Programmation avancée et application • 2020

Deuxième Session

Exercice I (2 points)

#Deuxième Session

Dans une classe UtilTime, définissez :

- *Une méthode qui prend en entrée trois nombres entiers représentant respectivement des heures, minutes et secondes, et qui retourne le nombre de secondes correspondant.*
- *Une méthode qui prend en entrée un nombre entier représentant des secondes, et qui retourne une chaîne de caractères correspondant à ce temps au format heures/minutes/secondes.*
- *On suppose qu'on a une liste de nombres entiers qui représentent des temps en secondes. Définissez une méthode qui permet d'obtenir le temps le plus long, sous forme de chaîne de caractères (au format heures/minutes/secondes).*

```java
package deuxiemeSessionExercice1;
import java.util.List;
public class UtilTime {
	/**
	 * Une méthode qui prend en entrée trois nombres entiers représentant respectivementdes heures, minutes et secondes, et qui retourne le nombre de secondes correspondant.
	 * @param h heures
	 * @param m minutes
	 * @param s secondes
	 * @return Le nombre de secondes.
	 * @throws IllegalArgumentException Si l'un des paramètres est négatif
	 */
	public static int getSecondes(int h, int m, int s) throws IllegalArgumentException{
		if( (h < 0) || (m < 0) || (s < 0) ) // Si l'un des paramètres est négatif
			throw new IllegalArgumentException("Au moins un des paramètres est négatif.");
		return ( (h * 3600) + (m * 60) + s);
	}

	/**
	 * Une méthode qui prend en entrée un nombre entier représentant des secondes, et qui retourne une chaîne de caractères correspondant à ce temps au format heures/minutes/secondes.
	 * @param tempsEnSecondes un nombre entier représentant des secondes
	 * @return une chaîne de caractères correspondant à ce temps au format heures/minutes/secondes.
	 * @throws IllegalArgumentException Si le paramètre est négatif
	 */
	public static String getHeuresMinutesSecondes(int tempsEnSecondes) throws IllegalArgumentException{
		if( tempsEnSecondes < 0 ) // Si le paramètre est négatif
			throw new IllegalArgumentException("Le paramètre est négatif.");

		int heures = tempsEnSecondes / 3600;
		int minutes = ( tempsEnSecondes - (heures * 3600) ) / 60;
		int secondes = tempsEnSecondes - (heures * 3600) - (minutes * 60);


		return ( heures + "/" + minutes + "/" + secondes );
	}

	/**
	 * Une méthode qui prend une liste de nombres entiers qui représentent des temps en secondes, et qui retourne une chaîne de caractères correspondant au temps le plus long au format heures/minutes/secondes.
	 * @param listeTemps Une liste de nombres entiers qui représentent des temps en secondes.
	 * @return Le temps le plus long, sous forme de chaîne de caractères (au format heures/minutes/secondes).
	 * @throws IllegalArgumentException Si la liste est vide.
	 */
	public static String getMaxTemps(List<Integer> listeTemps) throws IllegalArgumentException{
		if( listeTemps.size() == 0 ) // Si la liste est vide
			throw new IllegalArgumentException("Le liste est vide.");

		listeTemps.sort(null); // A null value indicates that the elements' natural ordering should be used (JavaDoc)
		return ( getHeuresMinutesSecondes(listeTemps.get(listeTemps.size()-1)) );
	}
}
```

```java
package deuxiemeSessionExercice1Tests;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

import deuxiemeSessionExercice1.UtilTime;
import java.util.List;
import java.util.ArrayList;
class UtilTimeTest {

	@Test
	void getSecondesTest() {
		assertEquals(12630, UtilTime.getSecondes(3, 30, 30));
	}

	@Test
	void getHeuresMinutesSecondesTest() {
		assertEquals("3/30/30", UtilTime.getHeuresMinutesSecondes(12630));
	}

	@Test
	void getMaxTempsTest() {
		List<Integer> liste = new ArrayList<Integer>();
		liste.add(12630);
		liste.add(100);
		assertEquals("3/30/30", UtilTime.getMaxTemps(liste));
	}
}
```

Exercice 3 (6 points)

#Deuxième Session

On peut représenter une formule logique,
(comme celles qui sont évaluées dans des expressions if ou while)
sous forme d'un arbre, dont les noeuds internes sont les connecteurs logiques
OR (||), AND (&&) et NOT (!)
et dont les feuilles sont les valeurs booléennes (true ou false).
On souhaite pouvoir représenter ces expressions et évaluer leur valeur.

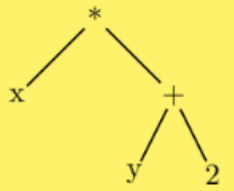
- Implémentez les classes nécessaires à cette représentation
- Donnez des tests unitaires pour la méthode qui permet d'évaluer un noeud AND.

Il est possible de représenter une expression mathématique sous forme d'un arbre binaire.

— Les noeuds internes de l'arbre représentent les opérateurs (+ - * /) et les fils représentent les termes de gauche et de droite de l'opération.

— Les feuilles représentent des variables ou des valeurs.

Exemple : l'expression $x * (y + 2)$ est représentée par l'arbre



TD9 Algorithmique et structures de données
L2 Informatique / Gael Mahe

Exercice VI (POO et opérations mathématiques)
On peut représenter une opération mathématique sous forme d'un arbre dont les noeuds sont des opérateurs, et les feuilles sont des valeurs numériques. On souhaite représenter de telles expressions et pouvoir évaluer leur valeur.

- La classe (abstraite) Operateur représente aussi bien un noeud qu'une feuille de l'arbre. Un Operateur est caractérisé par son arité, qui est 0 dans le cas des feuilles, et un entier > 0 pour les noeuds internes.
- Les opérateurs arithmétiques habituels (Somme, Soustraction, Multiplication, Division), d'arité 2, sont des classes filles d'Operateur.
- On peut également définir des classes AdditionNAire et MultiplicationNAire pour les versions générales de l'addition et la multiplication (l'arité est donc variable).
- Une Valeur a une arité nulle, et représente un nombre réel.

La classe Operateur contient une méthode abstraite **public abstract double** evaluer() ; qui retourne la valeur de cet opérateur pour ses opérandes.
Implémentez ces classes.

Programmation avancée et application
TD3 : Classes et héritage

```java
package deuxiemeSessionExercice3;

/**
 * La classe (abstraite) Operateur
 * représente aussi bien un noeud qu'une feuille de l'arbre.
 * Les noeuds internes sont les connecteurs logiques et les feuilles sont les valeurs booléennes.
 *
 * Les opérateurs logiques habituels (OR, AND, NOT), sont des classes filles d'Operateur.
 */
public abstract class Operateur {
	/**
	 * Un Operateur est caractérisé par son arité,
	 * qui est 0 dans le cas des feuilles,
	 * et un entier > 0 pourles noeuds internes.
	 *
	 * L'arité d'une fonction, ou opération,
	 * est le nombre d'arguments ou d'opérandes qu'elle requiert.
	 * (Source : Wikipedia - CC BY-SA 3.0)
	 */
	private int arite;

	public Operateur(int arite) {
		this.arite = arite;
	}

	/**
	 * La classe Operateur contient une méthode abstraite
	 * public abstract boolean evaluer();
	 * qui retourne la valeur de cet opérateur pour ses opérandes.
	 * @return la valeur de cet opérateur pour ses opérandes.
	 */
	public abstract boolean evaluer();
}
```

```java
package deuxiemeSessionExercice3;

/**
 * Les opérateurs logiques habituels (OR, AND, NOT), sont des classes filles d'Operateur.
 * And : une classe qui represente l'operateur &&  d'arité 2.
 */
public class And extends Operateur{
	private Operateur operandeNumero1;
	private Operateur operandeNumero2;

	public And(Operateur operandeNumero1, Operateur operandeNumero2) {
		super(2); // arité 2
		this.operandeNumero1 = operandeNumero1;
		this.operandeNumero2 = operandeNumero2;
	}

	@Override
	public boolean evaluer() {
		return( (operandeNumero1.evaluer()) && (operandeNumero2.evaluer()) );
	}
}
```

```java
package deuxiemeSessionExercice3;

/**
 * Les opérateurs logiques habituels (OR, AND, NOT), sont des classes filles d'Operateur.
 * Or : une classe qui represente l'operateur ||  d'arité 2.
 */
public class Or extends Operateur{
	private Operateur operandeNumero1;
	private Operateur operandeNumero2;

	public Or(Operateur operandeNumero1, Operateur operandeNumero2) {
		super(2); // arité 2
		this.operandeNumero1 = operandeNumero1;
		this.operandeNumero2 = operandeNumero2;
	}

	@Override
	public boolean evaluer() {
		return( (operandeNumero1.evaluer()) || (operandeNumero2.evaluer()) );
	}
}
```

```java
package deuxiemeSessionExercice3;

/**
 * Les opérateurs logiques habituels (OR, AND, NOT), sont des classes filles d'Operateur.
 * Not : une classe qui represente l'operateur !  d'arité 1.
 */
public class Not extends Operateur{
	private Operateur operande;

	public Not(Operateur operande) {
		super(1); // arité 1
		this.operande = operande;
	}

	@Override
	public boolean evaluer() {
		return( !(operande.evaluer()) );
	}
}
```

```java
package deuxiemeSessionExercice3;

/**
 * Une Valeur a une arité nulle,
 * et représente une valeur booléenne (true ou false).
 */
public class Valeur extends Operateur{
	private boolean valeur;

	public Valeur(boolean valeur) {
		super(0); // arité 0
		this.valeur = valeur;
	}

	@Override
	public boolean evaluer() {
		return valeur;
	}
}
```

```java
package deuxiemeSessionExercice3Tests;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

import deuxiemeSessionExercice3.And;
import deuxiemeSessionExercice3.Valeur;
class AndTests {

	@Test
	void evaluerTest_01() {
		And and = new And(new Valeur(1 > 2), new Valeur(5 < 10));
		assertEquals(false, and.evaluer());
	}

	@Test
	void evaluerTest_02() {
		And and = new And(new Valeur(1 < 2), new Valeur(5 < 10));
		assertEquals(true, and.evaluer());
	}

	@Test
	void evaluerTest_03() {
		And and = new And(new Valeur(1 > 2), new Valeur(5 > 10));
		assertEquals(false, and.evaluer());
	}
}
```

Exercice 2 (6 points)

#Deuxième Session

Une liste chaînée

est une structure de données représentant une collection ordonnée et détaillé arbitraire d'éléments du même type.

- Par "ordonnée", on veut dire que chaque élément a une position déterminée par un indice précis, contrairement à un ensemble (mais cela ne veut pas forcément dire que la liste est triée).
- On représente un de ces éléments sous forme d'un "maillon", c'est-à-dire une cellule qui contient une valeur et une référence vers le maillon suivant (ou **null** si c'est le dernier maillon de la chaîne).
- On peut donc manipuler une liste chaînée juste en connaissant son premier maillon.

Pour les questions suivantes, vous êtes invités à ajouter les éventuels attributs ou méthodes qui vous semblent nécessaires.

1. Définissez **la classe Maillon** qui contient une donnée stockée sous forme de chaîne de caractères, et une référence vers le Maillon suivant.
2. Définissez **la classe ListeChainee** qui permet de représenter une liste de chaines de caractères.

   Cette classe doit avoir au moins deux constructeurs :
   - → Un constructeur qui crée une liste vide ;
   - → Un constructeur qui prend en entrée un tableau de String, et construit la liste chainée correspondante.
3. Définissez **une méthode qui permet d'ajouter** une chaîne de caractères à une position donnée. Si cette position n'est pas possible (valeur négative, ou plus grande que le nombre d'éléments dans la liste), la méthode doit lever une exception d'un type particulier que vous aurez créé dans ce but.
4. Créez une méthode qui ajoute un élément à la **première position de la liste**, et une méthode qui ajoute un élément à la **dernière position de la liste**.
5. Créez **une méthode qui enregistre dans un fichier** (dont le chemin est passé en entrée) l'ensemble des éléments de la liste, avec un élément par ligne

```java
package deuxiemeSessionExercice2;

public class Maillon {
	/**
	 * une donnée stockée sous forme de chaîne de caractères.
	 */
	private String donneeStockee;

	/**
	 * Une référence vers le Maillon suivant.
	 */
	private Maillon maillonSuivant;

	public Maillon(String donneeStockee, Maillon maillonSuivant) {
		this.donneeStockee = donneeStockee;
		this.maillonSuivant = maillonSuivant;
	}

	public Maillon(String donneeStockee) {
		this(donneeStockee, null);
	}

	public Maillon getMaillonSuivant() {
		return maillonSuivant;
	}

	public String getDonneeStockee() {
		return donneeStockee;
	}

	public void setMaillonSuivant(Maillon maillonSuivant) {
		this.maillonSuivant = maillonSuivant;
	}
}
```

```java
package deuxiemeSessionExercice2;

public class ListeChaineeException extends Exception {
	private static final long serialVersionUID = 1L;

	public ListeChaineeException(String message) {
		super(message);
	}
}
```

```java
package deuxiemeSessionExercice2;

public class ListeChaineeTest {

	public static void main(String[] args) {
		String[] tab = {"2", "3", "4", "5"};
		try {
			ListeChainee liste = new ListeChainee(tab);
			liste.ajoutElementPremierePosition("0");
			liste.ajouterChaineDeCaracteresDansPosition("1", 1);
			liste.ajoutElementDernierePosition("6");
			liste.enregistreDansFichier("C:\\Users\\admin\\Desktop\\out.txt");
		} catch (ListeChaineeException e) {
			System.err.println( e.getMessage() );
		}

	}

}
```

```java
package deuxiemeSessionExercice2;

import java.io.PrintWriter;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class ListeChainee {
	/**
	 * On peut manipuler une liste chaînée juste en connaissant son premier maillon.
	 */
	private Maillon premierMaillon;

	/**
	 * Un constructeur qui crée une liste vide.
	 */
	public ListeChainee() {
		premierMaillon = null;
	}

	/**
	 * Un constructeur qui prend en entrée un tableau de String,
	 * et construit la liste chainée correspondante.
	 * @throws ListeChaineeException
	 */
	public ListeChainee(String[] tableau) throws ListeChaineeException {
		ajoutElementPremierePosition(tableau[0]);
		for(int i = 1 ; i < tableau.length ; i++)
			ajouterChaineDeCaracteresDansPosition(tableau[i], i);
	}

	/**
	 * Une méthode qui permet d'ajouter une chaîne de caractères
	 * à une position donnée.

	 * @param chaine une chaîne de caractères
	 * @param position	une position donnée.
	 * @throws ListeChaineeException Si la position n'est pas possible (valeur négative, ou plus grande que le nombre
d'éléments dans la liste).
	 */
	public void ajouterChaineDeCaracteresDansPosition(String chaine, int position) throws ListeChaineeException{
		if(position < 0) {
			throw new ListeChaineeException("La position n'est pas possible (valeur négative)");
		}

		Maillon positionActuelle = premierMaillon;
		Maillon positionPrecedente = null;
		boolean found = false; // La position demandée a été trouvée ?
		int i;
		for(i = 0 ; positionActuelle != null && found == false ; i++) {

			if( i == position ) {
				if( positionPrecedente != null )  // Si la position demandée n'est pas 0 (0 - la première
position de la liste).
					positionPrecedente.setMaillonSuivant( new Maillon(chaine, positionActuelle) );
				else // Si la position demandée est pas 0 (0 - la première position de la liste).
					premierMaillon = new Maillon(chaine, positionActuelle);
				found = true; // La position demandée a été trouvée (la boucle peut être arrêtée)
			}
			positionPrecedente = positionActuelle;
			positionActuelle = positionActuelle.getMaillonSuivant();
		}

		if( found == false ) { // Si nous avons déjà parcouru toute la liste sans trouver l'emplacement souhaité
			if(i == 0) // Si la liste est vide.
				premierMaillon = new Maillon(chaine, null);
			else if(i == position ) // Si la position demandée se trouve immédiatement après tous les éléments de la
liste
				positionPrecedente.setMaillonSuivant( new Maillon(chaine, null) ); // On ajoute l'élément à la
fin de la liste
			else
				throw new ListeChaineeException("La position n'est pas possible (plus grande que le nombre
d'éléments dans la liste)");
		}
	}
```

```java
	/**
	 * une méthode qui ajoute un élément à la première position de la liste
	 * @throws ListeChaineeException
	 */
	public void ajoutElementPremierePosition(String chaine) throws ListeChaineeException {
		ajouterChaineDeCaracteresDansPosition(chaine, 0);
	}

	/**
	 * une méthode qui ajoute un élément à la dernière position de la liste
	 * @throws ListeChaineeException
	 */
	public void ajoutElementDernierePosition(String chaine) throws ListeChaineeException {
		int i;
		Maillon positionActuelle = premierMaillon;
		for(i = 0 ; positionActuelle != null ; i++)
			positionActuelle = positionActuelle.getMaillonSuivant();

		ajouterChaineDeCaracteresDansPosition(chaine, i);
	}

	public void enregistreDansFichier(String cheminFichier) {
		try(PrintWriter printW = new PrintWriter(new BufferedWriter(new FileWriter(cheminFichier)))) {
			Maillon positionActuelle = premierMaillon;
			while( positionActuelle != null ) {
				printW.println( positionActuelle.getDonneeStockee() );
				positionActuelle = positionActuelle.getMaillonSuivant();
			} // for
		} catch(IOException e) {
			System.err.println( e.getMessage() );
			System.exit(1);
		}
	}
}
```