TD 2 : Compilation, mémoire

Objectifs pédagogiques : compilation, exécution et gestion de la mémoire, classes immutables.

2.1 Compilation et Exécution

L'exercice consiste à vous aider/entraîner à bien comprendre les différents principes mis en œuvre dans les étapes de compilation et d'exécution de programmes orientés objets en JAVA. La compréhension de ces étapes est nécessaire pour programmer correctement.

Question 1. Soit la classe suivante permettant de décrire un étudiant et définie dans un fichier nommé `Etudiant.java`:

Etudiant.java

```java
package pobj;

public class Etudiant {

  private String nom;
  private String prenom;
  private int numero;

  public Etudiant(String nom,String prenom,int numero) {
    nom = nom;
    prenom = prenom;
    numero = numero;
  }

  public String getNom() {
    return nom;
  }

  public String getPrenom() {
    return prenom;
  }

  public String getNumero() {
    return numero;
  }

  public void setPrenom(String nouveauPrenom){
    this.prenom = nouveauPrenom;
  }

  public String toString(){
    return(nom+" "+prenom+": numéro = "+numero);
  }
}
```

⇒ Donnez la commande de compilation.

⇒ Quelles sont les vérifications faites par le compilateur ?

⇒ Y-a-t-il des erreurs de compilation? Si oui, lesquelles? Corrigez les erreurs trouvées.

Question 2. Soit le code suivant :

MainEtudiant.java

```java
package pobj;

public class MainEtudiant {
  public static void main(String[] args) {
    String nom = "Durand";
    String prenom = "Gerard";
    int numero = 1;

    Etudiant et1 = new Etudiant(nom,prenom,numero);
    prenom = null;
    String nom2 = et1.getNom();
    Etudiant et2 = new Etudiant(nom2,"Paul",2);
    Etudiant et3 = new Etudiant("Durand","Gerard",3);

    et3 = null;
    et1 = et2;
    et1.setPrenom("Jacqueline");
    String s = et2.toString();
    System.out.println(s);
    System.out.println(et1);
  }
}
```

⇒ Donnez sous forme de diagrammes d'objets l'état de la mémoire après chaque ligne d'exécution. Notez que le programme est volontairement «tordu» afin de vous forcer à une bonne compréhension des mécanismes.

⇒ Expliquez ce qui se passe du point de vue du *Garbage Collector* après la dernière ligne.

Question 3. Donnez l'affichage que produit l'exécution du code suivant :

MainEtudiant2.java

```java
package pobj;

public class MainEtudiant2 {

  public static void main(String[] args) {
    String nom = "Durand";
    String prenom = "Paul";
    Etudiant et1 = new Etudiant(nom,prenom,1);
    Etudiant et2 = new Etudiant("Durand","Paul",1);

    System.out.println(et1 == et1);
    System.out.println(et1 == et2);
    System.out.println(et1.equals(et2));

    Etudiant et3 = new Etudiant(nom,prenom,1);
    System.out.println(et1.equals(et3));
  }
}
```

Question 4. On souhaite trier les étudiants alphabétiquement par leur nom suivi de prénom.

⇒ Ajoutez à la classe `Etudiant` une méthode `int compareTo(Etudiant other)` qui renvoie 1 si l'objet `other` doit se situer avant celui sur lequel on appelle la méthode, -1 s'il doit se placer après et 0 s'ils sont homonymes parfaits.

2.2 Visibilité des attributs

La classe suivante permet de représenter une liste d'étudiants.

ListeEtudiant.java

```java
package pobj;

/**
* La classe représente une liste d'étudiants
*/
public class ListeEtudiant {
  private ListeEtudiant next;
  private Etudiant etudiant;

  public ListeEtudiant(){}

  public ListeEtudiant(Etudiant et) {
    etudiant = et;
    next = null;
  }

  public boolean dernier() {
    return(next == null);
  }

  public ListeEtudiant ajouteAvant(Etudiant et) {
    ListeEtudiant lt = new ListeEtudiant(et);
    lt.setNext(this);
    return(lt);
  }

  public void setNext(ListeEtudiant lt) {
    next = lt;
  }

  public void setNext(Etudiant et) {
    next = new ListeEtudiant(et);
  }

  public String toString() {
    String s = etudiant.toString();
    if (dernier())
      return(s);
    else return(s+" -> "+next);
  }
```

Question 5. Il y a un problème de protection de données/méthodes dans cette définition. Pouvez l'identifier et l'expliquer ?

Proposez une solution pour résoudre ce problème de protection.

Question 6. Donnez l'état de la mémoire après chaque ligne d'exécution du code `MainListe.java`.

MainListe.java

```java
package pobj;

public class MainListe {

  public static void main(String[] args) {
    Etudiant et = new Etudiant("Durand","John",1);
    ListeEtudiant l1 = new ListeEtudiant(et);
    ListeEtudiant l2 = l1.ajouteAvant(new Etudiant("Dupond","Guillaume",2));

    System.out.println(l1.dernier());
    System.out.println(l2.dernier());
```

```
   System.out.println(l2);
   et.setPrenom("Jacques");
   System.out.println(l2);
  }
}
```

Question 7. Le tri à bulles est une méthode de tri simple (et peu efficace dans le cas général). Etant donné une liste d'éléments qu'on veut trier dans l'ordre croissant (respectivement décroissant), on parcourt tous les éléments et, si l'élément est supérieur (resp. inférieur) à son successeur, on les inverse. On recommence cette opération jusqu'à avoir parcouru toute la liste sans réaliser une seule inversion.

⇒ Proposez une implémentation du tri à bulles qui repose sur la liste définie précédemment et qui fasse appel à la méthode `compareTo(...)` définie à la Question 4.

2.3 Mutabilité/Immutabilité

Nous décrivons ici un extrait d'une classe très simple pour représenter les nombres complexes.

Complex.java

```
package pobj;

public class Complex implements Cloneable {
   private final double rel; // constante => final
   private final double img; // idem

   public Complex(double rel, double img) {
      this.rel = rel;
      this.img = img;
   }

   public double getReal() {
      return rel;
   }

   public double getImage() {
      return img;
   }

   public Complex add(Complex c) {
      return new Complex(getReal()+c.getReal(),getImage()+c.getImage());
   }

   public Complex subs(Complex c) {
      return new Complex(getReal()-c.getReal(),getImage()-c.getImage());
   }

   public Complex mult(Complex c) {
      return new Complex(getReal()*c.getReal()-getImage()*c.getImage(),
                         getReal()*c.getImage()+getImage()*c.getReal());
   }

   public String toString() {
      String str = ""+rel;
      if(img!=0)
         str=str+"+i"+img;
      return str;
   }
}
```

Question 8. Quel est l'intêret de déclarer les attributs avec le modifieur « final » ?

⇒ Ecrivez une classe `Triangle` dont le constructeur prend trois nombres complexes en paramètre (ces trois nombres complexes représentent les sommets d'un triangle). Ajouter à cette classe `Triangle` une méthode `public void translation()` qui ajoute 1 à la partie réelle de chacun des nombres complexes.

⇒ Compléter les opérateurs sur les complexes (multiplication, soustraction, ...).