Développement Logiciel

Éxamen session 1 - 2013-2014 (Jeudi 22 mai, 9:30h-11:30h)

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 (9pts, 1 pt par question)

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

MÉTHODES DE PROGRAMMATION

Étudiez le code source présenté en annexe I

Q1 : Quel est le nom complet de la classe ?

Q2 : Caractérisez cette classe (héritage, interfaces, etc.).

PROGRAMMATION CONCURRENTIELLE

Q3 : S'il y a deux méthodes `synchronized` dans une classe, peuvent-elles être exécutées simultanément par deux threads ? Expliquez.

Q4 : Quelle est la différence entre `sleep()` et `wait()` ?

INTERFACE GRAPHIQUE

Q5. Quel est le rendu graphique de l'annexe I ?

Q5. Comment se comporte le rendu graphique quand on redimensionne la fenêtre de l'annexe I ?

Q7. Qu'est ce qu'un `LayoutManager` ? Donnez un exemple.

ENTRÉES-SORTIES

Q8. Que se passe-t-il si vous oubliez de fermer un fichier ? Quelle instruction faut il utiliser ?

Q9. Pourquoi utilise-t-on la classe `File` ?

Partie 2 : Threads (3pts)

Le code présenté en annexe II simule une situation dans laquelle 20 threads (représentés chacun par une instance de **Lecteur**) accèdent à une ressource partagée (les 5 copies de 2 œuvres de littérature représentées par deux entiers dans une instance de la classe **Ouvrage**). Ces lecteurs tentent de prendre une copie d'un des deux ouvrages, de la garder pour une période aléatoire, et de la retourner pour que les autres lecteurs puissent l'emprunter. Lorsqu'ils rendent la copie, leur thread respectif (instance de **Lecteur**) s'arrête.

Q1 : Si on exécute ce code, les lecteurs accèdent à des copies négatives ! Ajoutez des tests pour s'assurer que les lecteurs puissent prendre une copie d'un ouvrage uniquement s'il y en a au moins une de disponible.

Q2 : Comment s'assurer que les ressources (copies) soient correctement partagées en utilisant le mot-clef **synchronized** et la méthode **Thread.sleep(long time)** dans le code ?

Q3 : Où faut-il mettre les méthodes **object.wait()** et **object.notify()** pour réduire le temps d'attente pour prendre une copie ? Donnez le code nécessaire. Attention, nous avons deux ouvrages différents auxquels on accède dans **Ouvrage**.

Partie 3 : Sérialisation (2pts)

La classe SerializationDemo (annexe III) est sérialisable.

Q1 : Complétez les fonctions writeObject et readObject, en prenant en compte le fait qu'on ne veut pas inclure l'objet `File` (paramètre f) dans la sérialisation, on ne veut garder que la chaîne de caractères correspondant au chemin du fichier. Cette chaîne de caractères sera utilisée lors de la dé-sérialisation pour récupérer l'objet `File`.

Q2 : La méthode main(...) permet de tester la sérialisation et la dé-sérialisation. Cependant, en l'état, une erreur est retournée (FileNotFoundException). Quelle erreur a été commise par le développeur dans sa méthode de test ?

Partie 4 : Swing (2pts)

La Figure 1 montre une boîte de dialogue permettant de sélectionner une couleur de différentes manières.

Q1 : Identifiez les composants Swing (widgets) qui s'y trouvent. **Note :** Vous pouvez faire une copie de la figure sur votre copie et associer directement une lettre unique à chaque composant identifié, comme nous l'avons fait ci-dessous avec le bouton "OK" (lettre A), et référencez ensuite cette lettre sur votre copie.

Q2 : Identifiez les écouteurs d'événements (Listener) pour les composants qui en nécessitent un et spécifiez le plus précisément possible le type d'événement qu'ils écoutent. Par exemple **MouseClick** est moins précis que **ValueChangeEvent** (même si vous ne connaissez pas le nom exact de l'écouteur d'événements, expliquez le type d'événements qu'il écoute).

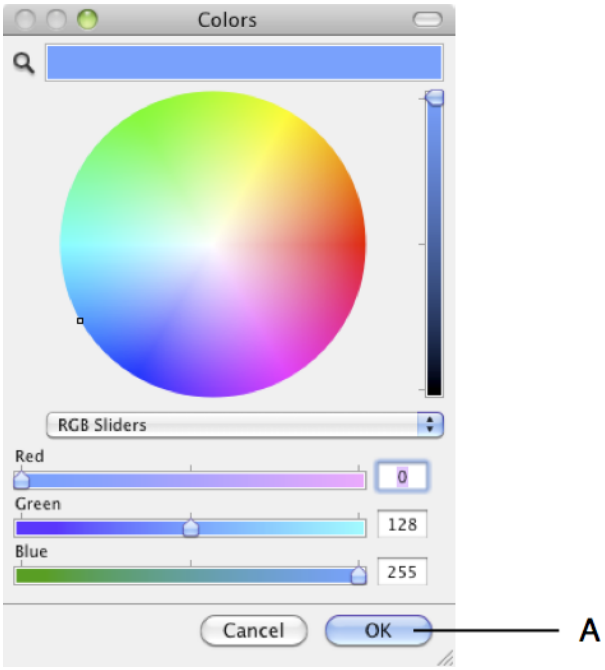

FIGURE 1 – Sélecteur de couleur.

Partie 5 : Architecture et Programmation (4 pts)

Vous concevez une simulation pour un jeu en ligne. Votre but est de créer des NPC (Non-Player Characters, ou personnages non-joueurs) pour peupler le **monde** du jeu. Ces NPC interagiront avec les vrais joueurs à travers leurs actions. Chaque NPC a plusieurs actions disponibles.

1. Tous les **NPC** ont des propriétés communes, telles que leur *sexe*, leur *nom*, leur *taille*, leur *poids* et leur *âge*.

2. Le **monde** a plusieurs races de **NPC** : **Humains**, **Elfes**, **Nains** et **Orques**.

3. Chacune de ces races a des capacités spéciales, exprimées comme des actions/méthodes disponibles. Les **Humains** peuvent *parler* avec toutes les autres races. Toutes les autres races peuvent *parler* avec leur propre race et les Humains. Les **Elfes** peuvent *donner des cartes au trésor*, Les **Orques** peuvent *transporter* des joueurs, et les *Nains* peuvent *vendre des armures*.

4. Les **NPC** peuvent avoir une orientation plutôt **bonne** ou **mauvaise**. S'ils sont **bons**, ils ont en plus les actions : *guérir* et *aider*. S'ils sont **mauvais**, ils ont en plus les actions : *attaquer* et *ignorer*.

Q1 : Décrivez l'architecture que vous utiliseriez pour exprimer les classes des Elfes, bons ou mauvais, des Humains, bons ou mauvais, etc. 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 NPC est un *thread* qui tourne indéfiniment. Lorsque le thread tourne, celui-ci fait **prendre au NPC une action aléatoire** parmi ses actions possibles (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 `Monde` qui contient l'ensemble des NPC. Dans la fonction *main()* (se trouvant dans cette classe), créez 2 NPC de chaque race et orientation (bon ou mauvais), c.à.d. 2 Elfes, 2 Humais, 2 Orques, 2 Nains, et commencez leur vie dans le monde (n'oubliez pas de démarrer les *threads*).

FIN DU SUJET

Annexe I

```
1
package fr.lri.MySwingDemos;
3
public class SwingDemo extends JFrame implements ActionListener {
5
    public void actionPerformed(ActionEvent event) {
7       JButton b = (JButton) (event.getSource());
        System.out.println(b.getText());
9   }

11  public void init() {
        Container cp = getContentPane();
13      this.setTitle("example 5");
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
15
        cp.setLayout(new GridLayout(5, 3));
17      for (int i = 0; i < 15; i++) {
            JButton b = new JButton("Button " + i);
19          b.addActionListener(this);
            cp.add(b);
21      }
    }
23
    public static void main(String[] args) {
25      SwingDemo frame = new SwingDemo();
        frame.init();
27      frame.setSize(150, 250);
        frame.setVisible(true);
29  }
}
```

Annexe II

```
1  public class Ouvrage {

3      private int copiesOuvrage0 = 5;
       private int copiesOuvrage1 = 5;
5
       public void prendreOuvrage( int ID_lecteur, int ouvrage)  {
7          if (ouvrage == 0){
               System.out.println(ID_lecteur + " : Je prends la copie   " +
9                                 copiesOuvrage0 + " de l'ouvrage 0");
               --copiesOuvrage0;
11         }else{
               System.out.println(ID_lecteur + " : Je prends la copie   " +
13                                 copiesOuvrage1 + " de l'ouvrage 1");
               --copiesOuvrage1;
15         }
       }
17
```

```
19      public void rendreOuvrage(int ID_lecteur, int ouvrage) {
            if(ouvrage == 0){
21              ++copiesOuvrage0;
                System.out.println(ID_lecteur + " : Je retourne la copie : " +
23                              copiesOuvrage0 + " de l'ouvrage 0");
            }else{
25              ++copiesOuvrage1;
                System.out.println(ID_lecteur + " : Je retourne la copie : " +
27                              copiesOuvrage1 + " de l'ouvrage 1");
            }
29      }

31      static public void main(String args[]) {
            Ouvrage O = new Ouvrage();
33          for (int i = 0; i <= 20; ++i) {
                Lecteur l = new Lecteur(i, O);
35              l.start();
            }
37      }
    };
39
    class Lecteur extends Thread {
41      private Ouvrage o;
        private int ID;
43
        public void init() { }
45
        public Lecteur(int ID, Ouvrage o) {
47          this.ID = ID;
            this.o = o;
49      }

51      public void run() {
            try {
53              o.prendreOuvrage(ID, ID%2);

55              sleep((int) (Math.random() * 200));

57              o.rendreOuvrage(ID, ID%2);
            } catch (InterruptedException e) { }
59      }
    };
```

Annexe III

```
2   public class SerializationDemo implements Serializable {
        int a;
4       char c;
        File f;
6
```

```
8       public SerializationDemo(int a, char c, File f) {
            this.a = a;  this.c = c;  this.f = f;
10      }

12      @Override
        public String toString() {
14          return "" + a + ", "  + c + ", " + f.getAbsolutePath();
        }
16
        private void writeObject(ObjectOutputStream out) throws IOException {
18          // PARTIE A COMPLETER
        }
20
        private void readObject(ObjectInputStream in) throws IOException,
22              ClassNotFoundException {
            // PARTIE A COMPLETER
24      }

26      public static void main(String[] args) {
            SerializationDemo test = new SerializationDemo(10, 'a',
28                                  new File("fichier.txt")));
            SerializationDemo test2 = null;
30
            try {
32              ObjectOutputStream out = new ObjectOutputStream(
                        new FileOutputStream("monFichierEcriture.sav"));
34              out.writeObject(test);
                out.close();
36          } catch (FileNotFoundException e) {
                e.printStackTrace();
38          } catch (IOException e) {
                e.printStackTrace();
40          }

42          try {
                ObjectInputStream in = new ObjectInputStream(new FileInputStream(
44                      "monFichierLecture.sav"));
                test2 = (SerializationDemo) (in.readObject());
46              in.close();
            } catch (FileNotFoundException e) {
48              e.printStackTrace();
            } catch (IOException e) {
50              e.printStackTrace();
            } catch (ClassNotFoundException e) {
52              e.printStackTrace();
            }
54
            System.out.println(test);
56          System.out.println(test2);
    }
```

FIN DES ANNEXES