



# TP1 : Initiation à Java et Eclipse

Systèmes d'Exploitation Avancés

## I. Objectifs du TP

Ce TP est une introduction au langage Java. Il vous permettra de comprendre les principes de base du langage, ainsi que la syntaxe dont on aura besoin pour le reste du programme. Il ne prétend pas être exhaustif, mais donne une idée assez complète sur les notions à connaître pour pouvoir programmer en Java. Plus tard, je vous invite à étudier plus en profondeur ce langage.

Pour pouvoir programmer en Java, on a besoin d'un JDK (*Java Development Kit*), qui représente l'environnement dans lequel le code Java est compilé afin que la machine virtuelle (*JVM* pour *Java Virtual Machine*) puisse l'interpréter. Il contient le compilateur Java (*javac*), l'archivageur (*jar*), le générateur de documentation (*javadoc*) et le débogueur (*jdb*). Il contient également l'environnement d'exécution Java (*JRE* pour *Java Runtime Environment*).

Pour compiler et exécuter un programme en Java, on peut appeler les commandes directement sur une console (*javac* pour la compilation et *java* pour l'exécution) ou bien utiliser un IDE (*Integrated Development Environment*). Un IDE est un programme regroupant un ensemble d'outils pour le développement de logiciels. Cet IDE, en collaboration avec le JDK, permet de faciliter la réalisation de toutes les étapes de compilation et d'exécution, grâce à une interface graphique dédiée. Il existe plusieurs IDE qu'on peut utiliser (Netbeans, Eclipse, JDeveloper, JBuilder...), nous allons en utiliser un pour cette année : Eclipse. La première partie de ce TP vous montre l'environnement Eclipse pour une utilisation de base, mais le but est loin de vous apprendre à l'utiliser en profondeur : Eclipse reste juste un outil, qu'on peut remplacer à tout moment par un autre.

## II. Environnement de Développement : Eclipse

Eclipse est un environnement de développement intégré libre extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT, d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

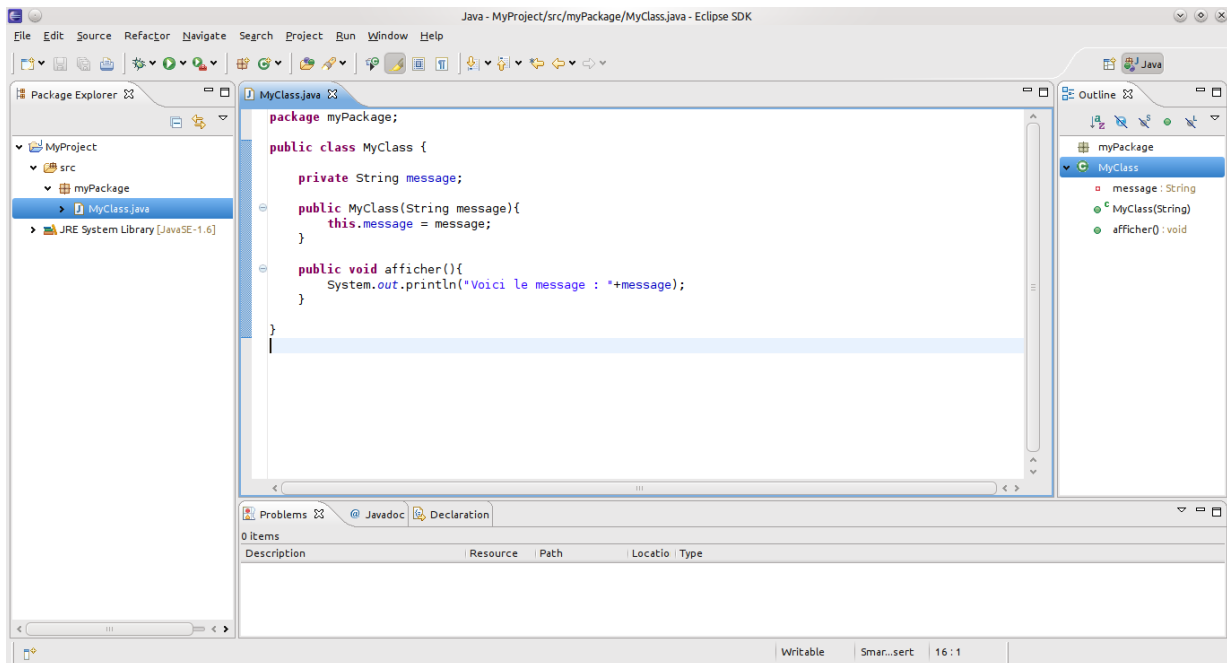
La spécificité d'Eclipse IDE (*Integrated Development Environment*) vient du fait de son architecture totalement développée autour de la notion de plugin (en conformité avec la norme OSGi) : toutes les fonctionnalités de cet atelier logiciel sont développées en tant que plug-in.



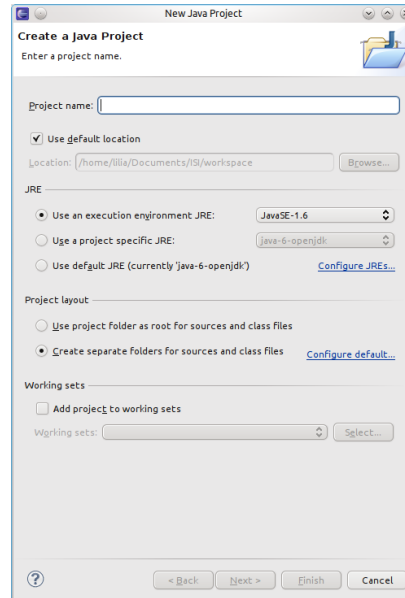
Plusieurs logiciels commerciaux sont basés sur ce logiciel libre, comme par exemple IBM Lotus Notes 8, IBM Symphony ou WebSphere Studio Application Developer.



## II. 1. Création d'un Projet sur Eclipse

- Ouvrez l'IDE Eclipse, on vous demandera de choisir l'espace de travail dans lequel vous allez trouver vos projets. Tapez le chemin du répertoire que vous avez créé sur le bureau. Si vous n'avez pas encore créé d'espace de travail, Eclipse le créera pour vous.
- Fermez la fenêtre de bienvenue qui apparaît.
- Vous vous trouvez actuellement dans votre espace de travail. La figure suivante vous indique les différentes vues disponibles.



- Créez un nouveau projet Java. Pour cela, appuyez sur File → New → Java Project. La fenêtre suivante apparaîtra:




- Tapez comme indiqué le nom de votre projet. Conservez les réglages par défaut, et appuyez sur Finish.
- Dans l'espace de travail, sous l'onglet Packages, vous verrez que le projet est créé, et qu'il contient déjà un répertoire *src* (qui doit contenir tous les fichiers source que vous créez), et *JRE System Library*, qui est utilisée pour compiler votre code.
- Pour créer un package sous le répertoire *src*, cliquer sur celui-ci, puis sur l'icône , ou clic-droit sur *src*, et choisir New -> Package. Choisissez un nom pour le package.
- Pour créer une classe :
  - Cliquer sur le package qui doit contenir la classe puis sur l'icône , ou clic-droit sur le package, et choisir New -> Class. Dans la fenêtre qui apparaît, choisissez le nom de la classe.
  - Si la classe n'est pas définie dans un package (ce qui est déconseillé), refaites l'opération ci-dessus à partir du répertoire *src*.
  - Vous pouvez générer automatiquement la méthode *main* en cliquant sur la case *public static void main(String[] args)*.
- La classe générée apparaîtra sous le package que vous avez choisi. Double-cliquez dessus pour modifier son code dans la partie édition. Vous verrez qu'un squelette de la classe vous est proposé. Vous n'aurez qu'à terminer le reste du code.
- Avec Eclipse, vous n'avez pas besoin de compiler explicitement votre code : la compilation se fait en temps réel. De plus, les erreurs de syntaxe seront affichées pendant l'écriture du code, avec des propositions de corrections.



## II. 2. Nouveau projet : Helloworld

### II. 2. 1. Helloworld – Version simplifiée

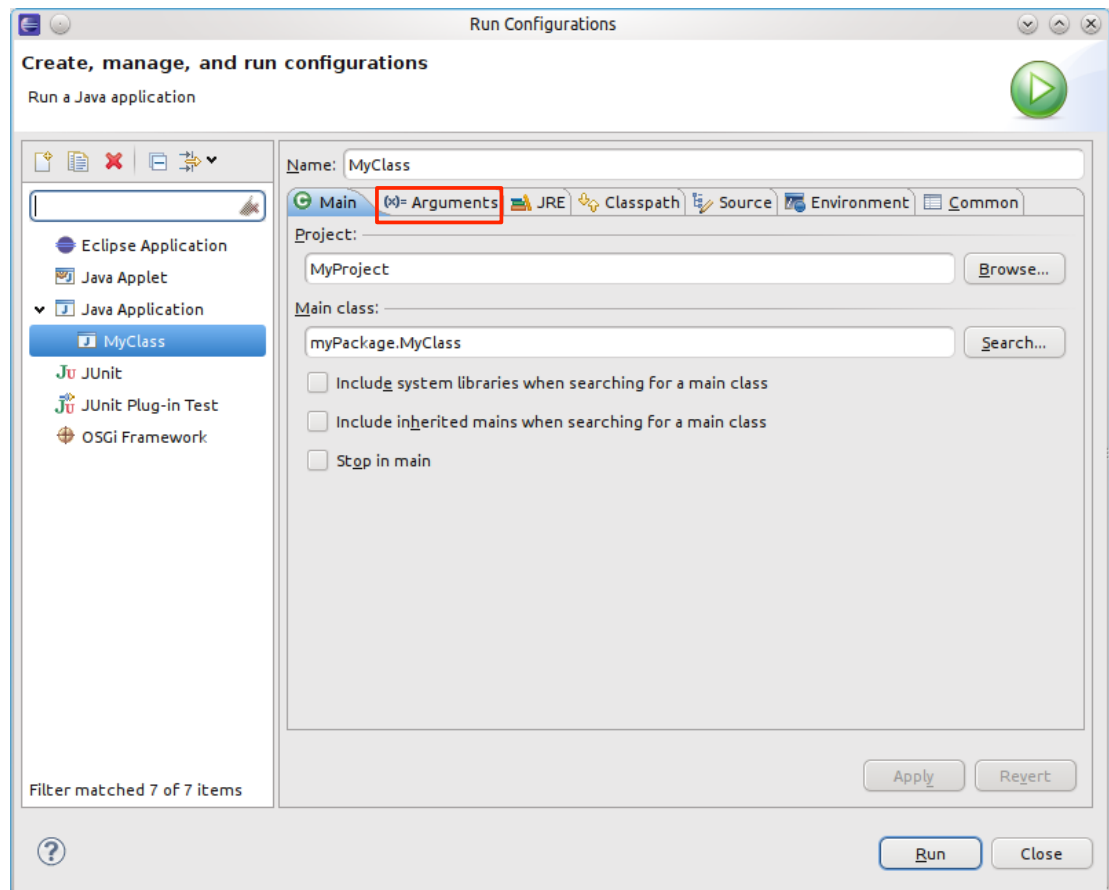
- Créer un nouveau projet Helloworld comme indiqué dans la partie précédente. Créer un package nommé *helloPack*, contenant une classe *Helloworld* qui contient une méthode *main*.
- Dans la méthode *main*, écrire `"System.out.println("Hello World!");"`
- Exécuter votre programme en cliquant directement sur l'icône . L'affichage apparaîtra dans la partie inférieure, sous l'onglet Console.


### II. 2. 2. Helloworld – Ajout d'arguments

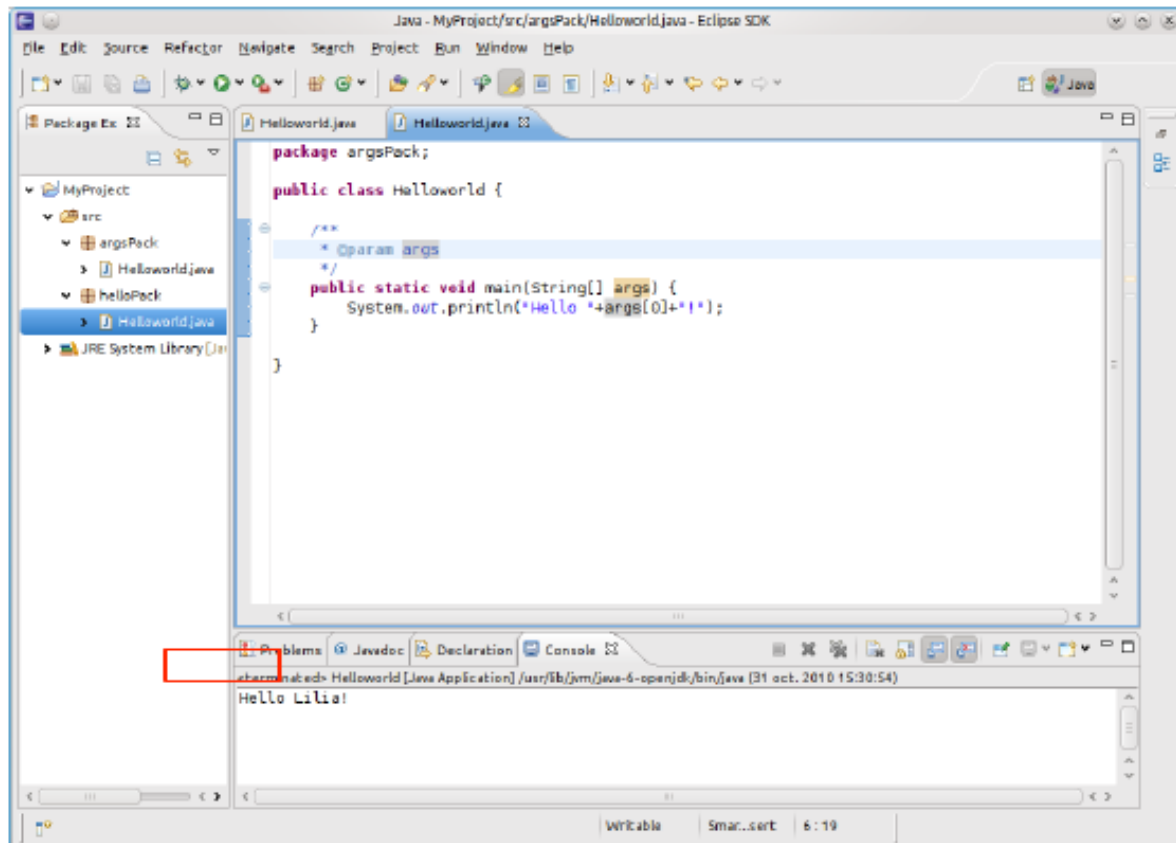
Dans cette partie, nous allons ajouter des arguments à la classe.

- Créer dans le même projet Helloworld, un nouveau package nommé *argsPack*, contenant une classe *Helloworld* avec une méthode *main*.
- Dans le code de la méthode *main*, écrire : `"System.out.println("Hello "+args[0]+"!");"`

- Pour définir des arguments à la classe, cliquer sur la flèche à côté de l'icône d'exécution et sélectionner *Run Configurations*, ou cliquer sur la classe que vous voulez exécuter, et aller à Run → Run Configurations... La fenêtre suivante va apparaître.



- Dans la partie de gauche, sélectionner *Java Application*, puis cliquer sur l'icône  (en haut, à gauche), pour ajouter une nouvelle configuration. Vous verrez que votre classe HelloWorld a été ajoutée sous *Java Application*.
- Sélectionner l'onglet *Arguments* (ci-dessus encadré en rouge) et, dans le cadre *Program Arguments*, tapez simplement votre nom.
- Cliquer ensuite sur *Run*. Vous verrez dans la console l'affichage "**Hello votre\_nom!**".



### Quelques manipulations du code Java

Vous pouvez maintenant écrire le code de la classe. Vous remarquerez qu'il est directement indenté. Si vous copiez du code non indenté ou si vous voulez rafraîchir l'indentation : sélectionner la partie concernée, clic droit → source → correct indentation (ou ctrl +I).

**Eclipse corrige quelques erreurs de compilation** Créer une variable `clavier` de type `Scanner` :

- une petite croix apparaît en début de ligne pour signaler un problème (qui est explicité en passant la souris sur la croix)
- une petite lumière signale une solution
- cliquer sur la lumière (clic gauche) et choisir la solution appropriée (ici : `import java.util.Scanner`)

**Complétion sémantique** Dans le corps de votre méthode `main`, tapez `clavier`. (c'est-à-dire "clavier" suivi de '.'). La liste des méthodes définies pour l'objet `clavier` est proposée; il suffit de choisir celle qui nous intéresse (par exemple `nextInt`) en double-cliquant dessus.

Eclipse nous permettra par la suite d'ajouter automatiquement beaucoup d'autres éléments dans le code et de le modifier facilement, par exemple pour renommer des éléments.



### III. Initiation à Java

Le langage Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld.

#### III. 1. Cas d'étude : Vente aux enchères

Nous allons dans cette partie vous initier aux concepts en base de Java en réalisant une petite application de *vente aux enchères*, qui permet à deux personnes (appelés *enchérisseurs*) de miser sur un objet en vente. Cet objet est défini par un numéro (par exemple 1273), un type (par exemple *tableau*) et une valeur (en dinars) représentant son prix à l'instant présent (par exemple 500). L'objet en vente est initialisé à la valeur que précise le responsable de la vente (appelé *commissaire-priseur*).

Les deux enchérisseurs ont chacun une limite de prix à ne pas dépasser : si le prix de l'objet dépasse cette limite, ils ne pourront pas acheter l'objet. Ils misent une somme à tour de rôle. Le premier dont la limite est atteinte quitte l'enchère, et c'est son adversaire qui acquiert l'objet.

**TAF :** Pour commencer, créer un projet qui s'appelle *VenteAuxEncheres* sur Eclipse. Dans le répertoire *src*, créer un package appelé *encheres* et un package appelé *main*. Dans le package *main*, créer une classe *Main* contenant une méthode *main* qu'on laissera vide pour l'instant.

#### III. 2. Classes et objets

Une classe représente la description informatique des caractéristiques d'un objet abstrait (destiné à modéliser un objet réel). Elle décrit les constituantes élémentaires de l'objet ainsi que les opérations (méthodes) pouvant s'appliquer à cet objet.

```
class Vehicule {  
    int age ;  
    float taille ;  
    float poids ;  
    boolean moteur ;  
  
    int getWeight() {  
        return (poids) ;  
    }  
}
```

Une classe *Vehicule* sauvegardée dans un fichier *Vehicule.java*

Quatre "attributs" : age, taille, poids et moteur

Une "méthode" : *getWeight* retourne le poids d'un véhicule



Un objet est une instance d'une classe. Il est généré à partir d'une classe et stocké dans une variable. Pour créer un nouvel objet, l'opérateur *new* est utilisé :

```
Vehicule v = new Vehicule() ;
```

Conventions de nommage:

1. Une classe commence toujours par une majuscule, tous les autres identifiants commencent en minuscule (les objets, les méthodes, les types primitifs, les variables...)
2. Si un identifiant est composé de plusieurs mots, ces mots sont collés l'un à la suite de l'autre avec une majuscule pour chaque début de mot, mais en respectant la convention 1. Par exemple, pour une classe : *NomDeClasse*, et pour un objet *nomDeLObjet*.

Un constructeur est une méthode spécifique appelée automatiquement lors de l'instanciation d'un objet. Le constructeur a les particularités suivantes :

1. C'est la seule méthode qui porte le nom de la classe
2. C'est la seule méthode qui ne définit pas de variable de retour
3. C'est la seule méthode dont le nom commence par une majuscule (puisque c'est le nom de la classe)

```
class Vehicule {  
    int age ;  
    float taille ;  
    float poids ;  
    boolean moteur ;  
  
    Vehicule() {  
        age = 0 ;  
        taille = 0.0 ;  
        poids = 0.0 ;  
        moteur = false ;  
    }  
  
    int getWeight() {  
        return(poids) ;  
    }  
}
```

Il est possible de définir un ou plusieurs constructeurs avec paramètres en plus du constructeur par défaut (sans paramètres)





```

class Vehicule {
    int age ;
    float taille ;
    float poids ;
    boolean moteur ;

    Vehicule(int a, float t, float p, boolean m) {
        age = a ;
        taille = t ;
        poids = p ;
        moteur = m ;
    }

    ...
}

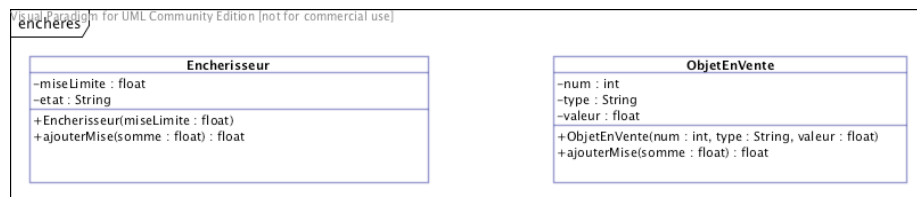
```

```

...
Vehicule v = new Vehicule(2,4.6,1.5,true) ;
...

```

**TAF :** Sous le package *encheres*, créer une classe *Encherisseur* et une classe *ObjetEnVente*, qui respectent le diagramme de classes suivant (pour l'instant, seuls les constructeurs sont à implémenter. Les autres méthodes restent vides). Pour la classe *Encherisseur*, l'attribut *etat* représente l'état de l'enchérisseur, s'il est encore en lice pour l'objet ou pas, et est initialisé à « *in* ».



### III. 3. Encapsulation

Il est possible de limiter la visibilité des attributs et méthodes définies à l'intérieure des classes.

- **public** : pas de limitation
- **protected** : visible depuis l'intérieur de la classe et depuis classes qui en héritent
- **private** : visible uniquement depuis intérieur de la classe



```
public class Vehicule {  
    public float age ;  
    protected float taille ;  
    private int poids ;  
    private boolean moteur ;  
  
    public Vehicule() {  
        age = 1.0F ;  
        taille = 1.0F ;  
        poids = 1 ;  
        moteur = false ;  
    }  
  
    public int getWeight() {  
        return(poids) ;  
    }  
}
```

La déclaration *public* des classes elles-mêmes répond à un impératif de sécurité, par une déclaration *public* elles seront rendues accessibles aux autres classes.

**TAF :** Ajouter les modificateurs de visibilité aux classes que vous avez créé de manière à ce que les méthodes soient visibles par tout le monde et les attributs privés.

### III. 4. Héritage

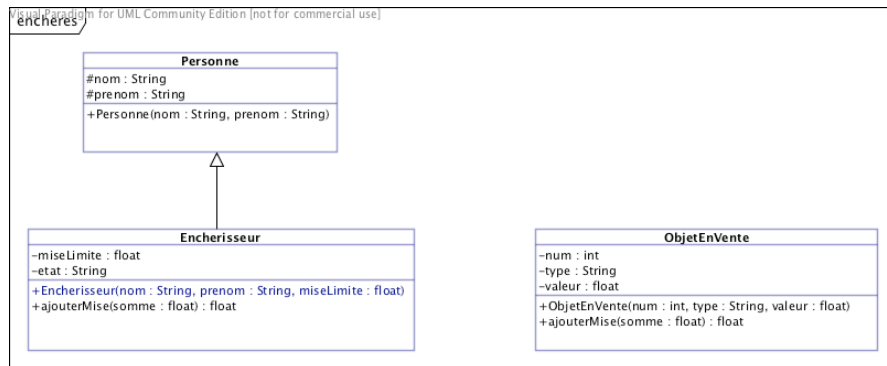
Il est possibilité de décrire une nouvelle classe en la faisant hériter (dériver) d'une autre classe. Cela implique que les attributs et méthodes sont hérités de la classe mère (superclasse) et des superclasses de celle-ci jusqu'à arriver à la classe *Object*, dont héritent implicitement toutes les classes.

```
public class Voiture extends Vehicule {  
    public int places ;  
  
    public Voiture() {  
        moteur = true ;  
        places = 4 ;  
    }  
}
```



**Important :** Il est impossible de faire dériver une classe de plus d'une autre classe

**TAF :** Ajouter une classe *Personne* dont hérite la classe *Encherisseur*, comme défini dans le diagramme suivant. Modifier le constructeur de la classe *Encherisseur* pour qu'il prenne en considération le constructeur de sa classe mère.



### III. 5. Association

Deux classes sont en association lorsque certains de leurs objets ont besoin s'envoyer des messages pour réaliser un traitement. En Java, une association est représentée par l'ajout une référence d'une classe comme attribut dans l'autre.

```

public class Vehicule {
    public float age ;
    protected float taille ;
    private int poids ;
    private boolean moteur ;

    private Personne proprietaire ;

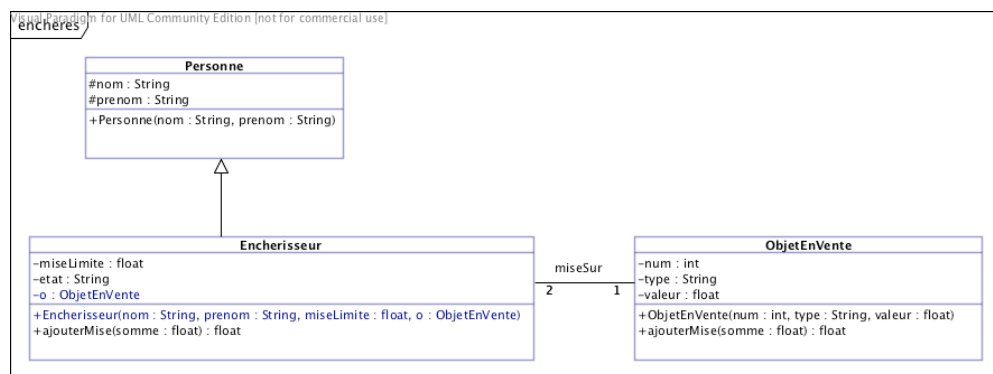
    public Vehicule() {
        age = 1.0F ;
        taille = 1.0F ;
        poids = 1 ;
        moteur = false ;
    }

    public int getWeight() {
        return(poids) ;
    }
}
  
```

Ainsi, dans l'exemple suivant, une association est réalisée entre un objet de type *Voiture* et un objet de type *Personne*. Chaque voiture a maintenant une instance de *Personne* qui en est le propriétaire.

### TAF :

1. Réaliser une association entre la classe *Encherisseur* et la classe *ObjetEnVente*, comme l'indique le diagramme suivant.



2. Implémenter toutes les méthodes manquantes dans les classe *Encherisseur* et *ObjetEnVente* de manière à ce que :
  - Quand un enchérisseur veut ajouter une somme à sa mise, il teste d'abord si la valeur de l'objet en vente a atteint sa mise limite. Si c'est le cas, il abandonne la partie en affichant : « l'enchérisseur *nom\_encherisseur* a abandonné la partie » et met son état à *out*. Si ce n'est pas le cas, il appelle la méthode *ajouterMise* de l'objet en vente, et met la somme qu'il propose comme paramètre.
  - Quand la méthode *ajouterMise* de la classe *ObjetEnVente* est appelée, la somme passée en paramètre sera ajoutée à l'attribut *valeur*.
  - Vous pouvez ajouter d'autres méthodes si besoin est.

## III. 6. Définition d'une variable partagée

Pour comprendre comment partager une variable en Java, il faut se mettre en tête deux notions importantes :

- En Java, tous les objets sont accessibles par des références (nouveau nom pour *pointeur*).
- La règle générale de Java est que les arguments sont passés par *valeur*.

De ce fait, si une variable de type primitif (*int* par exemple) est passée en paramètre à une méthode, cette variable est dupliquée pour la durée de l'exécution de la méthode.

La règle du passage des arguments par valeur s'applique aussi aux objets, mais alors c'est une *référence vers l'objet* qui est copiée. Ainsi, quand un objet est passé en paramètre à une méthode, sa référence est copiée, on obtient ainsi un



deuxième pointeur sur le *même objet*. Ainsi, toute modification de l'objet à l'intérieur de la méthode implique sa modification dans l'objet initial.

**TAF :** Implémentez la classe *Main* que vous avez créée dans le package *main*. Pour cela, il faut implémenter la seule méthode de cette classe : `public static void main(String[] args)`. Cette méthode doit faire les étapes suivantes :

- Créer un nouvel objet *obj* de type *ObjetAVendre*, qui porte le numéro 123, de type "tableau" et de valeur initiale 1000.
- Créer deux enchérisseurs : le premier s'appelle « Ali Loukil » et le second « Olfa Hajri ». Ali a mis comme mise limite 2000dt et Olfa 2500dt. Tous les deux veulent acheter l'objet *obj*.
- Tant que les deux enchérisseurs sont dans la course (*in*), Ali va ajouter la somme de 100 dt à la mise, et Olfa ajoute 200 dt.
- Le programme doit afficher à la fin le nom de l'enchérisseur qui a remporté l'objet, ainsi que la somme finale de l'objet.

*Remarque :* pour comparer deux objets de type *String*, on doit utiliser la méthode *equals* contenue dans la classe *String* de la manière suivante :

```
if (st1.equals(« val1 »)){...}
```

## IV. Homework

Le directeur d'un laboratoire de recherche désire contrôler l'utilisation abusive de l'imprimante. On désire donc réaliser un programme qui mémorise le nombre de documents imprimés par les membres du laboratoire en un mois en utilisant les notions vues auparavant.

Une seule imprimante existe dans le laboratoire, partagée par tous les membres. Un membre peut être soit un étudiant, soit un chercheur, soit le directeur lui-même. Quand un membre imprime une ou plusieurs feuilles, le nombre de feuilles imprimées est sauvegardé dans l'imprimante. Si le nombre total de feuilles imprimées atteint 1000, l'imprimante n'accepte plus d'autres impressions, et affiche le message : *nombre de feuilles max atteint*.

La limite d'impression pour un étudiant est 100 et pour un chercheur 200. Quand un membre veut faire une impression, l'imprimante vérifie s'il a atteint sa limite. Il ne peut pas imprimer si c'est le cas.

Réaliser cette application en Java sur Eclipse.