TD 2 : Compilation, mémoire, immutabilité

Objectifs pédagogiques :

- **compilation**
- **exécution et gestion de la mémoire**
- **classes immuables**

2.1 Compilation et exécution

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

Question 1. Quelles sont les vérifications faites sur le code Java par le compilateur ?

Question 2. Comment est compilé un programme Java ? Quelle est la nature des fichiers générés ?

Question 3. Qu'est ce qu'un *classpath* ? Un *modulepath* ?

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

`Etudiant.java`

```java
package pobj;

public class Etudiant {

  private static int compteur = 1;

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

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

  public String getNom() { return nom; }

  public String getPrenom() { return prenom; }

  public String getNumero() { return numero; }

  public void setNom(String nouveauNom) { this.nom = nouveauNom; }

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

  public static int getCompteur() { return this.compteur; }

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

Question 4. Quelles sont les erreurs indiquées par le compilateur sur la classe `Etudiant` ? Corrigez son implantation.

Question 5. Donnez la commande pour compiler cette classe. Comment s'assurer que les fichiers sources et les fichiers générés se trouvent dans des répertoires distincts, comme le fait Eclipse ?

Est-il possible d'exécuter la classe `Etudiant` ?

Question 6. Soit le code suivant :

MainEtudiant.java

```java
package pobj;

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

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

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

Donnez la ligne de commande pour compiler cette classe, et celle pour l'exécuter.

Donnez sous forme de diagramme 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 7. 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);
    Etudiant et2 = new Etudiant("Durand","Paul");

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

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

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 ListeEtudiant getNext() { return next; }

  public Etudiant getEtudiant() { return etudiant; }

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

  void setEtudiant(Etudiant e) { etudiant = e; }
```

Question 8. 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");
    ListeEtudiant l1 = new ListeEtudiant(et);
    ListeEtudiant l2 = l1.ajouteAvant(new Etudiant("Dupond","Guillaume"));

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

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

Question 9. Il y a un problème de protection des données dans la classe `ListeEtudiant`. Pouvez-vous l'identifier ?

Question 10. Proposez une solution pour résoudre ce problème de protection.

Question 11. Nous considérons maintenant le cas de listes d'étudiants triées par ordre alphabétique du nom suivi du prénom (c'est-à-dire que, si deux étudiants ont le même nom, le prénom est utilisé pour les ordonner).

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 la méthode est appelée, -1 s'il doit se placer après et 0 s'ils sont homonymes parfaits.

Proposez le code d'une méthode `boolean contains(ListeEtudiant list, Etudiant etu)` qui renvoie `true` si la liste triée d'étudiants `list` contient un étudiant dont le nom et le prénom correspond à celui de `etu`, et `false` sinon.

2.2 Mutabilité, immutabilité

Nous décrivons ici un extrait d'une classe très simple pour représenter une position sur une carte.

Position.java

```
package pobj;

public class Position {

   private final double x;
   private final double y;

   public Position(double x, double y) {
      this.x = x;
      this.y = y;
   }

   public double getX() { return x; }

   public double getY() { return y; }

   @Override
   public boolean equals(Object other) {
      if(other == this)
         return true;
      if(! ( other instanceof Position))
         return false;
      Position pos_other = (Position) other;
      return pos_other.x == x && pos_other.y == y;
   }

   @Override
   public String toString() {
      return "( " + x + " , " + y + " )";
   }

}
```

Question 12. Quel est l'intérêt de déclarer tous les attributs avec le modificateur `final` ?

Question 13. Ajoutez à la classe `Position`, une méthode `bound(double minX, double minY, double maxX, double maxY)` permettant de borner les coordonnées de la position.

Question 14. Quels sont les avantages et les inconvénients :

- des classes immuables ?
- des classes mutables ?