TD - TP 1

Partie 1 : TD

Exercice 1 (les différents sens de l'opérateur +)

Déterminer pour chacune des expressions suivantes si elle est valide ou non. Si oui, donner sa valeur, si non expliquer pourquoi.

a)	b)	c)

`1 + 2`
`'1' + '2'`
`'1' + 2`
`1 + "2"`
`"1" + 2`
`"1" + '2'`
`"1" + "2"`
	`5 + 3 + "a"`
	`5 + "a" + 3`
	`"a" + 5 + 3`
		`true + false`
		`"a"+ true`
		`'a' + true`

d)
`"" + 1 + 2`
	e)
	`Stack p = new Stack();`
	`System.out.println(p+".");`
	`System.out.println(p.toString());`
	`System.out.println(p);`	

Rappels :

- une expression est évaluée de gauche à droite ;
- un *caractère* (de type primitif char) peut-être implicitement (automatiquement) converti en un *entier* (de type primitif int) par le compilateur : sa valeur est alors son rang dans la table UNICODE ; exemple : **'a'** a le rang 97 dans la table UNCODE et **'0**' le rang 48.
- le type *booléen* (boolean) est incompatible avec tout type *primitif ;*
- l'opérateur + effectue une concaténation dès que *l'un des deux arguments* est une chaîne (instance de String) ; si l'autre argument n'est pas une chaîne, il est automatiquement converti en une chaîne : un argument de type primitif est converti en une chaîne correspondant à sa valeur telle qu'elle s'afficherait ("-123", "false",…) et un objet (ou un tableau) est converti en une chaîne par appel implicite à sa méthode toString.

Exercice 2 (pour commencer avec l'héritage)

On considère la classe A ci-dessous :

```
public class A
{
  private String nom;

  public A(String n) { nom = n; }
  public String laClasse() { return "A"; }
  public String toString()
  { return "Mon nom est "+ nom + ", je suis un "+ laClasse()}; }
}// fin classe A
```

Écrire une classe **B** héritant de **A** de façon à ce que les instructions ci-dessous conduisent à l'affichage "Mon nom est bob et je suis un B".

```
B b = new B("Bob");
System.out.println(b); // c-a-d b.toString();
```

Exercice 3 (préparation du TP 1)

Écrire l'algorithme de l'exercice 5 (évaluation d'expressions arithmétiques postfixées).

Partie 2 : TP Utilisation de la documentation du SDK

Première étape : trouver la documentation html des classes du SDK (soit sur votre machine, soit sur le site officiel : http://docs.oracle.com/javase/7/docs/api/index.html)

Si vous n'avez jamais consulté cette documentation :
- Affichez la liste des packages, affichez la liste des classes d'un package, affichez la documentation d'une classe (pas de panique, c'est normal que vous ne compreniez pas tout).
- Trouvez une classe dont vous connaissez le nom et le package (par exemple la classe String du package java.lang), soit par son package, soit directement par la liste des classes.

Exercice 4 (utilisation de la classe java.lang.String)

On imagine une classe **Mot**, qui permet de représenter des mots (éventuellement vides) et de tester certaines de leurs propriétés. On s'intéresse notamment à la propriété suivante : **palindrome**. Un mot est un palindrome s'il se lit de la même façon de gauche à droite que de droite à gauche. Par exemple : BOB, ABBA, ELUPARCETTECRAPULE.

Question 1. Écrire le squelette de la classe Mot :
- un Mot a un *attribut* privé qui représente la chaîne de caractères elle-même (une instance de la classe **java.lang.String**).
- deux *constructeurs* sont fournis : l'un qui initialise le mot à la chaîne vide, l'autre qui l'initialise avec une chaîne passée en paramètre ; ainsi que deux méthodes : *toString*, qui retourne la chaîne de caractères du mot et *isEmpty* qui teste si le mot est la chaîne vide.

Voir la documentation de la classe String pour trouver quelles méthodes permettent de connaître la **longueur** d'une chaîne de caractères ainsi que d'accéder au **i^{ème} caractère** de la chaîne (attention, comme pour un tableau, le premier caractère d'une chaîne a le rang 0).

Question 2. La méthode isPalindrome() retourne vrai si le mot est un palindrome et faux sinon. Écrire une version **itérative** de cette méthode.

Ajouter une méthode main permettant de la tester. La chaîne de caractères fournie pour créer un mot peut-être passée en paramètre via le tableau `args` de la méthode main, ou peut-être lue au clavier en utilisant une instance de la classe `java.util.Scanner`.

```
// Exemple d'utilisation de la classe Scanner
   Scanner sc = new Scanner(System.in);
   System.out.println("Entrez une chaine");
   String s = sc.nextLine();
```

Exercice 5 (utilisation de la classe java.util.Stack)

Dans le but d'utiliser la classe **Stack** du package **java.util,** nous allons nous intéresser à l'évaluation d'expressions arithmétiques postfixées (aussi appelées expressions en notation polonaise).

On considère ici les quatre opérateurs arithmétiques binaires +, -, x et /.
Une expression arithmétique est habituellement présentée sous forme infixée : l'opérateur se situe entre ses deux opérandes, comme dans a + b. Cette notation nécessite l'utilisation de parenthèses, l'expression ((a + b) * c) n'étant par exemple pas équivalente à a + b * c.

En notation postfixée, les opérateurs suivent immédiatement les opérandes sur lesquels ils agissent. Les parenthèses deviennent inutiles.
Exemples :

(a+b)	devient	ab+
(a+b)*c		ab+c*
a+b*c		abc*+
a*(b+c/d)		abcd/+*

La notation postfixée rend très simple l'évaluation d'une expression arithmétique, comme vous pourrez le constater. On procède de la façon suivante :
- l'expression est lue de gauche à droite
- une pile vide est créée
- lorsqu'un opérande est rencontré, il est empilé
- lorsqu'un opérateur (appelons-le op) est rencontré, on dépile une première fois, appelons B l'élément dépilé, et on dépile une deuxième fois, appelons A l'élément dépilé. On effectue l'opération (A op B) et on empile le résultat.
- lorsque l'expression a été entièrement lue, le résultat de son évaluation est le seul élément de la pile.

On considérera par la suite que les opérandes sont des valeurs littérales de type entier.

(a) Écrire un algorithme correspondant au procédé décrit plus haut.
(b) Une instance de Stack peut contenir des objets mais pas des données de type primitif. Comment faire ? Plusieurs solutions sont envisageables, dont l'utilisation de chaînes de caractères (String).
(c) Écrire une application qui, étant donnée une expression postfixée (fournie au programme comme vous le voulez), affiche le résultat de son évaluation.

Exercice 6 (utilisation des classes enveloppes)

Le package java.lang fournit une classe "enveloppe" pour chaque type primitif : Boolean, Character, Integer, Float, etc… Ces classes fournissent des méthodes permettant d'effectuer des conversions de données de type primitif en objet (par ex : boolean en Boolean ou String) et réciproquement.

On s'intéresse ici aux conversions float ↔ String. Utilisez la documentation du SDK, et plus précisément les méthodes des classes String et Float pour :
1) passer d'une valeur de type primitif float à une String
2) passer d'une String à une valeur de type primitif float (si possible).

Vous devriez trouver trois façons différentes d'effectuer chaque conversion.