

Projet BPO

Crazy Circus

Lundi 11 mars 2019

Vincent Nguyen Van Nguyen

102

Tristan Deligne-Luque

108



Table des matières

[Introduction 3](#_Toc2861824)

[Diagramme UML 4](#_Toc2861825)

[Test Unitaire 5](#_Toc2861826)

[Bilan du Projet 7](#_Toc2861827)

[Annexe 8](#_Toc2861828)

[Code de l’application 8](#_Toc2861829)

[Player.java 8](#_Toc2861830)

[Animaux.java 9](#_Toc2861831)

[Carte.java 9](#_Toc2861832)

[Paquet.java 11](#_Toc2861833)

[Game.java 15](#_Toc2861834)

[Appli.java 19](#_Toc2861835)

# Introduction

Au cours de la Période C du Semestre 2, nous avons dû réaliser un projet en langage de développement Java pour développer une application reprenant le principe d’un jeu de société : Crazy Circus.

Le jeu, comportant 2 podiums ainsi que 3 animaux, consiste à reproduire une situation correspondant au placement des animaux sur les différents podiums.

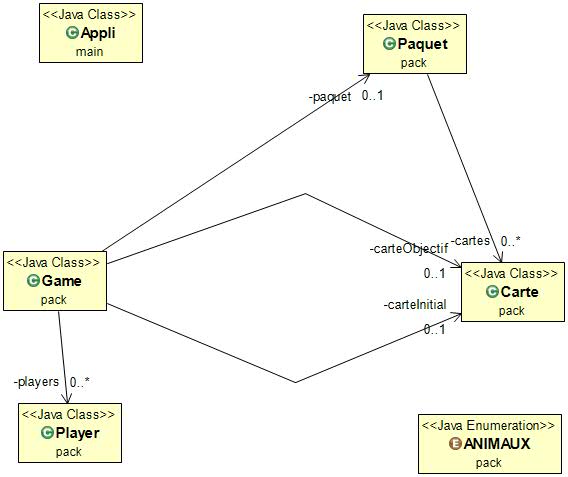
Pour se faire le jeu débute dans une situation initiale puis le joueur pioche une situation qu’il devra reproduire en utilisant 5 différents ordres distincts KI, LO, SO, NI, MA.

Pour gagner le joueur doit arriver à la situation décrite par la carte en utilisant le moins d’ordres possibles.

Pour programmer ce jeu, nous avons donc créer 6 classes qui collent à la réalité :

* **Player :** Regroupe les attributs d’un joueur comme son nombre de points accumulés, ainsi que son identifiant, son état ‘a joué’, et toutes les méthodes simples associés au jeu.
* **Animaux** : Enumère les différents animaux.
* **Carte** : Forme une carte identique au jeu réel, avec deux podiums sous forme de listes contenant les animaux. Les méthodes correspondent à la manipulation de ses cartes et à leur création une par une en implémentant les animaux dans les listes. Les méthodes permettant de déplacer les animaux à l’aide des ordres se trouvent aussi dans cette classe pour atteindre directement les listes (podiums).
* **Paquet :** Contient les 24 possibilités de cartes pouvant être piochés dans le jeu dans une liste. Les méthodes correspondent à l’utilisation réel d’un paquet, tel que piocher une carte. C’est dans cette classe que l’on a inséré manuellement les cartes du jeu pour remplir un paquet avec les bonnes situations différentes les unes des autres.
* **Game :** Permet les différentes fonctionnalités du jeu et les affichages.
* Et enfin **Appli** qui nous servira de Main où l’on pourra faire tourner une partie.

# Diagramme UML



# Test Unitaire

**package** JUnitTest.tests;

**import** org.junit.Assert;

**import** org.junit.Test;

**import** pack.ANIMAUX;

**import** pack.Carte;

**import** pack.Game;

**import** pack.Paquet;

**import** pack.Player;

**public** **class** testGame {

//PLAYER

@Test

**public** **void** testPlayerName() **throws** Exception{

Player p = **new** Player("Xavier");

Assert.*assertEquals*("Xavier", p.getId());

}

@Test

**public** **void** testPlayerScore() **throws** Exception{

Player p = **new** Player("Xavier");

Assert.*assertEquals*(0, p.getScore());

}

@Test

**public** **void** testPlayerWins() **throws** Exception{

Player p = **new** Player("Xavier");

p.hasWon();

Assert.*assertEquals*(1, p.getScore());

}

//CARTE

@Test

**public** **void** testCarte() **throws** Exception {

Carte c = **new** Carte();

c.addPodiumB(ANIMAUX.***OURS***);

c.addPodiumB(ANIMAUX.***LION***);

c.addPodiumB(ANIMAUX.***ELEPHANT***);

Carte ca = **new** Carte();

c.addPodiumB(ANIMAUX.***OURS***);

c.addPodiumB(ANIMAUX.***LION***);

c.addPodiumB(ANIMAUX.***ELEPHANT***);

Assert.*assertTrue*(c.isEqualTo(ca));

}

//PAQUET

@Test

**public** **void** testPaquet() **throws** Exception{

Paquet p = **new** Paquet();

Assert.*assertFalse*(p.isEmpty());

}

//GAME

@Test

**public** **void** testGameOver() **throws** Exception{

String[] s = {"a" , "b ", "c"};

Game g = **new** Game(s);

Assert.*assertFalse*(g.isFinished());

}

@Test

**public** **void** testGameIdentify() **throws** Exception{

String[] s = {"Xavier"};

Game g = **new** Game(s);

Assert.*assertEquals*("Xavier", g.identify("Xavier").getId());

}

}

# Bilan du Projet

Nous pouvons estimer que le projet est une réussite. En effet, l’application est fonctionnelle et l’affichage final correspond bien à celui de l’énoncé. Nous avons porté une attention particulière à la structuration du code. En effet, nous avons recommencé le projet à plusieurs reprises pour trouver des structures optimales. Nous n’avons pas hésité à commenter le code et à le simplifier au maximum pour permettre sa lecture à n’importe quel utilisateur.

Pendant un long moment, un problème nous a bloqué dans notre lancée. Nous souhaitions, au départ, créer un algorithme capable de générer les cartes automatiquement au début de la partie. Cependant, dans un souci de conception et afin d’alléger le programme, nous avons préféré rentrer toutes les possibilités de carte une par une.

De plus, nous avons pensé au temps de démarrage de l’application. En effet, générer les cartes aléatoirement en début de partie aurait fait attendre un peu plus l’utilisateur.

Ainsi, nous avons préféré ne pas nous pencher sur cet algorithme même si nous avions au départ tenté de le mettre au point.

D’autres difficultés nous ont ralenti. En effet, pour proposer des structures optimales nous avons essayé de raccourcir les méthodes au maximum, ce qui a parfois été compliqué.

De plus pour parfaire l’affichage final et l’affichage de chaque tour, il nous a fallu du temps et de la réflexion en fin de projet. Le tri et classement des joueurs ont été important aussi.

Enfin, étant donné que ce projet était notre premier en java, l’encapsulation des données était importante et nous y avions pensé à chaque étape.

Cependant, après avoir affronter ces difficultés, nous sommes arrivés à un résultat très concluant et qui nous a rendu content de notre travail et du projet donné.



# Annexe

## **Code de l’application**

### Player.java

**package** pack;

**public** **class** Player {

//id of the player

**private** String id;

//score of the player

**private** **int** score;

//player's status

**private** **boolean** hasPlayed;

//rang

**private** **int** rang;

**public** Player(String id) {

**this**.id = id;

score = 0;

hasPlayed = **false**;

rang = 0;

}

/\*\* **@return** the id of the current player \*/

**public** String getId() {

**return** id;

}

/\*\* **@return** the score of the current player \*/

**public** **int** getScore() {

**return** score;

}

/\*\* a player cannot play anymore in the round \*/

**public** **void** plays() {

hasPlayed = **true**;

}

/\*\* **@return** true if a player has already played \*/

**public** **boolean** hasPlayed() {

**return** **this**.hasPlayed;

}

/\*\* give a point to a player \*/

**public** **void** hasWon() {

score++;

}

/\*\* the player can play in the other round \*/

**public** **void** restart() {

hasPlayed = **false**;

}

/\*\* **@return** the rang of the player \*/

**public** **int** getRang() {

**return** rang;

}

}

### Animaux.java

**package** pack;

**public** **enum** ANIMAUX {

***OURS***,***ELEPHANT***,***LION***;

}

### Carte.java

**package** pack;

**import** java.util.LinkedList;

**public** **class** Carte {

**public** LinkedList<Enum<ANIMAUX>> podiumBleu;

**public** LinkedList<Enum<ANIMAUX>> podiumRouge;

**public** **static** **final** **int** ***MAX\_TAILLE*** = 3;

**public** Carte(){

podiumBleu = **new** LinkedList<>();

podiumRouge = **new** LinkedList<>();

}

/\*\* **@return** a card \*/

**public** String toString() {

**return** podiumBleu.toString() + podiumRouge.toString();

}

/\*\* **@param** i position of animal

\* **@return** animal at i in blue podium \*/

**public** Enum<ANIMAUX> getAnimalBAt(**int** i) {

**return** podiumBleu.get(i);

}

/\*\* **@param** i position of animal

\* **@return** animal at i in red podium\*/

**public** Enum<ANIMAUX> getAnimalRAt(**int** i) {

**return** podiumRouge.get(i);

}

/\*\* **@param** card c to compare

\* **@return** true if the current card if equal to the c card \*/

**public** **boolean** isEqualTo(Carte c) {

//compare the lengths

**if**(c.podiumBleu.size() != podiumBleu.size() || c.podiumRouge.size() != podiumRouge.size()) {

**return** **false**;

}

//compare in blue

**for**(**int** i = 0; i < podiumBleu.size(); i++) {

**if**(c.podiumBleu.get(i) != podiumBleu.get(i)) {

**return** **false**;

}

}

//compare in red

**for**(**int** i = 0; i < podiumRouge.size() && i < podiumRouge.size(); i++) {

**if**(c.podiumRouge.get(i) != podiumRouge.get(i)) {

**return** **false**;

}

}

**return** **true**;

}

/\*\* add an animal in blue podium \*/

**public** **void** addPodiumB(Enum<ANIMAUX> s) {

podiumBleu.add(s);

}

/\*\* add an animal in red podium \*/

**public** **void** addPodiumR(Enum<ANIMAUX> s) {

podiumRouge.add(s);

}

/\*\* set the card as a copy of another

\* **@param** c card to copy

\* **@return** copy of the card

\*/

**public** Carte setCarteAs(Carte c) {

**this**.podiumRouge = **new** LinkedList<>();

**this**.podiumBleu = **new** LinkedList<>();

**for**(Enum<ANIMAUX> s : c.podiumBleu) {

**this**.addPodiumB(s);

}

**for**(Enum<ANIMAUX> s : c.podiumRouge) {

**this**.addPodiumR(s);

}

**return** **this**;

}

//---------------------ORDERS

/\*\* move blue top to red top \*/

**public** **void** makeKI() {

**if**(!podiumBleu.isEmpty()) {

podiumRouge.add(podiumBleu.getLast());

podiumBleu.removeLast();

}

}

/\*\* move red top to blue \*/

**public** **void** makeLO() {

**if**(!podiumRouge.isEmpty()) {

podiumBleu.add(podiumRouge.getLast());

podiumRouge.removeLast();

}

}

/\*\* change blue top and red top \*/

**public** **void** makeSO() {

**if**(!podiumBleu.isEmpty() && !podiumRouge.isEmpty()) {

Enum<ANIMAUX> a = podiumBleu.getLast();

podiumBleu.removeLast();

Enum<ANIMAUX> b = podiumRouge.getLast();

podiumRouge.removeLast();

podiumBleu.add(b);

podiumRouge.add(a);

}

}

/\*\* push bottom of blue to top \*/

**public** **void** makeNI() {

**if**(!podiumBleu.isEmpty() && podiumBleu.size() > 1) {

podiumBleu.add(podiumBleu.getFirst());

podiumBleu.removeFirst();

}

}

/\*\* push bottom of red to top \*/

**public** **void** makeMA() {

**if**(!podiumRouge.isEmpty() && podiumRouge.size() > 1) {

podiumRouge.add(podiumRouge.getFirst());

podiumRouge.removeFirst();

}

}

}

### Paquet.java

package pack;

import java.util.LinkedList;

import java.util.Random;

public class Paquet {

/\*\* list of cards \*/

private LinkedList<Carte> cartes;

private Random rand;

public Paquet() {

cartes = new LinkedList<>();

createCartes();

rand = new Random();

}

/\*\* @return true if a deck is empty \*/

public boolean isEmpty() {

return cartes.isEmpty();

}

/\*\* choose a random card and take it off the deck

\* @return a new card of the deck \*/

public Carte piocherCarte() {

assert(!this.isEmpty());

Carte c = cartes.get(rand.nextInt(this.cartes.size()));

cartes.remove(c);

return c;

}

/\*\* create all the cards in the deck \*/

private void createCartes() {

Carte c1 = new Carte();

c1.podiumBleu.add(ANIMAUX.OURS);

c1.podiumBleu.add(ANIMAUX.ELEPHANT);

c1.podiumBleu.add(ANIMAUX.LION);

cartes.add(c1);

Carte c2 = new Carte();

c2.podiumBleu.add(ANIMAUX.OURS);

c2.podiumBleu.add(ANIMAUX.LION);

c2.podiumBleu.add(ANIMAUX.ELEPHANT);

cartes.add(c2);

Carte c3 = new Carte();

c3.podiumBleu.add(ANIMAUX.ELEPHANT);

c3.podiumBleu.add(ANIMAUX.OURS);

c3.podiumBleu.add(ANIMAUX.LION);

cartes.add(c3);

Carte c4 = new Carte();

c4.podiumBleu.add(ANIMAUX.ELEPHANT);

c4.podiumBleu.add(ANIMAUX.LION);

c4.podiumBleu.add(ANIMAUX.OURS);

cartes.add(c4);

Carte c5 = new Carte();

c5.podiumBleu.add(ANIMAUX.LION);

c5.podiumBleu.add(ANIMAUX.ELEPHANT);

c5.podiumBleu.add(ANIMAUX.OURS);

cartes.add(c5);

Carte c6 = new Carte();

c6.podiumBleu.add(ANIMAUX.LION);

c6.podiumBleu.add(ANIMAUX.OURS);

c6.podiumBleu.add(ANIMAUX.ELEPHANT);

cartes.add(c6);

Carte c7 = new Carte();

c7.podiumRouge.add(ANIMAUX.ELEPHANT);

c7.podiumRouge.add(ANIMAUX.OURS);

c7.podiumRouge.add(ANIMAUX.LION);

cartes.add(c7);

Carte c8 = new Carte();

c8.podiumRouge.add(ANIMAUX.ELEPHANT);

c8.podiumRouge.add(ANIMAUX.LION);

c8.podiumRouge.add(ANIMAUX.OURS);

cartes.add(c8);

Carte c9 = new Carte();

c9.podiumRouge.add(ANIMAUX.LION);

c9.podiumRouge.add(ANIMAUX.ELEPHANT);

c9.podiumRouge.add(ANIMAUX.OURS);

cartes.add(c9);

Carte c10 = new Carte();

c10.podiumRouge.add(ANIMAUX.LION);

c10.podiumRouge.add(ANIMAUX.OURS);

c10.podiumRouge.add(ANIMAUX.ELEPHANT);

cartes.add(c10);

Carte c11 = new Carte();

c11.podiumRouge.add(ANIMAUX.OURS);

c11.podiumRouge.add(ANIMAUX.ELEPHANT);

c11.podiumRouge.add(ANIMAUX.LION);

cartes.add(c11);

Carte c12 = new Carte();

c12.podiumRouge.add(ANIMAUX.OURS);

c12.podiumRouge.add(ANIMAUX.LION);

c12.podiumRouge.add(ANIMAUX.ELEPHANT);

cartes.add(c12);

Carte c13 = new Carte();

c13.podiumBleu.add(ANIMAUX.OURS);

c13.podiumBleu.add(ANIMAUX.LION);

c13.podiumRouge.add(ANIMAUX.ELEPHANT);

cartes.add(c13);

Carte c14 = new Carte();

c14.podiumRouge.add(ANIMAUX.OURS);

c14.podiumRouge.add(ANIMAUX.LION);

c14.podiumBleu.add(ANIMAUX.ELEPHANT);

cartes.add(c14);

Carte c15 = new Carte();

c15.podiumBleu.add(ANIMAUX.OURS);

c15.podiumBleu.add(ANIMAUX.ELEPHANT);

c15.podiumRouge.add(ANIMAUX.LION);

cartes.add(c15);

Carte c16 = new Carte();

c16.podiumRouge.add(ANIMAUX.OURS);

c16.podiumRouge.add(ANIMAUX.ELEPHANT);

c16.podiumBleu.add(ANIMAUX.LION);

cartes.add(c16);

Carte c17 = new Carte();

c17.podiumBleu.add(ANIMAUX.LION);

c17.podiumBleu.add(ANIMAUX.OURS);

c17.podiumRouge.add(ANIMAUX.ELEPHANT);

cartes.add(c17);

Carte c18 = new Carte();

c18.podiumRouge.add(ANIMAUX.LION);

c18.podiumRouge.add(ANIMAUX.OURS);

c18.podiumBleu.add(ANIMAUX.ELEPHANT);

cartes.add(c18);

Carte c19 = new Carte();

c19.podiumRouge.add(ANIMAUX.LION);

c19.podiumRouge.add(ANIMAUX.ELEPHANT);

c19.podiumBleu.add(ANIMAUX.OURS);

cartes.add(c19);

Carte c20 = new Carte();

c20.podiumBleu.add(ANIMAUX.LION);

c20.podiumBleu.add(ANIMAUX.ELEPHANT);

c20.podiumRouge.add(ANIMAUX.OURS);

cartes.add(c20);

Carte c21 = new Carte();

c21.podiumBleu.add(ANIMAUX.ELEPHANT);

c21.podiumBleu.add(ANIMAUX.LION);

c21.podiumRouge.add(ANIMAUX.OURS);

cartes.add(c21);

Carte c22 = new Carte();

c22.podiumRouge.add(ANIMAUX.ELEPHANT);

c22.podiumRouge.add(ANIMAUX.LION);

c22.podiumBleu.add(ANIMAUX.OURS);

cartes.add(c22);

Carte c23 = new Carte();

c23.podiumRouge.add(ANIMAUX.ELEPHANT);

c23.podiumRouge.add(ANIMAUX.OURS);

c23.podiumBleu.add(ANIMAUX.LION);

cartes.add(c23);

Carte c24 = new Carte();

c24.podiumBleu.add(ANIMAUX.ELEPHANT);

c24.podiumBleu.add(ANIMAUX.OURS);

c24.podiumRouge.add(ANIMAUX.LION);

cartes.add(c24);

}

}

### Game.java

**package** pack;

**import** java.util.ArrayList;

**public** **class** Game {

/\*\* list of players \*/

**private** ArrayList<Player> players;

**private** Paquet paquet;

**private** Carte carteInitial;

**private** Carte carteObjectif;

**private** **boolean** answerFound;

**public** Game(String[] allPlayers) {

players = **new** ArrayList<>();

registerPlayers(allPlayers);

paquet = **new** Paquet();

answerFound = **false**;

getFirstCards();

}

/\*\* Tells if the game is over \*/

**public** **boolean** isFinished() {

**return** carteInitial==**null** && paquet.isEmpty();

}

/\*\* Register players \*/

**public** **void** registerPlayers(String[] names) {

**for**(String s : names) {

Player pl = **new** Player(s);

players.add(pl);

}

}

/\*\* Identify a player with only his name \*/

**public** Player identify(String name) {

**assert**(!players.isEmpty());

**for**(Player p : players) {

**if**(p.getId().equals(name)) {

**return** p;

}

}

**return** **null**;

}

/\*\* make orders of the player and

\* **@return** a card corresponding to the player's moves \*/

**public** Carte act(String order) {

Carte c = **new** Carte();

c.setCarteAs(carteInitial);

String p = **null**;

**for**(**int** i = 0; i < order.length(); i = i + 2) {

//verify it is a two-letter word

**if**(order.length()%2 == 0) {

p = order.substring(i, i+2);

}

**else** {

p = "";

}

//make orders if called

**if**(p.equalsIgnoreCase("KI")) {

c.makeKI();

}

**if**(p.equalsIgnoreCase("LO")) {

c.makeLO();

}

**if**(p.equalsIgnoreCase("SO")) {

c.makeSO();

}

**if**(p.equalsIgnoreCase("NI")) {

c.makeNI();

}

**if**(p.equalsIgnoreCase("MA")) {

c.makeMA();

}

}

**return** c;

}

/\*\* Initialize the Initial Card and Objective Card with new cards drawn from the deck \*/

**private** **void** getFirstCards() {

**assert**(carteInitial == **null** && carteObjectif == **null**);

carteInitial = paquet.piocherCarte();

carteObjectif = paquet.piocherCarte();

}

/\*\* changes the Initial Card and Objective Card for a new round \*/

**public** **void** enchainer() {

**assert**(!isFinished());

//carteInitial becomes carteObjectif

carteInitial.setCarteAs(carteObjectif);

//carteObjectif becomes a new card from the deck

**if**(!paquet.isEmpty()) {

carteObjectif = paquet.piocherCarte();

} **else** { carteInitial = **null**; }

//all the players should not have already played in the new round

**for**(Player p : players) {

p.restart();

}

//the answer has to turn into 'not found'

answerFound = **false**;

}

/\*\* **@return** the last player of the list \*/

**public** Player getLastPlayer() {

**assert**(onePlayerLeft());

**for**(Player p : players) {

**if**(!p.hasPlayed()) {

**return** p;

}

}

**return** **null**;

}

/\*\* **@return** true if one player is left \*/

**public** **boolean** onePlayerLeft() {

**assert**(!players.isEmpty());

**int** cmpt = 0;

**for**(Player p : players) {

**if**(!p.hasPlayed()) {

cmpt++;

}

}

**return** cmpt==1;

}

/\*\* turns answerFound into true \*/

**public** **void** answerF() {

answerFound = **true**;

}

/\*\* **@return** end game display \*/

**public** StringBuilder endGame() {

**assert**(isFinished());

**this**.classement();

StringBuilder s= **new** StringBuilder();

s.append("This is the end!" + "\n" + "Voici le classement :" + "\n");

**for**(Player p : players) {

s.append(p.getId() + " : " + p.getScore() + " points\n");

}

s.append("Bravo " + players.get(0).getId() + ", vous avez gagn� la partie avec un score de "

+ players.get(0).getScore() + "!");

**return** s;

}

/\*\* **@return** the display of cards in a round\*/

**public** StringBuilder toStringBuilder() {

StringBuilder s = **new** StringBuilder();

s.append("\n");

**for**(**int** i = Carte.***MAX\_TAILLE***-1; i >=0 ; i--) {

//first colon

**if**(carteInitial.podiumBleu.size() > i) {

**if**(carteInitial.getAnimalBAt(i).equals(ANIMAUX.***ELEPHANT***)) {

s.append(carteInitial.getAnimalBAt(i) + " ");

}

**else** {

s.append(" " + carteInitial.getAnimalBAt(i) + " ");

}

}

**else** { s.append(" "); }

//second colon

**if**(carteInitial.podiumRouge.size() > i) {

**if**(carteInitial.getAnimalRAt(i).equals(ANIMAUX.***ELEPHANT***)) {

s.append(carteInitial.getAnimalRAt(i) + " ");

}

**else** {

s.append(" " + carteInitial.getAnimalRAt(i) + " ");

}

}

**else** { s.append( " "); }

//third colon

**if**(carteObjectif.podiumBleu.size() > i) {

**if**(carteObjectif.getAnimalBAt(i).equals(ANIMAUX.***ELEPHANT***)) {

s.append(carteObjectif.getAnimalBAt(i) + " ");

}

**else** {

s.append(" " + carteObjectif.getAnimalBAt(i) + " ");

}

}

**else** { s.append( " "); }

//fourth colon

**if**(carteObjectif.podiumRouge.size() > i) {

**if**(carteObjectif.getAnimalRAt(i).equals(ANIMAUX.***ELEPHANT***))

s.append(carteObjectif.getAnimalRAt(i));

**else**

s.append(" " + carteObjectif.getAnimalRAt(i));

}

s.append("\n");

}

s.append(" ---- ---- ==> ---- ----\n"

+ " BLEU ROUGE BLEU ROUGE\n"

+ "--------------------------------------------"

+ "\nKI : BLEU --> ROUGE NI : BLEU �\r\n" +

"LO : BLEU <-- ROUGE MA : ROUGE �\r\n" +

"SO : BLEU <-> ROUGE\n");

**return** s;

}

/\*\* **@return** Initial Card \*/

**public** Carte getCarteInitial() {

**return** carteInitial;

}

/\*\* **@return** Objective Card \*/

**public** Carte getCarteObjectif() {

**return** carteObjectif;

}

/\*\* **@return** true if answer is found during a round \*/

**public** **boolean** isAnswerFound() {

**return** answerFound;

}

/\*\* trie les joueurs \*/

**private** **void** classement() {

**for**(**int** i = 0; i < players.size(); i++) {

**for**(**int** y = i + 1; y < players.size(); y++) {

**if**(players.get(i).getScore() < players.get(y).getScore()) {

Player p = players.get(y);

players.remove(y);

players.add(i, p);

}

//trie par ordre alphab�tique

**if**(players.get(i).getScore() == players.get(y).getScore()) {

**if**(players.get(y).getId().compareTo(players.get(i).getId()) < 0){

Player p = players.get(y);

players.remove(y);

players.add(i, p);

}

}

}

}

}

}

### Appli.java

package main;

import java.util.Scanner;

import pack.Game;

import pack.Player;

public class Appli {

public static void main(String[] args) {

Game game = new Game(args);

Scanner sc = new Scanner(System.in);

int nbreTours = 1;

while(!game.isFinished()) {

System.out.println("Tour num�ro : " + nbreTours + "\n" + game.toStringBuilder());

while(!game.isAnswerFound()) {

sc = new Scanner(System.in);

String name = sc.next();

String order = sc.next();

//if it corresponds to a player

if(game.identify(name) != null){

//define the player who is trying to win

Player p = game.identify(name);

//if the player has not already played

if(!p.hasPlayed()){

p.plays();

if(game.getCarteObjectif().isEqualTo(game.act(order))){

p.hasWon();

game.answerF();

System.out.println(p.getId() + " has won 1 point \n");

}

else { System.out.println("wrong orders"); }

}

else { System.out.println("player has already played"); }

}

else { System.out.println("player does not exist"); }

if(game.onePlayerLeft() && !game.isAnswerFound()) {

game.getLastPlayer().hasWon();

game.answerF();

System.out.println(game.getLastPlayer().getId() + " has won 1 point for not having played \n");

}

}

nbreTours++;

game.enchainer();

}

sc.close();

System.out.println(game.endGame());

}

}