TP1 JAVA

Concepts abordés

- L'environnement Eclipse
- Définition de classes, instanciation d'objets
- Création et exécution d'un programme Java sous Eclipse
- Opérateurs, expressions
- Structures de contrôle
- Tableaux d'objets
- La classe String

Consignes générales de travail

- Commencez chaque TP dans un nouveau *Projet Java (ici TP1)*
- Créez toujours un (ou plusieurs) paquetages pour contenir ses classes (ne jamais utiliser le paquetage par défaut).
- Écrivez chaque nouvelle classe dans son propre fichier (portant le nom de la classe).
- Pour ce TP, déclarez systématiquement vos attributs en private et vos méthodes en *package friendly* ou public.
- Définissez systématiquement une méthode toString():String *pour chaque classe* que vous écrivez.
- Lisez chaque exercice en entier avant de commencer vos réponses.

Exercice 1: Première utilisation d'Eclipse (maximum 15 minutes)

Cet exercice est conçu pour vous présenter un processus (très sommaire) d'écriture, compilation et exécution d'un programme Java sous Eclipse.

Pour plus d'information sur Eclipse, il vous est vivement conseillé de suivre le tutoriel enligne: http://help.eclipse.org

1) Lancez Eclipse.
2) Si vous lancez Eclipse pour la première fois, vous seriez invité à choisir un répertoire qui servira d'*espace de travail*. Sélectionnez un répertoire sous votre compte personnel, ou sur une clé USB. Par la suite, n'accédez *jamais* à ce répertoire depuis le système de fichiers, pour éviter toute incohérence. Faîtes toute modification nécessaire (import, export, nouveau fichier, etc.) depuis l'interface d'Eclipse.
3) Si vous avez déjà lancé Eclipse au moins une fois, vous pouvez changer l'espace de travail dans le menu *Fichier*.
4) Dans le menu *Fichier*, créer un nouveau ***Projet Java de nom TP1***.
5) Dans le menu contextuel (click-droit) de votre nouveau projet, créez un nouveau paquetage.
6) Dans le menu contextuel du paquetage, créez une **classe HelloWorld**.
7) Ouvrez ce fichier (double-click-gauche), et compléter le contenu de la classe un programme principal affichant Hello World.
8) Notez qu'Eclipse compile en temps-réel vos classes; il n'y a pas d'étape de compilation explicite. Lors de la compilation, l'éditeur souligne toute erreur en rouge, tout avertissement en jaune. Si votre programme contient des erreurs ou avertissements, consultez l'onglet *Problèmes et Erreurs* pour visionner leur cause, et corrigez les erreurs (ignorez les avertissements pour le moment) avant de continuer.
9) Dans le menu contextuel de votre classe HelloWorld, sélectionnez *Exécutez en tant que → Programme Java*. Si cette option ne s'affiche pas, vérifiez que la définition de la méthode main est correcte, et que la classe est public.
10) A l'exécution, le programme affiche son message dans l'onglet *Console*.

Temps cumulé maxi: 15 minutes

Exercice 2: Afficher les arguments d'un programme Java

(maximum 15 minutes)

Ecrire une classe **PrintArgs** qui affiche les arguments d'un programme Java.

Indication: Dans le menu contextuel de votre classe PrintArgs, sélectionnez *Exécutez en tant que → Configurations. Puis dans l'onglet Arguments, entrer les arguments de votre programmae*

Les arguments de la ligne de commande sont stockés dans le tableau de chaînes de caractères passé en argument à la méthode public static main(String[] args).

1) Dans un premier temps, afficher le premier argument de la ligne de commande.
   Que ce passe-t'il si l'on ne passe pas d'argument lors de l'exécution du programme ?
2) Ecrire une boucle affichant le contenu du tableau en sachant qu'en Java les tableaux possèdent un champ (un attribut, une variable d'instance) length qui renvoie la taille du tableau.
3) Ecrire une seconde boucle utilisant la **syntaxe dite 'foreach'** ***for(Type value:array)***

   Exemple pour un tableau d'objets de type String:

```
String [] names = { "Le", "'langage", "Java",  "me", 'plait', "beaucoup'"};
for (String name : names)
        System.out.prinln(name.toString());
```

Ainsi en Java, dans un parcours de **conteneur** d'objets de type **Type** , un **objet** de type **Type** peut servir de variable de boucle et donc être utilisé à la place d'un entier.

Temps cumulé maxi: 30 minutes

Exercice 3: Calculette simple

(maximum 20 minutes)

Ecrire un programme prenant un nombre sur l'entrée standard et affichant celui-ci Pour cela, on utilisera un objet (une instance) de **classe Scanner** et particulièrement sa **méthode nextInt()**.

```
import java.util.Scanner;

public class Calc
{
      public static void main(String[] args)
    {
            Scanner scanner;
            scanner = new Scanner(System.in);
            int value;
            value = scanner.nextInt();
            // completer ici
      }
}
```

Afin de comprendre le programme, il est utile de regarder la **Javadoc** et les **tutoriels de Sun.**

1) Recopier le programme précédent et le compléter pour qu'il affiche le nombre saisi par l'utilisateur.
2) Indiquer dans le programme où sont les variables et quel est leur type associé.
   Modifier le programme pour déclarer et initialiser les variables en une seule ligne
3) Expliquer la ligne : *import java.util.Scanner*
4) Modifier le programme pour qu'il demande deux entiers et affiche la somme de ceux-ci.
5) Afficher en plus de la somme, la différence, le produit, le quotient et le reste.

Temps cumulé maxi: 50 minutes

Exercice 4: Conversion d'un objet de type String en un entier (maximum 25 minutes)

On souhaite écrire un programme affichant la somme d'entiers pris en paramêtre sur la ligne de commande.

Ce programme est décomposé en plusieurs méthodes :

1) Ecrire une méthode qui prend un tableau de chaînes de caractères en argument et renvoie un tableau d'entiers de même taille contenant les entiers issus des chaînes de caractères.
   La **méthode statique parseInt(String s)** de la classe **java.lang.Integer** permet de récuperer la valeur d'un entier stockée dans une chaîne de caractères.
2) Que veut dire statique pour une méthode ?
3) Que se passe-t'il lorsqu'un mot pris en argument n'est pas un nombre ?
4) Ecrire une méthode qui prend un tableau d'entiers en argument et renvoie la somme de ceux-ci.
5) Ecrire la méthode main qui utilise les deux méthodes précédentes pour afficher le tableau d'entiers ainsi que sa somme.
   Pour afficher le tableau d'entiers, deux solutions sont à écrire: la premère *avec* un **foreach** la seconde avec une méthode adéquate de la **classe java.util.Arrays.**

Temps cumulé maxi: 1h 15 minutes

Exercice 5: Classe Point

(maximum 20 minutes)

Ecrire une classe Point stockant un point graphique en coordonnées cartésiennes entières.

1) Déclarer une classe Point contenant les deux champs privés x et y.
   Puis essayer le code suivant dans une méthode main de la classe Point.

```
Point p = new Point();
System.out.println(p.x+" "+p.y);
```

2) Expliquer le phénomène observé.
3) Créer une classe Main (dans un fichier Main.java) et déplacer le main de Point dans la classe Main. Quel est le problème ? Comment peut-on le corriger ?
4) Pourquoi doit-on toujours déclarer les champs privés ?
5) Ecrire les accesseurs en lecture et en écrire pour les 2 attributs ?
6) Ecrire le constructeur à 2 paramètres initialisant les coordonnés d'un point.
7) Ecrire le constructeur de copie de la classe Point, en utilisant la notation this(....) permettant d'appeler un autre constructeur dans le contexte d'un constructeur.
8) Compléter la méthode main afin de tester les méthodes écrites précédemment.

Temps cumulé maxi: 1h 35 minutes

Exercice 6: Classe Point et test d'égalité

(maximum 40 minutes)

```
Point p1 = new Point(1,2);
Point p2 = p1;
Point p3 = new Point(1,2);
System.out.println(p1 == p2);
System.out.println(p1 == p3);
```

1) Qu'affiche le code ci-dessus ?
2) La classe générique **java.util.ArrayList** correspond à une liste chaînée implémentée sous la forme d'un tableau qui s'agrandit dynamiquement. A quoi sert la **méthode indexOf** ?
3) Executez le code suivant :

```
public static void main(String[] args)
{
        Point p1 = new Point(1,2);
        Point p2 = p1;
        Point p3 = new Point(1,2);
        ArrayList<Point> list = new ArrayList();
        list.add(p1);
        System.out.println(list.indexOf(p2));
        System.out.println(list.indexOf(p3));
}
```

4) Quel est le problème avec les résultats affiché sur la console.
5) Quelle méthode de Point est appelée par la méthode indexOf ? Lire la **Javadoc** !!!
6) Modifier la classe Point pour que indexOf() teste selon l'égalité des coordonnées de 2 points et non pas selon leurs références. Quelle méthode doit nécessairement être utilisée? Qu'affiche-t-elle? Par suite, faut-il la redéfinir? Si oui comment?
7) Utiliser l'annotation @Override pour vérifier que vous avez bien écrit la signature de la méthode ajoutée à Point. A quoi sert l'annotation @Override ?

Temps cumulé maxi: 2h 15 minutes

Exercice 7: Afichage d'un point

(maximum 5 minutes)

On veut afficher les caractéristiques d'un point, par le code Java suivant :

```
Point point = new Point(4,7);
System.out.println(point);
```

Pour cela, il faut redéfinir dans la classe Point une méthode **public String toString**() (la définition de cette méthode est dans la classe **java.lang.Object**) retournant une chaîne de caractères, construite à partir des attributs de l'objet.

Rappel: en Java on peut utiliser l'opérateur '+' entre un objet de type String et un objet de n'importe quel autre type ou d'un type simple. Le résultat est leur concaténation et le second opérande est vu comme une suite de caractères.

1) Ecrire cette méthode, pour obtenir par exemple l'affichage suivant : (x, y)
2) Doit-on utiliser l'annotation @Override ?

Temps cumulé maxi: 2h 20 minutes

Exercice 8: String: assignation, égalité, référence

(maximum 15 minutes)

Soit le code suivant:

```
String s1 = "toto";
String s2 = s1;
String s3 = new String(s1);
System.out.println(s1 == s2);
System.out.println(s1 == s3);
```

r

1) Qu'affiche le code ci-dessus? Expliquer?
2) Quelle est la méthode à utiliser si l'on veux tester le contenu des chaînes de caractère ?
3) Qu'affiche le code suivant ? Expliquer.

```
String s1 = "toto";
String s2 = "toto";
System.out.println(s1 == s2);
```

4) Expliquer pourquoi il est important que **java.lang.String** soit non mutable.
5) Qu'affiche le code suivant ? Expliquer.

```
String s = "hello";
s.toUpperCase();
System.out.println(s);
```

Temps cumulé maxi: 2h 35 minutes

Exercice 8: StringBuilder: assignation, égalité, référence (maximum 25 minutes)

Ecrire une **classe Morse** dont la seule méthode main permet lors de son exécution d'afficher les chaînes de caractères, passées en paramètres du programme Morse, séparées par des "Stop.".

Avec les 3 arguments ceci, est, drole

l'affichage est ceci Stop. est Stop. drole Stop.

1) Utiliser dans un premier temps, l'opérateur + qui permet la concaténation de chaînes de caractères.
2) A quoi sert la classe j**ava.lang.StringBuilder ?**
   Pourquoi sa méthode **append(String)** renvoie un objet de type **StringBuilder** ?
3) Modifier la **classe Morse** en utilisant la **classe StringBuilder**.
   Quel est l'avantage par rapport à la solution précédente ?
4) Recopier le code suivant dans une classe de Test :

```
public static void main(String[] args) {
  String first = args[0];
  String second = args[1];
  String las t= args[2];
  System.out.println(first + ' ' + second + ' ' + last);
}
```

Compiler le code puis utilise la commande javap pour afficher *l'assembleur* généré

```
javap -c Test
```

Que pouvez vous en déduire ?

Dans quel cas doit-on utiliser StringBuilder.append() plutôt que le + ?

Temps cumulé maxi: 3h