Td3 : Gestion des exceptions

Exercice 1

On souhaite gérer les exceptions afin de s'assurer que la longueur du rayon d'un cercle est bien un réel positif. Dans le cas où l'on donnera une valeur négative, la longueur du rayon du cercle sera initialisée à son opposé .Les étapes à suivre sont les suivantes :

1- Écrire la déclaration de la classe ValRayonValideException qui permettra d'instancier une exception dès qu'une longueur de rayon négative sera donnée.
2- Écrire la déclaration de la classe CouleurValideException qui permettra d'instancier une exception dès que la couleur est un objet nul ou de longueur nul ou de couleur rouge.
3- Construire la classe Cercle en gérant correctement le code de ses méthodes qui sont à risque.
4- Tester la classe Cercle dans la fonction main.

Le squelette de la classe Cercle est donné par :

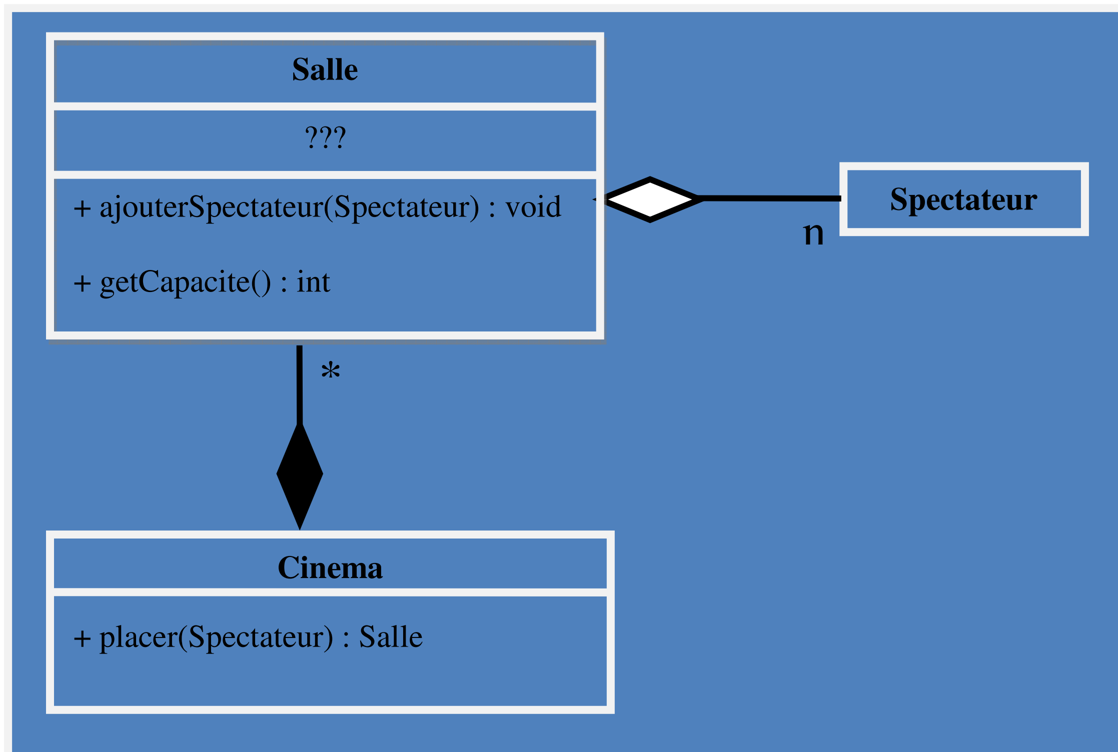
public class Cercle {

private double x, y, rayon;	(x, y) est le centre du cercle dont le rayon est donné par rayon.
private String couleur;	La couleur du cercle
public Cercle(double x, double y, double rayon, String couleur)	Constructeur de la classe
public double getRayon()	
public String getCouleur()	
public void setRayon(double newRayon)	Fonction permettant de modifier la valeur du rayon du cercle. Si la valeur du nouveau rayon donnée par newRayon est négative alors une exception de type ValRayonValideException est levée et traitée.
public void setCouleur(String newCouleur)	Fonction permettant de modifier la couleur du cercle courant, en gérant des exceptions de type NullPointerException si le paramètre newCouleur est nul ou s'il est une chaine de caractères vide ou si newCouleur est de couleur rouge ceci en utilisant la classe CouleurValideException
public String toString()	Fonction permettant d'afficher les caractéristiques du cercle courant

}

Exercice 2

On veut modéliser en Java le fonctionnement simplifié d'un cinéma (cf. figure ci-dessous) qui ne propose qu'un seul film mais dispose de plusieurs salles. Une **Salle** peut accueillir un nombre fixe (**n**) de **Spectateur**. Lorsqu'un spectateur arrive, on le place dans la dernière salle disponible. Si la salle est pleine, alors on ouvre une nouvelle salle. Un **Cinema** peut ouvrir autant de salles (*) qu'il le souhaite.



1- Définissez en Java une classe **SallePleineException** qui sera levée lorsque le cinéma essaye de placer un spectateur dans une salle pleine. Cette exception doit connaître le spectateur qui a déclenché sa levée.
2- Donnez l'implémentation de la classe Salle avec ses champs, son constructeur et la méthode **ajouterSpectateur(Spectateur)** qui ajoute un spectateur et lève, sans la traitée, l'exception **SallePleineException** lorsque la salle est pleine. Il faut choisir la structure de donnée qui vous semble être la plus appropriée (tableau, liste, …).
3- Donnez l'implémentation de la classe Cinema avec ses champs, son constructeur et la méthode **placer(Spectateur)** qui ajoute un spectateur à la dernière salle disponible. Si l'exception est levée, alors on créé une nouvelle salle avec sa propre capacité dans laquelle on place le spectateur.

Exercice 3

1- L'objectif de cet exercice est la construction d'une application qui permet de gérer des entreprises avec leurs employés. Un employé est une entité qui a un nom, une date de naissance, un numéro de téléphone et l'entreprise dans laquelle il travaille. Une entreprise est une entité qui a un nom et une liste des employés. On peut emboucher et renvoyer des employés d'une entreprise. Attention, on ne peut renvoyer un employé d'une entreprise dans laquelle il ne travaille pas ou ajouter un employé déjà embouché dans une entreprise donnée.
   L'association entre une entreprise et ses employés sera bidirectionnelle : une entreprise connaît ses employés et un employé connaît son entreprise ; attention de bien gérer les 2 "bouts" de l'association. Par exemple, si vous ajoutez un employé dans une entreprise ETR, l'employé doit être ajouté à la liste des employés de l'entreprise mais l'entreprise de l'employé doit aussi être mise à la valeur ETR.

2- En plus, on ajoute un employé particulier à chaque entreprise qui prend le poste de directeur. Un directeur a un salaire que l'on donne quand on le crée. On n'a qu'un seul directeur par entreprise. Un problème de conception se pose : comment faire pour empêcher la création de plusieurs directeurs dans une entreprise ?

   Indication pour la création de la fonction

   On peut utiliser une méthode creeDirecteur mais, cette fois-ci, cette méthode prendra, en plus, un paramètre du type Entreprise.

   Dans cette méthode, voici le dialogue qui va s'instaurer entre la classe Directeur et l'entreprise à qui on veut ajouter un directeur :

   - La classe Directeur : << Entreprise, as-tu déjà un directeur ? >>
   - L'entreprise : << Non >> (sinon la méthode creeDirecteur s'arrête ici)
   - La classe Directeur crée alors un nouveau directeur et le présente à l'entreprise :
   - La classe Directeur : << Entreprise, enregistre ce nouvel employé. >>
   - L'entreprise teste si le nouvel employé est un directeur. Si c'est le cas, elle ajoute le directeur à la liste de ses employés et elle enregistre le fait qu'elle a maintenant un directeur. (Si ça n'avait pas été un directeur, elle se serait contentée de l'ajouter comme employé.).

L'ensemble des fonctionnalités demandées sont données par les squelettes des classes à construire:

public class Employe{

private String nom;	ne doit pas être vide ni null
private String dateNaissance;	doit être une date valide avec un format bien défini donné par : yyyy-MM-dd
private String telephone;	doit être une chaine qui représente un entier valide et contenir exactement 10 chiffres.
private Entreprise entreprise	doit être null si l'employé n'est pas embouché par aucune entreprise

public Employe(String nom, String dateNaissance, String telephone)	Constructeur de la classe
public String getNom()	
public String getDateNaissance()	
public String getTelephone()	
public Entreprise getEntreprise()	
public void setNom(String newNom)	Fonction permettant de modifier le nom de l'employé courant, en gérant des exceptions de type NullPointerException si le paramètre newNom est nul ou s'il est une chaine de caractères vide ceci en utilisant la classe ChaineDeCharacteresException
public void setDateNaissance(String NewDateNaissance)	Fonction permettant de modifier la date de naissance d'un employé. Le format de la date valide est donnée par yyyy-MM-dd de plus une date valide est une date qui n'est pas postérieure à la date système. Cette méthode traite le type d'exception DateMalFormed si NewDateNaissance est une date non valide
public void setTelephone(String NewTelephone)	Fonction permettant de modifier le numéro de téléphone d'un employé. Le numéro de téléphone est valide s'il contient exactement 10 chiffres. Cette méthode gère l'exception de type numeroTelephoneException si le numéro est invalide.
public void setEntreprise(Entreprise NewEntreprise)	Fonction permettant de modifier l'entreprise employeur d'un employé. La modification vérifie le fait que l'employé sera embouché par le paramètre NewEntreprise et virer de l'ancienne entreprise si elle est non vide. La fonction gère une exception de type EmployeException dans le cas ou NewEntreprise contient déja l'employé.
public void demissionne() throws EmployeException	Fonction permettant de gérer la démission d'un employé de son entreprise sans être embouché par une autre entreprise
public String toString()	Fonction permettant d'afficher les infos d'un employé avec le nom, le numéro de téléphone, date de naissance et le nom de son entreprise employeur s'il n'est pas au chômage
public boolean valeurTelephoneCorrecte(String intEnString)	Fonction permettant de vérifier si une valeur sous format String représente un entier avec exactement 10 chiffres ou pas.
public boolean dateValide(String dateTeste)	Fonction permettant de vérifier si une date est valide ou non. Le format par défaut est yyyy-MM-dd
public boolean equals(Object emp)	Fonction permettant de comparer l'employé courant et l'employé emp donné comme argument à la fonction

}

public class Entreprise {

private String nom;	Le nom de l’entreprise
private ArrayList<Employe> ListEmployes;	Liste des employés de l’entreprise courante
public Entreprise(String nom)	Constructeur de la classe
public String getNom()	
private Directeur directeur	Directeur de l’entreprise qui est unique
public void setNom(String newNom)	Fonction permettant de modifier le nom de l'entreprise, en gérant des exceptions de type NullPointerException si le paramètre newNom est nul ou s'il est une chaine de caractères vide ceci en utilisant la classe ChaineDeCharacteresException
public ArrayList<Employe> getListEmployes( )	
public boolean estUnEmploye(Employe emp)	Fonction permettant de vérifier si l'employé donné comme paramètre est un employé de l'entreprise courante ou non
public void emboucherEmploye(Employe emp) throws EmployeException	Fonction permettant d'emboucher un employé, s'il ne l'est pas, dans l'entreprise courante. Si l'employé est déjà embouché par une autre entreprise non null alors il faut qu'il demissionne de cette dernière avant que l'entreprise courante puisse l'emboucher
public void virerEmploye(Employe emp) throws EmployeException	Fonction permettant de virer un employé de l'entreprise courante
public String toString()	Fonction permettant d'afficher le nom et la liste des noms de tous les employés de l'entreprise courante
public boolean equals(Object entr)	Fonction permettant de comparer l'entreprise courante à l'entreprise entr donnée comme paramètre à la fonction.

}

public class Directeur ?????

public class Main {

```
public static void main(String[ ] args) {
 Entreprise e1 = new Entreprise("IBM");
 Entreprise e2 = new Entreprise("Sun");
 Employe emp1 = new Employe("Dupond", "1999-02-23", "1234567890");
 e1.emboucherEmploye(emp1);
 System.out.println(e1);
 System.out.println(e2);
 Directeur dir = Directeur.creerDirecteur("ahmedDirecteur", "2001-12-22", "1234567877", 123.50, e2);
 System.out.println(e2);
 System.out.println(dir);
 //********** autre employés **************************
 Employe emp2 = new Employe("Karime", "1967-02-19", "1239997811");
 Employe emp3 = new Employe("Nada", "1559-12-14", "1234567888");
 Employe emp4 = new Employe("René", "1927-08-09", "1234567777");
 e2.emboucherEmploye(emp2);
 e2.emboucherEmploye(emp2);
 e2.emboucherEmploye(emp3);
 e1.emboucherEmploye(emp4);
 System.out.println("après ajout d'autre employés");
 System.out.println(e1);
 System.out.println(e2);
 System.out.println("Karime démissionne.");
 emp2.demissionne();
 System.out.println(e1);
 System.out.println(e2);
```

```
        System.out.println("ahmedDirecteur passe chez IBM");

        e1.emboucherEmploye(dir);

        e1.emboucherEmploye(dir);

        System.out.println(e1);

        System.out.println(e2);

        System.out.println(dir);

        // **************changement de directeur ****************************

        System.out.println("changement de directeur si c'est possible");

        Directeur dir_2 = Directeur.creerDirecteur("FaridDirecteur", "2001-12-22", "1234567877", 123.50, e2);

        System.out.println("dir_2 "+dir_2);

        System.out.println(e1);

        System.out.println(e2);

        e1.emboucherEmploye(dir_2);

        System.out.println(e1);

        System.out.println(e2);

        System.out.println(dir);

    }
```