**Durée : 1h AUCUN DOCUMENT - AUCUNE CALCULATRICE**

***Lisez attentivement chaque partie du sujet avant de vous lancer dans vos réponses.***

***Commentez brièvement votre code et importez les librairies nécessaires avant chaque classe.***

Des extraits de la documentation Javadoc sont fournis en **annexe**

Présentation du sujet : (source : examen du CNAM 2008)

L’objectif de ces exercices est de définir des tâches hiérarchiques et de les exploiter de différentes manières. L’architecture des tâches est donné par le diagramme des classes à la figure 1, où le détail des classes TacheElementaire et TacheComplexe n’est pas donné. Une tâche est caractérisée par un nom et un coût. Une tâche est soit une tâche élémentaire, soit une tâche complexe qui est alors composée de sous-tâches. Il est ainsi possible d’ajouter ou supprimer une sous-tâche à une tâche complexe, de trier une tâche complexe par coût de sous-tâches ou de sérialiser une tâche complexe.

«interface»

Tache

getCout(): int

getNom(): String

TacheComplexe

TacheElementaire

0..\*

sousTaches

FIG. 1 – Diagramme de classes

# Exercice 1 : Compréhension de l’architecture des tâches (6 points). Durée max : 15 min

* 1. Pourquoi Tache est-elle une interface ? Expliquer les relations utilisées dans ce diagramme de classes. (**2 points**)

C’est une interface car elle ne contient que des prototypes de méthodes. (1 point)

Les classe TacheElementaire et TacheComplexe implémentent l’interface Tache. TacheComplexe a une relation de composition avec Tache. (1 point)

**1.2** Le listing 1 ci-dessous donne le code de l’interface Tache. Écrire en Java la classe TacheElementaire qui implémente les interfaces Tache et Comparable, en y définissant les attributs privés, le constructeur paramétré et la méthode compareTo pour comparer le coût de 2 tâches par ordre croissant. (**4 points**)

Listing 1 – L’interface Tache

1. **public interface** Tache {
2. */\*\* Obtenir le nom de la tâche. \*/*
3. String getNom();

4

1. */\*\* Obtenir le coût de la tâche. \*/*
2. **int** getCout();
3. }

**Solution :**

1. **import** java.util.Collections;
2. **public class** TacheElementaire **implements** Tache, Comparable {
3. **private** String nom;
4. **private int** cout;

4

1. **public** TacheElementaire(String nom, **int** cout) {
2. **this**.nom = nom;
3. **this**.cout = cout;
4. }

9

1. **public** String getNom() {
2. **return this**.nom;
3. }

13

1. **public int** getCout() {
2. **return this**.cout;
3. }
4. **public int** compareTo(Tache t) {
5. **return** this.getCout()-t.getCout() ;
6. }

21

22 }

# Exercice 2 : Définition et sérialisation d’une tâche complexe (8 points). Durée max : 25 min

Nous nous intéressons maintenant à la classe TacheComplexe, en particulier à sa relation avec l’interface Tache. Une tâche complexe est composée d’un nombre quelconque de tâches. Le coût d’une tâche complexe est la somme des coûts des tâches qui la composent.

**2.1** Écrire en Java la classe TacheComplexe qui implémente les interfaces Tache et java.io.Serializable (0,5 pt). Elle contient les caractéristiques suivantes : (**5 points**)

* Des attributs privés : une ArrayList de tâches et le nom (0,5 pt)
* Un constructeur paramétré (0,5 pt)
* Une méthode ajouter qui ajoute une tâche et une méthode supprimer qui la supprime (1 pt)
* Une méthode trier qui trie les tâches par ordre croissant de coût (0,5 pt)
* une méthode serialiser qui sérialise la tâche complexe dans un fichier dont le nom est en paramètre, ferme le fichier et propage l’exception IOException (1 pt)
* La méthode getCout retourne le coût total des tâches (0,5 pt)
* La méthode getTaches retourne les tâches (0,5 pt)

**Solution :**

1. **import** java.util.Collections;
2. **import** java.util.ArrayList;
3. **import** java.io.\*;

4

1. **public class** TacheComplexe **implements** Tache, Serializable {
2. **private ArrayList**<Tache> sousTaches;
3. **private** String nom;

8

1. **public** TacheComplexe(String nom) {
2. **this**.nom = nom;
3. **this**.sousTaches = **new** ArrayList<Tache>();
4. }

13

1. **public void** ajouter(Tache tache) {
2. **this**.sousTaches.add(tache);
3. }

17

1. **public void** supprimer(Tache tache) {
2. **this**.sousTaches.remove(tache);
3. }

21

1. **public** String getNom() {
2. **return this**.nom;
3. }

25

1. **public int** getCout() {
2. **int** result = 0;
3. **for** (Tache t : sousTaches) {
4. result += t.getCout();
5. }
6. **return** result;
7. }

33

1. **public ArrayList**<Tache>getTaches() {
2. **return** sousTaches ;
3. **}**
4. **public void** trier(){
5. **Collections.sort**(soustaches) ;
6. }
7. **public void** serialiser(String nomfic)**throws** IOException{
8. **FileOutputStream** fos = **new** **FileOutputStream**(nomfic);
9. **ObjectOutputStream** oos = new **ObjectOutputStream**(fos);
10. **oox.writeObject(this);**
11. **}**

47 }

**2.2** Ecrire la classe TestTache avec le main qui effectue les traitements suivants : (**3 points**)

* Instancie une tâche complexe de nom « A »
* Instancie 2 tâches élémentaires : la première de nom « A1 » et de coût 20, la seconde de nom « A2 » et de coût 10 (0,5 pt)
* Ajoute les 2 tâches élémentaires à la tâche complexe (0,5 pt)
* Affiche le coût total de la tâche complexe et le nom et coût de ses tâches triées (1 pt)
* Sérialise la tâche complexe dans un fichier nommé « Tache.ser » et affiche un message d’erreur en cas d’exception IOException (1 pt)

1. **public class** TestTache1 {
2. **public static void** main(String[] args) {
3. TacheComplexe tA = **new** TacheComplexe("A");
4. tA.ajouter(**new** TacheElementaire("A1", 10));
5. tA.ajouter(**new** TacheElementaire("A2", 20));
6. System.out.println("Cout de tA = " + tA.getCout());



1. tA.trier() ;
2. **for** (Tache t : tA.getTaches() {
3. System.out.println("Nom et Cout = " + t.getNom() + t.getCout());



1. **try** {
2. tA.serialiser("Tache.ser") ;
3. }
4. **catch**(IOException) {
5. System.out.println("Erreur de sérialisation de la tâche") ;
6. }
7. }
8. }

# Exercice 3 : Interface graphique d’une tâche complexe (6 points). Durée max : 20 min

Nous définissons maintenant une interface graphique en Swing qui permet d’ajouter de nouvelles sous-tâches à une tâche complexe. Le code partiel de cette classe TacheComplexeSwing est donné au listing 2 ci-dessous.

Listing 2 – La classe TacheComplexeSwing

1. **import** javax.swing.\*;
2. **import** java.awt.\*;
3. **import** java.awt.event.\*;

4

5 **public class** TacheComplexeSwing **extends** JFrame **implements** ActionListener{

6

1. **private** TacheComplexe tache;
2. **final private** JTextField valeurNom = **new** JTextField(10);
3. **final private** JTextField valeurCout = **new** JTextField(10);
4. **final private** JButton Ajouter = **new** JButton("Ajouter");
5. **final private** JButton Quitter = **new** JButton("Quitter");

13

1. **public** TacheComplexeSwing(TacheComplexe tache) {
2. **super**("Nouvelletâche");



1. **this**.tache = tache;
2. Container c = **this**.getContentPane();
3. c.setLayout(**new** BorderLayout());
4. JPanel informations = **new** JPanel(**new** GridLayout(2,2));
5. informations.add(**new** JLabel("Nom : ", SwingConstants.RIGHT));



1. informations.add(valeurNom);
2. informations.add(**new** JLabel("Coût : ", SwingConstants.RIGHT));



1. informations.add(valeurCout);
2. c.add(informations, BorderLayout.CENTER);

24

1. JPanel boutons = **new** JPanel(**new** FlowLayout());
2. boutons.add(Ajouter);
3. boutons.add(Quitter);
4. c.add(boutons, BorderLayout.SOUTH);

29

30

1. **this**.pack();
2. **this**.setVisible(**true**);
3. }
4. **public static void** main(String[] args) {
5. **new** TacheComplexeSwing(**new** TacheComplexe("Test TacheComplexeSwing"));
6. }
7. }

**3.1** Dessiner la fenêtre (et les composants graphiques qu’elle contient) telle qu’elle est affichée quand cette classe est exécutée. **(2 points)**



**3.2** Compléter cette classe pour que les boutons Ajouter et Quitter deviennent actifs. Le bouton Quitter ferme la fenêtre. Le bouton Ajouter ajoute une nouvelle sous-tâche à la tâche complexe passée en paramètre du constructeur de cette classe : voir méthode ajouter de la question **2.1**. Le nom et le coût de cette sous-tâche sont saisis par l’utilisateur dans les zones de saisie prévues, valeurNom et valeurCout. Dans le cas où l’utilisateur saisit une information qui n’est pas un entier pour saisir le coût, on signalera l’erreur en mettant la couleur de fond de la zone de saisie correspondante en rouge (setBackground(Color.RED)). On rappelle que la méthode Integer.parseInt(String) renvoie l’entier correspondant à la chaîne de caractère passé en paramètre. Cette méthode lève l’exception NumberFormatException si la chaîne ne correspond pas à un entier. **(4 points)**

Quitter.addActionListener(**this**) ;

Ajouter.addActionListener(**this**) ;

**public void** actionPerformed(ActionEvent ev) {

**try** {

Object o = ev.getSource();

If (o == Ajouter) {

String nom = valeurNom.getText();

**int** cout = Integer.parseInt(valeurCout.getText());

tache.ajouter(**new** TacheElementaire(nom, cout));

System.out.println("cout total = " + tache.getCout());

}

else {

System.out.println("Appui sur Quitter...");

fenetre.dispose();

}

}

**catch** (NumberFormatException e) {

valeurCout.setBackground(Color.RED);

}

}

**ANNEXE : extraits de la documentation API**

## java.lang

## Interface Comparable<T>

|  |  |
| --- | --- |
| **Method Summary** | |
| int | [**compareTo**](https://docs.oracle.com/javase/6/docs/api/java/lang/Comparable.html#compareTo(T))([T](https://docs.oracle.com/javase/6/docs/api/java/lang/Comparable.html" \o "type parameter in Comparable) o)            Compares this object with the specified object for order. |

## java.util

## Class ArrayList<E>

## 

|  |  |  |
| --- | --- | --- |
| **Constructor Summary** | | |
| [**ArrayList**](https://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#ArrayList())()            Constructs an empty list with an initial capacity of ten. | |  |
| [**ArrayList**](https://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#ArrayList(java.util.Collection))([Collection](https://docs.oracle.com/javase/6/docs/api/java/util/Collection.html)<? extends [E](https://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html)> c)            Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator. | |  |
| **Method Summary** | | |
| boolean | [**add**](https://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#add(E))([E](https://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html) e)            Appends the specified element to the end of this list. | |
| boolean | [**remove**](https://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html#remove(java.lang.Object))**(**[**Object**](https://docs.oracle.com/javase/6/docs/api/java/lang/Object.html)**o)**  Removes the first occurrence of the specified element from this list, if it is present. | |
| [E](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html) | [**get**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#get%28int%29)**(int index)**           Returns the element (object) at the specified position in this list. | |
| int | [**size**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html#size%28%29)**()**           Returns the number of elements (objects) in this list. | |

## Class Collections

|  |  |
| --- | --- |
| **Method Summary** | |
| static   |  | | --- | | <T extends [Comparable](https://docs.oracle.com/javase/6/docs/api/java/lang/Comparable.html)<? super T>>  void | | [sort](https://docs.oracle.com/javase/6/docs/api/java/util/Collections.html#sort(java.util.List))([List](https://docs.oracle.com/javase/6/docs/api/java/util/List.html)<T> list)  Sorts the specified list into ascending order, according to the natural ordering of its elements. |

**javax.swing**

**Class JButton**

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**addActionListener**](http://java.sun.com/javase/6/docs/api/javax/swing/AbstractButton.html#addActionListener%28java.awt.event.ActionListener%29)([ActionListener](http://java.sun.com/javase/6/docs/api/java/awt/event/ActionListener.html" \o "interface in java.awt.event) l)            Adds an ActionListener to the button. |

**Class JTextField**

|  |  |
| --- | --- |
| **Method Summary** | |
| [String](http://java.sun.com/javase/6/docs/api/java/lang/String.html) | [**getText**](http://java.sun.com/javase/6/docs/api/javax/swing/AbstractButton.html#getText%28%29)()            Returns the text contained in this TextComponent. |
| void | [**setBackground**](https://docs.oracle.com/javase/6/docs/api/javax/swing/JComponent.html#setBackground(java.awt.Color))**(**[**Color**](https://docs.oracle.com/javase/6/docs/api/java/awt/Color.html)**bg)**  Sets the background color of this component. |

**Class JFrame**

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**dispose**](https://docs.oracle.com/javase/6/docs/api/java/awt/Window.html#dispose())**()**  Releases all of the native screen resources used by this Window, its subcomponents, and all of its owned children. |

**java.awt S**

**Class Container**

|  |  |
| --- | --- |
| **Method Summary** | |
| [Component](http://java.sun.com/javase/6/docs/api/java/awt/Component.html) | [**add**](http://java.sun.com/javase/6/docs/api/java/awt/Container.html#add%28java.awt.Component%29)([Component](http://java.sun.com/javase/6/docs/api/java/awt/Component.html) comp)            Appends the specified component to the end of this container. |

## java.awt.event

## Interface ActionListener

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**actionPerformed**](http://java.sun.com/javase/6/docs/api/java/awt/event/ActionListener.html#actionPerformed%28java.awt.event.ActionEvent%29)([ActionEvent](http://java.sun.com/javase/6/docs/api/java/awt/event/ActionEvent.html" \o "class in java.awt.event) e)            Invoked when an action occurs. |

## java.io

## Class FileOutputStream

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**FileOutputStream**](https://docs.oracle.com/javase/6/docs/api/java/io/FileOutputStream.html#FileOutputStream(java.lang.String))([String](https://docs.oracle.com/javase/6/docs/api/java/lang/String.html) name)            Creates an output file stream to write to the file with the specified name. |  |

## Class ObjectOutputStream

|  |  |  |
| --- | --- | --- |
| **Constructor Summary** | | |
|  | [**ObjectOutputStream**](https://docs.oracle.com/javase/6/docs/api/java/io/ObjectOutputStream.html#ObjectOutputStream(java.io.OutputStream))([OutputStream](https://docs.oracle.com/javase/6/docs/api/java/io/OutputStream.html" \o "class in java.io) out)            Creates an ObjectOutputStream that writes to the specified OutputStream. | |
| **Method Summary** | | |
| void | | [**close**](https://docs.oracle.com/javase/6/docs/api/java/io/ObjectInputStream.html#close())()            Closes the output stream. |
| void | | [**writeObject**](https://docs.oracle.com/javase/6/docs/api/java/io/ObjectOutputStream.html#writeObject(java.lang.Object))([Object](https://docs.oracle.com/javase/6/docs/api/java/lang/Object.html) obj)            Write the specified object to the ObjectOutputStream. |