Java et modélisation objet : Devoir surveillé

CIR3 - 2h00 - Pas de documents additionnels - Pas de supports

23 mars 2019

Consignes

Pour chaque question, vous veillerez à donner une réponse claire, synthétique et argumentée. Vous n'hésiterez pas à vous appuyer sur des schémas simples pour illustrer votre propos.

Exercice 1 (5 points)

1. Expliquez les trois principes fondamentaux de Java : Encapsulation, héritage, polymorphisme.
2. Quelle est la différence entre les fonctions **sleep** et **wait** dans le cadre de la programmation multithread ?
3. Quel est le rôle du mot-clé **final** ?
4. Quelle différence entre upcasting et downcasting ?

Exercice 2 (2 points)

On considère le code suivant :

```
public class Main {
  public static int f(int a,int b){
    return ((b==1)?a:a*f(a,b-1));
  }

  public static void main(String[] args) {
      System.out.println(f(5,2));
  }
}
```

Quel est le rôle de la fonction f ? Vous détaillerez l'exécution de la fonction pas à pas.

Exercice 3 (4 points)

A partir de la classe Song suivante :

```
public class Song {
    public String artist;
    public int duration;
    public String title;
}
```

1. L'attribut duration représente une durée en secondes. En utilisant une classe que vous définirez, proposez une solution pour que l'attribut duration soit représenté par deux valeurs minutes et seconds (sous forme d'entiers). Vous écrirez toutes les classes nécessaires et la classe ainsi modifiée.
2. On suppose une liste définie ainsi :

```
ArrayList<Song> arls = new ArrayList<Song>();
```

   On suppose la liste remplie de n instances de la classe Song. Proposez une fonction qui transforme cette liste en map, le titre étant la clé de la map.
3. A partir de la map créée, proposez une solution pour en supprimer tous les éléments d'une durée inférieure à 1 minute (on suppose que la durée est définie par la classe écrite plus haut).

Exercice 4 (4 points)

On suppose le code suivant :

```
public class Chat {
  private final Poil couleurPoil;
  private final int age;
  private final String nom;

  public Poil getCouleur() {
    return couleurPoil;
  }

  public int getAge() {
    return age;
  }

  public Chat(int age_, String couleurPoil_, String nom_) {
    age = age_;
    couleurPoil = new Poil(couleurPoil_);
    nom = nom_;
  }

  public void printCatInfo() {
    System.out.println("Age:"+this.age);
    System.out.println("Poil:"+this.getCouleur().getValeur());
    System.out.println("Nom:"+this.nom);
  }

  public static void main(String[] args) {
    Chat c = new Chat(7,"Blanc","Nom");
    c.printCatInfo();
  }
}


public class Poil {
  private final String valeur;

  public String getValeur() {
    return valeur;
  }

  public Poil(String couleurPoil) {
    this.valeur = couleurPoil;
  }
}
```

On rappelle ici la définition suivante :

La loi de Déméter est une règle de conception pour développer un logiciel, particulièrement du logiciel orienté objet. Cette règle peut être résumée en "Ne parlez qu'à vos amis immédiats". La notion fondamentale est qu'un objet devrait faire aussi peu d'hypothèses que possible à propos de la structure de quoi que ce soit d'autre, y compris ses propres sous-composants.

La Loi de Déméter pour les fonctions requiert que toute méthode f d'un objet O peut simplement invoquer les méthodes des types suivants d'objets :

- O lui-même
- Les paramètres de f
- Les objets que f instancie
- Les objets membres de O (les attributs)

Expliquez en quoi le code proposé est une violation de la loi de Déméter et proposez les modifications adéquates.

Exercice 5 (5 points)

On considère le code suivant :

```java
import java.util.ArrayList;
import java.util.concurrent.locks.ReentrantLock;

public class GetSpot extends Thread {
  public ArrayList<Integer> al;

  public GetSpot(ArrayList al_) {
    al = al_;
  }
  public void run() {
    for(int i=0;i<100;i++) {
        al.add(i);
    }
  }

  public static void main(String[] args) {
    ArrayList<Integer> al = new ArrayList<Integer>();

    new GetSpot(al).start();
    new GetSpot(al).start();
    new GetSpot(al).start();

    int i =0;
    while(i<al.size()) {
      System.out.println(al.get(i)+" ");
      i++;
    }
  }
}
```

1. Expliquez les problèmes renvoyés à l'exécution par le code suivant.
2. Proposez, en utilisant des solutions de synchronisation, une solution à ce problème.
3. On souhaite à présent être certain de n'afficher la liste al que lorsque tous les autres threads ont fini leur exécution (C'est à dire lorsque la liste est définitivement remplie). Modifiez le code en conséquence.
4. On souhaite garantir que la liste al sera remplie comme suit : $0,0,0,1,1,1,2,2,2,3,3,3 \ldots 99,99,99$. Comment garantir ce comportement ? Modifiez le code en conséquence.