Licence Informatique, 2002-2003
L2S : Programmation Orientée Objet en Java

Examen, Juin 2003

Durée de l'épreuve : 2 heures

L'usage des polycopiés du cours ***(pas les TD)*** *et des notes de cours est autorisé. Tous les autres documents, sous quelque forme que ce soit (papier, électronique ou autre) sont interdits.*

Dans votre code, choisissez des noms d'identificateurs convenables et respectez les conventions de Java pour ces noms. Indentez correctement. La lisibilité de votre code sera un critère important pour la note. Commentez les passages de votre code qui présentent une difficulté de compréhension mais n'écrivez pas de commentaires qui n'ajoutent rien à la compréhension.

L'utilisation du crayon à papier est autorisé pour écrire le code.

Quelques exercices

Exercice 1 : Iterateur

On souhaite doter la classe Collec d'un iterateur :

```
public interface Iterator
Method Summary :
   boolean hasNext()
        Returns true if the iteration has more elements.
   Object next()
        Returns the next element in the iteration.
   void remove()
        Removes from the underlying collection the last element returned
        by the iterator
```

Ecrivez la classe MonIterateur qui représente un iterateur de Collec et la methode iterator de la classe Collec.

Indications:

- la classe MonIterateur ne sera pas une classe interne de la classe Collec
- la methode remove effectura un décalage des éléments présents dans un objet Collec (pas de "trou" dans le tableau)

```
import java.util.Iterator;

public class Collec {

   public static final int TAILLE_MAX = 100;
   private Object[] tableau;
   private int tailleCourante;

   public Collec() {
      tableau = new Object[Collec.TAILLE_MAX];
      tailleCourante = 0;
   }

   public int getTaille() {
      return tailleCourante;
   }
```

```
   public void ajouter (Object o) {
      if (tailleCourante < Collec.TAILLE_MAX)
         tableau[tailleCourante++] = o;
   }

   public Object getObjet(int index) {
      if ((index >= 0) && (index <= tailleCourante))
         return tableau[index];
      else
         return null;
   }

   public void setObjet(int index, Object o) {
      if ((index >= 0) && (index <= tailleCourante))
         tableau[index] = o;
   }

   public void supprimer() {
      if (tailleCourante != 0)
         tableau[tailleCourante--] = null;
   }

   public Iterator iterator() {
       // A COMPLETER
   }
}
```

Exercice 2 : Ligne de commande et Exception

La fonction Integer.ParseInt est specifiée ainsi.

```
public static int parseInt(String s) throws NumberFormatException

      Parses the string argument as a signed decimal integer.
      The characters in the string must all be decimal digits,
      except that the first character may be an ASCII minus
      sign '-' ('\u002D') to indicate a negative value. The
      resulting integer value is returned, exactly as if the
      argument and the radix 10 were given as arguments to the
      parseInt(java.lang.String, int) method.

      Parameters:
          s - a String containing the int representation to be parsed
      Returns:
          the integer value represented by the argument in decimal.
      Throws:
          NumberFormatException - if the string does not contain
          a parsable integer.
```

Utiliser cette fonction pour faire la somme de tous les entiers donnés comme arguments de la ligne de commande. Les autres arguments génèreront un message d'erreur.

On aura, par exemple :

```
~> java Test fsq 123 56 hjgh 8
"fsq" ne représente pas un entier !
"hjgh" ne représente pas un entier !
```

Exercice 3 : Heritage et polymorphisme

Voici trois classes :

```
abstract class A {
    public void foo () { System.out.println("A::foo"); }
}

class B extends A {}

class C extends B {
    public void foo () { System.out.println("C::foo"); }
}
```

On utilise ces trois classes dans le programme suivant. Répondez aux questions.

```
public class Test {

    public static void main (String[] Args) {

        A a = new C(); // Quels constructeurs sont appeles ? (précisez
l'ordre)

        a.foo();
        ((A)a).foo();
        ((B)a).foo(); // Que s'est-il ecrit sur la sortie standard après
ces trois lignes ?

        /* B b2 = new A(); */  // Cette ligne est commentée car elle
contient deux erreurs : Expliquez lesquelles.

        B b = new B();
        b.foo(); // Que s'est-il écrit sur la sortie standard ?
    }
}
```

Le problème: formattage de lignes

On veut manipuler des lignes composées de caractères; dans une première étape, ces caractères seront les caractères au sens usuel (représentés par un code ASCII); dans une seconde étape, on considèrera de petites images comme pouvant tenir lieu d'éléments dans une ligne.

Le but général du problème consiste à permettre à l'utilisateur de rentrer des caractères dans une certaine taille et police, ayant donc une certaine largeur et hauteur, et des images ayant aussi une certaine largeur et hauteur. Il s'agit d'afficher le tout dans une fenêtre graphique. Eventuellement, l'on voudra pouvoir couper les lignes afin de s'accomoder d'une largeur maximum de la fenêtre graphique tel qu'illustré par la figure 1.

Voici un extrait d'un code de test

```
Ligne l = new Ligne(80);
// quelle que soit la largeur des
caracteres,
// une ligne n'aura pas plus de 80 car.
Fonte f1 = new Fonte("Courrier", 14);
Fonte f2 = new Fonte("Times", 10);
Caractere c1 = new Caractere('B', f1);
PetiteImage i1 = new PetiteImage(
new ImageIcon("/net/lib/jdk1.3/demo/
Java2D/images/globe.gif"));
l.addCar(c1);
l.addCar(i1);
l.addChaine("on", f2);
l.addChaine("jo", f1);
l.addChaine("ur !!", f2);
l.addChaine("123456789", f1);
l.addChaine("123456789", f2);
l.addChaine("123456789", f1);
l.addChaine("123456789", f2);
PetiteImage i2 = new PetiteImage(
new ImageIcon("/net/lib/jdk1.3/demo/
Java2D/images/bld.jpg"));
l.addCar(i2);
l.addChaine("123456789", f1);
l.addChaine("123456789", f2);
l.addChaine("123456789", f1);
l.addChaine("123456789", f2);
new GUI(l);
```

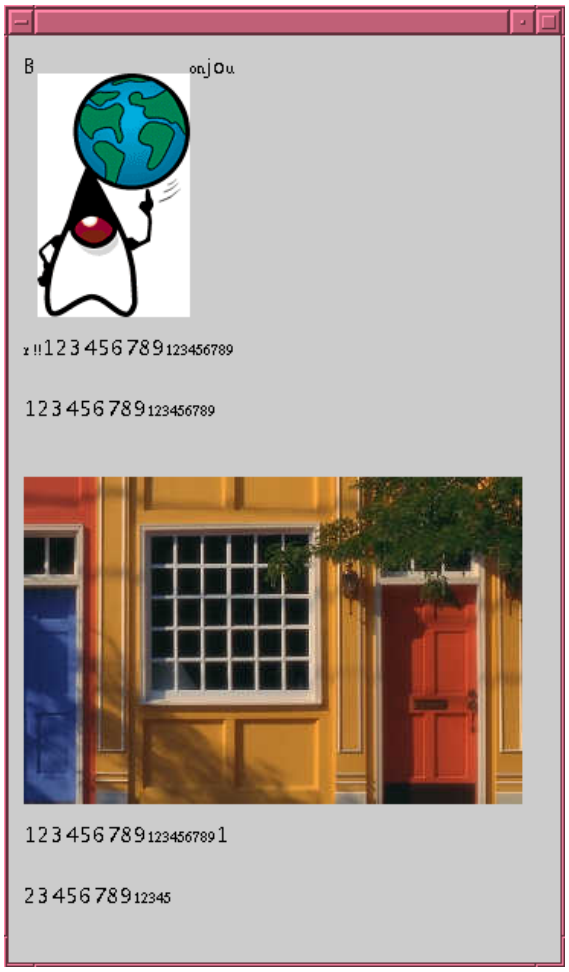

FIG. 1 – *Un bout de code pouvant apparaitre dans une méthode* main *et le résultat graphique que l'on désire obtenir, en ayant fixé une taille maximale en nombre de pixels pour la zone d'affichage (par ex. ici 140, sachant que l'image du globe qui apparait* ***sur la 1ère ligne*** *est de 100 de large). Si un caractère a une largeur supérieure, il sera seul sur sa ligne, mais sera quand même affiché (cf. la seconde image).*

```
/********* CLASSE FONTE *************************/
import java.awt.*;
import java.awt.font.FontRenderContext;

public class Fonte {
  private String nom;
  private int taille;
  private Font font;

  public Fonte(String nom, int taille) {
    this.nom = nom;
    this.taille = taille;
    font = new Font(nom, Font.PLAIN, taille);
  }
  public Font getFont() {
    return font;
  }
  public int getLargeur(char caractere, Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    // La largeur d'un caractere depend du contexte graphique ;
    // par exemple, depend de la largeur du trace.
    FontRenderContext frc = g2.getFontRenderContext();
    // On renvoie la valeur entiere approchee du double
    return (int)(font.getStringBounds(new String(new char[] {caractere}), frc)
                .getWidth() + 0.5);
  }
  public int getHauteur(char caractere, Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    // La hauteur d'un caractere depend du contexte graphique ;
    // par exemple, depend de la largeur du trace.
    FontRenderContext frc = g2.getFontRenderContext();
    // On renvoie la valeur entiere approchee du double
    return (int)(font.getStringBounds(new String(new char[] {caractere}), frc)
                .getHeight() + 0.5);
  }
}
```

```
/******* CLASSE CARACTERE ******************/
import java.awt.Graphics;

public class Caractere {
  private Fonte fonte;
  private char caractere;

  public Caractere(char caractere, Fonte fonte) {
    this.caractere = caractere;
    this.fonte = fonte;
  }
  public char getCar() {
    return caractere;
  }
  public Fonte getFonte() {
    return fonte;
  }
  public int getLargeur(Graphics g) {
    return fonte.getLargeur(caractere, g);
  }
  public int getHauteur(Graphics g) {
    return fonte.getHauteur(caractere, g);
  }
  public void afficheToi(Graphics g, int x, int y) {
    g.setFont(fonte.getFont());
    g.drawString(String.valueOf(caractere), x, y);
  }
}
```

```
/****************CLASSE GUI ***************/
import javax.swing.*;
import java.awt.*;
import java.util.List;

public class GUI extends JFrame {

  public GUI(final Ligne chaine) {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container cp = getContentPane();
    JPanel panel = new JPanel() {
        public void paintComponent(Graphics g) {
          super.paintComponent(g);
          l.afficheToi(g,10,25); // PROBLEME Q1
          /***** PROBLEME Q3  ***************/
          // hypothese: tailleMaxLigne doit suffir pour au moins afficher
          // le plus large des caracteres contenus dans la Ligne
          List listeLignes = Ligne.miseEnLignes(chaine, 140, g);
          int yLigne = 25;
          for (int i=0; i < listeLignes.size(); i++){
            Ligne l = (Ligne)listeLignes.get(i);
            l.afficheToi(g, 10, yLigne);
            yLigne += l.hauteurLigne(g)+25;
          }
          /*********************************/
        }
     };
    cp.add(panel, BorderLayout.CENTER);
    setSize(new Dimension(400, 600));
    setVisible(true);
  }

} // GUI
```

Problème Question 1: Une ligne formée uniquement de caractères ASCII

Il s'agit pour le moment de compléter le code de la classe `Ligne` donné ci-après, dans le cas le plus simple: on suppose que les caractères ou les chaînes rentrées dans la méthode *main* rentrent dans une ligne. Et il s'agit juste de compléter la classe Ligne afin d'afficher cette unique ligne. On vous donne également pour information la classe `GUI.java`.

```
import java.awt.Graphics;
import java.util.*;

public class Ligne {
  private int nbCars = 0;

  public Ligne(int nbMaxCar) {
      // COMPLETER probleme Q1
  }
  public int getNbCar() {
    return nbCars;
  }
  public void addCar ..... {
      // COMPLETER probleme Q1
  }
  public void addChaine .....{
      // COMPLETER probleme Q1
  }
  public void afficheToi(Graphics g, int x, int y) {
    for (int i=0; i < nbCars; i++){
      // COMPLETER probleme Q1
      car.afficheToi(g, x, y);
      x += car.getLargeur(g);
    }
  }
  public int largeurLigne(Graphics g) {
    int largeur = 0;
    // COMPLETER probleme Q3
    return largeur;
  }
  public int hauteurLigne(Graphics g) {
    int hauteur = 0;
    // COMPLETER probleme Q3
    return hauteur;
  }
  public static List miseEnLignes(Ligne ligne, int tailleMaxLigne,
                                   Graphics g) {
      // COMPLETER probleme Q3
  }
}
```

En d'autres termes, étant données les classes fournies que sont `Caractere.java` et `Fonte.java`, compléter le squelette de la classe `Ligne.java` pour ce qui concerne les méthodes étiquetées `COMPLETER probleme Q1`. Un exemple d'utilisation de ces méthodes se trouve en figure 1. On vous demande également de compléter la méthode `afficheToi` de la classe Ligne pour pouvoir afficher cette unique ligne (cf son utilisation dans la classe `GUI`).

Problème Question 1bis: cas d'une ligne pleine

Dans le cas où la ligne est déjà pleine (c'est-à-dire contient déjà le nombre maximum de caractères qu'elle peut contenir), les méthodes `addCar` et `addChaine` devront lancer une exception **contrôlée** de type `LignePleineException`. La méthode `addChaine` lèvera cette exception en ayant ajouté le plus de caractères possible. Ainsi, on vous demande à présent d'indiquer les modifications éventuelles à apporter aux autres classes déjà fournies pour gérer ce type d'exception.

Problème Question 1ter: un paquetage

Outre la classe `GUI`, ainsi qu'une autre classe servant de Test en définissant une méthode `main` (cf figure 1), on veut à présent regrouper toutes les classes dans un paquetage que l'on nommera **fr.unice.toto.imprimerie**. Indiquez les modifications à apporter au code déjà écrit. Pour toute la suite du sujet, vous devrez continuer à considérer la contrainte de ce paquetage (et donc, donner les ajouts nécessaires aux différentes classes).

Problème Question 2: Une ligne formée de caractères ASCII et de petites images

On veut pouvoir insérer des petites images. comme élément de ligne, au même titre qu'on l'avait fait pour des caractères ASCII.

Par conséquent, en considérant la classe `PetiteImage` qui vous est fournie ci-après, modifiez la classe `Ligne` en conséquence.

Indiquez si nécessaire les modifications à apporter ailleurs, dans les autres classes. Bien sûr, faites le moins de modifications possibles dans le code déjà écrit et ce en favorisant l'usage du polymorphisme.

Note: pensez à l'intérêt d'utiliser la notion d'interface.

```
/*************CLASSE PETITE IMAGE **************/
import javax.swing.ImageIcon;
import java.awt.Graphics;

public class PetiteImage {
  private ImageIcon image;

  public PetiteImage(ImageIcon image) {
    this.image = image;
  }
  public int getLargeur(Graphics g) {
    return image.getIconWidth();
  }
  public int getHauteur(Graphics g) {
    return image.getIconHeight();
  }
  public void afficheToi(Graphics g, int x, int y) {
    image.paintIcon(null, g, x, y);
  }

} // PetiteImage
```

Problème Question 3 : Mise en forme d'une ligne pour affichage sur plusieurs lignes

Ecrivez dans la classe `Ligne` une méthode utilitaire `miseEnLignes` qui découpe une ligne en plusieurs lignes dont la largeur totale ne dépasse pas une taille fixée. Un exemple de l'utilisation d'une telle méthode est donnée par la figure 1 ainsi que par le code de la classe `GUI` (voir partie étiquetée `PROBLEME Q3`). Vous remarquerez que l'affichage des lignes successives nécessite de considérer la hauteur maximum de la ligne précédente. En conséquence, il vous faut aussi compléter la méthode `hauteurLigne` de la classe `Ligne`.

Problème Question 4 : Sauvegarde

On désire pouvoir enregistrer des lignes (instances de la classe Ligne) afin de pouvoir, dans la méthode `main` par exemple, relire et reconstruire les lignes à partir d'un fichier pour ensuite les afficher. Complétez le code des différentes classes en conséquence et donnez une méthode main complète permettant de sauvegarder puis plus tard, de relire et afficher, les lignes ainsi sauvées.