amU Faculté des sciences Aix Marseille Université

Année universitaire 2024-2025

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☐ Aix-Montperrin ☐ Aubagne-SATIS

Sujet de : ☒ 1er semestre ☐ 2ème semestre ☐ Session 2 Durée de l'épreuve : 2h

Examen de : L3 Nom du diplôme : Licence informatique : parcours math-info

Code du module : SIN5U34 Libellé du module : Initiation au Génie logiciel

Calculatrices autorisées : NON Documents autorisés : OUI, notes de Cours, supports de cours

---

1 Dessins de glyphes

Question 1 : Quelles parties du code doivent être modifiées pour ajouter un nouveau type de points en forme de croix ?

Afin de pouvoir ajouter un nouveau type de forme, il faut ajouter une nouvelle valeur possible pour le l'`enum` (par exemple `CROSS`) et rajouter la gestion du cas pour lequel l'`enum` est égal à cette nouvelle valeur dans la méthode `draw`. La modification correspond au code suivant :

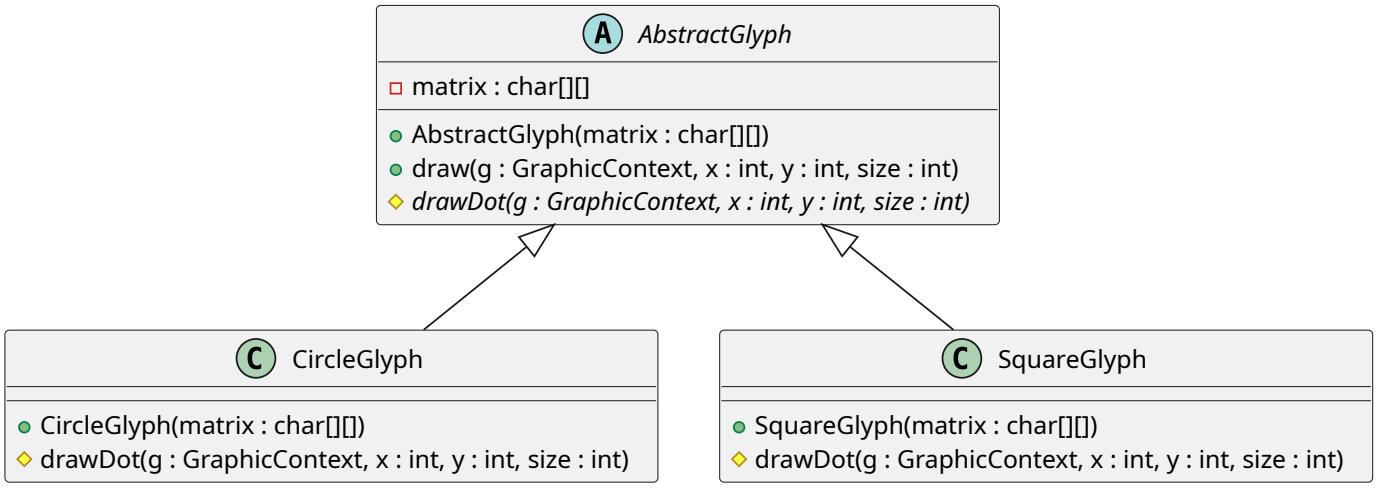
```
public class Glyph {
    public enum DotType {SQUARE, CIRCLE, CROSS} // ajout de la valeur CROSS

    private void drawDot(GraphicsContext g, int x, int y, int size) {
        switch (type) {
            case SQUARE -> drawSquare(g, x, y, size);
            case CIRCLE -> drawCircle(g, x, y, size);
            case CROSS -> drawCross(g,x,y,size); // ajout d'un cas CROSS
        }
    }

    // Ajout d'une méthode pour dessiner des croix
    private void drawCross(GraphicsContext g, int x, int y, int size) {
        // code du dessin d'une croix
    }
```

Question 2 : Parmi les 5 principes SOLID, lequel est violé ? Justifiez.

Le principe qui est violé est le principe OCP (Open/Closed Principle) qui dit qu'une classe doit être ouverte à l'extension, mais fermé à la modification interne. Pour que la classe `Glyph` respecte *OCP* pour l'ajout d'un nouveau type de *dot*, il faudrait qu'on puisse rajouter le nouveau type sans modifier le code existant de `Glyph` et simplement en ajoutant des classes pouvant potentiellement étendre la classe `Glyph`. Dans l'état actuel du code, ce n'est pas possible.

Question 3 : Donnez le diagramme de classes d'une nouvelle organisation du code (sans ajouter pour le moment de classes pour le dessin des croix) qui respecte les principes SOLID en utilisant le patron de conception "Patron de méthode" (*template method*).

L'application du patron de conception *template method* consiste à créer une classe abstraite `AbstractGlyph` qui aura quasiment tout le code de `Glyph` sauf le dessin du *dot* qui sera mis dans une méthode abstraite `drawDot`. Cette classe sera étendue par deux classes `CircleGlyph` et `SquareGlyph` qui implémenteront `drawDot`.



Question 4 : Implémentez le diagramme de classes que vous avez proposé à la question 3 et réécrivez en conséquence le code de la méthode `paint` de la classe `GlyphApplication` donnée en exemple.

Le code des trois classes `AbstractGlyph`, `CircleGlyph` et `SquareGlyph` est le suivant :

```
public abstract class AbstractGlyph {
    private final char[][] matrix;

    public AbstractGlyph(char[][] matrix) {
        this.matrix = matrix;
    }

    public void draw(GraphicsContext g, int x, int y, int size) {
        int n = matrix.length;
        int ds = size / n;
        for (int px = 0; px < n; px++)
            for (int py = 0; py < n; py++)
                if (matrix[py][px] == '#')
                    drawDot(g, x + px * ds + 1, y + py * ds + 1, ds - 1);
    }

    protected abstract void drawDot(GraphicsContext g, int x, int y, int size);
}
```

```
public class CircleGlyph extends AbstractGlyph{
    public CircleGlyph(char[][] matrix) {
        super(matrix);
    }

    protected void drawDot(GraphicsContext g, int x, int y, int size) {
        g.fillOval(x, y, size, size);
    }
}
```

```
public class SquareGlyph extends AbstractGlyph{
    public SquareGlyph(char[][] matrix) {
        super(matrix);
    }
```

```
    protected void drawDot(GraphicsContext g, int x, int y, int size) {
        g.fillRect(x + 1, y + 1, size - 1, size - 1);
    }
}
```

Le nouveau code de la méthode `paint` est le suivant :

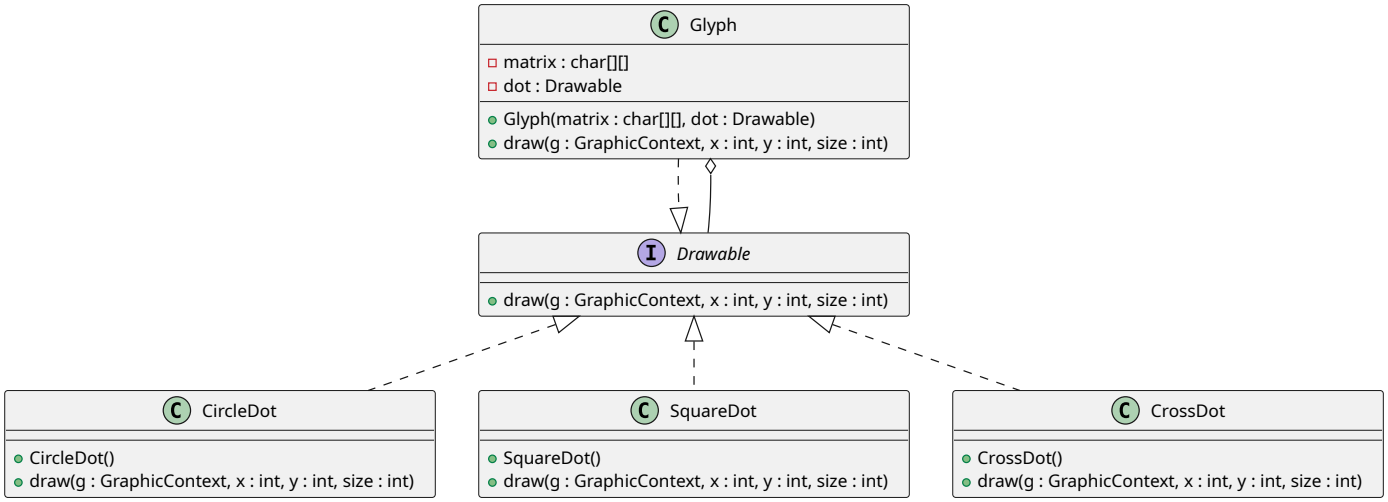
```
    private void paint(GraphicsContext graphicsContext) {
        graphicsContext.setFill(Color.BLACK);
        graphicsContext.setLineWidth(2);
        AbstractGlyph glyph2 = new CrossGlyph(c2);
        AbstractGlyph glyph3 = new SquareGlyph(c3);
        glyph2.draw(graphicsContext, 0, 0, 400);
        glyph3.draw(graphicsContext, 440, 0, 400);
    }
```

Question 5 : écrivez la classe permettant de dessiner les glyphes avec les points en forme de croix évoqués à la question 1. Vous utiliserez les deux méthodes de `GraphicsContext` ci-dessous :

Il suffit de rajouter une classe `CrossGlyph` qui étend `AbstractGlyph` de la façon suivante :

```
public class CrossGlyph extends AbstractGlyph {
    public CrossGlyph(char[][] matrix) {
        super(matrix);
    }

    protected void drawDot(GraphicsContext g, int x, int y, int size) {
        g.strokeLine(x,y,x+size,y+size);
        g.strokeLine(x,y+size,x+size,y);
        g.strokeRect(x,y,size,size);
    }
}
```

Question 6 : On souhaite modifier le code que la méthode `paint` ci-dessous produise l'affichage ci-après. Donnez le diagramme de classes d'une nouvelle organisation du code correspondant à cette modification. Les classes ou interfaces `Drawable`, `CircleDot`, `SquareDot`, `CrossDot` et `Glyph` devront donc apparaître dans votre diagramme.



Question 7 : Implémentez le diagramme de classes que vous avez proposé à la question 6.

```java
public interface Drawable {
    void draw(GraphicsContext g, int x, int y, int size);
}
```

```java
public class Glyph implements Drawable{
    private final Drawable dot;
    private final char[][] matrix;

    public Glyph(char[][] matrix, Drawable dot) {
        this.matrix = matrix;
        this.dot = dot;
    }

    public void draw(GraphicsContext g, int x, int y, int size) {
        int n = matrix.length;
        int ds = size / n;
        for (int px = 0; px < n; px++)
            for (int py = 0; py < n; py++)
                if (matrix[py][px] == '#')
                    dot.draw(g, x + px * ds + 1, y + py * ds + 1, ds - 1);
    }
}
```

```java
public class CrossDot implements Drawable{
    public void draw(GraphicsContext g, int x, int y, int size) {
        g.strokeLine(x,y,x+size,y+size);
        g.strokeLine(x,y+size,x+size,y);
        g.strokeRect(x,y,size,size);
    }
}
```

```java
public class CircleDot implements Drawable{
    public void draw(GraphicsContext g, int x, int y, int size) {
        g.fillOval(x, y, size, size);
    }
}
```

```java
public class SquareDot implements Drawable{
    public void draw(GraphicsContext g, int x, int y, int size) {
        g.fillRect(x + 1, y + 1, size - 1, size - 1);
    }
}
```