CSC_3TC36_TP – Programmation Orientée Objet en Java

Contrôle de connaissances (1h30)

28 avril 2025

Nom :
Prénom :
Groupe :

- Le sujet comporte 14 pages.
- Une seule feuille de papier A4 recto-verso de notes est autorisée. Téléphones et ordinateurs éteints.
- Le barème est donné à titre indicatif. Il pourra être revu si le correcteur l'estime nécessaire.
- Les réponses aux questions s'écrivent dans les boîtes prévues à cet effet. La taille des boîtes de réponses n'est pas forcément proportionnelle à la taille des réponses attendues.
- Vous pouvez répondre en français ou en anglais selon votre préférence.
- Si vous constatez une erreur dans l'énoncé, écrivez comment vous le corrigez et continuez votre rédaction.

1. (1 point) Dans le paradigme Orienté Objet (OO), des objets peuvent communiquer entre eux en s'échangeant des données tout en agissant les uns sur les autres. Décrire brièvement comment cette communication est réalisée en Java.

Les classes déclarent des attributs de type référence qui permettent d'appeler des méthodes sur les objets référencés.

2. (2 points) Expliquer la différence entre une variable de classe et d'instance. Quelle est la manière de déclarer ces deux types de variables en Java.

Une variable de classe possède une seule valeur en mémoire pour toutes les instances de la classe tandis que pour une variable d'instance, il y a une valeur par instance.

La déclaration de la variable de classe emploie le mot clé `static`.

3. (2 points)

3.1 (1 point) Un principe important de l'OO est l'encapsulation des données. Quelle notion en Java est utilisée pour permettre une telle encapsulation ? Quel mot clé est utilisé pour encapsuler les données d'une classe ?

La visibilité des attributs.

`private`

3.2 (1 point) On parle souvent de la transparence de mise en œuvre comme objectif de l'encapsulation des données. Expliquer brièvement en quoi cela consiste en donnant un exemple.

Cela permet à un aux objets qui manipulent les données des autres objets de le faire uniquement par des méthodes sans avoir à connaître leur représentation dans la classe. Voir exemple de la classe `Vector` du transparent 24 du cours de la TH4.

4. (1 point) Donner et expliquer brièvement deux usages des interfaces de programmation.

- Outil de conception
- Contrat
- Encapsulation
- Descriptif de propriétés

5. (3 points) On vous donne les définitions suivantes :

```
public interface Shape {

    String getName();

    double getArea();
}

public abstract class AbstractShape implements Shape {

    private String name;

    public AbstractShape(String name) {
        this.name = name;
    }

    @Override
    public String getName() {
        return name;
    }
}

public class Circle extends AbstractShape {

    private final double radius;

    public Circle(String name,
                  double radius) {
        super(name);

        this.radius = radius;
    }


     @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }
}

public class Rectangle extends AbstractShape {

    private final double width;

    private final double height;

    public Rectangle(String name,
                     double width,
                     double height) {
        super(name);

        this.width = width;
        this.height = height;
    }

    @Override
    public double getArea() {
        return width * height;
    }
}
```

5.1 (1 point) Parmi les expressions suivantes numérotées de 1 à 5, quelles sont celles qui ne compilent pas ?

```
1)  Shape shape1 = new AbstractShape();

2)  Shape shape = new Circle("Circle1", 12);

3)  Circle circle = new Shape();

4)  Rectangle rect = new Circle("Circle2", 2);
```

```
5)  AbstractShape shape2 = new Rectangle("rect1", 12, 15);
```

1, 3, et 4

5.2 (1 point) Pour chaque itération de la boucle **for** du code suivant, expliquer comment sera déterminée la classe de la méthode `getArea()` qui sera appelée.

```
List<Shape> shapes = new ArrayList<Shape>();
  shapes.add(new Rectangle("rect1", 12, 15));
  shapes.add(new Rectangle("rect2", 2, 6));
  shapes.add(new Circle("Circle1", 12));

  double totalArea = 0.0;

  for (Shape shape : shapes) {
      totalArea += shape.getArea();
  }

  System.out.println("The total area is: " + totalArea);
```

Elle sera déterminée par liaison dynamique, c'est-à-dire qu'à l'exécution du programme, la première méthode trouvée dans l'arbre d'héritage de la classe d'instanciation sera exécutée.

5.3 (1 point) Quel concept essentiel de l'OO en lien aux types des objets cet exemple met-il en œuvre ?

Le polymorphisme.

6. (2 points) Un collègue peu soucieux de la bonne gestion des exceptions a développé le code de la méthode `readLines` suivante :

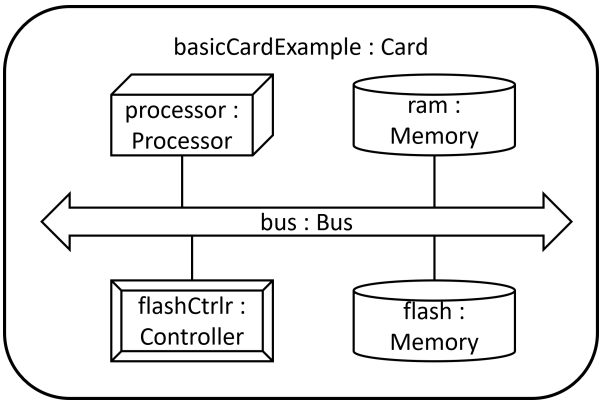
```
public List<String> readLines(String fileName,
                              double expectedLineLength) {
    try {
        List<String> lines = Files.readAllLines(Paths.get(fileName));

          for (String fileLine : lines) {
              if (fileLine.length() != expectedLineLength) {
                  throw new Exception("Invalid line length: " + fileLine.length());
                }
          }

          return lines;
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }

    return null;
}
```

Cette méthode doit lire et retourner une liste contenant toutes les lignes d'un fichier et lever une exception si l'une de ces lignes n'est pas de la longueur attendue. Sachant que `Files.readAllLines` est susceptible de lever l'exception `IOException`, proposez une meilleure implémentation de la méthode `readLines` de votre collègue afin que lors d'un appel à `readLines`, il soit possible de récupérer l'exception levée et de la traiter correctement.

```
public List<String> readLines(String fileName,
                                    int expectedLineWidth)
throws IOException, InvalidLineException {
    List<String> lines = Files.readAllLines(Paths.get(fileName));

    for (String fileLine : lines) {
          if (fileLine.length() != expectedLineWidth) {
              throw new InvalidLineException("Invalid line lenth: " + fileLine.length(),
lines.indexOf(fileLine));
           }
    }

    return lines;
}

public class InvalidLineException extends Exception {

    private final int lineNumber;

    public InvalidLineException(String message,
                                int lineNumber) {
          super(message);

          this.lineNumber = lineNumber;
    }

    public int getLineNumber() {
          return lineNumber;
    }
}
```

7. **Exercice de modélisation** (9 points)

Un professeur du département COMELEC de Télécom Paris ne connaissant pas l'OO veut développer un simulateur de **cartes électroniques** simples comme l'exemple de la carte illustrée par le diagramme suivant :



Afin de l'aider à développer son simulateur, programmer les classes servant à modéliser ces cartes électroniques. Votre modèle doit pouvoir caractériser les aspects suivants :

1. Faire l'hypothèse que la carte ne peut contenir que des **processeurs**, des **bus**, des **mémoires** et des **contrôleurs** de mémoire.
2. Certains composants tels que les processeurs, les mémoires et les contrôleurs peuvent se **connecter** entre eux via un ou plusieurs bus (voir le diagramme). Une carte est aussi un composant électronique mais elle ne peut pas être connectée à un ou des bus.
3. Chaque composant possède un fabriquant identifié par une chaîne de caractère.
4. Il faut modéliser la puissance électrique consommée par ces composants. On suppose que cette puissance consommée est constante dans le temps pour chaque composant (puissance statique).
5. On doit également pouvoir connaître la puissance consommée par une carte. Puisque la puissance des composants est constante dans le temps, il suffit de faire la somme de la puissance consommée de chaque composant de la carte pour connaître sa consommation. Par ailleurs, une carte ne possède pas de consommation intrinsèque, c'est-à-dire qu'elle ne consomme que la somme de la consommation de ses composants.

Pour la suite de l'exercice, concentrez-vous sur les concepts et l'application des bonnes pratiques de modélisation. La syntaxe est secondaire.

7.1 (6 points) Programmer les classes dont vous avez besoin pour modéliser les cartes électroniques en prenant en compte les hypothèses décrites précédemment.

```
public abstract class Component {

    private final String manufacturer;
    private final double power;

    protected Component(final String manufacturer,
                        final double power ) {
        this.manufacturer = manufacturer;
          this.power = power;
    }

    public double getPower() {
          return power;
    }

    public String getManufacturer() {
        return manufacturer;
    }
}

public class Bus extends Component {

    public Bus(String manufacturer,
               double power ) {
        super(manufacturer, power);
    }
}

public abstract class ConnectableComponent extends Component {

    private final Set<Bus> buses;

    protected ConnectableComponent(String manufacturer,
                                   double power) {
          super(manufacturer, power);

          buses = new HashSet<Bus>();
    }

    public boolean connect(Bus bus) {
          return buses.add(bus);
    }

    public boolean disconnect(Bus bus) {
          return buses.remove(bus);
    }
}

public class Controler extends ConnectableComponent {

    public Controler(String manufacturer,
                     double power) {
        super(manufacturer, power);
    }
}

public class Memory extends ConnectableComponent {

    public Memory(String manufacturer,
                  double power) {
        super(manufacturer, power);
```

```java
public class ElectronicCard extends Component {

    private final Set<Component> components;

    public Card(String manufacturer) {
          super(manufacturer, 0);

          components = new HashSet<Component>();
    }

    public boolean addComponent(Component component) {
          return components.add(component);
    }

    @Override
    public double getPower() {
          double totalPower = super.getPower();

          for (Component component : components) {
              totalPower += component.getPower();
          }

          return totalPower;
    }
}
```

7.2 (3 points) Fournir le code de création d'une carte électronique telle que celle du diagramme précédent. La carte est fabriquée par Tartempion SA. Elle possède un processeur, une mémoire RAM, une mémoire Flash et un contrôleur de mémoire Flash. Ces quatre composants sont reliés par un seul bus. On suppose que chaque composant est fabriqué par ST Microelectronics et consomme 5.5 mW. Votre code devra afficher la puissance consommée totale de la carte à la console.

```
ElectronicCard card = new ElectronicCard("Tartempion SA");

String componentManufacturer = "ST Microelectronics";
double componentPower = 5.5;

Bus bus = new Bus(componentManufacturer, componentPower);
card.addComponent(bus);

Memory flash = new Memory(componentManufacturer, componentPower);
card.addComponent flash);
flash.connect(bus);

Memory ram = new Memory(componentManufacturer, componentPower);
card.addComponent(ram);
ram.connect(bus);

Controler controler = new Controler(componentManufacturer, componentPower);
card.addComponent(controler);
controler.connect(bus);

Processor processor = new Processor(componentManufacturer, componentPower);
card.addComponent(processor);
processor.connect(bus);

System.out.println(The " + card.getManufacturer() + " card consumes " + card.getPower() + " mW.");
```

FIN DU SUJET