Programmation Java @ Et3
Polytech Paris-Sud | 2024-25

Nom : BRACQ
Prénom : Paolo

Exercice 5

On souhaite mettre en place un éditeur simple de dessins à base de formes géométriques, sous forme d'un package `et3.java.geometrie` regroupant les classes permettant de manipuler des formes géométriques bidimensionnelles simples, ici : des cercles, des rectangles et des carrés. On aura les contraintes suivantes :

— chaque forme possède un centre de gravité (instance de `java.awt.Point`) ainsi qu'une couleur (instance de `java.awt.Color`) ; les attributs correspondants ne devront pas être directement partagés avec l'extérieur de la classe ;
— un cercle possède un rayon ;
— un rectangle possède une largeur et une hauteur ;
— toute forme géométrique doit pouvoir avoir les comportements suivants :
— translation, prenant en paramètres deux nombres représentant un déplacement horizontal et vertical
— représentation sous la forme d'une chaîne de caractères donnant le nom de la forme et la description textuelle de chacun de ses attributs. Par exemple, la chaîne de caractères produite pour un cercle pourrait être :

```
[ Cercle
      [ centre de gravité : x =10 ; y =4]
      [ rayon : 20,5]
      [ couleur : r =82 ; g =255 ; b =0]
]
```

1. Implémentez la spécification demandée en mettant notamment en œuvre une classe abstraite.

#5.1 code

```java
package geometrie;

import java.awt.*;

abstract class Forme {
    private String Name;
    private Point gravityCenter;
    private Color color;

    /**
     * Constructeur de la classe abstraite forme
     * @param center type:Point
```

```
     * @param color type:Color
     */
    public Forme(String name, Point center, Color color){
        this.Name=name;
        this.color=color;
        this.gravityCenter=center;
    }

    /**
     * Méthode de translation en utilisant setLocation
     * @param x type:double
     * @param y type:double
     */
    public void Translation(double x, double y){
        gravityCenter.setLocation(x,y);
    }

    /**
     * Méthode d'accès à la couleur
     * @return type Color
     */
    public Color getColor(){
        return color;
    }

    /**
     * Méthode d'accès au centre de gravité
     * @return type:Point
     */
    public Point getGravityCenter(){
        return gravityCenter;
    }

    /**
     * Redéfinition de la fonction toString
     * @return chaine de caractères
     */

    @Override
    public String toString(){
        return Name +"\n "
                +" centre de gravité : x = "+gravityCenter.getX()+" ; y = "+gra-
vityCenter.getY()+"\n "
                + " couleur : r = "+color.getRed()+" ; g = "+color.getGreen()+"
; b = "+color.getBlue();
    }



}
```

```
package geometrie;

import java.awt.*;

public class Cercle extends Forme{
    private double rayon;

    /**
     * Constructeur d'un cercle
     * @param center type:Point
     * @param color type: Color
     * @param rayon type: double
```

```
     */
    public Cercle (Point center, Color color,double rayon){
        super("Cercle",center,color);
        this.rayon=rayon;
    }

    @Override
    public String toString(){
        return super.toString()+ " \n  rayon : "+ rayon;
    }

}
```

```
package geometrie;

import java.awt.*;

public class Rectangle extends Forme{
    private double largeur;
    private double longueur;

    /**
     * Constructeur d'un rectangle
     * @param center type:Point
     * @param color type: Color
     * @param largeur type: double
     * @param longueur type: double
     */
    public Rectangle (Point center, Color color, double largeur, double lon-
gueur){
        super("Rectangle",center,color);
        this.largeur=largeur;
        this.longueur=longueur;
    }

    @Override
    public String toString(){
        return super.toString()+ " \n  largeur : "+ largeur+" \n  longueur :
"+longueur;
    }
}
```

```
package geometrie;

import java.awt.*;

public class Carre extends Forme{
    private double size;

    /**
     * Constructeur d'un carré
     * @param center type:Point
     * @param color type: Color
     * @param size type: double
     */
    public Carre (Point center, Color color, double size){
        super("Carré",center,color);
        this.size=size;
    }
    @Override
    public String toString(){
        return super.toString()+ " \n  côté : "+ size;
```

```
    }
}
```

Et voici une fonction pour tester tout cela :

```
package geometrie;

import java.awt.*;

public class Test {
    public static void main(String[] args){
        Point centre=new Point( 10,4 );
        Color couleur=new Color(82,255,0);
        Cercle test1 =new Cercle(centre,couleur,20.5);
        Rectangle test2 =new Rectangle(centre,couleur,20.5,4);
        Carre test3 =new Carre(centre,couleur,20.5);
        System.out.println(test1.toString());
        System.out.println(test2.toString());
        System.out.println(test3.toString());

    }

}
```

Et le résultat :

```
Cercle
  centre de gravité : x = 10.0 ; y = 4.0
  couleur : r = 82 ; g = 255 ; b = 0
  rayon : 20.5
Rectangle
  centre de gravité : x = 10.0 ; y = 4.0
  couleur : r = 82 ; g = 255 ; b = 0
  largeur : 20.5
  longueur : 4.0
Carré
  centre de gravité : x = 10.0 ; y = 4.0
  couleur : r = 82 ; g = 255 ; b = 0
  côté : 20.5
```

2. On souhaite ajouter une forme de carré comme une spécialisation de la classe pour un rectangle. On veillera à ce que cette classe ne puisse plus être dérivée.

#5.2 code

On modifie dans un premier temps notre définition de forme :

```
public Forme(Point center, Color color){
    Name=getClass().getSimpleName();
    this.color=color;
    this.gravityCenter=center;
}
```

Et du coup carré devient :

```
public final class Carre extends Rectangle{
    private double size;
```

```
    /**
     * Constructeur d'un carré
     * @param center type:Point
     * @param color type: Color
     * @param size type: double
     */
    public Carre (Point center, Color color, double size){
        super(center,color,size,size);
        this.size=size;
    }

    @Override
    public String toString(){
        return super.toString()+ " \n  côté : "+ size;
    }
}
```

3. On souhaite à présent pouvoir construire des listes de formes géométriques pour opérer des regroupements d'objets. Commencez par créer une classe de test définissant un programme créant une collection sous forme de liste utilisant la classe `java.util.ArrayList<E>`. Ajoutez à cette collection un objet de chaque type de forme définie, et affichez le contenu de la liste à l'aide d'un itérateur.

#5.3 code

```
package geometrie;

import java.awt.*;
import java.util.*;
import java.util.List;

public class Test {
    public static void main(String[] args){
        Point centre=new Point( 10,4 );
        Color couleur=new Color(82,255,0);
        Cercle test1 =new Cercle(centre,couleur,20.5);
        Rectangle test2 =new Rectangle(centre,couleur,20.5,4);
        Carre test3 =new Carre(centre,couleur,20.5);

        List<Forme> listeFormes = new ArrayList<Forme>();
        listeFormes.add(test3);
        listeFormes.add(test2);
        listeFormes.add(test1);

        Iterator<Forme> iterateur = listeFormes.iterator();
        while (iterateur.hasNext()) {
            Forme forme = iterateur.next();
            System.out.println(forme.toString());
        }

    }

}
```

```
Les résultats sont :
Carre
  centre de gravité : x = 10.0 ; y = 4.0
```

```
  couleur : r = 82 ; g = 255 ; b = 0
  largeur : 20.5
  longueur : 20.5
  côté : 20.5
Rectangle
  centre de gravité : x = 10.0 ; y = 4.0
  couleur : r = 82 ; g = 255 ; b = 0
  largeur : 20.5
  longueur : 4.0
Cercle
  centre de gravité : x = 10.0 ; y = 4.0
  couleur : r = 82 ; g = 255 ; b = 0
  rayon : 20.5
```

4. Quelles sont les méthodes `toString` qui sont appelées pour chaque objet de la collection précédente ? Que se passe-t-il si l'on enlève un élément de la collection à un indice qui n'existe pas, et comment éviter une erreur à l'exécution ?

#5.4 réponses

Ce sont les méthodes toString des classes réelles. Lorsque l'on supprime un élément de la liste alors que l'index n'existe pas on obtient une exception :
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index 3 out of bounds for length 3.

Pour éviter une erreur à l'exécution, il faut avant de supprimer un élément, vérifier que l'indice est dans les limites de la liste (indiceASupprimer >= 0 && indiceASupprimer < listeFormes.size()). Si l'indice est valide, l'élément est supprimé. Sinon, un message d'erreur est affiché.

5. Créez à présent une classe pour représenter une simple collection pour des formes géométriques uniquement par réutilisation (héritage) de la classe `java.util.ArrayList<E>`. Ajoutez à cette classe une méthode `translation` qui déplace l'ensemble de ses formes géométriques, et redéfinissez la méthode `toString` de manière à ce qu'elle produise un affichage plus approprié pour la collection. Testez l'ensemble de ces méthodes.

#5.5 code

```java
package geometrie;

import java.util.ArrayList;

public class CollectionFormes extends ArrayList<Forme> {

    public void Translation(double x, double y){
        for (Forme forme:this) {
            forme.Translation(x,y);
        }
    }


    @Override
    public String toString() {
```

```
        StringBuilder sb = new StringBuilder();
        sb.append("Collection de formes :\n\n");
        int i=1;
        for (Forme forme : this) {
            sb.append("Forme n°"+i+": ").append(forme.toString()).ap-
pend("\n\n");
            i+=1;
        }
        return sb.toString();
    }


}
```

Et avec le test suivant :

```
package geometrie;

import java.awt.*;
import java.util.*;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        Point centre = new Point(10, 4);
        Color couleur = new Color(82, 255, 0);
        Cercle test1 = new Cercle(centre, couleur, 20.5);
        Rectangle test2 = new Rectangle(centre, couleur, 20.5, 4);
        Carre test3 = new Carre(centre, couleur, 20.5);

        CollectionFormes collection = new CollectionFormes();


        collection.add(test1);
        collection.add(test2);
        collection.add(test3);


        System.out.println(collection);


        collection.Translation(2, 3);


        System.out.println("Après translation : \n");
        System.out.println(collection);
    }
}
```

On obtient :

```
Forme n°1: Cercle
  centre de gravité : x = 2.0 ; y = 3.0
  couleur : r = 82 ; g = 255 ; b = 0
  rayon : 20.5

Forme n°2: Rectangle
  centre de gravité : x = 2.0 ; y = 3.0
  couleur : r = 82 ; g = 255 ; b = 0
  largeur : 20.5
```

```
  longueur : 4.0

Forme n°3: Carre
  centre de gravité : x = 2.0 ; y = 3.0
  couleur : r = 82 ; g = 255 ; b = 0
  largeur : 20.5
  longueur : 20.5
  côté : 20.5
```

6. On souhaite à présent définir une collection qui permet d'imposer que ses membres soient des formes géométriques d'un même type. Définissez une nouvelle classe sur le modèle de la précédente, avec notamment la définition d'une méthode pour la translation, ainsi que la redéfinition de la méthode `toString`. Confirmez à l'aide de tests qu'il n'est pas possible de mélanger des formes géométriques de types différents et que cela est bien détecté à la compilation.

#5.6 code

```
package geometrie;

import java.util.ArrayList;
public class CollectionFormesUniques <T extends Forme> extends ArrayList<T> {

    public void Translation(int x, int y) {
        for (T forme : this) {
            forme.Translation(x, y);
        }
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Collection de formes :\n");
        for (T forme : this) {
            sb.append("Forme n°"+i+": ").append(forme.toString()).ap-
pend("\n\n");
        }
        return sb.toString();
    }
}
```

```
Et si on teste
```

```
package geometrie;

import java.awt.*;
import java.util.*;
import java.util.List;

public class Test {
    public static void main(String[] args) {
        Point centre = new Point(10, 4);
        Color couleur = new Color(82, 255, 0);
        Cercle test1 = new Cercle(centre, couleur, 20.5);
        Rectangle test2 = new Rectangle(centre, couleur, 20.5, 4);
        Carre test3 = new Carre(centre, couleur, 20.5);
        Rectangle test4 = new Rectangle(centre, couleur, 20.5, 5);
```

```
        CollectionFormesUniques<Rectangle> collection = new CollectionFormesU-
niques();


        collection.add(test1);
        collection.add(test2);
        collection.add(test3);
        collection.add(test4);

        System.out.println(collection);


        collection.Translation(2, 3);


        System.out.println("Après translation : \n");
        System.out.println(collection);
    }
}
```

On obtient :
'add(geometrie.Rectangle)' in 'java.util.ArrayList' cannot be applied to '(geometrie.Cercle)' ca r en effet j'ai demandé une liste de rectangle or test1 est un cercle

Conclusion notre classe CollectionFormesUniques fonctionne correctement.