amU Faculté des sciences Aix Marseille Université

Année 2025-26

Domaine Sciences et Technologies
Licence Informatique : math-info

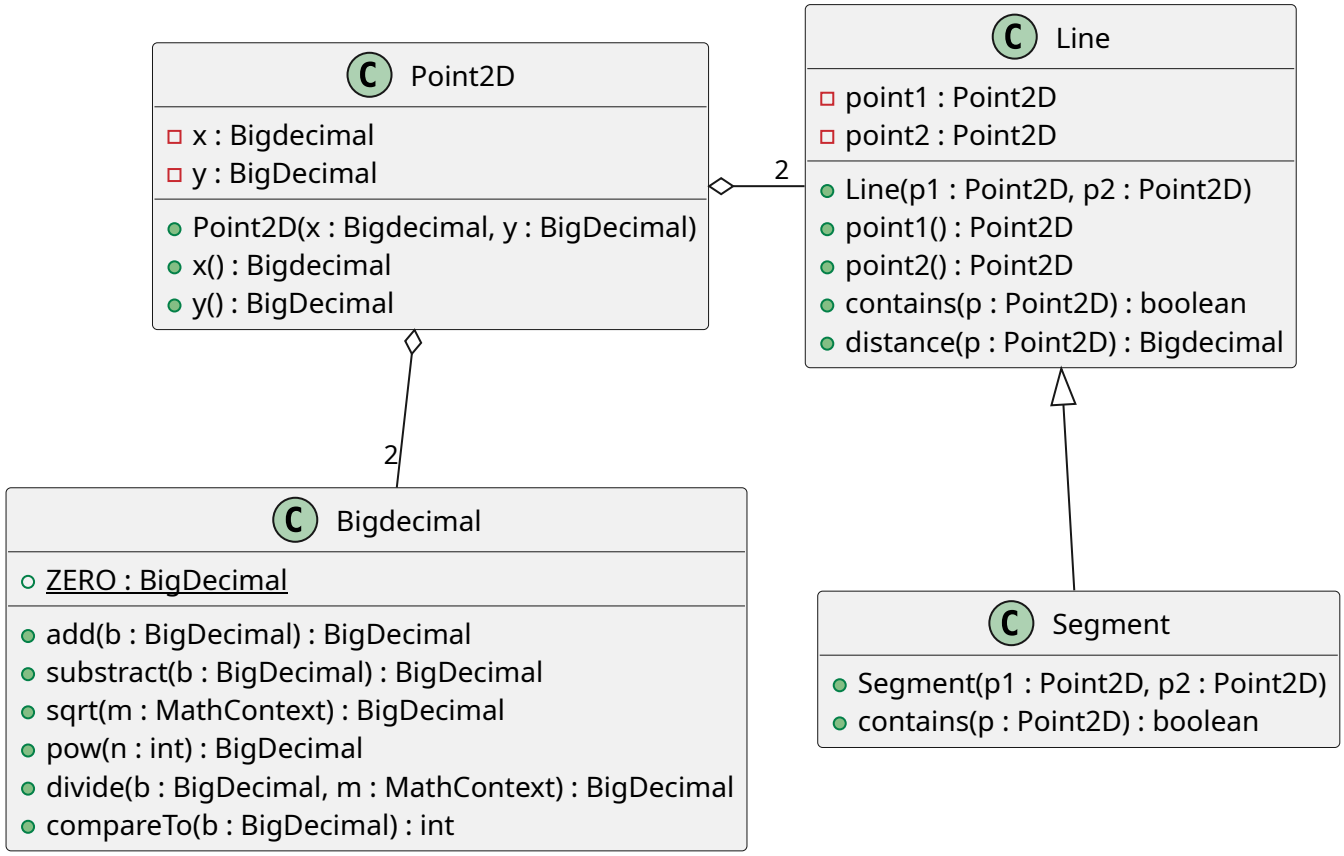
Initiation au Génie logiciel : TD 6

Codes UE : SIN5U34

Lignes et tchat

1 Lignes et segments

On considère les trois classes suivantes : `Point2D`, `line` et `Segment` qui forment le diagramme de classe suivant :



La méthode `distance` de `Line` calcule la distance du `point` à la droite définie par les deux points `point1` et `point2` passés en paramètres du constructeur. Les points sur la droite sont les points à distance zéro de la droite. Les coordonnées des points sont codés sous la forme de `BigDecimal` qui correspondent à des nombres décimaux signés de précision arbitraire. La classe `BigDecimal` fournit des opérations d'arithmétique et de comparaison.

Le code de la classe `Point2D` est le suivant :

```
public record Point2D(BigDecimal x, BigDecimal y) {}
```

Le code de la classe `Line` est le suivant :

```
public class Line {
  private final Point2D point1;
  private final Point2D point2;
  public Line(Point2D point1, Point2D point2) {
```

```
    this.point1 = point1;
    this.point2 = point2;
  }
  public BigDecimal distance(Point2D point) {
    // Context d'opération avec 10000 chiffres significatifs
    MathContext mathContext = new MathContext(10000);
    BigDecimal x2MinusX1 = point2.x().subtract(point1.x());
    BigDecimal y2MinusY1 = point2.y().subtract(point1.y());
    BigDecimal x1MinusX0 = point1.x().subtract(point.x());
    BigDecimal y1MinusY0 = point1.y().subtract(point.y());
    BigDecimal resultNumerator =
        (x2MinusX1.multiply(y1MinusY0)).subtract(x1MinusX0.multiply(y2MinusY1));
    BigDecimal resultDenominator =
        (x2MinusX1.pow(2).add(y2MinusY1.pow(2))).sqrt(mathContext);
    return (resultNumerator.divide(resultDenominator, mathContext)).abs();
  }
  public boolean contains(Point2D point) { return distance(point).compareTo(BigDecimal.ZERO) == 0; }
  public Point2D point1() { return point1; }
  public Point2D point2() { return point2; }
}
```

Le code de la classe `Segment` est le suivant :

```
public class Segment extends Line {
  public Segment(Point2D point1, Point2D point2) {
    super(point1, point2);
  }
  public boolean contains(Point2D point) {
    BigDecimal minX = point1().x().min(point2().x());
    BigDecimal maxX = point1().x().max(point2().x());
    BigDecimal minY = point1().y().min(point2().y());
    BigDecimal maxY = point1().y().max(point2().y());
    return super.contains(point)
            && point.x().compareTo(minX) >= 0 && point.x().compareTo(maxX) <= 0
            && point.y().compareTo(minY) >= 0 && point.y().compareTo(maxY) <= 0;
  }
}
```

Question 1 : Qu'affiche le code suivant ?

```
public class App {
    public static void main(String[] args) {
        BigDecimal five = new BigDecimal("5");
        BigDecimal ten = new BigDecimal("10");
        BigDecimal fifteen = new BigDecimal("15");
        BigDecimal twenty = new BigDecimal("20");
        BigDecimal thirty = new BigDecimal("30");
```

```
        Point2D p1 = new Point2D(ten, ten);
        Point2D p2 = new Point2D(twenty, twenty);
        Line line = new Line(p1, p2);
        Line segment = new Segment(p1, p2);
        System.out.println(line.contains(new Point2D(fifteen, fifteen)));
        System.out.println(line.contains(new Point2D(thirty, thirty)));
        System.out.println(line.contains(new Point2D(five, thirty)));
        System.out.println(segment.contains(new Point2D(thirty, thirty)));
    }
}
```

Question 2 : Quel principe SOLID est violé par les classes données ci-dessus ? Justifiez votre réponse.

Question 3 : Donnez le diagramme de classes d'une nouvelle organisation du code qui corrige ce problème. Vous devez faire en sorte de ne pas dupliquer le code de la méthode distance. On doit également pouvoir écrire le code suivant qui devra avoir le même comportement (même affichage) que le code de la Question 1 :

```
public class App {
    public static void main(String[] args) {
        BigDecimal five = new BigDecimal("5");
        BigDecimal ten = new BigDecimal("10");
        BigDecimal fifteen = new BigDecimal("15");
        BigDecimal twenty = new BigDecimal("20");
        BigDecimal thirty = new BigDecimal("30");
        Point2D p1 = new Point2D(ten, ten);
        Point2D p2 = new Point2D(twenty, twenty);
        Shape line = new Line(p1, p2);
        Shape segment = new Segment(p1, p2);
        System.out.println(line.contains(new Point2D(fifteen, fifteen)));
        System.out.println(line.contains(new Point2D(thirty, thirty)));
        System.out.println(line.contains(new Point2D(five, thirty)));
        System.out.println(segment.contains(new Point2D(thirty, thirty)));
    }
}
```

Question 4 : Donnez le code des classes et interfaces présentes dans le diagramme de classes que vous avez donné à la question précédente.

Question 5 : La nouvelle organisation du code permet-elle d'écrire le code de la question 1 ? Est-ce que c'est satisfaisant ? Justifiez votre réponse.

2 Tchat

Considérons la classe `User` d'une messagerie instantanée qui représente un utilisateur (une connexion). La méthode `executeCommand` est appelée à chaque nouvelle ligne envoyée par l'utilisateur au serveur. Chaque ligne commence par le nom d'une commande à exécuter.

```java
public class User {
    private Server server;
    private final boolean isAdmin;
    private String userName;
    public User(String userName, boolean isAdmin) {
        this.userName = userName;
        this.server = null;
        this.isAdmin = isAdmin;
    }
    public void executeCommand(String commandLine) {
        if(server == null) {
            throw new IllegalStateException("Server not initialized");
        }
        String[] tokens = commandLine.split(" ");
        String commandName = tokens[0];
        String commandArgument = tokens[1];
        switch (commandName) {
            case "broadcast" -> runBroadcastCommand(commandArgument);
            case "set_username" -> runSetUserNameCommand(commandArgument);
            case "kick" -> {if (isAdmin) runKickCommand(commandArgument);}
            default -> System.out.println("Unknown command: " + commandName);
        }
    }
    public void setUserName(String userName) { this.userName = userName; }
    public String username() { return userName; }
    private void runSetUserNameCommand(String commandArgument) {
        if (server.usernameExists(commandArgument)) return;
        this.setUserName(commandArgument);
    }
    private void runBroadcastCommand(String commandArgument) {
        server.broadcastMessage(this, commandArgument);
    }
    private void runKickCommand(String commandArgument) {
        User kickedUser = server.getUserByName(commandArgument);
        if (kickedUser==null) return;
        server.closeConnection(kickedUser);
    }
    public void setServer(Server server) { this.server = server; }
}
```

La classe `User` utilise l'interface `Server` suivante :

```java
/**
 * Represents a server interface that provides methods to manage users and broadcast
 * messages.
 */
public interface Server {
```

```
    /**
     * Checks if a username already exists among the connected users.
     *
     * @param username The username to check.
     * @return {@code true} if the username exists, {@code false} otherwise.
     */
    boolean usernameExists(String username);
    /**
     * Broadcasts a message from the sender to all connected users.
     *
     * @param sender  The user sending the message.
     * @param message The message to broadcast.
     */
    void broadcastMessage(User user, String message);
    /**
     * Retrieves a user object by their username.
     *
     * @param username the username of the user to retrieve
     * @return the {@link User} object if the user exists, or {@code null} if not found
     */
    User getUserByName(String username);
    /**
     * Removes a user from the server and terminates their connection.
     *
     * @param kickedUser The user to disconnect from the server.
     */
    void closeConnection(User kickedUser);
    /**
     * Adds a user to the server and establishes a connection.
     *
     * @param user The user to connect to the server.
     */
    void connectUser(User user);
}
```

Pour tester la classe `User`, on utilise le code suivant :

```
public class MainAppServer {
    public static void main(String[] args) {
        Server server = new ServerSimulator();
        User david = new User("david", true);
        server.connectUser(david);
        User jeanMichel = new User("jean-michel", false);
        server.connectUser(jeanMichel);
        david.executeCommand("broadcast message1");
        jeanMichel.executeCommand("kick david");
        david.executeCommand("kick jean-michel");
```

```
        david.executeCommand("set_username jean");
        david.executeCommand("broadcast message2");
    }
}
```

Question 6 : Écrivez une classe `ServerSimulator` implémentant l'interface `Server` de sorte que l'exécution du code ci-dessus donne l'affichage ci-dessous. La classe `ServerSimulator` aura un seul attribut correspondant à la liste des utilisateurs connectés.

```
User connected: david
User connected: jean-michel
Message sent by david to david: message1
Message sent by david to jean-michel: message1
User disconnected: jean-michel
Message sent by jean to jean: message2
```

Question 7 : Que doit-on modifier dans la classe `User` pour ajouter une nouvelle commande ? Quels principes SOLID sont violés par le code donné en début d'exercice ? Justifiez votre réponse.

Question 8 : Écrivez le code d'une interface `Command` qui correspond à une commande.

Question 9 : Donnez un diagramme de classes d'une nouvelle organisation du code qui corrige les défauts de conception de la classe `User`. On doit pouvoir écrire le code suivant afin de construire une instance de la classe `User` qui peut traiter les commandes `set_username` et `broadcast` :

```
User jeanMichel = new User("jean-michel");
jeanMichel.addCommand(new SetUsernameCommand());
jeanMichel.addCommand(new BroadcastCommand());
```

Question 10 : Quelle structure de données peut être utilisée pour retrouver facilement une instance qui implémente `Command` à partir du nom de la commande ?

Question 11 : Donnez le code des classes présentes dans le diagramme de classes que vous avez donné à la question 10.

Question 12 : Donnez le code d'une classe `UserBuilder` qui permet de construire un utilisateur non-administrateur ne supportant que les commandes `set_username` et `broadcast` de la façon suivante :

```
User jeanMichel = new UserBuilder().setName("jean-michel").build();
```

et un utilisateur administrateur supportant les commandes `set_username`, `broadcast` et `kick` de la façon suivante :

```
User david = new UserBuilder().setName("david").enableAdminCommands().build();
```