

Travaux dirigés n° 2

Introduction à la POO (2)

Exercice 1 (Surcharge de méthode)

Nous considérons le code suivant :

```
public class Exo1 {
    // Attribut(s)
    private int a;

    // Constructeur (par défaut)
    public Exo1() {
        a = 0;
    }

    // Méthodes
    public void test() {
        a++;
    }
    public void test(int b) {
        a += b;
    }
}

public void afficher() {
    System.out.println(a);
}

public class TestSurcharge {
    public static void main(String[] args) {
        Exo1 ref = new Exo1();

        ref.test();
        ref.afficher();

        ref.test(2);
        ref.afficher();
    }
}
```

1°) Que fait la série d'instructions du `main` et quel est l'affichage obtenu ?

2°) Est-il possible d'ajouter, dans la classe `Exo1`, la méthode suivante ?

```
public void test(int c) {
    a -= c;
}
```

3°) Peut-on écrire la version sans paramètre de `test` en faisant appel à l'autre version (avec paramètre) ? Et l'inverse ?

Exercice 2 (Une classe simple : les Complexes !)

Nous souhaitons écrire une classe `Complexe` permettant de représenter un nombre complexe.

1°) Écrivez la classe `Complexe` comprenant :

- les différents attributs
- un constructeur par défaut et un constructeur par initialisation
- les *getters* et les *setters* pour chaque attribut.

2°) Nous souhaitons afficher un complexe à l'écran :

- Rappelez les différents cas d'affichage possibles pour les nombres complexes en fonction de leurs parties réelle et imaginaire.
- Écrivez une méthode `toString` qui renvoie une chaîne de caractères décrivant le complexe.
- Écrivez une méthode `afficher` permettant d'afficher le complexe à l'écran. Que peut-on dire par rapport à la méthode `toString`.

3°) Ajoutez les méthodes suivantes à la classe `Complexe` :

- `estReel` : retournant *true* si le complexe est réel.

- `estImaginairePur` : retournant *true* si le complexe est imaginaire pur.
- `module` : calcule le module du complexe.
- `argument` : calcule l'argument du complexe.

4°) Écrivez les méthodes dont les signatures sont les suivantes :

- `public void additionner(double reel)` : ajoute la valeur `reel` au complexe courant
- `public void additionner(double reel, double imaginaire)` : ajoute la valeur `reel` et la valeur `imaginaire` au complexe courant (resp. à sa partie réelle et à sa partie imaginaire)
- `public void additionner(Complexe ref)` : ajoute à l'objet courant le nombre complexe dont la référence est `ref`

5°) Multiplication(s)

- Ajoutez une méthode `multiplier` permettant de multiplier le nombre complexe courant par un réel (passé en paramètre).
- Ajoutez une *surcharge* de cette méthode permettant de réaliser la multiplication par un nombre complexe

6°) Donnez le diagramme UML de la classe `Complexe`.

7°) Proposez une classe `TestComplexe` contenant un `main` et permettant d'utiliser les différentes méthodes de la classe `Complexe`.

Exercice 3 (Objets et références : représentation mémoire)

Nous considérons la classe `Complexe` écrite dans l'exercice 2.

1°) Représentez l'état de la mémoire aux différents points d'observation :

```
Complexe m, a, b, c;
// point d'observation 1

m = new Complexe(0,1);
a = m;
b = new Complexe(2,3);
// point d'observation 2

m = new Complexe(4,5);
c = m;
// point d'observation 3
```

2°) Décrivez la représentation mémoire au cours de l'exécution du code suivant :

```
Complexe[] t;

t = new Complexe[5];

for(int i=0 ; i<t.length ; i++) {
    t[i] = new Complexe(2*i,2*i+1);
}
```

3°) Et cette fois?

```
Complexe[] t;
Complexe m;

t = new Complexe[5];

for(int i=0 ; i<t.length ; i++) {
    m = new Complexe(2*i,2*i+1);
    t[i] = m;
}
```