TP8

Ce TP se compose de 2 parties: - La première sera consacrée à la création de classes et d'interface. - La seconde porte sur les threads

Première partie

Modèle objet

Dans un premier temps, représentez les articles que vends un magasin de matériaux.

Le magasin vend: - des poutres qui sont de différents bois et de différentes dimensions - des tournevis cruciformes ou plat

Chacun des articles possède un prix unitaire.

Les articles peuvent être vendus unitairement ou par lots. Dans ce cas, il est possible d'appliquer une réduction sur le prix du lot.

Chacune des classes doit être capable d'afficher ses informations.

Vous tirerez profit de la mécanique d'héritage afin de vous assurer que les classes implémentent toutes les méthodes requises.

Facturation

Afin de contractualiser ses ventes, le magasin emet des factures. Chacune de ces facture comporte un numéro, un nom de client, la liste des articles, et un prix total.

Elle doit aussi pouvoir être imprimée de façon à comporter les éléments suivants:

Quantité	Nom	Prix unitaire	Reduction	Prix total
1	Poutre 20*10	10	0	10
20	Tournevis plat	7	10	126

Total: 136

Utilisez une interface pour traiter de manière similaire les articles et les lots d'articles.

Vous testerez votre programme avec le code suivant:

```
public class Test {

    public void main(String args[]) {
```

```
        ////////////////////////////
        // Création des instances //
        ////////////////////////////

        // Création d'une poutre de 20*10 à 10 euros pièce
        Article poutre = new Poutre("Pin", 20, 10, 10);

        // Création d'un tournevis plat à 7 euros pièce
        Article tournevisPlat = new Tournevis(TournevisType.PLAT, 7);

        // Création d'un lot dé 20 tournevis avec 10% de remise
        Lot lotTournevis = new Lot(tournevisPlat, 20, 10);

        System.out.println(poutre);
        System.out.println(tournevisPlat);
        System.out.println(lotTournevis);

        /////////////////
        // Facturation //
        /////////////////

        Facture facture = new Facture(1, "Mario");

        facture.addLine(poutre);
        facture.addLine(lotTournevis);

        System.out.println("Total : " + facture.getTotal());

        System.out.println(facture);
    }
}
```

Threads

Le but de ce TP est de parcourir un labyrinthe alétoirement et de trouver une sortie.

Pour cela, le code qui génère un labyrinthe vous est fourni. Il est composé de 2 classes: - une classe **Labyrinthe**: elle représente un labyrinthe qui se crée lorsque vous instanciez la classe. Elle expose une méthode: * *getEntry* permettant de récupérer une entrée du Labyrinthe. L'instance retourné est de type Crossing

- une classe **Crossing**: elle représente une intersection du labyrinthe. Chaque crossing possède des coordonnées qui correspondent au chemin parcouru pour y arriver. Exemple: un crossing dont le chemin est: 2-1-3 signifie que le joueur a pris l'entrée 2, puis a choisi le chemin 1, puis le 3. Elle expose quatre méthodes:
  - *isExit*: renvoie True si le Crossing et une sortie, False sinon
  - *getNbCrossings*: renvoie un entier représentant le nombre de crossing disponibles
  - *getCrossing(int i)*: renvoie l'instance Crossing n°i
  - *getPath*: retourne une liste d'Integer représentant le chemin du Crossing

Dans un premier temps, vous implémenterez dans une fonction 'main' l'algorithme de parcours

d'un aventurier pour trouver la sortie. A chaque itération vous devrez choisir un crossing au hasard, et afficher le chemin emprunté.

PS: Tout les chemins ont une fin !

Classe Labyrinthe:

```
package org.supinternet.lab;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Labyrinthe {

    // La liste d'entrées
    private List<Crossing> entries;
    // Le nombre d'entrées
    private final static int NB_ENTRIES = 4;

    //Constructeur
    public Labyrinthe() {

        entries = new ArrayList<>();
        // Création des entrées
        for ( int i = 1 ; i <= NB_ENTRIES ; i++) {
            // Initialisation du tableau de chemin
            List<Integer> path = new ArrayList<Integer>();
            // La première valeure
            path.add(i);
            entries.add(new Crossing(path));
        }
    }

    // Retourne une entrée au hasard
    public Crossing getEntry() {
        Collections.shuffle(entries);
        return entries.get(0);
    }
}
```

Classe Crossing

```
package org.supinternet.lab;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class Crossing {

    // La liste de crossings suivants
    private List<Crossing> crossings;
```

```
    // Est-ce une sortie ?
    private boolean exit = false;
    // Le chemin du crossing
    private List<Integer> path;

    // Le nombre de crossings suivants maximum
    private static final int NB_MAX_NEXT = 4;
    // La longueur maximum d'un chemin dans le labyrinthe
    private static final int MAX_LENGTH = 10;

    public Crossing(List<Integer> path) {

        this.path = path;

        // Choix du nombre de crossings suivants
        int nbCrossings = (int) Math.round(Math.random() * NB_MAX_NEXT);

        // Si il n'y a pas de crossings suivants, c'est une sortie
        if (nbCrossings == 0 || path.size() > MAX_LENGTH) {
            exit = true;
        //Sinon création des crossings
        } else {
            crossings = new ArrayList<Crossing>();

            // Création de nb_crossings
            for ( int i = 1 ; i <= nbCrossings ; i++ ) {
                // Copie du path pour éviter une modification externe par référence
                List<Integer> next_path = new ArrayList<Integer>(path);
                next_path.add(i);
                // Instanciation du Crossing
                crossings.add(new Crossing(next_path));
            }
        }
    }

    // Retourne True si le crossing est une sortie
    public boolean isExit() {
        return exit;
    }

    // Retourne le nombre de crosing suivants
    public int getNbCrossings() {
        return crossings.size();
    }

    // Retourne un crossing en particulier
    public Crossing getCrossing(int i) {
        return crossings.get(i);
    }

    // Retourne une description du chemin
    public String getPath() {
        return String.join("-", path.stream().map(Object::toString)
                    .collect(Collectors.toList()));
    }

    public String toString() {
```

```
        return getPath();
    }
}
```

Base Classe Main

```
package org.supinternet.main;

import org.supinternet.lab.Crossing;
import org.supinternet.lab.Labyrinthe;

public class Main {

public static void main(String[] args) {
    Labyrinthe labyrinthe = new Labyrinthe();

    // Insert your code here
    //
    // Ex: Crossing c = lab.getEntry();
    }
}
```

Dans un second temps, vous créerez 5 personnages qui exploreront le labyrinthe simultanément. Vous pouvez utiliser au choix Thread ou Runnable. Chaque joueur patientera 1/2 seconde entre chaque décision.

Enfin, vous ajouterez une méthode à la classe Crossing permettant de signaler la présence d'un joueur. Si la case sur laquelle arrive un joueur est occupée, il devra alors patienter une seconde et retenter, tant que la case est occupée. Attention aux accès concurrents !

Maintenant, tentez avec 50 aventuriers !

Notation

Première partie

- 1 point par classe
- 2 points pour une bonne utilisation de l'héritage
- 1 point pour un affichage correct des informations de chacune des classes
- 2 points pour la création d'une interface significative

Seconde partie

- 2 points pour l'algorithme de base
- 3 points pour la création d'un thread
- 1 point pour le lancement de 5 aventuriers en parallèle
- 1 point pour l'attente entre chaque décision
- 2 point pour l'ajout de la fonction d'occupation d'un crossing
- 1 pour la double attente en cas d'occupation