Développement Logiciel

Éxamen 2011-2012 (Vendredi 1^{er} juin, 9h-11h)

*Seuls les documents liés **directement** au cours sont autorisés (comme dit sur le site) : sujets de TD, notes de cours, notes personnelles manuscrites, version imprimée du cours. En cas de doute, donnez la réponse qui vous semble la plus vraisemblable (et justifiez au besoin). Ce sujet contient X pages.*

N'OUBLIEZ PAS DE NUMÉROTER VOS FEUILLES, DE REPORTER VOS NOMS/PRÉNOMS ET DE RENDRE VOS COPIES ANONYMES.

Partie 1 : Architecture (5 pts)

Cet exercice est une simulation d'une bibliothèque. Il touche plus particulièrement l'organisation des classes qui expriment les différents ouvrages possédés par la bibliothèque. Voici une description du fonctionnement de la bibliothèque :

1. La **bibliothèque** à plusieurs types d'**ouvrages** : en **papier** (ex. livres) ou **électronique** (ex. DVDs). Tous les ouvrages ont une *date de publication* et le *nombre d'exemplaires disponibles* et *exemplaires possédés* par la bibliothèque.

2. Tous les ouvrages peuvent être **prêtable** ou **non-prêtable** (si ils sont trop vieux ou rares). Les ouvrages prêtables, nous pouvons *emprunter* un exemplaire pour une période *t*. Pour les ouvrages non-prêtables nous pouvons *accéder-sur-place* (c.à.d. les voir seulement dans la bibliothèque) un exemplaire pour une période *t'*. À la fin des périodes t ou t', nous retournons l'exemplaire de l'ouvrage avec l'action *retourner*.

3. Nous pouvons *photocopier* tous les ouvrages en papier, et *copier* tous les ouvrages électroniques.

4. Les ouvrages en papier sont les **romans**, les **bandes dessinées** et les **revues périodiques**. Ils ont tous un numéro *ISDN*, une *maison d'imprimerie* et un *titre*.

5. Les romans ont un *auteur* et un *genre* (policier, science-fiction, fantasy,...). Les bandes dessinées possèdent également un *auteur* et un *dessinateur* (exemple Uderzo pour Astérix). Et les revues possèdent un *éditeur* et une *table des matières*.

6. Les ouvrages électroniques sont des **DVD**s et des **CD**s. Ils ont tous un *titre* et un *label*.

7. Les DVDs ont un *type* (jeu, film, musique, ...) et les CDs ont une *table des matières*.

Question 1 (2 pts) : Décrivez l'architecture que vous utiliseriez pour la simulation de la bibliothèque en termes de classes (classes abstraites, interfaces, héritage), etc. Utilisez au **minimum** une classe abstraite, une interface et un héritage. Nous vous noterons sur la qualité de l'architecture.

Question 2 (1 pts) : Si vous ne l'avez pas déjà fait, décrivez les méthodes et les champs que vous voyez dans la description avec leur visibilité (private, public, etc). N'implémentez pas les méthodes, donnez seulement leur nom, visibilité et paramètres.

Question 3 (2 pts) : Nous allons maintenant ajouter des personnes dans la simulation qui veulent emprunter ou accéder à des exemplaires des ouvrages. Comment pouvez-vous assurer que les exemplaires sont disponibles ? Ajoutez **personnes** dans votre architecture et utilisez «synchronised» pour s'assurer que les exemplaires sont donnés aux personnes dès qu'ils sont disponibles.

Partie 2 : Compréhension de notions (6pts, 1 pt par question)

On vous demande des réponses courtes et précises (1 à 2 lignes max).

MÉTHODES DE PROGRAMMATION

Etudiez le code source présenté en annexe I.

Question 1 : Quelle est le nom complet de la classe ?

Question 2 : Donnez la visibilité des trois champs pour tous les cas (avec ou sans relation d'héritage, même package ou non)

Question 3 : Caractérisez cette classe (héritage ? etc.)

Question 4 : A quoi sert le mot clé Cloneable ?

ENTRÉES-SORTIES

Question 5 : Que se passe t'il si vous oubliez de fermer un fichier ? Quelle instruction faut il utiliser ?

Question 6 : A quoi sert le mot clé "transient" ? (donnez le contexte)

Partie 3 : SERIALISATION (3pts)

On reprend le programme dont le code source est décrit en annexe I. Ré-écrivez ce programme pour pouvoir sauvegarder et charger les valeurs stockées dans un objet. Attention à bien vérifier que le fichier existe lors du chargement.

Partie 4 : THREADS (3pts)

Le code présenté en annexe II simule une situation dans laquelle 20 threads (les objets Cyclistes) accèdent à une ressource partagée (10 Velibs). Ces objets Cyclistes essayent de prendre un vélo, le garder pour une période aléatoire, et le retourner pour que les autres Cyclistes puissent l'utiliser. Les Cyclistes s'arrêtent quand ils ont pris une velo.

Question 1 : Si on exécute ce code, les Cyclistes accèdent des vélos négatifs ! Ajoutez des testes pour assurer que les Cyclistes accèdent aux vélos seulement s'il y au minimum un disponible.

Question 2 : Comment est-ce qu'on peut assurer que les ressources (vélos) sont proprement partagés en utilisant «synchronised» dans le code ?

Question 3 : Est-ce qu'on peut avoir des blocages en utilisant en utilisant les fonction *wait()* et *notify()* ?

Partie 5 : Swing (3pts)

Regardez le code du programme donné en annexe III.

Question 1 : Dessinez le rendu graphique de ce programme.

Question 2 : Que permet de faire ce programme ?

FIN DU SUJET.

Annexe I

```
1 package picotoolbox;
  public class ParameterSet extends Properties implements Cloneable {
3     private ArrayList _keys = new ArrayList();
      ArrayList _values = new ArrayList();
5     protected String _name = "(no name)";

7     public ParameterSet () { }
      public ParameterSet ( String __name ) { this._name = __name; }
9 }
```

Annexe II

```
1  public class Velib {
       private int velos = 10;
3
       public void prendreVelo(int ID_cycliste)  {
5          System.out.println(ID_cycliste + " : Je prends velo : " + velos);
           --velos;
7      }

9      public void rendreVelo(int ID_cycliste) {
           ++velos;
11         System.out.println(ID_cycliste + " : Je returne velo : " + velos);
       }
13
       static public void main(String args[]) {
15         Velib V = new Velib();
           for (int i = 0; i <= 20; ++i) {
17             Cycliste c = new Cycliste(i, V);
               c.start();
19         }
       }
21 };

23 class Cycliste extends Thread {
       private Velib v;
25     private int ID;

27     public void init() { }

29     public Cycliste(int ID, Velib v) {
           this.ID = ID;
31         this.v = v;
       }
33
       public void run() {
35         try {
               v.prendreVelo(ID);
37
               sleep((int) (Math.random() * 200));
39
               v.rendreVelo(ID);
41         } catch (InterruptedException e) { }
       }
43 };
```

Annexe III

```
import java.io.File ;
import javax.swing.*;
import java.awt.*;

public class Mystery extends JFrame {
    private JLabel parentdirs;
    private JList filelist;
    private JButton btn_up, btn_open;
    private String directory, file;

    private ActionListener btnListener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JButton button = (JButton) e.getSource();
            if (button == btn_open) {
                file = (String) filelist.getSelectedValue();
                if (!show(getFilePath())) { parentdirs.setText(directory); }
            } else if (button == btn_up) { go_up(); }
        }
    };

    private ListSelectionListener listListener = new ListSelectionListener(){
        public void valueChanged(ListSelectionEvent e) {
            if (e.getValueIsAdjusting()) { return ; }
            btn_open.setEnabled(filelist.getSelectedIndex() != -1);
        }
    };

    public Mystery(String title) {
        super(title);

        Container pane = getContentPane();
        pane.setLayout(new BorderLayout());

        parentdirs = new JLabel();
        pane.add(parentdirs, BorderLayout.PAGE_START);

        filelist = new JList();
        filelist.addListSelectionListener(listListener);
        pane.add(new JScrollPane(filelist), BorderLayout.CENTER);

        JPanel buttons = new JPanel();
        buttons.setLayout(new BoxLayout(buttons, BoxLayout.LINE_AXIS));
        buttons.add(Box.createHorizontalGlue());
        btn_up = new JButton("Up");
        btn_up.addActionListener(btnListener);
        btn_open = new JButton("Open");
        btn_open.addActionListener(btnListener);
        buttons.add(btn_up);
        buttons.add(btn_open);
        pane.add(buttons, BorderLayout.PAGE_END);
```

```
51
            show("/");
53          pack();
            setVisible(true);
55      }

57      Boolean show(String path) {
            File dir = new File(path);
59          if (!dir.exists() || !dir.isDirectory()) { return false; }

61          directory = dir.getAbsolutePath();
            file = null;
63          btn_open.setEnabled(false);
            parentdirs.setText(directory);
65
            String[] files = dir.list();
67          if (files != null) {
                for (int i = 0; i < files.length; i++) {
69                  File f = new File(path, files[i]);
                    if (f.isDirectory())
71                      files[i] = files[i] + File.separator;
                }
73              filelist.setListData(files);
            }
75          return true;
        }
77
        private void go_up() {
79          String path = "/";
            String[] all_parents = directory.split(File.separator, 0);
81          for (int i = 0; i < all_parents.length - 1; ++i)
                path += all_parents[i] + File.separator;
83          show(path);
        };
85
        public String getFilePath() {
87          if (directory == null || file == null) { return null; }
            return directory + File.separator + file;
89      }

91      static public void main(String args[]) {
            FileSelector fs = new Mystery ("...");
93      }

95  }
```

FIN DES ANNEXES