Préparation Java pour le T.P. Système Distribué (Informatique : Master 1)

Programmation Orientée Objet

- Paradigme ***impérative*** (Structurelle / procédurale) V.S. ***P.O.O.*** (Programmation Orienté Objet)
- Définition des classes d'objets (*Analyse des besoins et du domaine d'application – classes techniques*)
- Définition des attributs / propriétés des objets
  - Attributs des instances
  - Attributs des classes d'objets (communs à toutes les instances comme les *constantes*)
  - Attributs sont toujours invisibles (*privés* à la classe : sauf les constantes qui sont publiques)
- Définition des opérations / méthodes des objets (responsabilité de chaque classe)
  - Traitements spécifiques aux instances
  - Traitements spécifiques aux classes d'objets
  - Méthodes sont, *en général*, visibles (*publiques* ou *protégées*).
- Relations entre les classes
  - Héritage
  - Agrégation et Composition (une composition est une agrégation forte)
  - Association / relations de domaine d'application
- Communication entre objets
  - Messages (appels de méthodes)
  - Etat d'un objet
  - Coopérations entre les objets → Solution du problème
- Programmation parallèle / Réseaux
  - Threads : programmation parallèle et concurrente
    - Communication entre les threads à travers la mémoire du processus les contenant
    - Synchronisation entres les threads : exclusion mutuelle, point de rendez-vous, etc.
  - Sockets : Communication entres des processus (programmes en cours d'exécution)
    - Pas de mémoire partagée
    - Les processus s'exécutent sur des machines différentes (sites différents)
    - Plus adapté aux systèmes distribués (à utiliser pour le T.P.)

Conventions d'écriture du code source Java

- Chaque classe commence par une Majuscule
- Nom de classe composé de plusieurs noms
- Chaque classe est définie dans un fichier java portant le même nom de la classe
- Un fichier java contient une seule classe (*pas obligatoire*)
- Les attributs sont ***private*** et les méthodes sont ***public***
- Les constantes sont toujours en majuscule (**public** et ***static***).
- Les méthodes et attributs commencent avec une lettre minuscule
- Nom de méthodes et attributs composé de plusieurs noms.
- Regrouper les classes dans des packages suivant une certaine logique
- Commenter les classes, attributs et méthodes (/** */ pour le JavaDoc)

Les critères du bon programme

- Respect des conventions citées ci-dessus
- Bonne conception du domaine (du problème)
- L'interactivité, la convivialité et la facilité d'utilisation de l'interface graphique de l'application

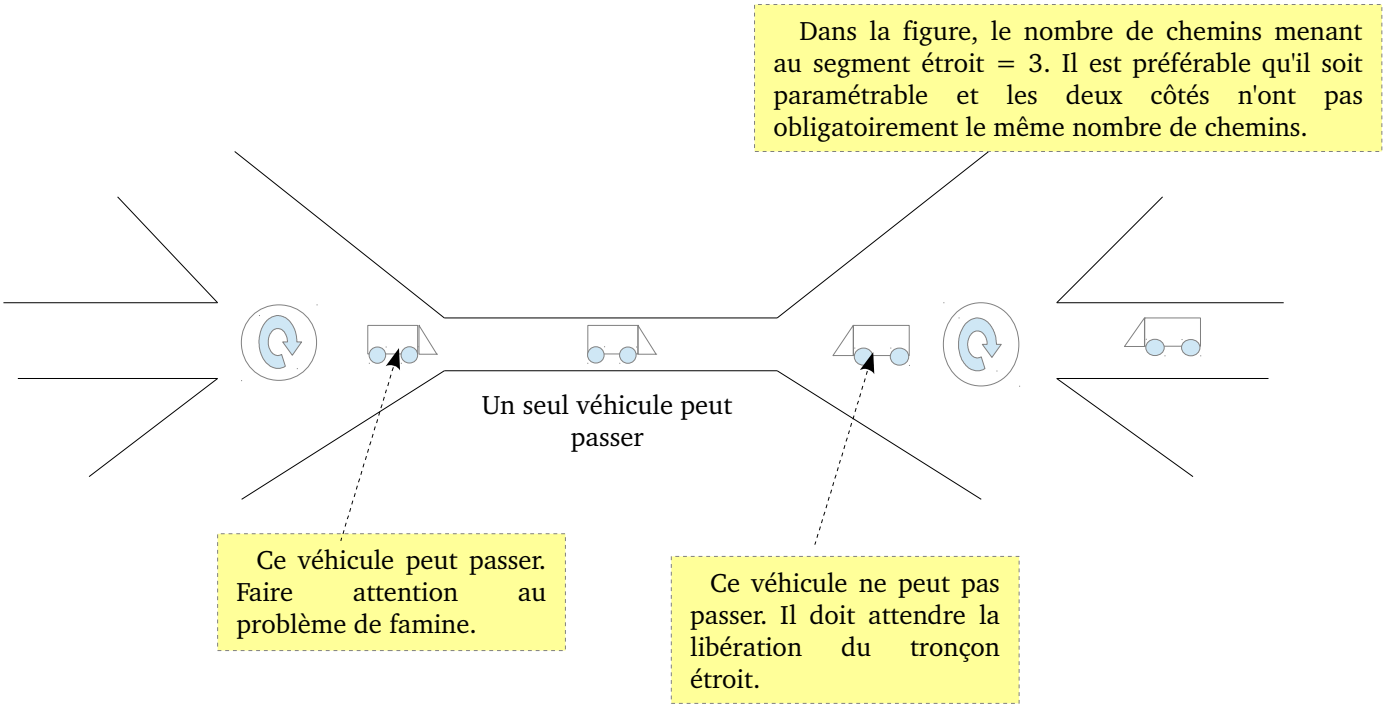
Pour les besoins du TP, nous utiliserons l'éditeur Eclipse (à télécharger et à installer)

- Ecrire tous le code source sans utiliser un générateur automatique
- Pour les interfaces graphiques, n'utiliser pas un éditeur graphique visuelle (WYSIWYG : What You See Is What You Get)
- Utiliser principalement la bibliothèque ***Swing*** et partialement ***AWT*** pour créer les interfaces graphiques
- Utilisation des sockets pour communiquer entre les différents processus (différents sites)
- Définition des différents messages entre les processus
- Bien modéliser la réalité du problème à résoudre (domaine d'application)

Énonce du T.P. Système Distribué
Informatique : Master I
2013 / 2014

Problème de carrefour

La figure suivante illustre une situation conflictuelle de circulation de véhicule : un tronçon d'une route étroit permettant le passage aux véhicules venant de deux côtés (gauche et droite).



Si un véhicule *a* veut accéder au segment alors qu'il y a au moins un autre véhicule, venant du sens inverse sur le tronçon, alors ce véhicule *a* doit attendre jusqu'à que le tronçon soit libéré.

Travail Demandé

le ***développement d'une application Java*** permettant d'émuler cette situation en suivant les règles suivantes :

- ✔ chaque véhicule est implémenté sous forme d'un processus
- ✔ la seule façon pour la communication entre les processus est l'échange de message (pas de mémoire partagée, pas de fichiers, *etc.*)
- ✔ utiliser les sockets pour faire communiquer les processus
- ✔ la durée nécessaire pour le passage d'un véhicule sur le tronçon est aléatoire (entre *n* et *m* unité de temps)
- ✔ un véhicule accédant à l'un des ronds-points (gauche ou droite), il un nombre de choix :
  - ◆ soit il retourne sur la route par laquelle il est venu
  - ◆ soit il choisit d'accéder au segment étroit : *dans ce cas il y a une vérification*
  - ◆ soit il retourne sur l'un des chemins restants
- ✔ une bonne application doit être paramétrable (le nombre de routes menant aux ronds-points, la durée nécessaire pour le passage, *etc.*)
- ✔ accompagné l'application d'un compte-rendu (entre 5 et 10 pages) montrant les fonctionnalités à *implémenter* et *effectivement implémentées*, l'*analyse* et la *conception* de l'application ainsi que les idées sous-adjacentes utilisées.