Rapport de projet Java

Equipe 33 : Rams Léo, Charier Marion

Table des matières

1) Présentation du projet

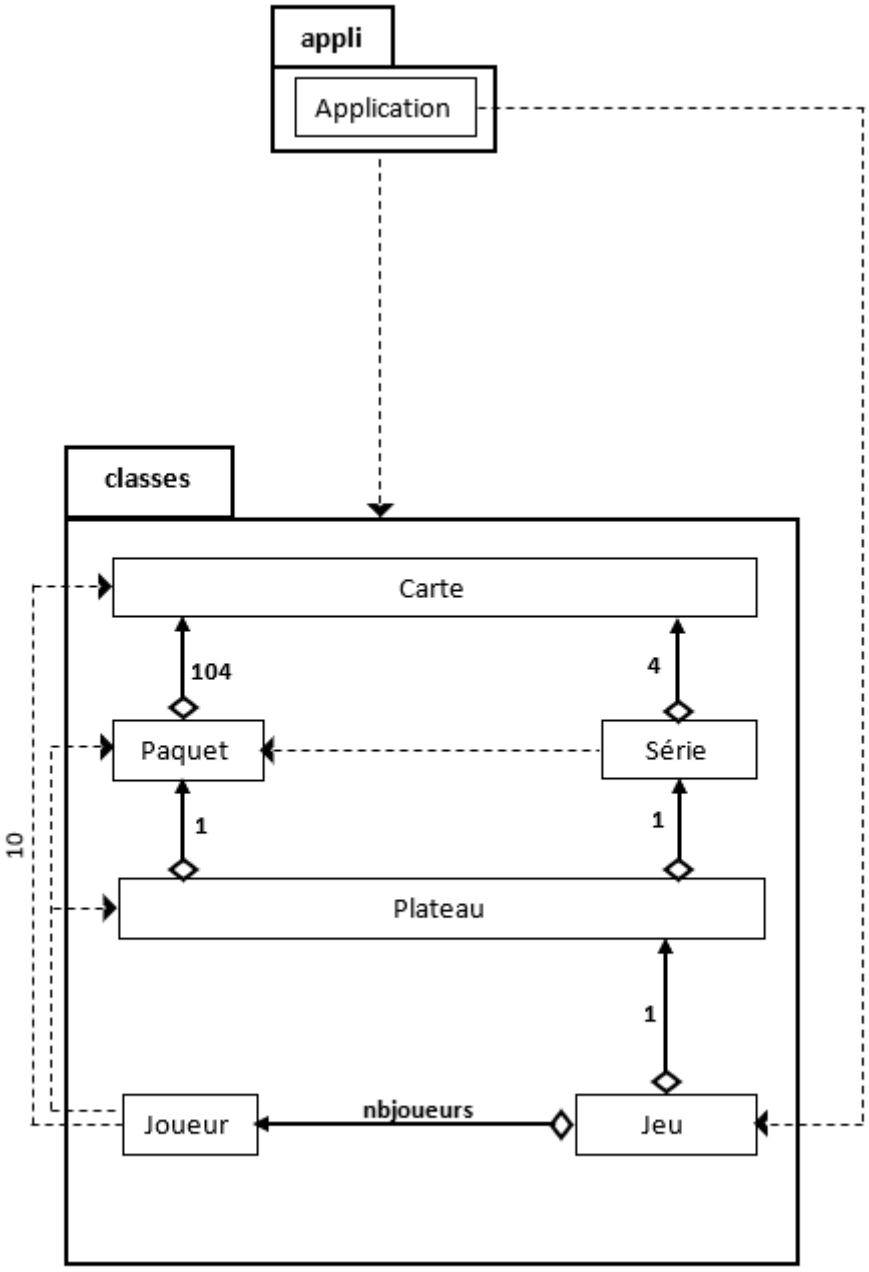
L'objectif de ce projet était de réaliser un programme permettant de jouer au jeu « 6-Qui-Prend ». Le langage que nous devions utiliser était le Java.

Le jeu du « 6-Qui-Prend » est un jeu de cartes pouvant accueillir de 2 à 10 joueurs. Au début, le paquet est composé de 104 cartes : chacune possède une valeur numérique. En fonction de cette valeur numérique, une carte possède une valeur en « têtes de bœuf ». Au début du jeu, chaque joueur reçoit un paquet de 10 cartes. Ensuite, une carte est placée dans chaque série. Chacun des joueurs doit choisir une carte parmi sa main de 10 cartes, et en fonction de la valeur numérique de la carte, elle sera placée dans une des séries ou le joueur dont c'est le tour devra ramasser les cartes d'une série au choix pour la placer.

La partie se termine lorsque tous les joueurs ont posé leurs cartes. Le but du jeu est d'accumuler le moins de têtes de bœuf, celles-ci étant indiquées sur les cartes.

Nous devions donc faire un programme qui permette d'initialiser et de jouer une partie complète de 6 qui prend. Ce programme devait être orienté objet et la structuration du code y avait une importance particulière.

2) Diagramme UML



nbjoueurs est défini à partir du fichier « config.txt »

3) Bilan du projet

Difficultés rencontrées :

Durant ce projet, nous n'avons pas rencontré beaucoup de difficultés, même si la structuration du code était parfois difficile à imaginer car il fallait l'optimiser au maximum, c'est pourquoi nous avons décidé d'opter pour cette configuration ou chaque objet en contient d'autres. Il fallait également penser à toutes les méthodes qui seraient utiles au bon déroulement du jeu sans que cela ne compromette la structuration du code. Par exemple, il fallait changer certaines données dans que celles-ci ne soient trop accessibles via d'autres classes.
La partie algorithmique cependant était très facile car il ne s'agissait pas de l'objectif principal de ce projet. Les algorithmes nécessaires ou presque tous été trouvés immédiatement et lorsque l'un d'eux posait un problème, celui-ci était résolu relativement vite, contrairement au projet en C par exemple ou certains problèmes prenaient parfois des heures à résoudre. Le fait de coder sur Eclipse a sûrement beaucoup aidé, en disant clairement où est le problème s'il est repérable par la machine.

Ce qui est réussi :

Ce projet est un travail qui nous a permis de progresser en langage Java et de comprendre mieux ce qu'est le développement orienté objet. En réalisant ce projet, nous avons pu faire l'usage de plusieurs types de fonctions mais aussi de bibliothèques existantes comme « Collections ». Le jeu du 6 qui prend est un plutôt bon choix étant donné que sa structuration n'est pas trop complexe et que la réalisation de son code était plutôt agréable. Par ailleurs, l'environnement de développement Eclipse nous a beaucoup aidé avec les différents raccourcis qu'il propose. Les erreurs sont facilement repérables, ainsi que corrigeables.
Enfin, le sujet était parfaitement clair et nous n'avons eu aucun problème à le comprendre. Le logiciel permettant de tester si son programme respecte le format voulu était également très bien fait.

Ce qui est à améliorer :

La partie algorithmique, bien qu'elle ne soit pas censée être au centre du projet, aurait pu être plus complexe ; car ici il manquait la partie résolution de problèmes qui était intéressante sur les deux premiers projets.

4) Code source

Carte.java

```
package classes;

public class Carte implements Comparable<Carte>{

	private int valeur;
	private int tetes;
	private static final int MIN = 1;
	private static final int MAX = 104;
	private static int cptcartes = MIN; // compteur permettant d'initialiser un
	paquet de cartes

	/*
	 * Trouve le nombre de têtes de boeufs que doit avoir une carte en fonction de sa
	valeur
	 *
	 * @return Le nombre de déplacements de disques effectués
	 */
	private int setTetes() {
		int nbtetes = 0;
		String chiffres = String.valueOf(this.valeur); // Transforme la valeur de
		la carte en String pour pouvoir lire ses chiffres
		// Si tous les chiffres sont identiques alors on augmente les têtes de 5
		if(chiffres.length()==2 && chiffres.charAt(0)==chiffres.charAt(1)) {
			nbtetes += 5;
		}
		// Si le dernier chiffre est un 5 alors on augmente les têtes de 2
		if(chiffres.charAt(chiffres.length()-1)=='5') {
			nbtetes += 2;
		}
		// Si le dernier chiffre est un 0 alors on augmente les têtes de 3
		else if(chiffres.charAt(chiffres.length()-1)=='0') {
			nbtetes += 3;
		}
		// Si aucune condition n'est remplie alors le nombre de têtes est de 1
		if(nbtetes==0) {
			++nbtetes;
		}
		return nbtetes;
	}

	/*
	 * Méthode d'initialisation d'une carte
	 */
	public Carte() {
		this.valeur = cptcartes++; // on initialise sa valeur grâce au compteur
		this.tetes = this.setTetes();
	}

	/*
	 * Récupère la valeur d'une carte
	 *
	 * @return La valeur de la carte
	 */
	public int getValeur() {
		return this.valeur;
	}

	/*
	 * Récupère le nombre de têtes d'une carte
```

```
	 *
	 * @return Le nombre de têtes de la carte
	 */
	public int getTetes() {
		return this.tetes;
	}

	/*
	 * Récupère la valeur minimale que peut prendre une carte
	 *
	 * @return La valeur minimale d'une carte
	 */
	public static int getMIN() {
		return MIN;
	}

	/*
	 * Récupère la valeur maximale que peut prendre une carte
	 *
	 * @return La valeur maximale d'une carte
	 */
	public static int getMAX() {
		return MAX;
	}

	/*
	 * Méthode d'affichage d'une carte qui affiche sa valeur avec son nombre de têtes
	entre parenthèses
	 */
	public String toString() {
		return (this.getValeur() + " (" + this.getTetes() + ")");
	}

	/*
	 * Méthode de comparaison de deux cartes à partir de leurs valeurs respectives
	 */
	public int compareTo(Carte c) {
		return (this.getValeur() - c.getValeur());
	}


}
```

Jeu.java

```
package classes;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

import util.Console;

public class Jeu {

	private static List<Joueur> joueurs; // ensemble des joueurs participant à la
	partie
	private static int nbjoueurs;
	private static final int JMIN = 2; // nombre minimum de joueurs dans une partie
	private static final int JMAX = 10; // nombre maximum de joueurs dans une partie
```

```
/*
 * Méthode d'initialisation du jeu
 *
 * @param nom
 *                Nom du fichier permettant de récupérer les noms des joueurs
 */
public Jeu(String nom) {
	new Paquet();
	Paquet.melanger();
	new Plateau();
	joueurs = new ArrayList<Joueur>();
	// On récupère les joueurs à partir du fichier dont le nom est donné
	try {
		Scanner in = new Scanner(new File(nom));
		while(in.hasNext()) {
			++nbjoueurs;
			// Si le nombre de joueurs est trop grand on affiche un message
			// d'erreur et on arrête le programme
			if(nbjoueurs>JMAX) {
				System.out.println("Il y a trop de joueurs");
				System.exit(0);
			}
			Joueur j = new Joueur(in.next());
			joueurs.add(j);
		}
		in.close();
	} catch (FileNotFoundException e) {
		System.out.println("Impossible d'ouvrir le fichier");
	}
	// Si le nombre de joueurs est trop petit on affiche un message
	// d'erreur et on arrête le programme
	if(nbjoueurs<JMIN) {
		System.out.println("Il n'y a pas assez de joueurs");
		System.exit(0);
	}
	// Message d'introduction du jeu
	else {
		System.out.print("Les " + nbjoueurs + " joueurs sont ");
		for(int i=0; i<nbjoueurs;++i) {
			if(i==nbjoueurs-1)
				System.out.print(" et ");
			System.out.print(joueurs.get(i));
			if(i<(nbjoueurs-2))
				System.out.print(", ");
		}
		System.out.println(". Merci de jouer à 6 qui prend !");
	}
}

/*
 * Récupère le nombre de joueurs participant à la partie
 *
 * @return Le nombre de joueurs dans la partie
 */
public static int getNbJoueurs() {
	return nbjoueurs;
}

/*
 * Permet de jouer un tour complet
 */
public static void jouerTour() {
	for(int i=0; i<nbjoueurs; ++i) {
```

```
                if(i>0)
                        // A chaque fois qu'un joueur doit jouer, on nettoie l'écran
                        Console.clearScreen();
                System.out.println("A " + joueurs.get(i) + " de jouer.");
                // Quand un joueur doit jouer, on attend qu'il se manifeste en
                appuyant sur une touche
                Console.pause();
                Plateau.afficherPlateau();
                joueurs.get(i).afficherMain();
                joueurs.get(i).carteTour();
        }
        classerJoueurs();
        Console.clearScreen();
        // Si certains joueurs ont choisi une carte dont la valeur est trop petite
        // alors on prévient que les cartes vont être posées car le joueur en
        question
        // devra choisir une carte à récupérer
        if(existeValeursTropPetites()) {
                System.out.print("Les cartes ");
                for(int i=0; i<nbjoueurs; ++i) {
                        System.out.print(joueurs.get(i).getCarteJouee().getValeur() + "
                        (" + joueurs.get(i) + ")");
                        if(i<nbjoueurs-2)
                                System.out.print(", ");
                        if(i==nbjoueurs-2)
                                System.out.print(" et ");
                }
                System.out.println(" vont être posées.");
        }
        // On joue le tour de chaque joueur
        for(int i=0; i<nbjoueurs; ++i) {
                joueurs.get(i).jouerTour();
        }
        finTour();
}

/*
 * Termine un tour
 */
private static void finTour() {
        System.out.print("Les cartes ");
        for(int i=0; i<nbjoueurs; ++i) {
                System.out.print(joueurs.get(i).getCarteJouee().getValeur() + " (" +
                joueurs.get(i) + ")");
                if(i<nbjoueurs-2)
                        System.out.print(", ");
                if(i==nbjoueurs-2)
                        System.out.print(" et ");
        }
        System.out.println(" ont été posées.");
        Plateau.afficherPlateau();
        // On affiche les têtes de boeufs ramassées par les joueurs
        boolean TetesRamassees = false;
        for(int i=0; i<nbjoueurs; ++i) {
                if(joueurs.get(i).getTetesRamassees()>0) {
                        TetesRamassees = true;
                        System.out.println(joueurs.get(i) + " a ramassé " +
        joueurs.get(i).getTetesRamassees() + " têtes de boeufs");
                }
        }
        if(!TetesRamassees)
                System.out.println("Aucun joueur ne ramasse de tête de boeufs.");
        // On termine le tour de chaque joueur
        for(int i=0; i<nbjoueurs; ++i) {
```

```
                joueurs.get(i).finTour();
            }
        }

        /*
         * Classe les joueurs en fonction de la carte qu'ils ont choisi de jouer pendant
        le tour
         */
        private static void classerJoueurs() {
            // On parcourt la liste des joueurs en échangeant les positions
            // de deux joueurs s'ils ne sont pas à leur place
            for(int i=0; i<nbjoueurs; ++i) {
                for(int k=0; k<nbjoueurs-1; ++k) {
                    // Si un joueur est censé jouer avant celui qui se trouve avant
                    lui
                    // dans la liste, alors on échange leurs positions
                    if(!(joueurs.get(k).joueAvant(joueurs.get(k+1))))
                        Collections.swap(joueurs, k, k+1);
                }
            }
        }

        /*
         * Permet de savoir si certains joueurs ont choisi de poser une carte
         * dont la valeur est trop petite pour être posée sur le plateau
         *
         * @return true si un joueur a choisi une carte dont la valeur est trop petite,
        false sinon
         */
        public static boolean existeValeursTropPetites() {
            for(int i=0; i<nbjoueurs; ++i) {
                if(Plateau.valeurTropPetite(joueurs.get(i).getCarteJouee()))
                    return true;
            }
            return false;
        }

        /*
         * Permet de jouer une partie complète
         */
        public static void Partie() {
            // Joue des tours jusqu'à ce que les joueurs n'aient plus de cartes en main
            for(int i=0; i<Joueur.getTailleMain(); ++i) {
                jouerTour();
            }
            finPartie();
        }

        /*
         * Termine la partie
         */
        public static void finPartie() {
            // Affiche les scores de tous les joueurs dans l'ordre croissant
            // En cas d'égalité, les joueurs sont classés par ordre alphabétique
            System.out.println("** Score final");
            Collections.sort(joueurs);
            for(int i=0; i<nbjoueurs; ++i)
                System.out.println(joueurs.get(i) + " a ramassé " +
                joueurs.get(i).getScore() + " têtes de boeufs");
        }

}
```

Joueur.java

```java
package classes;

import java.util.*;

public class Joueur implements Comparable<Joueur>{

	private List<Carte> main; // ensemble des cartes dans la main d'un joueur
	private int cartesMain; // nombre de cartes dans la main d'un joueur
	private int score; // nombre de têtes ramassées par le joueur au total
	private String nom;
	private Carte carteJouee; // carte que le joueur a décidé de jouer pour le tour
	en cours
	private int tetesRamassees; // nombre de têtes ramassées par le joueur durant le
	tour en cours
	private static final int tailleMain = 10; // nombre de cartes au maximum dans la
	main d'un joueur

	/*
	 * Méthode d'initialisation d'un joueur
	 *
	 * @param Nom
	 *                le nom du joueur
	 */
	public Joueur(String Nom) {
		this.score = 0;
		this.nom = Nom;
		this.tetesRamassees = 0;
		this.cartesMain = tailleMain;
		main = new ArrayList<Carte>();
		this.piocher(); // pioche jusqu'à remplir sa main
	}

	/*
	 * Permet à un joueur de piocher jusqu'à ce que sa main soit remplie
	 */
	public void piocher() {
		for(int i=0; i<tailleMain; ++i) {
			this.main.add(Paquet.piocher());
		}
		Collections.sort(this.main); // les cartes sont rangées dans l'ordre
		croissant de leurs valeurs
	}

	/*
	 * Méthode d'affichage de la main d'un joueur
	 * Affiche toutes les cartes de la main du joueur
	 */
	public void afficherMain() {
		System.out.print("- Vos cartes : ");
		for (int i=0; i<this.cartesMain; ++i) {
			if(i>0)
				System.out.print(", ");
			System.out.print(this.main.get(i));
		}
		System.out.print("\n");
	}

	/*
	 * Ajoute des points à un joueur
	 *
	 * @param points
	 *                Les points à ajouter
	 */
```

```
public void ajouterPoints(int points) {
	this.score += points;
}

/*
 * Permet à un joueur de récupérer les têtes d'une série dont l'indice est donné
 *
 * @param indice
 *		Indice de la série dont on veut récupérer les têtes
 *
 * @param c
 *		Carte que le joueur ajoute pour récupérer les têtes
 */
private void recupTetes(int indice, Carte c) {
	this.tetesRamassees = Plateau.getSerie(indice).getTetes();
	this.score += this.tetesRamassees;
	Plateau.remplacerSerie(indice, c); // on remplace les cartes de la série
	par la carte ajoutée par le joueur
}

/*
 * Permet à un joueur de récupérer une série
 *
 * @param c
 *		La carte que le joueur ajoute pour récupérer une série
 */
private void recupSerie(Carte c) {
	// Si la valeur de la carte est trop petite alors le joueur doit choisir
	une série à récupérer
	if(Plateau.valeurTropPetite(c)) {
		System.out.println("Pour poser la carte " +
		this.carteJouee.getValeur() + ", " + this.nom + " doit choisir la
		série qu'il va ramasser.");
		Plateau.afficherPlateau();
		System.out.print("Saisissez votre choix : ");
		Scanner sc = new Scanner(System.in);
		int numChoix = 1;
		boolean erreur = false;
		// Tant qu'autre chose qu'un entier est saisi ou que le numéro saisi
		est invalide,
		// le joueur doit saisir un nouveau numéro de série à récupérer
		try {
			numChoix = sc.nextInt();
			if(numChoix>Plateau.getNbSeries() || numChoix<1)
				erreur = true;
		} catch(Exception e) {
			sc.nextLine();
			erreur = true;
		}
		while(erreur) {
			System.out.print("Ce n'est pas une série valide, saisissez
			votre choix : ");
			try {
				numChoix = sc.nextInt();
				if(numChoix<=Plateau.getNbSeries() && numChoix>=1)
					erreur = false;
			} catch(Exception e) {
				sc.nextLine();
				continue;
			}
		}
		sc.close();
		// Une fois un numéro valide saisi, le joueur récupère les têtes de
la série choisie
```

```
                this.recupTetes(numChoix-1, c);
        }
        // Si la valeur de la carte n'est pas trop petite alors la série à
        récupérer est choisie automatiquement
        else {
                int indice = Plateau.getSerieOuPoser(c);
                Plateau.getSerie(indice).ajouterCarte(c);
                // les cartes de la série sont remplacées par la prochaine carte du
                paquet
                this.recupTetes(indice, Paquet.piocher());
        }
}

/*
 * Permet à un joueur d'ajouter une carte au plateau
 *
 * @param c
 *                La carte que le joueur veut ajouter
 */
public void ajouterCarte(Carte c) {
        // Si la carte peut être ajoutée alors on l'ajoute
        if(Plateau.ajoutPossible(c))
                Plateau.ajouterCarte(c);
        // Si la carte ne peut pas être ajoutée alors le joueur récupère une série
        else
                this.recupSerie(c);
        this.main.remove(this.getIndiceCarte(c.getValeur()));
        --this.cartesMain;
}

/*
 * Récupère le score total d'un joueur
 *
 * @return Le score du joueur
 */
public int getScore() {
        return this.score;
}

/*
 * Récupère l'indice d'une carte dans la main d'un joueur
 *
 * @param numero
 *                Valeur de la carte dont on veut connaître l'indice
 *
 * @return L'indice de la carte si elle est dans la main du joueur, -1 sinon
 */
public int getIndiceCarte(int numero) {
        for(int i=0; i<this.cartesMain; ++i) {
                if(numero==this.main.get(i).getValeur())
                        return i;
        }
        return -1; // retourne -1 si le joueur n'a pas la carte donnée en main
}

/*
 * Récupère une carte d'un indice donné dans la main du joueur
 *
 * @param indice
 *                Indice de la carte que l'on veut récupérer
 *
 * @return La carte dont on a donné l'indice
 */
public Carte getCarte(int indice) {
```

```
		return this.main.get(indice);
	}

	/*
	 * Récupère le nombre de cartes que peut avoir un joueur dans sa main au maximum
	 *
	 * @return Le nombre max de carte dans une main
	 */
	public static int getTailleMain() {
		return tailleMain;
	}

	/*
	 * Récupère le nombre de cartes au total dans la main d'un joueur
	 *
	 * @return Le nombre de cartes dans la main du joueur
	 */
	public int getNbCartes() {
		return this.cartesMain;
	}

	/*
	 * Permet à un joueur de choisir la carte qu'il va jouer pendant un tour
	 *
	 * @return La carte que le joueur a choisi
	 */
	private Carte choisirCarte() {
		System.out.print("Saisissez votre choix : ");
		Scanner sc = new Scanner(System.in);
		int numero = 0;
		boolean erreur = false;
		// Tant que le joueur saisit autre chose qu'un entier
		// ou qu'il n'a pas la carte saisie en main, il doit ressaisir une carte
		try {
			numero = sc.nextInt();
			if(this.getIndiceCarte(numero)==-1)
				erreur = true;
		} catch(Exception e) {
			sc.nextLine();
			erreur = true;
		}
		while(erreur) {
			System.out.print("Vous n'avez pas cette carte, saisissez votre choix
			: ");
			try {
				numero = sc.nextInt();
				if(this.getIndiceCarte(numero)!=-1)
					erreur = false;
			} catch(Exception e) {
				sc.nextLine();
				continue;
			}
		}
		sc.close();
		return this.main.get(this.getIndiceCarte(numero));
	}

	/*
	 * Attribue à un joueur une carte jouée pendant le tour en cours
	 */
	public void carteTour() {
		this.carteJouee = this.choisirCarte();
	}
```

```java
/*
 * Récupère la carte jouée par un joueur pendant le tour en cours
 *
 * @return La carte jouée par le joueur
 */
public Carte getCarteJouee() {
	return this.carteJouee;
}

/*
 * Permet à un joueur de jouer son tour en ajoutant une carte
 */
public void jouerTour() {
	// ajoute sur le plateau la carte que le joueur a choisi de jouer pendant
ce tour

this.ajouterCarte(this.main.get(this.getIndiceCarte(this.carteJouee.getValeur())
);
}

/*
 * Termine le tour d'un joueur
 */
public void finTour() {
	this.carteJouee = null;
	this.tetesRamassees = 0;
}

/*
 * Récupère le nombre de têtes ramassées par un joueur pendant le tour en cours
 *
 * @return Le nombre de têtes ramassées par le joueur
 */
public int getTetesRamassees() {
	return this.tetesRamassees;
}

/*
 * Méthode d'affichage d'un joueur
 * Affiche le nom du joueur
 */
public String toString() {
	return this.nom;
}

/*
 * Permet de savoir si la main d'un joueur est triée
 *
 * @return false si la main n'est pas triée, true sinon
 */
public boolean mainTriee() {
	for(int i=0; i<tailleMain-1; ++i) {
		// Si une valeur donnée est suivie d'une valeur inférieure à celle
		ci, alors la main n'est pas triée
		if(this.main.get(i+1).getValeur()<this.main.get(i).getValeur())
			return false;
	}
	return true;
}

/*
 * Méthode de comparaison des joueurs
 * Compare les joueurs en fonction de leur score et classe les joueurs par ordre
alphabétique en cas d'égalité
```

```
	 *
	 * @param j
	 *                Joueur qu'on compare au joueur donné
	 */
	public int compareTo(Joueur j) {
		if((this.score - j.score)==0 && !(this.nom.equals(j.nom))) {
			int i=0;
			while(this.nom.charAt(i)==j.nom.charAt(i))
				++i;
			return(this.nom.charAt(i)-j.nom.charAt(i));
		}
		return(this.score - j.score);
	}

	/*
	 * Permet de savoir si un joueur joue avant un autre joueur
	 *
	 * @param j
	 *                Joueur dont on veut savoir s'il joue après le joueur donné
	 *
	 * @return true si le premier joueur joue avant l'autre, false sinon
	 */
	public boolean joueAvant(Joueur j) {
		// Si la valeur de la carte jouée par le premier joueur est inférieure à
		// celle jouée par le deuxième joueur, alors le premier joueur joue avant
		return this.carteJouee.getValeur()<j.carteJouee.getValeur();
	}

}
```

Paquet.java

```
package classes;

import java.util.*;

public class Paquet {

	private static int nbcartes;
	private static List<Carte> paquet; // ensemble des cartes du paquet

	/*
	 * Méthode d'initialisation du paquet
	 */
	public Paquet() {
		nbcartes = Carte.getMAX(); // Le nombre de cartes correspond à la valeur
		maximale d'une carte
		paquet = new ArrayList<Carte>();
		// On ajoute toutes les cartes au paquet
		for(int i=0; i<nbcartes; ++i) {
			Carte c = new Carte();
			paquet.add(c);
		}
	}

	/*
	 * Mélange le paquet de cartes pour que les cartes soient dans le désordre
	 */
	public static void melanger() {
		Collections.shuffle(paquet);
	}

	/*
```

```
	 * Permet de piocher une carte en prenant la dernière du paquet
	 *
	 * @return La carte piochée
	 */
	public static Carte piocher() {
		--nbcartes;
		return paquet.get(nbcartes);
	}

	/*
	 * Récupère le nombre de carte restantes dans le paquet
	 *
	 * @return Le nombre de cartes
	 */
	public static int getNbCartes() {
		return nbcartes;
	}

}
```

Plateau.java

```
package classes;

import java.util.ArrayList;
import java.util.List;

public class Plateau {

	private static List<Serie> plateau; // ensemble des séries présentes sur le
plateau
	private static final int nbseries = 4;

	public Plateau() {
		plateau = new ArrayList<Serie>();
		// On initialise toutes les séries du plateau
		for(int i=0; i<nbseries; ++i) {
			Serie s = new Serie();
			plateau.add(s);
		}
	}

	/*
	 * Permet de savoir si une carte donnée peut être ajoutée à une des séries du
	plateau
	 *
	 * @param c
	 *                La carte que l'on veut ajouter
	 *
	 * @return true si la carte peut être ajoutée à une série, false sinon
	 */
	public static boolean ajoutPossible(Carte c) {
		for(int i=0; i<nbseries; ++i) {
			if(plateau.get(i).ajoutPossible(c))
				return true;
		}
		return false;
	}

	/*
	 * Permet de savoir si la valeur d'une carte donnée est trop petite
	 * pour pouvoir ajouter la carte à une des séries du plateau
	 *
```

```
	 * @param c
	 *                La carte que l'on veut ajouter
	 *
	 * @return false si la valeur de la carte n'est pas trop petite, false sinon
	 */
	public static boolean valeurTropPetite(Carte c) {
		for(int i=0; i<nbseries; ++i) {
			// Si la valeur de la carte c est plus grande que la dernière carte
			// d'une des séries alors cette valeur n'est pas trop petite
			if(c.getValeur()>plateau.get(i).getLast().getValeur())
				return false;
		}
		return true;
	}

	/*
	 * Récupère l'indice de la série ou doit être posée une carte donnée
	 *
	 * @param c
	 *                La carte que l'on veut ajouter
	 *
	 * @return l'indice de la série où poser la carte
	 */
	public static int getSerieOuPoser(Carte c) {
		int valMIN = Carte.getMAX(), difference, numChoix = 0;
		// Si la carte peut être ajoutée à une série alors
		// il faut trouver une série où la carte peut effectivement être posée
		if(ajoutPossible(c)) {
			for(int i=0; i<nbseries; ++i) {
				difference = c.getValeur()-
				plateau.get(i).getLast().getValeur();
				// On cherche la série où la différence de valeur entre la
				carte c
				// et la dernière carte de la série sera la plus faible
				if(valMIN>difference && difference>0 &&
				plateau.get(i).ajoutPossible(c)) {
					valMIN = difference;
					numChoix = i;
				}
			}
		}
		// Si la carte ne peut pas être posée on trouve la série où elle aurait dû
		être ajoutée
		else {
			for(int i=0; i<nbseries; ++i) {
				difference = c.getValeur()-
				plateau.get(i).getLast().getValeur();
				if(valMIN>difference && difference>0) {
					valMIN = difference;
					numChoix = i;
				}
			}
		}
		return numChoix;
	}

	/*
	 * Ajoute une carte donnée au plateau
	 *
	 * @param c
	 *                La carte que l'on veut poser
	 */
	public static void ajouterCarte(Carte c) {
		// On ajoute la carte à une série déterminée automatiquement
```

```
			plateau.get(getSerieOuPoser(c)).ajouterCarte(c);
		}

		/*
		 * Remplace les cartes d'une série d'un indice donné par une carte donnée
		également
		 *
		 * @param indice
		 *                Indice de la série dont on veut remplacer les cartes
		 *
		 * @param c
		 *                Carte que l'on veut mettre à la place des cartes de la série
		 */
		public static void remplacerSerie(int indice, Carte c) {
			plateau.get(indice).remplacer(c);
		}

		/*
		 * Récupère le nombre de séries présentes sur le plateau
		 *
		 * @return Le nombre de séries du plateau
		 */
		public static int getNbSeries() {
			return nbseries;
		}

		/*
		 * Récupère sur le plateau une série d'un index donné
		 *
		 * @param indice
		 *                Indice de la série qu'on veut récupérer
		 *
		 * @return La série correspondant à l'indice
		 */
		public static Serie getSerie(int indice) {
			return plateau.get(indice);
		}

		/*
		 * Méthode d'affichage du plateau
		 * Affiche toutes les séries du plateau
		 */
		public static void afficherPlateau() {
			for(int i=0; i<nbseries; ++i) {
				System.out.println(plateau.get(i));
			}
		}

}
```

Serie.java

```
package classes;

import java.util.ArrayList;
import java.util.List;

public class Serie {

	private int nbcartes;
	private List<Carte> serie; // ensemble des cartes de la série
	private int tetes; // nombre de têtes cumulées de la série
	private int numero;
```

```
private static int cptnumero = 1; // compteur permettant d'initialiser les séries
et leurs numéros
// Nombre de cartes maximum et minimum dans une série
private static final int MIN = 1;
private static final int MAX = 5;

/*
 * Méthode d'initialisation d'une série
 */
public Serie() {
	this.nbcartes = MIN;
	this.numero = cptnumero++; // On initialise son numéro à partir du compteur
	this.serie = new ArrayList<Carte>();
	this.serie.add(Paquet.piocher()); // On pioche une carte pour la mettre
dans la série
	this.tetes += serie.get(0).getTetes();
}

/*
 * Retire toutes les cartes d'une série pour y mettre une nouvelle carte
 *
 * @param c
 *                La carte à placer
 */
public void remplacer(Carte c) {
	this.serie.set(0, c); // remplace la première carte par la carte c
	this.tetes = serie.get(0).getTetes();
	// Retire toutes les autres cartes
	for(int i=this.nbcartes-1; i>0; --i) {
		this.serie.remove(i);
	}
	this.nbcartes = 1;
}

/*
 * Permet de savoir si une carte donnée peut être ajoutée ou non à la série
 *
 * @param c
 *                La carte que l'on veut ajouter
 *
 * @return true si la carte peut être ajoutée, false sinon
 */
public boolean ajoutPossible(Carte c) {
	// Si la valeur de la carte est supérieure à celle de la dernière carte de
	la série
	// et si la série a un nombre de cartes suffisamment faible alors la carte
	peut être ajoutée
	if(this.getLast().getValeur()<c.getValeur() && this.nbcartes<MAX)
		return true;
	return false;
}

/*
 * Ajoute une carte donnée à la série
 *
 * @param c
 *                La carte à ajouter
 */
public void ajouterCarte(Carte c) {
	this.serie.add(c);
	++this.nbcartes;
	this.tetes += c.getTetes();
}
```

```
	/*
	 * Récupère le nombre de cartes d'une série
	 *
	 * @return Le nombre de cartes de la série
	 */
	public int getNbCartes() {
		return this.nbcartes;
	}

	/*
	 * Récupère le nombre de têtes cumulées d'une série
	 *
	 * @return le nombre de têtes de la série
	 */
	public int getTetes() {
		return this.tetes;
	}

	/*
	 * Récupère la dernière carte d'une série
	 *
	 * @return La dernière carte de la série
	 */
	public Carte getLast() {
		return this.serie.get(nbcartes-1);
	}

	/*
	 * Récupère le nombre de cartes que peut contenir une série au maximum
	 *
	 * @return Le nombre maximum de cartes d'une série
	 */
	public static int getMAX() {
		return MAX;
	}

	/*
	 * Récupère le numéro d'une série sur la plateau
	 *
	 * @return le numéro de la série
	 */
	public int getNumero() {
		return this.numero;
	}

	/*
	 * Méthode d'affichage de la série
	 * Affiche le numéro de la série suivi de toutes ses cartes
	 */
	public String toString() {
		String s = "- série n° " + this.numero + " : ";
		for(int i=0; i<this.nbcartes; ++i) {
			if(i>0)
				s += ", ";
			s += this.serie.get(i);
		}
		return s;
	}

}
```

Main.java

```java
package main;

import classes.*;

public class Main {

	// nom du fichier permettant de récupérer les noms des joueurs
	public static String nomFichier = "config.txt";

	/*
	 * Récupère le nom du fichier permettant de récupérer les noms des joueurs
	 *
	 * @return Le nom du fichier avec les noms des joueurs
	 */
	public static String getNomFichier() {
		return nomFichier;
	}

	/*
	 * Fonction main
	 * Initialise et commence une partie
	 */
	public static void main(String args[]) {
		new Jeu(nomFichier);
		Jeu.Partie();
	}

}
```

TESTS

CarteTest.java

```java
package tests;

import static org.junit.Assert.*;

import org.junit.Test;
import classes.Carte;

public class CarteTest {

	@Test
	public void testCarte() {
		// On initialise deux cartes
		Carte c1 = new Carte();
		Carte c2 = new Carte();
		// On vérifie que la valeur de la première carte est cohérente par rapport
		à la deuxième
		assertTrue(c1.getValeur()>=Carte.getMIN());
		assertTrue(c1.getValeur()<c2.getValeur());
		// On vérifie que le nombre de têtes de la carte de valeur 1 est bien de 1
		assertEquals(c1.getTetes(),1);
	}

}
```

JeuTest.java

```java
package tests;

import static org.junit.Assert.*;
```

```java
import org.junit.Test;
import classes.*;
import main.Main;

public class JeuTest {

	@Test
	public void testJeu() {
		// Initialise une partie et vérifie que le nombre de joueurs est correct
		new Jeu(Main.getNomFichier());
		assertTrue(Jeu.getNbJoueurs()>=2);
		assertTrue(Jeu.getNbJoueurs()<=10);
	}

}
```

JoueurTest.java

```java
package tests;

import static org.junit.Assert.*;

import org.junit.Test;
import classes.Joueur;
import classes.Paquet;
import classes.Plateau;

public class JoueurTest {

	@Test
	public void test() {
		new Paquet();
		Joueur j = new Joueur("J1");
		new Plateau();
		// Vérifie que la main du joueur a été correctement triée
		assertTrue(j.mainTriee());
		j.ajouterCarte(j.getCarte(0));
		// Vérifie que le nombre de cartes du joueur a bien diminué et que son
		score est de 0
		assertEquals(j.getNbCartes(),Joueur.getTailleMain()-1);
		assertEquals(j.getScore(),0);
		// Vérifie que l'ajout de points fontionne correctement
		j.ajouterPoints(10);
		assertEquals(j.getScore(),10);
	}

}
```

PaquetTest.java

```java
package tests;

import static org.junit.Assert.*;

import org.junit.Test;
import classes.Paquet;

public class PaquetTest {

	@Test
	public void testPaquet() {
		new Paquet();
		// Vérifie que la pioche fonctionne et que les valeurs des cartes du paquet
```

```
			sont correctes
			for(int i=Paquet.getNbCartes(); i>0; --i) {
				assertEquals(Paquet.piocher().getValeur(),i);
			}
		}

}
```

PlateauTest.java

```
package tests;

import static org.junit.Assert.*;

import org.junit.Test;
import classes.Plateau;
import classes.Paquet;
import classes.Carte;

public class PlateauTest {

		@Test
		public void test() {
			new Paquet();
			Carte c1 = Paquet.piocher();
			new Plateau();
			// Vérifie que la dernière carte du paquet (104) peut être ajoutée au
			plateau
			// Comme elle a la plus grande valeur possible, elle doit pouvoir être
			ajoutée
			assertTrue(Plateau.ajoutPossible(c1));
			// Vérifie que la valeur de la carte n'est pas trop petite
			assertFalse(Plateau.valeurTropPetite(c1));
			Carte c2 = Paquet.piocher();
			// Vérifie que la série où poser la carte est la première
			assertEquals(Plateau.getSerieOuPoser(c2),0);
		}

}
```

SerieTest.java

```
package tests;

import static org.junit.Assert.*;

import org.junit.Test;
import classes.Serie;
import classes.Paquet;
import classes.Carte;

public class SerieTest {

		@Test
		public void testSerie() {
			new Paquet();
			Serie s1 = new Serie();
			Serie s2 = new Serie();
			// Vérifie que les numéros des séries sont correctement initialisés
			assertTrue(s1.getNumero()<s2.getNumero());
			Carte c = new Carte();
			// Vérifie que le remplacement de séries fonctionne bien
			s2.remplacer(c);
```

```
            assertEquals(s2.getLast(),c);
            // Vérifie que la carte de valeur 1 ne peut pas être ajoutée à la série
            assertFalse(s2.ajoutPossible(c));
        }

}
```