Vendredi 17 mars 2023

IUT de Paris - Rives de Seine
Université Paris Cité

# Compte Rendu Projet JAVA

# CRAZY CIRCUS GAME

Gobigan MATHIALAHAN
Eric ZHANG
Groupe: 110 & 112

SOMMAIRE

# INTRODUCTION DU PROJET

Crazy Circus est un jeu de société qui met au défi les joueurs de trouver la séquence d'ordres la plus rapide pour exécuter les instructions pour trois animaux sur le podium rouge ou bleu. Ce jeu est un excellent moyen pour tester votre capacité de réflexion et de stratégie, et de combiner la logique, la mémoire et la rapidité.

Le jeu se compose de 24 cartes, chacune représentant une situation initiale différente, ainsi qu'une situation finale que les joueurs doivent atteindre. Les joueurs devront résoudre chaque situation en utilisant leurs compétences de résolution de problèmes pour trouver la meilleure séquence d'ordres.

Le projet consistait en la création d'un exécutable pour le jeu, développé et testé avec succès à l'aide de l'IDE IntelliJ IDEA. Avec une équipe de deux étudiants en informatique, le délai pour accomplir ce projet était de trois semaines, du 23 février au 17 mars. Les résultats ont été satisfaisants, l'application répondant parfaitement aux exigences du cahier des charges du projet.

Le jeu peut être joué individuellement ou en équipe, et est adapté à tous les âges. Avec ses défis passionnants, Crazy Circus promet des heures de plaisir pour tous les amateurs de jeux de société.

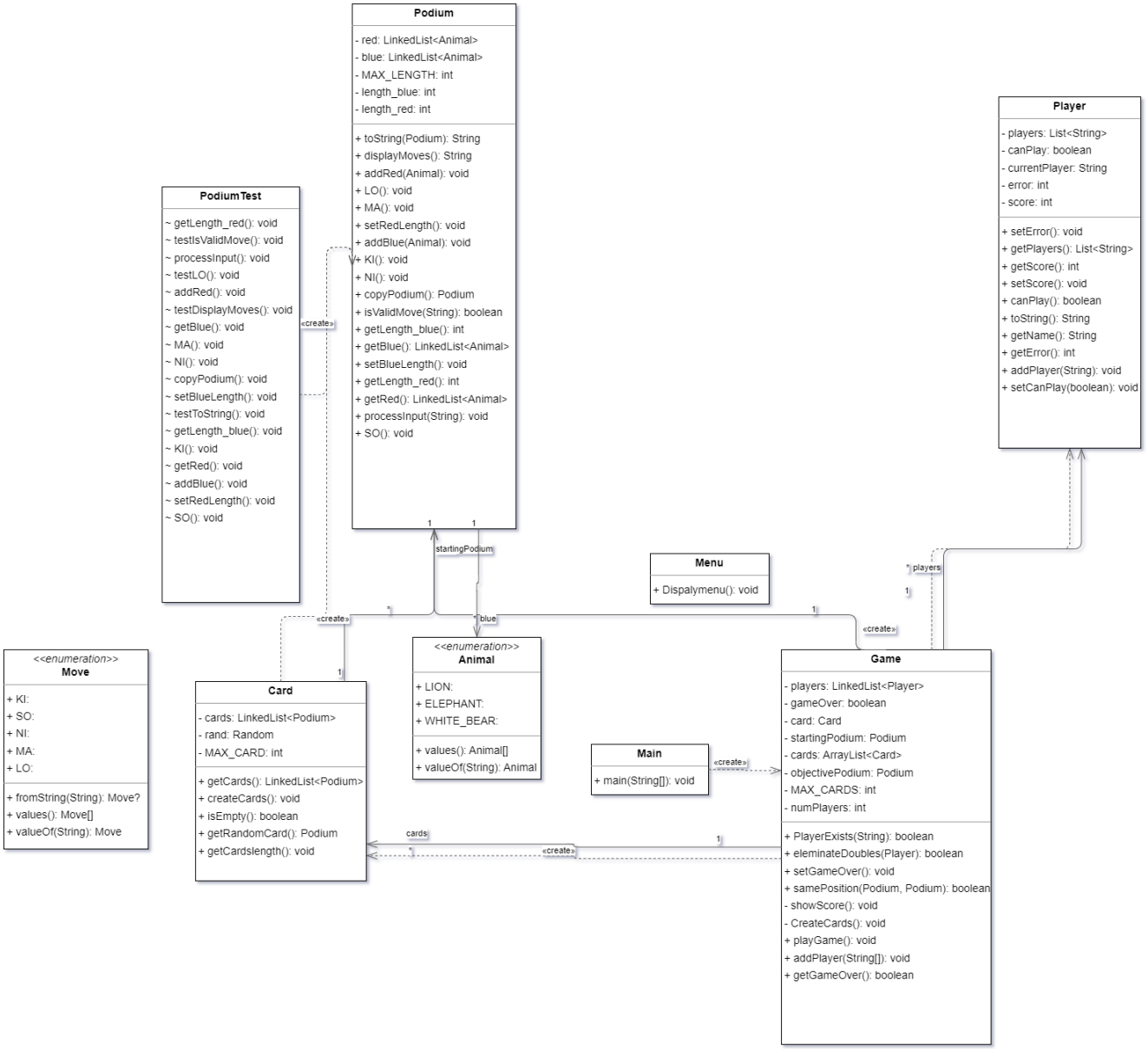
# BILAN DE PROJET

Ce projet a été enrichissant et a permis à l'équipe de développement d'améliorer nos compétences en matière de développement d'application en Java. Nous avons rencontré des défis techniques tout au long du projet.
Tout d'abord nous avons un problème concernant la création des cartes. En effet, nous devons créer 24 cartes toutes différentes les unes des autres. Pour cela, nous avons eu le choix de soit créer manuellement les 24 cartes ou soit créer automatiquement. Nous avons rencontré des difficultés à les créer automatiquement, nous avons donc opté pour créer manuellement. Nous avons ensuite eu des problèmes par rapport à l'affichage des podiums.

De plus, pendant le processus de développement de l'application, nous avons rencontré des difficultés techniques liées à l'affichage des podiums dans l'interface utilisateur. Nous avons essayé différentes approches pour résoudre ce problème. Après plusieurs essaie et échec nous avons finalement trouvé une solution satisfaisante en utilisant des éléments graphiques personnalisés.

Malgré ces défis, le projet a été une expérience extrêmement enrichissante pour l'équipe de développement. Nous avons acquis des compétences précieuses en matière de développement d'application en Java et avons appris à travailler efficacement en équipe pour surmonter les obstacles. Nous sommes également fiers d'avoir créé une application fonctionnelle qui répond aux exigences du cahier des charges.

En fin de compte, le jeu Crazy Circus est un excellent exemple de la façon dont les jeux de société peuvent être adaptés au monde numérique, tout en offrant une expérience de jeu enrichissante et stimulante pour les joueurs de tous niveaux. Nous espérons que les joueurs apprécieront notre application et nous avons hâte de continuer à développer nos compétences dans le domaine de l'informatique et du développement d'applications.


Podium
- red: LinkedList<Animal>
- blue: LinkedList<Animal>
- MAX_LENGTH: int
- length_blue: int
- length_red: int
+ toString(Podium): String
+ displayMoves(): String
+ addRed(Animal): void
+ LO(): void
+ MA(): void
+ setRedLength(): void
+ addBlue(Animal): void
+ KI(): void
+ NI(): void
+ copyPodium(): Podium
+ isValidMove(String): boolean
+ getLength_blue(): int
+ getBlue(): LinkedList<Animal>
+ setBlueLength(): void
+ getLength_red(): int
+ getRed(): LinkedList<Animal>
+ processInput(String): void
+ SO(): void
PodiumTest
~ getLength_red(): void
~ testIsValidMove(): void
~ processInput(): void
~ testLO(): void
~ addRed(): void
~ testDisplayMoves(): void
~ getBlue(): void
~ MA(): void
~ NI(): void
~ copyPodium(): void
~ setBlueLength(): void
~ testToString(): void
~ getLength_blue(): void
~ KI(): void
~ getRed(): void
~ addBlue(): void
~ setRedLength(): void
~ SO(): void
«create»
Player
- players: List<String>
- canPlay: boolean
- currentPlayer: String
- error: int
- score: int
+ setError(): void
+ getPlayers(): List<String>
+ getScore(): int
+ setScore(): void
+ canPlay(): boolean
+ toString(): String
+ getName(): String
+ getError(): int
+ addPlayer(String): void
+ setCanPlay(boolean): void
1
1
startingPodium
Menu
+ Dispalymenu(): void
* players
1
«create»
«create»
*
*
blue
1
1
<<enumeration>>
Move
+ KI:
+ SO:
+ NI:
+ MA:
+ LO:
+ fromString(String): Move?
+ values(): Move[]
+ valueOf(String): Move
Card
- cards: LinkedList<Podium>
- rand: Random
- MAX_CARD: int
+ getCards(): LinkedList<Podium>
+ createCards(): void
+ isEmpty(): boolean
+ getRandomCard(): Podium
+ getCardslength(): void
<<enumeration>>
Animal
+ LION:
+ ELEPHANT:
+ WHITE_BEAR:
+ values(): Animal[]
+ valueOf(String): Animal
Main
+ main(String[]): void
«create»
Game
- players: LinkedList<Player>
- gameOver: boolean
- card: Card
- startingPodium: Podium
- cards: ArrayList<Card>
- objectivePodium: Podium
- MAX_CARDS: int
- numPlayers: int
+ PlayerExists(String): boolean
+ eleminateDoubles(Player): boolean
+ setGameOver(): void
+ samePosition(Podium, Podium): boolean
- showScore(): void
- CreateCards(): void
+ playGame(): void
+ addPlayer(String[]): void
+ getGameOver(): boolean
cards
*
«create»
1

Voici les tests :

Tous les tests sont fonctionnels.

```
package Card;

import org.junit.jupiter.api.Test;

import java.util.LinkedList;

import static org.junit.jupiter.api.Assertions.*;

class PodiumTest {

    @Test
    void getBlue() {
        Podium blue = new Podium();
        blue.addBlue(Animal.LION);
        blue.addBlue(Animal.ELEPHANT);
        blue.addBlue(Animal.WHITE_BEAR);

        LinkedList<Animal> blueList = blue.getBlue();
        assertEquals(Animal.LION, blueList.get(0));
        assertEquals(Animal.ELEPHANT, blueList.get(1));
        assertEquals(Animal.WHITE_BEAR, blueList.get(2));
    }

    @Test
    void getRed() {
        Podium red = new Podium();
        red.addRed(Animal.LION);
        red.addRed(Animal.ELEPHANT);
        red.addRed(Animal.WHITE_BEAR);

        LinkedList<Animal> redList = red.getRed();
        assertEquals(Animal.LION, redList.get(0));
        assertEquals(Animal.ELEPHANT, redList.get(1));
        assertEquals(Animal.WHITE_BEAR, redList.get(2));
    }

    @Test
    void getLength_blue() {
        Podium blue = new Podium();
        blue.addBlue(Animal.LION);
        blue.addBlue(Animal.ELEPHANT);
        blue.addBlue(Animal.WHITE_BEAR);

        assertEquals(3, blue.getLength_blue());
    }
```

```java
@Test
    void getLength_red() {
        Podium red = new Podium();
        red.addRed(Animal.LION);
        red.addRed(Animal.ELEPHANT);
        red.addRed(Animal.WHITE_BEAR);

        assertEquals(3, red.getLength_red());
    }

    @Test
    void addBlue() {
        Podium blue = new Podium();
        blue.addBlue(Animal.LION);
        blue.addBlue(Animal.ELEPHANT);
        blue.addBlue(Animal.WHITE_BEAR);

        LinkedList<Animal> blueList = blue.getBlue();
        assertEquals(Animal.LION, blueList.get(0));
        assertEquals(Animal.ELEPHANT, blueList.get(1));
        assertEquals(Animal.WHITE_BEAR, blueList.get(2));
    }

    @Test
    void addRed() {
        Podium red = new Podium();
        red.addRed(Animal.LION);
        red.addRed(Animal.ELEPHANT);
        red.addRed(Animal.WHITE_BEAR);

        LinkedList<Animal> redList = red.getRed();
        assertEquals(Animal.LION, redList.get(0));
        assertEquals(Animal.ELEPHANT, redList.get(1));
        assertEquals(Animal.WHITE_BEAR, redList.get(2));
    }
    @Test
    void setBlueLength() {
        Podium blue = new Podium();
        blue.addBlue(Animal.LION);
        blue.addBlue(Animal.ELEPHANT);
        blue.addBlue(Animal.WHITE_BEAR);

        LinkedList<Animal> blueList = blue.getBlue();
        assertEquals(Animal.LION, blueList.get(0));
        assertEquals(Animal.ELEPHANT, blueList.get(1));
        assertEquals(Animal.WHITE_BEAR, blueList.get(2));

        assertEquals(3, blue.getLength_blue());
    }
```

```java
@Test
    void setRedLength() {
        Podium red = new Podium();
        red.addRed(Animal.LION);
        red.addRed(Animal.ELEPHANT);
        red.addRed(Animal.WHITE_BEAR);

        LinkedList<Animal> redList = red.getRed();
        assertEquals(Animal.LION, redList.get(0));
        assertEquals(Animal.ELEPHANT, redList.get(1));
        assertEquals(Animal.WHITE_BEAR, redList.get(2));

        assertEquals(3, red.getLength_red());
    }

    @Test
    void KI() {
        LinkedList<Animal> blueList = new LinkedList<>();
        LinkedList<Animal> redList = new LinkedList<>();

        blueList.add(Animal.LION);
        blueList.add(Animal.ELEPHANT);

        redList.add(Animal.WHITE_BEAR);

        Podium test = new Podium(blueList, redList);
        test.KI();
        assertEquals(Animal.WHITE_BEAR, test.getRed().get(0));
        assertEquals(Animal.ELEPHANT, test.getRed().get(1));
        assertEquals(Animal.LION, test.getBlue().get(0));

        LinkedList<Animal> blueList1 = new LinkedList<>();
        LinkedList<Animal> redList1 = new LinkedList<>();

        blueList1.add(Animal.LION);
        blueList1.add(Animal.ELEPHANT);
        blueList1.add(Animal.WHITE_BEAR);

        Podium test1 = new Podium(blueList1, redList1);
        test1.KI();
        assertEquals(Animal.WHITE_BEAR, test1.getRed().get(0));
        assertEquals(Animal.LION, test1.getBlue().get(0));
        assertEquals(Animal.ELEPHANT, test1.getBlue().get(1));
    }
```

```java
@Test
    void testLO() {
        LinkedList<Animal> blueList = new LinkedList<>();
        LinkedList<Animal> redList = new LinkedList<>();

        blueList.add(Animal.LION);

        redList.add(Animal.WHITE_BEAR);
        redList.add(Animal.ELEPHANT);

        Podium test = new Podium(blueList, redList);
        test.LO();

        assertEquals(Animal.ELEPHANT, test.getBlue().get(1));
        assertEquals(Animal.WHITE_BEAR, test.getRed().getFirst());
        assertEquals(2, test.getBlue().size());
        assertEquals(1, test.getRed().size());
    }

    @Test
    void SO() {
        LinkedList<Animal> blueList = new LinkedList<>();
        LinkedList<Animal> redList = new LinkedList<>();

        blueList.add(Animal.LION);
        blueList.add(Animal.ELEPHANT);

        redList.add(Animal.WHITE_BEAR);

        Podium test = new Podium(blueList, redList);
        test.SO();
        assertEquals(Animal.WHITE_BEAR, test.getBlue().get(1));
        assertEquals(Animal.LION, test.getBlue().get(0));
        assertEquals(Animal.ELEPHANT, test.getRed().get(0));

        LinkedList<Animal> blueList1 = new LinkedList<>();
        LinkedList<Animal> redList1 = new LinkedList<>();

        blueList1.add(Animal.LION);

        redList1.add(Animal.WHITE_BEAR);
        redList1.add(Animal.ELEPHANT);

        Podium test1 = new Podium(blueList1, redList1);
        test1.SO();
        assertEquals(Animal.ELEPHANT, test1.getBlue().get(0));
        assertEquals(Animal.LION, test1.getRed().get(1));
        assertEquals(Animal.WHITE_BEAR, test1.getRed().get(0));
    }
```

```java
@Test
void MA() {
    LinkedList<Animal> BlueList = new LinkedList<>();
    LinkedList<Animal> RedList = new LinkedList<>();


    RedList.add(Animal.LION);
    RedList.add(Animal.ELEPHANT);
    RedList.add(Animal.WHITE_BEAR);

    Podium test = new Podium(BlueList, RedList);
    test.MA();

    assertEquals(Animal.LION, test.getRed().get(2));
    assertEquals(Animal.WHITE_BEAR, test.getRed().get(1));
    assertEquals(Animal.ELEPHANT, test.getRed().getFirst());

    LinkedList<Animal> BlueList1 = new LinkedList<>();
    LinkedList<Animal> RedList1 = new LinkedList<>();

    RedList1.add(Animal.LION);
    RedList1.add(Animal.ELEPHANT);

    Podium test1 = new Podium(BlueList1, RedList1);
    test1.MA();

    assertEquals(Animal.LION, test1.getRed().get(1));
    assertEquals(Animal.ELEPHANT, test1.getRed().getFirst());
}
```

```java
@Test
    void NI() {
        LinkedList<Animal> BlueList = new LinkedList<>();
        LinkedList<Animal> RedList = new LinkedList<>();

        BlueList.add(Animal.LION);
        BlueList.add(Animal.ELEPHANT);
        BlueList.add(Animal.WHITE_BEAR);

        Podium test = new Podium(BlueList, RedList);
        test.NI();

        assertEquals(Animal.LION, test.getBlue().get(2));
        assertEquals(Animal.WHITE_BEAR, test.getBlue().get(1));
        assertEquals(Animal.ELEPHANT, test.getBlue().getFirst());

        LinkedList<Animal> BlueList1 = new LinkedList<>();
        LinkedList<Animal> RedList1 = new LinkedList<>();

        BlueList1.add(Animal.LION);
        BlueList1.add(Animal.ELEPHANT);

        Podium test1 = new Podium(BlueList1, RedList1);
        test1.NI();

        assertEquals(Animal.LION, test1.getBlue().get(1));
        assertEquals(Animal.ELEPHANT, test1.getBlue().getFirst());

    }

    @Test
    void testIsValidMove() {
        // Valid input
        assertTrue(Podium.isValidMove("KILO"));
        assertTrue(Podium.isValidMove("LOMAKI"));
        assertTrue(Podium.isValidMove("SOKINILOMA"));
        assertTrue(Podium.isValidMove("NIMA"));
        assertTrue(Podium.isValidMove("MAKI"));
        assertTrue(Podium.isValidMove("LO"));
        assertTrue(Podium.isValidMove("SO"));
        assertTrue(Podium.isValidMove("MA"));
        assertTrue(Podium.isValidMove("NI"));
    }
```

```java
@Test
void processInput() {
    //KI
    LinkedList<Animal> blueList = new LinkedList<>();
    LinkedList<Animal> redList = new LinkedList<>();

    blueList.add(Animal.LION);
    blueList.add(Animal.ELEPHANT);

    redList.add(Animal.WHITE_BEAR);

    Podium test = new Podium(blueList, redList);
    test.processInput("KI");
    assertEquals(Animal.WHITE_BEAR, test.getRed().get(0));
    assertEquals(Animal.ELEPHANT, test.getRed().get(1));
    assertEquals(Animal.LION, test.getBlue().get(0));

    //LO
    LinkedList<Animal> blueList1 = new LinkedList<>();
    LinkedList<Animal> redList1 = new LinkedList<>();
    blueList1.add(Animal.LION);

    redList1.add(Animal.WHITE_BEAR);
    redList1.add(Animal.ELEPHANT);

    Podium test1 = new Podium(blueList1, redList1);
    test1.processInput("LO");
    assertEquals(Animal.ELEPHANT, test1.getBlue().get(1));
    assertEquals(Animal.WHITE_BEAR, test1.getRed().getFirst());
    assertEquals(2, test1.getBlue().size());
    assertEquals(1, test1.getRed().size());

    //SO
    LinkedList<Animal> blueList2 = new LinkedList<>();
    LinkedList<Animal> redList2 = new LinkedList<>();
    blueList2.add(Animal.LION);
    blueList2.add(Animal.ELEPHANT);

    redList2.add(Animal.WHITE_BEAR);

    Podium test2 = new Podium(blueList2, redList2);
    test2.processInput("SO");
    assertEquals(Animal.WHITE_BEAR, test2.getBlue().get(1));
    assertEquals(Animal.LION, test2.getBlue().get(0));
    assertEquals(Animal.ELEPHANT, test2.getRed().get(0));
```

```
        //MA
        LinkedList<Animal> blueList3 = new LinkedList<>();
        LinkedList<Animal> redList3 = new LinkedList<>();
        redList3.add(Animal.LION);
        redList3.add(Animal.ELEPHANT);
        redList3.add(Animal.WHITE_BEAR);

        Podium test3 = new Podium(blueList3, redList3);
        test3.processInput("MA");

        assertEquals(Animal.LION, test3.getRed().get(2));
        assertEquals(Animal.WHITE_BEAR, test3.getRed().get(1));
        assertEquals(Animal.ELEPHANT, test3.getRed().getFirst());

        //NI
        LinkedList<Animal> blueList4 = new LinkedList<>();
        LinkedList<Animal> redList4 = new LinkedList<>();
        blueList4.add(Animal.LION);
        blueList4.add(Animal.ELEPHANT);
        blueList4.add(Animal.WHITE_BEAR);

        Podium test4 = new Podium(blueList4, blueList4);
        test4.processInput("NI");

        assertEquals(Animal.LION, test4.getBlue().get(2));
        assertEquals(Animal.WHITE_BEAR, test4.getBlue().get(1));
        assertEquals(Animal.ELEPHANT, test4.getBlue().getFirst());
    }
```

```
@Test
    void testDisplayMoves() {
        Podium p = new Podium();
        String expected = " KI : Bleu-->Rouge     |     MA : Rouge
^\n" +
                " LO : Blue<--Rouge     |     NI : Bleu ^\n" +
                " SO : Bleu<-->Rouge";
        String result = p.displayMoves();
        assertEquals(expected, result);
    }
    @Test
    void testToString() {
    }

    @Test
    void copyPodium() {
        LinkedList<Animal> blueList = new LinkedList<>();
        LinkedList<Animal> redList = new LinkedList<>();

        blueList.add(Animal.LION);
        blueList.add(Animal.ELEPHANT);

        redList.add(Animal.WHITE_BEAR);

        Podium test = new Podium(blueList, redList);
        Podium test1 = test.copyPodium();

        assertEquals(test.getBlue(), test1.getBlue());
        assertEquals(test.getRed(), test1.getRed());
    }
}
```

Voici les classes :

```java
package Card;

public enum Animal { // On crée une liste d'éléments
	LION,
	ELEPHANT,
	WHITE_BEAR,

}
```

```java
package Card;

public enum Move {
    KI,
    LO,
    SO,
    NI,
    MA;
    public static Move fromString(String str) {
        try {
            return valueOf(str.toUpperCase());
        } catch (IllegalArgumentException e) {
            return null;
        }
    }
}
```

```java
package Card;

import java.util.*;
import java.util.stream.Collectors;

public class Podium {
    private LinkedList<Animal> blue;
    private LinkedList<Animal> red;
    private final int MAX_LENGTH = 3;
    private int length_blue;
    private int length_red;

    //-----------Constructors----------------
    public Podium() {
        blue = new LinkedList<>();
        red = new LinkedList<>();
        length_blue = 0;
        length_red = 0;
    }
    public Podium(LinkedList<Animal> bluePodium, LinkedList<Animal>
redPodium) {
        this.blue = bluePodium;
        this.red = redPodium;
        setBlueLength();
        setRedLength();
    }
    public Podium(Podium podium) {
        this.blue = new LinkedList<>(podium.blue);
        this.red = new LinkedList<>(podium.red);
        this.length_blue = podium.length_blue;
        this.length_red = podium.length_red;
    }
```

```java
//------------Getters----------------
    public LinkedList<Animal> getBlue() {
        return blue;
    }

    public LinkedList<Animal> getRed() {
        return red;
    }
    public int getLength_blue() {
        return length_blue;
    }
    public int getLength_red() {
        return length_red;
    }


//------------Setters----------------

    /**
     * @param animal Permet d'ajouter les animaux dans le podium bleu
     */
    public void addBlue(Animal animal) {
        blue.add(animal);
        setBlueLength();
    }

    /**
     * @param animal Permet d'ajouter les animaux dans le podium rouge
     */
    public void addRed(Animal animal) {
        red.add(animal);
        setRedLength();
    }
    public void setBlueLength() {
        length_blue = blue.size();
    }
    public void setRedLength() {
        length_red = red.size();
    }
```

```java
//------------Les Méthodes----------------

    /**
     * Permet de deplacer les animaux du podium bleu vers le podium
rouge
     */
    public void KI() {
        if (blue.size() != 0) {
            Animal animal = blue.removeLast();
            red.addLast(animal);
        } else {
            System.out.println("Ordre incorrect ! Vous n'avez plus le
droit de jouer ce tour.");
        }
    }

    /**
     * Permet de deplacer les animaux du podium rouge vers le podium
bleu
     */
    public void LO() {
        if (red.size() != 0) {
            Animal animal = red.removeLast();
            blue.addLast(animal);
        } else {
            System.out.println("Ordre incorrect ! Vous n'avez plus le
droit de jouer ce tour.");
        }
    }

    /**
     * Permet de swap les animaux
     */
    public void SO() {
        if (blue.size() != 0 && red.size() != 0) {
            Animal animal = blue.removeLast();
            Animal animal1 = red.removeLast();
            red.addLast(animal);
            blue.addLast(animal1);
        } else {
            System.out.println("Ordre incorrect ! Vous n'avez plus le
droit de jouer ce tour.");
        }
    }
```

```java
/**
     * 1Permet de deplacer l'animal du podium rouge vers le sommet du
podium rouge
     */
    public void MA() {
        if (red.size() > 0 && red.size() <= 3) {
            Animal animal = red.removeFirst();
            Animal animal1 = red.removeLast();
            red.add(animal1);
            red.addLast(animal);
        } else {
            System.out.println("Ordre incorrect ! Vous n'avez plus le
droit de jouer ce tour.");
        }
    }

    /**
     * @breif Permet de deplacer l'animal du podium bleu vers le
sommet du podium bleu
     */
    public void NI() {
        if (blue.size() > 0 && blue.size() <= 3) {
            Animal animal = blue.removeFirst(); // Retire le dernier
élément de la liste
            Animal animal1 = blue.removeLast(); // Retire le premier
élément de la liste
            blue.add(animal1); // Ajoute l'élément à la fin de la
liste
            blue.addLast(animal); // Ajoute l'élément au début de la
liste
        } else {
            System.out.println("Ordre incorrect ! Vous n'avez plus le
droit de jouer ce tour."); // Affiche un message d'erreur
        }
    }
```

```java
/**
     * @breif Permet de Vérifier si la chaîne d'entrée est une
séquence valide de mouvements
     * @param input chaîne de caractères à vérifier
     * @return true si l'ordre est valide, false sinon
     */
    public static boolean isValidMove(String input) {
        /*
        Set<String> validCommands = Arrays.stream(Move.values())
                .map(Enum::name)
                .collect(Collectors.toSet());
         */
        // Crée un ensemble de chaînes de caractères valides à partir
des valeurs de l'énumération Move
        Set<String> validCommands = new HashSet<>(); // Création d'un
objet HashSet
        for (Move move : Move.values()) { // Parcours de la liste des
mouvements
            validCommands.add(move.name()); // Ajout des mouvements
dans la liste
        }
        int i = 0; // Initialisation de i à 0
        while (i < input.length()) { // Tant que i est inférieur à la
longueur de la chaîne de caractères
            String command = input.substring(i, i + 2); // On récupère
les deux premiers caractères
            // Vérifie si la commande est valide en la comparant avec
l'ensemble des commandes valides
            if (validCommands.contains(command)) {
                i += 2;
            } else {
                return false; // Si la commande n'est pas valide,
retourne false
            }
        }
        return true;  // Si toutes les commandes sont valides,
retourne true
    }
```

```java
/**
     * @breif Permet de traiter les séquences de l'utilisateur
     * @param input Permet de traiter les entrées de l'utilisateur
     */
    public void processInput(String input) {
        input = input.toUpperCase(); // On met la chaîne de caractères en majuscule
        for (int i = 0; i < input.length(); i += 2) {
            String move = input.substring(i, i + 2); // On récupère les deux premiers caractères
            if (!isValidMove(input)) { // Si la commande n'est pas valide
                System.out.println("Invalid move: " + move); // On affiche un message d'erreur
                return; // On arrête le traitement
            }
            switch (move) {  // On traite la commande
                case "KI":  // Si la commande est KI
                    KI();  // On appelle la méthode KI
                    break; // On arrête le traitement
                case "LO": // Si la commande est LO
                    LO();// On appelle la méthode LO
                    break; // On arrête le traitement
                case "SO": // Si la commande est SO
                    SO(); // On appelle la méthode SO
                    break; // On arrête le traitement
                case "NI": // Si la commande est NI
                    NI();// On appelle la méthode NI
                    break; // On arrête le traitement
                case "MA": // Si la commande est MA
                    MA(); // On appelle la méthode MA
                    break; // On arrête le traitement
                default: // Si la commande n'est pas valide
                    System.out.println("Invalid move: " + move); // On affiche un message d'erreur
                    return; // On arrête le traitement
            }
        }
    }
```

```java
/**
     * @breif Permet d'afficher les mouvements possibles
     * @return Retour de la chaîne de caractères finale
     */
    public String displayMoves() {
        return String.format(" %5s %5s  %15s\n %5s %5s %15s\n %s", "KI
: Bleu-->Rouge", "|", "MA : Rouge ^",
                "LO : Blue<--Rouge", "|", "NI : Bleu ^", "SO : Bleu<--
>Rouge");
    }
```

```java
/**
     * @breif Permet d'afficher le podium
     * @return Retour de la chaîne de caractères finale
     */
    public String toString(Podium objectivePodium) {
        StringBuilder sb = new StringBuilder();
        int maxStackSize = Math.max(blue.size(), red.size());

        Podium objectiveState = objectivePodium == null ? null : new
Podium(objectivePodium);
        for (int i = maxStackSize ; i >= 0; i--) {
            // Blue stack
            if (i < blue.size()) {
                sb.append(String.format("%-12s", blue.get(i)));
            } else {
                sb.append(String.format("%-12s", ""));
            }

            sb.append("     ");

            // Red stack
            if (i < red.size()) {
                sb.append(String.format("%-12s", red.get(i)));
            } else {
                sb.append(String.format("%-12s", ""));
            }

            if (objectiveState != null) {
                sb.append("            ");

                // Objective Blue stack
                if (i < objectiveState.blue.size()) {
                    sb.append(String.format("%-12s",
objectiveState.blue.get(i)));
                } else {
                    sb.append(String.format("%-12s", ""));
                }
                sb.append("     ");

                // Objective Red stack
                if (i < objectiveState.red.size()) {
                    sb.append(String.format("%-12s",
objectiveState.red.get(i)));
                } else {
                    sb.append(String.format("%-12s", ""));
                }
            }
            sb.append("\n");
        }
```

```
// Podium separator
        for (int i = 0; i < 12; i++) {
            sb.append("-");
        }

        sb.append("     ");

        for (int i = 0; i < 12; i++) {
            sb.append("-");
        }

        if (objectiveState != null) {
            sb.append("            ");

            for (int i = 0; i < 12; i++) {
                sb.append("-");
            }

            sb.append("     ");

            for (int i = 0; i < 12; i++) {
                sb.append("-");
            }
        }

        sb.append("\n");

        // Podium labels
        sb.append(String.format("%-12s", "BLUE"));
        sb.append("     ");
        sb.append(String.format("%-12s", "ROUGE"));

        if (objectiveState != null) {
            sb.append("            ");
            sb.append(String.format("%-12s", "BLUE"));
            sb.append("     ");
            sb.append(String.format("%-12s", "ROUGE"));
        }
```

```
sb.append("\n");
        sb.append("-------------------------------------------------
--------------------");
        sb.append("\n");
        sb.append(this.displayMoves());
        sb.append("\n");
        return sb.toString();
    }



    /**
     * @breif Permet de copier le podium
     * @return Retourne le podium copié
     */
    public Podium copyPodium() {
        Podium copy = new Podium();
        copy.blue = new LinkedList<>(blue);
        copy.red = new LinkedList<>(red);
        return copy;
    }
```

Class Card :

```java
package Card;

import java.util.*;

import java.util.LinkedList;


public class Card {

    private LinkedList<Podium> cards;
    private Random rand;
    private final int MAX_CARD = 24;

    /**
     * @brief Constructeur de la classe Card
     */
    public Card() {
        this.cards = new LinkedList<>();
        this.rand = new Random();
    }

    /**
     * @brief permet de récupérer la liste de carte
     * @return the cards
     */
    public LinkedList<Podium> getCards() {
        return cards;
    }

    /**
     * @brief permet de récupérer la taille de la liste de carte
     */
    public void getCardslength() {
        System.out.println((cards.size()));
    }
```

```java
    /**
     * @brief permet de savoir si la liste de carte est vide
     * @return true si la liste est vide, false sinon
     */
    public boolean isEmpty() {
        return cards.isEmpty();
    }

    /**
     * @brief permet de récupérer une carte aléatoire
     * @return une carte aléatoire
     */
    public Podium getRandomCard() {
        assert (!this.isEmpty()); // On vérifie que la liste n'est pas
vide
        Podium p = cards.get(rand.nextInt(this.cards.size())); // On
récupère une carte aléatoire
        cards.remove(p); // On la supprime de la liste
        return p; // On la retourne
    }
```

```java
/**
 * @brief permet de créer les cartes
 */
public void createCards() {
    Animal A = Animal.LION;
    Animal B = Animal.ELEPHANT;
    Animal C = Animal.WHITE_BEAR;

    Podium CARD_ZERO = new Podium(); //1
    CARD_ZERO.addBlue(C);
    CARD_ZERO.addBlue(B);
    CARD_ZERO.addBlue(A);
    cards.add(CARD_ZERO);

    Podium CARD_ONE = new Podium(); //2
    CARD_ONE.addBlue(C);
    CARD_ONE.addBlue(A);
    CARD_ONE.addBlue(B);
    cards.add(CARD_ONE);

    Podium CARD_TWO = new Podium();     //3
    CARD_TWO.addBlue(B);
    CARD_TWO.addBlue(C);
    CARD_TWO.addBlue(A);
    cards.add(CARD_TWO);

    Podium CARD_THREE = new Podium();  //4
    CARD_THREE.addBlue(B);
    CARD_THREE.addBlue(A);
    CARD_THREE.addBlue(C);
    cards.add(CARD_THREE);

    Podium CARD_FOUR = new Podium(); //5
    CARD_FOUR.addBlue(A);
    CARD_FOUR.addBlue(B);
    CARD_FOUR.addBlue(C);
    cards.add(CARD_FOUR);

    Podium CARD_FIVE = new Podium(); //6
    CARD_FIVE.addBlue(A);
    CARD_FIVE.addBlue(C);
    CARD_FIVE.addBlue(B);
    cards.add(CARD_FIVE);
```

```
Podium CARD_SIX = new Podium(); //7
        CARD_SIX.addRed(B);
        CARD_SIX.addRed(C);
        CARD_SIX.addRed(A);

        Podium CARD_SEVEN = new Podium(); //8
        CARD_SEVEN.addRed(B);
        CARD_SEVEN.addRed(A);
        CARD_SEVEN.addRed(C);
        cards.add(CARD_SEVEN);

        Podium CARD_EIGHT = new Podium(); //9
        CARD_EIGHT.addRed(A);
        CARD_EIGHT.addRed(B);
        CARD_EIGHT.addRed(C);
        cards.add(CARD_EIGHT);

        Podium CARD_NINE = new Podium(); //10
        CARD_NINE.addRed(A);
        CARD_NINE.addRed(C);
        CARD_NINE.addRed(B);
        cards.add(CARD_NINE);

        Podium CARD_TEN = new Podium(); //11
        CARD_TEN.addRed(C);
        CARD_TEN.addRed(B);
        CARD_TEN.addRed(A);
        cards.add(CARD_TEN);

        Podium CARD_ELEVEN = new Podium(); //12
        CARD_ELEVEN.addRed(C);
        CARD_ELEVEN.addRed(A);
        CARD_ELEVEN.addRed(B);
        cards.add(CARD_ELEVEN);

        Podium CARD_TWELVE = new Podium(); //13
        CARD_TWELVE.addBlue(C);
        CARD_TWELVE.addBlue(A);
        CARD_TWELVE.addRed(B);
        cards.add(CARD_TWELVE);

        Podium CARD_THIRTEEN = new Podium(); //14
        CARD_THIRTEEN.addRed(C);
        CARD_THIRTEEN.addRed(A);
        CARD_THIRTEEN.addBlue(B);
        cards.add(CARD_THIRTEEN);
```

```
Podium CARD_FOURTEEN = new Podium(); //15
        CARD_FOURTEEN.addBlue(C);
        CARD_FOURTEEN.addBlue(B);
        CARD_FOURTEEN.addRed(A);
        cards.add(CARD_FOURTEEN);

        Podium CARD_FIFTEEN = new Podium(); //16
        CARD_FIFTEEN.addRed(C);
        CARD_FIFTEEN.addRed(B);
        CARD_FIFTEEN.addBlue(A);
        cards.add(CARD_FIFTEEN);

        Podium CARD_SIXTEEN = new Podium(); //17
        CARD_SIXTEEN.addBlue(A);
        CARD_SIXTEEN.addBlue(C);
        CARD_SIXTEEN.addRed(B);
        cards.add(CARD_SIXTEEN);

        Podium CARD_SEVENTEEN = new Podium(); //18
        CARD_SEVENTEEN.addRed(A);
        CARD_SEVENTEEN.addRed(C);
        CARD_SEVENTEEN.addBlue(B);
        cards.add(CARD_SEVENTEEN);

        Podium CARD_EIGHTEEN = new Podium(); //19
        CARD_EIGHTEEN.addBlue(A);
        CARD_EIGHTEEN.addBlue(B);
        CARD_EIGHTEEN.addRed(C);
        cards.add(CARD_EIGHTEEN);

        Podium CARD_NINETEEN = new Podium(); //20
        CARD_NINETEEN.addRed(A);
        CARD_NINETEEN.addRed(B);
        CARD_NINETEEN.addBlue(C);
        cards.add(CARD_NINETEEN);

        Podium CARD_TWENTY = new Podium(); //21
        CARD_TWENTY.addBlue(B);
        CARD_TWENTY.addBlue(A);
        CARD_TWENTY.addRed(C);
        cards.add(CARD_TWENTY);

        Podium CARD_TWENTYONE = new Podium(); //22
        CARD_TWENTYONE.addRed(B);
        CARD_TWENTYONE.addRed(A);
        CARD_TWENTYONE.addBlue(C);
        cards.add(CARD_TWENTYONE);
```

```
        Podium CARD_TWENTYTWO = new Podium(); //23
        CARD_TWENTYTWO.addRed(B);
        CARD_TWENTYTWO.addRed(C);
        CARD_TWENTYTWO.addBlue(A);
        cards.add(CARD_TWENTYTWO);

        Podium CARD_TWENTYTHREE = new Podium(); //24
        CARD_TWENTYTHREE.addBlue(B);
        CARD_TWENTYTHREE.addBlue(C);
        CARD_TWENTYTHREE.addRed(A);
        cards.add(CARD_TWENTYTHREE);
    }
}
```

```java
package Game;

import Card.*;

import java.util.*;

public class Game {
    private static final int MAX_CARDS = 24; // Nombre de cartes dans le jeu
    private static Card card; // On crée une carte
    private static LinkedList<Player> players; // On crée une liste de joueurs
    private static Podium startingPodium; // On crée un podium de départ
    private static Podium objectivePodium; // On crée un podium d'arrivée
    private ArrayList<Card> cards; // On crée une liste de cartes
    private static boolean gameOver; // On crée une variable qui permet de savoir si la partie est finie
    private int numPlayers; // On crée une variable qui permet de savoir le nombre de joueurs

    //------------Constructors----------------
    public Game(String[] args) { // On crée un constructeur qui prend en paramètre un tableau de String
        players = new LinkedList<>(); // On initialise la liste de joueurs
        card = new Card(); // On initialise la carte
        gameOver = false; // On initialise la variable gameOver à false
        card.createCards(); // On crée les cartes
        addPlayer(args); // On ajoute les joueurs
        CreateCards(); // On lance la première partie
    }


    /**
     * @param name Le nom du joueur
     * @return true si le nom du joueur est déjà pris, false sinon
     * @brief Permet de savoir si le nom du joueur est déjà pris
     */
    public static boolean PlayerExists(String name) {
        return eleminateDoubles(new Player(name)); // On crée un joueur temporaire pour pouvoir utiliser la méthode eleminateDoubles: true si le nom du joueur est déjà pris, false sinon
    }
```

```java
/**
     * @param p permet de vérifier si le nom du joueur est déjà pris
     * @return true si le nom du joueur est déjà pris, false sinon
     * @brief Permet de verifier si le nom du joueur est déjà pris
     */
    public static boolean eleminateDoubles(Player p) {
        for (Player player : players) { // On parcourt la liste de joueurs
            if (player != p && player.getPlayers().equals(p.getPlayers())) { // Si le nom du joueur est déjà pris
                return true; // On retourne true
            }
        }
        return false; // On retourne false
    }


    /**
     * @param startingPodium  Le podium de départ
     * @param objectivePodium Le podium d'objectif
     * @return true si les animaux sur les deux podiums sont à la même position, false sinon
     * @breif Permet de savoir si les animaux sur les deux podiums sont à la même position
     */
    public static boolean samePosition(Podium startingPodium, Podium objectivePodium) {
        return startingPodium.getBlue().equals(objectivePodium.getBlue()) && startingPodium.getRed().equals(objectivePodium.getRed()); // On retourne true si les animaux sur les deux podiums sont à la même position, false sinon
    }
```

```java
/**
     * @Brief Permet de jouer une partie
     *
     */
    public static void playGame() {
        Scanner scanner = new Scanner(System.in);

        String input; // On récupère l'input de l'utilisateur

        boolean finished = false; // On initialise la variable
finished à false pour pouvoir jouer


        int numErrors = 0; // Permet de compter le nombre d'erreurs

        Player previousPlayerWhoMadeAnError = null; // Permet de
savoir qui a fait une erreur

        while (!finished) { // Tant que la partie n'est pas finie

            System.out.println(startingPodium.toString(objectivePodium
)); // On affiche le podium de départ et le podium d'objectif

            input = scanner.nextLine(); // On récupère l'input de
l'utilisateur

            Podium copyOfStaringPodium = startingPodium.copyPodium();
// On crée une copie du podium de départ

            String name = input.substring(0, 2).toUpperCase(); // On
récupère le nom du joueur

            String command = input.substring(3).toUpperCase(); // On
récupère la commande

            int winCount = 0; // Permet de compter le nombre de
victoires
            if (previousPlayerWhoMadeAnError != null &&
previousPlayerWhoMadeAnError.getName().equals(name)) { // Si le joueur
a déjà fait une erreur, il ne peut plus jouer ce tour
                System.out.println("Vous avez déjà fait une erreur,
vous ne pouvez plus jouer ce tour");
            }
```

```
// Si le joueur existe et qu'il n'a pas fait d'erreur, on peut jouer
            if (PlayerExists(name) && !(previousPlayerWhoMadeAnError
!= null && previousPlayerWhoMadeAnError.getName().equals(name))) {
                System.out.println();
                try { // On essaye de jouer
                    startingPodium.processInput(command); // On joue
                    if (samePosition(startingPodium, objectivePodium))
{ // Si les animaux sont à la même position, on a gagné
                        System.out.println("Bien joué! Vous avez
gagné!");
                        winCount++;
                        for (int i = 0; i < players.size(); i++) { //
On met à jour le score du joueur

                            if (players.get(i).getName().equals(name))
{ // Si le joueur existe, on met à jour son score

                                players.get(i).setScore(); // On met à
jour le score du joueur

                            }
                        }
                        System.out.println("Voulez-vous continuer a
jouer? (O/N)"); // On demande si on veut continuer à jouer
                        String answer = scanner.nextLine(); // On
récupère l'input de l'utilisateur
                        if (answer.toUpperCase().equals("O")) { // Si
on veut continuer à jouer
                            CreateCards(); // On crée des nouveaux
cartes
                            playGame(); // On relance une partie

                        } else {
                            System.out.println("Voici le score: "); //
On affiche le score
                            showScore();
                            System.exit(0); // On quitte le jeu

                        }
                    }
```

```
else { // Si les animaux ne sont pas à la même position, on a perdu
                        System.out.println("Vous avez perdu!");
                        for (int i = 0; i < players.size(); i++) { //
On met à jour le score du joueur

                            if (players.get(i).getName().equals(name))
{

                                players.get(i).setError(); // On met à
jour le nombre d'erreurs du joueur

                                if (!(previousPlayerWhoMadeAnError ==
players.get(i))) { //Si le joueur n'a pas fait d'erreur, on incrémente
le nombre d'erreurs
                                    numErrors++;
                                }
                                previousPlayerWhoMadeAnError =
players.get(i); // On met à jour le joueur qui a fait une erreur
                                startingPodium = copyOfStaringPodium;
// On remet le podium de départ à son état initial
                            }
                            if (numErrors == players.size()) { // Si
le nombre d'erreurs est égal au nombre de joueurs, on change la carte
objective
                                startingPodium = card.getRandomCard();
// On met à jour le podium de départ
                                objectivePodium =
card.getRandomCard(); // On change la carte objective
                                numErrors = 0; // On remet le nombre
d'erreurs à 0
                                previousPlayerWhoMadeAnError = null;
// On remet le joueur qui a fait une erreur à null
                                System.out.println("Carte objective a
changé");

                            }
                        }
                    }
                }
```

```
            catch (Exception e) { // Si on ne peut pas jouer, on affiche un
message d'erreur
                        System.out.println("Jouer n'existe pas");
                    }
                }
                else if (!PlayerExists(name)) { // Si le joueur n'existe
pas, on affiche un message d'erreur
                    System.out.println("Ce joueur n'existe pas");
                }
                setGameOver(); // On met à jour la variable gameOver

                if(getGameOver()) { // Si gameOver est à true, on arrête
la partie
                    finished = true;
                    System.out.println("La partie est terminée");
                    System.out.println("Voici le score: ");
                    showScore(); // On affiche le score

                }
            }
        }



    /**
     * @Brief Permet de créer les cartes de départ et d'objectif
     */
    private static void CreateCards() {
        startingPodium = card.getRandomCard(); // On crée une carte de
départ
        objectivePodium = card.getRandomCard(); // On crée une carte
d'objectif
    }


    /**
     * @breif Permet d'afficher le score
     */
    private static void showScore(){
        for (Player p : players) { // On affiche le score
            System.out.println(p.getName() + " : " + p.getScore());
        }
    }
}
```

```java
//------------Getters----------------

    /**
     * @brien Permet de savoir si la partie est terminée
     * @return gameOver
     */
    public static boolean getGameOver() {
        return gameOver; // On retourne la valeur de gameOver
    }

    //------------Setters----------------

    /**
     * @brief Permet de mettre à jour la variable gameOver
     */
    public static void setGameOver() {
        if (card.isEmpty()) { // Si le paquet de cartes est vide, on
met gameOver à true
            gameOver = true;
        }
    }
```

```java
/**
     * @brief Permet d'ajouter les joueurs
     * @param name Le nom du joueur
     * @throws IllegalArgumentException Si le nom du joueur n'est pas
composé de deux lettres
     */
    public void addPlayer(String[] name) throws
IllegalArgumentException {
        for (String s : name) { // On ajoute les joueurs
            try {
                if (s.length() != 2) { // Si le nom du joueur n'est
pas composé de deux lettres, on affiche un message d'erreur
                    throw new IllegalArgumentException("Le nom du
joueur doit être composé de deux lettres");
                } else { // Si le nom du joueur est composé de deux
lettres, on ajoute le joueur
                    Player p = new Player(s.toUpperCase());
                    players.add(p);
                    System.out.println("Le joueur " + s + " a été
ajouté.");
                    numPlayers++;
                }
            } catch (IllegalArgumentException e) { // Si on ne peut
pas ajouter le joueur, on affiche un message d'erreur
                System.out.println(e.getMessage());
                System.exit(1);
            }
        }
    }
```

```java
package Game;

import java.util.LinkedList;
import java.util.List;

public class Player {
    private List<String> players; // Liste des joueurs
    private  String currentPlayer; // Joueur courant
    private static int score; // Score du joueur
    private int error; // Nombre d'erreur du joueur
    private boolean canPlay; // Peut-il jouer?

    public Player(String player) { // Constructeur
        this.players = new LinkedList<>(); // On crée une liste de
joueurs
        this.currentPlayer = player; // On définit le joueur courant
        this.players.add(player); // On ajoute le joueur courant à la
liste des joueurs
        score = 0; // On initialise le score à 0
        error = 0; // On initialise le nombre d'erreur à 0
        canPlay = true; // On initialise le fait que le joueur peut
jouer à true
    }

    //------------Getters----------------
    /**
     * @brif permet de récupérer la liste des joueurs
     * @return the players
     */
    public List<String> getPlayers() {
        return players;
    }

    /**
     * @breif permet de récupérer le score du joueur
     * @return the score
     */
    public static int getScore() {
        return score;
    }

    /**
     * @breif permet de récupérer le nombre d'erreur du joueur
     * @return the error
     */
    public int getError() {
        return error;
    }
```

```java
/**
     * @breif permet de récupérer le fait que le joueur peut jouer
     * @return the peutJouer
     */
    public boolean canPlay() {
        return canPlay;
    }
    /**
     * @breif permet de récupérer le joueur courant
     * @return the currentPlayer
     */

    public String getName() {
        return this.currentPlayer;
    }
```

```java
    //------------Setters----------------
    /**
     * @param player Le joueur to set
     */
    public void addPlayer(String player) {
        this.players.add(player);
    }
    /**
     * @breif permet d'ajouter un point au joueur
     */
    public void setScore() {
        score++;
    }
    /**
     * @breif permet d'ajouter une erreur au joueur
     */
    public void setError() {
        error++;
    }
    /**
     * @param canPlay the peutJouer to set
     */
    public void setCanPlay(boolean canPlay) {
        this.canPlay = canPlay;
    }


    //------------Methods----------------
    @Override
    public String toString() {
        return  getPlayers().toString() +  " [points=" + getScore() +
", erreur=" + getError() + "]";
    }
}
```

```java
import Game.Game;

public class Main {
    public static void main(String[] args) {
        Game game = new Game(args); // On crée une nouvelle partie
        try {
            System.out.println("Bienvenue dans le jeu de CRAZY
CIRCUS!");
            System.out.println("A vous de jouer!");
            game.playGame();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```