee;
import data.access.ExecutiveEmployee;
import java.text.ParseException;

/**

* Business logic layer — employee management operation.

*/
public class EmployeeManagement {
    private DAOEmployee dao;

    public EmployeeManagement(DAOEmployee dao) {
    this.dao = dao;

    }

    // Adds an employee via the DAO

    public void addExecutiveEmployee(ExecutiveEmployee executiveEmployee)
throws ParseException {
    dao.addExecutiveEmployee(executiveEmployee);

    }

    // Adds an employee via the DAO

    public void addTemporaryEmployee(TemporaryEmployee temporaryEmployee)
throws ParseException {
    dao.addTemporaryEmployee(temporaryEmployee);

    }

    // Displays employees via the DAO

    public void view() {
    dao.view();

    }
}

package presentation;

import javax.swing.JOptionPane;
import data.access.Employee;
import data.access.TemporaryEmployee;
import data.access.ExecutiveEmployee;
import data.access.DAOEmployee;
import business.logic.EmployeeManagement;
import java.text.ParseException;
import java.util.logging.Level;
```

```
import java.util.logging.Logger;

/**
 *
 * @author Mohamed Mohammedi
 */
public class EmployeeInterface extends javax.swing.JFrame {

    /**
     * Creates new form ExecutiveEmployee
     */

    // DAO initialization
        DAOEmployee dao = new DAOEmployee();

        // Business layer
        EmployeeManagement management = new EmployeeManagement(dao);


    public EmployeeInterface() {
        initComponents();// Method generated to initialize components

// Check the current selection and disable fields if necessary
    if (jComboBox1.getSelectedItem().toString().trim().equals("Executive
Employee")) {
        jLabel10.setEnabled(false);
        jTextField8.setEnabled(false);
        jLabel11.setEnabled(false);
        jTextField9.setEnabled(false);
    }
    }

    /**
     * This method is called from within the constructor to initialize the
form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jComboBox1 = new javax.swing.JComboBox<>();
        jTextField1 = new javax.swing.JTextField();
        jTextField2 = new javax.swing.JTextField();
        jTextField3 = new javax.swing.JTextField();
        jTextField4 = new javax.swing.JTextField();
        jTextField5 = new javax.swing.JTextField();
        jLabel8 = new javax.swing.JLabel();
        jTextField6 = new javax.swing.JTextField();
        jLabel9 = new javax.swing.JLabel();
```

```
        jTextField7 = new javax.swing.JTextField();
        jLabel10 = new javax.swing.JLabel();
        jTextField8 = new javax.swing.JTextField();
        jLabel11 = new javax.swing.JLabel();
        jTextField9 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jLabel13 = new javax.swing.JLabel();
        jButton3 = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Adding an Employee");

        jPanel1.setBackground(new java.awt.Color(255, 204, 255));
        jPanel1.setAlignmentX(0.1F);
        jPanel1.setAlignmentY(0.1F);
        jPanel1.setCursor(new
java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));

        jLabel1.setFont(new java.awt.Font("Segoe UI", 1, 24)); // NOI18N
        jLabel1.setText("Adding an Employee");

        jLabel2.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel2.setText("Employee type:");

        jLabel3.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel3.setText("First Name:");

        jLabel4.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel4.setText("Last Name:");

        jLabel5.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel5.setText("Date of birth:");

        jLabel6.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel6.setText("Position:");

        jLabel7.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel7.setText("Salary:");

        jComboBox1.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "Executive Employee", "Temporary Employee" }));
        jComboBox1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jComboBox1ActionPerformed(evt);
            }
        });

        jTextField1.setText("Michael");

        jTextField2.setText("Wilson");

        jTextField3.setText("1968-04-07");

        jTextField4.setText("Director");

        jTextField5.setText("10000.00");

        jLabel8.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel8.setText("Department:");
```

```
        jTextField6.setText("Human Resources");

        jLabel9.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel9.setText("Car allowance:");

        jTextField7.setText("1500.00");

        jLabel10.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel10.setText("Contract duration:");

        jTextField8.setText("Contract duration");

        jLabel11.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        jLabel11.setText("Hourly rate:");

        jTextField9.setText("Hourly rate");

        jButton1.setBackground(new java.awt.Color(255, 0, 153));
        jButton1.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton1.setText("Cancel");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jButton2.setBackground(new java.awt.Color(0, 255, 204));
        jButton2.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton2.setText("Save");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jLabel13.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/Presentation/businessman_add
(2).png"))); // NOI18N

        jButton3.setBackground(new java.awt.Color(255, 204, 51));
        jButton3.setFont(new java.awt.Font("Segoe UI", 1, 12)); // NOI18N
        jButton3.setText("Displays");
        jButton3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton3ActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(70, 70, 70)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
```

```
                        .addGroup(jPanel1Layout.createSequentialGroup()
                            .addGap(422, 422, 422)
                            .addComponent(jLabel13)
                            .addGap(0, 19, Short.MAX_VALUE))
                        .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.TRAILING)

.addGroup(jPanel1Layout.createSequentialGroup()
                                        .addComponent(jButton1)
                                        .addGap(61, 61, 61)
                                        .addComponent(jButton2)
                                        .addGap(61, 61, 61)
                                        .addComponent(jButton3))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                        .addComponent(jLabel8)

.addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                                .addComponent(jLabel9)
                                                .addComponent(jLabel7)
                                                .addComponent(jLabel6)
                                                .addComponent(jLabel5)
                                                .addComponent(jLabel4)
                                                .addComponent(jLabel3)
                                                .addComponent(jLabel10)
                                                .addComponent(jLabel11))
                                            .addGap(134, 134, 134)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                                                .addComponent(jTextField9,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                .addComponent(jTextField8,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                .addComponent(jTextField5,
javax.swing.GroupLayout.PREFERRED_SIZE, 85,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                .addComponent(jTextField4,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                                .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE, 85,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                                            .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jTextField1,
javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                            .addComponent(jComboBox1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))))
                            .addComponent(jLabel2))

.addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(205, 205, 205)
                .addComponent(jLabel1)
                .addGap(0, 0, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                    .addComponent(jLabel13)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(0, 26, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                            .addComponent(jLabel2,
javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(jComboBox1,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.LEADING)
                            .addComponent(jTextField1,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel3,
javax.swing.GroupLayout.Alignment.TRAILING))
                        .addGap(18, 18, 18)
```

```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                            .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel4))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                            .addComponent(jLabel5)
                            .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addGap(7, 7, 7)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField4,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel6))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField5,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel7))
                .addGap(26, 26, 26)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jLabel8)
                    .addComponent(jTextField6,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(18, 18, 18)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField7,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel9))
                .addGap(29, 29, 29)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jTextField8,
javax.swing.GroupLayout.PREFERRED_SIZE,
```

```
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel10))
                .addGap(26, 26, 26)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jLabel11)
                    .addComponent(jTextField9,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(47, 47, 47)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignme
nt.BASELINE)
                    .addComponent(jButton1)
                    .addComponent(jButton2)
                    .addComponent(jButton3))
                .addGap(45, 45, 45))
        );

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(29, Short.MAX_VALUE)
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton2ActionPerformed
    // TODO add your handling code here:
    // Display a confirmation message
        String Rec = jComboBox1.getSelectedItem().toString().trim();
        String lastName = jTextField1.getText().trim();
        String firstName = jTextField2.getText().trim();
        String dateOfbirth = jTextField3.getText().trim();
        String position = jTextField4.getText().trim();
```

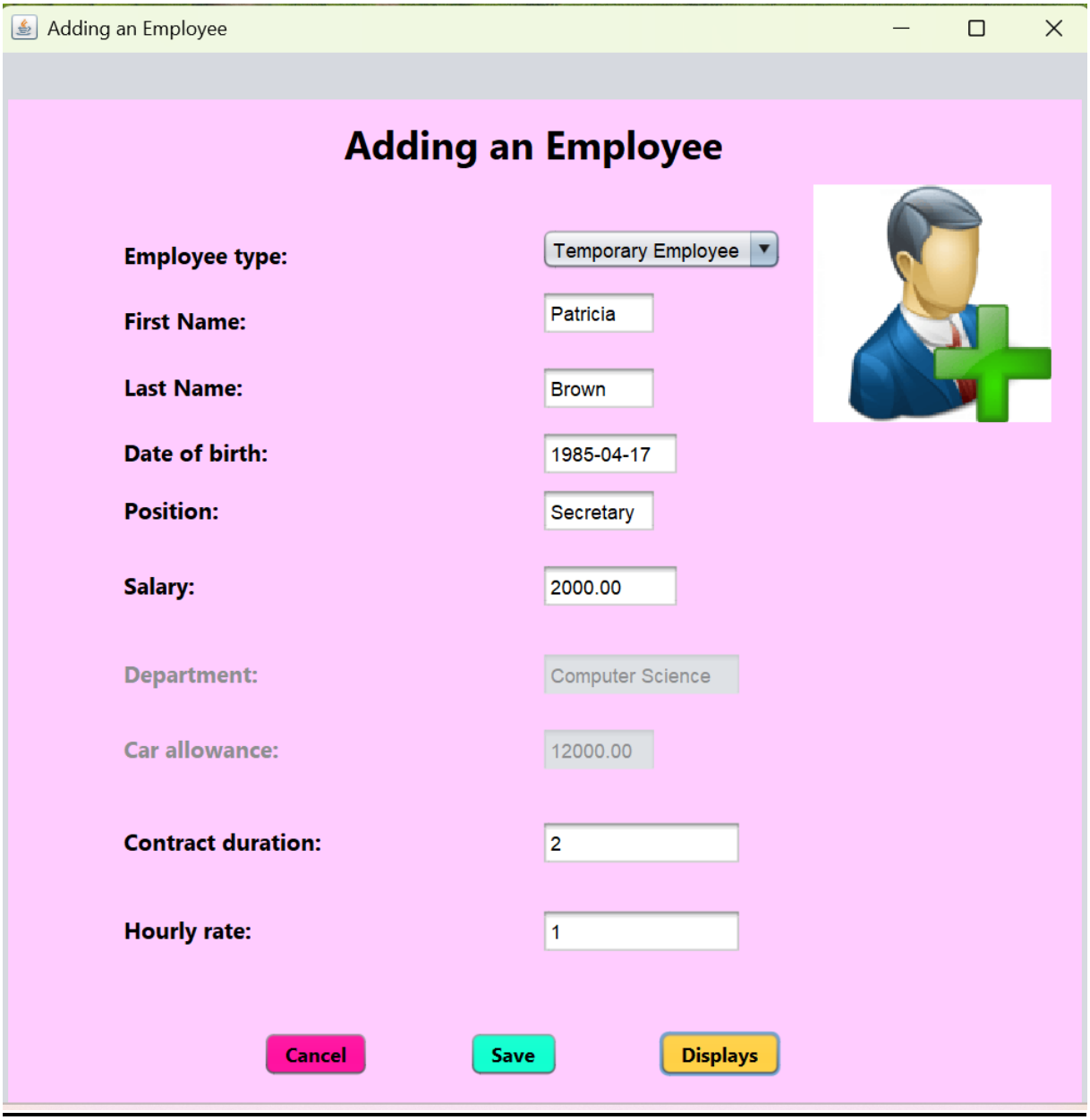
```
        try {
        double salary = Double.parseDouble(jTextField5.getText().trim());

        if (Rec.equals("Executive Employee")) {
        String department = jTextField6.getText().trim();
        double carAllowance =
Double.parseDouble(jTextField7.getText().trim());

        // Check field validity
        if (isValidExecutiveEmployee(lastName, firstName, dateOfbirth,
position, salary, department, carAllowance)) {

        // Create and add employees
        ExecutiveEmployee executiveEmployee = new
ExecutiveEmployee(lastName, firstName, dateOfbirth, position, salary,
department, carAllowance);
        management.addExecutiveEmployee(executiveEmployee);

        JOptionPane.showMessageDialog(this, "The information for
ExecutiveEmployee " + lastName + " " + firstName + " has been successfully
added!");

        } else {
        JOptionPane.showMessageDialog(this, "Failed to add
ExecutiveEmployee information. Please check the information.", "Error",
JOptionPane.ERROR_MESSAGE);
        }

        } else {
        int contractDuration =
Integer.parseInt(jTextField8.getText().trim());

        double hourlyRate =
Double.parseDouble(jTextField9.getText().trim());

        // Check field validity
        if (isValidTemporaryEmployee(lastName, firstName, dateOfbirth,
position, salary, contractDuration, hourlyRate)) {
        TemporaryEmployee temporaryEmployee = new
TemporaryEmployee(lastName, firstName, dateOfbirth, position, salary,
contractDuration, hourlyRate);
        management.addTemporaryEmployee(temporaryEmployee);

        JOptionPane.showMessageDialog(this, "The information for Temporary
Employee " + lastName + " " + firstName + " has been successfully added!");
        } else {
        JOptionPane.showMessageDialog(this, "Failed to add temporary
employee information. Please check the information.", "Error",
JOptionPane.ERROR_MESSAGE);
        }
        }
        } catch (NumberFormatException e) {
        // Handle the conversion error if the salary or other numeric field
is not a valid number
        JOptionPane.showMessageDialog(this, "Please enter valid numeric
values.", "Error", JOptionPane.ERROR_MESSAGE);
        }   catch (ParseException ex) {

Logger.getLogger(EmployeeInterface.class.getName()).log(Level.SEVERE, null,
ex);
```

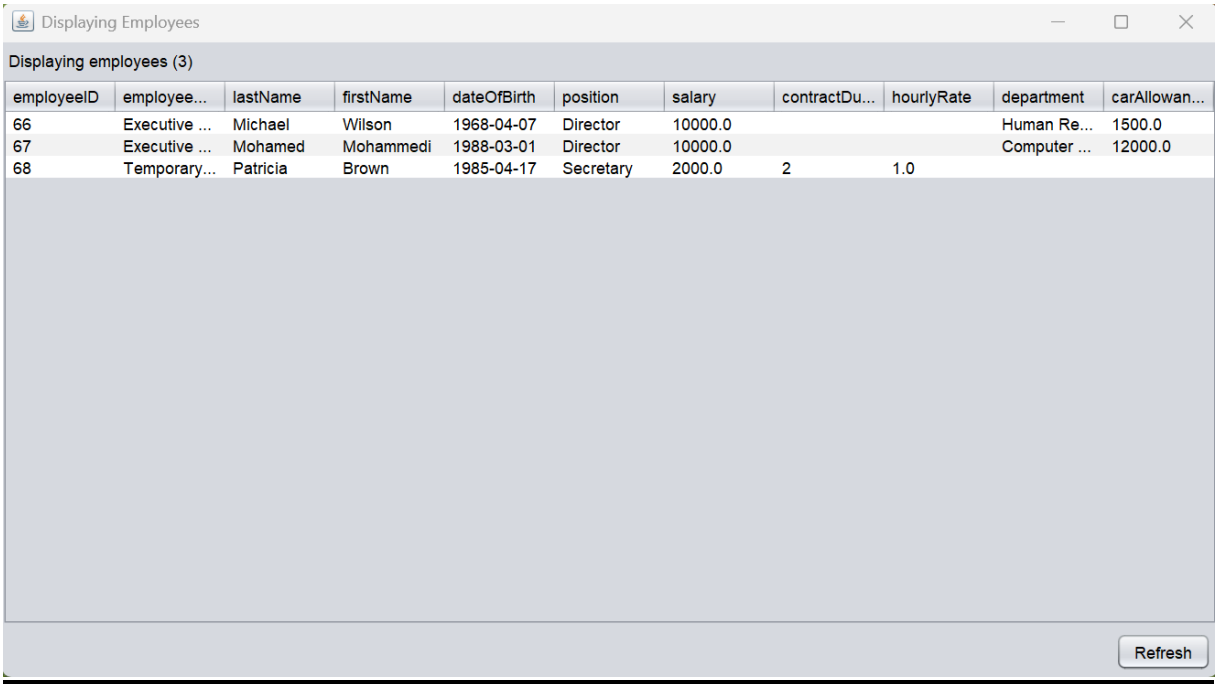
```
            }
    }//GEN-LAST:event_jButton2ActionPerformed

    // Validation methods for the ExecutiveEmployee
    private boolean isValidExecutiveEmployee(String lastName, String
firstName, String dateOfbirth, String position, double salary, String
department, double carAllowance) {
    return !lastName.isEmpty() && !firstName.isEmpty() &&
!dateOfbirth.isEmpty() && !position.isEmpty() && salary >= 0 &&
!department.isEmpty() && carAllowance >= 0;
    }

    // Validation methods for the temporary employee
    private boolean isValidTemporaryEmployee(String lastName, String
firstName, String dateOfBirth, String position, double salary, int
contractDuration, double hourlyRate) {
    return !lastName.isEmpty() && !firstName.isEmpty() &&
!dateOfBirth.isEmpty() && !position.isEmpty() &&  salary >= 0 &&
contractDuration >= 0 && hourlyRate >= 0;

    }

    private void jComboBox1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jComboBox1ActionPerformed
        // TODO add your handling code here:
        //Always activate the fields
    jLabel10.setEnabled(true);
    jTextField8.setEnabled(true);
    jLabel11.setEnabled(true);
    jTextField9.setEnabled(true);

    jLabel8.setEnabled(true);
    jTextField6.setEnabled(true);
    jLabel9.setEnabled(true);
    jTextField7.setEnabled(true);

    String Rec = jComboBox1.getSelectedItem().toString().trim();

    // Disable fields based on selection
    if (Rec.equals("Executive Employee")) {
        jLabel10.setEnabled(false);
        jTextField8.setEnabled(false);
        jLabel11.setEnabled(false);
        jTextField9.setEnabled(false);
    } else {
        jLabel8.setEnabled(false);
        jTextField6.setEnabled(false);
        jLabel9.setEnabled(false);
        jTextField7.setEnabled(false);
    }
    }//GEN-LAST:event_jComboBox1ActionPerformed

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
        // TODO add your handling code here:
        //Reset the fields
       jTextField1.setText("");
       jTextField2.setText("");
       jTextField3.setText("");
       jTextField4.setText("");
       jTextField5.setText("");
```

```
        jTextField6.setText("");
        jTextField7.setText("");
        jTextField8.setText("");
        jTextField9.setText("");
    }//GEN-LAST:event_jButton1ActionPerformed

    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton3ActionPerformed
        // TODO add your handling code here:
        // Display all registered employees
        management.view();
    }//GEN-LAST:event_jButton3ActionPerformed

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel
setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with
the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(EmployeeInterface.class.getName()).log(j
ava.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new EmployeeInterface().setVisible(true);
            }
        });
    }
```

```
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JComboBox<String> jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    private javax.swing.JTextField jTextField6;
    private javax.swing.JTextField jTextField7;
    private javax.swing.JTextField jTextField8;
    private javax.swing.JTextField jTextField9;
    // End of variables declaration//GEN-END:variables
}
```

The execution result is:



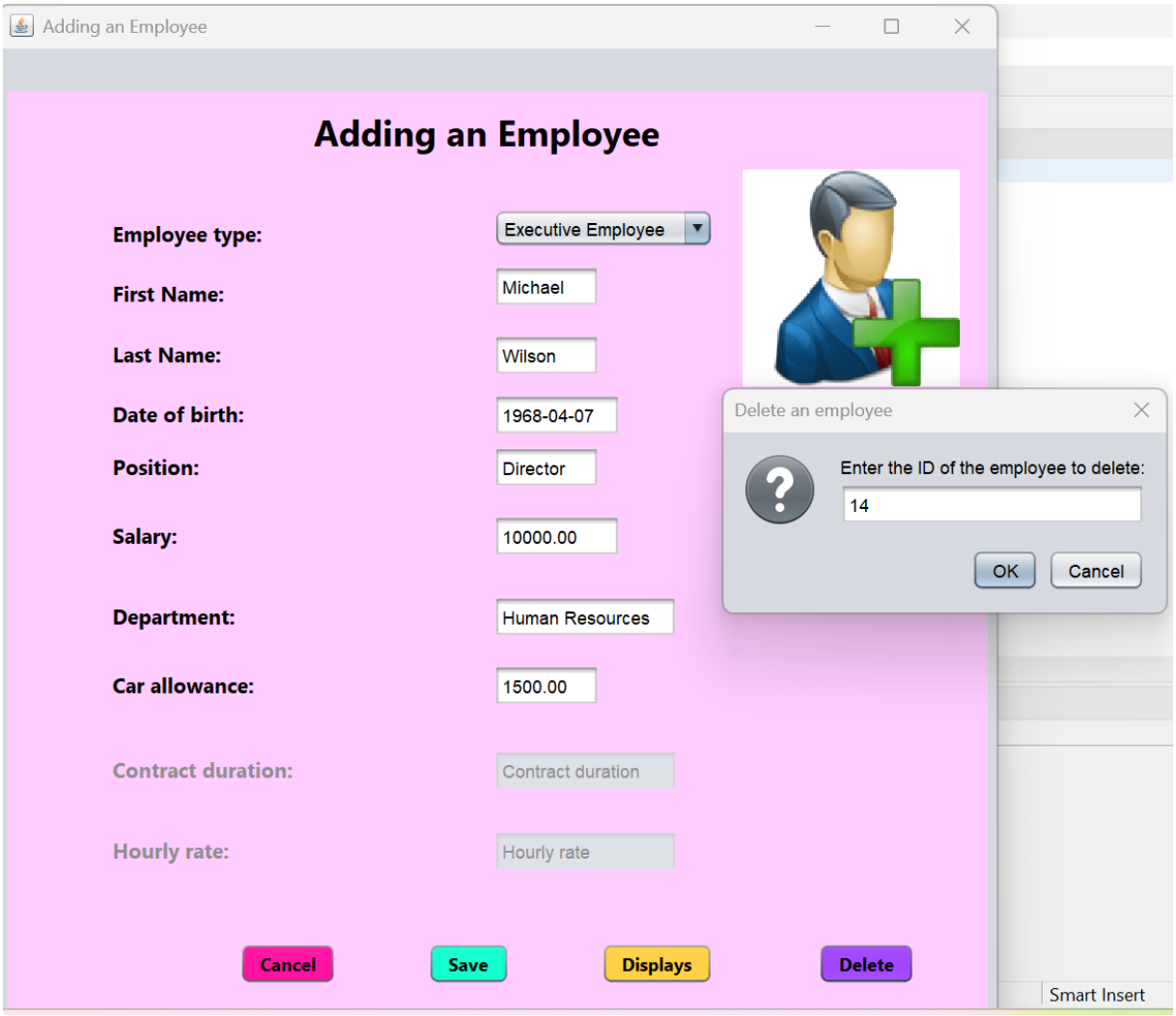


2. Implement a deletion feature: when the ''Delete'' button is clicked, display a dialog box asking for the ID of the employee to be deleted, request confirmation, and then delete the corresponding record in the database.

```
public void deleteEmployeeWithDialog() {
// parent: the parent window (can be null)

// Request the ID via an input dialog box

String input = JOptionPane.showInputDialog(null, "Enter the ID of the
employee to delete:","Delete an employee",JOptionPane.QUESTION_MESSAGE);

if (input == null) {

// The user canceled the dialog box

return;

}

input = input.trim();

if (input.isEmpty()) {

JOptionPane.showMessageDialog(null,

"Empty ID. Operation canceled.",

"Error",

JOptionPane.ERROR_MESSAGE);

return;

}
```

```
int id;

try {

id = Integer.parseInt(input);

} catch (NumberFormatException ex) {
JOptionPane.showMessageDialog(null,
"Invalid ID. Please enter an integer.",

"Error",

JOptionPane.ERROR_MESSAGE);

return;

}

// Ask for final confirmation
int confirm = JOptionPane.showConfirmDialog(null,

"Are you sure you want to delete the employee with id = " + id + "?",

"Confirm deletion",

JOptionPane.YES_NO_OPTION,

JOptionPane.WARNING_MESSAGE);

if (confirm != JOptionPane.YES_OPTION) {

// The user selected No or closed the dialog box

return;

}

// Execute the deletion in the database
String sql = "DELETE FROM employee WHERE employeeID = ?";
try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement ps = conn.prepareStatement(sql)) {

ps.setInt(1, id);

int rowsAffected = ps.executeUpdate();

if (rowsAffected > 0) {
JOptionPane.showMessageDialog(null,
"Employee successfully deleted (id = " + id + ").",

"Deletion successful",

JOptionPane.INFORMATION_MESSAGE);

} else {

JOptionPane.showMessageDialog(null,

"No employees found with id = " + id + ".",
```

```
"No results",

JOptionPane.INFORMATION_MESSAGE);
}
} catch (SQLException e) {
JOptionPane.showMessageDialog(null,
"Error deleting: " + e.getMessage(),
"SQL Error",
JOptionPane.ERROR_MESSAGE);

}
}
```



Lab 8: Mini Project

Outline

Goals

- Understanding of object-oriented programming concepts.
- Practical application of the MVC pattern.
- Interaction with a database.
- Development of graphical interfaces.
- Exception handling and robustness.
- Creation of an executable.
- Practical experience in software development.

8.1. Introduction

In this chapter, we will explore the practical application of fundamental object-oriented programming concepts through a mini-project. This project, titled ''MyApplication'', will serve as a learning framework for applying the principles of the MVC (Model-View-Controller) model, interacting with a database, and developing user-friendly graphical interfaces.

The main objective is to provide hands-on experience in software development, integrating exception handling, executable creation, and the robustness required to ensure the application's proper functioning. The following sections will detail the mini-project brief, the design and development steps, and answer keys to facilitate understanding and learning of the concepts covered.

This project represents a unique opportunity to deepen your knowledge and strengthen your programming skills while creating a functional application that simulates a user, product, and order management system.

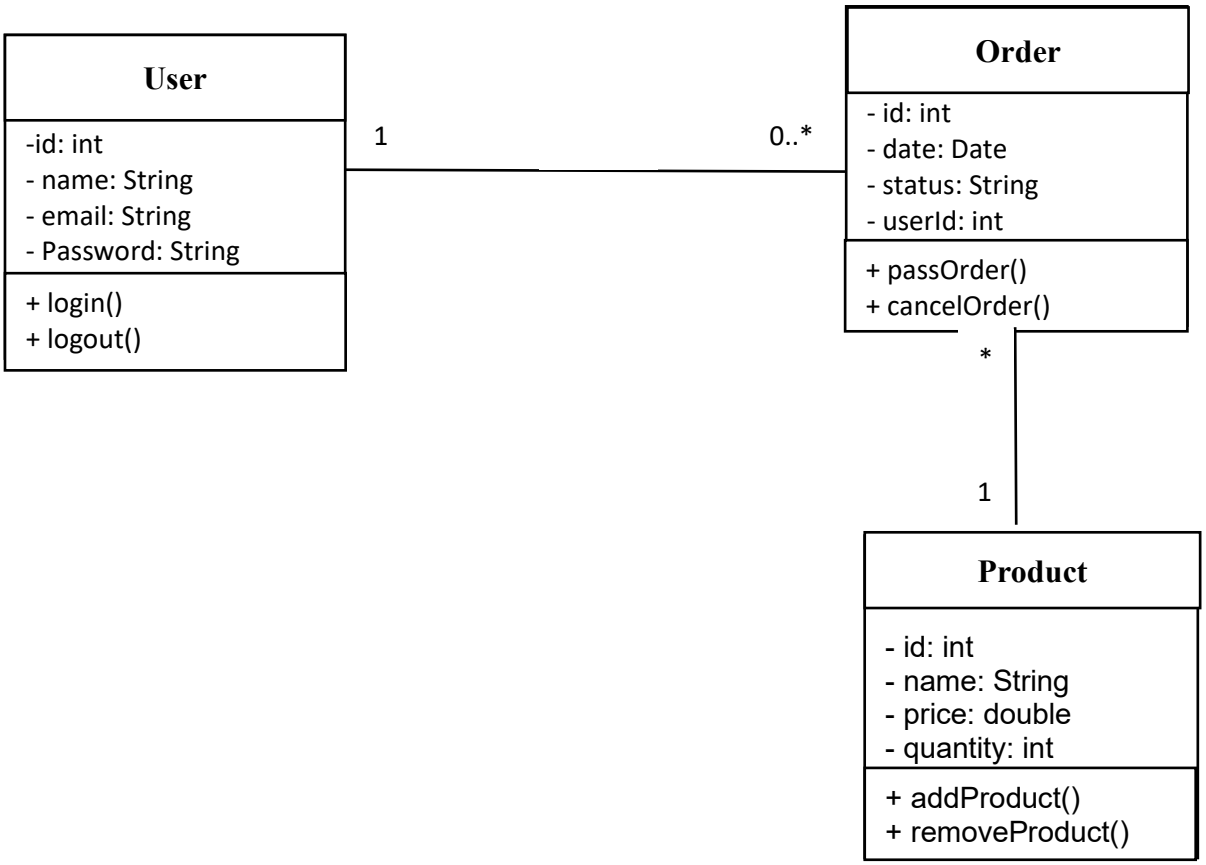
8.2. Mini Project Statement

a) Create a Java Project in NetBeans.

- Open NetBeans and create a new Java project. Name it ''MyApplication''.

b) Class Diagram

- Complete the class diagram by including the entities required for your project. Here's an example diagram:



- Class Description:

1. User
    a. Attributes: id, name, email, password
    b. Methods: login(), logout()

2. Order
    a. Attributes: id, date, status, userId
    b. Methods: placeOrder(), cancelOrder()
    c. Relationship: A user can have multiple orders.

3. Product
    a. Attributes: id, name, price, quantity
    b. Methods: addProduct(), deleteProduct()
    c. Relationship: An order can contain multiple products.

c) Database

- Create the database in your database management system (e.g., MySQL). Name it ''my_database''.
- Create the necessary tables with referential integrity constraints (e.g., users, products, orders).

d) Application Interfaces

- Design and create the graphical interfaces that provide access to all application features (e.g., a login interface, a dashboard, an order form, etc.).

e) Implement the Application Using the MVC Pattern

- Organize your code using the MVC (Model-View-Controller) pattern to separate business logic, the user interface, and data control.

f) Connect Your Project to Your Database

- Use JDBC (Java Database Connectivity) to establish a connection between your Java application and the database you created. Make sure to handle exceptions and close connections properly.

g) Output Statuses

- Display a result related to the statement, for example, an order summary or a user's product list.

h) Application Executable

- Generate your application's executable. You can create an executable JAR file from your project to facilitate deployment.

8.3. Correction of the mini project

This program uses Java Swing for the user interface and JDBC for the database connection.

Project Structure

The general structure of the project is as follows:

```
MyApplication/
├── src/
│   ├── Database.java
│   ├── User.java
│   ├── Order.java
│   ├── Product.java
│   ├── ProductDAO.java
│   ├── InterfaceConnection.java
│   └── Controller.java
└── lib/
    └── mysql-connector-java-x.x.x.jar
```

1. Database Class

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
public class Database {
private static final String URL =
"jdbc:mysql://localhost:3306/my_database";
private static final String USER = "root"; // Change according to
your configuration
private static final String PASSWORD = "your_password"; // Change
according to your configuration

public static Connection getConnection() throws SQLException {
return DriverManager.getConnection(URL, USER, PASSWORD);
}
}
```

2. User Class

```
public class User {
private int id;
private String name;
private String email;
private String password;

public User(int id, String name, String email, String password) {
this.id = id;
this.name = name;
this.email = email;
this.password = password;
}

public String getEmail() {
return email;
}

public String getPassword() {
return password;
}

public void logIn() {
System.out.println(name + " is logged in.");
}

public void logOut() {
System.out.println(name + " is logged out.");
}
}
```

3. Class Order

```
import java.util.Date;

public class Order {
private int id;
private Date date;
private String status;
private int userId;

public Order(int id, Date date, String status, int userId) {
this.id = id;
this.date = date;
this.status = status;
this.userId = userId;
```

```
}

public void placeOrder() {
System.out.println("Order " + id + " placed.");
}

public void cancelOrder() {
System.out.println("Order " + id + " canceled.");
}
}
```

4. Product Class

```
public class Product {
private int id;
private String name;
private double price;
private int quantity;

public Product(int id, String name, double price, int quantity) {
this.id = id;
this.name = name;
this.price = price;
this.quantity = quantity;
}

public String getName() {
return name;
}

public void addProduct() {
System.out.println(name + "added.");
}

public void removeProduct() {
System.out.println(name + "removed.");
}
}
```

5. ProductDAO Class

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ProductDAO {
public List<Product> getAllProducts() {
List<Product> products = new ArrayList<>();
try (Connection conn = Database.getConnection();
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM products")) {

while (rs.next()) {
Product product = new Product(rs.getInt("id"), rs.getString("name"),
rs.getDouble("price"), rs.getInt("quantity"));
products.add(product);
}
} catch (SQLException e) {
e.printStackTrace();
}
return products;
}
```

```
}
```

6. Controller Class

```
public class Controller {
private User user;

public Controller(User user) {
this.user = user;
}

public void login(String email, String password) {
if (user.getEmail().equals(email) &&
user.getPassword().equals(password)) {
user.Login();
} else {
System.out.println("Login failed.");
}
}
}
```

7. ConnectionInterface Class

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ConnectionInterface {
public static void main(String[] args) {
JFrame frame = new JFrame("Connection");
JPanel panel = new JPanel();
frame.setContentPane(panel);
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

JTextField emailField = new JTextField(20);
JPasswordField passwordField = new JPasswordField(20);
JButton loginButton = new JButton("Login");

panel.add(new JLabel("Email:"));
panel.add(emailField);
panel.add(new JLabel("Password:"));
panel.add(passwordField);
panel.add(loginButton);

// Example user
User user = new User(1, "Jean Dupont", "jean@example.com",
"password");

loginButton.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
String email = emailField.getText();
String password = new String(passwordField.getPassword());
Controller controller = new Controller(user);
controller.connect(email, password);
}
});

frame.pack();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

```
        }
        }
```

Execution Instructions

1. Configure MySQL:

– Ensure MySQL is installed and running.

– Create the "my_database" database and the ‘’users’’, ‘’products’’, and ‘’orders’’ tables as described above.

2. Add the JDBC Driver:

– Download the MySQL Connector/J driver and add it to your project's classpath (in the ‘’lib’’ folder or via the project properties in NetBeans).

3. Compile and Run:

– In NetBeans, compile the project.

– Run the ‘’ConnectionInterface’’ class.

h) Application Executable

Generate the JAR File:

– Right-click on the project in NetBeans > ‘’Clean and Build’’.

– The JAR file will be generated in your project's ‘’dist’’ folder.

Note:

The full program above allows you to log in as a user and view the basic features of the completed application. You can extend it with additional features such as order and product management.

Conclusion

This lab booklet was designed to provide a comprehensive and immersive learning experience in Java programming. Through the various sections, we explored key concepts ranging from installing the NetBeans development environment to creating interactive applications and database management. Each lab booklet was structured to foster the acquisition of the skills needed to complete real-world projects, while also providing a deep understanding of the principles of object-oriented programming and software architecture.

The included exercises and answer keys are designed to strengthen your independence and encourage critical thinking. The final mini-project provides a valuable opportunity to put all the knowledge you've acquired into practice, preparing you to tackle real-world challenges in software development.

In conclusion, we hope this booklet has helped you develop your technical skills while stimulating your curiosity and creativity. Learning to program is an ongoing process, and I encourage you to continue your exploration and get involved in future projects. Thanks to the methodical approach adopted in this document, you are now better equipped to navigate the world of software development and confidently tackle larger projects.

The author of this lab booklet, Dr. Mohamed Mohammedi, would like to express his gratitude to all the students who have used this document. Your commitment and passion for learning are a source of inspiration. He hopes that the knowledge you have acquired here will accompany you throughout your professional career and help you become competent and innovative players in the field of software development.

Bibliography

[1] GODIN, Robert et LEMIRE, Daniel. Java pas à pas: Introduction à la programmation et au langage Java. 2024.

[2] ING39 Introduction à NetBeans FIP/Mise à jour Java 2021-2022. https://fr.readkong.com/page/slides/ing39-introduction-a-netbeans-fip-mise-a-jour-java-3523125

[3] Romain LEMOUNEAU et Thomas BROUSSARD. Apache NetBeans : Développez vos applications en Java, Éditeur ENI, 2021.

[4] KERBOEUF, Mickaël. Algorithmique et programmation objet-Java en résumé-200 exercices corrigés. Editions Ellipses, 2020.

[5] K. BEDJOU, Génie logiciel, support de TP, Université de Bejaia, https://elearning.univ-bejaia.dz/enrol/index.php?id=4389, 2019.

[6] Hugues Bersini. La programmation orientée objet Cours et exercices UML 2 avec Java, C#, C++, Python, PHP et LINQ. Collections : Noire, Edition 6, Editions eyrolles.

[7] GROUSSARD, Thierry. Java 6: les fondamentaux du langage Java. Editions ENI, 2009.

[8] DELANNOY, Claude. Programmer en Java. Editions Eyrolles, 2008.

[9] ROSSI, Fabrice. Initiation à la programmation Java. Université Paris-IX Dauphine, 1997.

Abstract

This practical workbook has been designed to support students in the practical learning of Java programming, by addressing fundamental and advanced software development concepts through a series of practical exercises, while emphasizing essential tools and methodologies. Its objectives are to familiarize students with the NetBeans development environment, review the basics of the Java language and deepen their understanding of object-oriented programming through practical exercises, explore the creation of graphical interfaces and the MVC (Model-View-Controller) architecture, as well as integrate modeling concepts such as the transition from UML (Unified Modeling Language) to the relational model and learn how to establish connections to MySQL databases. The document is structured in several chapters, each dedicated to a specific theme, including an introduction, practical exercises to apply knowledge and answer keys to facilitate self-assessment, with a final mini-project allowing students to synthesize their learning by creating a complete application. Thus, this booklet aims to offer a complete educational approach, combining theory and practice, in order to prepare students for the challenges of software development in a professional environment.

Who is this book for?

This document is intended for third-year undergraduate students in Mathematics and Computer Science (MCS), National Recruitment (NR), as well as for graduate school students wishing to deepen their knowledge of Java development. It is also intended for anyone wishing to acquire practical programming skills, whether in an educational or self-taught setting.