## Introduction aux outils de Java

### La documentation

Il y a deux sources indispensables de documentation sur Java : la documentation sur le JDK incluant un manuel de référence complet sur les classes disponibles en standard avec le JDK, et le *tutorial*, qui permet d'apprendre Java par l'exemple. Cette documentation est disponible au format HTML.

Ouvrez un navigateur HTML, et accédez à la documentation en ligne de Java :
`http://docs.oracle.com/javase/7/docs/api/`

Le tutorial est disponible à l'adresse :
`http://docs.oracle.com/javase/tutorial/index.html`
Commencez par *Learning the Java Language* !

Enfin, la documentation sur les outils disponibles en standard avec le kit de développement Java (Java Development Kit, JDK) : `http://docs.oracle.com/javase/7/docs/index.html`.

Ces adresses sont données sur l'espace du cours dans Moodle. Vous pouvez d'ores et déjà ouvrir ces trois pages dans votre navigateur.

**Exercice 1 :** *Ecrire, compiler et exécuter un programme Java*
Créez chez vous un répertoire `java`, puis, dans ce répertoire, un nouveau répertoire `tp1`. Pour ce faire, vous pouvez soit utiliser un gestionnaire de fichiers, soit ouvrir une console, et tapez les commandes suivantes :

```
>mkdir java
>ls
>cd java
>mkdir tp1
>cd tp1
```

Vous allez utiliser l'éditeur `gedit` : menu *Applications/Accessoires/éditeur de texte.*

Vous allez saisir le code Java suivant dans un fichier nommé `Premiere.java`, dans le répertoire `java/tp1`. Il définit une classe `Premiere` qui affiche la chaîne de caractères *"Bonjour le monde"* sur la console. N'oubliez pas les commentaires.

```
/**
 * Ma premiere classe en Java
 *
 * @author Nom Prenom
 * @version 1.0
 */
public class Premiere {

    public static void main(String [] args) {
        System.out.println("Bonjour le monde!");
    }
}
```

Pour compiler cette classe, on utilise le compilateur Java fournit en standard dans le JSDK de Sun : `javac` (il en existe d'autres : `guavac, jikes`). Dans une console, tapez
`javac Premiere.java`

Un fichier `Premiere.class` est créé dans votre répertoire. Il contient la classe compilée, que vous allez maintenant pouvoir interpréter. Il suffit d'appeler la machine virtuelle Java suivie du nom de la classe contenant la méthode `main` **sans l'extension .class** :

```
java Premiere
```

**Exercice 2 :** *Générer la documentation*

Parce que la documentation est importante, le JSDK fournit en standard un générateur de documentation HTML à partir du code source de la classe. Cet outil utilise pour cela les commentaires commençant par `/**`. Il recherche ensuite des balises particulières : `@author`, `@version`, `@param`, `@return`, `@exception`, etc.

Pour plus d'informations, consultez la documentation de Javadoc avec un navigateur HTML

On change un peu la classe `Premiere` : on lui ajoute un attribut `message` et une méthode `affiche`. La méthode `main` créé maintenant un objet `Premiere` avant d'appeler sa méthode `affiche`.

```
/**
 * Ma premiere classe en Java
 *
 * @author Nom Prenom
 * @version 1.0
 */
public class Premiere {

     String message = "Bonjour le monde!";

    /**
     * On utilise le constructeur par defaut
     */

    /**
     * Cette methode affiche le contenu de l'attribut prive
     * message a l'ecran.
     */
    public void affiche() {
        System.out.println(message);
    }

    /** methode principale de mon premier programme. */
    public static void main(String [] args) {
        // Cree un objet de type Premiere
        Premiere prem = new Premiere();
        prem.affiche();
    }
}
```

Recompilez `Premiere`, réinterprétez-la. Rien n'a changé.

Lancez le générateur de documentation sur votre premier programme java :

```
javadoc Premiere.java
```

Pour visualiser la documentation, il suffit alors d'ouvrir le fichier `index.html` dans votre navigateur HTML.

**Exercice 3 :** *Enrichir cette première classe*

On désire maintenant enrichir la classe `Premiere`.

— Déclarez un constructeur qui prend en paramètre une chaîne de caractères qui sera utilisée pour initialiser l'attribut message.

— Déclarez un accesseur `getMessage()` et un modificateur `setMessage()` pour l'attribut `message`.

— Modifiez la méthode `main` pour prendre en compte ces changements.

Vous ferez attention à bien documenter chaque nouvelle méthode.

**Exercice 4 :** *Mettre un peu d'ordre*

Si vous jetez un coup d'oeil à votre répertoire courant, vous trouvez pêle-mêle un fichier source java, un fichier compilé, des fichiers HTML générés par javadoc, etc.

On utilisera la convention suivante pour les prochains TPs :

```
TPx // repertoire pour le TP en cours
 |
 ---- src // fichiers source Java (.java)
 |
 ---- bin // fichiers compiles (.class)
 |
 ---- doc   // documentation (.html)
```

Utiliser la commande `mkdir src bin doc` pour créer ces répertoires.

On utilisera pour cela l'option `-d` de `javac` et de `javadoc`.

```
cd
javac -d bin src/Premiere.java
java -classpath bin Premiere
javadoc -d doc src/Premiere.java
```

**Exercice 5 :** *Plusieurs classes dans un même fichier*

Il est possible de déclarer plusieurs classes dans un même fichier, à condition qu'une seule d'entre elles soit publique.

Créez une classe `Test` dans laquelle vous déplacerez la méthode `main()`.

Compilez le fichier `Premiere.java`. Que remarquez-vous ?

**Exercice 6 :** *La classe Vélo*

Programmez la classe `Velo` vue en cours. Écrivez dans un fichier `Velo.java` :

```
/** cette classe modelise l'etat du velo d'un cycliste qui roule.*/
public class Velo {
  /** la vitesse actuelle du velo. */
  private int vitesse = 0;

  /** le cycliste accelere.
  @param increment indique de combien la vitesse du velo augmente.
  */
  public void accelerer(int increment) {
    vitesse = vitesse + increment;
  }

  /** le cycliste freine.
```

```
  @param decrement indique de combien la vitesse du velo baisse.
  */
  public void freiner(int decrement) {
    vitesse = vitesse - decrement;
  }

  /** affiche l'etat du velo, i.e. sa vitesse. */
  public void imprimeEtat() {
    System.out.println("vitesse: " + vitesse);
  }
}
```

et d'une autre classe pour tester ce que vous venez d'écrire (écrivez-la dans un autre fichier `DemoVelo.java`) :

```
/** Classe de tests pour la classe @class velo */
public class DemoVelo {
  public static void main(String[] args) {
    // Genere deux objets differents du type Velo
    Velo velo1 = new Velo();
    Velo velo2 = new Velo();

    // Invoque les methodes
    velo1.accelerer(10);
    velo1.imprimeEtat();
    velo2.accelerer(20);
    velo2.imprimeEtat();
  }
}
```

N'oubliez pas les commentaires. Compilez, interprétez.

**Exercice 7 :** *On teste un peu notre vélo*
On change un peu le cahier des charges de la classe `Velo` :

1. Le vélo accepte de donner sa vitesse. Implémentez une telle méthode. Comment allez-vous l'appeler ?
2. Dans la classe `DemoVelo`, on veut maintenant faire accélerer `velo1`, par paliers de 3 km/h, jusqu'à ce que sa vitesse dépasse les 40 km/h. Après chaque accélération, le vélo doit afficher sa vitesse. Comment faire ?
3. On fait ensuite freiner le vélo, par paliers de 5 km/h, vingt fois.
4. Quel est le problème ? Comment le résoudre ?

**Exercice 8 :** *La classe Point et la classe Rectangle*
On programme maintenant les classes suivantes :

```
public class Point {
  /** Abscisse du point */
  public int x = 0;

  /** Ordonnee du point */
  public int y = 0;
```

```
  /** On peut construire un point en donnant ses coordonnees.
  @param x abscisse du point
  @param y ordonn\'ee du point
  */
  public Point(int unX, int unY) {
    x = unX;
    y = unY;
  }

  /** Par defaut, le point construit est l'origine du repere. */
  public Point() {
    this(0, 0);
  }
}
```

et

```
public class Rectangle {
  private Point origine;
  private int largeur = 0;
  private int hauteur = 0;

  public Rectangle(Point p, int l, int h) {
    origine = p;
    largeur = l;
    hauteur = h;
  }
}
```

et enfin une classe de tests :

```
public class DemoRectangle {
  public staic void main(String[] args){
    Point unPoint = new Point(23, 94);
    rectangle = new Rectangle(unPoint, 100, 200);
  }
}
```

Effectuez des modifications pour répondre aux questions suivantes :

1. un rectangle doit pouvoir communiquer sa largeur, sa longueur, l'abscisse de son origine et l'ordonnée de son origine (quatre méthodes) ;
2. un rectangle doit pouvoir retourner la valeur de son périmètre ;
3. un rectangle doit pouvoir retourner la valeur de sa surface ;
4. testez vos modifications dans la classe `DemoRectangle` ;
5. à la suite de ces instructions, modifiez la valeur des coordonnées de `unPoint`. Comment faire ?
6. Affichez ensuite la valeur des coordonnées de l'origine du rectangle. Que s'est-il passé ? Comment résoudre le problème ?

**Exercice 9 :** *Utiliser des tableaux*

Pour ce dernier exemple, reprenez l'exemple suivant, Compilez, exécutez.

```
public class DemoTableau {
  public static void main(String[] args) {
    // declaration
    int[] unTableau;

    // allocation de memoire
    unTableau = new int[3];

    // initialisation
    unTableau[0] = 100;
    unTableau[1] = 200;
    unTableau[2] = 300;

    for (int i = 0; i < unTableau.length; i++){
      System.out.println("Element "+i+" : " + unTableau[i]);
    }

    String[][] noms = {{"Mr. ", "Mrs. ", "Ms. "}, {"Smith", "Jones"}};

    System.out.println(noms[0][0] + noms[1][0]);
    System.out.println(noms[0][2] + noms[1][1]);
    for (int i = 0; i < noms.length; i++){
      for(int j = 0; j < noms[i].length; j++){
        System.out.println("Indices "+i+", "+j+" : "+noms[i][j]);
      }
    }
  }
}
```