

LICENCE INFORMATIQUE

Module FLIN 504 – Programmation par objets 1

Session 1 – Janvier 2009

Tous documents autorisés

Problème (Héritage) 45 mn

Nous nous intéressons à la réalisation d'un logiciel destiné à une agence de voyage. Nous étudions une petite partie de la modélisation de réservations hôtelières.

Pour les besoins de cette gestion, les réservations sont décrites par un numéro d'identification (chaîne de caractères).

On distingue ensuite deux sortes de réservations.

Les réservations simples ou *séjours*, concernent une période de séjour sans interruption dans un même hôtel. On enregistrera dans leur description :

- une date de début,
- un nombre de journées,
- le nom de l'hôtel,
- le prix d'une journée.

Les *réservations groupées* sont des réservations qui se composent de plusieurs autres.

Elles sont vides à la création et une méthode `ajoute(Reservation r)` permet de leur intégrer successivement des réservations (séjours ou réservations groupées).

Le *prix* d'un séjour s'obtient comme le produit du nombre de journées par le prix d'une journée.

Le *prix* d'une réservation groupée s'obtient comme la somme des prix de ses constituants, à laquelle s'applique une réduction. La réduction est de 50% lorsque la réservation groupée se compose d'au moins 7 séjours différents pour un nombre de journées cumulées supérieur à 31. La réduction est de 10% lorsque la réservation regroupe au moins 2 séjours différents pour un nombre de journées cumulées supérieur à 15.

Une *facture* comprend dans tous les cas le numéro de la réservation, le nombre de journées cumulées et le prix total. Dans le cas des réservations groupées, on complète par le nombre de séjours. Pour l'écrire en Java, proposez une méthode de signature `String facture()`.

Question. Donnez un schéma de l'ensemble des classes que vous envisagez, puis écrivez le code Java correspondant, incluant les constructeurs et toutes les méthodes, à l'exception des accesseurs que vous supposerez néanmoins exister. Les attributs seront privés.

Exercice 1 (exceptions) 15 mn

Soit les classes `CastingException` et `Sauvageon` suivantes :

```
public class CastingException extends Exception{}

public class Sauvageon
{
    private String nom;
    private int age;

    public Sauvageon (String nom, int age) throws CastingException
    {
        System.out.println("début constructeur");
        if (age<20 || age>30) throw new CastingException();
        this.nom = nom;
        this.age = age;
        System.out.println("fin constructeur");
    }

    public void setAge(int age) throws CastingException
    {
        System.out.println("début setAge");
        if (age<20 || age>30) throw new CastingException();
        this.age=age;
        System.out.println("fin setAge");
    }

    public void vieillir()
    {
        System.out.println("début vieillir");
        try
        { setAge(age+1); }
        catch (CastingException c)
        { System.out.println("erreur de casting
            récupérée par la méthode vieillir"); }
        System.out.println("fin vieillir");
    }
}
```

Indiquer ce qui sera affiché lors de l'exécution de la méthode main dans chacun des 3 cas suivants (seul ce qui est écrit à partir des `System.out.println()` nous intéresse) :

```
1) public static void main(String[] args) throws CastingException
{
    System.out.println("début main");
    Sauvageon s = new Sauvageon("Jojo",5);
    System.out.println("fin main");
}
```

```
2) public static void main(String[] args)
{
    System.out.println("début main");
    try
    {
        Sauvageon s = new Sauvageon("Jojo",22);
        s.setAge(33);
    }
    catch (CastingException c)
    { System.out.println("erreur de casting récupérée par main"); }
    System.out.println("fin main");
}
```

```

    }
3) public static void main(String[] args)
{
    System.out.println("début main");
    try
    {
        Sauvageon s = new Sauvageon("Jojo",30);
        s.vieillir();
    }
    catch (CastingException c)
    { System.out.println("erreur de casting récupérée par main"); }
    System.out.println("fin main");
}

```

Exercice 2 (généricité paramétrique) 30 mn

Cet exercice prolonge le contrôle continu.

Dans le cadre d'un système de gestion de tableaux d'affichage d'annonces de diverses sortes, on propose une première description des annonces par l'interface suivante.

```

public interface Annonce
{
    boolean vivante();
}

```

Puis on propose une classe paramétrée représentant les tableaux d'annonces.

```

class TableauAnnonces<E extends Annonce>
{
    public static int dateJour=77;
    private E[][] surface;
    public TableauAnnonces(E[][] zoneStockage)
    {surface=zoneStockage;}
    public void accrocher(int x, int y, E p){surface[x][y]=p;}
    public E decrocher(int x, int y)
    {
        E p = surface[x][y];
        if (p == null)
        {
            System.out.println("Pas de postit dans cette case");
            return null;
        }
        if (p.vivante())
        {System.out.println("decrochage impossible, postit encore
valable");
            return null;}
        else
        {surface[x][y]=null; return p;}
    }
}

```

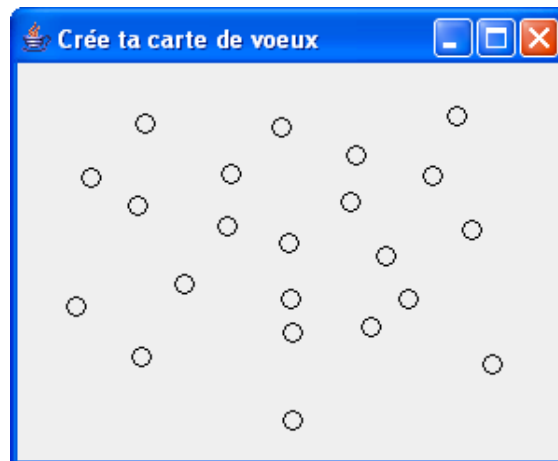
Question 1. Proposez une classe `RapportErreur` décrivant les rapports d'erreurs dans un système informatique et respectant le type `Annonce`. Un rapport d'erreur comprend le nom de son auteur, un texte descriptif, la date de création, et un booléen permettant de savoir si l'erreur a été traitée. Un rapport d'erreur est une annonce vivante tant que l'erreur n'a pas été traitée.

Question 2. Etudiez l'ajout d'une méthode `void affiche()` à la classe paramétrée `TableauAnnonces` qui affiche le contenu des annonces vivantes du tableau. Modifiez pour cela l'interface `Annonce` et par conséquent la classe `RapportErreur` avec une opération qui donne accès au contenu des annonces.

Question 3. En utilisant l'héritage et l'invocation (instanciation de classe paramétrée), proposer une sous-classe non paramétrée de `TableauAnnonces`, spécialisée pour les rapports d'erreurs : cette sous-classe que vous appellerez `TableauErreur` doit permettre de manipuler et notamment d'afficher des rapports d'erreur. Redéfinissez sa méthode `affiche` de manière à ajouter l'entête « Erreurs à traiter » avant l'affichage des erreurs non traitées.

Exercice 3 (programmation événementielle) 30 mn

On se place dans le cadre d'un logiciel de dessin pour enfants. On se restreindra dans cet exercice à une fonctionnalité de ce logiciel : celle qui permet de créer des flocons de neige par simple clic souris.



La fenêtre principale de l'application contient un panneau, sur lequel est affiché le dessin (seulement des cercles dans cet exercice). Le panneau est représenté par une instance d'une classe `Panneau` héritant de `JPanel`. Les cercles sont représentés par la classe suivante :

```
public class Cercle
{
    protected int col; //colonne du point d'ancrage
    protected int lig; //ligne du point d'ancrage
    protected int lh; //long. côté horizontal du rectangle englobant
    protected int lv; //long. côté vertical du rectangle englobant

    protected Color cTrait ; // couleur du trait

    public Cercle (int col, int lig, int lh, int lv)
    {
        this.col = col;
        this.lig = lig;
        this.lh = lh;
        this.lv = lv;
    }
}
```

```

    cTrait = Color.BLACK;
}

public Cercle (int col, int lig, int lh, int lv, Color cTrait)
{
    this.col = col;
    this.lig = lig;
    this.lh = lh;
    this.lv = lv;
    this.cTrait = cTrait;
}

public void dessine (Graphics g)
{
    g.setColor(cTrait);
    g.drawOval(col,lig,lv,lh);
}
}

```

On vous demande d'écrire la classe Panneau de façon à ce que :

- un clic souris sur le panneau à une position (x,y) crée un cercle à cette position ;
- l'ensemble des cercles tracés est affiché en permanence.

Vous n'avez que la classe Panneau à écrire (les classes représentant la fenêtre et l'application ne nous intéressent pas). Concentrez-vous sur l'écriture du code qui implémente la fonctionnalité demandée.