Développement Logiciel

Éxamen session 1 - 2012-2013 (Vendedi 25 mai, 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, livres. En cas de doute, donnez la réponse qui vous semble la plus vraisemblable (et justifiez au besoin). Ce sujet contient* ***7*** *pages.*

N'oubliez pas de numéroter vos feuilles, de reporter vos noms/prénoms et d'anonymiser vos copies*.*

Partie 1 : Questions de cours (7pts, 1 pt par question)

On vous demande des réponses **courtes et précises** (2 à 3 lignes max).

MÉTHODES DE PROGRAMMATION

Etudiez le code source présenté en annexe I

Q1 : Quel est le nom complet de la classe ?

Q2 : Donnez la visibilité des champs de la classe dans les quatre cas suivants : accès depuis l'extérieur, depuis une sous-classe, depuis le même package, depuis la classe elle-même.

ENTRÉES-SORTIES

Q3 : Pourquoi utilise-t-on un flux de données `Buffered` (BufferedReader ou Writer) ?

Q4 : Quelle est la différence entre l'interface `Serializable` et `Externalizable` ?

INTERFACE GRAPHIQUE

Q5. Comment se comporte l'image quand on redimensionne la fenêtre en annexe I ?

Q6. Une fenêtre peut-elle être considérée comme un *thread* ? Justifiez votre réponse.

Q7. Qu'est qu'un `ActionListener` ?

Partie 2 : Sérialisation (3pts)

La classe SerializationDemo (annexe II) est sérialisable.

Q1 : Complétez les fonctions writeObject et readObject.

Q2 : La méthode main(...) teste la sérialisation et la dé-sérialisation des 2 copies d'un objet, et vérifie qu'ils ont les mêmes valeurs dans leur champs. Cependant, en l'état, une erreur (assertion) est retournée. Quelle erreur a été commise par le développeur ?

Q3 : Proposez une solution (en code) pour résoudre le problème (vous pouvez introduire des méthodes).

Partie 3 : Threads (4pts)

Le code présenté en annexe III simule une ligne d'assemblage dans une usine de production automobile. L'équipage `Carrosserie` (un thread) assemble la carrosserie de la voiture, et place la voiture dans une unité de stockage avec une capacité de **50 voitures** (`ArrayList`). Le thread `Carrosserie` peut produire **1** voiture toutes les 30-60 minutes. L'équipage `Moteur` (un autre thread) a un travail plus facile : il peut retirer une voiture de l'unité de stockage (s'il y en a une), ajouter le moteur en 30 minutes et la mettre sur une autre chaîne d'assemblage (qui mettra par exemple les roues, la peinture, etc.). Si l'unité de stockage (`ArrayList`) est vide, l'équipe `Moteur` doit attendre que l'équipe `Carrosserie` produise une voiture et l'ajoute dans l'unité de stockage. Si l'unité de stockage est pleine (c.à.d. qu'elle contient 50 voitures) l'équipe `Carrosserie` doit arrêter sa production jusqu'au moment où l'équipe `Moteur` aura retiré une voiture pour s'en occuper. La simulation tourne jusqu'à ce que 500 voitures sont produites et traitées.

Q1 : Si on exécute ce code, rien ne se passe. Pourquoi ? Modifiez le code pour que la simulation puisse commencer.

Q2 : La modification apportée à la question précédente ne suffit pas, car on ne teste pas si l'unité de stockage a de la place pour accueillir une voiture, ni si elle contient une voiture qui peut être prise. Modifiez le code pour assurer que les opérations de traitement de l'unité de stockage (`prendre()` et `mettre()`) fonctionnent correctement.

Q3 : Comment s'assurer que les ressources (unité de stockage) soient correctement partagées en utilisant le mot-clef `synchronized` et la méthode `Thread.sleep(long time)` dans le code ?

Q4 : Où mettre les méthodes `object.wait()` et `object.notify()` pour réduire le temps d'attente pour mettre/prendre une voiture ? Donnez le code nécessaire.

Partie 4 : Swing (2pts)

Lisez le code du programme donné en annexe IV.

Q1 : Dessinez le rendu graphique de ce programme.

Q2 : Que permet de faire ce programme ?

Partie 5 : Architecture et Programmation (4 pts)

Cet exercice concerne la simulation d'un zoo et plus particulièrement l'organisation des classes qui expriment les différents animaux s'y trouvant. Voici une description du fonctionnement de ce petit zoo :

1. Le **zoo** a plusieurs types d'**animaux** : **oiseaux** (ex. pingouin, aigle), **reptiles** (ex. crocodile, tortue), et **mammifères** (ex. lion, loutre). À chaque animal est associé un *nom scientifique*, leur *sexe*, leur *age*, leur *nom* et *pays d'origine* (tous ces attributs sont des chaînes de caractères). Á chaque animal est aussi associé un *nombre d'actions possibles* (il s'agit d'un entier).

2. Les animaux ont des actions communes (*manger, dormir*), et des actions spécifiques à leur espèce (*voler* pour l'aigle, *nager* pour le pingouin, crocodile ou la loutre, *marcher* pour le pingouin, crocodile, lion, la tortue ou la loutre). Les actions prennent comme paramètre la durée de l'action t en secondes.

Q1 : Décrivez l'architecture que vous utiliseriez pour exprimer les classes des animaux pingouin, aigle, crocodile, tortue, lion et loutre. Utilisez au **minimum** une classe abstraite, une interface et un héritage. Nous vous noterons sur la qualité de l'architecture.

Q2 : 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.

Q3 : Chaque animal est un *thread* qui tourne indéfiniment. Lorsque le thread tourne, celui-ci fait **prendre à l'animal une action aléatoire** parmi ses actions possibles, pour une durée aléatoire (vous pouvez utiliser la fonction `Random.nextInt(Integer n)` pour récupérer un nombre aléatoire). Implémentez la méthode *run()* qui définit ce comportement aléatoire.

Q4 : Nous souhaitons désormais construire la simulation. Créez une classe `Zoo` qui contient l'ensemble des animaux. Dans la fonction *main()* (se trouvant dans cette classe), créez 2 animaux de chaque classe d'animaux (c.à.d. 2 pingouins, 2 aigles, etc) et commencez leur vie dans le zoo (n'oubliez pas de démarrer les *threads*).

FIN DU SUJET

Annexe I

```
package imagetests;

import java.awt.*;
import javax.*;

public class myImageResize extends JPanel {

    BufferedImage image;
    double ratio;

    public myImageResize  (BufferedImage image) {
        this.image = image;
        ratio = image.getHeight() / image.getWidth();
    }

    protected void paintComponent(Graphics g) {
        g.drawImage(image, 0, 0, getWidth(), (int) (getWidth() * ratio), 0, 0,
        image.getWidth(), image.getHeight(), this);
    }

    public static void main(String[] args) throws IOException {
        BufferedImage image = ImageIO.read(new File("image.png"));
        myImageResize  myDemo = new myImageResize (image);

        JFrame f = new JFrame();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new JScrollPane(myDemo));
        f.setSize(400, 400);
        f.setVisible(true);
    }
}
```

Annexe II

```
import java.io.*;
import java.util.Vector;

public class SerializationDemo implements Serializable {

    int a;
    char c;
    Vector<Double> v;

    public SerializationDemo(int a, char c, double[] elements) {
        this.a = a;
        this.c = c;
        this.v = new Vector<Double>(elements.length);
        for (double ev : elements) {
            v.add(new Double(ev));
        }
    }
```

```
19      private void writeObject(ObjectOutputStream out) throws IOException {
            // PARTIE A COMPLETER
21      }

23      private void readObject(ObjectInputStream in) throws IOException,
                ClassNotFoundException {
25          // PARTIE A COMPLETER
        }
27
        public static void main(String[] args) {
29          SerializationDemo test = new SerializationDemo(10, 'a', new double[] {
                    5, .16, 7.3, 44.33 });
31          SerializationDemo test1 = null;
            SerializationDemo test2 = null;
33
            try {
35              ObjectOutputStream out1 = new ObjectOutputStream(
                        new FileOutputStream("monFichier1.sav"));
37              out1.writeObject(test);
                out1.close();
39              ObjectOutputStream out2 = new ObjectOutputStream(
                        new FileOutputStream("monFichier2.sav"));
41              out2.writeObject(test);
                out2.close();
43          } catch (FileNotFoundException e) {
                e.printStackTrace();
45          } catch (IOException e) {
                e.printStackTrace();
47          }

49          try {
                ObjectInputStream in1 = new ObjectInputStream(new FileInputStream(
51                      "monFichier1.sav"));
                test1 = (SerializationDemo) (in1.readObject());
53              in1.close();
                ObjectInputStream in2 = new ObjectInputStream(new FileInputStream(
55                      "monFichier2.sav"));
                test2 = (SerializationDemo) (in1.readObject());
57              in1.close();
            } catch (FileNotFoundException e) {
59              e.printStackTrace();
            } catch (IOException e) {
61              e.printStackTrace();
            } catch (ClassNotFoundException e) {
63              e.printStackTrace();
            }
65
            assert (test1 != test2) : "Probleme avec la serialization !";
67      }
    }
```

Annexe III

```
public class Assemblage {
    ArrayList<Integer> stockage = new ArrayList<Integer>();
    public int i = 0;

    public Integer prendre() {
        Integer voiture = stockage.remove(stockage.size() - 1);
        System.out.println("Prendre voiture" + i);
        return i;
    }

    public void mettre() {
        System.out.println("Mettre voiture" + i);
        ++i;
        stockage.add(i);
    }

    public static void main(String[] args) {
        Assemblage myAssemblage = new Assemblage();
        Carrosserie c = new Carrosserie(myAssemblage);
        Moteur m = new Moteur(myAssemblage);
    }
}

class Carrosserie extends Thread {
    private Assemblage myAssemblage;

    public Carrosserie(Assemblage myAssemblage) {
        this.myAssemblage = myAssemblage;
    }

    public void run() {
        while (myAssemblage.i < 500) {
            myAssemblage.mettre();
            try {
                Thread.sleep(1800 + random.nextInt(1800));
            } catch (InterruptedException e) { }
        }
    }
}

class Moteur extends Thread {
    private Assemblage myAssemblage;

    public Moteur(Assemblage myAssemblage) {
        this.myAssemblage = myAssemblage;
    }

    public void run() {
        Random random = new Random();

```

```
        while (myAssemblage.i <= 500) {
52          Integer voiture = myAssemblage.prendre();
            try {
54              Thread.sleep(1800);
            } catch (InterruptedException e) { }
56      }
    }
58  }
```

Annexe IV

```
import java.awt.*;
2   import java.awt.event.*;
    import javax.swing.*;
4
    public class ProgressBarDemo extends JFrame {
6
        private static final long serialVersionUID = 1L;
8       int count = 0;
        JProgressBar jBar = new JProgressBar();
10      GridLayout layout = new GridLayout(2, 1);
        JButton btnAction = new JButton("Go!");
12
        public ProgressBarDemo() {
14          setTitle("Count and Go");

16          Container contentPane = getContentPane();
            contentPane.setLayout(layout);
18          contentPane.add(jBar);
            contentPane.add(btnAction);
20
            jBar.setMinimum(0);
22          jBar.setMaximum(10);

24          btnAction.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
26                  count++;
                    jBar.setValue(count);
28                  if (count == 10)
                        System.exit(-1);
30              }
            });
32          pack();
            setVisible(true);
34      }

36      public static void main(String[] args) {
            new ProgressBarDemo();
38      }
    }
```

FIN DES ANNEXES