

# ALGORITHMIQUE III

## PROGRAMMATION ORIENTÉE OBJETS

### EXAMEN JANVIER 2009

Vous avez 1h30 pour faire autant de questions que possible. En sortant, vous rendrez une copie manuscrite contenant vos commentaires, justifications, etc. Si vous avez du code source à rendre, créez un répertoire portant comme nom le numéro de votre copie, mettez vos fichiers dans ce code source et copiez le répertoire dans le répertoire partagé dont votre surveillant vous donnera le nom.

Chaque section est notée sur environ 20 points. N'oubliez pas de commenter vos réponses, puisque, en l'absence de commentaires et de justification, le correcteur se réserve le droit de ne même pas lire ce que vous avez écrit. Vous serez notés entre autres sur la pertinence de ces justifications.

Vous avez le droit d'utiliser un ordinateur et d'accéder à Internet, en particulier au WikiSciences. Par contre, les documents sont interdits, ainsi que le mail, le chat, les autres félins, les SMS, la pause pipi avant 1h et tout autre moyen de communiquer avec l'extérieur.

## Héritage

Pour toute cette section, nous allons partir de l'extrait suivant. Cet extrait définit une classe qui sert à représenter les couples d'éléments :

```
public class Couple
{
    public final Object premier;
    public final Object deuxième;
}
```

**Exercice 1.** Que représente `Object` en Java ? Quels types de données peut-on mettre dans `premier` ou/et dans `deuxième` ? Comment peut-on modifier les données qui apparaissent dans `premier` et dans `deuxième` ?

**Correction** *En Java, `Object` est la classe dont héritent toutes les autres classes. En particulier, cela signifie qu'il est possible de mettre dans `premier` ou/et dans `deuxième` n'importe quel type de valeurs, qu'il s'agisse d'entiers, de textes, d'autres couples, etc.*

*Notons que, une fois qu'une valeur est mise dans `premier` ou/et `deuxième`, elle ne peut pas être modifiée, puisque ces champs ont été déclarés comme `final`.*

**Exercice 2.** Telle quelle, cette classe ne compile pas. Complétez-la pour en faire quelque chose qui compile. Précisez et justifiez chacune de vos modifications.

*Le problème de cette classe est que `premier` et `deuxième` ont été déclarés comme `final`, ce qui signifie d'une part que leur valeur ne doit jamais changer et d'autre part qu'elle doit être fixée dès la construction de l'objet. En d'autres termes, il manque à `Couple` un constructeur, qui fixera la valeur de `premier` et `deuxième`. Nous pouvons par exemple écrire :*

```
public class Couple
{
    public final Object premier;
    public final Object deuxième;
    public Couple(Object premier, Object deuxième)
    {
        this.premier = premier;
        this.deuxième= deuxième;
    }
}
```

```
}
```

**Exercice 3.** Définissez une classe `MonoCouple` qui étend la classe `Couple` et qui servira à représenter le cas particulier des couples dont les deux éléments sont égaux. Bien entendu, à vous de vous débrouiller pour que `premier` et `deuxième` soit toujours égaux dans les instances de `MonoCouple`.

*Pour ce faire, il suffit de définir une classe qui étend `Couple` et d'invoquer le constructeur avec deux fois la même valeur.*

```
public class MonoCouple extends Couple
{
    public MonoCouple(Object o)
    {
        super(o, o);
    }
}
```

**Exercice 4.** Supposons que nous disposons d'un objet `c` de la classe `Couple` et tel que `c.premier` et `c.deuxième` soient égaux. Alors, que vaudra `(c instanceof MonoCouple)` ?

*Ceci était une question vaguement piégée. En effet, rappelons que `instanceof` permet de demander à Java si `c` a pour type dynamique `MonoCouple`. Si `c` a été construit comme un `Couple` (c'est-à-dire si `c` a pour type dynamique `MonoCouple`), `(c instanceof MonoCouple)` vaudra `false`, quelles que soient les valeurs de `c.premier` et `c.deuxième`. Par contre, si `c` a été construit comme un `MonoCouple`, `(c instanceof MonoCouple)` vaudra `true`.*

## Entrées/sorties

**Exercice 5.** L'extrait suivant devrait définir une fonction qui permet de compter les lignes d'un fichier. Malheureusement, ne compile et ne marche pas. Procédez aux modifications nécessaires pour le faire compiler et marcher. Précisez *et justifiez* chacune de vos modifications :

```
public class CompteurDeLignes
{
    /**
     * Compte le nombre de lignes dans un fichier.
     *
     * @param fichier Le nom du fichier.
     * @return Le nombre de lignes dans le fichier (0 si le fichier est vide)
     */
    public static final void compterLignes(File fichier)
    {
        BufferedReader lecteur = new BufferedReader(fichier);
        int lignes = 0;
        while(lecteur != null)
        {
            lecteur.readline ();
            lignes ++;
        }
        return lignes;
    }
}
```

**Correction** *Regardons les erreurs les unes après les autres :*

- *Cette fonction est supposée renvoyer un nombre de lignes, comme l'indiquent la documentation et le `return lignes`. Il faut donc la déclarer avec comme type de retour `int` et non pas `void`.*

- La documentation indique que `fichier` est un nom de fichier et pas un objet `File`. Il faut donc le déclarer comme `String` et non pas `File`.
- Cette fonction manipule les fichiers et doit donc soit gérer les erreurs elle-même, soit faire remonter les erreurs. Choisissons cette deuxième possibilité et ajoutons donc à la déclaration `throws IOException`.
- Pour construire un `BufferedReader`, nous devons commencer par construire un `Reader`, par exemple un `FileReader`, à l'aide de `new FileReader(fichier)`.
- Une fois que nous aurons fini d'utiliser le fichier, nous devons le refermer, qu'il y ait eu une erreur ou non. Pour ce faire, ajoutons `lecteur.close()` dans le cas où il n'y aurait pas eu d'erreur, et `try { ... } catch(IOException e) { lecteur.close(); throw e}`, pour fermer même en cas d'erreur.
- Le test de nullité doit s'effectuer non pas sur le lecteur, que nous avons construit nous-même est qui n'est donc pas `null`, mais sur sa valeur de retour. Pour ceci, plusieurs manières. La plus simple consiste à déclarer une chaîne de caractères `lu`, qui vaudra le résultat de `lecteur.readLine()`, et dont nous testerons la nullité.
- La méthode s'appelle `readLine` et non `readline`.

En rassemblant toutes ces modifications, nous obtenons :

```
public class CompteurDeLignes
{
    /**
     * Compte le nombre de lignes dans un fichier.
     *
     * @param fichier Le nom du fichier.
     * @return Le nombre de lignes dans le fichier (0 si le fichier
     est vide)
     */
    public static final int compterLignes(String fichier)
        throws IOException
    {
        BufferedReader lecteur = new BufferedReader(
            new FileReader(fichier)
        );
        try
        {
            int lignes = 0;
            String lu = lecteur.readLine();
            while(lu != null)
            {
                lu = lecteur.readLine ();
                lignes ++;
            }
            lecteur.close();
            return lignes;
        } catch (IOException e) {
            lecteur.close();
            throw e;
        }
    }
}
```

**Exercice 6.** Modifiez l'extrait pour respecter la documentation suivante :

```

/**
 * Compte le nombre de lignes dans un fichier.
 *
 * Cette version de la fonction prend en compte une notion de longueur maximale
 * de ligne. Si une ligne s'avère trop longue, cela signifie probablement que
 * le fichier ne contient pas du texte mais, par exemple, une image ou du son.
 * Le cas échéant, la fonction s'arrête et lance une IOException.
 *
 * @param fichier Le nom du fichier.
 * @param longueur Un entier positif qui représente la longueur maximale
 * d'une ligne
 * @return Le nombre de lignes dans le fichier.
 * @throws IOException Si la longueur d'une ligne dépasse le paramètre longueur.
 */

```

**Rappel** La classe `String` dispose d'une méthode `int length()`, qui permet de déterminer la longueur d'une chaîne de caractères.

### Correction

*Il suffit d'ajouter un test sur la longueur de `lu`, comme suit :*

```

public class CompteurDeLignes2
{
    public static final int compterLignes(String fichier,
        int longueur)
        throws IOException
    {
        BufferedReader lecteur = new BufferedReader(
            new FileReader(fichier)
        );
        try
        {
            int lignes = 0;
            String lu = lecteur.readLine();
            while(lu != null)
            {
                if(lu.length() >= longueur)
                {
                    throw new IOException("Ligne trop longue");
                }
                lu = lecteur.readLine ();
                lignes ++;
            }
            lecteur.close();
            return lignes;
        } catch (IOException e) {
            lecteur.close();
            throw e;
        }
    }
}

```

Bon courage