Ministère de l'Enseignement supérieur et de la Recherche Scientifique
Université Abderrahmane Mira Bejaia
Faculté des sciences exactes
Département d'informatique

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Manuel des Travaux pratiques du module POO

A l'usage des étudiants de 2ème année Licence informatique MI, RN et Ecoles supérieures

Etabli par Dr. EL BOUHISSI Houda Epouse BRAHAMI

2021 / 2022

Préambule

Le présent support de travaux pratiques est un élément indispensable de l'enseignement, qui soutient et illustre l'enseignement du cours et offre aux étudiants un accompagnement pédagogique intéressant en vue de synthétiser leurs connaissances.

Ainsi, ce document est le fruit de plusieurs années d'enseignement du module « *Programmation Orientée Objet* » aux étudiants de la 2ème année informatique. Il est établi conformément au programme d'enseignement et il sera mis à la disposition de l'ensemble de la communauté universitaire élargie.

Ce fascicule propose plusieurs séries de travaux pratiques relatives à l'offre de formation homologuée. Chaque série de travaux pratiques regroupe un ensemble d'activités et chaque activité est accompagnée avec son corrigé-type.

Le fascicule propose entre autres un ensemble d'activités de perfectionnement, j'ai mis une solution-type uniquement pour les activités pertinentes.

Objectif du support de TP

Ce polycopié permet à l'étudiant d'acquérir les connaissances nécessaires pour écrire des programmes orientés objets en langage java.

Objectifs spécifiques

- Comprendre les concepts de base de la programmation orientée objet.
- Implémenter des classes d'objets : créer une classe en précisant ses propriétés et ses opérations et leur visibilité, et en définissant ses constructeurs.
- Comprendre le principe d'héritage.
- Comprendre le principe du polymorphisme
- Comprendre les notions d'interfaces et des classes abstraites.
- Apprendre à utiliser un environnement de réalisation de projets en java.
- Comprendre le principe de gestion des exceptions.
- Apprendre à gérer des collections

Prérequis

Programmation, algorithmique et structure de données.

Table des matières

Introduction ..... Page 5

Notions fondamentales ..... Page 6

Memento Java ..... Page 9

L'environnement Netbeans ..... Page 24

TP 1 : Initiation à la programmation Orientée Objet ..... Page 33

TP 2 : Tests et boucles ..... Page 44

TP 3 : Classes, Objets et méthodes ..... Page 49

TP 4 : Héritage, Polymorphisme, classes abstraites et interfaces ..... Page 52

TP 5 : Interfaces graphiques ..... Page 69

TP 6 : Collections ..... Page 78

TP 7 : Exceptions ..... Page 85

Activités de perfectionnement ..... Page 87

Références

Introduction

Ce document traite en grande partie le programme du module « Programmation orientée objet » selon le canevas académique de licence informatique établi par la tutelle et pourrait être utilisé pour les étudiants de la spécialité « Recrutement national en informatique et des écoles supérieures ainsi que les étudiants de la filière Mathématique-Informatique.

Il introduit des notions fondamentales (memento) du langage java à savoir, les types de données, les structures alternatives et répétitives pour faciliter l'écriture des programmes ainsi que des notes de cours jugés utiles pour faciliter les activités pédagogiques.

Le présent document regroupe en outre un ensemble de travaux pratiques sur les briques de base de la programmation orientée objet et du langage de programmation java.

Aussi, il renferme différentes activités pédagogiques de travail de réflexion et de manipulation, certaines activités sont accompagnées de corrigés types pour assister l'étudiant, d'autres nécessitant des manipulations sont laissé aux soins de l'étudiant.

Notions fondamentales

1. Introduction

Cette partie a pour but de montrer comment certains concepts de la programmation orientée objet sont implémentés en Java, à commencer, bien entendu par les notions fondamentales d'objet et de classe.

En préliminaire, notons bien que contrairement au C++, Java n'autorise ni les fonctions globales ni les variables globales ce qui a pour effet d'imposer l'encapsulation de toute déclaration de variables ou de tout code à l'intérieur de classes. Ceci fait de Java un langage à objets par opposition au C++ qui reste un langage orienté objets.

2. Les classes

Rappelons qu'un objet est une entité cohérente qui regroupe des données sous la forme d'attributs et le code qui l'exploite sous forme de méthodes.

Une classe est la description d'un ensemble d'objets de structure identique. Elle définit l'ensemble des attributs qui composent un objet et fournit le code des méthodes.

En Java, les classes sont organisées en package qui forment une arborescence. Cette notion est orthogonale à celle d'héritage. Un package est un ensemble de classes attachées à un même concept. Le meilleur exemple est assurément donné par l'API Java fournie en standard avec le compilateur. On trouve les packages suivants :

- java.io où on trouve toutes les classes associées aux entrées sorties
- java.net contient les classes spécialisées dans la programmation des réseaux.

Les packages peuvent à leur tour se ramifier en sous packages. Par exemple, le package java.rmi possède plusieurs sous packages dont java.rmi.server et java.rmi.activation. L'idée consiste à faire descendre tous vos packages d'une même racine qui permette de vous identifier rapidement comme le concepteur des classes.

Le nom de ce package de haut niveau peut être, par exemple, l'acronyme de votre université. Par exemple, la plupart des classes que vous rencontrerez au cours de ce document vont appartenir au package elearning.univ-bejaia.dz.

Il est important de noter que l'organisation sur disque de vos fichiers doit refléter celle des packages. En effet, chaque package ou sous package est associé à un répertoire du disque. L'imbrication des package étant directement traduite par celle des répertoires. Par exemple, une classe appartenant au package elearning.univ-bejaia.dz se trouvera dans un sous répertoire elearning/univ-bejaia/dz.

En effet, l'emplacement de la racine peut se trouver n'importe où sur votre disque pour peu que sa racine soit spécifiée dans une variable d'environnement nommée CLASSPATH.

En fait, il est même possible de spécifier plusieurs racines différentes à condition qu'elles soient toutes présentes dans le CLASSPATH. Pour bien comprendre son utilité, il nous faut étudier le modèle d'exécution du Java.

De même que le nom du package correspond au chemin menant au fichier, le nom du fichier source Java est représentatif de la classe qu'il contient. Par exemple, si vous voulez créer une classe Journal dans le package elearning.univ-bejai.dz le fichier doit s'appeler Journal.java et doit être situé dans le sous répertoire elearning/univ-bejaia/dz soit au final elearning/univ-bejaia/dz/Journal.java.

Si le fichier doit porter le même nom que la classe, est ce que cela signifie qu'un même fichier de source (d'extension .java) ne peut contenir qu'une seule classe ? pas tout à fait. Il ne peut contenir qu'une seule classe *publique,* c'est à dire une seule classe de visibilité universelle.

On dit qu'une classe est publique si elle est visible, c'est à dire référençable par une autre classe n'appartenant ni au même fichier source ni au même package. Un même fichier source peut donc contenir autant de classes non publiques que vous le souhaitez, mais une seule classe publique. Selon le modificateur de visibilité que vous utiliserez, les classes non publiques pourront être accessibles aux classes appartenant au même package ou uniquement aux classes contenues dans le même fichier source.

Les package permettent de rassembler les classes associées à un même concept. L'organisation sur disque des fichiers source doit refléter l'organisation en packages. Un même fichier source (.java) ne peut contenir qu'une seule classe publique universellement accessible mais peut également contenir plusieurs autres classes dont l'accès est restreint aux classes du même package ou même aux classes appartenant au même fichier source. Il est possible d'imbriquer des classes.

3. Compiler du code source java

Rappelons que la compilation d'un fichier source .java ne fournit pas du code objet natif mais un format intermédiaire connu sous le nom de byte code. C'est ce format standard qui sera ensuite interprété vers des instructions natives sur la machine d'exécution par la machine virtuelle java.

Si vous optez pour l'environnement de développement minimaliste fourni par Sun et nommé JDK (Java Development toolkit) le compilateur porte le nom de javac (java compiler). Pour compiler un fichier .java, il suffit de taper :

javac {chemin_complet}fichier.java

Bien entendu, il est possible de fournir des options au compilateur. Toutefois, du fait des spécificités du langage et en particulier le fait de passer par un byte code et l'absence de préprocesseur font qu'il y a très peu d'options de compilation.

Memento du langage Java

Dans ce qui suit, nous allons présenter brièvement quelques notions syntaxiques de base utilisées pour les activités du TP, en particulier, les types de données, les tableaux et chaines de caractères, et les structures conditionnelles et répétitives.

1. Types primitifs

Le tableau suivant illustre les types primitifs du langage Java

Type	Classe éq.	Valeurs	Portée	Défaut
`boolean`	`Boolean`	`true` ou `false`	N/A	`false`
`byte`	`Byte`	entier signé	{-128..128}	o
`char`	`Character`	caractère	{\uoooo..\uFFFF}	\uoooo
`short`	`Short`	entier signé	{-32768..32767}	0
`int`	`Integer`	entier signé	{-2147483648..2147483647}	0
`long`	`Long`	entier signé	$\{-2^{31}..2^{31}-1\}$	0
`float`	`Float`	réel signé	$\{-3,4028234^{38}..3,4028234^{38}\}$ $\{-1,40239846^{-45}..1,40239846^{-45}\}$	0.0
`double`	`Double`	réel signé	$\{-1,797693134^{308}..1,797693134^{308}\}$ $\{-4,94065645^{-324}..4,94065645^{-324}\}$	0.0

Java est un langage très rigoureux sur le typage des données. Il est interdit d'affecter à une variable la valeur d'une variable d'un type différent si cette seconde variable n'est pas explicitement transformée. (Voir exemple).

```
int a ;
double b = 5.0 ;
a = b ;
```

est interdit et doit être écrit de la manière suivante :

```
int a ;
double b = 5.0 ;
a = (int)b ;
```

Pour calculer le reste d'une division de a par b on utilise l'instruction : reste =a%b ; L'instruction '' = '' sert à affecter une valeur à une variable Tandis que l'instruction '' == '' sert à comparer deux valeurs.

2. Tableaux

Les tableaux en Java sont statiques (seulement, il existe aussi des tableaux dynamiques) ; la taille d'un tableau est fixée à la création.

L'accès à un tableau statique est plus rapide que l'accès à un tableau dynamique

2.1.Tableaux statiques

Les deux syntaxes suivantes sont acceptées pour déclarer un tableau d'entiers :

```
int[] mon_tableau ; // une déclaration possible
int mon_tableau2[]; // une autre déclaration possible
```

Un tableau a toujours une taille fixe (seulement il existe les tableaux dynamiques) qui doit être précisée avant l'affectation de valeurs à ses indices, de la manière suivante : `int[] mon_tableau = new int[20];`

De plus, la taille de ce tableau est disponible dans une variable *length* appartenant au tableau et accessible par ***mon_tableau.length***. On peut également créer des matrices ou des tableaux à plusieurs dimensions en multipliant les crochets, par exemple :

Int [][] ma_matrice;

On accède aux éléments d'un tableau en précisant un indice entre crochets : mon_tableau[3] : est le quatrième entier du tableau

Un tableau de taille n stocke ses éléments à des indices allant de *0 à n-1*.

2.2.Tableaux dynamiques

La librairie Java contient deux classes qui permettent de créer des tableaux dynamiques ; les deux classes font partie du package *java.util* et s'appellent *Vector* et *ArrayList. C*es tableaux ne peuvent contenir que des références vers des objets.

Une ArrayList peut contenir n'importe quel type d'objet ; l'ajout d'un type de données entre crochets fait que Java vérifie que vous affectez les types appropriés à une liste.

Vous pouvez omettre les crochets et le type de données après le nom de la classe *ArrayList*, mais vous recevez un avertissement vous indiquant que vous utilisez une opération non contrôlée ou non sécurisée.

Le constructeur par défaut crée une *ArrayList* avec une capacitée de 10 éléments. La capacité d'une *ArrayList* est le nombre d'éléments qu'elle peut tenir sans avoir à augmenter sa taille. Par définition, la capacité d'une *ArrayList* est supérieure ou égale à sa taille.

Vous pouvez également spécifier une capacité si vous le souhaitez. L'exemple suivant déclare une *ArrayList* qui peut contenir 20 noms.

```
ArrayList< String> noms = new ArrayList< String>(20);
```

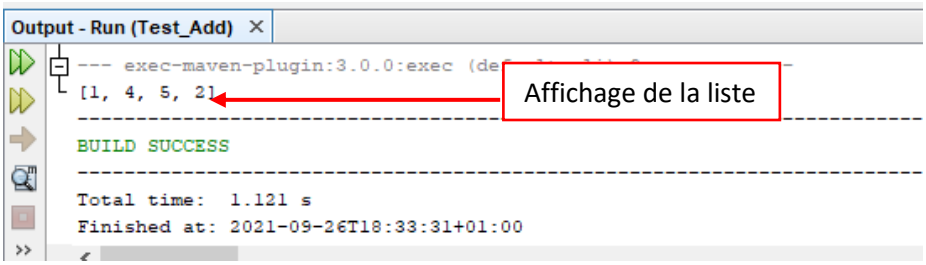
Remarque : Si vous savez que vous aurez besoin de plus de 10 éléments au départ, il est plus efficace de créer une liste de tableaux avec une plus grande capacité.

Le tableau suivant résume quelques méthodes utiles *ArrayList*

Méthode	Description
`public void add(Object)`	Ajouter un élément à une ArrayList; la version par défaut ajoute un élément au prochain emplacement disponible; une version surchargée vous permet de spécifier une position à laquelle nous voulons ajouter l'élément
`public void add(int, Object)`	
`public void remove(int)`	Supprimer un élément d'une ArrayList à un emplacement spécifié
`public void set(int, Object)`	Modifier un élément à un emplacement spécifié dans une ArrayList
`Object get(int)`	Récupérer un élément d'un emplacement spécifié dans une ArrayList
`public int size()`	Renvoyer la taille actuelle de ArrayList

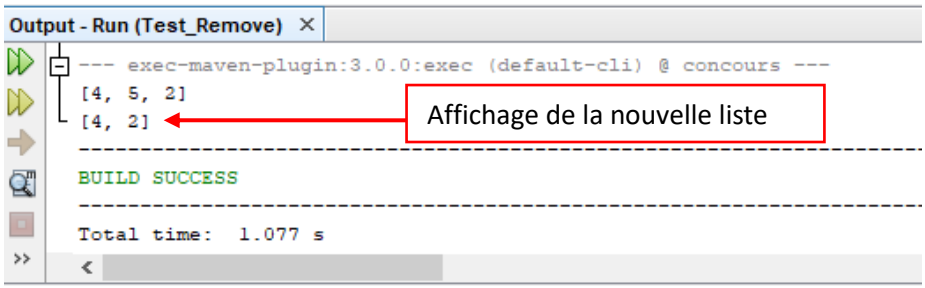
Exemple 1 : Ce programme permet d'ajouter des éléments de différents types de données à la liste.

```
import java.util.ArrayList;
public class Test_Add {
    public static void main(String args[]) {
      ArrayList< Integer> liste = new ArrayList< Integer>();
      liste.add(4);
      liste.add(5);
      liste.add(2);
      liste.add(0, 1);
      System.out.println(liste);
    }
}
```



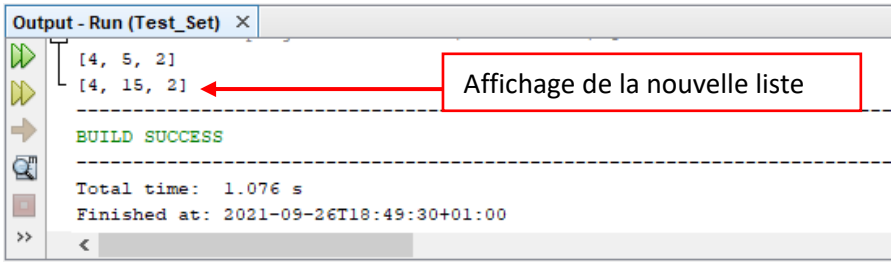
Exemple 2 : Ce programme illustre l'utilisation de la méthode remove pour supprimer des éléments de la liste.

```
import java.util.ArrayList;
 public class Test_Remove {
    public static void main(String args[]) {
      ArrayList< Integer> liste = new ArrayList< Integer>();
      liste.add(4);
      liste.add(5);
      liste.add(2);
      System.out.println(liste);
      liste.remove(1);
      System.out.println(liste);
   }
}
```



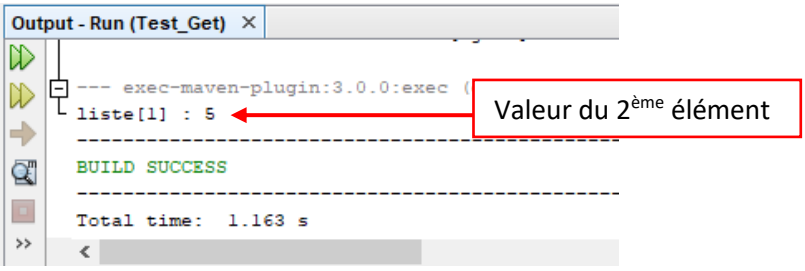
Exemple 3 : Ce programme montre comment modifier la valeur des éléments d'une liste.

```
import java.util.ArrayList;
 public class Test_Set {
    public static void main(String args[]) {
      ArrayList< Integer> liste = new ArrayList< Integer>();
      liste.add(4);
      liste.add(5);
      liste.add(2);
      System.out.println(liste);
      liste.set(1, 15);
      System.out.println(liste);
   }
}
```



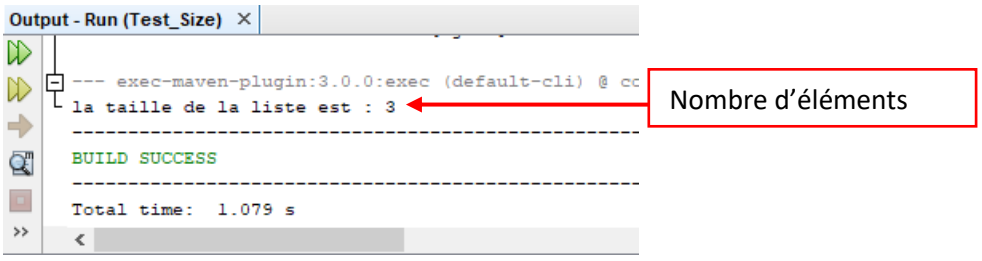
Exemple 4 : Ce programme montre comment récupérer les valeurs des éléments d'une liste.

```
import java.util.ArrayList;
 public class Test_Get {
    public static void main(String args[]) {
      ArrayList< Integer> liste = new ArrayList< Integer>();
      liste.add(4);
      liste.add(5);
      liste.add(2);
      System.out.println("liste[1] : " + liste.get(1));
   }
}
```



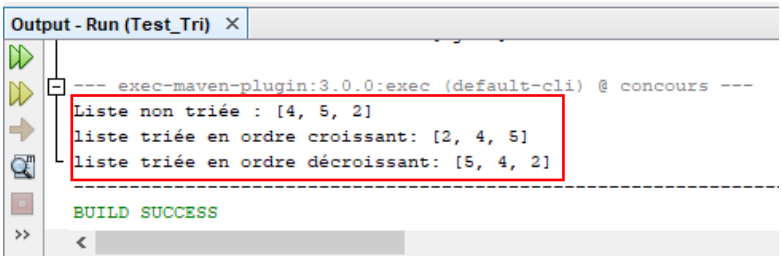
Exemple 5 :

```
import java.util.ArrayList;
 public class Test_Size {
    public static void main(String args[]) {
      ArrayList< Integer> liste = new ArrayList< Integer>();
      liste.add(4);
      liste.add(5);
      liste.add(2);
      System.out.println("la taille est : " + liste.size());
   }
}
```



Exemple 6 : Vous pouvez trier une *ArrayList* à l'aide de la méthode *Collections.sort()* et en fournissant *ArrayList* comme argument. Pour utiliser cette méthode, vous devez importer le package *java.util.Collections* en haut du fichier.

```
import java.util.ArrayList;
import java.util.Collections;
 public class Test_Tri {
    public static void main(String args[]) {
      ArrayList< Integer> liste = new ArrayList< Integer>();
      liste.add(4);
      liste.add(5);
      liste.add(2);
      System.out.println("Liste non triée : " + liste);
      Collections.sort(liste); // tri par ordre croissant
      System.out.println("liste triée par ordre croissant: " + liste);
       Collections.reverse(liste); // pour inverser le tri
      System.out.println("liste triée par ordre décroissant: " + liste);
   }
}
```



La classe java *Vector* implémente un tableau d'objets similaire à *ArrayList*. Les éléments de Vector sont accessibles à travers d'un indice entier. La taille d'un *Vector* est variable selon le besoin (ajouter ou supprimer des éléments) après la création.

Vector est synchronisée ce qui veut dire qu'il permet plusieurs opérations au même temps toutefois, il ne garantit pas de bonnes performances lors d'une utilisation multi-threading. Il est recommandé d'utiliser *ArrayList* qui donne de bonnes performances si vous n'aurez pas besoin de synchroniser la liste.

Vector implémente l'interface List comme *ArrayList*, son inconvénient est qu'il donne de faible performance comme on l'a mentionné à cause de sa synchronisation dans les opération d'ajout, recherche, suppression et modification de ces éléments.

Comme dans *ArrayList* et d'autres collections d'objets, *Vector* implémente l'interface

List et l'interface Collections dont il hérite de nombreuses méthodes d'ajout, suppression, recherche, modification.

Le tableau suivant résumé les principales méthodes de la classe *Vector*

Informations sur l'état du vecteur	
isEmpty()	indique si le Vector est vide
size()	renvoie le nombre d'éléments
elementAt(int)	obtient un objet à partir de son indice
Opérations qui modifient l'état du vecteur	
addElement(Object)	ajoute un objet à la fin du Vector
insertElementAt(Object, int)	insère l'objet à l'indice indiqué
removeElementAt(int)	retire l'objet dont l'indice est donné
removeAllElements()	vide le Vector
setElementAt(Object, int)	place l'objet à l'indice donné

3. Chaînes de caractères

Les chaînes de caractères ne sont pas considérées en Java comme un type primitif ou comme un tableau. On utilise une classe particulière, nommée String, fournie dans le package java.lang. On peut utiliser l'opérateur « + » pour concaténer deux chaînes de caractères :

```
String s1 = ”hello” ;
String s2 = ”world” ;     Après ces instructions s3 vaut ”hello
String s3 = s1 + ” ” +    world”
s2 ;
```

L'initialisation d'une chaîne de caractères s'écrit :

```
String s = new String(); // pour une chaine vide
String s2 = new String(”hello world”); // pour une chaîne de valeur ”hello world”
```

- Création d'une chaîne de caractères à partir d'un tableau de caractères :

```
char[] s = { 'I', 'N', 'F', '1', '5', '6', '3' };
String sigle = new String(s);
```

- Création d'une chaîne de caractères vide :

```
String vide = new String();  // équivalent à String vide = "";
```

- Création d'une copie d'une chaîne de caractères :

```
String sigle2 = new String(sigle);
```

Il est possible de convertir n'importe quel type primitif en une chaine de caractères en utilisant la méthode de classe *valueOf* de la classe *String*. Lorsqu'on fait une concaténation qui fait intervenir des types primitifs, c'est ce qui se passe de manière implicite ; on aurait donc pu s'écrire :

```
String age = String.valueOf (18);
System.out.println ("J'ai " + age + " ans");
```

Pour convertir un objet en une chaine de caractères, plus précisément pour obtenir une représentation d'un objet sous forme de chaine de caractères, on utilise la méthode *toString* qui est applicable sur tous les objets. L'exemple suivant montre l'utilisation de cette méthode sur un objet de type *Date* :

```
Date date = new Date (2008 - 1900, 7, 1);
String s = date.toString();
System.out.println (s);
```

En fait, lorsque vous utilisez *System.out.print* et *System.out.println* avec un objet en paramètre, la méthode *toString* est implicitement appelée. Par exemple, les deux instructions suivantes sont tout à fait équivalentes :

```
System.out.println (date);
System.out.println (date.toString());
```

Voici des Séquences d'échappement à utiliser dans les chaines de caractères :

Séquence	Signification
\b	Backspace
\t	Tabulation
\n	Nouvelle ligne
\r	Retour chariot
\"	Guillemet double
\'	Guillemet simple
\\	Backslash

Quelques méthodes usuelles de la classe String

Méthode	Description
`char charAt (int index)`	*Renvoie le caractère en position `index` (le premier caractère est en position 0)*
`String concat (String str)`	*Concatène l'objet cible avec `str`*
`boolean endsWith (String str)`	*Teste si l'objet cible se termine par `str`*
`boolean equalsIgnoreCase (String str)`	*Compare l'objet cible avec `str` sans tenir compte de la casse*
`int indexOf (String str)`	*Renvoie la position de la première occurrence de `str` si l'objet cible contient la chaine `str`, sinon renvoie -1. Des variantes de cette méthode existent.*
`int length()`	*Renvoie le nombre de caractères de l'objet cible*
`boolean startsWith (String str)`	*Teste si l'objet cible commence par `str`*
`String substring (int begin, int end)`	*Extrait la sous-chaine de l'objet cible compris entre les positions `begin` et `end`*
`String toLowerCase()`	*Renvoie la chaine correspondant à l'objet cible dont toutes les lettres ont été changées en minuscules*
`String toUpperCase()`	*Renvoie la chaine correspondant à l'objet cible dont toutes les lettres ont été changées en majuscules*

4. Structures conditionnelles

Les mots-clés java correspondant aux structures conditionnelles sont if, else, else if et switch.

1.1. *L'instruction if* : permet d'exécuter une série d'instructions si une condition est réalisée. Sa syntaxe est : *if (condition réalisée) {liste d'instructions}*

- La condition doit être entre des parenthèses
- Il est possible de définir plusieurs conditions à remplir avec les opérateurs ET et OU (&& et ||) par exemple l'instruction suivante teste si les deux conditions sont vraies :

if ((condition1)&&(condition2))

L'instruction suivante exécutera les instructions si l'une ou l'autre des deux conditions est vraie : *if ((condition1)||(condition2))*

- S'il n'y a qu'une instruction, les accolades ne sont pas nécessaires...
- Les instructions situées dans le bloc qui suit ***if*** sont les instructions qui seront exécutées si la ou les conditions ne sont pas remplies.

1.2. *L'instruction if ... else*

L'expression *if ... else* permet d'exécuter une autre série d'instruction en cas de non-réalisation de la condition. Sa syntaxe est : *if (condition réalisée) {liste d'instructions }*

else {autre série d'instructions }

```
public class Conditionnelle1
{
 public static void main (String [] args)
   {    int a = 21, b = 22;
     if(a < b)  {System.out.println("a est moins grand que b");}
     else  {System.out.println("a est plus grand que b");}
 }
 }
```

1.3. *L'instruction switch*

Elle permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Ce branchement conditionnel simplifie beaucoup le test de plusieurs valeurs d'une variable, car cette opération aurait été compliquée (mais possible) avec des *if* imbriqués.

Sa syntaxe est la suivante :

```
switch (Variable) {
case Valeur1 :
        Liste d'instructions
        break;
case Valeur2 :
        Liste d'instructions
        break;
case Valeurs... :
        Liste d'instructions
        break;
default:
        Liste d'instructions
        break;
}
```

2. Structures répétitives :

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée.

2.1. *La boucle for*

Dans la syntaxe de for, il suffit de préciser le nom de la variable qui sert de compteur (et éventuellement sa valeur de départ, la condition sur la variable pour laquelle la boucle s'arrête et enfin une instruction qui incrémente (ou décrémente) le compteur.

La syntaxe de cette expression est la suivante :

```
for (compteur; condition; modification du compteur)
        { Liste d'instructions }
```

Exemple:

```
for (int i=1; i<6; i++) {System.out.println((char)i);}
```

Cette boucle affiche 5 fois la valeur de *i*, c'est-à-dire 1, 2, 3, 4,5. Elle commence à i=1, vérifie que i est bien inférieur à 6, etc... jusqu'à atteindre la valeur i=6, pour laquelle la condition ne sera plus réalisée, la boucle s'interrompra et le programme continuera son cours. D'autre part, Java autorise la déclaration de la variable de boucle dans l'instruction *for* elle-même.

2.2. L'instruction while

L'exécution de cette instruction suit les étapes suivantes :

1. La condition (qui doit renvoyer une valeur booléenne) est évaluée. Si celle-ci est vraie on passe à l'étape 2, sinon on passe à l'étape 4 ;
2. Le bloc est exécuté ;
3. Retour à l'étape 1 ;
4. La boucle est terminée et le programme continue son exécution en interprétant les instructions suivant le bloc.

Syntaxe : *while (<condition>) <bloc>*

Exemple : *while (a != b) a++;*

2.3. La boucle do..while

L'exécution de cette instruction suit les étapes suivantes :

1. Le bloc est exécuté ;
2. La condition (qui doit renvoyer une valeur booléenne) est évaluée. Si celle-ci est vraie on retourne à l'étape 1, sinon on passe à l'étape 3 ;
3. La boucle est terminée et le programme continue son exécution en interprétant les instruction suivant le bloc.

Syntaxe : *do <bloc> while (<condition>);*

Exemple : ***do*** *a++* ***while*** *(a != b);*

3. Les packages

Un package en Java est utilisé pour regrouper des classes apparentées (Similaire à un dossier dans un répertoire de fichiers). Nous utilisons les packages pour éviter les conflits de noms, et pour écrire un code plus facile à maintenir. Les packages sont divisés en deux catégories :

- Packages intégrés ou prédéfinis (provenant de l'API Java)
- Packages définis par l'utilisateur (créez vos propres packages)

6.1. Packages intégrés

L'API Java est une bibliothèque de classes pré-écrites, dont l'utilisation est gratuite, incluse dans l'environnement de développement Java. Cette bibliothèque contient des composants pour la gestion des entrées, la programmation des bases de données et bien d'autres choses encore.

La liste complète peut être consultée sur le site Web d'Oracle **:** ***https://docs.oracle.com/javase/8/docs/api/***.

La bibliothèque est divisée en paquets et en classes. Cela signifie que vous pouvez soit importer une seule classe (avec ses méthodes et ses attributs), soit un paquet entier qui contient toutes les classes appartenant au paquet spécifié.

Pour utiliser une classe ou un paquet de la bibliothèque, vous devez utiliser le mot-clé **import** :

```
import package.name.Class ; // Importer une seule classe

import package.name.* ; // Importer le package entier
```

Voici les packages de base :

• java.lang : contient les classes de base (chaînes de caractères, mathématiques, tâches, exceptions ...)

• java.util : contient des classes comme vecteurs, piles, files, tables ...

• java.io : contient les classes liées aux entrées/sorties texte et fichier

• java.awt : contient les classes pour les interfaces (fenêtres, boutons, menus, graphique, événements ...)

• javax.swing : contient les classes pour les interfaces (évolution de awt)

• java.net : contient les classes relatives à internet (sockets, URL ...)

• java.applet : contient les classes permettant de créer des applications contenues dans des pages en HTML.

Pour pouvoir utiliser les classes de ces bibliothèques il faut y faire référence en début de fichier en utilisant la directive **import** suivie du nom de la classe de la bibliothèque ou de * pour accéder à toutes les classes de cette bibliothèque, par exemple : **import java.io.***

Certaines de ces bibliothèques contiennent des sous-bibliothèques qu'il faut explicitement nommer (par exemple java.awt.events.* pour les événements).

Remarque : les classes de la bibliothèque de base java.lang n'ont pas besoin de la commande import

6.2. Packages non intégrés

Pour créer un package, il suffit d'ajouter la ligne : *package <nom>* ; au début du fichier.

```
package mypackage;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
}
```

4. **Quelques méthodes usuelles :**

- *La classe java.lang.Math*

La classe java.lang.Math contient une série de méthodes et variables mathématiques. Comme la classe Math fait partie du package java.lang, elle est automatiquement importée. De plus, il n'est pas nécessaire de déclarer un objet de type Math car les méthodes sont toutes static.

Exemple 1 *: La méthode sqrt(double) :*

Cette méthode calcule la racine carrée d'un nombre (de type double)

```
public class Math1 {
public static void main(java.lang.String[] args) {
System.out.println(" = " + Math.sqrt(3.0));
}
}
```

Exemple 2 : *La méthode abs(x)*

Cette méthode donne la valeur absolue de x (les nombres négatifs sont convertis en leur opposé). La méthode est définie pour les types int, long, float et double.

```
public class Math1 {
public static void main(String[] args) {
System.out.println(" abs(-5.7) = "+Math.abs(-5.7));
}
}
```

Exemple 3 : *La méthode* pow(double, double)

Cette méthode élève le premier argument à la puissance indiquée par le second.

```
public class Math3 {
public static void main(String[] args) {
System.out.println(" 5 au cube = "+Math.pow(5.0, 3.0) );
}
 }
```

- *La classe Scanner*

La classe **Scanner** contient des méthodes permettant d'extraire des valeurs d'un périphérique d'entrée. Chaque valeur extraite est un jeton, qui est un ensemble de caractères séparé du prochain ensemble par des espaces. Le plus souvent, cela signifie que les données sont acceptées lorsqu'un utilisateur appuie sur la touche Entrée, mais cela peut également signifier qu'un jeton est accepté après un espace ou une tabulation.

Scanner clavier = new Scanner(System.in);

Le tableau suivant récapitule certaines des méthodes les plus utiles pour lire différents types de données à partir du périphérique d'entrée par défaut.

Méthode	Description
nextDouble()	Récupère l'entrée en tant que Double
nextInt()	Récupère l'entrée en tant que int
nextLine()	Récupère la ligne de données suivante et la renvoie sous forme de chaîne (String)
next()	Récupère le prochain jeton complet sous forme de chaîne
nextShort()	Récupère l'entrée en tant que Short
nextByte()	Récupère l'entrée en tant que octet (Byte)

Méthode	Description
nextFloat()	Récupère les entrées sous forme de float. Notez que lorsque vous entrez une valeur d'entrée qui sera stockée sous forme de float, vous ne tapez pas de F. Le F est utilisé uniquement avec les constantes codées dans un programme.
nextLong()	Récupère l'entrée en tant que long. Notez que lorsque vous entrez une valeur d'entrée qui sera stockée longtemps, vous ne tapez pas un L. Le L est utilisé uniquement avec des constantes codées dans un programme.

La classe Scanner ne contient pas de méthode nextChar(). Pour extraire un seul caractère du clavier, vous pouvez utiliser la méthode nextLine(), puis la méthode charAt(int).

Voilà un exemple récapitulatif illustrant l'utilisation du package Scanner. Il demande à l'utilisateur d'introduire son nom, son âge et son salaire et il affiche sur la même ligne ces informations comme présentée sur l'image.

```
import java.util.Scanner; // importer le package
 public class Test {
   public static void main(String args[]) {
       String nom;
       int age;
      double salaire;
      Scanner clavier = new Scanner(System.in);
      System.out.print("Saisir votre nom : ");
      nom = clavier.nextLine();// saisie d'une chaine de caractères
      System.out.print("Saisir votre age : ");
      age = clavier.nextInt(); // saisie d'un entier
      System.out.print("Saisir votre salaire : ");
      salaire = clavier.nextDouble(); // saisie d'un nombre décimal
      System.out.println("[nom = " + nom + " , age = " + age + " , salaire =" + salaire + "]");
      // fermer les ressources
      clavier.close();
    }
}
```

L'environnement NetBeans

Cette patrie présente le minimum vital pour une utilisation de l'environnement de développement intégré (IDE) NetBeans. Il s'agit d'un premier jet, il sera ultérieurement complété. Les captures d'écran ont été réalisées avec la version 8.2 de NetBeans sous Windows 10.

1. Configuration de l'environnement de travail :

Les TP(s) de programmation orientée objet supposent que votre environnement de travail est configuré pour Java 7 et supérieur (Java 8 est fortement recommandé).

Pour vérifier les versions de votre JDK et de votre JRE, tapez les commandes suivantes à l'invite javac –version et java -version. Pour installer le JDK sur un PC personnel, il vous faudra le télécharger depuis le site d'Oracle pour Windows ou installer le package OpenJDK pour un système Linux.

Vous êtes libres d'utiliser votre outil préféré pour l'édition de votre code source. Néanmoins il est plus pratique d'utiliser un outil de développement intégré (IDE). Les IDE les plus connus pour le développement JAVA sont Netbeans, Eclipse et IntelliJ.

Dans le cadre de nos TP, nous utiliserons Netbeans 8.2. et nous allons présenter un bref aperçu de cet environnement.

2. Organisation du travail :

Vous enregistrerez les différents fichiers que vous créerez pour résoudre les exercices, dans un répertoire java/tp (ou poo/tp). L'emplacement exact du répertoire java est à votre convenance.

Chaque TP aura son propre répertoire et, sauf exceptions, chaque exercice aura aussi son propre répertoire, ce qui vous permettra de conserver différentes versions d'une même classe. Les exceptions sont les cas où, par exemple, on ajoute des méthodes ou des variables à une classe sans toucher ce qui a déjà été écrit dans les exercices précédents (mais même dans ces cas, il est conseillé d'avoir des répertoires

différents). Vous aurez ainsi à créer, par exemple, un fichier java/tp/tp1/exo1/HelloWorld.java.

Si vous n'avez pas terminé un TP, vous devez le terminer seul (ou avec l'aide de vos camarades) pour le prochain TP. Vous pouvez demander de l'aide en envoyant un message électronique à un de vos enseignants si un point précis n'est trouvé par aucun étudiant.

3. Notion de projet NetBeans

Avec NetBeans[1] vous travaillez toujours au sein d'un projet. Un projet est un ensemble de fichiers source JAVA et des informations associées (classpath, comment construire et exécuter le projet, etc...). L'IDE NetBeans stocke l'information associée à un projet dans un dossier projet (project folder) qui inclus un script build Ant (fichier `build.xml` équivalent d'un `Makefile`, Ant est un projet open source de la fondation Apache[2] qui contrôle la compilation et l'exécution, et un répertoire `nbproject` qui contient différents fichiers de configuration de votre projet.

Avec NetBeans vous pouvez soit

- Créer des projets standards qui s'appuient sur des scripts « Ant » générés par NetBeans et sur une organisation standard des fichiers du projet.
- Créer vos projets avec vos propres scripts et votre propre organisation de fichiers.

Dans cette introduction à NetBeans nous ne nous intéresserons qu'aux projets Standards, et particulièrement aux projets de type Application Java.

4. Création d'une application Java avec NetBeans

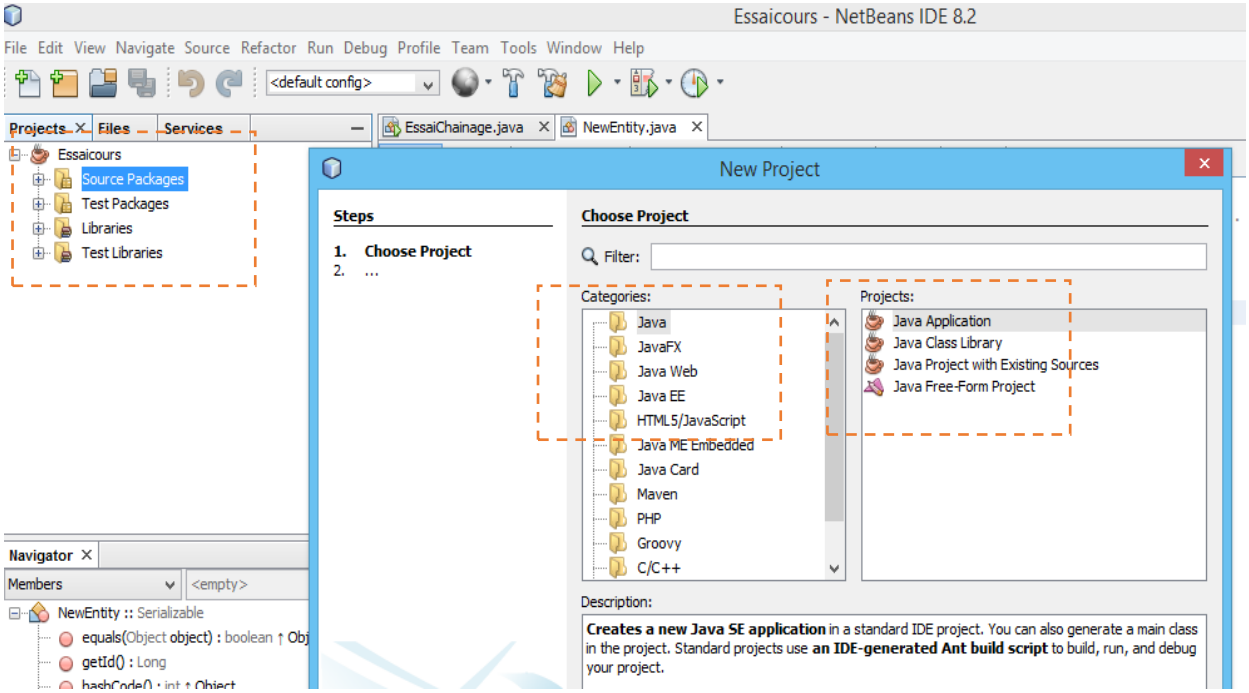
Pour créer un nouveau projet d'application JAVA :

1. Choisissez **File > New Project** (Maj-Ctrl-N). Vous pouvez aussi faire directement un clic droit dans la vue **Projects**

[1] http://www.netbeans.org

[2] http://ant.apache.org

2. Dans la boite de dialogue pour la création de nouveaux projets qui apparaît sélectionnez Java sous la rubrique Categories,
3. Sous la rubrique Projects, sélectionnez Java Application
4. Cliquez sur Next.

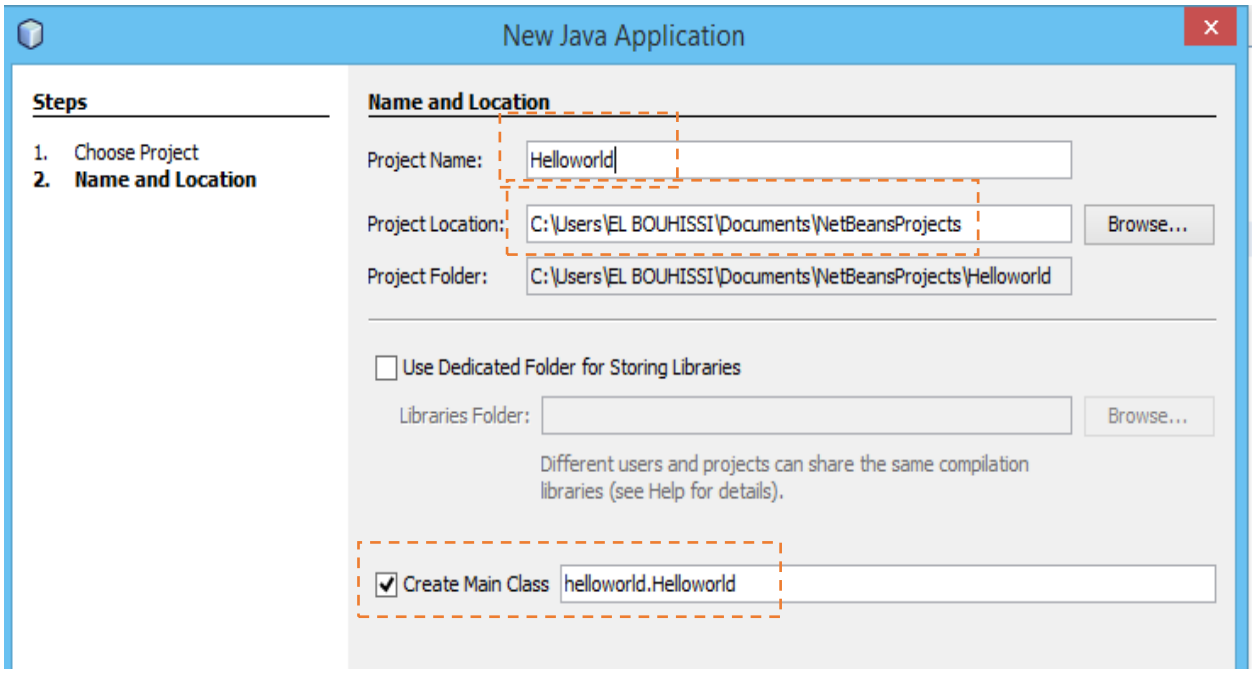


Une boite de dialogue pour la création d'une nouvelle application Java (JSE) est alors affichée.

1. Sous Project Name, introduisez HelloWorld.
2. Avec Project Location choisissez un emplacement pour les fichiers de votre projet.
3. Demandez à Netbeans de créer automatique une classe contenant un programme principal (méthode public static void main(String[] args)).

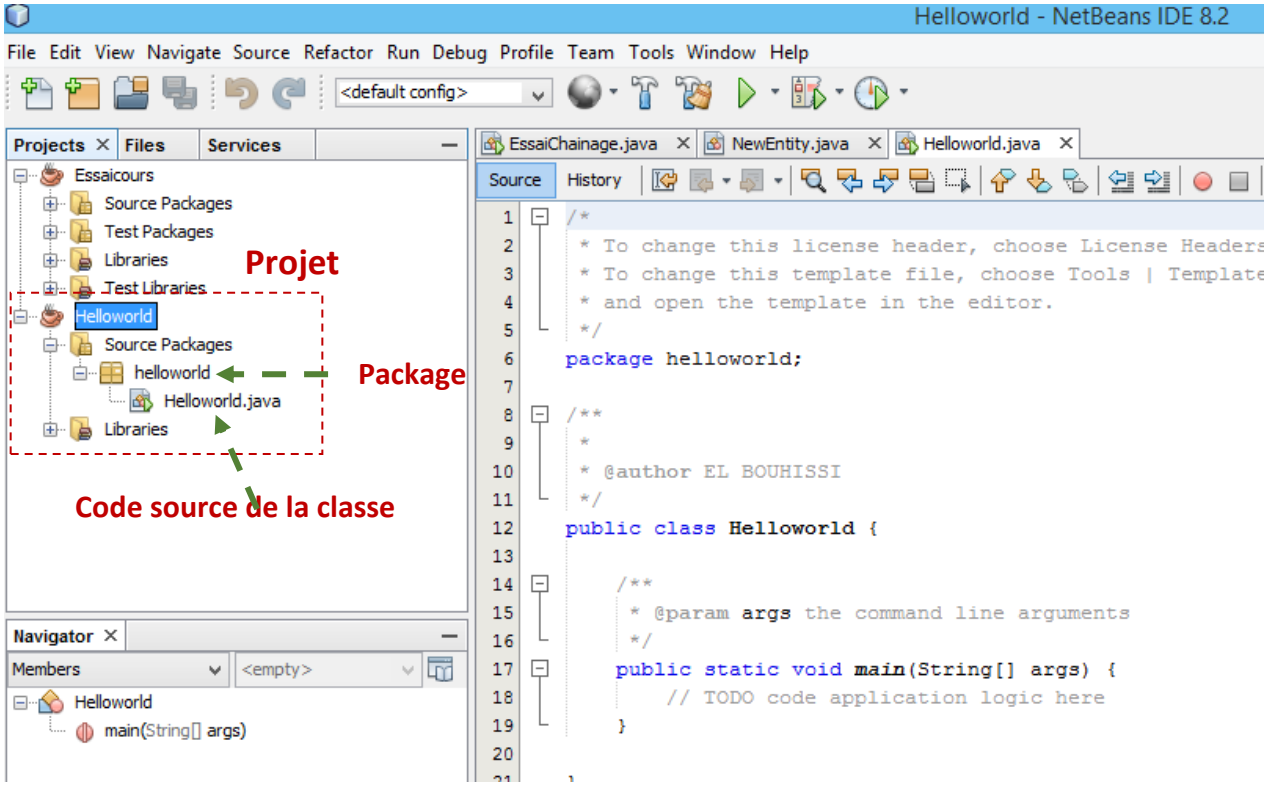
Par défaut Netbeans vous propose de créer un classe HelloWorld (de même nom que votre projet) dans un package helloworld (même nom que le project mais en minuscules).

4. Cliquez sur Finish.



NetBeans crée un dossier `HelloWorld` dans le répertoire que vous avez choisi pour votre projet (dans l'exemple ci-dessus `C:\Users\EL BOUHISSI\Documents\NetBeansProjects`.

Le projet `HelloWorld` s'ouvre et la classe `HelloWorld.java` est affichée dans l'éditeur de sources.



L'onglet `Projects` permet de visualiser la structure logique du projet (les différents packages Java qui le constituent), l'onglet `Files` permet de visualiser sa structure de fichiers.

5. Utilisation de l'éditeur

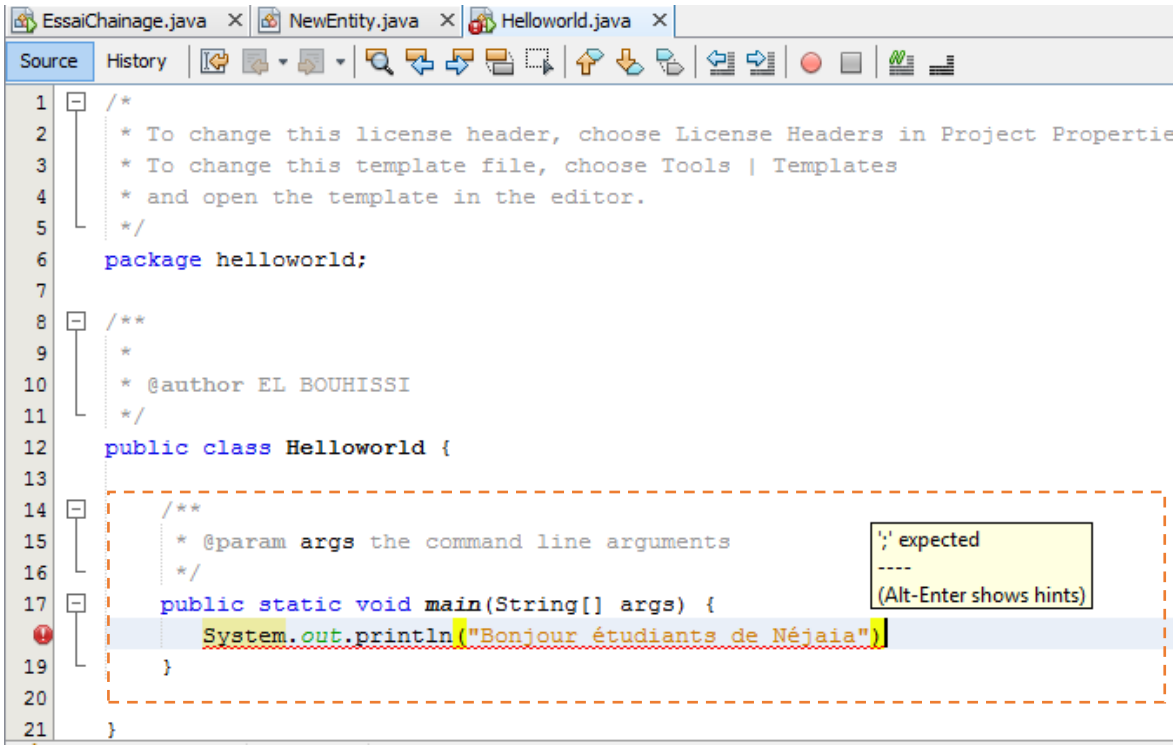
Modifiez le programme `HelloWorld` en complétant la méthode main avec l'instruction :

System.out.println("Bonjour le Monde");

Lorsqu'il le peut, l'éditeur vous propose une complétion automatique du code. Par exemple arrêtez la frappe après avoir tapé `System.out.` , l'éditeur vous propose alors toutes les méthodes possibles avec la documentation javadoc associée.

L'éditeur offre la complétion de code pour toutes les classes et méthodes dans le `classpath` de compilation du projet. Vous pouvez à tout moment activer la complétion par `Ctrl-Espace`. L'éditeur n'attend pas toujours qu'une compilation ait été lancée pour vous signaler d'éventuelles erreurs. Au fur et à mesure que vous tapez votre texte, la syntaxe java est vérifiée et en cas d'erreur un petit point d'exclamation rouge est positionné en face de la ligne correspondante.

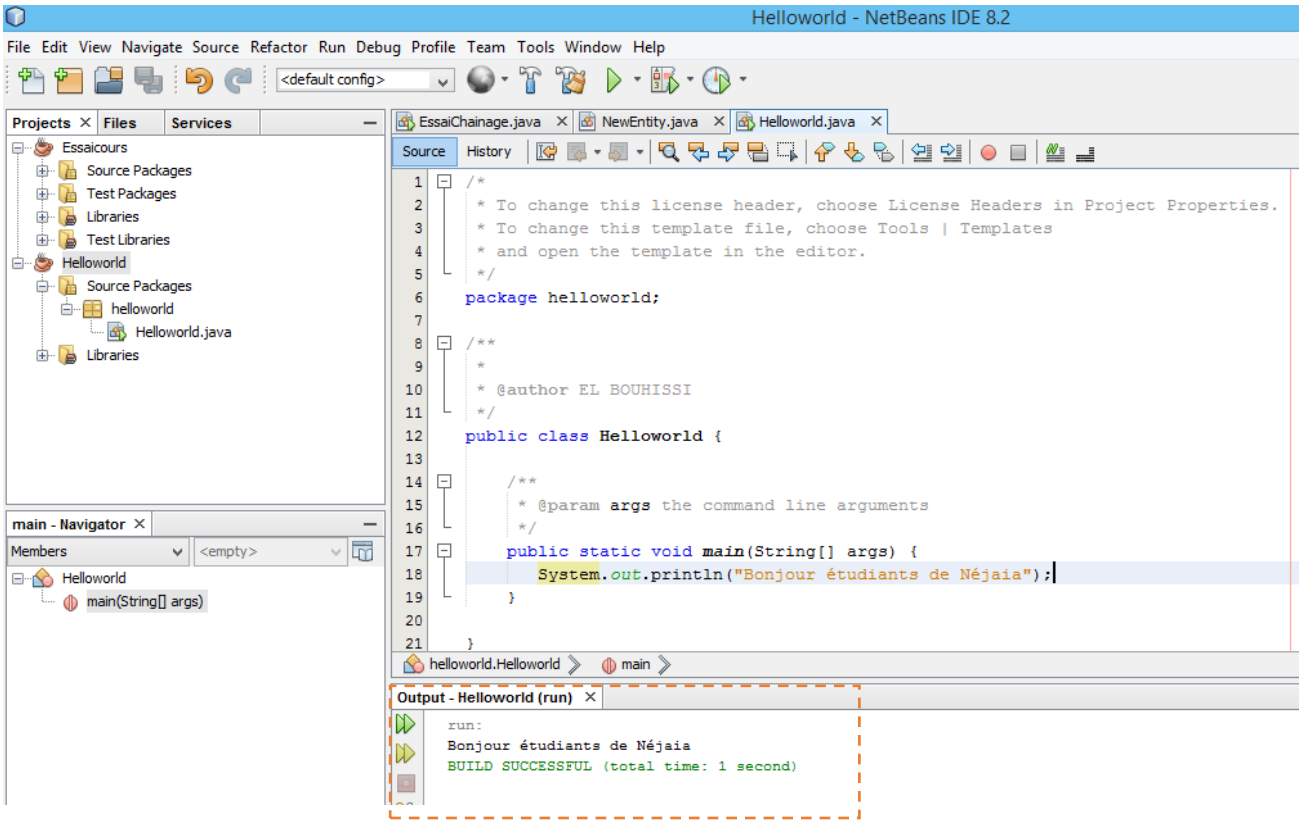
En positionnant le curseur de la souris sur celui-ci vous aurez un message d'explication indiquant la nature de l'erreur. Pour apprécier cette fonctionnalité ne mettez pas de `;` à la fin de votre instruction `System.out.println`.



Corrigez l'erreur de syntaxe et sauvegardez le fichier `HelloWorld.java` (`File->Save` ou `Ctrl-S`). Si le fichier a été modifié et n'a pas été sauvegardé son nom apparait en gras dans l'onglet correspondant.

6. Compiler et exécuter le projet

Une fois le fichier sauvegardé vous pouvez lancer la compilation et l'exécution de votre projet par `Menu Run-> Project (F6)` ou en cliquant sur le bouton  de la barre d'outils. Cela a pour effet d'exécuter les différentes commandes du script *Ant* `build.xml` pour compiler puis exécuter votre application. Les sorties de ces commandes *Ant* puis de votre application sont affichées sur la fenêtre `Output` en bas de la fenêtre de `NetBeans`.



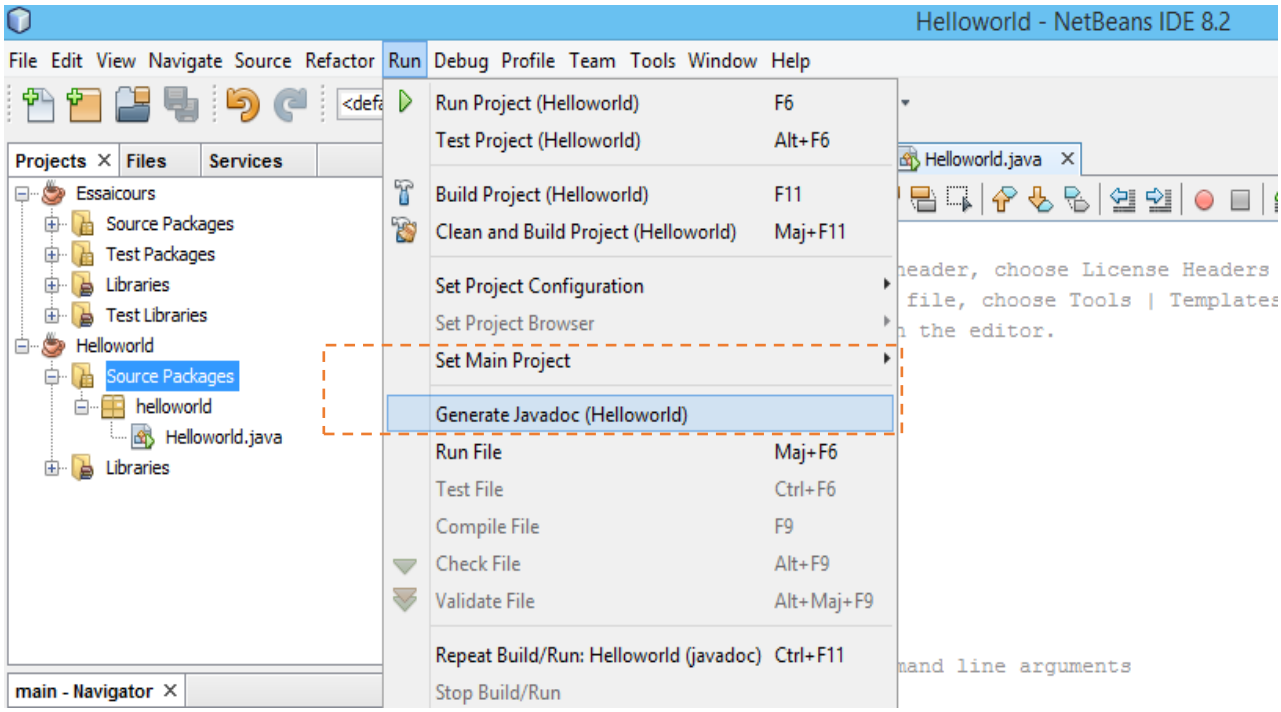
7. Interrompre l'exécution d'un programme

Fermez le projet `HelloWorld` (clic droit dans la fenêtre projet et item `Close Project` du menu contextuel).

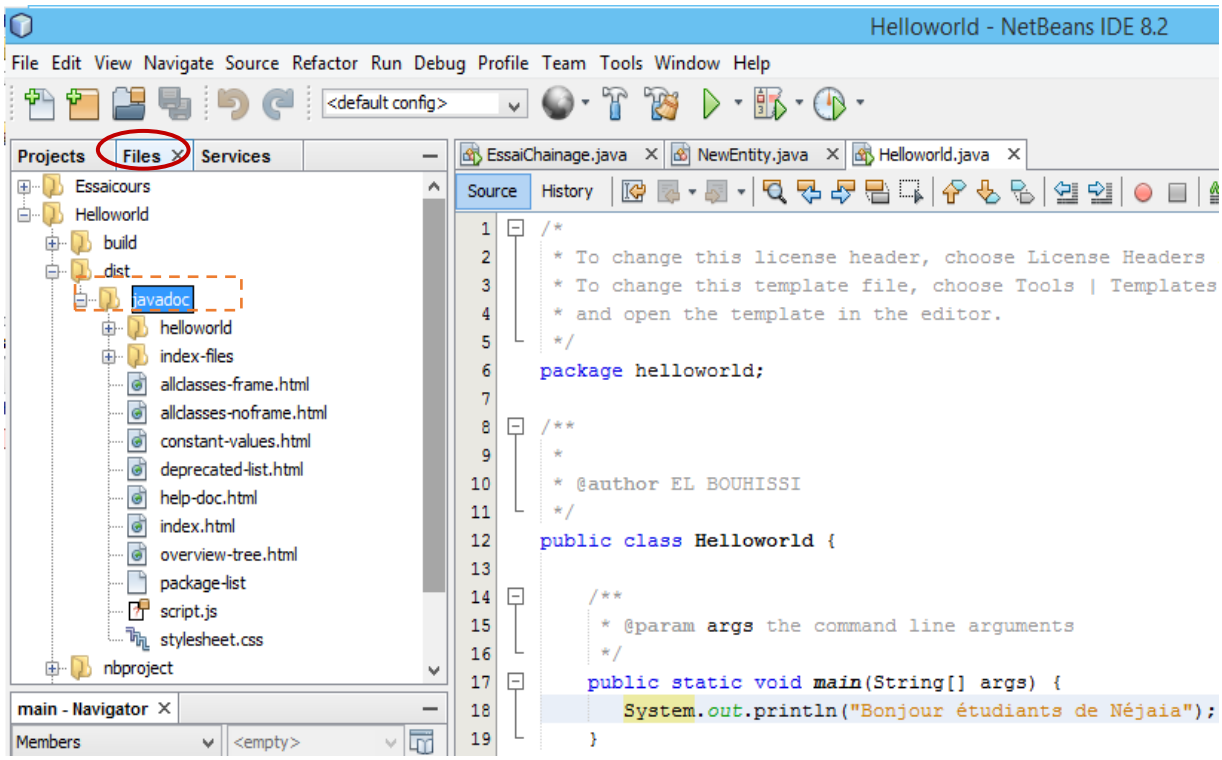
Lorsqu'il est lancé depuis une console, ce programme peut être arrêté en tapant `CTRL-C`. Le bouton situé à gauche de la fenêtre console vous permet d'interrompre l'exécution d'un programme depuis *NetBeans*.

8. Générer la documentation Javadoc d'un projet

Pour générer la documentation javadoc d'un projet à partir des sources il suffit d'activer la commande `Generate Javadoc` depuis le menu `Run`.



Une fois la documentation générée *NetBeans* ouvre une fenêtre du navigateur par défaut affichant cette documentation. Les fichiers générés le sont dans le répertoire `dist/javadoc` situé sur la racine de votre projet. Pour les voir, cliquez sur l'onglet `Files` dans l'explorateur du projet et développez le noeud `dist`.



9. La documentation en ligne de NetBeans

Nous avons vu dans ce qui précède le strict minimum pour l'utilisation de NetBeans dans l'écriture d'applications Java. Pour profiter aux mieux des très riches fonctionnalités de cet environnement, nous vous conseillons de parcourir la documentation en ligne.

Pour obtenir cette aide, menu : `Help--> Help Contents`.

Travaux pratiques

TP 1 : Initiation à la programmation Orientée Objet

Objectifs :

- Savoir écrire des petits programmes en Java
- Gérer les entrées/sorties

Activité 01 :

Ecrivez le code source d'une classe HelloWorld dans le fichier HelloWorld.java. La classe HelloWorld contient un attribut privé message qui est du type String. HelloWorld doit également contenir deux méthodes : un constructeur qui attribue une valeur au message et une méthode public String getMessage() qui renvoie la valeur du message. Compilez cette classe.

La classe HelloWorld n'est pas exécutable dans son état actuel. Une classe exécutable est une classe qui contient une méthode spécifique (main) utilisée comme point de départ de l'exécution. La méthode main d'une classe s'écrit de la manière suivante :

```
public static void main(String[] args){
   //début du code à exécuter
}
```

Ajoutez une méthode main à la classe HelloWorld. Dans cette méthode créez une instance de HelloWorld et affichez son message. Pour afficher du texte à l'écran utilisez la méthode System.out.println(String t). Compilez comme précédemment.

Créez un package elearning.univ-bejaia.dz (donc un répertoire elearning/univ-bejaia/dz) et mettez votre fichier HelloWorld.java. Déclarez que la classe HelloWorld appartient au package elearning.univ-bejaia.dz.

Modifiez le programme HelloWorld.java de manière à ce qu'il provoque l'affichage suivant :

BONJOUR

CECI EST MON

PREMIER PROGRAMME JAVA

Créez le fichier Hello2.java (dans le même répertoire) :

```
public class Hello2 {
   public static void main(String[] args) {
      if (args.length == 0)
          System.out.println("Bonjour inconnu(e)");
      else
        for (int i = 0; i < args.length; i++)
           System.out.println("Bonjour " +  args[i]);
}
}
```

Compilez ce programme et exécutez-le successivement avec les commandes :

- java Hello2
- java Hello2 Mahmoud
- java Hello2 Lylia Mahmoud Meriem

En observant le code source du programme Hello2.java, que pouvez-vous en déduire sur le rôle du paramètre args de la méthode main ?

Activité 02 : *Gestion simple des entrées/sorties*

On a vu que la méthode **System.out.println** permet d'afficher une chaîne de caractère à l'écran puis de sauter une ligne. De même la méthode **System.out.print** permet d'afficher une chaîne de caractère sans saut de ligne. Enfin une méthode **System.out.printf** permet d'écrire une chaîne formatée sur la sortie standard, le comportement est analogue à la fonction **printf** du langage C.

La classe **Scanner** est une classe utilitaire qui facilite grandement la lecture sur l'entrée standard. Elle dispose de méthode de lecture pour tous les types primitifs et les chaînes de caractères. Cette classe est simple d'usage comme le montre l'exemple suivant :

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

Pour rendre la classe accessible, il faut ajouter la ligne suivante en tout début de fichier.

```
import java.util.Scanner;
```

1. A l'aide des méthodes d'affichage et de la classe **Scanner**, définissez une classe **TestMoyenne**. Cette classe dispose d'une méthode **main** qui demande à l'utilisateur de rentrer un certain nombre de notes puis affiche la moyenne des notes rentrées.
2. La classe **Scanner** contient une méthode **hasNextInt** sans paramètre qui renvoie **true** si la prochaine entrée lue est bien un nombre entier et **false** sinon. De même, il existe une méthode **next** qui lit la prochaine chaîne de caractères sur l'entrée standard. Utilisez ces méthodes pour vérifier la cohérence des entrées de l'utilisateur (il ne rentre que des entiers, le nombre de note est positif, chaque note est comprise entre 0 et 20).

Pour effectuer des lectures au clavier de valeurs, inspirez-vous de l'exemple suivant :

```
import java.util.Scanner ; // importer le package
public class Lectures {
public static void main(String[] args) {
Scanner clavier = new Scanner(System.in);
int i; float x;
System.out.print("donnez un nombre entier ");
i = clavier.nextInt();
System.out.println("valeur lue: " + i);
System.out.print("donnez un nombre flottant ");
x = clavier.nextFloat();
System.out.println("valeur lue: " + x);
}
}
```

Vous pouvez remplacer aussi la 3ème ligne du programme par : **`java.util.Scanner clavier = new java.util.Scanner(System.in) ;`** en supprimant le package **`import java.util.Scanner ;`**

<u>Activité 03</u> :

Ecrire un programme qui demande à l'utilisateur d'introduire son nom, son prénom ainsi que son âge et il les affiche par la suite. Par exemple, si l'utilisateur introduit les valeurs suivantes :

Nom : BACHA
Prénom : Mahmoud
Age : 9

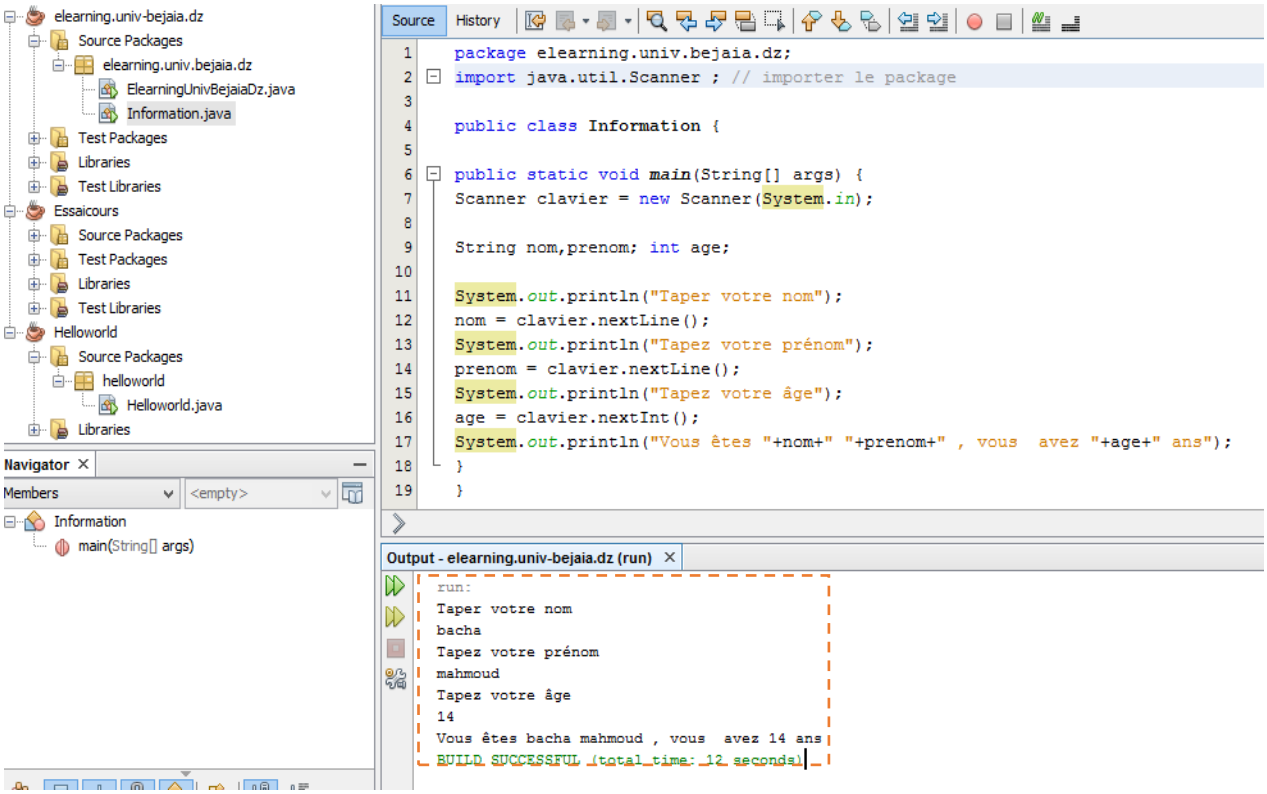
L'ordinateur affiche :
Vous êtes BACHA Mahmoud et vous avez 9 ans.

Corrigé-type :

Voici le code du programme :

```
package elearning.univ.bejaia.dz;
import java.util.Scanner ; // importer le package de lecture
public class Information {
public static void main(String[] args) { // méthode main pour l'exécution
Scanner clavier = new Scanner(System.in);
String nom,prenom; int age;
System.out.println("Taper votre nom");
nom = clavier.nextLine();// lecture d'une chaine de caractère
System.out.println("Tapez votre prénom");
prenom = clavier.nextLine();
System.out.println("Tapez votre âge");
age = clavier.nextInt();// lecture d'un entier
System.out.println("Vous êtes "+nom+" "+prenom+" , vous avez "+age+" ans");
}
}
```

La figure suivante présente l'exécution sur machine :



Activité 04 :

10. Écrire une classe Nombre ayant pour attribut un nombre qui sera initialisé (nombre aléatoire de 0 à 9) par le constructeur de la classe.

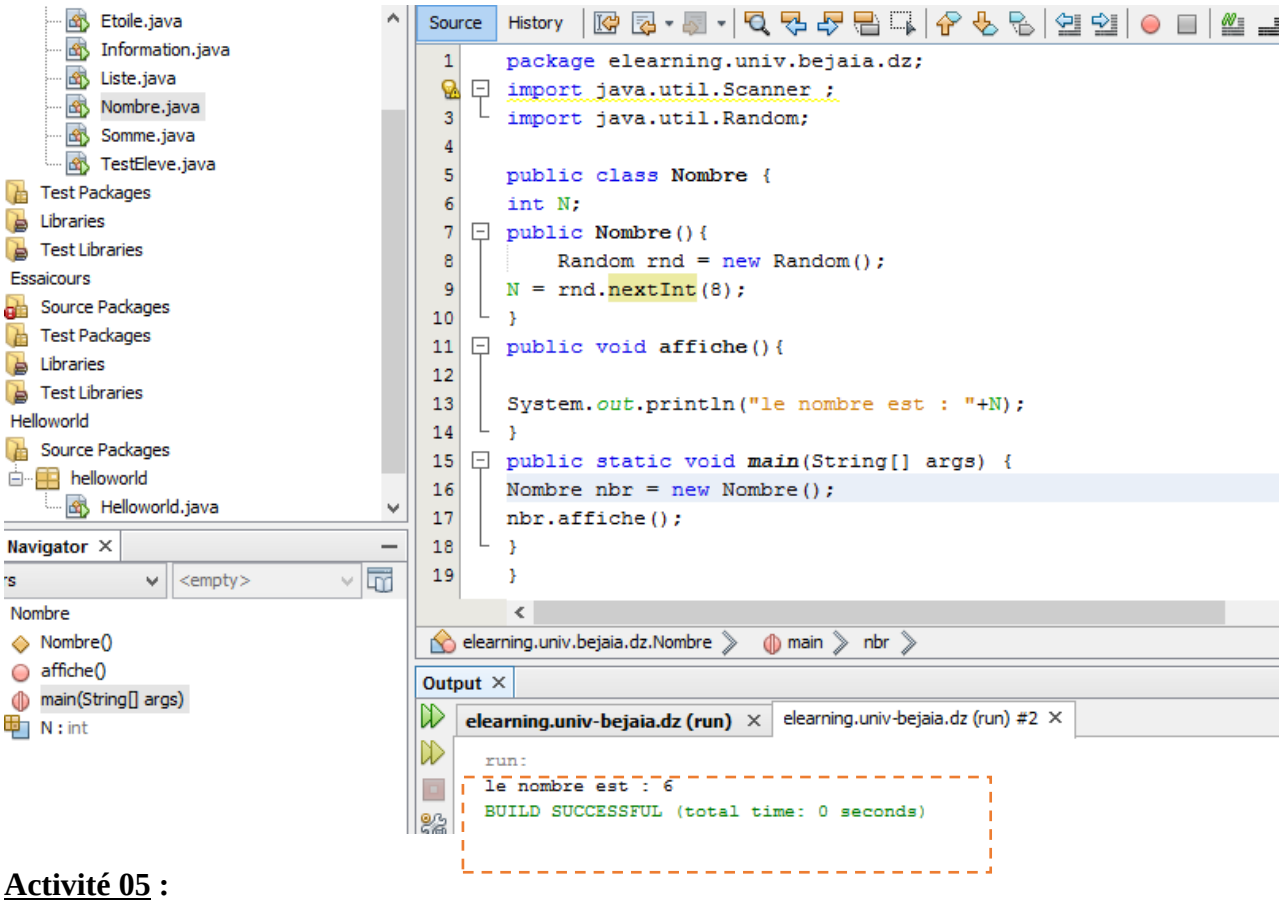
11. Une méthode afficher() permettra d'afficher le nombre.
12. Écrire une autre classe contenant la méthode main() pour tester la classe Nombre.

Corrigé-Type :

```
package elearning.univ.bejaia.dz;
import java.util.Scanner ;
import java.util.Random;
public class Nombre {
int N;
public Nombre(){
    Random rnd = new Random();
N = rnd.nextInt(8);
}
public void affiche(){

System.out.println("le nombre est : "+N);
}
public static void main(String[] args) {
Nombre nbr = new Nombre();
nbr.affiche();
}
}
```

Ce programme peut être résolu autrement avec la classe Math, la figure suivante montre Et l'exécution sur machine :



Activité 05 :

Voici le code source de la classe Livre :

```
public class Livre {
  // Variables
  private String titre, auteur;
  private int nbPages
  // Constructeur
  public Livre(String unAuteur, String unTitre) {
    auteur = unAuteur;
    titre = unTitre;
  }
  // Accesseur
  public String getAuteur() {
    return auteur;
  }
  // Modificateur
  public void setNbPages(int n) {
    nbPages = nb;
}
```

- Corrigez quelques petites erreurs et ajoutez une méthode main pour
- Créer 2 livres,
- Faire afficher les auteurs de ces 2 livres.
- Lancez l'exécution de la classe Livre.

Activité 06 :

Soit le programme suivant :

```
public class Birthday {
private Date date;
private String name;
public Birthday(Date date, String name) {
this.date = date;
this.name = name;
}
public String toString() {
return "[ Anniversaire de " + name + " le " + date + "]";
}
public static void main(String arg[]) {
Date d = new Date(20, 3, 2013);
Birthday e = new Birthday(d, "Anatole");
System.out.println(e);
d.setMonth(6);
System.out.println(e);
}
}
```

Qu'affiche ce programme ? Respecte-t-il le principe d'encapsulation des données ? Si non, comment le corriger ?

Activité 07 :

Un élève sera ici modélisé par la classe Eleve d'un paquetage nommé gestionEleves, de la façon suivante.

La classe Eleve possède trois attributs privés :

- Son nom, nommé nom, de type String,
- Un ensemble de notes, nommé listeNotes, qui sont des entiers rangés dans un ArrayList<Integer>
- Une moyenne de type **double**, nommée moyenne, qui doit toujours être égale à la moyenne des notes contenues dans l'attribut listeNotes. Un élève sans aucune note sera considéré comme ayant une moyenne nulle.

La classe Eleve possède un constructeur permettant uniquement d'initialiser le nom de l'élève.

La classe Eleve possède aussi cinq méthodes publiques :

- Un getter pour la moyenne de l'élève c'est-à-dire une méthode d'en-tête **public double** getMoyenne() renvoie la valeur de l'attribut moyenne ;
- Un getter pour le nom de l'élève c'est-à-dire une méthode d'en-tête **public** String getNom() renvoie le nom de l'élève ;

- Un getter pour la liste des notes de l'élève c'est-à-dire une méthode d'en-tête **public** ArrayList<Integer> getListeNotes() renvoie la liste des notes de l'élève ;
- La méthode d'en-tête **public void** ajouterNote(**int** note) ajoute la note reçue en paramètre à listeNotes ; si la note reçue en paramètre est négative, la note introduite est 0 ; si la note reçue en paramètre est supérieure à 20, la note introduite est 20 ; la méthode actualise en conséquence l'attribut moyenne ; l'actualisation est faite à temps constant, et non pas en un temps proportionnel au nombre de notes déjà enregistrées.
- La méthode d'en-tête **public String** toString() qui retourne une description de l'élève considéré (par exemple : "Sophie (12.25)").

Indication : si note est une variable de type **int**, l'instruction :

listeNotes.add(note); ajoute un Integer contenant la valeur note à listeNotes.

Après avoir *terminé* la classe Eleve, écrire un programme qui teste cette classe.

Indication (détail) : si la méthode toString décrite ci-dessus a été définie dans la classe Eleve, si eleve est de type Eleve, l'instruction :

System.out.println(eleve); est équivalente à l'instruction :

System.out.println(eleve.toString());

Corrigé-Type :

Voilà le code source de la classe Eleve :

```
package elearning.univ.bejaia.dz;
import java.util.ArrayList;
public class Eleve {
    private String nom;
    private ArrayList<Integer> listeNotes = new ArrayList<Integer>();
    private double moyenne;
    public Eleve(String nom) {
      this.nom = nom;
    }
        public double getMoyenne() {
      return moyenne;
    }
        public String getNom() {
      return nom;
    }
    public ArrayList<Integer> getListeNotes() {
```

```
        return listeNotes;
    }
  /* Une note inferieure a 0 sera considere comme egale a 0 ;
    une note superieure a 20 sera considere comme egale a 20 */
    public void ajouterNote(int note) {
        int nbNotes = this.listeNotes.size();
        if (note < 0) note = 0;
        else if (note > 20) note = 20;
        this.moyenne = (this.moyenne * nbNotes + note) / (nbNotes + 1);
        listeNotes.add(note);
    }
    public String toString() {
        return nom + " (" + (int)(100 * moyenne)/100.0 + ")";
    }
}
```

Voilà un code source d'une classe de test :

```
package elearning.univ.bejaia.dz;
class TestEleve {
      public static void main(String[] arg) {
            Eleve eleve = new Eleve(arg[0]);
            for (int i = 1; i < arg.length; i++) {
                  eleve.ajouterNote(Integer.parseInt(arg[i]));
                  System.out.println("Moyenne de " + eleve.getNom() +
                              " : " + eleve.getMoyenne());
            }
            System.out.println(eleve);
      }
}
```

<u>Activité 08</u> :

Réaliser une classe *Point* permettant de représenter un point sur un axe. Chaque point sera caractérisé par un nom (de type *char*) et une abscisse (de type *double*). On prévoira :

- Un constructeur recevant en arguments le nom et l'abscisse d'un point,
- Une méthode *affiche* imprimant (en fenêtre console) le nom du point et son abscisse,
- Une méthode *translate* effectuant une translation définie par la valeur de son argument.

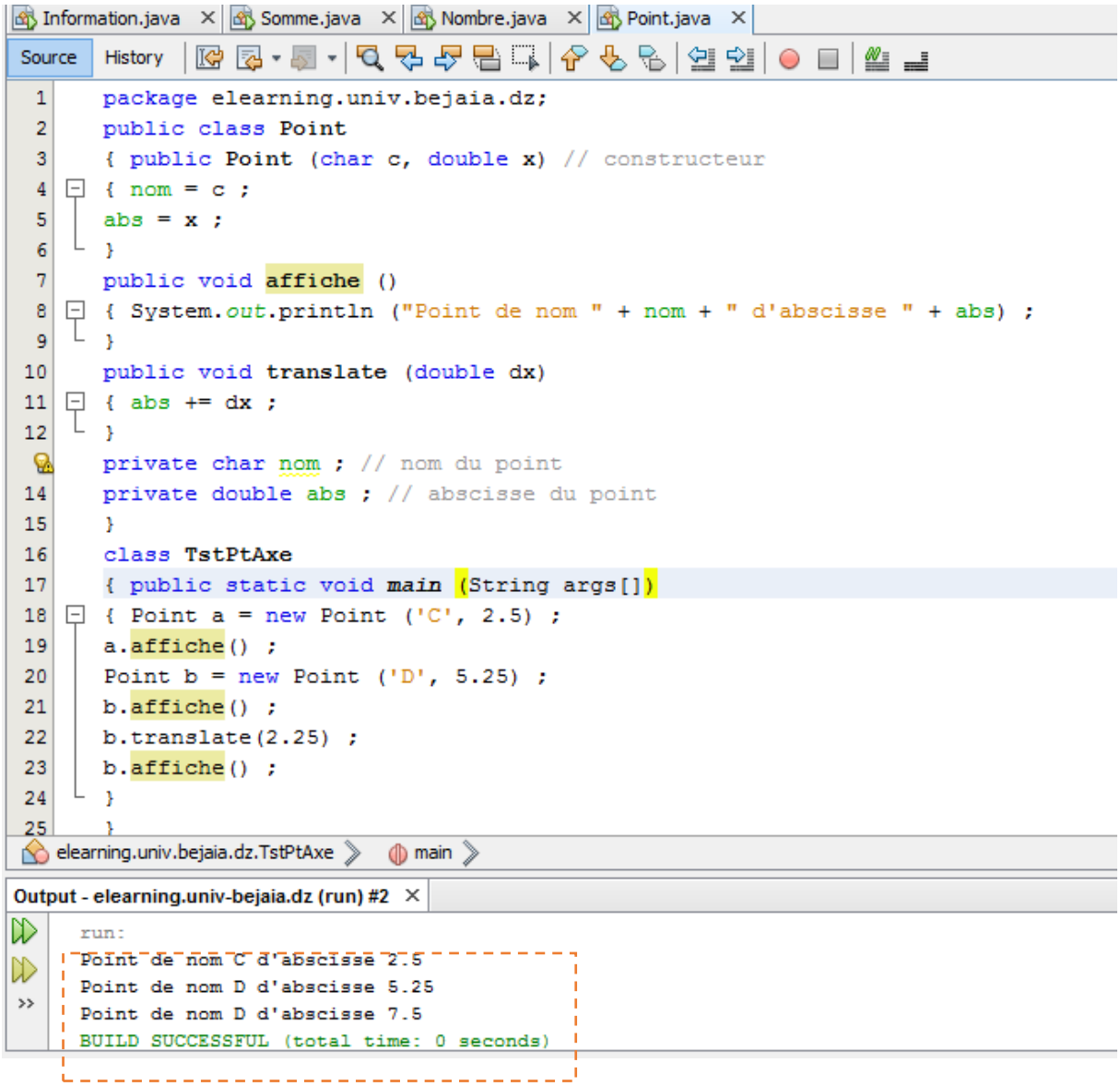
Écrire un petit programme utilisant cette classe pour créer un point, en afficher les caractéristiques, le déplacer et en afficher à nouveau les caractéristiques. (Vous devez respecter le principe d'encapsulation).

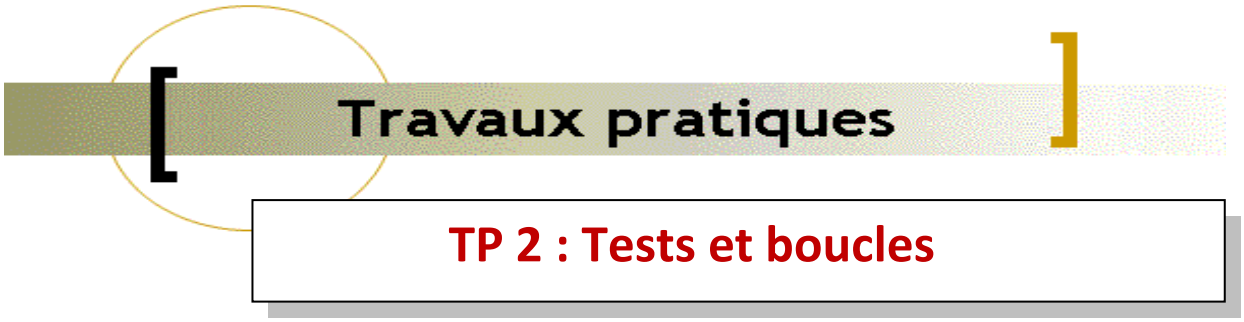
<u>Corrigé-Type</u> :

Le code source de la classe Point ainsi que sa classe de test, ici nous avons utilisé un seule fichier java avec Point comme classe principale :

```
package elearning.univ.bejaia.dz;
public class Point {
public Point (char c, double x) // constructeur {
nom = c ;
abs = x ;
}
public void affiche () {
System.out.println ("Point de nom " + nom + " d'abscisse " + abs) ;
}
public void translate (double dx) {
abs += dx ;
}
private char nom ; // nom du point
private double abs ; // abscisse du point
}
class TstPtAxe {
public static void main (String args[]) {
Point a = new Point ('C', 2.5) ;
a.affiche() ;
Point b = new Point ('D', 5.25) ;
b.affiche() ;
b.translate(2.25) ;
b.affiche() ;
}
}
```

Et la figure de l'exécution sur machine :


Information.java
Somme.java
Nombre.java
Point.java
Source
History
1 package elearning.univ.bejaia.dz;
2 public class Point
3 { public Point (char c, double x) // constructeur
4 { nom = c ;
5 abs = x ;
6 }
7 public void affiche ()
8 { System.out.println ("Point de nom " + nom + " d'abscisse " + abs) ;
9 }
10 public void translate (double dx)
11 { abs += dx ;
12 }
14 private char nom ; // nom du point
15 private double abs ; // abscisse du point
16 }
17 class TstPtAxe
18 { public static void main (String args[])
19 { Point a = new Point ('C', 2.5) ;
20 a.affiche() ;
21 Point b = new Point ('D', 5.25) ;
22 b.affiche() ;
23 b.translate(2.25) ;
24 b.affiche() ;
25 }
elearning.univ.bejaia.dz.TstPtAxe
main
Output - elearning.univ-bejaia.dz (run) #2
run:
Point de nom C d'abscisse 2.5
Point de nom D d'abscisse 5.25
Point de nom D d'abscisse 7.5
BUILD SUCCESSFUL (total time: 0 seconds)



TP 2 : Tests et boucles

Objectif :

Le but de ce TP est d'acquérir les notions fondamentales relatives au langage de programmation java à savoir, les types de données, les structures conditionnelles ainsi que les structures répétitives.

Activité 01 :

Ecrire un programme en langage java qui affiche les informations fournies par les méthodes suivantes (N est saisi au clavier) :

- Une méthode qui calcule la somme des N premiers entiers pairs (exemple : pour N= 5, on aura 2+4+6+8+10=30)
- Une méthode qui calcule qui affiche les dix nombres suivants le nombre N. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Corrigé-type partiel :

Voilà une partie du code concernant la méthode qui affiche les dix nombres succédant un nombre donné :

```
package elearning.univ.bejaia.dz;
import java.util.Scanner ; // importer le package
public class Liste {
public static void main(String[] args) {
Scanner clavier = new Scanner(System.in);
int Nombre;
System.out.println("Taper votre Nombre");
Nombre = clavier.nextInt();
for (int i=Nombre+1; i<=Nombre+10; i++)  {
System.out.println(i);
```

```
}
}
}
```

```
Source | History
1  package elearning.univ.bejaia.dz;
2  import java.util.Scanner ; // importer le package
3  public class Liste {
4  public static void main(String[] args) {
5  Scanner clavier = new Scanner(System.in);
6  int Nombre;
7  System.out.println("Taper votre Nombre");
8  Nombre = clavier.nextInt();
9  for (int i=Nombre+1; i<=Nombre+10; i++)  {
10 System.out.println(i);
11 }
12 }
13 }
```

```
Output - elearning.univ-bejaia.dz (run) #2
run:
Taper votre Nombre
17
18
19
20
21
22
23
24
25
26
27
```

Dix nombres après le 17

<u>Activité 02</u> :

Écrire une classe Etoile qui utilise une méthode affiche() permettant de visualiser un triangle isocèle formé d'étoiles.

```
    *
  * * *
* * * * *
```

La hauteur du triangle sera fournie en donnée.

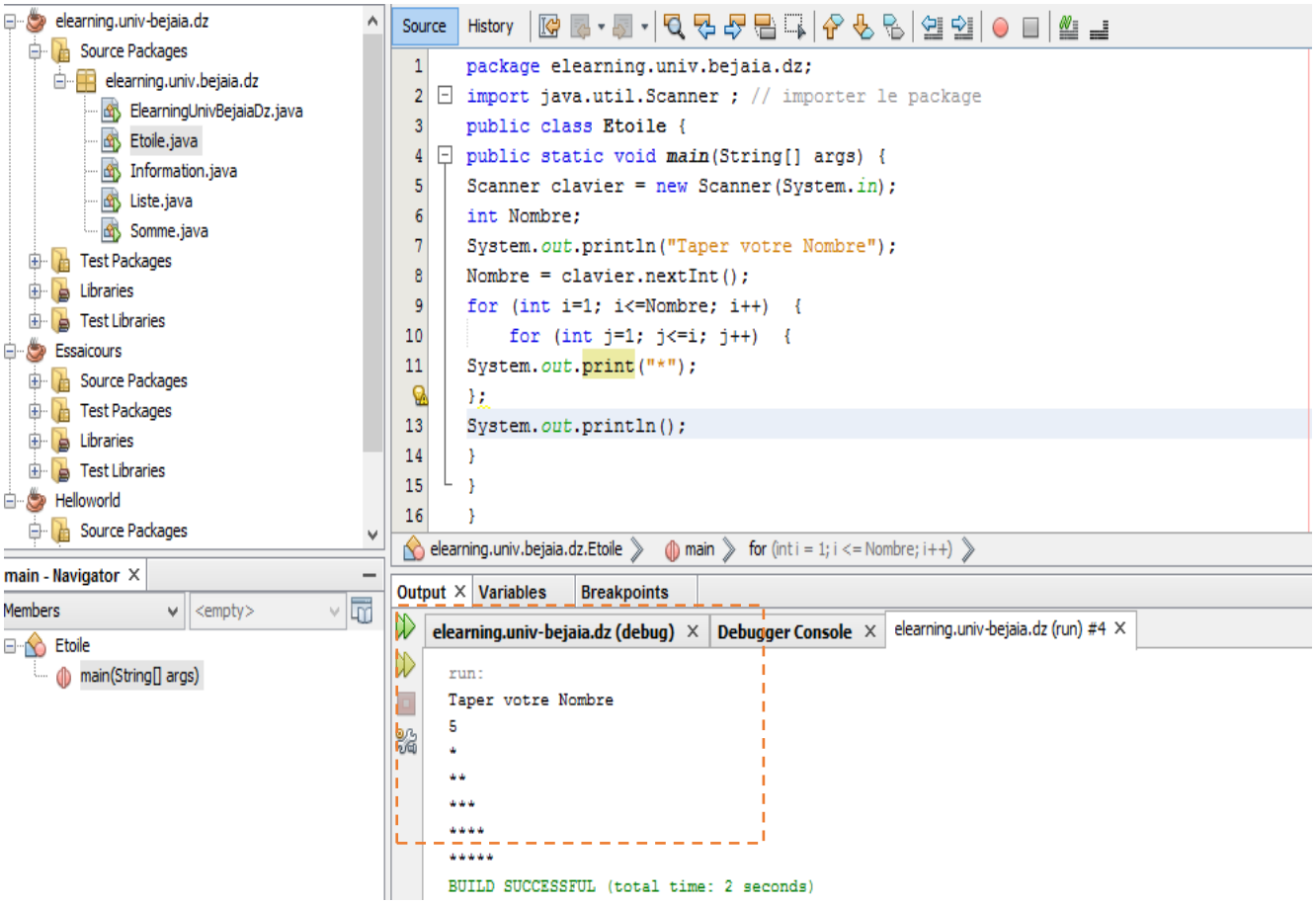
<u>Corrigé-Type</u> :

Ce qui suit est une ébauche de la solution, que l'étudiant doit améliorer.

```
package elearning.univ.bejaia.dz;

import java.util.Scanner ; // importer le package
```

```
public class Etoile {
public static void main(String[] args) {
Scanner clavier = new Scanner(System.in);
int Nombre;
System.out.println("Taper votre Nombre");
Nombre = clavier.nextInt();
for (int i=1; i<=Nombre; i++)  {
    for (int j=1; j<=i; j++)  {
System.out.print("*");
};
System.out.println();
}
}
}
```



<u>Activité 03</u> :

Ecrire un programme en langage java qui calcule la somme des éléments impairs d'un tableau de 20 entiers.

Activité 04 :

- Compiler le fichier **ProgObjet1.java** puis le faire exécuter.
- Noter le résultat de son exécution :

```
class ObjetTableau { // début de la classe
static int nbobjets = 0;
int n ;
int[] t ;
ObjetTableau(int a) { // le constructeur
n = a; // n prend la valeur a
t = new int[n] ; // on réserve la place pour le tableau t
for (int i=0;i<n;i++) { // remplissage du tableau
t[i] = i+1 ;
}
nbobjets++;
} // fin du constructeur
void affiche() {
System.out.println("affichage d'un objet");
System.out.println("n vaut "+n);
for (int i = 0; i < n; i++){
System.out.print(t[i]+" * ");
}
System.out.println("");
}
static void nombreObjets() {
System.out.println("Nombre d'objets : " + nbobjets) ;
}
} // fin de la classe

public class ProgObjet1 { // début de classe
public static void main (String args[ ]) { // le main
ObjetTableau.nombreObjets() ;
ObjetTableau obj1 = new ObjetTableau(6) ;
obj1.affiche();
ObjetTableau.nombreObjets() ;
ObjetTableau obj2 = new ObjetTableau(3) ;
obj2.affiche();
ObjetTableau.nombreObjets() ;
} // fin du main
} // fin de la classe
```

Noter dans le tableau suivant si la méthode appelée est une méthode de classe ou d'instance, et préciser quel est l'affichage correspondant :

instruction	méthode : classe / instance	affichage
`ObjetTableau.nombreObjets();`		
`ObjetTableau obj1 = new ObjetTableau(6);`		
`obj1.affiche();`		
`ObjetTableau.nombreObjets();`		
`ObjetTableau obj2 = new ObjetTableau(3);`		
`obj2.affiche();`		
`ObjetTableau.nombreObjets();`		

Travaux pratiques

TP 3 : Classes, Objets et Méthodes

Objectif :

Le but de ce TP est d'acquérir les briques de base de la programmation orientée objet.

Activité 01 :

À chaque étape de cet exercice, compilez votre fichier pour contrôler qu'il est syntaxiquement correct. Hormis pour la procédure `main`, vous ne devez pas utiliser les mots-clés `final, private, protected, public et static.`

1. Créez, dans un fichier nommé `Personne.java`, une classe `Personne`.
2. Ajoutez dans cette classe les attributs `nom` et `prénom` de type chaîne de caractères.
3. Ajoutez un attribut `âge` de type entier.
4. Ajoutez un constructeur qui initialise `nom` et `prénom`.
5. Ajoutez une méthode principale (`main`) qui crée un objet de type Personne.

Dans ce `main`, faites afficher le nom et le prénom de l'objet créé. Exécutez votre programme et contrôlez qu'il écrit bien le nom et le prénom de la personne à l'écran.

6. Ajoutez un constructeur qui initialise `âge` en plus de `nom` et `prénom`.
7. Ajoutez une méthode nommée `anniversaire` qui augmente `âge` de 1.
8. Ajoutez, dans la classe `Personne`, une méthode (procédure) nommée `setNom` qui modifie la valeur de `nom` en fonction d'un paramètre. Faites de même pour `prénom` et `âge`.
9. Ajoutez une méthode nommée `getNom` qui rend la valeur de `nom`. Faites de même pour `prénom` et `âge`.
10. Ajoutez une méthode nommée `afficher` qui affiche pour une personne ses trois attributs.
11. Modifiez la méthode `main` dans le but de faire usage du second constructeur, d'`anniversaire`, des `setx` et `getx`. Utilisez la procédure *`afficher`* pour contrôler

le résultat des *getx* et l'effet des *setx*. Exécutez votre programme et vérifiez que tout se passe comme prévu.

Activité 02 :

Il est possible en java de définir des données membres et des méthodes de classes existantes en un seul exemplaire quelque soit le nombre d'objets instances d'une classe. De même, ces données membres et ces méthodes peuvent être appelées indépendamment de toute allocation.

Implémentez la classe suivante :

```
class Myclass2 {
static int n ;
void print() {
System.out.println(n);
}
}
```

Mettre en œuvre cette classe de la façon suivante, commenter :

```
Myclass2 my1 = new Myclass2() ;
Myclass2 my2 = new Myclass2() ;
my1.print();
my2.print();
my2.n=3;
my1.print();
my2.print();
System.out.println(my1 + " " + my2);
mMyclass2.n=10;
Myclasse2.print();
my2.print();
```

Redéfinir la classe de la façon suivante :

```
class Myclass2 {
static int n ;
static void print() {
System.out.println(n);
}
}
```

Mettre alors cette classe en œuvre de la façon suivante, commenter :

```
Myclasse2 my = newMyclasse2() ;
```

```
my.print() ;
Myclasse2.n=10 ;
Myclasse2.print() ;
my.print() ;
```

Activité 03 :

```
class D {
public int x;
D() {
x=3;
};
D( int a){
this();
x=x+a;
};
D( int a, int b) {
this(b);
x= x-a;
}
}
```

```
class DD {
public static void main (String args[]) {
D a=new D(5,6);
D b=a;
System.out.println(a.x);
}
}
```

1. Prédire manuellement le résultat du programme et vérifier sur machine :
2. Revoir l'exécution en mettant le constructeur sans paramètre de la classe D à « private ». expliquez ?
3. Rendre le programme à son état d'origine et afficher a ? Que remarquez-vous ?

Travaux pratiques

TP 4 : Héritage, Polymorphisme, classes abstraites et interfaces

Objectif :

L'objectif ce TP est de présenter l'héritage, le polymorphisme et l'utilisation des interfaces en Java. Il est **fortement** conseillé d'avoir les notes de cours sous les yeux sur l'espace dédié au cours sur la plateforme e-learning (Licence 2 – Module Programmation orientée Objet). Ce TP peut faire l'objet de plusieurs séances, avec avis de l'enseignant responsable.

Ce TP doit être préparé par les étudiants au préalable et corrigé pendant les séances de TP. Les étudiants doivent être munis de leurs flash disques pour sauvegarder leurs programmes.

Dans ces activités, nous allons modéliser des plantes, des animaux, des mammifères et des oiseaux mais aussi des chiens, des aigles, des lapins et des hommes.

Activité 01 : Héritage

On souhaite développer une application permettant de gérer une médiathèque qui contient des livres, des CDs et des DVDs. Chacun de ces média sera naturellement modélisé par une classe.

Tous les médias sont décrits par un titre et un nom d'auteur. Les utilisateurs peuvent donner leur avis sur la qualité d'un média en leur attribuant une note comprise entre 0 et 5. Il est possible :

- De représenter un média par une chaine de caractères (méthode toString). Le format de cette représentation est : "Titre" par Auteur
- De donner son avis sur le média grâce à la méthode vote(int note), note devant être comprise entre 0 et 5

- D'obtenir la moyenne des votes reçus par la méthode moyenneNotes(). Si le média n'a reçu aucune note, cette méthode renvoie 0.

Les DVD possèdent une caractéristique supplémentaire permettant la gestion des zones (une zone est décrite par un entier compris entre 0 et 8, la zone 0 indique que le DVD est lisible dans toutes les zones). La méthode readable(int[] zones) renvoie true si le DVD est lisible dans une des zones passées en paramètre, false sinon.

La description du CD comporte également le format du contenu décrit par une chaine de caractères ("CD musical", "OGG", ou "MP3"). La méthode toString est modifiée pour indiquer le format et renvoie, par exemple : "Some Kind Of Trouble" par James Blunt [CD musical]

La description des livres ne comporte aucun élément particulier.

1. Donnez le code de la classe Livre
2. On souhaite maintenant développer la classe CD. Comme cette classe comportera de nombreuses méthodes communes avec la classe Livre, on souhaite introduire une classe mère générique « Media ».
   - Modifiez la classe Livre pour qu'elle hérite d'une classe Media et réorganisez votre code de manière à tirer profit de cette nouvelle organisation.
   - Ecrivez le code de la classe CD.
3. Donnez le code de la classe DVD.

Remarque : dans tous les constructeurs, le premier paramètre correspondra au nom de l'auteur, le second au titre.

Corrigé-Type :

Nous allons maintenant donner le code source de chaque classe ; il est demandé à l'étudiant de préparer ses classes sur papier et vérifier, ensuite implémenter ces classes sur machine pour faire les tests nécessaires :

- La classe Media :

```
public class Media {
private String author;
private String title;
private int nVotes;
private int notes;
public Media(String author, String title) {
this.title = title;
this.author = author;
```

```
this.notes = 0;
this.nVotes = 0;
}
public String toString() {
return "\"" + this.title + "\" par " + this.author;
}
public double moyenneNotes() {
if (this.nVotes == 0) {
return 0.0;
} else {
return this.notes / this.nVotes;
}
}
public void vote(int note) {
if (note < 0 || note > 5) {
return;
}
this.notes += note;
}
public boolean isSame(String critere, String valeur) {
if (critere.equals("titre") && this.title.equals(valeur)) {
return true;
}
if (critere.equals("auteur") && this.author.equals(valeur)) {
return true;
}
if (critere.equals("media")) {
if (valeur.equalsIgnoreCase("livre")) {
return this instanceof Livre;
} else if (valeur.equalsIgnoreCase("dvd")) {
return this instanceof DVD;
} else if (valeur.equalsIgnoreCase("cd")) {
return this instanceof CD;
}
}
return false;
}
}
```

- La classe Mediatheque :

```
import java.util.ArrayList;
import java.util.List;
public class Mediatheque {
private List<Media> collection;
public Mediatheque() {
this.collection = new ArrayList<Media>();
}
public List<Media> filtre(String critere, String valeur) {
ArrayList<Media> res = new ArrayList<Media>();
for (Media el : this.collection) {
if (el.isSame(critere, valeur)) {
```

```
res.add(el);
}
}
return res;
}
public void add(Media el) {
this.collection.add(el);
}
public static void main(String[] a) {
Mediatheque m = new Mediatheque();
m.add(new Livre("a", "b"));
for (Media el : m.filtre("media", "livre")) {
System.out.println(el);
}
}
}
```

- La classe CD :

```
public class CD extends Media {
private String type;
public CD(String author, String title, String type) {
super(author, title);
this.type = type;
}
@Override
public String toString() {
return super.toString() + " [" + this.type + "]";
}
}
```

- La classe DVD :

```
public class DVD extends Media {
private int zone;
public DVD(String author, String title, int zone) {
super(author, title);
this.zone = zone;
}
public boolean readable(int[] zones) {
if (this.zone == 0) {
return true;
}
for (int z : zones) {
if (this.zone == z) {
return true;
}
}
return false;
}
}
```

- La classe Livre :

```
public class Livre extends Media {
```

```
 public Livre(String author, String title) {
 super(author, title);
}
}
```

- La classe TestCD:

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class TestCD {
@Test
public void testCD() {
String expectedRes = "\"Some Kind Of Trouble\" par James Blunt [CD musical]";
CD c = new CD("James Blunt", "Some Kind Of Trouble", "CD musical");
assertEquals(expectedRes, c.toString());
}
}
```

- La classe TestDVD :

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestDVD {
@Test
public void testReadable() {
DVD d1 = new DVD("Lucas", "Starwars", 0);
assertTrue(d1.readable(new int[] {1, 2, 3}));
DVD d2 = new DVD("Lucas", "Starwars", 2);
assertTrue(d2.readable(new int[] {1, 2, 3}));
DVD d3 = new DVD("Lucas", "Starwars", 2);
assertFalse(d3.readable(new int[] {1, 3, 4}));
}
}
```

- La classe TestLivre :

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestLivre {
@Test
public void testLivre() {
Livre l = new Livre("Sartre", "La Nausée");
assertEquals("\"La Naus仔\" par Sartre", l.toString());
}
@Test
public void testVote() {
Livre l = new Livre("Sartre", "La Nausée");
assertEquals(0, l.moyenneNotes());
l.vote(1);
l.vote(2);
assertEquals(1/3, l.moyenneNotes(), 0.000001);
}
@Test
public void ignoreNote() {
```

```
Livre l = new Livre("Sartre", "La Nausée");
l.vote(1);
l.vote(2);
l.vote(12);
assertEquals(1/3, l.moyenneNotes(), 0.000001);
}
}
```

- La classe TestMediatheque:

```
import static org.junit.Assert.assertEquals;
import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
public class TestMediatheque {
@Test
public void testFilterAuteur() {
Mediatheque m = new Mediatheque();
CD cd1 = new CD("Queen", "The show must go on", "mp3");
CD cd2 = new CD("James Blunt", "Some Kind Of Trouble", "CD musical");
Livre l1 = new Livre("Sartre", "La Nausée");
Livre l2 = new Livre("Sartre", "L'enfer");
Livre l3 = new Livre("Baudelaire", "Le spleen de Paris");
m.add(l1);
m.add(l2);
m.add(l3);
m.add(cd1);
m.add(cd2);
ArrayList<Media> cds = new ArrayList<Media>();
cds.add(cd1);
cds.add(cd2);
ArrayList<Media> sartre = new ArrayList<Media>();
sartre.add(l1);
sartre.add(l2);
assertEquals(sartre, m.filtre("auteur", "Sartre"));
}
@Test
public void testFilterType() {
Mediatheque m = new Mediatheque();
CD cd1 = new CD("Queen", "The show must go on", "mp3");
CD cd2 = new CD("James Blunt", "Some Kind Of Trouble", "CD musical");
Livre l1 = new Livre("Sartre", "La Nausée");
Livre l2 = new Livre("Sartre", "L'enfer");
Livre l3 = new Livre("Baudelaire", "Le spleen de Paris");
m.add(l1);
m.add(l2);
m.add(l3);
m.add(cd1);
m.add(cd2);
ArrayList<Media> cds = new ArrayList<Media>();
cds.add(cd1);
cds.add(cd2);
ArrayList<Media> sartre = new ArrayList<Media>();
sartre.add(l1);
sartre.add(l2);
assertEquals(cds, m.filtre("media", "CD"));
}
@Test
```

```
public void testFilterTitre() {
Mediatheque m = new Mediatheque();
CD cd1 = new CD("Queen", "The show must go on", "mp3");
CD cd2 = new CD("James Blunt", "Some Kind Of Trouble", "CD musical");
Livre l1 = new Livre("Sartre", "La Nausée");
Livre l2 = new Livre("Sartre", "L'enfer");
Livre l3 = new Livre("Baudelaire", "Le spleen de Paris");
m.add(l1);
m.add(l2);
m.add(l3);
m.add(cd1);
m.add(cd2);
ArrayList<Media> cds = new ArrayList<Media>();
cds.add(cd2);
assertEquals(cds, m.filtre("titre", "Some Kind Of Trouble"));
}
}
```

Activité 02 : Polymorphisme

On veut écrire une classe Agenda permettant de stocker un nombre fixe (MAX_SIZE) d'évènements. Ces évènements pourront être du type Birthday ou Meeting. La classe Agenda doit permettre :

- D'ajouter un évènement à un agenda (on ne gèrera pas les éventuels conflits de date ou d'horaire),
- De supprimer tous les évènements ayant lieu avant une date passée en paramètre,
- D'afficher tous les événements ayant lieu à une date donnée.

Avant d'écrire la classe Agenda, réfléchissez à l'organisation de votre code. Comment implémenter la notion d'évènement tout en factorisant au mieux le code ? Cela induit-il des modifications des classes Birthday et Meeting ? Si oui, précisez-les. Ecrivez un programme de test de la classe Agenda.

Les activités 3..6 sont liées et représentent un projet complet

Activité 03 :

Dans ces activités, nous allons modéliser des plantes, des animaux, des mammifères et des oiseaux mais aussi des chiens, des aigles, des lapins et des hommes.

- Modélisez cette hiérarchie (directement en Java) avec des classes. Ajoutez à chaque classe fille au moins une variable et une méthode nouvelle selon votre imagination (âge, nom, sexe, …).
- Ajoutez une classe exécutable Test et créez des instances de chacune des classes. Affichez avec la méthode System.out.println() chacun des objets (attributs).

- Redéfinissez la méthode toString() (héritée de Object, voir dans la documentation) de la super classe pour qu'elle affiche "Je suis un animal" (ou "Je suis une plante"). Exécutez la classe Test. Expliquer le résultat en étudiant la classe java.lang.System.
- Modifiez la méthode précédente pour qu'elle affiche "Je suis un animal et mon identifiant est X".

<u>Activité 04</u> : Polymorphisme simple

- Avec quels types de références pouvez-vous manipuler les classes précédentes ?
- Dans la méthode main de la classe Test, créez des tableaux qui permettent de voir certains des animaux créés dans l'activité précédente comme des collections (de taille fixe) d'animaux ou de mammifères.
- Ajouter à chaque classe une méthode String getInfo() (qui appellera celle de la superclasse) pour que l'affichage du tableau suivant :

```
{new Animal(12), new Animal(), new Chien(5,"Medor"), new Homme(), new Homme(25,"Robert")}
Soit de la forme :
Je suis un animal âgé de 12 an(s).
Je suis un animal.
```

Je suis un animal âgé de 5 an(s). Je suis un mammifère. Je suis un chien de nom Médor.
Je suis un animal. Je suis un mammifère. Je suis un homme.
Je suis un animal âgé de 25 an(s). Je suis un mammifère. Je suis un homme de nom Robert.

<u>Activité 05</u> : Classes abstraites

- Modifiez les classes pour s'assurer que tous les Animaux possèdent la méthode String moyenExpression qui retourne une phrase du type "Je parle", "J'aboie", ou "Je fais des bulles" (l'aigle trompette, glapit ou glatit ; le lapin clapit).
- Est-ce que le moyen d'expression peut être définit pour toutes les classes ? Peut-on alors créer une instance de la classe Animal, quelle est votre conclusion pour s'assurer de cela ? Faites de même pour toutes les classes pour les lesquelles cela vous semble nécessaire.
- Écrivez une méthode afficherAnimaux() dans la classe Animal qui prend en paramètre un tableau d'animaux et qui affiche leur référence et leur cri. Testez-la sur le tableau de la question précédente.

Activité 06 : Interface

On souhaite maintenant considérer que certains animaux (pas forcément tous) sont des carnivores (ils comportent la méthode void manger avec comme paramètre un autre animal) ou des herbivores (ils comportent la méthode void manger avec comme paramètre une plante) ou les deux. Créez deux interfaces pour décrire ces comportements.

- Modifiez vos classes pour qu'elles implantent ces interfaces :

  ✓Les carnivores : quand ils mangent, les chiens affichent "Je mord X", et les aigles affichent "Je déchire X".

  ✓Les herbivores : les vaches affichent "Je broute X", et les lapins "Je grignotte X".

  ✓Les hommes sont à les fois carnivores et herbivores, proposez deux solutions pour le représenter :

  ✓Dans un premier temps, sans créer de nouvelle interface.

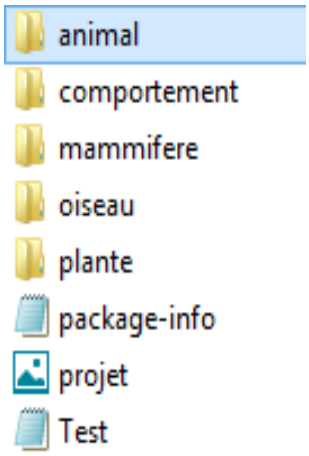
  ✓Dans un second temps en créant une nouvelle interface IOmnivore.

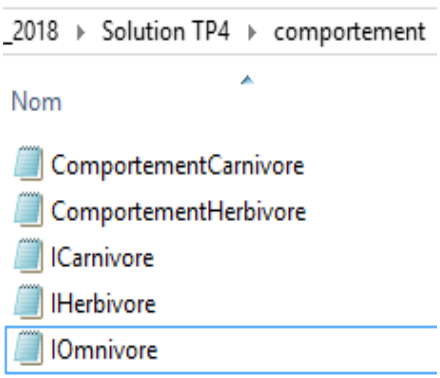
- Le polymorphisme est aussi utilisable avec les interfaces :

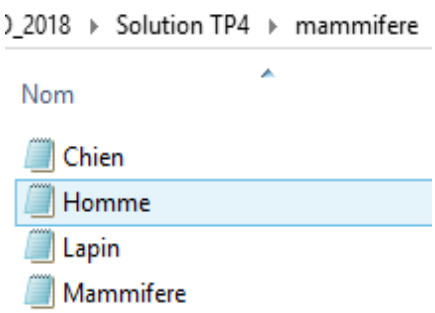
Appliquez la méthode manger(unePlante sur la Vache que avez déjà créée en la voyant à travers une référence de type IHerbivore. Peut-on appliquer la méthode moyenExpression() sur cette référence ?

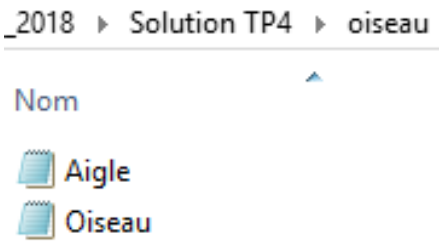
Créez deux tableaux lesCarnivores' et lesHerbivores'' pour manipuler les animaux existants. Créez un lapin, et faites le manger par tous les carnivores, créez une plante et faites la manger par les herbivores.

Corrigé-Type :










```
package elearning.univ-bejaia.dz.animal;
/** The Class Animal.
 *
 * Author EL BOUHISSI
 */
public abstract class Animal {
public final static int NB_MAX_ANIMAUX = 10;
private static Animal[] animaux = new Animal[NB_MAX_ANIMAUX];
private static int lastAnimal = 0;
public enum Sexe {
MALE, FEMELLE};
private Sexe sexe;

	public Sexe getSexe() {
	return sexe;
	}
	public void setSexe(Sexe sexe) {
	this.sexe = sexe;
	}
	public String toString() {
	return "Je suis un animal.";
	}
	public Animal(Sexe sexe) {
	super();
	this.sexe = sexe;
	animaux[lastAnimal++] = this;
	}
	public static void afficherTout() {
	System.out.println("-- Affichage de tous les animaux --");
	int i = lastAnimal;
	while (i-- > 0)
	System.out.println(animaux[i]+animaux[i].getMoyenExpression());
	}
public abstract String getMoyenExpression();
}
package elearning.univ-bejaia.dz.animal.comportement;
import elearning.univ-bejaia.dz.animal.Animal;
public class ComportementCarnivore {
public void manger(Animal animal) {
System.out.println("Je mange " + animal.getClass().getCanonicalName());
}
}
package elearning.univ-bejaia.dz.animal.comportement;
import elearning.univ-bejaia.dz.plante.Plante;
```

```
public class ComportementHerbivore {
public void manger(Plante plante) {
System.out.println("Je mange " + plante + " " + plante.getNom());
}
}
package elearning.univ-bejaia.dz.animal.comportement;
import elearning.univ-bejaia.dz.animal.Animal;
public interface ICarnivore {
public void manger(Animal proie);
}
package elearning.univ-bejaia.dz.animal.comportement;
import elearning.univ-bejaia.dz.plante.Plante;
public interface IHerbivore {
public void manger(Plante plante);
}
package elearning.univ-bejaia.dz.animal.comportement;
public interface IOmnivore extends IHerbivore, ICarnivore {
}
package elearning.univ-bejaia.dz.animal.mammifere;
import elearning.univ-bejaia.dz.animal.Animal;
import elearning.univ-bejaia.dz.animal.comportement.ComportementCarnivore;
import elearning.univ-bejaia.dz.animal.comportement.ICarnivore;
public class Chien extends Mammifere implements ICarnivore {
public Chien(Sexe sexe, int dureeGestation) {
super(sexe, dureeGestation);
}
private boolean collier;
public boolean isEquipeCollier() {
return collier;
}
public void setCollier(boolean collier) {
this.collier = collier;
}
public String toString() {
return super.toString() + "Je suis un chien.";
}
public String getMoyenExpression() {
return "J'aboie";
}
private ComportementCarnivore comportementCarnivore = new
ComportementCarnivore();
public void manger(Animal animal) {
comportementCarnivore.manger(animal);
}
}
package elearning.univ-bejaia.dz.animal.mammifere;
import elearning.univ-bejaia.dz.animal.Animal;
import elearning.univ-bejaia.dz.animal.comportement.ComportementCarnivore;
```

```
import elearning.univ-bejaia.dz.animal.comportement.ComportementHerbivore;
import elearning.univ-bejaia.dz.animal.comportement.IOmnivore;
import elearning.univ-bejaia.dz.plante.Plante;
public class Homme extends Mammifere implements IOmnivore {
public Homme(Sexe sexe, int dureeGestation) {
super(sexe, dureeGestation);
}
public String toString() {
return super.toString() + "Je suis un homme.";
}
public String getMoyenExpression() {
return "Je parle";
}
private ComportementCarnivore comportementCarnivore = new
ComportementCarnivore();
public void manger(Animal animal) {
comportementCarnivore.manger(animal);
}
private ComportementHerbivore comportementHerbivore = new
ComportementHerbivore();
public void manger(Plante plante) {
comportementHerbivore.manger(plante);
}
}
package elearning.univ-bejaia.dz.animal.mammifere;
import elearning.univ-bejaia.dz.animal.comportement.ComportementHerbivore;
import elearning.univ-bejaia.dz.animal.comportement.IHerbivore;
import elearning.univ-bejaia.dz.plante.Plante;
public class Lapin extends Mammifere implements IHerbivore {
public Lapin(Sexe sexe, int dureeGestation) {
super(sexe, dureeGestation);
}
public String getMoyenExpression() {
return "Je clapis";
}
private ComportementHerbivore comportementHerbivore = new
ComportementHerbivore();
public void manger(Plante plante) {
comportementHerbivore.manger(plante);
}
}
package elearning.univ-bejaia.dz.animal.mammifere;
import elearning.univ-bejaia.dz.animal.Animal;
public abstract class Mammifere extends Animal {
public Mammifere(Sexe sexe, int dureeGestation) {
```

```
super(sexe);
this.dureeGestation = dureeGestation;
}
private int dureeGestation;
public int getDureeGestation() {
return dureeGestation;
}
public void setDureeGestation(int dureeGestation) {
this.dureeGestation = dureeGestation;
}
public String toString() {
return super.toString() + "Je suis un mammifere.";
}
}
package elearning.univ-bejaia.dz.animal.oiseau;

import elearning.univ-bejaia.dz.animal.Animal;
import elearning.univ-bejaia.dz.animal.comportement.ComportementCarnivore;
import elearning.univ-bejaia.dz.animal.comportement.ICarnivore;
public class Aigle extends Oiseau implements ICarnivore {
private int envergure;
public Aigle(Sexe sexe, int envergure) {
super(sexe);
this.envergure = envergure;
}
public int getEnvergure() {
return envergure;
}
public void setEnvergure(int envergure) {
this.envergure = envergure;
}
public String toString() {
return super.toString() + "Je suis un aigle.";
}
public String getMoyenExpression() {
return "Je trompette";
}
private ComportementCarnivore comportementCarnivore = new ComportementCarnivore();
public void manger(Animal animal) {
comportementCarnivore.manger(animal);
}
}
package elearning.univ-bejaia.dz.animal.oiseau;
```

```
import elearning.univ-bejaia.dz.animal.Animal;
public abstract class Oiseau extends Animal {
private int tailleDuBec;
public int getTailleDuBec() {
return tailleDuBec;
}
public void setTailleDuBec(int tailleDuBec) {
this.tailleDuBec = tailleDuBec;
}
public Oiseau(Sexe sexe) {
super(sexe);
}
public String toString() {
return super.toString() + "Je suis un oiseau.";
}

}
package elearning.univ-bejaia.dz.plante;
public class Plante {
private String nom;
public Plante(String nom) {
super();
this.setNom(nom);
}
public void setNom(String nom) {
this.nom = nom;
}
public String getNom() {
return nom;
}
}
package elearning.univ-bejaia.dz;
import elearning.univ-bejaia.dz.animal.*;
import elearning.univ-bejaia.dz.animal.Animal.Sexe;
import elearning.univ-bejaia.dz.animal.comportement.ICarnivore;
import elearning.univ-bejaia.dz.animal.mammifere.Chien;
import elearning.univ-bejaia.dz.animal.mammifere.Homme;
import elearning.univ-bejaia.dz.animal.mammifere.Lapin;
import elearning.univ-bejaia.dz.animal.mammifere.Mammifere;
import elearning.univ-bejaia.dz.animal.oiseau.Aigle;
import elearning.univ-bejaia.dz.animal.oiseau.Oiseau;
```

```
public class Test {
public static void main(String[] args) {
        // Création de trois classes concrètes
        Chien unChien = new Chien(Animal.Sexe.MALE, 4);
        Homme unHomme = new Homme(Animal.Sexe.MALE, 9);
        Aigle unAigle = new Aigle(Animal.Sexe.MALE, 100);
        Lapin unLapin = new Lapin(Animal.Sexe.FEMELLE, 1);
        // Création de tableaux génériques
        Animal[] animaux = { unChien, unHomme, unAigle, unLapin };
        animaux[0].setSexe(Sexe.MALE);
        Mammifere[] mammiferes = { unHomme, unChien };
        mammiferes[0].setDureeGestation(9);
        Oiseau[] oiseaux = { unAigle };
        oiseaux[0].setTailleDuBec(5);
        // Un exemple de downcast
        ((Aigle) oiseaux[0]).setEnvergure(95);

        // Utilisation du polymorphisme
        for (Mammifere unMam : mammiferes)
                System.out.println(unMam + " " + unMam.getDureeGestation());
        Animal.afficherTout();
        // Utilisation du polymorphisme avec des Interfaces
        // Les classes utilisent la delegation par aggregation
        // pour factoriser du code (ComportementCarnivore et
        // ComportementHerbivore)
        ICarnivore[] carnivores = { unChien, unHomme, unAigle };
        for (ICarnivore carnivore : carnivores)
                carnivore.manger(unLapin);
    }

}
```

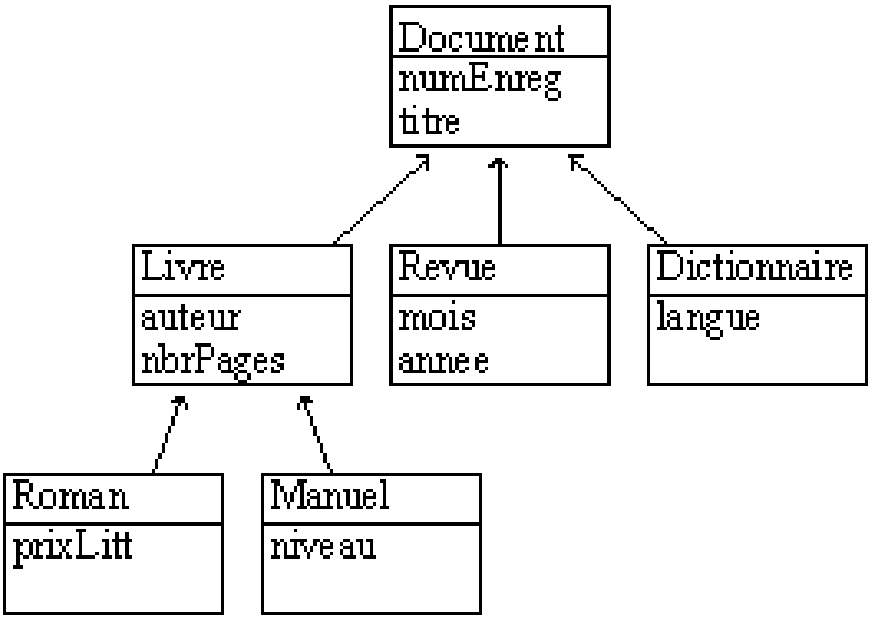
<u>Activité 07</u> : Héritage

Pour la gestion d'une bibliothèque on nous demande d'écrire une application traitant des *documents* de nature diverse : des *livres*, qui peuvent être des *romans* ou des *manuels*, des *revues*, des *dictionnaires*, etc.

Tous les documents ont un numéro d'enregistrement (un entier) et un titre (une chaîne de caractères). Les livres ont, en plus, un auteur (une chaîne) et un nombre de pages (un entier). Les romans ont éventuellement un prix littéraire (un entier

conventionnel, parmi : **GONCOURT**, **MEDICIS**, **INTERALLIE**, etc.), tandis que les manuels ont un niveau scolaire (un entier). Les revues ont un mois et une année (des entiers) et les dictionnaires ont une langue (une chaîne de caractères appartenant à un ensemble prédéfini, comme **"anglais"**, **"allemand"**, **"espagnol"**, etc.).

Tous les divers objets en question ici (*livres*, *revues*, *dictionnaires*, *romans*, etc.) doivent pouvoir être manipulées en tant que *documents*.



A. Définissez les classes **Document**, **Livre**, **Roman**, **Manuel**, **Revue** et **Dictionnaire**, entre lesquelles existeront les liens d'héritage que la description précédente suggère.

Dans chacune de ces classes définissez :

• le constructeur qui prend autant arguments qu'il y a de variables d'instance et qui se limite à initialiser ces dernières avec les valeurs des arguments,

• une méthode **public String toString()** produisant une description sous forme de chaîne de caractères des objets,

• si vous avez déclaré **private** les variables d'instance (c'est conseillé, sauf indication contraire), définissez également des « accesseurs » publics *get...* permettant de consulter les valeurs de ces variables.

Écrivez une classe exécutable **TestDocuments** qui crée et affiche plusieurs documents de types différents.

B. Une bibliothèque sera représentée par un *tableau de documents* (on verra un autre jour de meilleures manières de faire, basées sur l'emploi d'une **Collection**). Définissez une classe **Bibliotheque**, avec un tel tableau pour variable d'instance et les méthodes :

• **Bibliotheque(int capacité)** - constructeur qui crée une bibliothèque ayant la capacité (nombre maximum de documents) indiquée,

• **void afficherDocuments()** - affiche *tous* les ouvrages de la bibliothèque,

• **Document document(int i)** - renvoie le ième document,

• **boolean ajouter(Document doc)** - ajoute le document indiqué et renvoie **true** (**false** en cas d'échec),

• **boolean supprimer (Document doc)** - supprime le document indiqué et renvoie **true** (**false** en cas d'échec)

• **void afficherAuteurs()** - affiche la liste des auteurs de tous les ouvrages qui ont un auteur (au besoin, utilisez l'opérateur **instanceof**)

C. Définissez, avec un effort minimal, une classe **Livrotheque** dont les instances ont les mêmes fonctionnalités que les **Bibliotheques** mais sont *entièrement constituées de livres*. Comment optimiser dans la classe **Livrotheque** la méthode **afficherAuteurs** ?

Travaux pratiques

TP 5 : Interfaces graphiques

Objectifs :

- Maitriser les outils de base des interfaces graphiques.
- Savoir créer des interfaces graphiques

1. Généralités :

La construction d'interfaces graphiques en Java repose sur un système à base de fenêtres. Deux bibliothèques de composants d'interfaces (boutons, fenêtre de listes, boîtes combo, cases à cocher etc.) sont disponibles. La plus ancienne porte le nom de AWT (A… Windowing Toolkit) et a jeté les bases du système de fenêtrage.

Le système le plus récent est né d'une collaboration entre Sun et Netscape et porte le nom de SWING. Swing est une surcouche posée sur AWT (la plupart des classes de Swing dérivent des classes d'AWT) qui apporte de nombreuses améliorations :

- Richesse fonctionnelle et de présentation supplémentaires
- Nouveau modèle de gestion des événements
- Modèle Document / Instance pour la création de composants
- Modèle Document / Vue pour les composants d'affichage de données

2. Les bases de construction d'une interface graphique

Une interface graphique est basée sur 2 grandes composantes différentes :

- L'aspect visuel qui comprend les cadres et les composants d'interfaces qui sont placés dessus
- L'interaction avec les utilisateurs (action de la souris, du clavier) c'est à dire en fait la gestion des événements.

La programmation visuelle s'appuie sur le concept de fenêtre cher aux environnements modernes. Toute programmation graphique s'appuie sur un conteneur global. Celui-ci peut être :

- Une fenêtre cadre
- Une fenêtre de dialogue
- Une applet

Les deux premiers types sont assez semblables car ils sont généraux à tout type de système fenêtré. Par exemple, sous Windows, une fenêtre cadre correspond à la fenêtre principale d'une application.

- Les composants visuels
- Les composants
- Tout composant visuel descend plus ou moins directement de la classe
- La gestion des événements

Les événements sont des objets spéciaux qui sont générés par les actions de l'utilisateur (un clic de souris par exemple) ou par le système lui-même (une défaillance du système de fichier).

Par défaut, ils sont placés dans la file d'événements (events queue). Certains événements sont gérés par le système lui-même alors que les autres doivent être pris en compte par l'utilisateur. La file des messages a une taille limitée qui fait que les événements les plus anciens et qui n'ont pas été traités sont détruits afin de laisser la place aux événements les plus récents.

Il y a deux grands moyens pour gérer les événements :

1. Le programme va lui-même pécher un événement dans la file et établit une réponse appropriée. Il a alors la possibilité de retirer l'événement s'il juge sa réponse suffisante ou bien de le laisser dans la file de manière à ce qu'il soit à nouveau géré par un mécanisme automatique du système.
2. Le programme met en place des objets spécialisés dits écouteurs d'événements spécialisés dans le traitement d'un type particulier d'événement (par exemple le déplacement de la souris) et associés à un élément d'interface

précis (une fenêtre par exemple). Le système redirige alors les événements idoines vers une méthode particulière de l'objet.

3. Les listeners

Ce sont des objets spécialisés dans le traitement des événements. Comme explicité ci-dessus, ils doivent implémenter des interfaces particulières qui leur permettent de traiter les événements.

Chaque interface est associée à un grand type d'événements. Par exemple, l'interface java.awt.event.ActionListener correspond aux événements de type Commande (clicker sur un bouton, activer un menu etc.). Cette interface contient la méthode :

```
void actionPerformed(ActionEvent e)
```

Cette méthode est destinée à être redéfinie dans la classe de votre objet Listener. Elle sera appelée automatiquement lorsque l'utilisateur cliquera sur un bouton par exemple. On ajoute un objet listener à un élément d'interface à l'aide d'une méthode addXXXListener ou XXX dénote le type d'événement traité. Les éléments suivants sont à prendre en compte :

- Un même objet peut répondre à différents types d'événements pour peu qu'il implémente les interfaces adéquates
- Un même objet peut être le listener de plusieurs sources
- Une même source peut avoir autant de listeners que de types d'événements qu'elle peut générer.
- On peut même connecter plusieurs listeners sur le même type d'événements de la même source. Ils seront appelés dans l'ordre ***inverse*** de leur ajout

4. Gestion des événements :

Ce mécanisme de gestion comporte trois éléments : la source de l'événement (e.g. un bouton), l'objet de l'événement et l'écouteur d'événement. Pour spécifier qu'un événement vient de se produire, la source envoie un objet événementiel à l'écouteur qui pourra l'utiliser pour répondre à l'événement.

Pour gérer un événement, le programmeur doit inscrire un écouteur d'événement (souvent à l'aide de la méthode *addActionListener*) et définir un gestionnaire

d'événement (dont le nom est passé comme paramètre à *addActionListener*). Par exemple, un clic sur un bouton sera géré de la manière suivante :

```
Jbutton bouton = new Jbutton("Bouton");
ButtonHandler gestionnaire = new ButtonHandler();
bouton.addActionListener(gestionnaire);
```

Dans cet exemple, la classe ButtonHandler (qui constitue le gestionnaire d'événement) doit être définie par le programmeur et doit implémenter l'interface *ActionListener*. La méthode *actionPerformed* de cette interface doit dès lors être définie :

```
class ButtonHandler implements ActionListener {
public void actionPerformed(ActionEvent evenement) { ... }
}
```

Parmi les types d'événements, on retrouve *ActionEvent*, *MouseEvent*, *KeyEvent*, *ItemEvent*, etc. À chacun de ces événements est associé un écouteur correspondant et de nom semblable (e.g. *ActionListener* <> *ActionEvent*). La documentation de la classe java.awt.Event permet d'avoir plus d'informations sur les différents types d'événements et écouteurs utilisables en Java.

5. Notion d'adaptateurs

Nous avons vu au cours qu'une classe implémentant une interface doit redéfinir toutes les méthodes présentes dans l'interface. Or, il arrive parfois que l'on souhaite n'utiliser que l'une ou l'autre des méthodes de l'interface écouteur.

On voudra alors éviter de devoir définir toutes les méthodes de cette interface ce qui sera rendu possible par les adaptateurs.

Un adaptateur fournit une implémentation par défaut pour toutes les méthodes de l'interface écouteur de nom semblable (e.g. à l'écouteur *MouseListener* correspond un adaptateur nommé *MouseAdapter*).

Le programmeur peut alors hériter de la classe adaptateur et ne surcharger que les méthodes qui lui sont réellement utiles.

Remarque : toutes les méthodes par défaut d'un adaptateur ont un corps vide, c'est-à-dire qu'elles n'ont aucune action particulière.

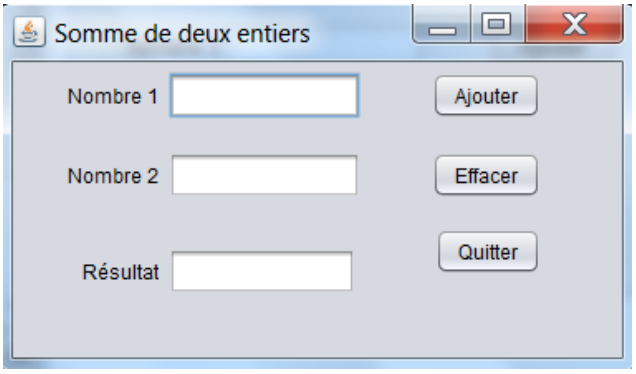
Voici quelques composants de javax.swing :

Composant	*Description*
JLabel	Zone d'affichage d'un texte ou d'une icône immuable.
JTextField	Zone dans laquelle l'utilisateur entre des données au clavier. Cette zone peut également afficher des informations.
JTextArea	Zone qui permet d'afficher ou de saisir plusieurs lignes de texte.
JButton	Zone d'activation d'un événement lors d'un clic de souris.
JCheckBox	Composant GUI qui est soit sélectionné, soit non sélectionné.
JComboBox	Liste déroulante d'éléments où l'utilisateur peut faire un choix, soit d'un clic de souris, soit en introduisant le nom de l'élément dans une case réservée à cet effet.
JList	Zone d'affichage d'une liste d'éléments. L'utilisateur peut sélectionner d'un clic de souris un élément de la liste. Un double-clic génère un événement d'action. La sélection peut porter sur plusieurs éléments.
JPanel	Un conteneur qui reçoit d'autres composants.
JFrame	Fenêtre disposant d'une barre de titre et d'une bordure (reprend le style GUI propre à la plate-forme locale sur laquelle est exécutée le programme).
JSlider	Composant présentant un curseur et une glissière à graduations.

Activité 1 :

Réaliser l'application suivante :

- Le bouton « ajouter » permet de faire la somme de Nombre 1 et Nombre 2, le résultat est mis dans la zone du résultat.



- Le bouton Effacer permet de vider les champs Nombre1, Nombre2 et Résultat.
- Le bouton Quitter permet de fermer l'application.

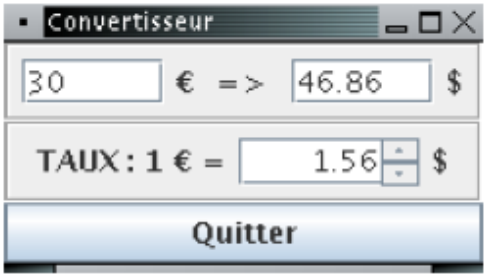
On donne à titre indicatif :

Conversion d'un type primitif en une chaîne de caractères	String chaine = String.valueOf(valeur);

Activité 2 :

Le but de cet exercice est de réaliser un convertisseur graphique Euros/Dollars. Avec les fonctionnalités suivantes :

- Si l'utilisateur entre une valeur dans la case euros et valide (touche entrée) alors le montant correspondant en dollars est calculé et affiché dans la case correspondante, de plus une flèche => est affichée pour indiquer le sens de la conversion.
- De même avec la case euros (et une flèche <=).
- L'utilisateur doit aussi pouvoir changer le taux de change à l'aide des flèches d'un bouton JSpinner (voir image ci-dessous).
- Enfin, si l'utilisateur ferme la fenêtre ou clique sur quitter l'exécution du programme se termine.
- Les saisies erronées seront traitées par des exceptions.

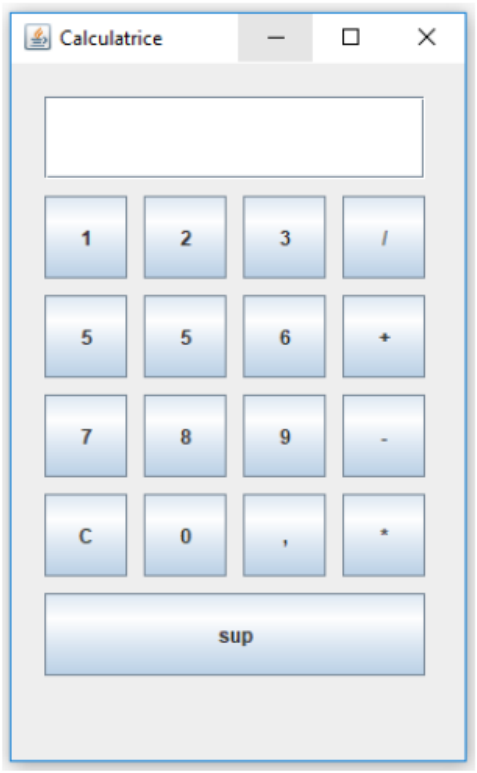


Activité 3 :

Cet exercice consiste à réaliser une calculatrice comme le montre la figure ci-dessous (page suivante).

Cette calculatrice doit permettre d'exécuter des opérations arithmétiques sur des nombres réels.

- Le bouton C sera utilisé pour effacer tout le contenu de la zone texte
- Le bouton sup sera utilisé pour effacer le dernier caractère ajouté dans la zone texte
- Pour mieux contrôler la saisie, il faut seulement autoriser la saisie de valeurs dans la zone de texte via les touches de la calculatrice.



- Le code permettant de créer la fenêtre est donné dans la page suivante
- On ne peut pas utiliser le bouton virgule s'il a déjà été utilisé pour la saisie en cours

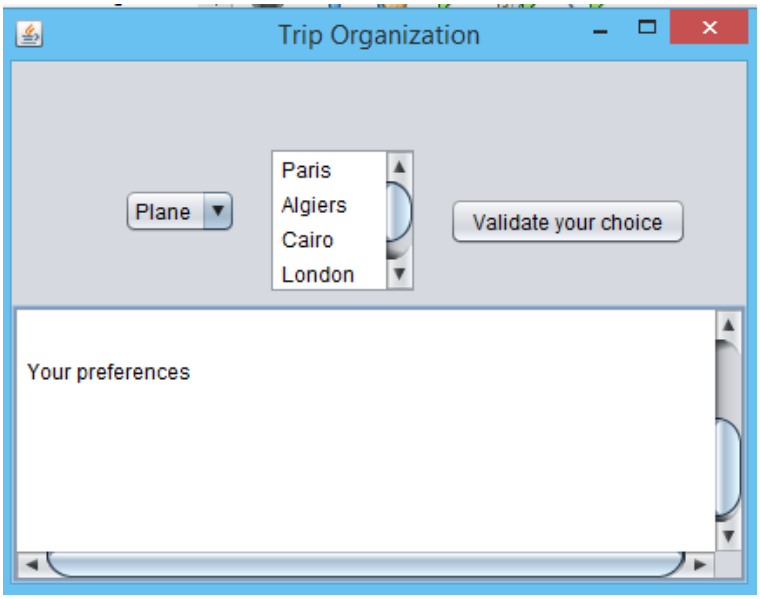
Corrigé-type :

```
import javax . swing . JButton ;
import javax . swing . JFrame ;
import javax . swing . JTextField ;
public class Calculette extends JFrame {
JTextField text = new JTextField ();
JButton but1 = new JButton ("1");
JButton but2 = new JButton ("2");
JButton but3 = new JButton ("3");
JButton but4 = new JButton ("5");
JButton but5 = new JButton ("5");
JButton but6 = new JButton ("6");
JButton but7 = new JButton ("7");
JButton but8 = new JButton ("8");
JButton but9 = new JButton ("9");
JButton but0 = new JButton ("0");
JButton plus = new JButton ("+");
JButton moins = new JButton ("-");
JButton fois = new JButton ("*");
JButton div = new JButton ("/");
JButton point = new JButton (",");
JButton clean = new JButton ("C");
JButton sup = new JButton (" sup");
public void afficherCalculette () {
```

```
text . setBounds (20 , 20, 230 , 50) ;
but1 . setBounds (20 , 80, 50, 50) ; but2 . setBounds (80 , 80, 50, 50); but3 .
setBounds (140 , 80, 50, 50) ; div . setBounds (200 , 80, 50, 50);
but4 . setBounds (20 , 140 , 50, 50) ; but5 . setBounds (80 , 140 , 50, 50) ; but6
.
setBounds (140 , 140 , 50, 50) ; plus . setBounds (200 , 140 , 50, 50);
but7 . setBounds (20 , 200 , 50, 50) ; but8 . setBounds (80 , 200 , 50, 50) ; but9
.
setBounds (140 , 200 , 50, 50) ; moins . setBounds (200 , 200 , 50, 50) ;
clean . setBounds (20 , 260 , 50, 50); but0 . setBounds (80 , 260 , 50, 50) ;
point .
setBounds (140 , 260 , 50, 50) ; fois . setBounds (200 ,260 , 50, 50);
sup . setBounds (20 , 320 , 230 , 50);
this . setLayout ( null );
this . add( but1 );
this . add( but2 );
this . add( but3 );
this . add( but4 );
this . add( but5 );
this . add( but6 );
this . add( but7 );
this . add( but8 );
this . add( but9 );
this . add( but0 );
this . add( clean );
this . add( plus );
this . add( moins );
this . add( fois );
this . add( div);
this . add( text );
this . add( point );
this . add( sup);
this . setVisible ( true );
this . setSize (290 , 460) ;
this . setTitle (" Calculatrice ");
this . setLocationRelativeTo ( null );
}
}
```

Activité 4 :

On souhaite obtenir l'interface suivante, comportant : un bouton de choix (de la classe **Choice**), une liste (de la classe **List**), un bouton (de la classe **Button**) et un champ de texte (de la classe **TextArea**).



Soit un code source partiel de l'image ci-dessus :

```
import java.awt.* ;
class Fenetre extends Frame {
protected Panel p ;
protected Choice c ;
protected List l ;
protected Button suite ;
protected TextArea texte ;
Fenetre() { // le constructeur
p = new Panel() ;
c = new Choice() ;
... // ajout des items dans l'objet Choice
p.add(c) ;
l = new List(5) ;
... // ajout des items dans l'objet List
p.add(l) ;
... // création du bouton suite
p.add(suite) ;
add("North", p) ;
texte = new TextArea();
... // remplissage du champ de texte
add("South", texte) ;
} }
public class UtilAwt {
public static void main(String args[]) {
Fenetre f = new Fenetre() ;
f.pack() ; f.show() ;
}
}
```

- Compléter le constructeur Fenetre() afin d'aboutir au résultat escompté en suivant les instructions mises en commentaires.

Travaux pratiques

TP 6 : Les Collections

Objectif

L'objectif de ce TP est d'expliquer la procédure de tri d'une collection (nous allons utiliser le langage Java et nous focaliser sur le type ArrayList).

Par défaut, les éléments dans ArrayList sont affichés dans l'ordre de leur insertion dans la liste, mais parfois on a besoin de parcourir ces éléments dans l'ordre croissant ou décroissant.

Les codes suivants implémentent la méthode *Collections.sort()* qui fait le tri d'un Arraylist dans l'ordre croissant et décroissant.

- ***Exemple1 : Trier ArrayList dans l'ordre croissant***

```
import java.util.ArrayList; // ligne obligatoire
import java.util.Collections; // ligne obligatoire (on peut les résumer par import java.util.*
public class Tri {
  public static void main(String[] args) {
    ArrayList maliste = new ArrayList();
    maliste.add("01");
    maliste.add("0A");
    maliste.add("0B");
    maliste.add("ETX");
    maliste.add("00");
    maliste.add("0C");
    maliste.add("NUL");
    maliste.add("05");
    maliste.add("19");
    maliste.add("0001011");
   System.out.println("Avant le tri") ;
    for(int i=0; i < maliste.size(); i++)
      System.out.println(maliste.get(i));
      System.out.println("\nAprès le tri") ;
    Collections.sort(maliste);
    for(int i=0; i < maliste.size(); i++)
      System.out.println(maliste.get(i));
  }
```

}

Exécution :

```
Avant le tri
01
0A
0B
ETX
00
0C
NUL
05
19
0001011

Après le tri
00
0001011
01
05
0A
0B
0C
19
ETX
NUL
```

On applique la méthode *Collections.reverseOrder()* après l'appel de la méthode *Collections.sort()* dans le but d'inverser la liste.

Il peut être donné avec deux façons :

On tri la liste avec la méthode *sort()*, puis on fait appel à la méthode *reverse()*:

```
Collections.sort();
Collections.reverse();
```

et la deuxième, l'appel de la méthode *reveseOrder()* se fait dans la méthode sort():

```
Collections.sort(maliste, Collections.reverseOrder());
```

Trier un ArrayList d'objets avec les interfaces Comparable et Comparator

Dans ce suit, nous allons voir comment faire le tri d'un ArrayList qui contient des objets à l'aide des deux interfaces *java.lang.Comparable* et *java.util.Comparator*.

L'interface Comparable est utilisée lorsqu'on veut trier une liste d'objets en comparant deux objets avec la méthode prédéfinie objet1.compareTo(object2).

- ***Exemple 2 : Trier un ArrayList en utilisant Comparable***

Soit la classe Employe :

```
public class Employe {
    private String nom;
   private int age;
   private long salaire;
    public String getName() {
       return nom;
   }
    public int getAge() {
       return age;
   }
    public long getSalary() {
       return salaire;
   }
    public Employe(String nom, int age, int salaire) {
       this.nom = nom;
       this.age = age;
       this.salaire = salaire;
   }
    @Override
   //cette méthode affiche les informations de l'employé
   public String toString() {
       return "[nom=" + this.nom + ", age=" + this.age + ", salaire=" +
              this.salaire + "]";
   }
}
```

Maintenant, on veut créer une liste d'objets ArrayList<Employe>:

```
import java.util.ArrayList;
import java.util.Collections;
public class Tri3 {
 public static void main(String[] args) {
  ArrayList<Employe> Employers = new ArrayList<Employe>();
  Employers.add(new Employe("mateo", 32, 20000));
  Employers.add(new Employe("katia", 26, 10000));
  Employers.add(new Employe("aline", 30, 40000));
  Employers.add(new Employe("salim", 28, 35000));
   System.out.println("liste ordonnée des employés :");
         Collections.sort(Employers);
  for(Employe e: Employers)
  System.out.println(e);
```

```
 }
}
```

L'appel de la méthode *Collections.sort()* génère une erreur car elle est appliquée sur un ArrayList d'objets. Pour cela, nous allons utiliser les deux interfaces *Comparable* et *Comparator*. Une erreur de compilation s'affiche:

Supposons qu'on veut trier la liste selon la propriété: âge de l'employé. Cela est peut être fait en implémentant l'interface *Comparable*, puis java fait une comparaison automatique avec la méthode *compareTo()* redéfinie .

Après l'implémentation de l'interface *Comparable* dans la classe Employe.java, voici le résultat:

```
public class Employe implements Comparable<Employe> {
    private String nom;
   private int age;
   private long salaire;
  public String getName() {
      return nom;
   }
    public int getAge() {
      return age;
   }
    public long getSalary() {
      return salaire;
   }
    public Employe(String nom, int age, int salaire) {
      this.nom = nom;
      this.age = age;
      this.salaire = salaire;
   }
    @Override
   //cette méthode affiche les informations de l'employé
   public String toString() {
      return "[nom=" + this.nom + ", age=" + this.age + ", salaire=" +
            this.salaire + "]";
   }
 @Override
 public int compareTo(Employe emp) {
  // trions les employés selon leur âge dans l'ordre croissant
  // retourne un entier négative, zéro ou positive si l'âgé
  //de cet employé est moins que, égale à ou supérieur à l'objet comparé avec
      return (this.age - emp.age);
 }
}
```

Maintenant, on peut créer la liste d'employés et appeler la méthode *Collections.sort()* (la classe Tri3 améliorée)

```
import java.util.ArrayList;
import java.util.Collections;
public class Tri3 {
 public static void main(String[] args) {
  ArrayList<Employe> Employers = new ArrayList<Employe>();
  Employers.add(new Employe("mateo", 32, 20000));
  Employers.add(new Employe("katia", 26, 10000));
  Employers.add(new Employe("aline", 30, 40000));
  Employers.add(new Employe("salim", 28, 35000));
  Collections.sort(Employers);
   System.out.println("liste ordonnée des employés selon l'âge:");
  for(Employe e: Employers)
   System.out.println(e);
 }
}
```

Exécution :

liste ordonnée des employés par age:
[nom=katia, age=26, salaire=10000]
[nom=salim, age=28, salaire=35000]
[nom=aline, age=30, salaire=40000]
[nom=mateo, age=32, salaire=20000]

- ***Exemple 3 : Trier un ArrayList en utilisant Comparator***

La liste des employés ci-dessus est triée dans l'ordre croissant mais pourquoi utiliser l'interface *Comparator* alors ? Parce qu'il y aura peut-être des situations dont le tri est basé sur plusieurs paramètres contrairement à Comparable. Par exemple, on veut bien que le tri des employés soit basé sur le nom, l'âge et le salaire.

La méthode *compare(Object o1, Object o2)* de l'interface *Comparator* prend deux objets. Dans notre exemple, elle doit être implémentée trois fois (nom, âge et salaire) de manière qu'elle retourne un entier négative si le premier argument est plus petit que le deuxième et zéro s'ils sont égaux et un entier positive si le premier argument est plus grand que le deuxième.

(nous allons modifier la classe Employe : Employe2)

```
import java.util.Comparator;
public class Employe2 {
    private String nom;
   private int age;
   private long salaire;
```

```
    public String getNom() {
        return nom;
    }
  public int getAge() {
        return age;
    }
     public long getSalaire() {
        return salaire;
    }
     public Employe2(String nom, int age, int salaire) {
        this.nom = nom;
        this.age = age;
        this.salaire = salaire;
    }
     @Override
    //cette méthode affiche les informations de l'employé
    public String toString() {
        return "[nom=" + this.nom + ", age=" + this.age + ", salaire=" +
                this.salaire + "]";
    }
    // Comparator pour le tri des employés par nom
    public static Comparator<Employe2> ComparatorNom = new Comparator<Employe2>() {
        @Override
        public int compare(Employe2 e1, Employe2 e2) {
            return e1.getNom().compareTo(e2.getNom());
        }
    };
   //  Comparator pour le tri des employés par âge
    public static Comparator<Employe2> ComparatorAge = new Comparator<Employe2>() {
      @Override
        public int compare(Employe2 e1, Employe2 e2) {
            return (int) (e1.getAge() - e2.getAge());
        }
    };
    // Comparator pour le tri des employés par salaire
       public static Comparator<Employe2> ComparatorSalaire = new Comparator<Employe2>() {
     @Override
        public int compare(Employe2 e1, Employe2 e2) {
            return (int) (e1.getSalaire() - e2.getSalaire());
        }
    };
}
```

La classe de test main:

```
import java.util.ArrayList;
import java.util.Collections;
public class main{
 public static void main(String[] args) {
  ArrayList<Employe2> Employers = new ArrayList<Employe2>();
  Employers.add(new Employe2("mateo", 32, 20000));
  Employers.add(new Employe2("katia", 26, 10000));
  Employers.add(new Employe2("aline", 30, 40000));
```

```
 Employers.add(new Employe2("salim", 28, 35000));
  //tri par nom
 Collections.sort(Employers, Employe2.ComparatorNom);
   System.out.println("liste ordonnée des employés par nom:");
 for(Employe2 e: Employers)
  System.out.println(e);
  //tri par age
 Collections.sort(Employers, Employe2.ComparatorAge);
 System.out.println("liste ordonnée des employés par age:");
 for(Employe2 e: Employers)
  System.out.println(e);
  //tri par salaire
 Collections.sort(Employers, Employe2.ComparatorSalaire);
  System.out.println("liste ordonnée des employés par salaire:");
 for(Employe2 e: Employers)
  System.out.println(e);
 }
}
```

Exécution :

liste ordonnée des employés par nom:
[nom=aline, age=30, salaire=40000]
[nom=katia, age=26, salaire=10000]
[nom=mateo, age=32, salaire=20000]
[nom=salim, age=28, salaire=35000]
liste ordonnée des employés par age:
[nom=katia, age=26, salaire=10000]
[nom=salim, age=28, salaire=35000]
[nom=aline, age=30, salaire=40000]
[nom=mateo, age=32, salaire=20000]
liste ordonnée des employés par salaire:
[nom=katia, age=26, salaire=10000]
[nom=mateo, age=32, salaire=20000]
[nom=salim, age=28, salaire=35000]
[nom=aline, age=30, salaire=40000]

Travaux pratiques

TP 7 : Les Exceptions

Objectifs :

- Acquérir le mécanisme de gestion des exceptions.
- Savoir mettre en place et gérer une exception.

Activité 1 :

Quel est la sortie du code suivant ?

```
public class Main {
public static void main(String args[]) {
try {
System.out.print("Calculer :" + " " + 1 / 0);
}
catch(ArithmeticException e) {
System.out.print("Exception : Division par zéro");
}
}
}
```

Exception : Division par zéro

Activité 2

Toutou est une classe avec deux propriétés privées String nom et int nombrePuces.

Ecrire un constructeur public Toutou (String n, int np) qui propage des exceptions de type IllegalArgumentException lorsque le nom n est null ou lorsque le nombre de puces np est négatif. Utiliser ce constructeur dans une méthode main pour contrôler les appels new Toutou ("milou", 4) et new Toutou ("medor", -11) et afficher les erreurs éventuelles lors de l'exécution des constructeurs.

```
public class Toutou {
private String nom;
private int nombrePuces;
public Toutou (String n, int np) throws IllegalArgumentException {
if (n == null)
throw new IllegalArgumentException("pas de nom !");
this.nom = n;
if (np < 0)
```

```
throw new IllegalArgumentException("nombre negatif de puces !");
this.nombrePuces = np;
}
public String toString() {
return nom + " a " + nombrePuces + " puces.";
}
public static void main(String[] args) {
try {
System.out.println("creation d'un premier toutou");
Toutou milou = new Toutou ("milou", 4);
System.out.println("le voici : " + milou);
System.out.println("creation d'un second toutou");
Toutou medor = new Toutou ("medor", -11);

System.out.println("le voici : " + medor);
}
catch (IllegalArgumentException e) {
System.out.println("un toutou rate !! " + e);
}
}
}
```

Exécution :

Création d'un premier toutou
Le voici : milou a 4 puces.
creation d'un second toutou
un toutou rate !! java.lang.IllegalArgumentException: nombre negatif de puces !

Activité 3

La méthode parseInt est spécifiée ainsi :

public static intparseInt(Strings) throws NumberFormatException

NumberFormatException - if the string does not contain a parsable integer.

Utilisez cette méthode pour faire la somme de tous les entiers donnés en argument de la ligne de commande, les autres arguments étant ignorés.

```
class Somme {
public static void main(String[] args) {
int somme = 0;
for(int i=0;i<args.length;i++) {
try {
somme += Integer.parseInt(args[i]);
}
catch (NumberFormatException e) {}
}
System.out.println(somme);
} }
```

Travaux pratiques

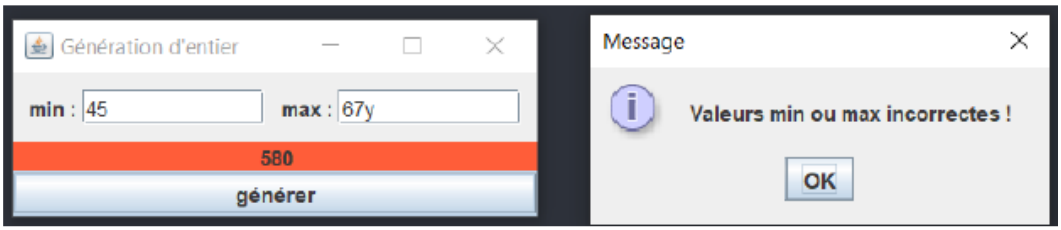
Activités de perfectionnement

Cette dernière partie du fascicule concerne des activités de perfectionnement. L'apprenant pourrait toujours faire des TPs pour améliorer son niveau. Certaines activités sont accompagnées par une solution-type.

Activité 1 :

Écrivez un programme qui permet de générer un entier aléatoirement, entre deux valeurs données, comme dans l'image ci-dessous ou le nombre sur fond rouge est celui qui est généré.

On peut modifier les valeurs min et max, et générer un entier en cliquant sur le bouton. Si les min et max ne sont pas correctes (min>max ou valeur non entière), une fenêtre de dialogue indique l'erreur.



Solution-Type:

```
public class Alea extends JFrame implements ActionListener{
private JTextField min, max;
private JLabel alea;
public Alea(){
super("G\u00e9n\u00e9ration d'entier");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.min = new JTextField(10);
this.max = new JTextField(10);
this.alea = new JLabel(" ",javax.swing.SwingConstants.CENTER);
this.alea.setBackground(new java.awt.Color(255,87,51));
this.alea.setOpaque(true);
JButton jb = new JButton(" g\u00e9n\u00e9rer ");
jb.setSize(new java.awt.Dimension(100,30));
jb.addActionListener(this);
Box b = new Box(BoxLayout.X_AXIS);
```

```
b.setBorder(javax.swing.BorderFactory.createEmptyBorder(10,10,10,10));
b.add(new JLabel("min : "));
b.add(this.min);
b.add(b.createHorizontalStrut(10));
b.add(new JLabel("max : "));
b.add(this.max);
this.add(b,java.awt.BorderLayout.NORTH);
this.add(this.alea,java.awt.BorderLayout.CENTER);
this.add(jb,java.awt.BorderLayout.SOUTH);
this.pack();
this.setVisible(true);
}
public void actionPerformed(ActionEvent e){
try{
int min = Integer.parseInt(this.min.getText());
int max = Integer.parseInt(this.max.getText());
if(min>max) throw new NumberFormatException();
this.alea.setText(""+((int)(min+Math.random()*(max-min+1))));
}
catch(NumberFormatException ex){ javax.swing.JOptionPane.showMessageDialog(this,"Valeurs min
ou max incorrectes !"); }
}
public static void main(String[] t){
new Alea();
}
}
```

Activité 2 :

Écrivez les classes A et B de manière à ce que l'exécution du programme suivant produise l'affichage 2,2,5,3,5.

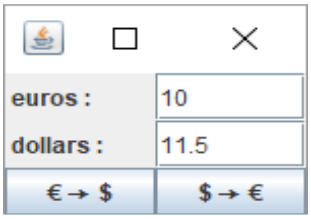
Solution-Type :

```
A a = new A();
A b = new B(5);
System.out.println(a.i);
System.out.println(b.i);
System.out.println(((B) b).i);
System.out.println(a.getI());
System.out.println(b.getI());
 class A{
int i;
public A(){
this.i = 2;
}
public int getI(){
return 3;
}
}
class B extends A{
int i;
```

```
public B(int i){
super();
this.i = i;
}
public int getI(){
return this.i;
}
public static void main(String[] t){
A a = new A();
A b = new B(5);
System.out.println(a.i);
System.out.println(b.i);
System.out.println(((B) b).i);
System.out.println(a.getI());
System.out.println(b.getI());
}
}
```

Activité 3 :

Compléter la classe Convertisseur1 qui représente une interface permettant de convertir un montant d'euros en dollars ou inversement, avec deux boutons. Si le montant n'est pas correct (Par exemple l'utilisateur a saisi des lettres), il faut afficher une fenêtre d'erreur. On peut prendre comme taux 1.15 dollar pour un euro.

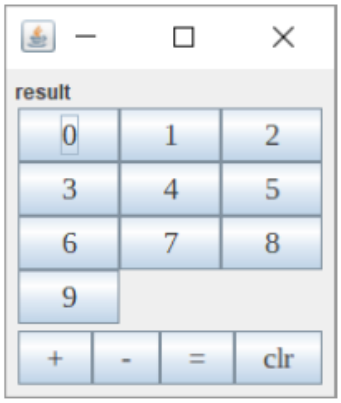



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Convertisseur1 extends JFrame {
JTextField euros, dollars;
public Convertisseur1() {
super("Convertisseur");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.setLayout(new GridLayout(3,2));
euros = new JTextField();
this.add(new JLabel(" euros : "));
this.add(euros);
dollars = new JTextField();
this.add(new JLabel(" dollars : "));
this.add(dollars);
JButton jb = new JButton("\u20ac \u2192 \u0024");
this.add(jb);
/** à Compléter */
```

```
jb = new JButton("\u0024 \u2192 \u20ac");
this.add(jb);
/** à Compléter */
this.pack();
this.setVisible(true);
}
/** à Compléter */
public static void main(String[] args) {
javax.swing.SwingUtilities.invokeLater(new Runnable() {
public void run() { new Convertisseur1(); }
});
}
}
```

Activité 4 :

Compléter la classe Calculatrice correspondant à l'interface, en négligeant la gestion des événements (il faut uniquement créer et placer les composants).

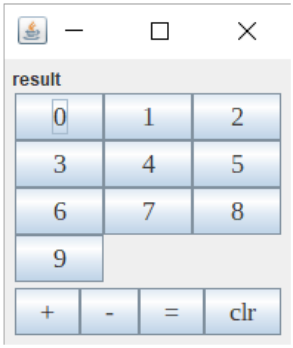



```
import javax.swing.*;
import java.awt.*;
import javax.swing.border.*;
public class Calculatrice extends JFrame {
public Calculatrice() {
super("Calculatrice");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
Font f = new Font("Liberation Serif",Font.PLAIN,18);
UIManager.put("Button.font", f);
UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
/** A compléter*/
this.pack();
this.setVisible(true);
}
public static void main(String[] args) {
javax.swing.SwingUtilities.invokeLater(new Runnable() {
public void run() { new Calculatrice(); }
```

```
});
}
}
```

Activité 5 :

Compléter la classe Calculatrice2 correspondant à l'interface, en négligeant la gestion des événements (il faut uniquement créer et placer les composants).




```
import javax.swing.*;
import java.awt.*;
import javax.swing.border.*;
public class Calculatrice2 extends JFrame {
public Calculatrice2() {
super("Calculatrice");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
/** A COMPLETER */
this.pack();
this.setVisible(true);
}
public static void main(String[] args) {
javax.swing.SwingUtilities.invokeLater(new Runnable() {
public void run() { new Calculatrice2(); }
});
}
}
```

Activité 6 :

Placez le code suivant dans un fichier Personne.java, compilez et exécutez-le.

```
class Personne{
String nom;
int age;
Personne(String nom, int age){
this.nom = nom;
this.age = age;
}
String getNom(){ return this.nom; }
```

```
int getAge(){ return this.age; }
void setAge(int a){ this.age = a; }
}
```

Normalement, on crée un fichier par classe (modularité). C'est même obligatoire si les classes sont publiques.

Pour qu'il se passe quelque chose, ajoutez une méthode principale qui crée 2 personnes, affiche leurs noms et âges, ajoute 1 à leurs âges, et réaffiche leurs âges.

Activité 7 :

Soit le programme suivant :

```
package TP7;

public class Factoriel {

public static void main(String[] args) {

// TODO Auto-generated method stub

int f=1;

int n=Integer.parseInt(args[0]);

for(int i=2;i<=n;i++)

f=f*i;

System.out.println("la valeur de factoriel est:"+f);
```

Il s'agit de reprendre ce programme pour l'améliorer en attrapant les différentes exceptions de façon à préciser la difficulté à l'utilisateur, lorsque le calcul de la factorielle est impossible.

Les exceptions à prévoir sont :

- ArrayIndexoutOfBoundsException : se lance si l'accès à un tableau se fait avec un indice erroné (indice négatif ou supérieur à la taille du tableau)
- NumberFormatException : se lance lorsque le paramètre n'est pas entier.
- NombreNegatifException : se lance lorsque le paramètre est négatif.
- Donner le code source permettant de gérer l'exception NumberFormatException
- Donner le code source permettant de gérer l'exception ArrayIndexoutOfBoundsException.
- Donner le code source permettant de gérer l'exception NombreNegatifException

Activité 8 :

Ecrire un programme permettant la gestion simplifiée des comptes bancaires d'une banque. On utilisera une classe compte avec les attributs numéro de compte, solde, un constructeur et les méthodes créditer, débiter et afficher, puis une classe TestBanque pour servir de classe de lancement.

Exemple d'affichage

```
Creation du compte c1
Creation du compte c2
Affichage des comptes
Compte Num : 1000
Solde   =  : 1000
Compte Num : 1300
Solde   =  : 1000
Operation de credit de 500 francs sur c1
Compte Num : 1000
Solde   =  : 1500
Operation de debit de 2000 francs sur c2
Operation impossible : solde insuffisant
```

Modifier la classe compte en ajoutant dans la classe compte un attribut titulaire. On définit alors une classe compte avec les attributs et les méthodes nécessaires, une classe Personne avec les attributs et les méthodes nécessaires, et enfin une classe TestBanque pour servir de classe de lancement.

Exemple d'affichage

```
Creation de la personne p1
Creation de la personne p2
Creation du compte c1 pour Monsieur Bon
Creation du compte c2 pour Monsieur Celere
Afichage des comptes
Compte NUM : 1000
Solde     : 1000
Compte NUM : 1000
Solde     : 1000
Operation de credit de 500 francs sur c1
Compte NUM : 1000
Solde     : 1500
Operation de debit de 000 francs sur c2
Operation impossible : solde insuffisant
```

Activité 9 :

On désire représenter les nombres rationnels sous la forme d'une fraction de 2 entiers.

On écrira des constructeurs permettant d'instancier des nombres rationnels, de manière à obtenir la forme canonique (c'est à dire sous la forme d'une faction qui ne peut pas être simplifiée).

Pour simplifier la fraction, on divisera les 2 entiers par leur PGCD.

On écrira donc une fonction PGCD utilisant l'algorithme d'Euclide.

Rappel : Calcul du PGCD de deux entiers naturels m et n tels que m > n.
Itérer les manipulations suivantes :
effectuer la division euclidienne de m par n. Soit r le reste.
Remplacer m par n et n par r. On a n > r d'après la définition de la division euclidienne.
Le PGCD est le dernier reste non nul. (Si n divise m, le PGCD est n)

On voudrait récupérer le numérateur ou le dénominateur, afficher un rationnel, tester l'égalité de deux rationnels, d'additionner, de multiplier, de diviser des rationnels, de calculer l'inverse.

Activité 10 :

Le but de cette activité est de vérifier si une expression est correctement parenthésée.

Voici quelques exemples :

((a+b)-(c+d))	Expression Correcte
((a+b)+c	Expression InCorrecte
a+(b+(c+d))	Expression Correcte
)a+b(	Expression InCorrecte

Ecrire une classe *VerifParentheses* qui est composée d'une String pour mémoriser l'expression. Ecrire un constructeur pour lequel on passe une chaîne de caractères en paramètre.

Encapsuler la String en privé et écrire une méthode String getString();

3 cas peuvent se présenter :

1. L'expression est correcte
2. L'expression est Incorrecte :
    - Il y a des parenthèses ouvrantes en trop
    - Il y a des parenthèses fermantes en trop

Votre programme doit lever l'exception en cas d'erreur.

Activité 11 :

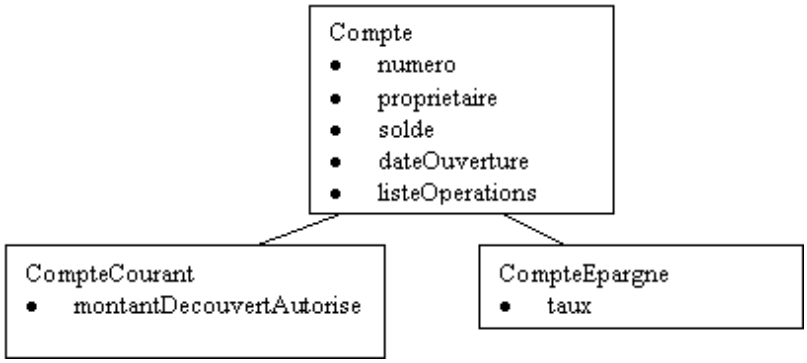
Dans cette activité, on s'intéresse à la gestion des comptes d'une banque.

Une opération bancaire est définie par le montant de l'opération (positif ou négatif), la date de cette opération et un libellé, par exemple :

10/06/2021 | cheque n° 123456 | -53,60 €

Ecrire une classe Operation permettant de modéliser de telles opérations : définir les attributs nécessaires, qui seront encapsulés ; écrire un constructeur ; les assesseurs (méthodes get…), et redéfinir la méthode toString

On simplifiera les types de comptes bancaires en considérant le diagramme de classes suivant :



Ecrire chacune de ces classes. Choisir le type de chaque attribut, en expliquant plus particulièrement le choix de la structure pour représenter sous la forme d'une liste l'historique des opérations effectuées sur un compte.

Ecrire pour chaque classe un ou plusieurs constructeurs appropriés.

Encapsuler chaque attribut, et écrire une méthode get_ correspondant à chaque attribut.

Redéfinir la méthode toString de chaque classe permettant de construire une String représentant les informations caractérisant un de ces comptes bancaires.

Dans la classe Compte, écrire 2 méthodes verser(float montant, String libelle) et retirer(float montant, String libelle) permettant de verser et de retirer de l'argent sur le compte, en créant à chaque fois un objet de la classe Operation qui sera rangé dans la liste des opérations, afin d'en garder un historique.

Ecrire une classe EMontantMinException, héritant de la classe Exception qui pourra être déclenchée par la méthode retirer si on essaye de retirer plus d'argent que le solde du compte ou le montant du découvert autorisé.

NB : la méthode retirer devra être définie au niveau de la classe Compte et redéfinie au niveau de la classe CompteCourant pour traiter le cas du découvert autorisé.

Ecrire une méthode float CalculInterets dans la classe CompteEpargne qui calcule les intérêts produits par ce compte (résultat de la fonction). Ces intérêts seront versés sur le compte.

Ecrire une méthode Historique(int nombreOperations)affichant le solde du compte et la succession des dernières " nombreOperations " opérations.

Ecrire un programme Java pour utiliser ces classes.
Ce programme proposera tout d'abord de créer un compte en demandant s'il s'agit d'un compte courant ou d'un compte épargne.

Ensuite ; il proposera ensuite un système de choix permettant d'effectuer une succession d'opérations sur ce compte : versement, retrait, calcul et ajout des intérêts (uniquement pour les comptes épargnes), consultation du solde et de l'historique.

Solution-Type

Nous allons ici présenter des classes proposées pour résoudre ce problème. Des erreurs glissées volontairement dans le code pour permettre à l'étudiant de comprendre le code et déficeler les erreurs.

```
public class BancaireException extends Exception {
  public BancaireException() {
  }
 public BancaireException(String msg) {
     super(msg);
  }
}
```

```
public class Banque {
   public static void main(String[] args) {
      PersonneMorale aricia=new PersonneMorale("Aricia",412908048);
      String[] prenoms={"François"};
      PersonnePhysique francois=new PersonnePhysique("Bonneville",
            prenoms,
            new FrenchGregorianCalendar(1966,13,12));
            CompteEpargne ce=new CompteEpargne(francois,0.2f);
      ce.depot(500,new FrenchGregorianCalendar(),"économie");
      System.out.println(ce.listeOperations(10));
   }
```

```
}
```

```
public interface Client {
          String identification();
}
```

```
import java.util.*;
public class CompteBancaire {
   static long nbComptes=0;
   private final Client proprietaire;
   protected double solde;
   private final long numeroCompte;
   protected TreeSet<OperationBancaire> listeOperations;
   CompteBancaire (Client proprietaire)
   {this.proprietaire=proprietaire;
    nbComptes=nbComptes+1;
    numeroCompte=nbComptes;
    listeOperations=new TreeSet();}
   public double getSolde() {return solde;}
   public long getNumeroCompte() {return numeroCompte;}
   public Client getProprietaire() {return proprietaire;}
   public String toString() {
        return "Compte n°"+numeroCompte+" "+
              proprietaire.identification()+" dispose de "+
              solde+"Euros";
        }
   public double depot(double montant,FrenchGregorianCalendar date,String libelle) {
        solde=solde+montant;
        listeOperations.add(new OperationBancaire(montant,date,libelle));
        return solde;}
   public double retrait(double montant,String libelle, FrenchGregorianCalendar date) throws
BancaireException {
      if (montant> solde) throw new BancaireException("Pas assez d'argent sur le compte");
         else solde=solde-montant;
         listeOperations.add(new OperationBancaire(-montant,date,libelle));
      return solde;
   }
    public String listeOperations(int n){
      Iterator<OperationBancaire> i =listeOperations.iterator();
      int index=0;
      String resultat="";
      while (index<n && i.hasNext()) {
         resultat=resultat+i.next().toString()+"\n";
         index=index+1;
```

```
    }
    return resultat;
  }
}
```

```
public class CompteCourant extends CompteBancaire {
  private double montantDecouvertAutorise;
    public CompteCourant(Client proprietaire, double decouvert) {
    super(proprietaire);
    montantDecouvertAutorise=decouvert;
  }
  public double getMontantDecouvertAutorise() {
    return montantDecouvertAutorise;
  }
  public double retrait(double montant,String libelle, FrenchGregorianCalendar date) throws
BancaireException {
    if (montant> solde+montantDecouvertAutorise) throw new BancaireException("Pas assez
d'argent sur le compte");
       else solde=solde-montant;
    listeOperations.add(new OperationBancaire(montant,date,libelle));
    return solde;
  }
}
```

```
public class CompteEpargne extends CompteBancaire {
  private float tauxInterets;

  public CompteEpargne(Client proprietaire,float taux) {
    super(proprietaire);
    tauxInterets=taux;
  }
  public float getTauxInterets() {
    return tauxInterets;
  }
    public double calculInterets() {
    solde=solde+solde*tauxInterets;
    return solde;}
  }
public class CompteJoint {
  }
```

```
public class FrenchGregorianCalendar extends GregorianCalendar{
  public FrenchGregorianCalendar() {
    super();
  }
```

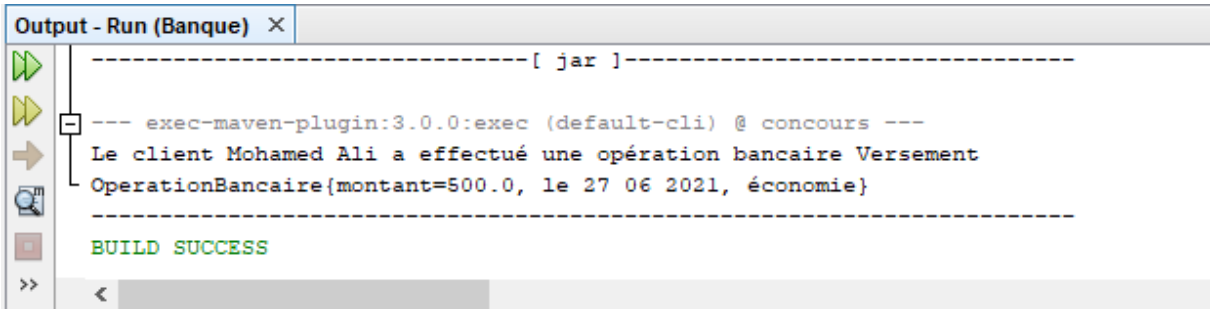
```
    public FrenchGregorianCalendar(int year, int month, int dayOfMonth) {
        super(year, month, dayOfMonth);
    }
     public String toString(String format) {
        SimpleDateFormat sdf=new SimpleDateFormat(format);
        return sdf.format(this.getTime());
    }
  public String toString() {
        return toString("dd mm YYYY");
    }
}
```

```
public class OperationBancaire implements Comparable<OperationBancaire> {
    private final double montant;
    private final FrenchGregorianCalendar dateOperation;
    private final String libelle;
    public OperationBancaire(double montant, FrenchGregorianCalendar dateOperation, String libelle){
        this.montant = montant;
        this.dateOperation = dateOperation;
        this.libelle = libelle;
    }
    public double getMontant() {
        return montant;
    }
    public FrenchGregorianCalendar getDateOperation() {
        return dateOperation;
    }
    public String getLibelle() {
        return libelle;
    }
    public String toString() {
        return "OperationBancaire{montant=" + montant + ", le "+ dateOperation + ", " + libelle + '}';
    }
    public int compareTo(OperationBancaire o) {
        int c=o.dateOperation.compareTo(this.dateOperation);
        if (c==0) return o.libelle.compareTo(this.libelle);
        else return c;
    }
}
```

```
public class PersonneMorale implements Client {
  private String nom;
  private long numeroSiret;
  public String identification() {return nom+" "+numeroSiret;}
```

```
 PersonneMorale (String unNom,long numeroSiret) {
   nom = unNom;
   this.numeroSiret=numeroSiret;
  }
  public String getNom() {
   return (nom);
  }
 public String toString() {return identification();}
}
```

```
import etatcivil.Personne;
public class PersonnePhysique extends Personne implements Client {
   public PersonnePhysique(String nom, String[] prenom, FrenchGregorianCalendar naissance) {
      super(nom, prenom, naissance);
   }
 public String identification() {return toString();}
}
```



Activité 12 :

On souhaite développer une application permettant de gérer une médiathèque qui contient des livres, des CDs et des DVDs. Chacun de ces media sera naturellement modélisé par une classe.

Tous les médias sont décrits par un titre et un nom d'auteur. Les utilisateurs peuvent donner leur avis sur la qualité d'un média en leur attribuant une note comprise entre 0 et 5. Il est possible :

- De représenter un média par une chaine de caractères (méthode toString). Le format de cette représentation est : "Titre" par Auteur
- De donner son avis sur le média grâce à la méthode vote(int note), note devant être comprise entre 0 et 5.
- D'obtenir la moyenne des votes reçus par la méthode moyenneNotes(). Si le média n'a reçu aucune note, cette méthode renvoie 0.

Les DVD possède une caractéristique supplémentaire permettant la gestion des

zones (une zone est décrite par un entier compris entre 0 et 8, la zone 0 indique que le DVD est lisible dans toutes les zones).

La méthode readable(int[] zones) renvoie true si le DVD est lisible dans une des zones passées en paramètre, false sinon.

La description du CD comporte également le format du contenu décrit par une chaine de caractères ("CD musical", "OGG", ou "MP3"). La méthode toString est modifiée pour indiquer le format et renvoie, par exemple :

"Some Kind Of Trouble" par James Blunt [CD musical]

La description des livres ne comporte aucun élément particulier.

1. Donnez le code de la classe Livre
2. On souhaite maintenant développer la classe CD. Comme cette classe comportera de nombreuses méthodes communes avec la classe Livre, on souhaite introduire une classe mère générique, Media.
   - Modifiez la classe Livre pour qu'elle hérite d'une classe Media et réorganisez votre code de manière à tirer profit de cette nouvelle organisation.
   - Ecrivez le code de la classe CD.
3. Donnez le code de la classe DVD.

Remarque : dans tous les constructeurs, le premier paramètre correspondra au nom de l'auteur, le second au titre.

Dans une classe Médiathèque, la base de données de la médiathèque est une ArrayList de Medium.

4. Ecrivez une méthode add(Media m) qui permet d'ajouter un média à la collection.
5. Ecrivez une méthode filtre(String critere, String valeur qui retourne un ArrayList<Media> contenant l'ensemble des média vérifiant le critère passé en paramètre. Ce critère peut porter :
   - sur l'auteur ;
   - sur le titre ;
   - sur le type de média (on pourra utiliser l'instruction nomObjet instanceof nomClasse qui retourne true si l'objet nomObjet est une instance de la classe nomClasse ou de l'une de ses classes filles)

Toutes les comparaisons ne tiendront pas compte de la casse.

Par exemple : m.filtre("titre", "ToTo") renverra l'ensemble des média dont le titre en minuscule est toto et m.filtre("media", "CD") renverra l'ensemble des CD de de la collection.

Solution-Type :

```
public class CD extends Media {
   private String type;
      public CD(String author, String title, String type) {
           super(author, title);
           this.type = type;
   }
   @Override
   public String toString() {
           return super.toString() + " [" + this.type + "]";
   }
}
```

```
public class DVD extends Media {
      private int zone;
      public DVD(String author, String title, int zone) {
           super(author, title);
           this.zone = zone;
   }
      public boolean readable(int[] zones) {
           if (this.zone == 0) {
              return true;
           }
           for (int z : zones) {
              if (this.zone == z) {
                      return true;
              }
           }
           return false;
   }
}
```

```
public class Livre extends Media {
   public Livre(String author, String title) {
```

```
        super(author, title);
    }
}
```

```
public class Media {
    private String author;
    private String title;
    private int nVotes;
    private int notes;
    public Media(String author, String title) {
        this.title = title;
        this.author = author;
        this.notes = 0;
        this.nVotes = 0;
    }
    public String toString() {
        return "\"" + this.title + "\" par " + this.author;
    }
    public double moyenneNotes() {
        if (this.nVotes == 0) {
            return 0.0;
        } else {
            return this.notes / this.nVotes;
        }
    }
    public void vote(int note) {
        if (note < 0 || note > 5) {
            return;
        }
        this.notes += note;
    }
    public boolean isSame(String critere, String valeur) {
        if (critere.equals("titre") && this.title.equals(valeur)) {
```

```
            return true;
        }

        if (critere.equals("auteur") && this.author.equals(valeur)) {
            return true;
        }

        if (critere.equals("media")) {
            if (valeur.equalsIgnoreCase("livre")) {
                return this instanceof Livre;
            } else if (valeur.equalsIgnoreCase("dvd")) {
                return this instanceof DVD;
            } else if (valeur.equalsIgnoreCase("cd")) {
                return this instanceof CD;
            }
        }
        return false;
    }
    }
```

```
import java.util.ArrayList;
import java.util.List;
public class Mediatheque {
    private List<Media> collection;
        public Mediatheque() {
            this.collection = new ArrayList<Media>();
    }
        public List<Media> filtre(String critere, String valeur) {
            ArrayList<Media> res = new ArrayList<Media>();
                    for (Media el : this.collection) {
            if (el.isSame(critere, valeur)) {
                    res.add(el);
            }
```

```
        }
        return res;
    }
     public void add(Media el) {
        this.collection.add(el);
    }
      public static void main(String[] a) {
        Mediatheque m = new Mediatheque();
        m.add(new Livre("a", "b"));
        for (Media el : m.filtre("media", "livre")) {
           System.out.println(el);
        }
    }
}
```

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class TestCD {
    @Test
    public void testCD() {
        String expectedRes = "\"Some Kind Of Trouble\" par James Blunt [CD musical]";
        CD c = new CD("James Blunt", "Some Kind Of Trouble", "CD musical");
        assertEquals(expectedRes, c.toString());
    }
}
```

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestDVD {
    @Test
    public void testReadable() {
        DVD d1 = new DVD("Lucas", "Starwars", 0);
        assertTrue(d1.readable(new int[] {1, 2, 3}));
        DVD d2 = new DVD("Lucas", "Starwars", 2);
```

```
        assertTrue(d2.readable(new int[] {1, 2, 3}));
        DVD d3 = new DVD("Lucas", "Starwars", 2);
        assertFalse(d3.readable(new int[] {1, 3, 4}));
    }
}
```

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestLivre {
    @Test
    public void testLivre() {
        Livre l = new Livre("Sartre", "La Nausée");
        assertEquals("\"La Nausée\" par Sartre", l.toString());
    }
      @Test
    public void testVote() {
        Livre l = new Livre("Sartre", "La Nausée");
        assertEquals(0, l.moyenneNotes());
        l.vote(1);
        l.vote(2);
        assertEquals(1/3, l.moyenneNotes(), 0.000001);
    }
      @Test
    public void ignoreNote() {
        Livre l = new Livre("Sartre", "La Nausée");
        l.vote(1);
        l.vote(2);
        l.vote(12);
        assertEquals(1/3, l.moyenneNotes(), 0.000001);
    }
}
```

```
import static org.junit.Assert.assertEquals;
import java.util.ArrayList;
```

```
import java.util.List;
import org.junit.Test;
public class TestMediatheque {
   @Test
   public void testFilterAuteur() {
         Mediatheque m = new Mediatheque();
         CD cd1 = new CD("Queen", "The show must go on", "mp3");
         CD cd2 = new CD("James Blunt", "Some Kind Of Trouble", "CD musical");
         Livre l1 = new Livre("Sartre", "La Nausée");
         Livre l2 = new Livre("Sartre", "L'enfer");
         Livre l3 = new Livre("Baudelaire", "Le spleen de Paris");
         m.add(l1);
         m.add(l2);
         m.add(l3);
         m.add(cd1);
         m.add(cd2);
         ArrayList<Media> cds = new ArrayList<Media>();
         cds.add(cd1);
         cds.add(cd2);
         ArrayList<Media> sartre = new ArrayList<Media>();
         sartre.add(l1);
         sartre.add(l2);
         assertEquals(sartre, m.filtre("auteur", "Sartre"));
   }
   @Test
   public void testFilterType() {
         Mediatheque m = new Mediatheque();
         CD cd1 = new CD("Queen", "The show must go on", "mp3");
         CD cd2 = new CD("James Blunt", "Some Kind Of Trouble", "CD musical");
         Livre l1 = new Livre("Sartre", "La Nausée");
         Livre l2 = new Livre("Sartre", "L'enfer");
         Livre l3 = new Livre("Baudelaire", "Le spleen de Paris");
```

```
        m.add(l1);
        m.add(l2);
        m.add(l3);
        m.add(cd1);
        m.add(cd2);
        ArrayList<Media> cds = new ArrayList<Media>();
        cds.add(cd1);
        cds.add(cd2);
        ArrayList<Media> sartre = new ArrayList<Media>();
        sartre.add(l1);
        sartre.add(l2);
        assertEquals(cds, m.filtre("media", "CD"));
    }
    @Test
    public void testFilterTitre() {
        Mediatheque m = new Mediatheque();
        CD cd1 = new CD("Queen", "The show must go on", "mp3");
        CD cd2 = new CD("James Blunt", "Some Kind Of Trouble", "CD musical");
        Livre l1 = new Livre("Sartre", "La Nausée");
        Livre l2 = new Livre("Sartre", "L'enfer");
        Livre l3 = new Livre("Baudelaire", "Le spleen de Paris");
        m.add(l1);
        m.add(l2);
        m.add(l3);
        m.add(cd1);
        m.add(cd2);
        ArrayList<Media> cds = new ArrayList<Media>();
        cds.add(cd2);
        assertEquals(cds, m.filtre("titre", "Some Kind Of Trouble"));
    }
}
```

Références

1. Claude Delannoy, Programmer en Java, Eyrolles,2004**.**
2. Claude Delannoy, Exercices en Java, Eyrolles,2017**.**
3. Houda EL BOUHISSI, Programmation Orientée Objet avec java, les pages bleues, 2020.
4. Luc GERVAIS, Apprendre la Programmation Orientée Objet avec le langage Java, Eni, 2020.
5. Vincent Granet, Jean-Pierre Regourd, Aide-Mémoire-java, Dunod , 2015.

▪ **Quelques liens vers des pages Web :**

- https://openclassrooms.com/courses/apprenez-a-programmer-en-java
- https://www.jmdoudoux.fr/java/dej/chap-poo.htm
- http://www.supinfo.com/articles/single/967--initiation-programmation-orientee-objet-poo
- https://florat.net/tutorial-java/chapitre01.html
- https://home.mis.u-picardie.fr/~furst/prog_objet.php
- https://www.ukonline.be/cours/java/apprendre-java