amU Faculté des sciences Aix Marseille Université

Année universitaire 2025-2026

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

---

Le barème est indiqué entre parenthèses pour chaque question. Il est indicatif et provisoire, mais vous donne une idée du temps à consacrer à chaque question.

1 Dessins de formes (12,5 points)

On considère le code suivant simulant (on fera juste des affichages de textes via des `println`) un dessin de formes géométriques simples entre deux points dans un plan 2D.

```
public record Point2D(int x, int y) {
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
    public double distanceTo(Point2D other) {
        int dx = other.x - this.x;
        int dy = other.y - this.y;
        return Math.sqrt(dx * dx + dy * dy);
    }
}

public class LinePainter {
    public void paint(Point2D p1, Point2D p2) {
        System.out.println("Paint line from point " + p1 + " to point " +  p2);
    }
}

public class RectanglePainter {
    public void paint(Point2D topLeftCorner, int width, int height) {
        System.out.println("Paint rectangle with top left corner "
            + topLeftCorner + ", width: " + width + ", height: " + height);
    }
}
```

```
public record Shape(Point2D p1, Point2D p2, Object shapePainter) {
    public void draw() {
        if (shapePainter instanceof LinePainter linePainter) {
            linePainter.paint(p1, p2);
        } else
        if (shapePainter instanceof RectanglePainter rectanglePainter) {
            Point2D topLeftCorner = new Point2D(
                    Math.min(p1.x(), p2.x()),
                    Math.min(p1.y(), p2.y()));
            int width = Math.abs(p2.x() - p1.x());
            int height = Math.abs(p2.y() - p1.y());
            rectanglePainter.paint(topLeftCorner, width, height);
        } else {
            throw new IllegalArgumentException("Unknown shape painter");
        }
    }
}

public class Main {
    static void main() {
        Point2D p1 = new Point2D(10, 20);
        Point2D p2 = new Point2D(30, 5);
        Shape[] shapes = new Shape[]{
            new Shape(p1, p2, new LinePainter()),
            new Shape(p1, p2, new RectanglePainter())
        };

        for (Shape shape : shapes) {
            shape.draw();
        }
    }
}
```

Question 1 (2 points) : Quelle est la sortie produite par l'exécution de la méthode `main` de la classe `Main` ? Justifiez votre réponse.

On souhaite maintenant ajouter une nouvelle forme géométrique, un cercle en utilisant le premier point comme centre du cercle et la distance entre les deux points comme rayon du cercle. On ajoute donc la classe `CirclePainter` suivante.

```
public class CirclePainter {
    public void paint(Point2D center, int radius) {
        System.out.println("Paint circle with center "
            + center + " and radius : " + radius);
    }
}
```

Question 2 (2 points) : On souhaite utiliser cette nouvelle classe `CirclePainter` dans le code existant pour définir et dessiner des cercles avec la classe `Shape`. Quel(s) changement(s) faut-il apporter au code existant pour permettre cela ? Expliquez en détail les modifications nécessaires. Quel(s) principe(s) SOLID sont-ils violés par cette approche ?

Question 3 (4 points) : On souhaite réorganiser le code pour permettre d'ajouter de nouvelles formes géométriques sans avoir à modifier pour chaque ajout la classe `Shape`. Proposez sous forme d'un diagramme UML une nouvelle organisation qui modifie la classe `Shape` et ajoute des classes ou interfaces, mais qui ne modifie pas les classes `Point2D`, `LinePainter` et `RectanglePainter`.

Question 4 (4,5 points) : Donnez le code Java complet (en vous basant sur votre diagramme UML de la question précédente) permettant de dessiner les trois formes géométriques (ligne, rectangle et cercle) en utilisant la nouvelle organisation proposée. Puisque les classes `Point2D`, `LinePainter` et `RectanglePainter` ne doivent pas être modifiées, vous n'avez pas besoin de donner leur code.

2 Ventilateur de plafond (7,5 points)

On considère le code Java suivant simulant un ventilateur de plafond avec deux vitesses différentes. L'interface de contrôle est une chaîne à tirer (`pull`) qui permet de changer la vitesse du ventilateur à chaque tirage de la chaîne. Pour le moment, le ventilateur a trois vitesses possibles : éteint, vitesse basse (*low speed*) et vitesse moyenne (*medium speed*). À chaque tirage de la chaîne, le ventilateur passe à l'état suivant dans l'ordre : éteint → vitesse basse → vitesse moyenne → éteint.

```
public class CeilingFanPullChain {
    enum FanSpeed {
        OFF,
        LOW_SPEED,
        MEDIUM_SPEED
    }
    private FanSpeed fanSpeed;

    public CeilingFanPullChain() {
        fanSpeed = FanSpeed.OFF;
    }

    public void printSpeed() {
        if (fanSpeed == FanSpeed.OFF) {
            System.out.println("off");
        } else if (fanSpeed == FanSpeed.LOW_SPEED) {
            System.out.println("low speed");
        } else if (fanSpeed == FanSpeed.MEDIUM_SPEED) {
            System.out.println("medium speed");
        }
    }

    public void pullChain() {
        if (fanSpeed == FanSpeed.OFF) {
            fanSpeed = FanSpeed.LOW_SPEED;
        } else if (fanSpeed == FanSpeed.LOW_SPEED) {
            fanSpeed = FanSpeed.MEDIUM_SPEED;
        } else if (fanSpeed == FanSpeed.MEDIUM_SPEED) {
            fanSpeed = FanSpeed.OFF;
        }
    }
}
```

Question 5 (1,5 points) : Supposons qu'on veuille ajouter une nouvelle vitesse correspondant à une vitesse élevée (*high speed*) pour le ventilateur. Quelles modifications faut-il apporter à la classe `CeilingFanPullChain` pour permettre cela ? Expliquez en détails les changements nécessaires. Quel(s) principe(s) SOLID sont-ils violés par cette approche ?

Question 6 (1,5 points) : Quel patron de conception (*design pattern*) pourrait être utilisé pour réorganiser le code afin de permettre l'ajout de nouvelles vitesses au ventilateur sans avoir à modifier la classe `CeilingFanPullChain` pour chaque ajout ?

Question 7 (2,5 points) : Proposez une nouvelle organisation du code sous forme d'un diagramme UML utilisant le patron de conception de la question précédente.

Question 8 (2 points) : Donnez le code Java complet (en vous basant sur votre diagramme UML de la question précédente) permettant d'implémenter un ventilateur de plafond avec quatre vitesses : éteint, basse, moyenne et élevée.