EPHEC

Nom : .....

Prénom : .....

Classe : .....

Langage avancé de programmation

Interrogation de janvier 2013

Professeur : V. Van den Schrieck

08/01/2013

Durée : 2 heures

Matériel autorisé : De quoi écrire uniquement

Nombre de pages : 15

Classe : 2TL1 - 2TL2

Répondre aux questions directement sur ce document.

Pondération

	Points pour la partie	Points de l'étudiant
UML	5	
POO	7	
Structures de données	4	
Tests et documentation	6	
Prog. concurrence	8	
Total	**30**	

Contexte

Le programme sur lequel l'examen se base représente un bar à bières. Le bar possède un stock de différentes bières, et plusieurs serveurs s'occupent de gérer les commandes en parallèle et d'encaisser le prix des bières.
Vous trouverez en fin du feuillet les classes qui composent ce programme. Commencez par les lire attentivement, avant de répondre aux questions.Les classes BeerNotFoundException et InsufficientException ne sont pas fournies, mais vous pouvez considérer qu'elles sont disponibles.

1 UML (/5)

Question 1 [/5] : Dessinez le diagramme UML de l'application de gestion de bar. Dans ce diagramme doivent figurer aussi bien les classes dont le code vous est fourni à la fin de ce document, que les classes de l'API Java qui y sont explicitement mentionnées.

Programmation orientée-objet (/7)]

Question 2 [/3] : Utilisation d'objets

Complétez la méthode main de la classe TestBar, afin de remplir le bar avec un stock initial de 10 bouteilles de Chimay à 2 euros pièce et à 9 degrés d'alcool, ainsi qu'un stock de 5 bouteilles de Westmalle à 10 degrés et à 3 euros pièce.

```
public class TestBar {
    public static void main(String [] args) throws BeerNotFoundException,
                                                    InterruptedException{
        Bar monBar = new Bar();









        /* Suite de la methode : Cfr fichier annexe */
    }


}
```

Question 3 [/4] : Héritage

Un nouveau client s'est présenté à vous, et souhaite utiliser l'application dans un bar-restaurant. Dans ce type d'établissement, le prix de la bière est augmenté par une taxe de 5%. Comment pouvez-vous compléter le programme de manière simple pour traiter cette demande?
Expliquez d'abord la solution, en complétant au passage le diagramme UML (uniquement les classes concernées). Ensuite, implémentez-la.

Explication et diagramme UML :

Implémentation :

Structures de données (/4)

Question 4 [/4] : Listes chaînées

La classe BeerLinkedList définit une structure de données permettant de stocker les bières par ordre de degré d'alcool. Elle contient deux méthodes de recherche : la première recherche la bière la plus proche d'un degré d'alcool donné en paramètre, la seconde recherche une bière sur base de son nom. Il vous est demandé d'écrire le code de la seconde, en vous basant sur sa spécification.

```
/**
 * Cette methode permet de trouver une biere sur base de son nom.
 * @param name : Une chaine de caractere non vide representant le nom de la biere
 * @return La biere portant le nom name
 * @throws BeerNotFoundException
 */
public Beer find (String name) throws BeerNotFoundException {









}
```

Documentation et tests (/6)

Question 5 [/3] : Documentation

Documentez la méthode find(int degree) de la classe Beer.

```
/**
  *
  *
  *
  *
  *
  *
  *
  *
  *
  *
  *
  */
public Beer find (int degree){
        if(first == null || first.beer.getDegree() > degree)
            return null;
        Node current = first;
        while(current.next != null && current.next.beer.getDegree() <= degree){
            current=current.next;
        }
        return current.beer;
    }
```

Question 6 [/3] : Tests

Expliquez en quoi consistent les tests Black-Box, et quelle est la différence avec les tests White-Box. Proposez ensuite un ensemble de tests Black-Box pour la méthode compareTo de la classe Beer, sous forme d'une liste de paires valeurs d'entrée - résultat attendu (sous forme de description textuelle, pas de code java).

Programmation concurrente (/8)

Question 7 [/3] : Gestion des threads

Après la création du bar et du stock (que vous avez du implémenter), la méthode `main` de la classe TestBar crée quatre objets de type Commande et les utilise, avant d'imprimer la recette encaissée.

1. Combien de threads fonctionneront en parallèle?
2. A quoi servent les appels à la méthode `join()`?
3. Qu'affichera le programme à la console à la fin de l'exécution?

Question 8 [/5] : Partage de ressources

La méthode run() de la classe Commande exécute la méthode serve() de la classe Bar. Cette méthode fait deux choses : D'une part, elle modifie le stock de la bière commandée, et d'autre part, elle incrémente la recette du bar. Dans le cadre de la programmation concurrente, expliquez ce qu'il faut modifier pour garantir un fonctionnement correct de la méthode serve(), et pourquoi :

- dans la classe Bar (Attention à ne pas perdre les bénéfices de la programmation concurrente)
- dans la classe Beer

Classes

Classe Beer

```java
package gestionBar;
/**
 * Classe Beer
 */
 public class Beer{
     private final String name;
     private final int price;
     private final int degree;
     private int stock;

     /**
      * Constructeur avec arguments
      * @param name : Represente le nom de la biere
      * @param price : Un entier strictement superieur a 0
      * @param degree : Un entier strictement superieur a 0
      */
     public Beer(String name, int price, int degree, int stock){
         this.name=name;
         this.price=price;
         this.degree=degree;
         this.stock=stock;
     }

    /**
     * @return Le nom de la biere
     */
    public String getName() {
        return name;
    }

    /**
     * @return Le prix de la biere, >0.
     */
    public int getPrice() {
        return price;
    }

    /**
     * @return Le degre de la biere, >0.
     */
    public int getDegree() {
        return degree;
    }

    /**
     * Cette methode permet de comparer deux bieres
     * sur base de leur degre d'alcool
     * @param b : Un objet Beer non null
     * @return  une valeur negative si la biere courante a un degre d'alcool < b
     *          0 si les degres d'alcool des deux bieres sont identiques
     *          une valeur positive si la biere courante a un degre d'alcool > b
     */
    public int compareTo(Beer b) {
        return this.degree-b.degree;
    }
    /**
     * @return Le nombre de bouteilles en stock pour cette biere
     */
    public int getStock() {
        return stock;
    }
```

```
    /************************
     *  Classe Beer, suite  *
     ************************/

    /**
     * Cette methode retire une bouteille du le stock si celui-ci n'est pas vide.
     * @throws InsufficientStockException : Si le stock est vide.
     */
    public void getBottle() throws InsufficientStockException{
        if(stock==0){
            throw new InsufficientStockException("Plus assez de stock");
        }
        stock--;
    }
    /**
     * Ajoute numBottles bouteilles de la biere courante dans le stock.
     * @param numBottles : entier positif.
     */
    public void addStock(int numBottles) {
        if(numBottles>0)
            this.stock+=numBottles;
    }
}
```

Classe BeerLinkedList

```java
package gestionBar;
// BeerLinkedList.java
/**
 * Cette classe definit une liste chanee permettant de stocker des bieres de maniere
 * ordonnee, suivant leur degre d'alcool (ordre croissant)
 * @author S. Combefis, V. Van den Schrieck
 *
 */
public class BeerLinkedList
{
    private Node first;
    /**
     * Constructeur sans argument
     */
    public BeerLinkedList(){
        first = null;
    }
    /**
     * Cette methode ajoute une biere a la liste, en respectant l'ordre des degres.
     * @param b : Une biere != null.
     */
    public void add(Beer b){
        Node newNode = new Node(b);
        if (first == null){
            first = newNode;
        }
        //Si la nouvelle biere a le plus bas degre d'alcool : mise a jour de first.
        else if (first.beer.compareTo(newNode.beer) > 0){
            newNode.next = first;
            first = newNode;
        }
        else{
            Node current = first;
            while (current.next != null && current.next.beer.compareTo(newNode.beer) < 0){
                current = current.next;
            }
            newNode.next = current.next;
            current.next = newNode;
        }
    }
    /**
     *  A DOCUMENTER
     *
     */
    public Beer find (int degree){
        if(first == null || first.beer.getDegree() > degree)
            return null;
        Node current = first;
        while(current.next != null && current.next.beer.getDegree() <= degree){
            current=current.next;
        }
        return current.beer;
    }
    /**
     * Cette methode permet de trouver une biere sur base de son nom.
     * @param name : Une chane de caractere non vide representant le nom de la biere
     * @return La biere portant le nom
     * @throws BeerNotFoundException
     */
    public Beer find (String name) throws BeerNotFoundException{
       /*
        *  A completer
        */
    }
```

```
    /*********************************
     *  Classe BeerLinkedList, suite  *
     *********************************/
    /**
     * Noeud de la liste simplement chanee
     * @author S. Combefis
     *
     */
    private class Node{
        private Node next;
        private final Beer beer;

        public Node (Beer beer){
            next = null;
            this.beer = beer;
        }
    }

}
```

Classe Bar

```
package gestionBar;

public class Bar {
    private float revenue;
    private BeerLinkedList list;

    /*
     * Constructeur sans parametre.
     */
    public Bar(){
        list = new BeerLinkedList();
    }
    /**
     * Ajoute une nouvelle biere au stock.
     * @param name : Le nom de la nouvelle biere.  Doit etre != null.
     * @param degree : Le degre de la biere name.  Doit etre positif.
     * @param price : Le prix de la biere name.  Doit etre positif.
     */
    public void addBeer(String name, int degree, int price){
        list.add(new Beer(name, price, degree, 0));
    }
    /**
     * Ajoute numBottles bouteilles de la biere name au stock
     * @param name : Nom de la biere contenue dans les bouteilles. Doit etre != null;
     * @param numBottles : Nombre de bouteilles    ajouter.  Doit etre >0.
     * @throws BeerNotFoundException : Si la biere n'existe pas encore dans le stock.
     */
    public void addStock(String name, int numBottles) throws BeerNotFoundException {
        Beer beer = list.find(name);
        beer.addStock(numBottles);
    }
    /**
     * Cette methode fournit le prix    payer dans ce bar pour la biere de nom name.
     * @param name : Nom de la biere    trouver.  Doit etre != null
     * @return Le prix d'une bouteille de biere de nom name.
     * @throws BeerNotFoundException si la biere n'existe pas dans le stock.
     */
    public float getPrice(String name) throws BeerNotFoundException{
        return list.find(name).getPrice();
    }
    /**
     * Cette methode permet de servir une biere    un client, et
     * d'encaisser le prix correspondant.
     * @param name : Le nom d'une biere.
     * @throws BeerNotFoundException : Si la biere n'est pas servie dans le bar
     * @throws InsufficientStockException : Si la biere n'est plus en stock
     */
    public void serve(String name) throws BeerNotFoundException,
                                          InsufficientStockException{
        Beer beer = list.find(name);
        beer.getBottle();
        revenue+=beer.getPrice();
    }
    /**
     * Renvoie la recette actuelle du bar.
     */
    public float getRevenue(){
        return this.revenue;
    }
}
```

Classe Commande

```
package gestionBar;


/**
 * Classe simulant un serveur executant des commandes au bar.
 * @author V. Van den Schrieck
 */
public class Commande extends Thread {
    private static int total=0;
    private int numCommand=0;
    private String beerName;
    private int numBottles;
    private Bar bar;
    /**
     * Constructeur avec nom du serveur en argument
     */
    public Commande(Bar b, String name, int num){
        super();
        this.beerName=name;
        this.numBottles=num;
        numCommand=total+1;
        total=total+1;
        bar = b;
    }
    /*
     *  Code    ex cuter par le thread
     */
    public void run() {
        int i=0;
        try{
            for(i=0; i < numBottles; i++){
                bar.serve(beerName);
            }
        }
        catch(InsufficientStockException e){
            System.out.println("Erreur : Stock insuffisant.  Seules "+(i) +
                               " bieres ont pu  tre  servies pour la commande "+numCommand);
        }
        catch(BeerNotFoundException e){
            System.out.println("Erreur : Biere inconnue");
        }
    }

}
```

Classe TestBar

```
package gestionBar;

public class TestBar {
    public static void main(String [] args) throws BeerNotFoundException,
                                                    InterruptedException{
        Bar monBar = new Bar();
        /*
         *  A COMPLETER : CREATION DU STOCK DE BIERES DU BAR
         *      - 10 Chimay a 9 degres et 2 euros la bouteille
         *      - 5 Westmalle    10 degres et 3 euros la bouteille
         *
         */

        Commande c1 = new Commande(monBar, "Chimay", 5);
        Commande c4 = new Commande(monBar, "Westmalle", 3);
        Commande c2 = new Commande(monBar, "Chimay", 5);
        Commande c3 = new Commande(monBar, "Chimay", 5);

        c1.start();
        c2.start();
        c3.start();
        c4.start();
        c1.join();
        c2.join();
        c3.join();
        c4.join();
        System.out.println("Recette du bar : "+monBar.getRevenue());
    }


}
```