BIRADEPAN Brunthaban - 111
LUTETE NTOTO Junior - 101

BUT INFO
IUT de Paris – Rives de Seine

IUT de Paris - Rives de Seine
Université Paris Cité

Projet – Bases de la programmation orientée objet

Jeu du Crazy Circus

DATE DE RENDU : LE VENDREDI 17 MARS 2023

PROFESSEURS : M. JULIEN ROSSIT et M. DENIS POITRENAUD

SOMMAIRE :

Contenu du dossier de développement applicatif :

I. PRÉSENTATION DU PROJET

1) Le rôle fonctionnel de l'application

Nous avons développé une application appelée « Crazy Circus », qui est un jeu de Dominique Ehrhard.

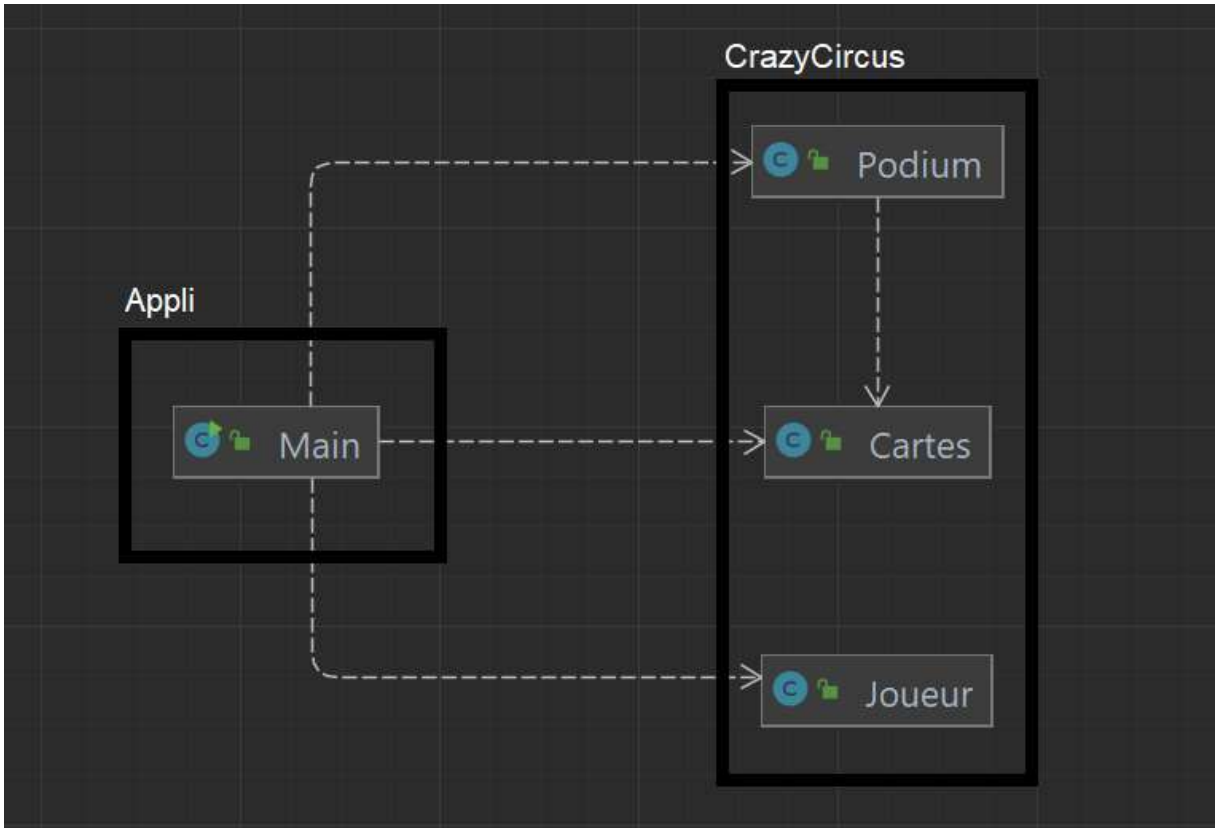
L'application prend en paramètre le nom des joueurs et en déduit le nombre de joueurs.

Le but du jeu est de trouver les ordres permettant de passer d'une configuration définie au hasard à une configuration d'arrivée défini également au hasard. Afin de se faire, les podiums contiennent des animaux qu'il faut déplacer entre les deux tours de chaque podium. Les ordres pour déplacer les animaux sont rappelés à chaque nouvelle manche.

Lorsque qu'une manche est terminée, le joueur ayant entré la bonne combinaison gagne un point et la configuration d'arrivée devient la prochaine configuration de départ.

Le programme se termine lorsque toutes les configurations de cartes sont épuisées. À ce moment-là, l'application rend en sortie le classement des joueurs ainsi que leurs scores.

2) Le diagramme UML des classes de l'application



II. PHASES DE TESTS

1) Organisation des tests unitaire de l'application

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

class MainTest {

    @org.junit.jupiter.api.Test
    void main() {

        String[] nomJoueurs = {"BB", "JN"};
        Joueur tousLesJoueurs = new Joueur(nomJoueurs);
        assertTrue(Joueur.getNbrJoueurs() == 2);

        Cartes EnsembleCarte = new Cartes();
        Cartes.initialiseCarte("C:\\Users\\brunt\\Documents\\Matiere -
Ressource\\R2.01 Programmation JAVA\\Projet Bonus\\JeuDeCarte.txt");
        assertTrue(Cartes.getJeuCarteTourBleu().size() == 24);
        assertTrue(Cartes.getJeuCarteTourRouge().size() == 24);

        Podiums Tour = new Podiums();
        Podiums.ajouterDepart(0, EnsembleCarte);
        Podiums.ajouterArrivee(0, EnsembleCarte);
        assertTrue(Cartes.getJeuCarteTourBleu().size() == 22);
        assertTrue(Cartes.getJeuCarteTourRouge().size() == 22);

        String[] mancheEnCours = ("JK KIMA").split(" ");
        String nomJoueurEnCours = mancheEnCours[0];
        String commandeEnCours = mancheEnCours[1];
        assertFalse(Joueur.getScores().containsKey(nomJoueurEnCours));

        mancheEnCours = ("BB KIMA").split(" ");
        nomJoueurEnCours = mancheEnCours[0];
        commandeEnCours = mancheEnCours[1];
        Podiums.KI();
        Podiums.MA();
        assertTrue(Podiums.verifierPodium());

assertTrue(Podiums.getPodiumsTestBleu().equals(Podiums.getPodiumsFinalBleu(
)));

assertTrue(Podiums.getPodiumsTestRouge().equals(Podiums.getPodiumsFinalRoug
e()));
    }
}
```

III. BILAN DU PROJET

1) Difficultés rencontrées

Groupe : Le Crazy Circus est un jeu qui semblait assez simple à coder mais en réalité il a été complexe.

Notamment sur le choix du type que nous devions utiliser, au début nous voulions effectuer le projet en utilisant comme type des ArrayList de String mais par la suite on a découvert que l'utilisation d'une LinkedList allait être plus utile.

L'utilisation LinkedList nous a permis d'accéder de façon plus efficace aux éléments qu'ils contiennent et le fait qu'on allait pouvoir accéder à n'importe quel élément de la liste pour pouvoir déplacer le dernier élément de la liste au sommet par exemple.

Il y a aussi eu d'autre difficulté notamment lors de la création des 24 cartes. Nous avions pensé à l'effectuer de manière dynamique.

2) Différentes améliorations possibles

On a remarqué plusieurs améliorations qu'on pouvait apporter au programme afin de le rendre plus efficace et plus logique dans la programmation objet.

- Les attributs des classes sont statiques et mal nommés. La classe Cartes ne contient que des attributs (et méthodes) statiques.
- Il y a des confusions avec des noms de classes au singulier alors que c'est un ensemble d'éléments.
- Les animaux sont représentés par des chaînes de caractères, ce qui n'est pas une bonne pratique. On aurait pu utiliser un type énuméré pour représenter les trois possibilités d'animaux.
- La méthode d'affichage contient des chaînes de caractères de type « espaces » qui compliquent le changement de nom des animaux.
- Faire une fonction qui génère les vingt-quatre cartes aléatoirement sans devoir passer par un fichier texte.

Il s'agit pour nous de notre premier projet en Java, donc la notion de programmation objet était un peu flou sans compter les normes de nommages. Par manque de temps lié aux dst, on n'a pas pu continuer ce projet comme on le souhaitait.

ANNEXES

```java
package CrazyCircus;

import java.util.*;

public class Joueur {
    //stocke les scores des joueurs et le nom des joueurs
    private static HashMap<String, Integer> scores;

    //stocke le nom des joueurs qui ont déja joué
    private static ArrayList<String> dejaJoue;

    //stocke le nombre de joueur
    private static int nbrJoueurs;

    /**
     * @param args Construit des instances d'objets joueurs en fonction des
arguments saisies dans le programme "Main"
     */
    public Joueur(String[] args) {
        nbrJoueurs = args.length;
        scores = new HashMap<String, Integer>();
        dejaJoue = new ArrayList<String>();
        for (int i = 0; i < nbrJoueurs; i++) {
            scores.put(args[i], 0);
        }
    }

    /**
     * @return le score de tout les joueurs
     */
    public static HashMap<String, Integer> getScores() {
        return scores;
    }

    /**
     * ajoute un point au joueur vainqueur de la manche
     * @param nomJoueur nom du joueur vainqueur de la manche
     */
    public static void ajouterScores(String nomJoueur) {
        scores.put(nomJoueur, scores.get(nomJoueur) + 1);
    }

    /**
     *
     * @return le nom des joueurs ayant déja joués
     */
    public static ArrayList<String> getDejaJoue() {
        return dejaJoue;
    }

    /**
     * @param nom du joueur ayant déja joué
     * ajoute le nom du joueur dans la liste des joueurs ayant déja joué
     */
    public static void setDejaJoue(String nom) {
        Joueur.dejaJoue.add(nom);
    }
```

```
    /**
     * @return le nombre de joueur
     */
    public static int getNbrJoueurs() {
        return nbrJoueurs;
    }


    /**
     * affiche le gagnant ainsi que la classement de la partie
     */
    public static void afficherGagnant() {
        // trier les entrées de la hashmap par score décroissant, avec un
tri alphabétique des noms de joueurs en cas d'égalité
        List<Map.Entry<String, Integer>> list = new
ArrayList<>(scores.entrySet());

        for (int i = 0; i < list.size(); i++) {
            for (int j = i + 1; j < list.size(); j++) {
                Map.Entry<String, Integer> entry1 = list.get(i);
                Map.Entry<String, Integer> entry2 = list.get(j);
                int score1 = entry1.getValue();
                int score2 = entry2.getValue();
                if (score2 > score1 || (score2 == score1 &&
entry2.getKey().compareTo(entry1.getKey()) < 0)) {
                    list.set(i, entry2);
                    list.set(j, entry1);
                }
            }
        }

        System.out.println("Classement:");
        int rank = 1;
        for (Map.Entry<String, Integer> entry : list) {
            System.out.println(rank++ + ". " + entry.getKey() + ": " +
entry.getValue() + " point" + (entry.getValue() <= 1 ? "" : "s"));
        }
    }
}
```

```java
package CrazyCircus;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Random;

public class Cartes {

    //contient la carte du podium bleu
    private static LinkedList<String[]> JeuCarteTourBleu;

    //contient la carte du podium rouge
    private static LinkedList<String[]> JeuCarteTourRouge;

    //génère un nombre alétoire pour tirer une carte/configuration
    private static Random rand = new Random();

    /**
     * @return la carte tirée pour le podium bleu
     */
    public static LinkedList<String[]> getJeuCarteTourBleu() {
        return JeuCarteTourBleu;
    }

    /**
     * @return la carte tirée pour le podium rouge
     */
    public static LinkedList<String[]> getJeuCarteTourRouge() {
        return JeuCarteTourRouge;
    }

    /**
     * @param fichier Créer les instances de Cartes en fonction du fichier
texte
     */
    public Cartes(String fichier) {
        //initialisation des cartes , créations de tableaux
        JeuCarteTourBleu = new LinkedList<>();
        JeuCarteTourRouge = new LinkedList<>();

        try {
            BufferedReader lecteur = new BufferedReader(new
FileReader(fichier));
            String ligne;
            boolean premiereListe = true;
            while ((ligne = lecteur.readLine()) != null) {
                if (ligne.equals("----")) {
                    premiereListe = true;
                } else {
                    String[] elements = ligne.split(",");
                    if (premiereListe) {
                        JeuCarteTourBleu.add(elements);
                        premiereListe = false;
                    } else {
                        JeuCarteTourRouge.add(elements);
                    }
                }
            }
            lecteur.close();
```

```
        } catch (IOException e) {
            System.err.println("Erreur de lecture du fichier : " +
e.getMessage());
        }

    }

    /**
     * @param index supprime la carte tirée aléatoirement grâce à son index
     */
    public static void deleteCarte(int index) {
        JeuCarteTourBleu.remove(index);
        JeuCarteTourRouge.remove(index);
    }

    /**
     * @return si les cartes sont vides
     */
    public static boolean isEmpty() {
        return JeuCarteTourBleu.size() == 0 && JeuCarteTourRouge.size() ==
0;
    }

    /**
     * génère un indice aléatoire
     * @return chiffre au hasard entre 0 et inférieur au nombre de carte
     */
    public static int aleatoire() {
        int taille = JeuCarteTourBleu.size();
        return rand.nextInt(taille);
    }
}
```

```java
package CrazyCircus;

import java.util.LinkedList;

public class Podium {

    // podium de configuration de départ
    private static LinkedList<String> podiumsBleu;
    private static LinkedList<String> podiumsRouge;

    // podium de configuration d'arrivée
    private static LinkedList<String> podiumsFinalBleu;
    private static LinkedList<String> podiumsFinalRouge;

    // copie des podiums initiaux podiums modifiés
    private static LinkedList<String> podiumsTestBleu;
    private static LinkedList<String> podiumsTestRouge;

    /**
     * @return le podium de départ bleu
     */
    public static LinkedList<String> getPodiumsBleu() {
        return podiumsBleu;
    }

    /**
     * @return le podium de départ rouge
     */
    public static LinkedList<String> getPodiumsRouge() {
        return podiumsRouge;
    }

    /**
     * @return le podium final bleu
     */
    public static LinkedList<String> getPodiumsFinalBleu() {
        return podiumsFinalBleu;
    }
    /**
     * @return le podium final rouge
     */
    public static LinkedList<String> getPodiumsFinalRouge() {
        return podiumsFinalRouge;
    }

    /**
     * @return une copie podium bleu qui va servir de test
     */
    public static LinkedList<String> getPodiumsTestBleu() {
        return podiumsTestBleu;
    }
    /**
     * @return une copie podium bleu qui va servir de test
     */
    public static LinkedList<String> getPodiumsTestRouge() {
        return podiumsTestRouge;
    }
    /**
     * Constructeur des podiums
     */
    public Podium() {
```

```
        podiumsBleu = new LinkedList<>();
        podiumsRouge = new LinkedList<>();
        podiumsFinalBleu = new LinkedList<>();
        podiumsFinalRouge = new LinkedList<>();
        podiumsTestBleu = new LinkedList<>();
        podiumsTestRouge = new LinkedList<>();
    }

    /**
     * @param index indice de la carte à ajouter dans la configuration de
départ
     * @param cartes cartes à ajouter dans la configuration de départ
     */
    public static void ajouterDepart(int index, Cartes cartes) {
        String[] podiumBleu =
cartes.getJeuCarteTourBleu().get(index)[0].split(" ");
        String[] podiumRouge =
cartes.getJeuCarteTourRouge().get(index)[0].split(" ");
        for (int i = 0; i < podiumBleu.length; i++) {
            if (podiumBleu[i].equals("") == false) {
                podiumsBleu.add(podiumBleu[i]);
                podiumsTestBleu.add(podiumBleu[i]);
            }
        }
        for (int i = 0; i < podiumRouge.length; i++) {
            if (podiumRouge[i].equals("") == false) {
                podiumsRouge.add(podiumRouge[i]);
                podiumsTestRouge.add(podiumRouge[i]);
            }
        }
        Cartes.deleteCarte(index);
    }

    /**
     * @param index indice de la carte à ajouter dans la configuration
souhaitée
     * @param cartes cartes à ajouter dans la configuration d'arrivée
souhaitée
     */
    public static void ajouterArrivee(int index, Cartes cartes) {
        String[] podiumBleu =
cartes.getJeuCarteTourBleu().get(index)[0].split(" ");
        String[] podiumRouge =
cartes.getJeuCarteTourRouge().get(index)[0].split(" ");
        for (int i = 0; i < podiumBleu.length; i++) {
            if (podiumBleu[i].equals("") == false) {
                podiumsFinalBleu.add(podiumBleu[i]);
            }
        }
        for (int i = 0; i < podiumRouge.length; i++) {
            if (podiumRouge[i].equals("") == false) {
                podiumsFinalRouge.add(podiumRouge[i]);
            }
        }
        Cartes.deleteCarte(index);
    }

    /**
     * @param original le podium de départ (bleu ou rouge)
     * @param copie le podium qui va subir les déplacements/tests (bleu ou
rouge)
```

```
     */
    public static void clone(LinkedList<String> original,
LinkedList<String> copie) {
        copie.clear();
        copie.addAll(original);
    }

    /**
     * déplace l'animal qui se trouve en haut du podium bleu au sommet du
podium rouge
     */
    public static void KI() {
        if (podiumsTestBleu.size() > 0) {
            podiumsTestRouge.addFirst(podiumsTestBleu.removeFirst());
        }
    }
    /**
     * déplace l'animal qui se trouve en haut du podium rouge au sommet du
podium bleu
     */
    public static void LO() {
        if (podiumsTestRouge.size() > 0) {
            podiumsTestBleu.addFirst(podiumsTestRouge.removeFirst());
        }
    }


    /**
     * échange les animaux qui se trouve au sommet des deux podiums
     */
    public static void SO() {
        if (podiumsTestRouge.size() > 0 && podiumsTestBleu.size() > 0) {
            String tempSommetRouge = podiumsTestRouge.removeFirst();
            String tempSommetBleu = podiumsTestBleu.removeFirst();
            podiumsTestBleu.addFirst(tempSommetRouge);
            podiumsTestRouge.addFirst(tempSommetBleu);
        }
    }
    /**
     * déplace l'animal qui se trouve en bas du podium bleu au sommet de
cette dernière
     */
    public static void NI() {
        if (podiumsTestBleu.size() > 1) {
            podiumsTestBleu.addFirst(podiumsTestBleu.removeLast());
        }
    }
    /**
     * déplace l'animal qui se trouve en bas du podium rouge au sommet de
cette dernière
     */
    public static void MA() {
        if (podiumsTestRouge.size() > 1) {
            podiumsTestRouge.addFirst(podiumsTestRouge.removeLast());
        }
    }

    /**
     * @return compare la taille des podiums qui ont subis
     * des ordres avec la configuration d'arrivée souhaité
     */
```

```
    public static boolean verifierPodium() {
        if (podiumsFinalRouge.size() != podiumsTestRouge.size() ||
podiumsFinalBleu.size() != podiumsTestBleu.size()) {
            return false;
        }

        for (int i = 0; i < podiumsFinalRouge.size(); i++) {
            if (podiumsFinalRouge.get(i).equals(podiumsTestRouge.get(i)) ==
false) {
                return false;
            }
        }

        for (int i = 0; i < podiumsFinalBleu.size(); i++) {
            if (podiumsFinalBleu.get(i).equals(podiumsTestBleu.get(i)) ==
false) {
                return false;
            }
        }

        return true;
    }

    /**
     * @return la longueur maximum entre les deux podiums de la meme
configuration
     */
    public static int longueurMax() {
        int max1 = Math.max(podiumsBleu.size(),podiumsRouge.size());
        int max2 =
Math.max(podiumsFinalBleu.size(),podiumsFinalRouge.size());
        return Math.max(max1,max2);
    }

    /**
     * affiche les podiums ainsi qu'un rappel des ordres
     */
    public static void affichage() {
        int maximum = longueurMax();
        for (int i = 0; i < maximum; i++) {
            if (i < podiumsBleu.size()) {
                if (podiumsBleu.get(i).equals("OURS")) {
                    System.out.print("   " + podiumsBleu.get(i) + "   ");
                }
                if (podiumsBleu.get(i).equals("LION")) {
                    System.out.print("   " + podiumsBleu.get(i) + "   ");
                }
                if (podiumsBleu.get(i).equals("ELEPHANT")) {
                    System.out.print(podiumsBleu.get(i));
                }
            } else {
                System.out.print("         ");
            }

            if (i < podiumsRouge.size()) {
                if (podiumsRouge.get(i).equals("OURS")) {
                    System.out.print("    " + podiumsRouge.get(i) + "
");
                } else if (podiumsRouge.get(i).equals("LION")) {
                    System.out.print("    " + podiumsRouge.get(i) + "
");
```

```
                } else if (podiumsRouge.get(i).equals("ELEPHANT")) {
                    System.out.print(" " + podiumsRouge.get(i) + "
");
                }
            } else {
                System.out.print("                ");
            }

            if (i < podiumsFinalBleu.size()) {
                if (podiumsFinalBleu.get(i).equals("OURS")) {
                    System.out.print("  " + podiumsFinalBleu.get(i) + "
");
                } else if (podiumsFinalBleu.get(i).equals("LION")) {
                    System.out.print("  " + podiumsFinalBleu.get(i) + "
");
                } else if (podiumsFinalBleu.get(i).equals("ELEPHANT")) {
                    System.out.print(podiumsFinalBleu.get(i));
                }
            } else {
                System.out.print("        ");
            }

            if (i < podiumsFinalRouge.size()) {
                if (podiumsFinalRouge.get(i).equals("OURS")) {
                    System.out.println("   " + podiumsFinalRouge.get(i));
                } else if (podiumsFinalRouge.get(i).equals("LION")) {
                    System.out.println("   " + podiumsFinalRouge.get(i));
                } else if (podiumsFinalRouge.get(i).equals("ELEPHANT")) {
                    System.out.println(" " + podiumsFinalRouge.get(i));
                }
            } else {
                System.out.println("        ");
            }
        }
        System.out.println("  ----      ----    ==>    ----      ----");
        System.out.println("  BLEU      ROUGE          BLEU      ROUGE");
        System.out.println("-----------------------------------------");
        System.out.println("KI : BLEU --> ROUGE    NI : BLEU  ^");
        System.out.println("LO : BLEU <-- ROUGE    MA : ROUGE ^");
        System.out.println("SO : BLEU <-> ROUGE");
    }
}
```

```java
import CrazyCircus.Cartes;
import CrazyCircus.Joueur;
import CrazyCircus.Podium;

import java.util.Scanner;

public class Main {
    /**
     * @param args Les paramètres reçus en ligne de commande sont le nom
des joueurs
     *             avec le quel on peut fixer le nombre de joueurs
     */
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Le nombre de joueurs est incorrect");
            return;
        }
        //contient le nombres de joueurs en fonction du nombre d'arguments
entrés dans le main
        Joueur tousJoueurs = new Joueur(args);

        Scanner scanner = new Scanner(System.in);

        Cartes EnsembleCarte = new Cartes("JeuDeCarte.txt");
        //crée une instance tour du type podium
        Podium Tour = new Podium();
        //stocke l'indice tiré au hasard
        int index = Cartes.aleatoire();
        //ajoute la carte dans la configuration de départ
        Podium.ajouterDepart(index, EnsembleCarte);

        while (Cartes.isEmpty() == false) {
            //stocke le nouvel indice tiré au hasard
            index = Cartes.aleatoire();
            // ajoute la nouvelle carte dans la configuration d'arrivée
souhaitée
            Podium.ajouterArrivee(index, EnsembleCarte);
            boolean mancheTerminee = false;

            Podium.affichage();

            while (mancheTerminee == false) {
                if (Joueur.getDejaJoue().size() == Joueur.getNbrJoueurs() -
1) {
                    for (String key : Joueur.getScores().keySet()) {
                        if (!tousJoueurs.getDejaJoue().contains(key)) {
                            mancheTerminee = true;

                            Podium.clone(Podium.getPodiumsFinalBleu(),
Podium.getPodiumsBleu());
                            Podium.clone(Podium.getPodiumsBleu(),
Podium.getPodiumsTestBleu());
                            Podium.getPodiumsFinalBleu().clear();

                            Podium.clone(Podium.getPodiumsFinalRouge(),
Podium.getPodiumsRouge());
                            Podium.clone(Podium.getPodiumsRouge(),
Podium.getPodiumsTestRouge());
                            Podium.getPodiumsFinalRouge().clear();

                            Joueur.getDejaJoue().clear();
```

```
                            Joueur.ajouterScores(key);
                            System.out.println("Le joueur " + key + " étant
le seul joueur à ne pas avoir joué, il remporte la manche");
                            break;
                        }
                    }
                } else {
                    String ligne = scanner.nextLine();
                    String[] mancheEnCours = ligne.split(" ");
                    String nomJoueurEnCours = mancheEnCours[0];
                    String commandeEnCours = mancheEnCours[1];

                    if (Joueur.getScores().containsKey(nomJoueurEnCours)) {
                        if
(Joueur.getDejaJoue().contains(nomJoueurEnCours)) {
                            System.out.println("Le joueur " +
nomJoueurEnCours + " a déjà joué");
                        } else {
                            String[] commandesTableau = new
String[(commandeEnCours.length() / 2)];

                            for (int i = 0; i < commandesTableau.length;
i++) {
                                commandesTableau[i] =
commandeEnCours.substring(i * 2, i * 2 + 2).toUpperCase();
                            }

                            boolean commandesValides = true;
                            for (String commande : commandesTableau) {
                                if (!(commande.equals("KI") ||
commande.equals("LO") || commande.equals("SO") ||
                                        commande.equals("NI") ||
commande.equals("MA"))) {
                                    commandesValides = false;
                                    break;
                                }
                            }

                            if (commandesValides) {
                                for (String commande : commandesTableau) {
                                    //permet d'identier les différents
ordres KI,LO,SO,NI,MA
                                    switch (commande) {
                                        case "KI":
                                            Podium.KI();
                                            break;
                                        case "LO":
                                            Podium.LO();
                                            break;
                                        case "SO":
                                            Podium.SO();
                                            break;
                                        case "NI":
                                            Podium.NI();
                                            break;
                                        case "MA":
                                            Podium.MA();
                                            break;
                                    }
                                }
```

```
                                if (Podium.verifierPodium()) {
                                    mancheTerminee = true;


Podium.clone(Podium.getPodiumsFinalBleu(), Podium.getPodiumsBleu());
                                Podium.clone(Podium.getPodiumsBleu(),
Podium.getPodiumsTestBleu());
                                Podium.getPodiumsFinalBleu().clear();


Podium.clone(Podium.getPodiumsFinalRouge(), Podium.getPodiumsRouge());
                                Podium.clone(Podium.getPodiumsRouge(),
Podium.getPodiumsTestRouge());
                                Podium.getPodiumsFinalRouge().clear();

                                Joueur.getDejaJoue().clear();
                                Joueur.ajouterScores(nomJoueurEnCours);
                                //le joueur a entré la bonne
combinaison
                                System.out.println("Le joueur " +
nomJoueurEnCours + " a trouvé la bonne combinaison");
                                break;
                            } else {
                                Podium.clone(Podium.getPodiumsBleu(),
Podium.getPodiumsTestBleu());
                                Podium.clone(Podium.getPodiumsRouge(),
Podium.getPodiumsTestRouge());

                                if
(!tousJoueurs.getDejaJoue().contains(nomJoueurEnCours)) {

Joueur.setDejaJoue(nomJoueurEnCours);
                                }
                                // le joueur s'est trompé de
combinaison
                                System.out.println("Le joueur " +
nomJoueurEnCours + " n'a pas fourni la bonne combinaison ");
                            }

                        } else {
                            System.out.println("Au moins une commande
saisie n'est pas valide.");
                            if
(!tousJoueurs.getDejaJoue().contains(nomJoueurEnCours)) {
                                Joueur.setDejaJoue(nomJoueurEnCours);
                            }
                        }
                    }
                } else {
                    // le joueur qui n'existe pas (n'a pas entré son
nom dans les arguments)
                    System.out.println("Le joueur " + nomJoueurEnCours
+ " n'existe pas.");
                }
            }
        }
        tousJoueurs.afficherGagnant();
    }
}
```