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

Nom : BENCHIKHI
Prénom : Anas

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

```
package et3.java.geometrie;
// package et3.java.geometrie;

import java.awt.Color;
import java.awt.Point;
import java.util.Objects;

public abstract class Forme {
    private Point centreGravite;
    private Color couleur;
```

```
    public Forme(Point centreGravite, Color couleur) {
        this.centreGravite = new Point(centreGravite);
        this.couleur = couleur;
    }

    public void translater(int dx, int dy) {
        centreGravite.translate(dx, dy);
    }

    public Point getCentreGravite() {
        return new Point(centreGravite);
    }

    public Color getCouleur() {
        return couleur;
    }

    @Override
    public String toString() {
        return String.format("[ centre de gravité : x =%d ; y
=%d]\n[ couleur : r =%d ; g =%d ; b =%d]",
                centreGravite.x, centreGravite.y,
couleur.getRed(), couleur.getGreen(), couleur.getBlue());
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return
false;
        Forme forme = (Forme) obj;
        return centreGravite.equals(forme.centreGravite) &&
couleur.equals(forme.couleur);
    }

    @Override
    public int hashCode() {
        return Objects.hash(centreGravite, couleur);
    }

    public abstract String getNom();
}
```

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

```
package et3.java.geometrie;
```

```
import java.awt.Color;
import java.awt.Point;

public class Carre extends Rectangle {

    public Carre(Point centreGravite, Color couleur, double cote)
{
        super(centreGravite, couleur, cote, cote);
    }

    @Override
    public String getNom() {
        return "Carré";
    }

    @Override
    public String toString() {
        return String.format("[ %s\n%s\n[ côté : %.2f]\n]",
getNom(), super.toString(), getLargeur());
    }
}
```

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

```
import et3.java.geometrie.Carre;
import et3.java.geometrie.Cercle;
import et3.java.geometrie.Forme;
import et3.java.geometrie.Rectangle;
import java.awt.Color;
import java.awt.Point;
import java.util.ArrayList;
import java.util.Iterator;

public class TestFormes {
    public static void main(String[] args) {
        // Création de la liste de formes
        ArrayList<Forme> formes = new ArrayList<>();

        // Ajout d'un Cercle
        Cercle cercle = new Cercle(new Point(10, 5), new
Color(255, 0, 0), 15.0);
        formes.add(cercle);
```

```
        // Ajout d'un Rectangle
        Rectangle rectangle = new Rectangle(new Point(20, 10), new
Color(0, 255, 0), 30.0, 20.0);
        formes.add(rectangle);

        // Ajout d'un Carré
        Carre carre = new Carre(new Point(5, 5), new Color(0, 0,
255), 10.0);
        formes.add(carre);

        // Affichage du contenu de la liste à l'aide d'un
itérateur
        System.out.println("Contenu de la liste des formes :");
        Iterator<Forme> it = formes.iterator();
        while (it.hasNext()) {
            Forme forme = it.next();
            System.out.println(forme);
        }
    }
}
```

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

Si on essaie de retirer un élément d'une collection à un indice qui n'existe pas (par exemple, un indice négatif ou un indice supérieur ou égal à la taille de la liste), une exception IndexOutOfBoundsException sera levée à l'exécution.

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

```
import et3.java.geometrie.Forme;
import java.util.ArrayList;

public class CollectionFormes extends ArrayList<Forme> {

    // Méthode pour translater toutes les formes de la collection
    public void translation(int dx, int dy) {
        for (Forme forme : this) {
            forme.translater(dx, dy);
```

```
        }
    }

    // Redéfinition de la méthode toString
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("Collection de
formes :\n");
        for (Forme forme : this) {
            sb.append(forme).append("\n");
        }
        return sb.toString();
    }
}
```

Tests :

```
import et3.java.geometrie.Carre;
import et3.java.geometrie.Cercle;
import et3.java.geometrie.Rectangle;
import java.awt.Color;
import java.awt.Point;

public class TestCollectionFormes {
    public static void main(String[] args) {
        // Création de la collection de formes
        CollectionFormes collection = new CollectionFormes();

        // Ajout d'un Cercle
        Cercle cercle = new Cercle(new Point(10, 5), new
Color(255, 0, 0), 15.0);
        collection.add(cercle);

        // Ajout d'un Rectangle
        Rectangle rectangle = new Rectangle(new Point(20, 10), new
Color(0, 255, 0), 30.0, 20.0);
        collection.add(rectangle);

        // Ajout d'un Carré
        Carre carre = new Carre(new Point(5, 5), new Color(0, 0,
255), 10.0);
        collection.add(carre);

        // Affichage initial de la collection
        System.out.println("Avant translation :");
        System.out.println(collection);

        // Translation de toutes les formes
        collection.translation(5, -3);

        // Affichage après translation
        System.out.println("Après translation :");
```

```
        System.out.println(collection);
    }
}
```

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

```
import et3.java.geometrie.Forme;
import java.util.ArrayList;

public class CollectionHomogene<T extends Forme> extends
ArrayList<T> {

    // Méthode pour translater toutes les formes de la collection
    public void translation(int dx, int dy) {
        for (T forme : this) {
            forme.translater(dx, dy);
        }
    }

    // Redéfinition de la méthode toString
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("Collection homogène
de formes :\n");
        for (T forme : this) {
            sb.append(forme).append("\n");
        }
        return sb.toString();
    }
}


Tests :


import et3.java.geometrie.Cercle;
import java.awt.Color;
import java.awt.Point;

public class TestCollectionHomogene {
    public static void main(String[] args) {
        // Création d'une collection homogène de cercles
        CollectionHomogene<Cercle> collectionCercles = new
CollectionHomogene<>();
```

```
        // Ajout de cercles
        Cercle cercle1 = new Cercle(new Point(10, 5), new
Color(255, 0, 0), 15.0);
        Cercle cercle2 = new Cercle(new Point(20, 10), new
Color(0, 255, 0), 10.0);
        collectionCercles.add(cercle1);
        collectionCercles.add(cercle2);

        // Affichage initial de la collection
        System.out.println("Collection de cercles avant
translation :");
        System.out.println(collectionCercles);

        // Translation de tous les cercles
        collectionCercles.translation(5, -3);

        // Affichage après translation
        System.out.println("Collection de cercles après
translation :");
        System.out.println(collectionCercles);

        // Tentative de mélange de types (décommenter pour tester)
        // CollectionHomogene<Cercle> collectionMixte = new
CollectionHomogene<>();
        // collectionMixte.add(new Rectangle(new Point(5, 5), new
Color(0, 0, 255), 10.0, 20.0)); // Erreur de compilation
    }
}
```