TP 6 : Les exceptions en Java

christina.boura@uvsq.fr, stephane.lopes@uvsq.fr

25 mars 2016

Exercice 1 *Calculer des racines d'une équation quadratique*
Le but de cet exercice est d'écrire un programme qui calcule et renvoie la plus grande des deux racines de l'équation $ax^2 + bx + c = 0$, où a, b et c sont des nombres réels passés en paramètre au programme par l'utilisateur.

1. Écrire une fonction

   ```
   public static double calculRacine (double a, double b, double c)
   ```

   qui prend en entrée les trois coefficients réels de l'équation et qui renvoie la plus grande de deux racines dans le cas où l'équation a une solution. Si l'équation n'a pas de solutions il faut alors créer et lancer une exception qui sera instance de la classe `IllegalArgumentException`, avec un message d'erreur descriptif. Il faut également lancer une exception du même type si le coefficient a est zéro.
2. L'utilisateur doit passer les trois constantes a, b et c comme paramètres à la fonction `main`. Ces trois valeurs seront sauvegardés dans la tableau `args[]` de la fonction `main`. Pour lire et utiliser ces valeurs vous pouvez utiliser la fonction `Double.parseDouble(String s)`.
3. Inclure la partie du code qui pourrait éventuellement lever une exception dans un bloc `try`.
4. Refléchir aux différentes types d'exceptions qui peuvent arriver. Créer un bloc `catch` pour attraper chaque exception qui pourrait être levée dans le bloc `try`. Faire en sorte d'avoir au moins deux blocs catch différents. Afficher dans chaque bloc `catch` un message descriptif de l'erreur ainsi que de l'information sur l'exception.
5. Tester votre code avec des entrées différentes. Vérifier ce qui se passe dans les cas suivants :
   — Le tableau `args[]` est vide ou de longueur plus petite que 3.
   — Au moins un des éléments du tableau `args[]` n'est pas de type `double`.
   — Le coefficient a est zéro.
   — Le discriminant est négatif.
   Vérifier que votre programme attrape bien les exceptions qui pourraient être levées dans chacune de ces situations.

Exercice 2 *Chaîne de restauration*
Le but de cet exercice est d'écrire un programme capable de gérer le système de reservation d'une chaîne de restauration qui possède plusieurs restaurants dans la région parisienne. Ces restaurants offrent trois types de menus, un menu "Entrée/Plat ou Plat/Dessert", un menu "Entrée/Plat/Dessert" et un menu "Dégustation" à 5 plats. Le prix du menu à 3 plats est 20% plus cher que le menu à 2 plats et le menu à 5 plats est 50% plus cher que le menu à 2 plats. Les boissons sont comprises dans le prix de chaque menu et des commandes "à la carte" ne sont pas possibles.

Chaque restaurant sera représenté comme une instance d'une classe `Restaurant` définie comme suit :

— une chaîne de caractères qui contient la localisation géographique du restaurant, (par ex. *Versailles*, Saint-Denis, ...)
— un entier représentant le nombre de places du restaurant
— un réel représentant le prix du menu à 2 plats
— un entier représentant le nombre de menus à 2 plats vendus
— un entier représentant le nombre de menus à 3 plats vendus
— un entier représentant le nombre de menus à 5 plats vendus

Les valeurs des 3 premières caractéristiques (nom du restaurant, nombre de places, prix du menu) sont fixées lors de la création d'un objet `Restaurant` (c'est-à-dire, sont passés en paramètre au constructeur).

La classe `Restaurant` possède les méthodes suivantes :

— `public int nombrePlacesDisponibles()` : Cette méthode calcule est renvoie le nombre de places encore disponibles dans le restaurant.

— `public void vendreMenus(int nombre, int quelMenu)` : Cette méthode permet de vendre des menus pour ce restaurant. La variable `nombre` indique le nombre de menus demandés et la variable `quelMenu` indique quel menu est choisi. Par convention, `quelMenu` vaut 2 pour un menu à 2 plat, 3 pour un menu à 3 plats, et 5 pour un menu à 5 plats. Si le nombre de menus demandés est supérieur au nombre de places disponibles dans le restaurant, la vente n'est pas effectuée et la méthode affiche un message indiquant que la vente n'est pas possible. Sinon, les variables d'instance correspondant au nombre de menus vendus sont mises à jour et le prix à payer est affiché.

— `public void remiseAZero()` : Cette méthode permet lorsque le service est terminé de remettre à 0 les compteurs pour le nombre de menus vendus en vue du prochain service.

— `public double chiffreAffaires()` : Cette méthode renvoie le chiffre d'affaires produit par le restaurant pour le service en cours (total des ventes depuis la création de l'objet restaurant ou la dernière remise à zero du nombre de menus vendus).

— `public double tauxRemplissage()` : Renvoie le taux (pourcentage) de remplissage du restaurant.

— `public int getNombrePlaces()` : Renvoie le nombre de places du restaurant.

— `public int getNombreClients()` : Renvoie le nombre de menus réservés au restaurant.

— `public String toString()` : Cette méthode renvoie une représentation sous forme d'une chaîne de caractères de l'objet `Restaurant`. Cette chaîne indique la valeur de chacun des attributs (chacune des variables d'instances) de l'objet.

— `public void affichage()` : Cette méthode affiche à l'écran la representation de l'objet codée par la méthode `toString()`. Par exemple, pour un restaurant de 40 places à Versailles, dont 20 menus à 2 plats (15 euros/menu), 6 menus à 3 plats et 10 menus à 5 plats ont été vendus, l'affichage de la chaîne renvoyée par `toString()` pourrait être la suivante :

```
Restaurant : Versailles
Nombre de places : 40
Prix du menu a 2 plats : 15 euros
20 menus a 2 plats vendus,
6 menus a 3 plats vendus,
10 menus a 5 plats vendus.
```

1. **Ecrire le code pour la classe** `Restaurant` en respectant toutes les spécifications décrites ci-dessus.

2. **Tester la classe** `Restaurant`. Créer deux instances de la classe `Restaurant` correspondant aux informations définies dans la table ci-dessous. Dix menus à 2 plats, quinze menus à 3 plats et quatre menus à 5 plats doivent être achetés pour le premier restaurant. Pour le deuxième restaurant, quatorze menus à 2 plats, 16 menus à 3 plats et 5 menus à 5 plats doivent être achetés. Finalement les attributs des deux restaurants doivent être affichés ainsi que le nombre de places encore disponibles et le chiffre d'affaires produit.

Nom	**Nombre de places**	**Prix du menus à 2 plats**
Paris Marais	50	15
Montreuil	35	13.5

3. **Ecrire une classe** `ChaineRestauration`. Cette classe doit avoir une seule instante de classe, qui sera un objet `ArrayList<Restaurant>`. Elle sera composée des constructeurs et méthodes suivants :
   — Un constructeur sans paramètres.
   — Une méthode `public void ajouterRestaurant(Restaurant r)` qui ajoute un restaurant à la liste des restaurants de la chaîne.
   — Une méthode `public int taille()` qui doit renvoyer la taille de la liste, c'est-à-dire le nombre de restaurants de la chaîne.
   — Une méthode `public Restaurant getRestaurant(int i)` qui doit renvoyer le i-ième restaurant de la chaîne.

4. **Ecrire une classe** `Reservation` respectant les spécifications suivantes. Ce programme permet d'enregistrer les reservations effectuées dans les différents restaurants et de calculer et d'afficher le taux d'occupation et le chiffre d'affaires produit par chaque restaurant lorsque la vente des menus pour le service est terminé.

   Le programme de réservation est lancé au début de la mise en vente des menus pour le prochain service. Chaque restaurant est identifié par un numéro unique (les numéros allant de 1 à n, n étant le nombre total de restaurants). Lorsqu'un client se connecte, il tape le numéro du restaurant auquel il souhaite aller. Le programme affiche alors les différents attributs du restaurant sélectionné (le nom du restaurant, le nombre de places, le nombre de menus vendus...). Le client fournit ensuite au programme le nombre de menus qu'il souhaite acheter en indiquant également quel menu il désire. Si la demande du client peut être satisfaite le programme affiche le prix à payer sinon il affiche un message indiquant que le nombre de places demandé est incorrect. Lorsque l'achat est terminé, le programme affiche alors pour chaque restaurant son état (la valeur de ses attributs), son taux d'occupation et le chiffre d'affaires produit. Le programme calcule aussi le chiffre d'affaires total et l'affiche.

   La structure du programme pourrait être la suivante :

```
creation des objets Restaurant
venteTerminée un boolean initialisé à false
tantque (! venteTerminée)
      lire un numero de restaurant
           si le numéro du restaurant est correct alors
               afficher les informations du restaurant
               lire le nombre de menus à acheter ainsi que le type du menu
               vendre pour le restaurant sélectionné le nombre de menus demandés
               demander au client s'il veut poursuivre ou non la reservation
               selon la réponse mettre à jour venteTerminée
           sinon
                afficher un message d'erreur "numéro de restaurant incorrect"
fintantque


calculer et afficher pour chaque restaurant le taux de remplissage et le chiffre d'affaires
produit
afficher le chiffre d'affaires total
```

   Pour lire les données que l'utilisateur tape au clavier, vous pouvez utilisez la classe `Scanner` (`import java.util.Scanner;`). Initialiser une instance de cette classe de la façon suivante :

```
Scanner scanner = new Scanner(System.in);
```

   Pour lire un entier, vous pouvez alors écrire

```
int entier = scanner.nextInt();
```

   Tester votre programme pour une chaîne de restauration qui est composée de six restaurants dont la description est donnée ci-dessous.

Nom	**Nombre de places**	**Prix du menus à 2 plats**
Paris Marais	50	15
Paris Batignolles	25	15.6
Paris Montparnasse	54	14.2
Montreuil	35	13.5
Montrouge	43	13.8
Versailles	47	14
Viroflay	63	12.5

5. **Modifier les classes précédentes pour prendre en compte certaines des exceptions qui peuvent être lancées pendant l'execution :**

— Créer une classe d'exception

`public class MenuInexistantException extends Exception`

avec un constructeur qui prend comme paramètre un entier i et qu'utilise le constructeur de sa super-classe avec comme paramètre le message `"Un menu à i plats n'est disponible dans ce restaurant."`

— Modifier la méthode `vendreMenus(int i nombre, int quelMenu)` pour qu'elle lance une exception du type
`MenuInexistantException` si le numéro du menu demandé n'est pas 2, 3 ou 5.

— Créer une classe d'exception

`public class PasAssezDePlacesException extends Exception`

avec un constructeur qu'utilise le constructeur de sa super-classe avec comme paramètre le message `"Malheureusement, il n'y a pas assez de places disponibles dans le restaurant."`

— Modifier la méthode `vendreMenus(int i nombre, int quelMenu)` pour qu'elle lance de plus une exception du type
`PasAssezDePlacesException` si le nombre de menus demandés dépasse le nombre de places actuellement disponibles dans le restaurant choisi.

— Créer une classe d'exception

`public class RestaurantInexistantException extends ArrayIndexOutOfBoundsException`

avec un constructeur qui prend comme paramètre un entier i et qu'utilise le constructeur de sa super-classe avec comme paramètre le message `"Le restaurant avec numero i n'existe pas."`.

— Modifier la méthode `getRestaurant(int i)` pour qu'elle lance une exception du type
`RestaurantInexistantException` si le numéro du restaurant n'est pas valide.

— Modifier les méthodes de la classe `Reservation` pour qu'elles attrapent les exceptions des trois types définies ci-dessus qui peuvent être lancées. Mettre le code susceptible à introduire des erreurs dans un bloc `try` et créer un nombre de blocs `catch` correspondant aux nombre d'exceptions qu'on souhaite gérer. Faire en sorte que si une exception d'un de ces trois types se produit, le programme ne se termine pas. À la place, un message doit s'afficher à l'écran et l'utilisateur doit être demandé de reprendre la saisie erronée.