IUT Charlemagne
Département Informatique

Approfondissement du langage Java
2008-2009

TD 2 - Exercices de manipulation des flux Java

1 Copie de fichier texte

Programmez une classe qui copie un fichier texte source vers un fichier texte destination, selon deux méthodes possibles : caractère par caractère, ou ligne par ligne. Son squelette est le suivant :

```
public class CopieFichierTexte {
  private String source;
  private String destination;

  public CopieFichierTexte(...) {
    ...
  }
  public static void main(String[] args) {
    try {
      ...
    } catch (...) {
      System.out.println("erreur a l'ouverture des flux");
    } catch (...) {
      System.out.println("erreur lors des lectures/ecritures");
    }
  }
  public void copieCaracteres() throws ... {
    ...
  }
  public void copieLignes() throws ... {
    ...
  }
}
```

2 Ecritures formatées

Programmez une classe qui a comme attribut un réel, et qui propose une méthode remplissant un fichier texte avec les multiples entiers de ce réel selon le format suivant (dans cet exemple, le réel est 0.3, et on demande les multiples jusqu'à 5) :

```
1 0.3
2 0.6
3 0.9
4 1.2
5 1.5
```

Son squelette est le suivant :

```
public class Multiples {
    private double val;
    public Multiples(...) {
        ...
    }
    public static void main(String[] args) {
        try {
            ...
        } catch (IOException e) {
            System.out.println("probleme a l'ouverture du fichier");
        }
    }
    public void remplitFichier(...,...) throws ... {
        ...
    }
}
```

3 Fichiers de données binaires

Programmez une classe qui a comme attribut un réel, et qui propose une méthode remplissant un fichier binaire avec les multiples entiers de ce réel (précédés du coefficient de multiplication entier comme dans l'exercice précédent). Cette classe doit de plus proposer une méthode qui analyse un tel fichier de façon à écrire à l'écran le réel dont les multiples ont été calculés, ainsi que le nombre de multiples calculés.

Son squelette est le suivant :

```
public class MultiplesBin {
    private double val;

    public MultiplesBin(...) {
        ...
    }
    public static void main(String[] args) {
        try {
            ...
        } catch (...) {
            System.out.println("probleme a l'ouverture du fichier");
        } catch (...) {
```

```
            System.out.println("probleme en lecture ou ecriture");
        }
    }
    public void remplitFichier(...,...) throws ... {
        ...
    }
    public static void extraitFichier(...) throws ... {
        ...
    }
}
```

4 Flux d'objets

Modifiez la classe UnRectangle de façon à pouvoir sauvegarder un rectangle dans un fichier binaire, ou fabriquer un rectangle à partir d'un fichier de sauvegarde. Programmez une classe de test avec une méthode `main()` qui intercepte les exceptions susceptibles d'être levées par les méthodes de sauvegarde et de restitution.

Rappel :

```
public class UnPoint {
   private int abscisse,ordonnee;
   public UnPoint(int a,int o) {
      abscisse=a;
      ordonnee=o;
   }
   public void translation(int dx,int dy) {
      abscisse=abscisse+dx;
      ordonnee=ordonnee+dy;
   }
}
```

```
public class UnRectangle {
    private UnPoint coin;
    private int largeur,hauteur;
    public UnRectangle(UnPoint cig,int l,int h) {
        if (cig==null) coin=new UnPoint(0,0);
        else coin=cig;
        if (l<1) largeur=1; else largeur=l;
        if (h<1) hauteur=1; else hauteur=h;
    }
    public void translation(int dx,int dy) {
        coin.translation(dx,dy);
    }
}
```

5 Grep

Le but de cet exercice est d'écrire une application Java en ligne de commande qui permet d'afficher les lignes d'un fichier contenant un certain mot. Par exemple, voici la liste des lignes qui contiennent le mot "public" dans le fichier `UnRectangle.java` ci-dessus :

On lance la commande :

```
java Grep public UnRectangle.java
```

Et on obtient le résultat :

```
UnRectangle.java (1:0) : public class UnRectangle {
UnRectangle.java (4:4) :     public UnRectangle(UnPoint cig,int l,int h) {
UnRectangle.java (10:4) :     public void translation(int dx,int dy) {
```

Dans la parenthèse `(x:y)`, on trouve en premier `x`, l'indice de la ligne et en deuxième `y`, l'indice de la première lettre du mot "public" dans la ligne `x`.

La commande à utiliser a la forme générale :

```
java Grep mot fichier1 fichier2 ...
```

Remarque : lorsqu'un programme Java est lancé avec des arguments en ligne de commande, ces arguments sont disponibles sous forme de chaînes de caractères placées dans le tableau paramètre de la méthode `main`. Dans l'exemple qui précède, si l'entête de la méthode principale est

```
public static void main(String[] args)
```

alors lors de l'exécution, `args.length` vaut 2, ce qui indique qu'il y a deux arguments en ligne de commande, `args[0]` est la chaîne `"public"` et `args[1]` est la chaîne `"UnRectangle.java"`.

On pourra programmer la classe `Grep` selon les commentaires javadoc suivants :

```
import java.io.*;

/**
 * Cette classe se charge d'ouvrir le fichier, de le lire en extrayant
 * les lignes ou se trouve le mot recherche (methode litLigne), et de fermer le fichier
 * (methode close).  Elle contient aussi la méthode main() qui permet d'utiliser
 * la forme générale de la commande Grep .
 **/
public class Grep {
    /** Nom du fichier a ouvrir. */
    private String nomfich;

    /** Mot recherche. */
    private String motrech;

    /** BufferedReader correspondant a nomfich. */
    private BufferedReader src;

    /**
     * Contenu de la derniere ligne lue.
     * La valeur (null) correspond à la fin du
     * fichier.
     **/
    private String ligne;

    /**
     * Index du mot recherche au sein de la derniere
     * ligne lue.
     **/
    private int index;

    /**
     * Numero de la derniere ligne lue au
     * sein du fichier.
     **/
```

```
private int num;

/**
 * Constructeur de la classe Grep. Initialise certains attributs,
 * ouvre le fichier et crée le BufferedReader correspondant
 *
 * @param nom nom du fichier a lire
 * @param mot mot recherche dans le fichier
 * @exception FileNotFoundException fichier non trouvé
 *
 **/


/**
 * La methode valLigne renvoie la valeur de l'attribut ligne.
 *
 * @return derniere ligne lue
 *
 **/

/**
 * La methode valIndex renvoie la valeur de l'attribut index.
 *
 * @return index du mot recherche
 *
 **/

/**
 * La methode numLigne renvoie la valeur de l'attribut num.
 *
 * @return numero de la derniere ligne lue
 *
 **/


/**
 * La methode litLigne retourne la prochaine ligne
 * du fichier contenant le mot recherche. Elle
 * la stocke dans l'attribut ligne et calcule l'index
 * du mot recherche dans l'attribut index.
 *
 * @return derniere ligne lue
 * @exception IOException exception liée aux entrées/sorties
 *
 **/
```

```
    /**
     * La methode close ferme le BufferedReader correspondant au fichier.
     *
     **/

    /**
     * Methode statique main. Elle analyse la
     * ligne de commande (variable args) et lance
     * la methode litFichier sur chacun des noms
     * de fichiers fournis en ligne de commandes.
     * @param args ligne de commandes
     *
     **/

    /**
     * La methode litFichier cree un objet Grep et parcourt
     * le fichier a la recherche du mot, réalise les impressions
     * demandées pour chaque ligne contenant le mot puis ferme le
     * fichier.
     *
     * @param nom nom du fichier à analyser
     * @param mot mot recherche
     * @exception IOException exception liée aux entrées/sorties
     *
     **/
}
```