Projet de programmation
Lilian DAMIEN – 1A
Année 2020 – 2021

M2107 - Projet de programmation

Cahier de conception

Les Bâtisseurs : Moyen-Âge

IUT Vannes
8 Rue Montaigne - 56017 VANNES Cedex
02 97 62 64 64 - www.iutvannes.fr

Tables des matières

1) Diagramme de classes d'analyse

Le diagramme de classes d’analyse, nous permet de comprendre facilement et rapidement le fonctionnement général du projet.

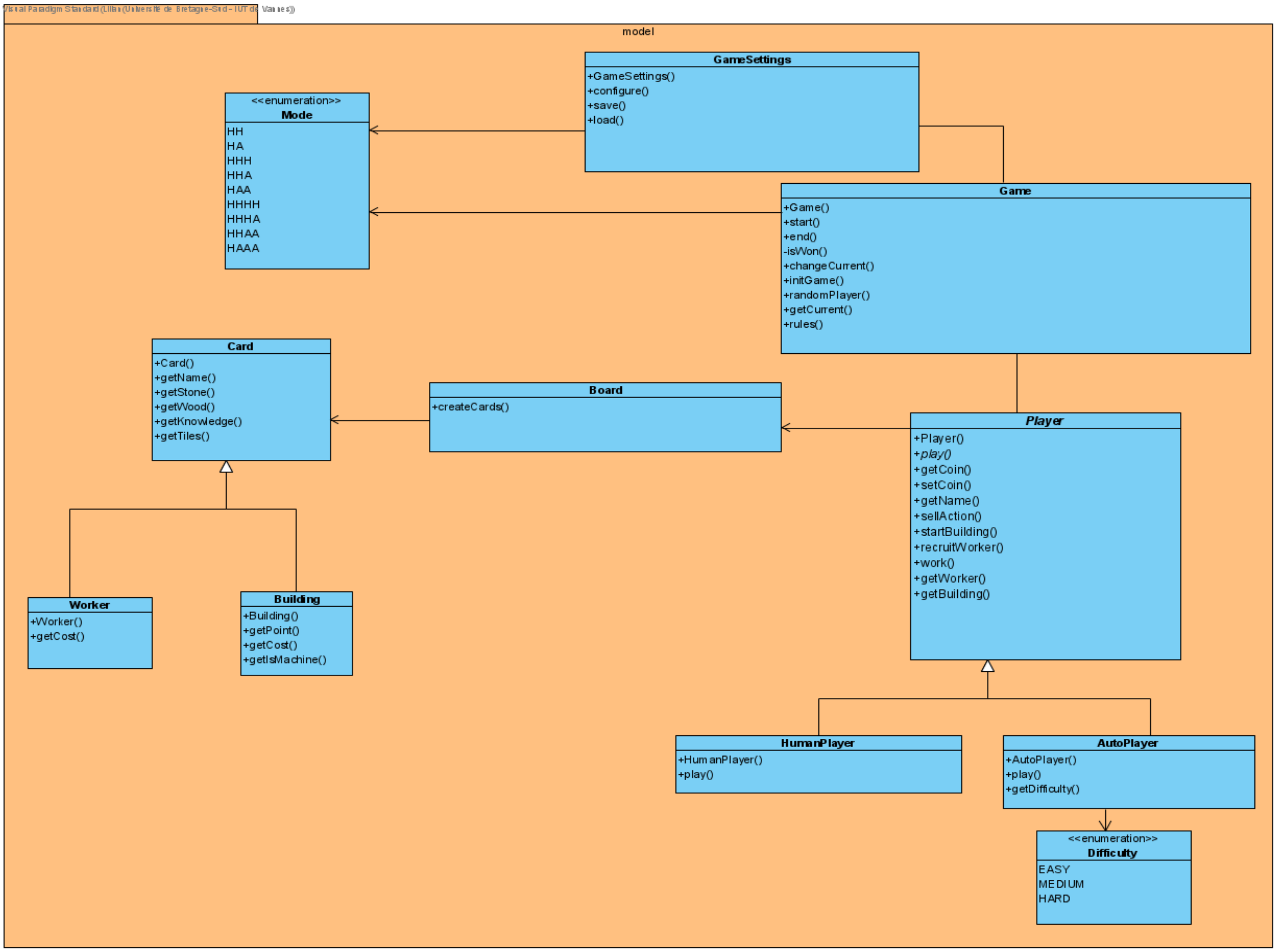
GameSettings est la classe permettant de lire les fichiers de configuration, sauvegarder ou reprendre une partie.

Game est la classe qui va initialiser une partie, c'est la classe principale du jeu.

Player est une classe abstraite et une super classe de ***HumanPlayer*** et ***AutoPlayer***, elle possède toutes les méthodes de jeu (actions).

Board est la classe qui va créer un jeu de carte et les afficher sur le plateau

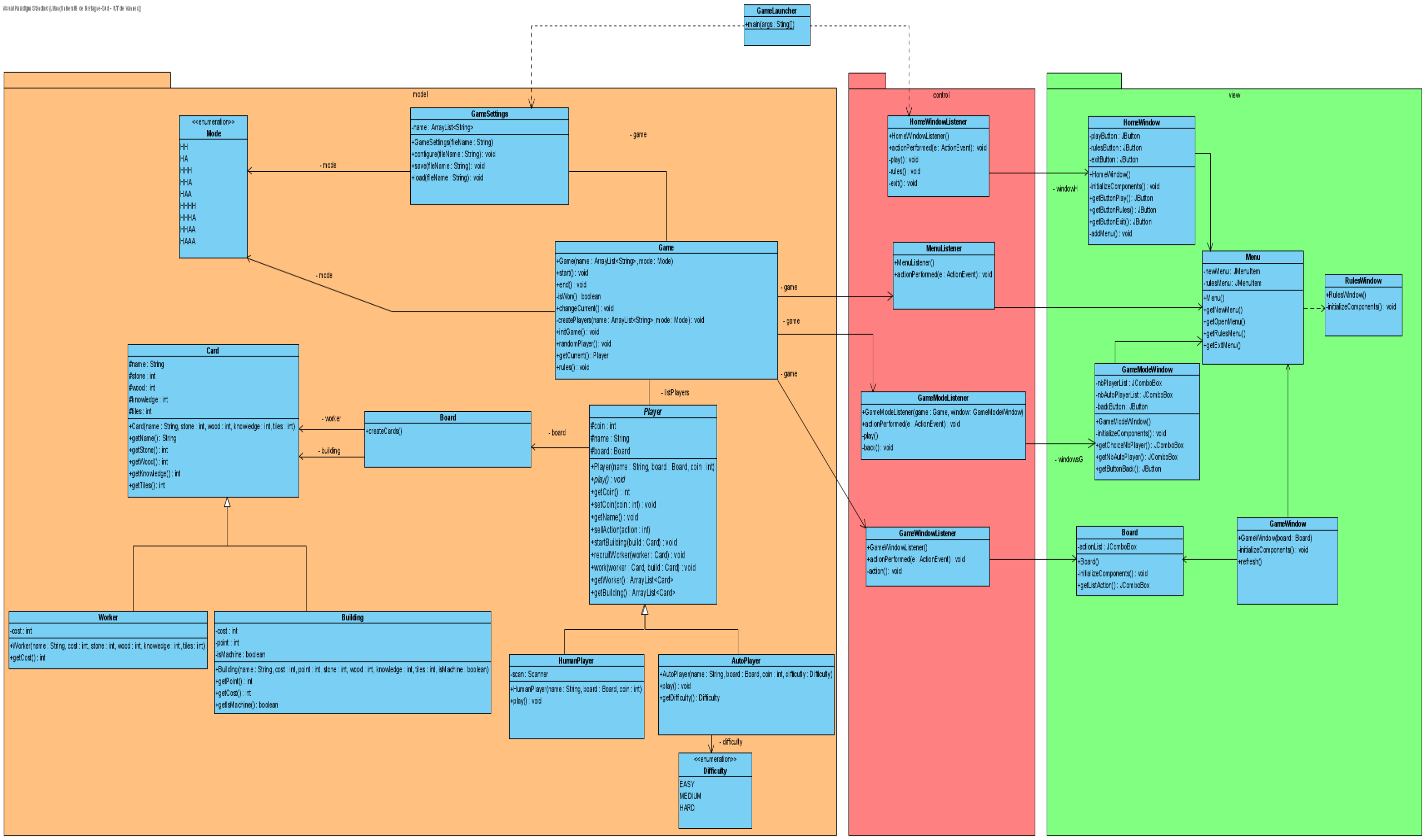
Card est la super classe de ***Worker*** et ***Builder*** elle représente chaque carte lu dans les fichiers de configuration

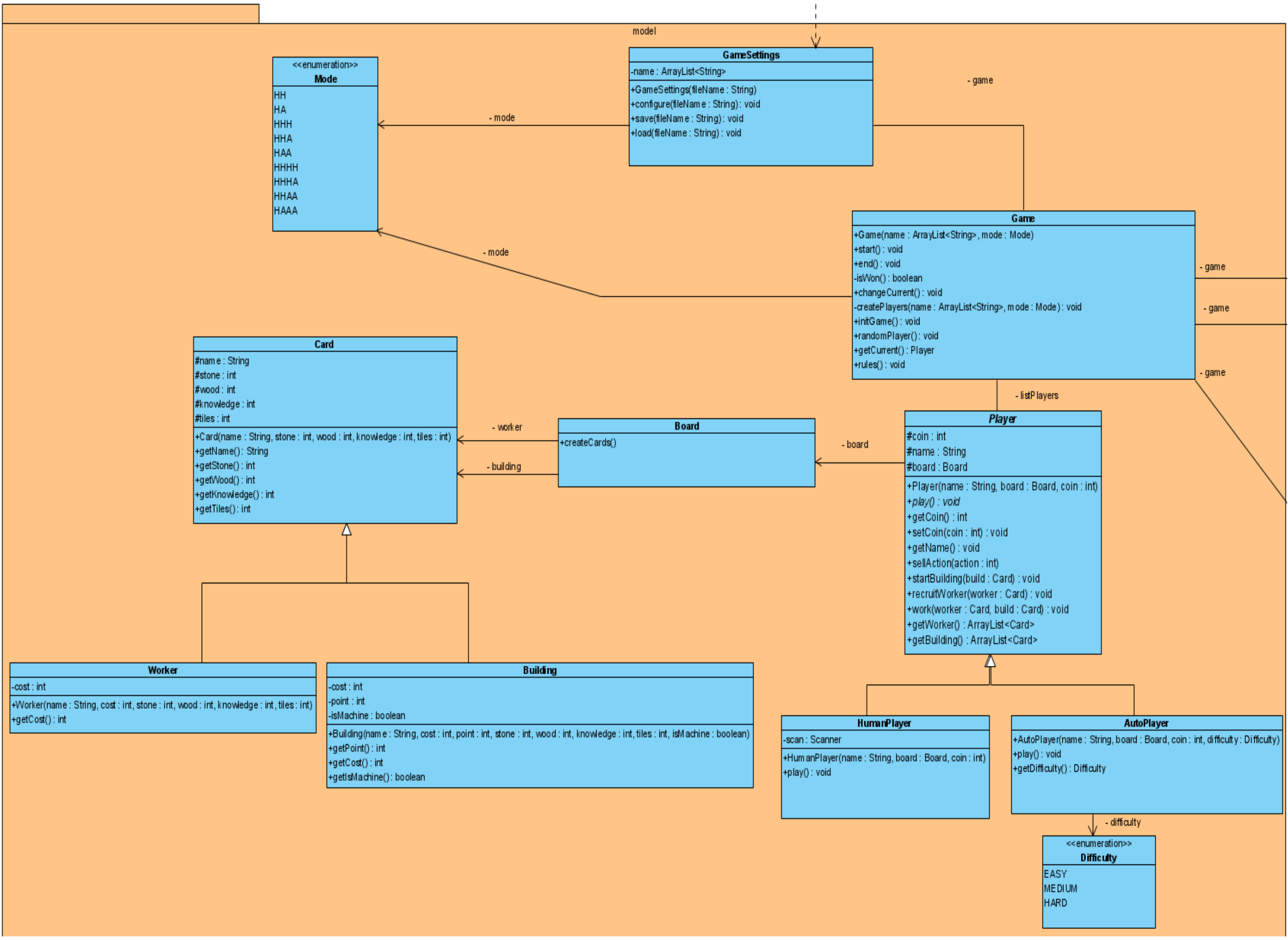


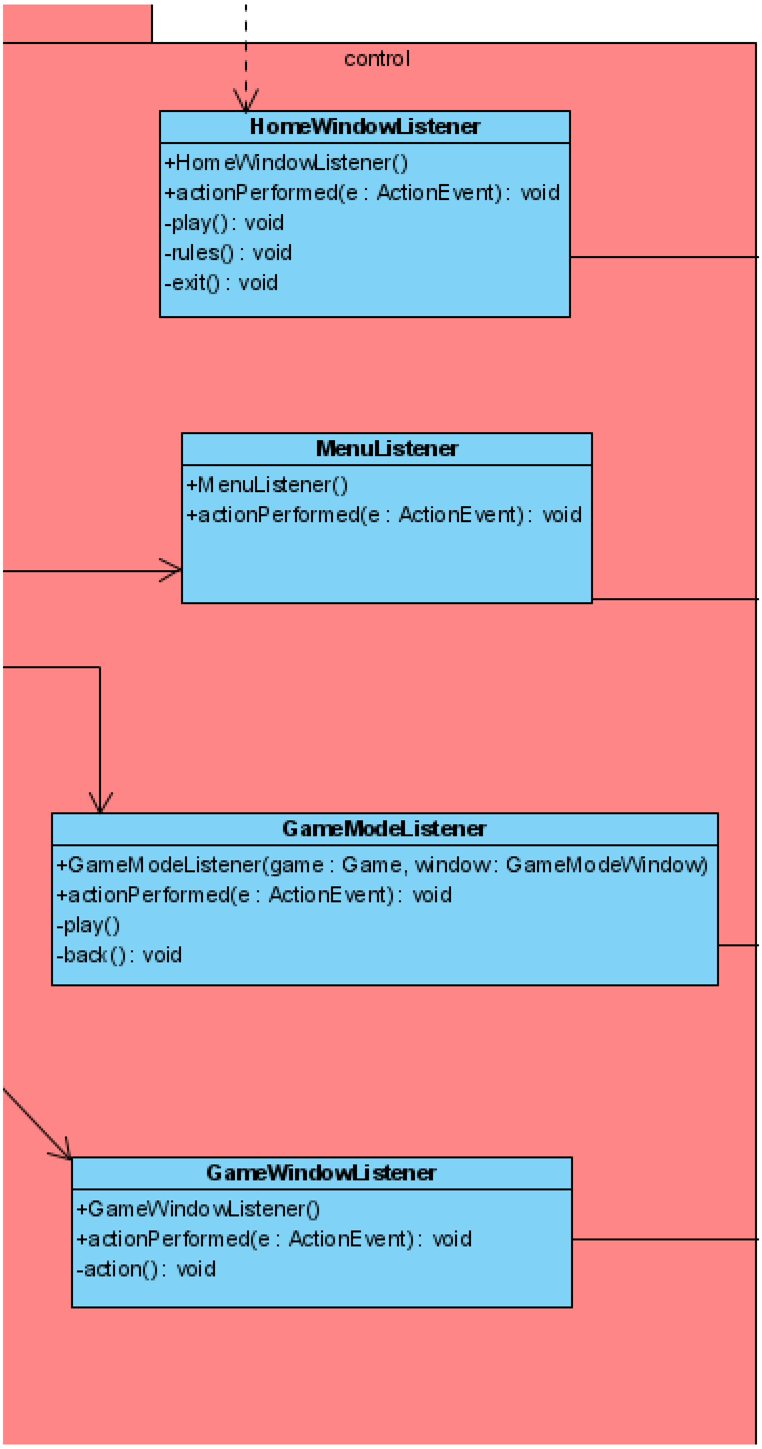
2) Diagramme de classes de conception

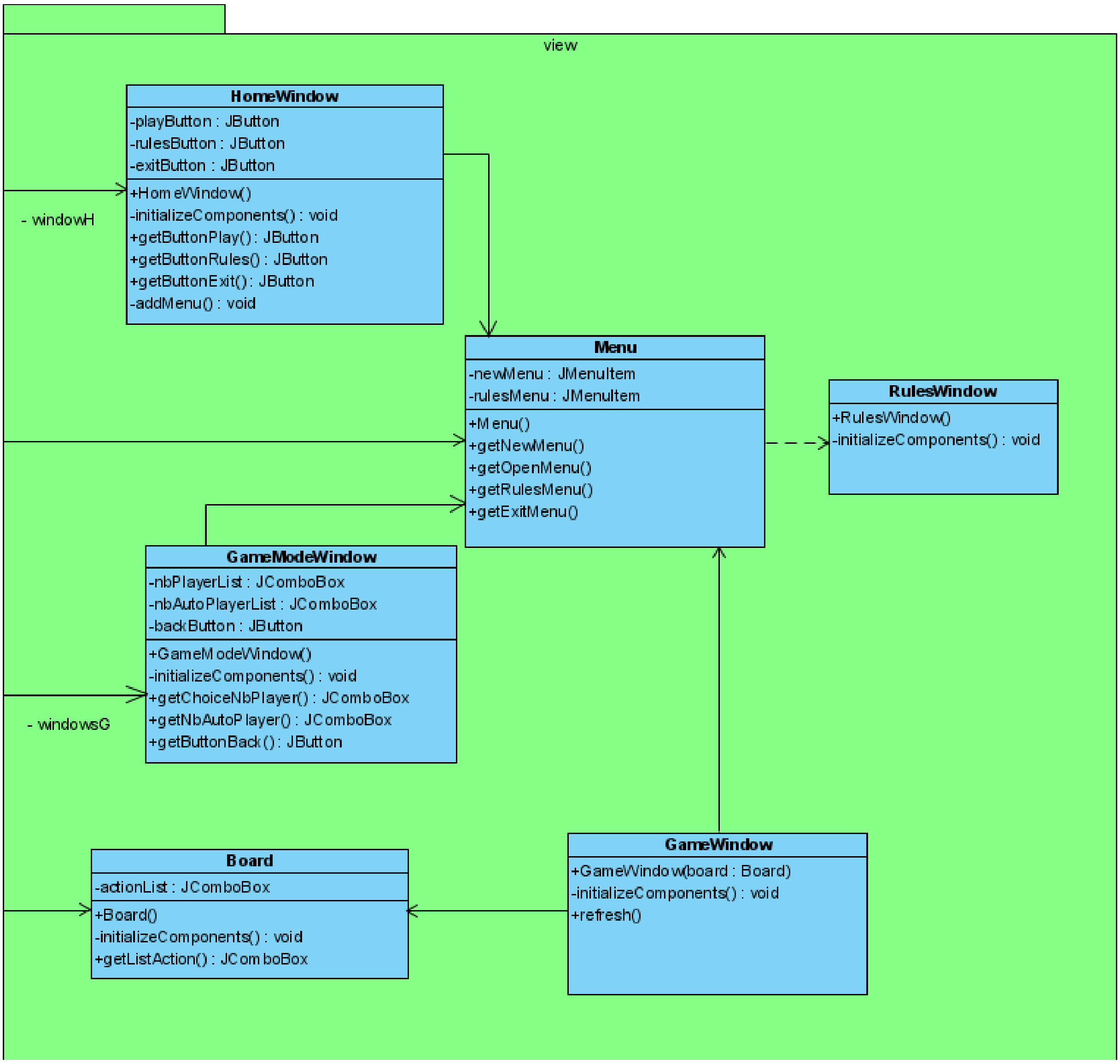
Ce diagramme de classes de conception est fait de manière a pouvoir ajouté ou modifiés certaines fonctionnalités, cependant ce diagramme essaye de s'approcher le plus possible de la conception finale de l'application. Ce diagramme est basé sur le modèle MVC et donc contient 3 packages principaux qui sont :

- le package **model** qui contiendra les classes du jeu
- le package **view** qui contiendra les classes graphiques du jeu
- le package **control** qui contiendra les classes d'interaction entre le package model et le package view


Visual Paradigm Standard(Lilian(Université de Bretagne-Sud - IUT de Vannes))
GameLauncher
+main(args : Sting[])
model
<<enumeration>>
Mode
HH
HA
HHH
HHA
HAA
HHHH
HHHA
HHAA
HAAA
- mode
GameSettings
-name : ArrayList<String>
+GameSettings(fileName : String)
+configure(fileName : String) : void
+save(fileName : String) : void
+load(fileName : String) : void
- game
- mode
Game
+Game(name : ArrayList<String>, mode : Mode)
+start() : void
+end() : void
-isWon() : boolean
+changeCurrent() : void
-createPlayers(name : ArrayList<String>, mode : Mode) : void
+initGame() : void
+randomPlayer() : void
+getCurrent() : Player
+rules() : void
- game
- game
- game
- listPlayers
Card
#name : String
#stone : int
#wood : int
#knowledge : int
#tiles : int
+Card(name : String, stone : int, wood : int, knowledge : int, tiles : int)
+getName() : String
+getStone() : int
+getWood() : int
+getKnowledge() : int
+getTiles() : int
- worker
- building
Board
+createCards()
- board
Player
#coin : int
#name : String
#board : Board
+Player(name : String, board : Board, coin : int)
+play() : void
+getCoin() : int
+setCoin(coin : int) : void
+getName() : void
+sellAction(action : int)
+startBuilding(build : Card) : void
+recruitWorker(worker : Card) : void
+work(worker : Card, build : Card) : void
+getWorker() : ArrayList<Card>
+getBuilding() : ArrayList<Card>
Worker
-cost : int
+Worker(name : String, cost : int, stone : int, wood : int, knowledge : int, tiles : int)
+getCost() : int
Building
-cost : int
-point : int
-isMachine : boolean
+Building(name : String, cost : int, point : int, stone : int, wood : int, knowledge : int, tiles : int, isMachine : boolean)
+getPoint() : int
+getCost() : int
+getIsMachine() : boolean
HumanPlayer
-scan : Scanner
+HumanPlayer(name : String, board : Board, coin : int)
+play() : void
AutoPlayer
+AutoPlayer(name : String, board : Board, coin : int, difficulty : Difficulty)
+play() : void
+getDifficulty() : Difficulty
- difficulty
<<enumeration>>
Difficulty
EASY
MEDIUM
HARD
control
HomeWindowListener
+HomeWindowListener()
+actionPerformed(e : ActionEvent) : void
-play() : void
-rules() : void
-exit() : void
MenuListener
+MenuListener()
+actionPerformed(e : ActionEvent) : void
GameModeListener
+GameModeListener(game : Game, window : GameModeWindow)
+actionPerformed(e : ActionEvent) : void
-play()
-back() : void
GameWindowListener
+GameWindowListener()
+actionPerformed(e : ActionEvent) : void
-action() : void
view
- windowH
HomeWindow
-playButton : JButton
-rulesButton : JButton
-exitButton : JButton
+HomeWindow()
-initializeComponents() : void
+getButtonPlay() : JButton
+getButtonRules() : JButton
+getButtonExit() : JButton
-addMenu() : void
Menu
-newMenu : JMenuItem
-rulesMenu : JMenuItem
+Menu()
+getNewMenu()
+getOpenMenu()
+getRulesMenu()
+getExitMenu()
RulesWindow
+RulesWindow()
-initializeComponents() : void
GameModeWindow
-nbPlayerList : JComboBox
-nbAutoPlayerList : JComboBox
-backButton : JButton
+GameModeWindow()
-initializeComponents() : void
+getChoiceNbPlayer() : JComboBox
+getNbAutoPlayer() : JComboBox
+getButtonBack() : JButton
- windowsG
Board
-actionList : JComboBox
+Board()
-initializeComponents() : void
+getListAction() : JComboBox
GameWindow
+GameWindow(board : Board)
-initializeComponents() : void
+refresh()

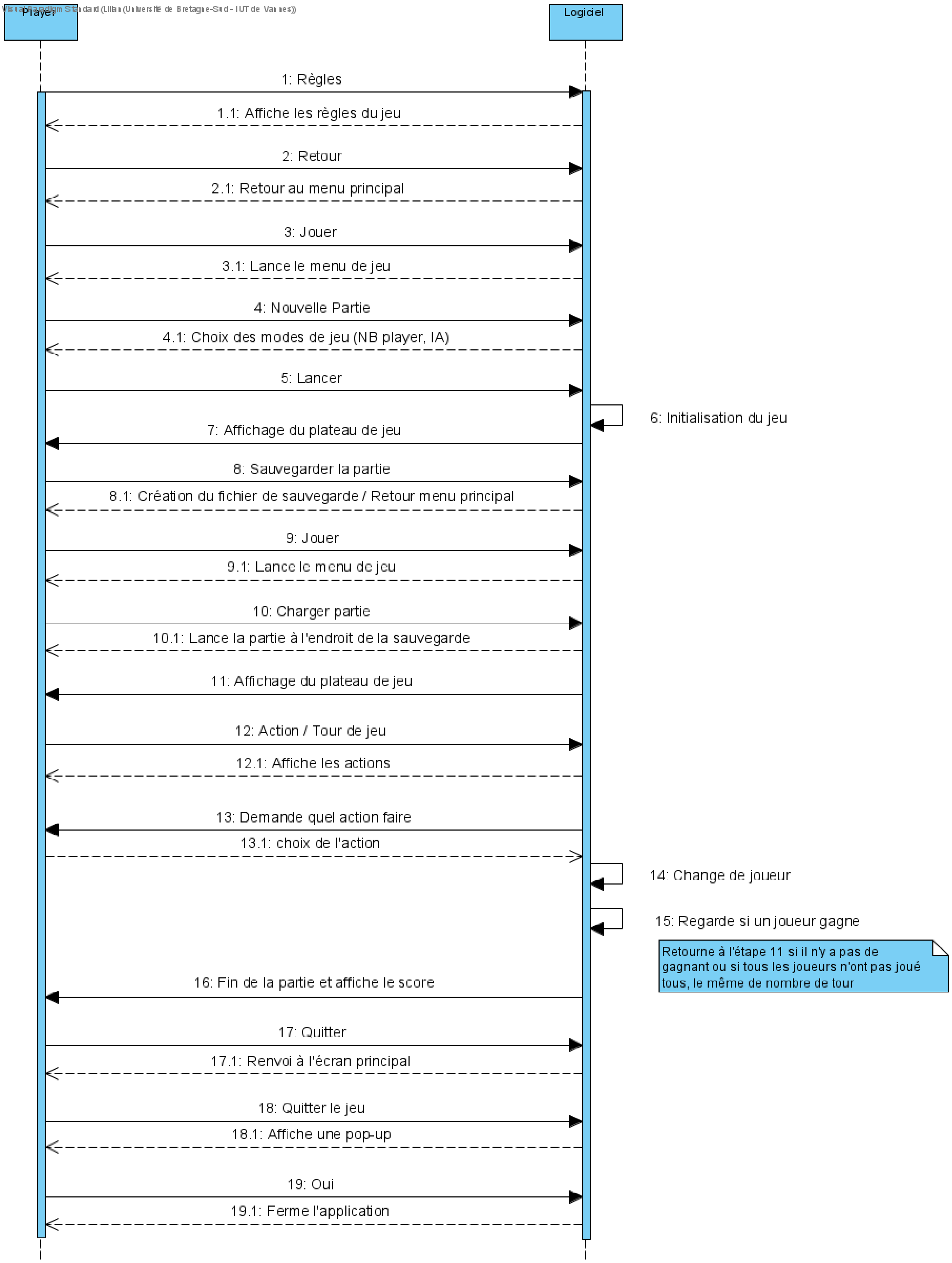

model
<<enumeration>>
Mode
HH
HA
HHH
HHA
HAA
HHHH
HHHA
HHAA
HAAA
- mode
GameSettings
-name : ArrayList<String>
+GameSettings(fileName : String)
+configure(fileName : String) : void
+save(fileName : String) : void
+load(fileName : String) : void
- game
- mode
Game
+Game(name : ArrayList<String>, mode : Mode)
+start() : void
+end() : void
-isWon() : boolean
+changeCurrent() : void
-createPlayers(name : ArrayList<String>, mode : Mode) : void
+initGame() : void
+randomPlayer() : void
+getCurrent() : Player
+rules() : void
- game
- game
- game
- listPlayers
Card
#name : String
#stone : int
#wood : int
#knowledge : int
#tiles : int
+Card(name : String, stone : int, wood : int, knowledge : int, tiles : int)
+getName() : String
+getStone() : int
+getWood() : int
+getKnowledge() : int
+getTiles() : int
- worker
- building
Board
+createCards()
- board
Player
#coin : int
#name : String
#board : Board
+Player(name : String, board : Board, coin : int)
+play() : void
+getCoin() : int
+setCoin(coin : int) : void
+getName() : void
+sellAction(action : int)
+startBuilding(build : Card) : void
+recruitWorker(worker : Card) : void
+work(worker : Card, build : Card) : void
+getWorker() : ArrayList<Card>
+getBuilding() : ArrayList<Card>
Worker
-cost : int
+Worker(name : String, cost : int, stone : int, wood : int, knowledge : int, tiles : int)
+getCost() : int
Building
-cost : int
-point : int
-isMachine : boolean
+Building(name : String, cost : int, point : int, stone : int, wood : int, knowledge : int, tiles : int, isMachine : boolean)
+getPoint() : int
+getCost() : int
+getIsMachine() : boolean
HumanPlayer
-scan : Scanner
+HumanPlayer(name : String, board : Board, coin : int)
+play() : void
AutoPlayer
+AutoPlayer(name : String, board : Board, coin : int, difficulty : Difficulty)
+play() : void
+getDifficulty() : Difficulty
- difficulty
<<enumeration>>
Difficulty
EASY
MEDIUM
HARD


control
HomeWindowListener
+Hom eWindowListener()
+actionPerformed(e : ActionEvent) : void
-play() : void
-rules() : void
-exit() : void
MenuListener
+M enuListener()
+actionPerformed(e : ActionEvent) : void
GameModeListener
+GameM odeListener(game : Game, window: GameModeWindow)
+actionPerformed(e : ActionEvent) : void
-play()
-back() : void
GameWindowListener
+GameWindo wListener()
+actionPerformed(e : ActionEvent) : void
-action() : void


view
HomeWindow
-playButton : JButton
-rulesButton : JButton
-exitButton : JButton
+HomeWindow()
-initializeComponents() : void
+getButtonPlay() : JButton
+getButtonRules() : JButton
+getButtonExit() : JButton
-addMenu() : void
- windowH
Menu
-newMenu : JMenuItem
-rulesMenu : JMenuItem
+Menu()
+getNewMenu()
+getOpenMenu()
+getRulesMenu()
+getExitMenu()
RulesWindow
+RulesWindow()
-initializeComponents() : void
GameModeWindow
-nbPlayerList : JComboBox
-nbAutoPlayerList : JComboBox
-backButton : JButton
+GameModeWindow()
-initializeComponents() : void
+getChoiceNbPlayer() : JComboBox
+getNbAutoPlayer() : JComboBox
+getButtonBack() : JButton
- windowsG
Board
-actionList : JComboBox
+Board()
-initializeComponents() : void
+getListAction() : JComboBox
GameWindow
+GameWindow(board : Board)
-initializeComponents() : void
+refresh()

3) Diagramme de séquence de boîte noire

Le diagramme de séquence de boîte noire illustre le déroulement d'un test effectué par un utilisateur qui ne connaît pas le fonctionnement de l'application, et donc fais le test de chaque fonctionnalité afin de s'assurer que l'application ne contiennent aucun bug et afin de comprendre son utilisation.



Player
Logiciel
1: Règles
1.1: Affiche les règles du jeu
2: Retour
2.1: Retour au menu principal
3: Jouer
3.1: Lance le menu de jeu
4: Nouvelle Partie
4.1: Choix des modes de jeu (NB player, IA)
5: Lancer
6: Initialisation du jeu
7: Affichage du plateau de jeu
8: Sauvegarder la partie
8.1: Création du fichier de sauvegarde / Retour menu principal
9: Jouer
9.1: Lance le menu de jeu
10: Charger partie
10.1: Lance la partie à l'endroit de la sauvegarde
11: Affichage du plateau de jeu
12: Action / Tour de jeu
12.1: Affiche les actions
13: Demande quel action faire
13.1: choix de l'action
14: Change de joueur
15: Regarde si un joueur gagne
Retourne à l'étape 11 si il n'y a pas de gagnant ou si tous les joueurs n'ont pas joué tous, le même de nombre de tour
16: Fin de la partie et affiche le score
17: Quitter
17.1: Renvoi à l'écran principal
18: Quitter le jeu
18.1: Affiche une pop-up
19: Oui
19.1: Ferme l'application

4) Spécifications des formats de fichier

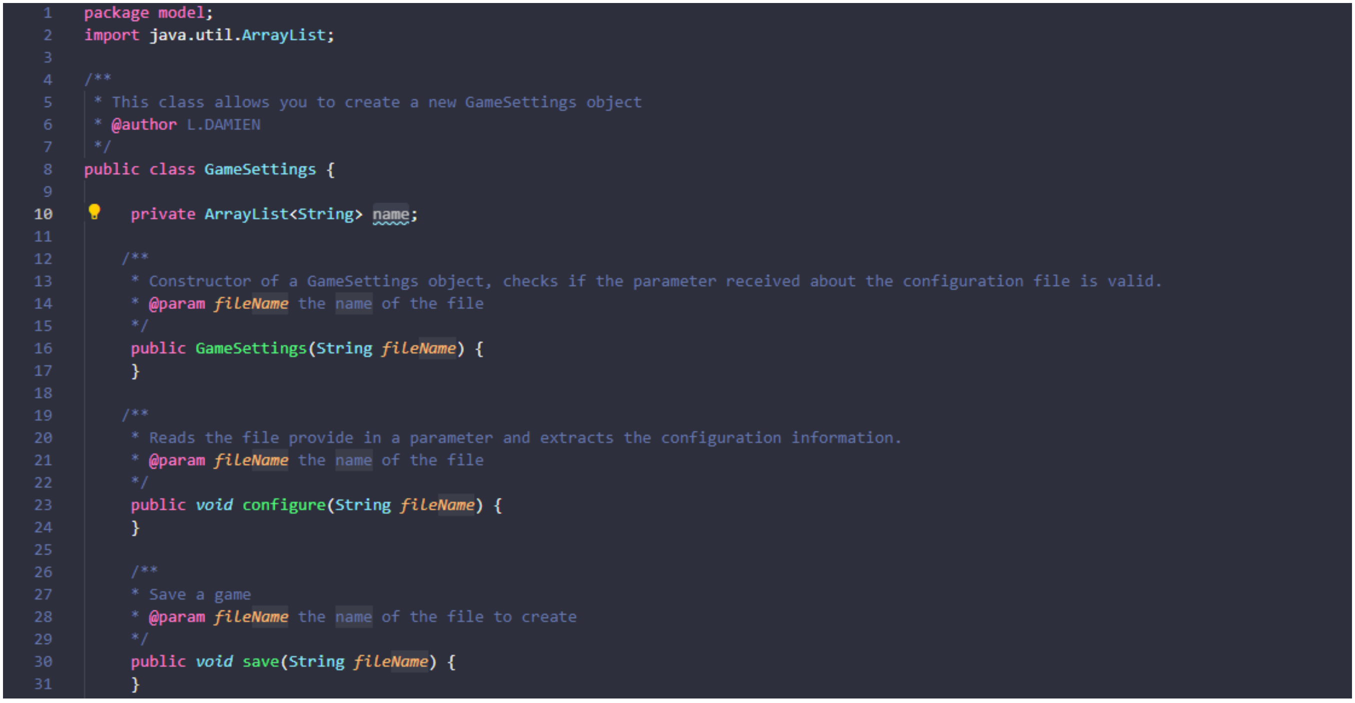
Afin de pouvoir répondre à la demande d'avoir la possibilité de sauvegarder et de charger des parties nous allons utiliser le procédé de sérialisation, en utilisant la classe Serializable du package java.io. Pour cela les classes du package model devront implémenter l’interface Serializable.

Pour la partie de chargement d'une nouvelle partie c'est la méthode load de la classe GameSettings qui sera appelé avec comme paramètre le nom de la sauvegarde.

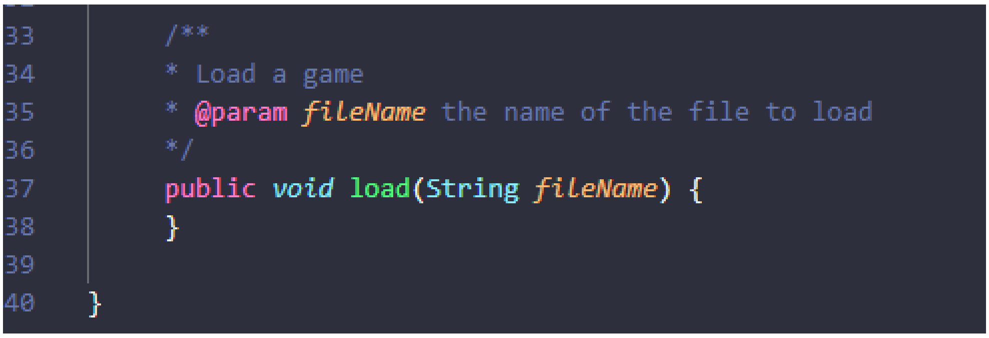
Chaque nom de sauvergarde sera générée de la façon suivante : date – heure

5) Modèle des classes principales

La classe GameSettings :

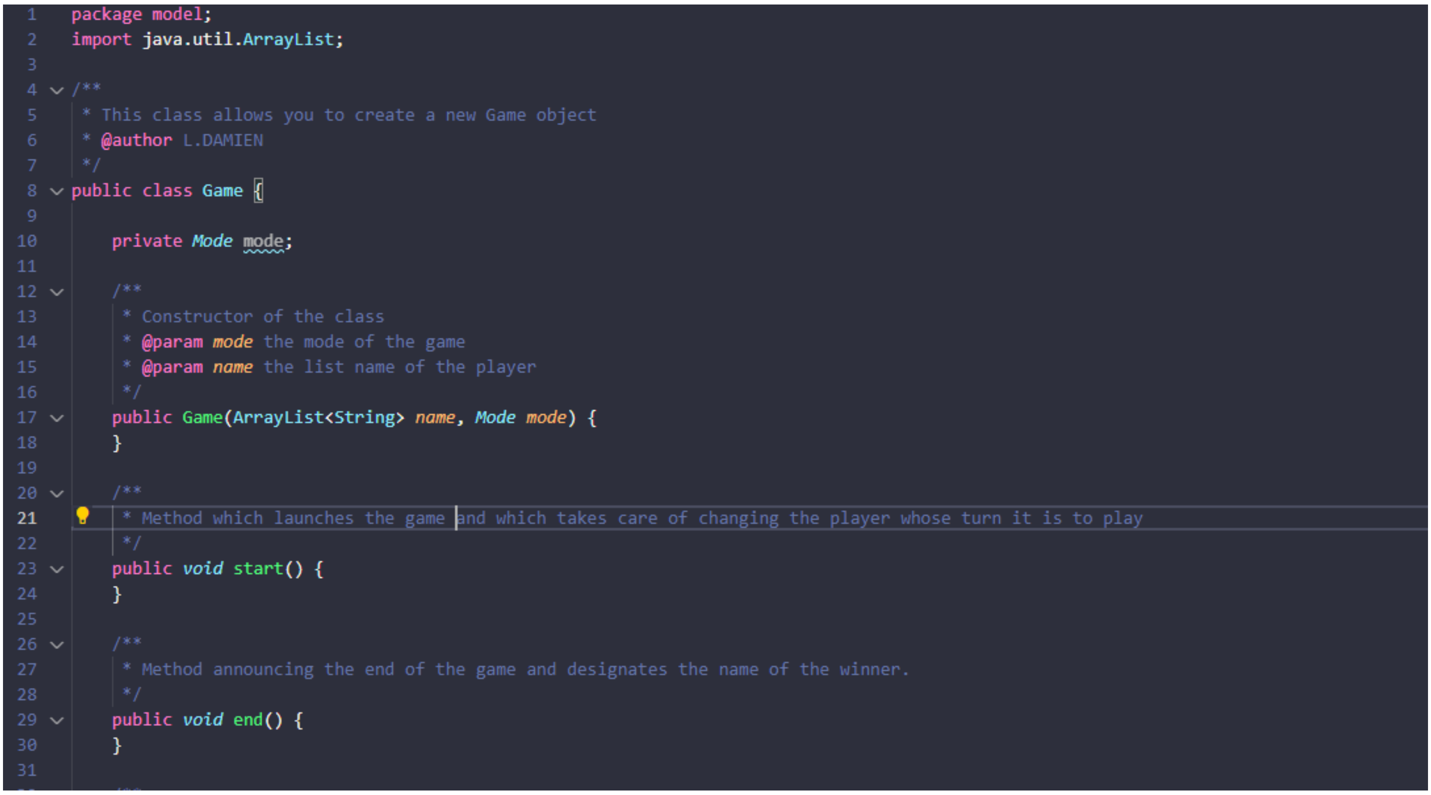


```java
package model;
import java.util.ArrayList;

/**
 * This class allows you to create a new GameSettings object
 * @author L.DAMIEN
 */
public class GameSettings {

    private ArrayList<String> name;

    /**
     * Constructor of a GameSettings object, checks if the parameter received about the configuration file is valid.
     * @param fileName the name of the file
     */
    public GameSettings(String fileName) {
    }

    /**
     * Reads the file provide in a parameter and extracts the configuration information.
     * @param fileName the name of the file
     */
    public void configure(String fileName) {
    }

     /**
     * Save a game
     * @param fileName the name of the file to create
     */
    public void save(String fileName) {
    }
```

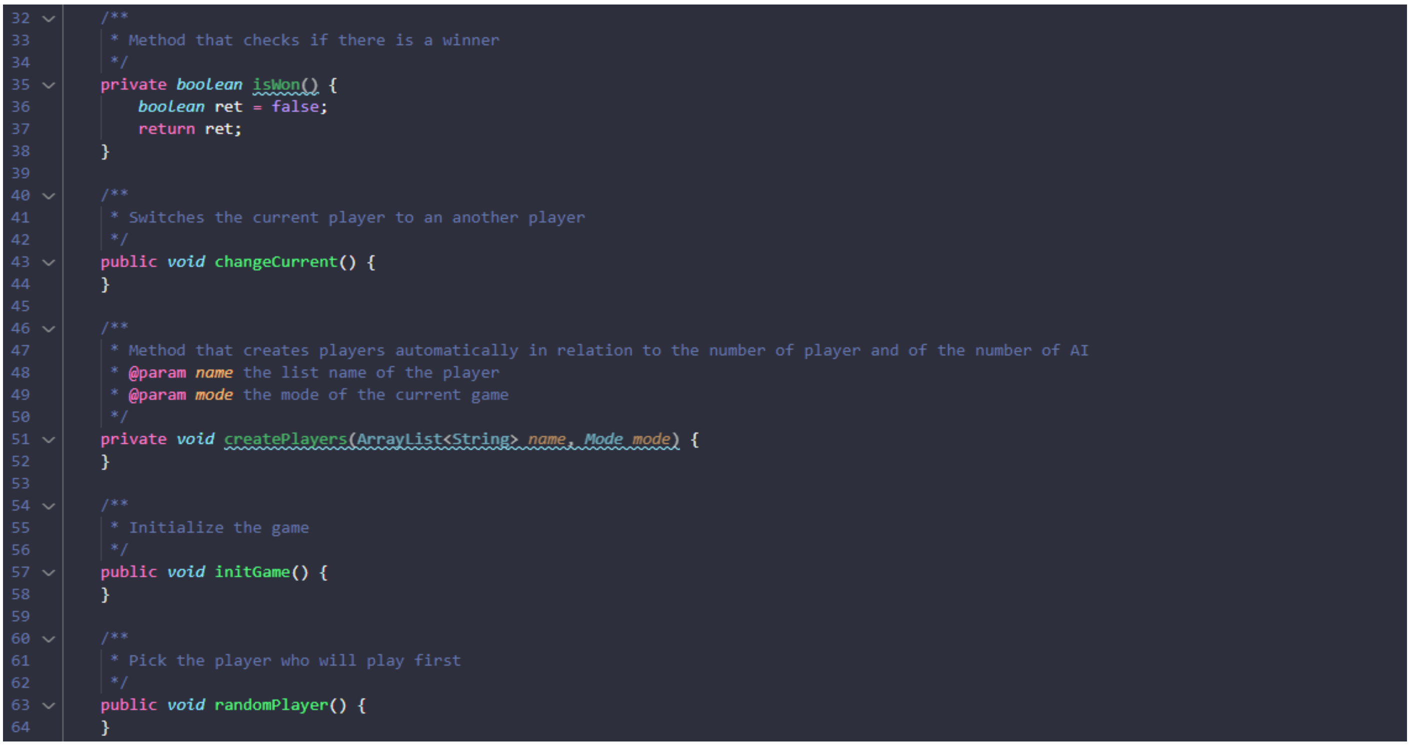


```java
    /**
    * Load a game
    * @param fileName the name of the file to load
    */
    public void load(String fileName) {
    }

}
```

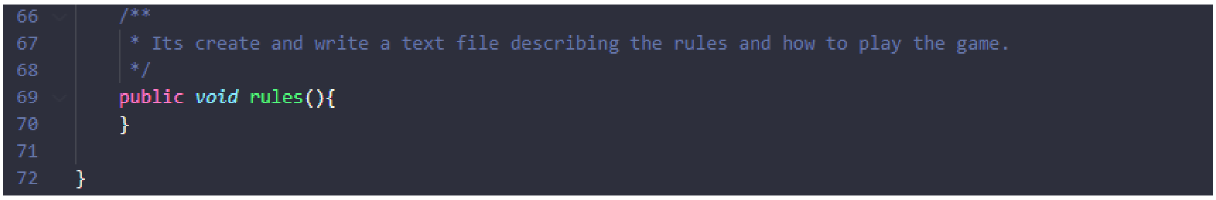
La classe Game :



```java
package model;
import java.util.ArrayList;

/**
 * This class allows you to create a new Game object
 * @author L.DAMIEN
 */
public class Game {

    private Mode mode;

    /**
     * Constructor of the class
     * @param mode the mode of the game
     * @param name the list name of the player
     */
    public Game(ArrayList<String> name, Mode mode) {
    }

    /**
     * Method which launches the game and which takes care of changing the player whose turn it is to play
     */
    public void start() {
    }

    /**
     * Method announcing the end of the game and designates the name of the winner.
     */
    public void end() {
    }

```

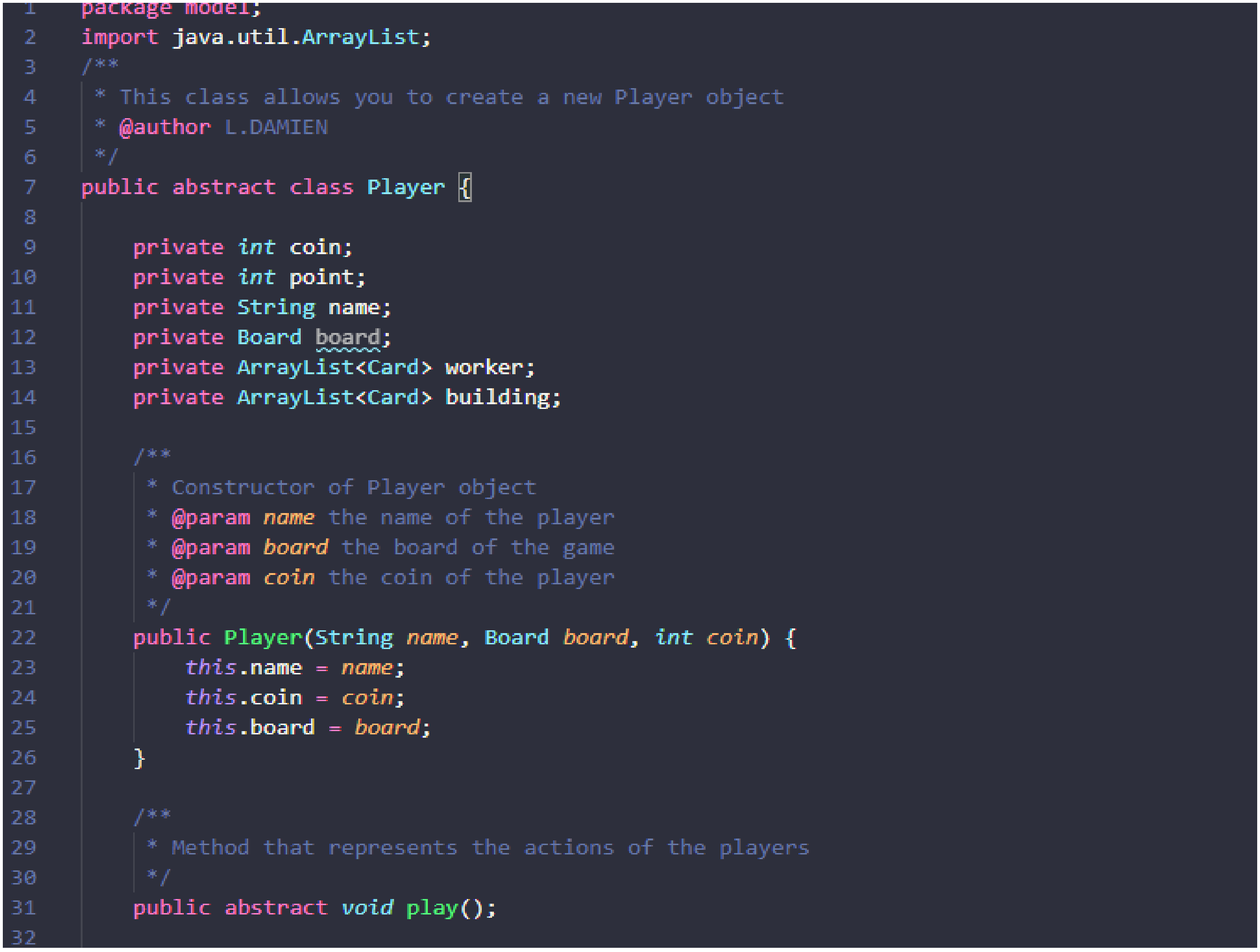


```java
    /**
     * Method that checks if there is a winner
     */
    private boolean isWon() {
        boolean ret = false;
        return ret;
    }

    /**
     * Switches the current player to an another player
     */
    public void changeCurrent() {
    }

    /**
     * Method that creates players automatically in relation to the number of player and of the number of AI
     * @param name the list name of the player
     * @param mode the mode of the current game
     */
    private void createPlayers(ArrayList<String> name, Mode mode) {
    }

    /**
     * Initialize the game
     */
    public void initGame() {
    }

    /**
     * Pick the player who will play first
     */
    public void randomPlayer() {
    }
```

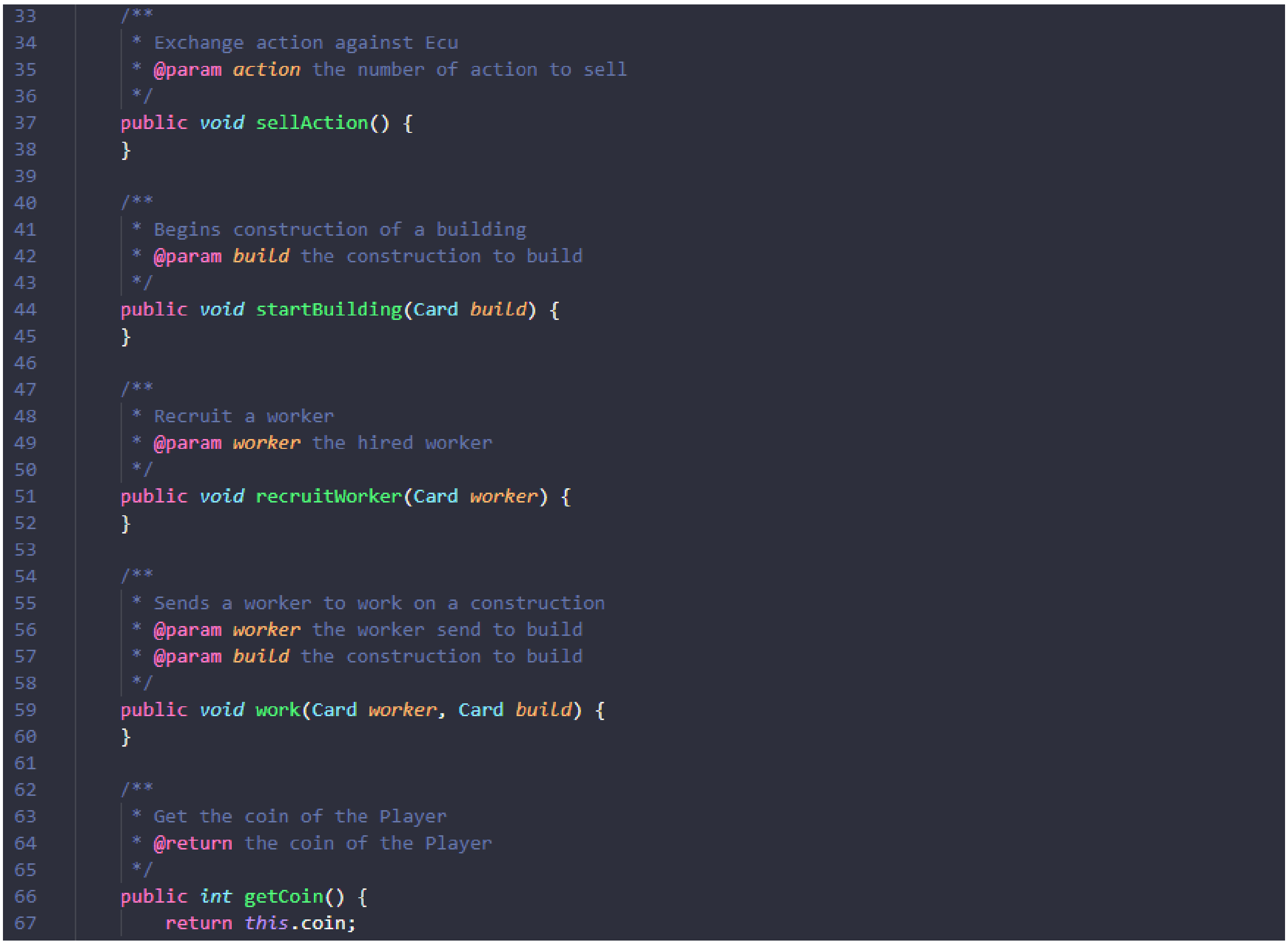


```java
    /**
     * Its create and write a text file describing the rules and how to play the game.
     */
    public void rules(){
    }

}
```

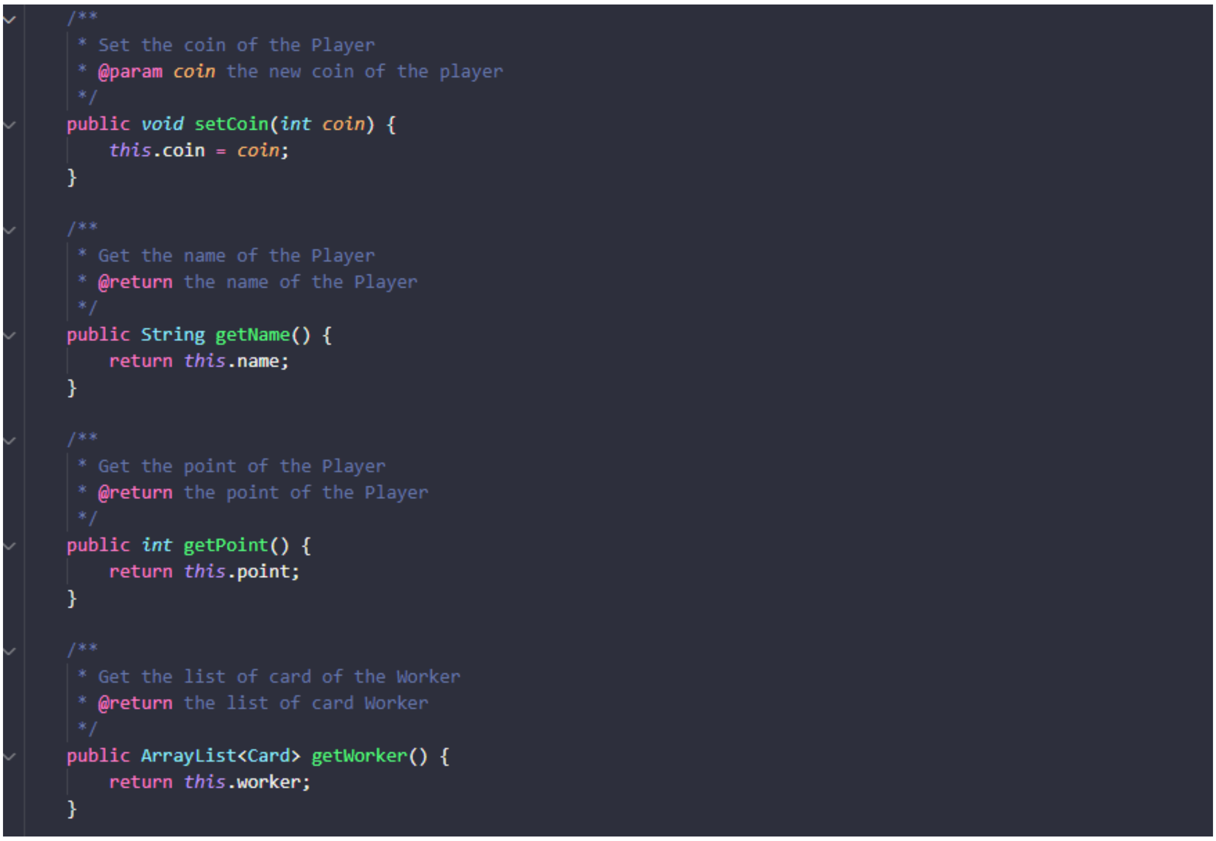
La classe Player :



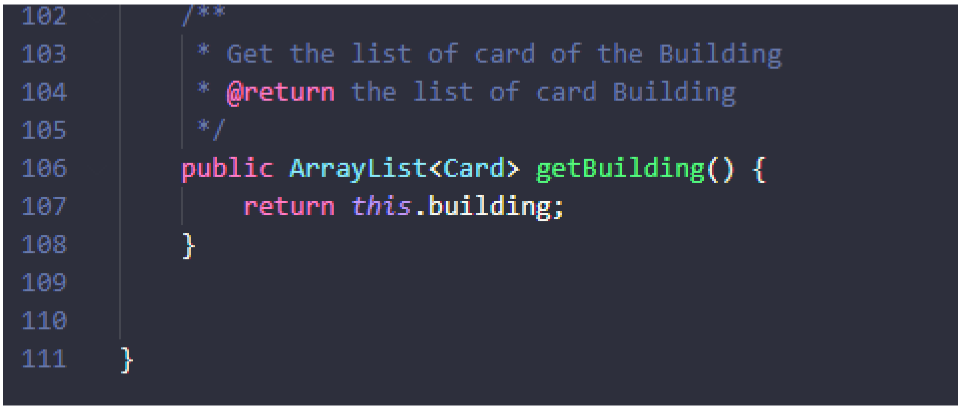
```java
package model;
import java.util.ArrayList;
/**
 * This class allows you to create a new Player object
 * @author L.DAMIEN
 */
public abstract class Player {

    private int coin;
    private int point;
    private String name;
    private Board board;
    private ArrayList<Card> worker;
    private ArrayList<Card> building;

    /**
     * Constructor of Player object
     * @param name the name of the player
     * @param board the board of the game
     * @param coin the coin of the player
     */
    public Player(String name, Board board, int coin) {
        this.name = name;
        this.coin = coin;
        this.board = board;
    }

    /**
     * Method that represents the actions of the players
     */
    public abstract void play();

```



```java
    /**
     * Exchange action against Ecu
     * @param action the number of action to sell
     */
    public void sellAction() {
    }

    /**
     * Begins construction of a building
     * @param build the construction to build
     */
    public void startBuilding(Card build) {
    }

    /**
     * Recruit a worker
     * @param worker the hired worker
     */
    public void recruitWorker(Card worker) {
    }

    /**
     * Sends a worker to work on a construction
     * @param worker the worker send to build
     * @param build the construction to build
     */
    public void work(Card worker, Card build) {
    }

    /**
     * Get the coin of the Player
     * @return the coin of the Player
     */
    public int getCoin() {
        return this.coin;
```

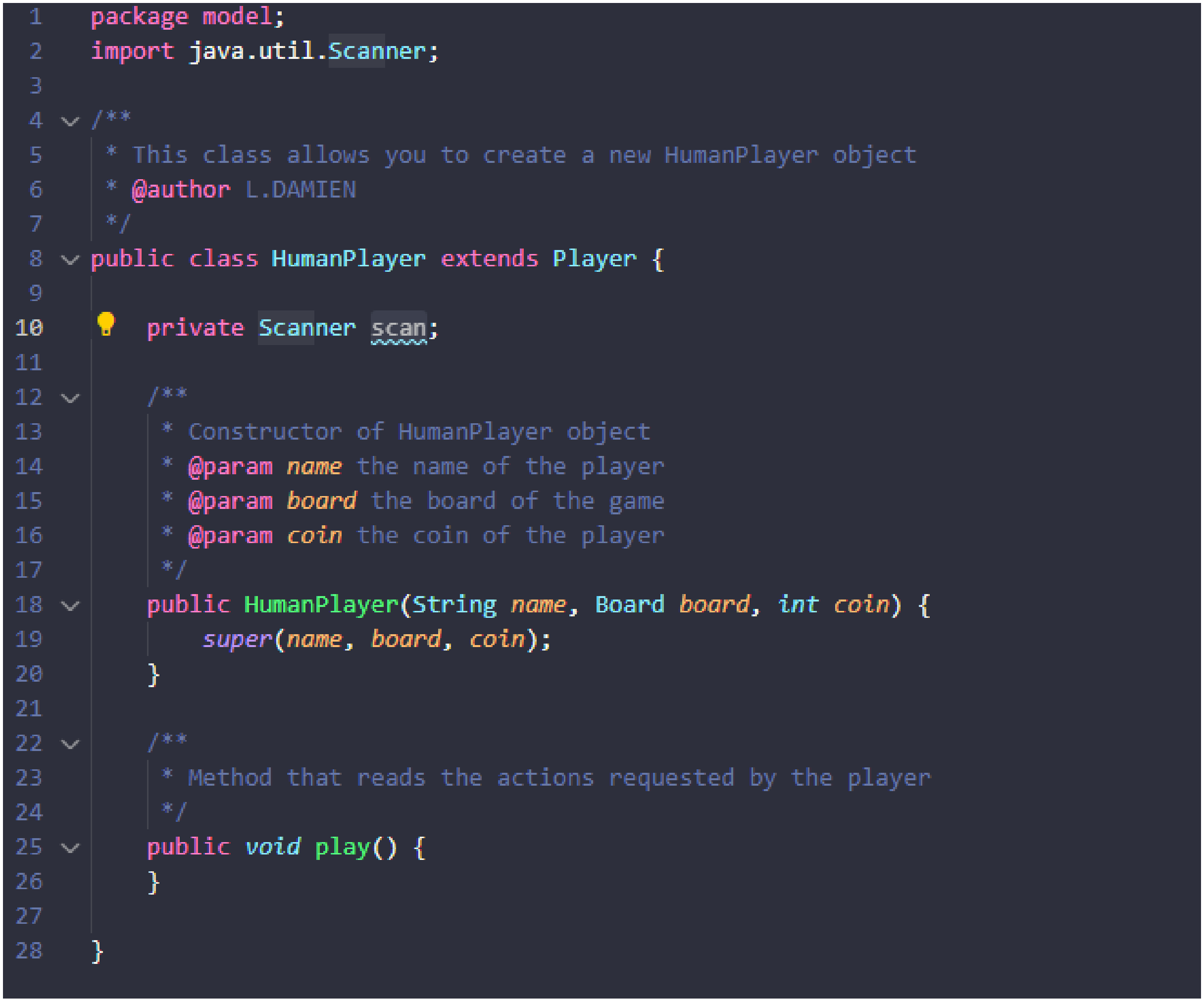


```java
    /**
     * Set the coin of the Player
     * @param coin the new coin of the player
     */
    public void setCoin(int coin) {
        this.coin = coin;
    }

    /**
     * Get the name of the Player
     * @return the name of the Player
     */
    public String getName() {
        return this.name;
    }

    /**
     * Get the point of the Player
     * @return the point of the Player
     */
    public int getPoint() {
        return this.point;
    }

    /**
     * Get the list of card of the Worker
     * @return the list of card Worker
     */
    public ArrayList<Card> getWorker() {
        return this.worker;
    }
```



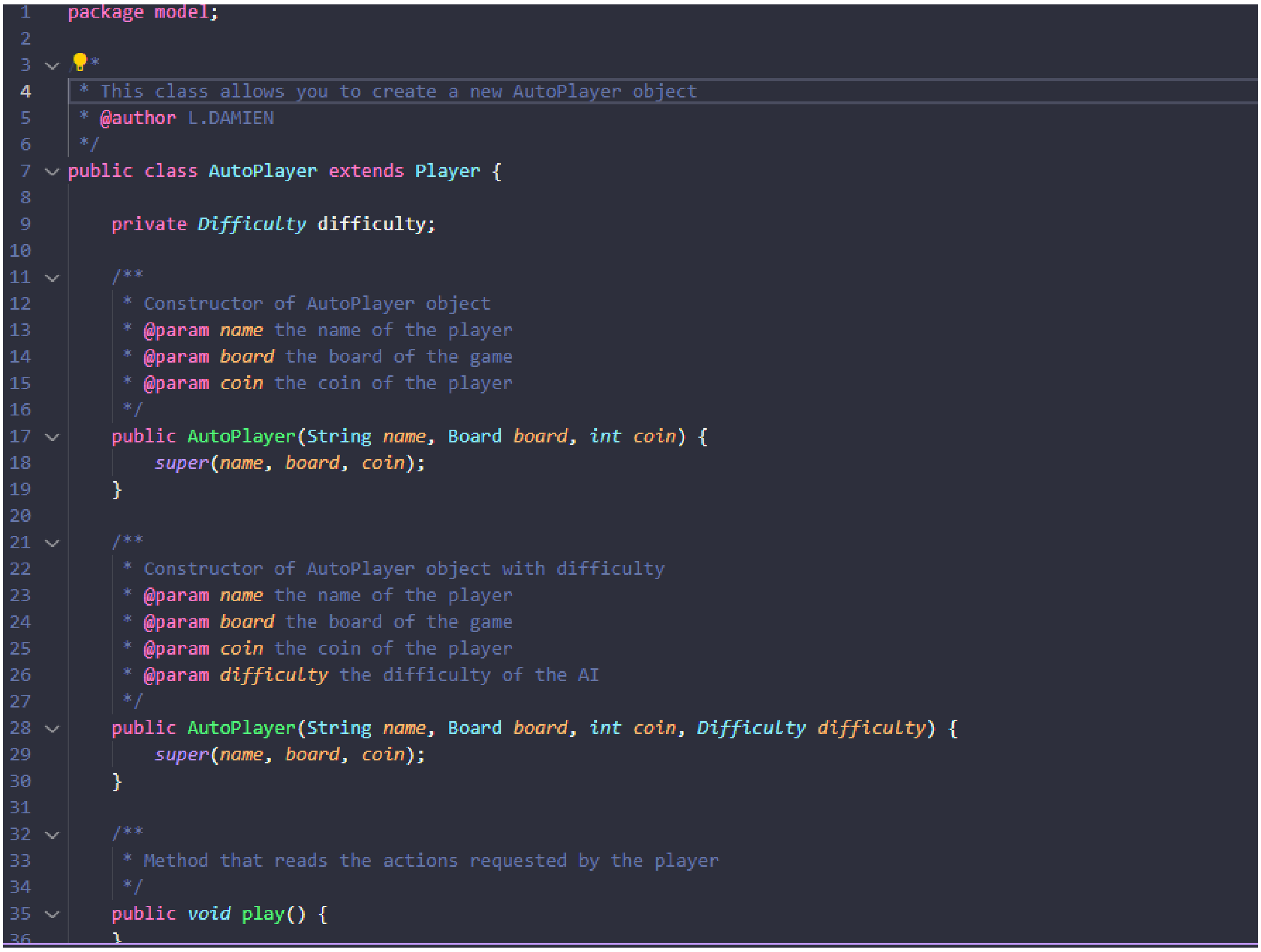
```
    /**
     * Get the list of card of the Building
     * @return the list of card Building
     */
    public ArrayList<Card> getBuilding() {
        return this.building;
    }


}
```

La classe HumanPlayer :

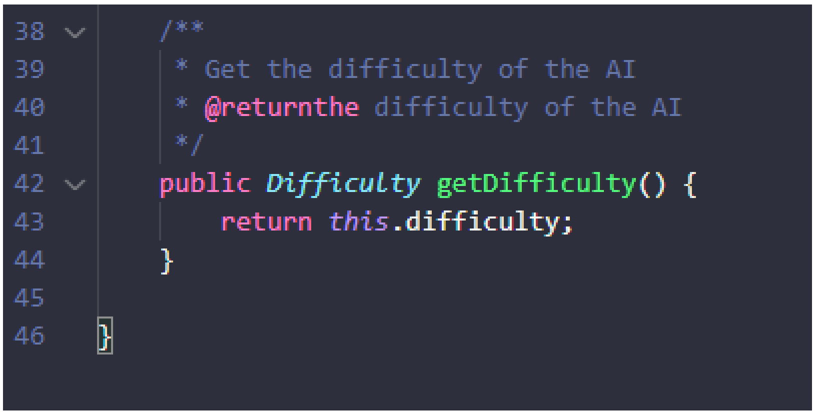


```
package model;
import java.util.Scanner;

/**
 * This class allows you to create a new HumanPlayer object
 * @author L.DAMIEN
 */
public class HumanPlayer extends Player {

    private Scanner scan;

    /**
     * Constructor of HumanPlayer object
     * @param name the name of the player
     * @param board the board of the game
     * @param coin the coin of the player
     */
    public HumanPlayer(String name, Board board, int coin) {
        super(name, board, coin);
    }

    /**
     * Method that reads the actions requested by the player
     */
    public void play() {
    }

}
```

La classe AutoPlayer :

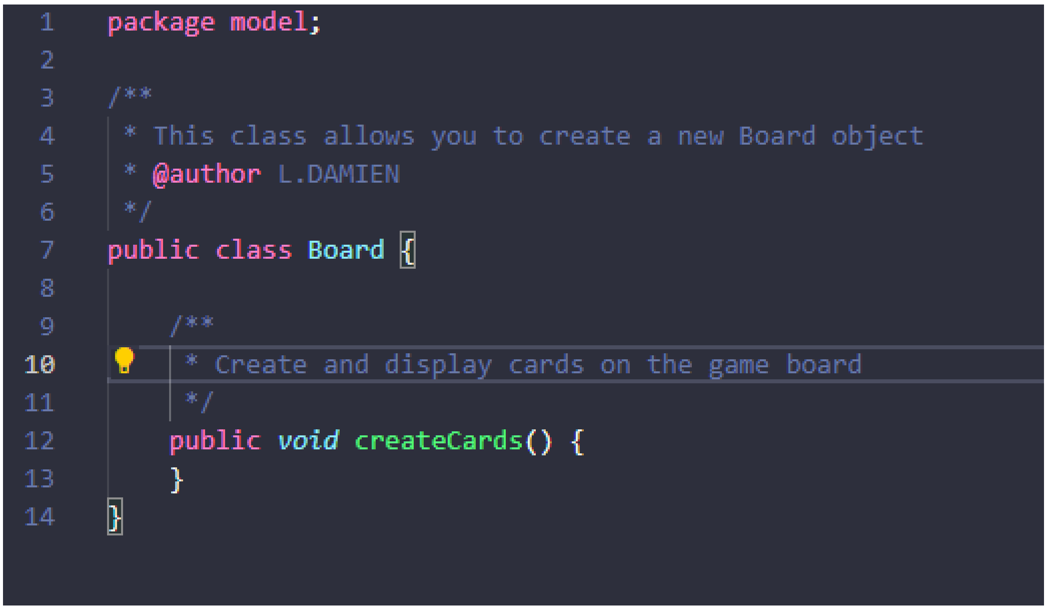


```java
package model;

 *
 * This class allows you to create a new AutoPlayer object
 * @author L.DAMIEN
 */
public class AutoPlayer extends Player {

    private Difficulty difficulty;

    /**
     * Constructor of AutoPlayer object
     * @param name the name of the player
     * @param board the board of the game
     * @param coin the coin of the player
     */
    public AutoPlayer(String name, Board board, int coin) {
        super(name, board, coin);
    }

    /**
     * Constructor of AutoPlayer object with difficulty
     * @param name the name of the player
     * @param board the board of the game
     * @param coin the coin of the player
     * @param difficulty the difficulty of the AI
     */
    public AutoPlayer(String name, Board board, int coin, Difficulty difficulty) {
        super(name, board, coin);
    }

    /**
     * Method that reads the actions requested by the player
     */
    public void play() {
    }
```



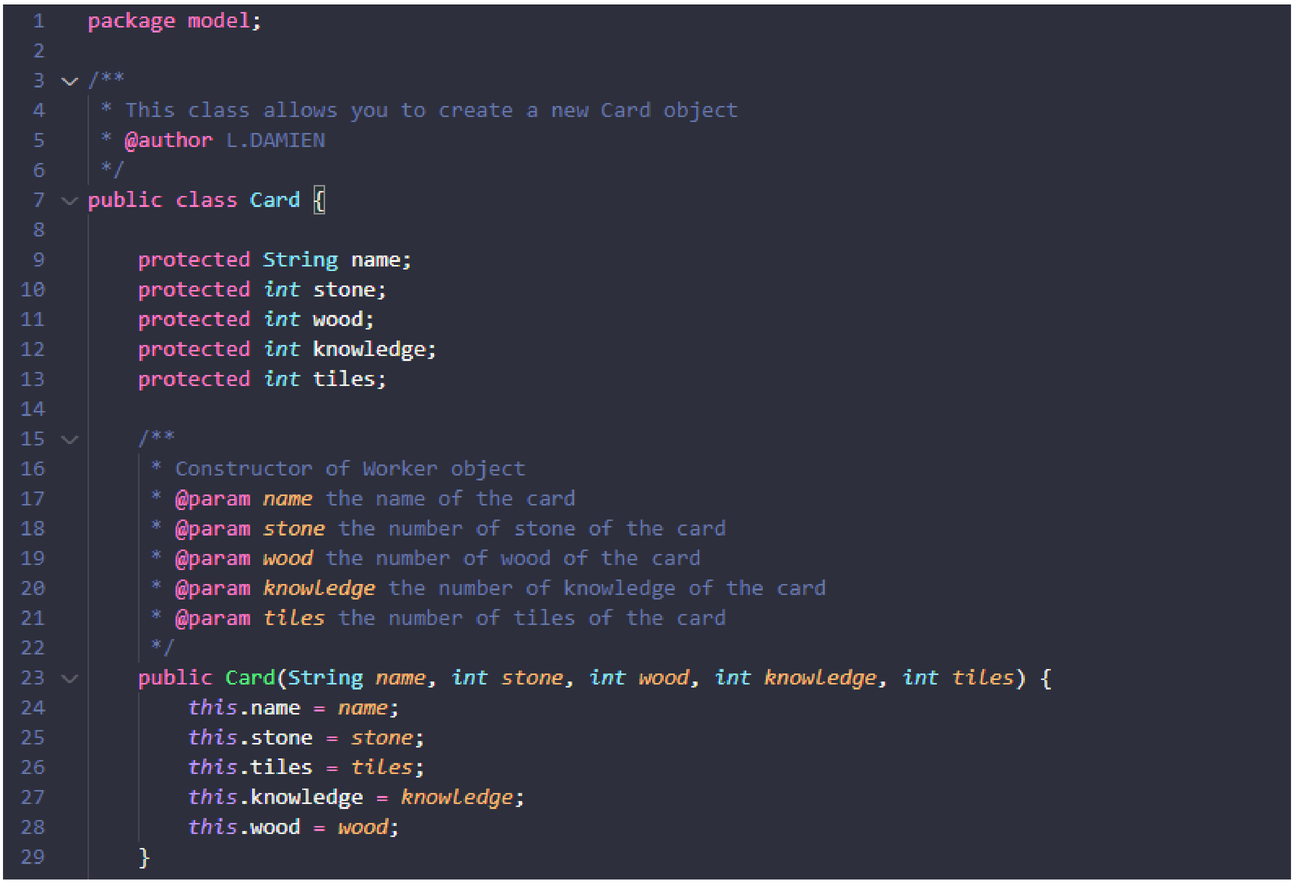
```java
    /**
     * Get the difficulty of the AI
     * @returnthe difficulty of the AI
     */
    public Difficulty getDifficulty() {
        return this.difficulty;
    }

}
```

La classe Board :

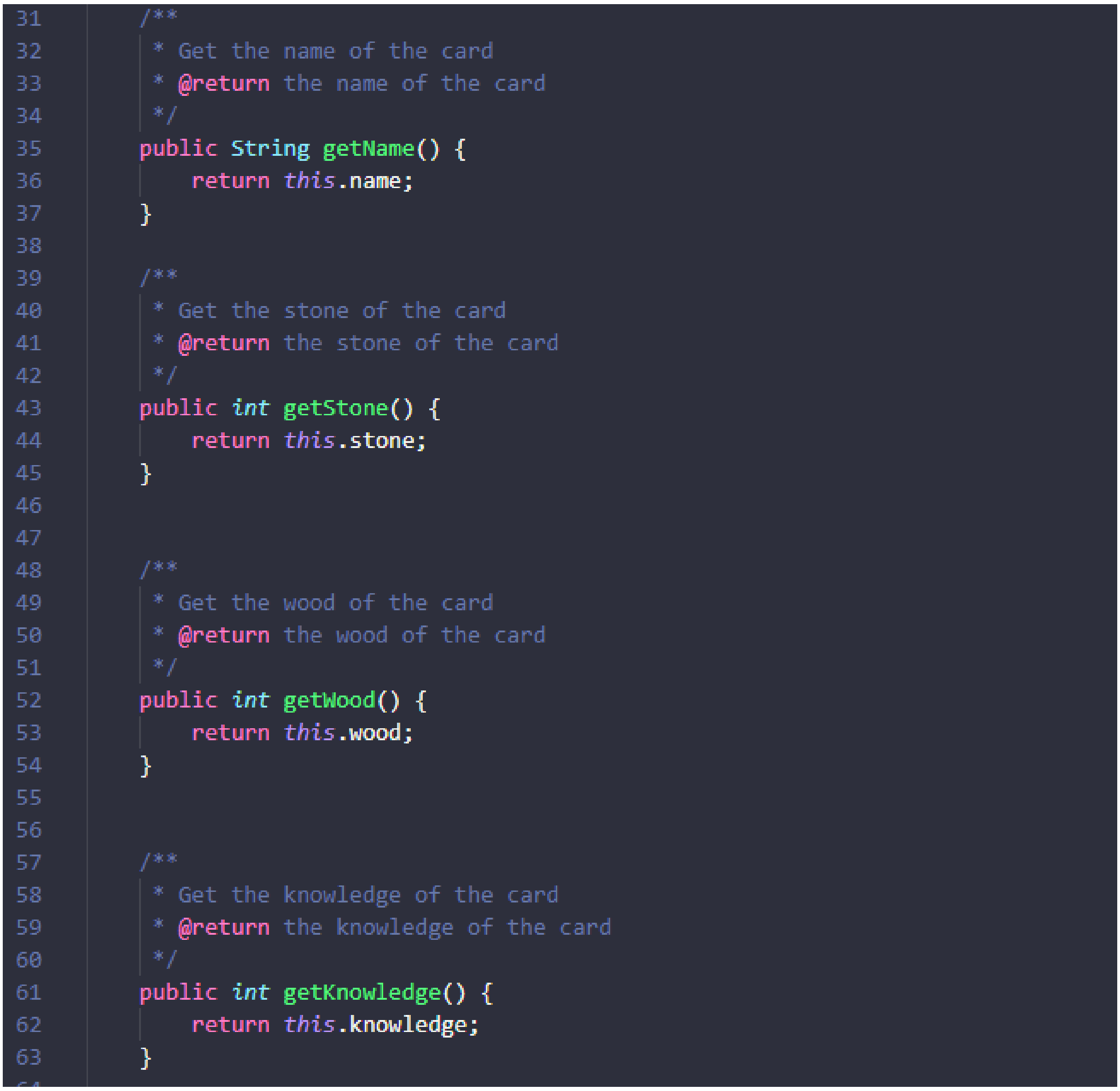


```java
package model;

/**
 * This class allows you to create a new Board object
 * @author L.DAMIEN
 */
public class Board {

    /**
     * Create and display cards on the game board
     */
    public void createCards() {
    }
}
```

La classe Card :

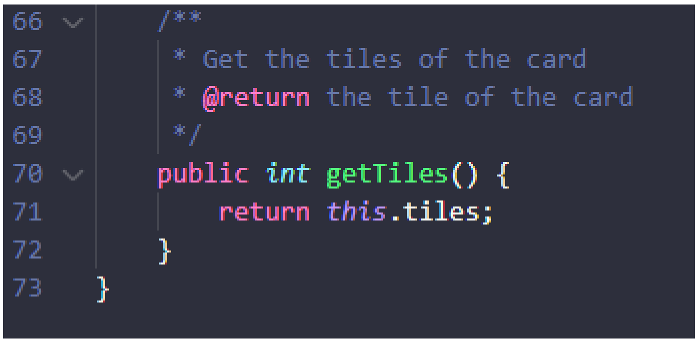


```java
package model;

/**
 * This class allows you to create a new Card object
 * @author L.DAMIEN
 */
public class Card {

    protected String name;
    protected int stone;
    protected int wood;
    protected int knowledge;
    protected int tiles;

    /**
     * Constructor of Worker object
     * @param name the name of the card
     * @param stone the number of stone of the card
     * @param wood the number of wood of the card
     * @param knowledge the number of knowledge of the card
     * @param tiles the number of tiles of the card
     */
    public Card(String name, int stone, int wood, int knowledge, int tiles) {
        this.name = name;
        this.stone = stone;
        this.tiles = tiles;
        this.knowledge = knowledge;
        this.wood = wood;
    }
```

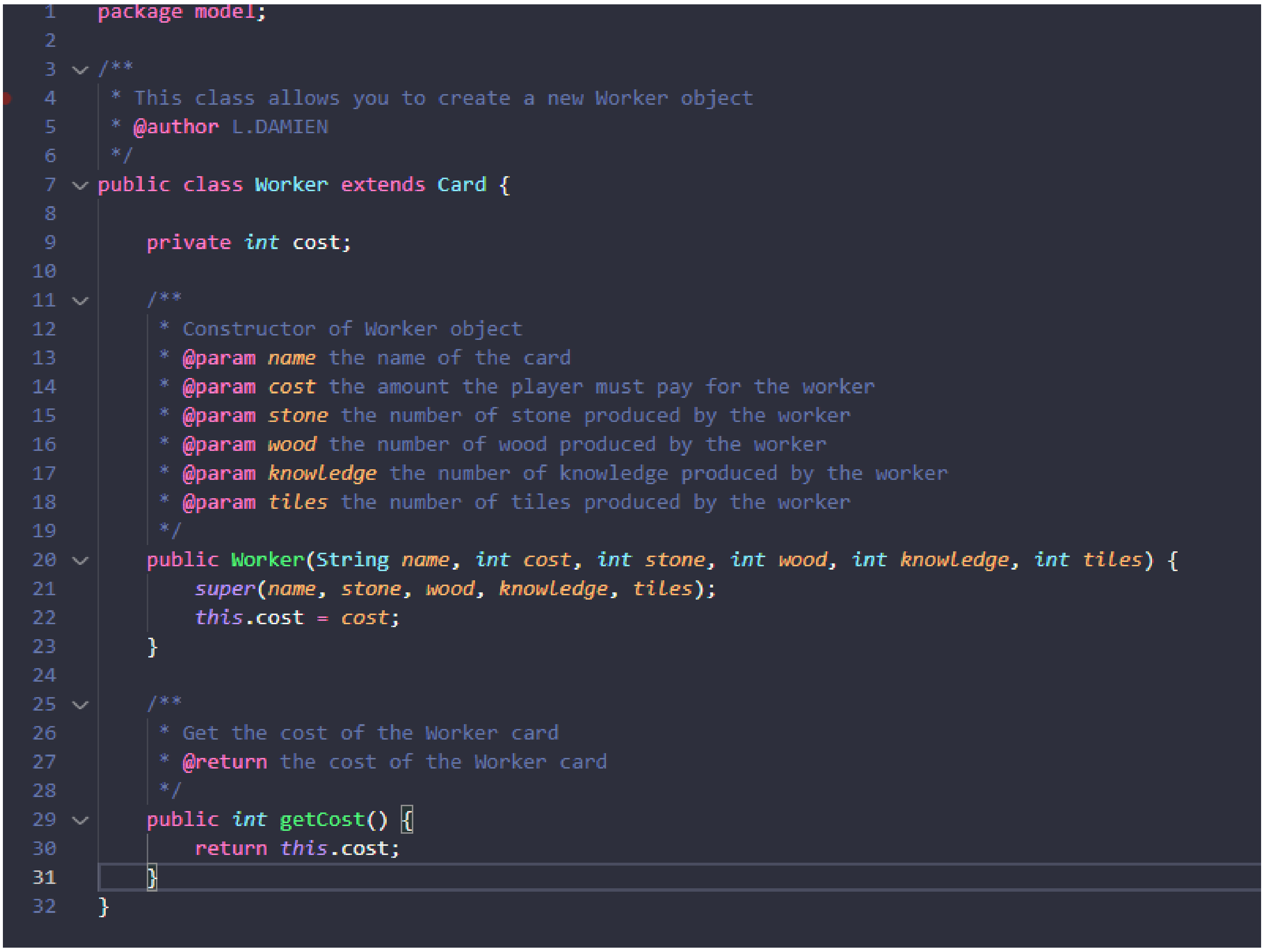


```java
    /**
     * Get the name of the card
     * @return the name of the card
     */
    public String getName() {
        return this.name;
    }

    /**
     * Get the stone of the card
     * @return the stone of the card
     */
    public int getStone() {
        return this.stone;
    }


    /**
     * Get the wood of the card
     * @return the wood of the card
     */
    public int getWood() {
        return this.wood;
    }


    /**
     * Get the knowledge of the card
     * @return the knowledge of the card
     */
    public int getKnowledge() {
        return this.knowledge;
    }
```

La classe Worker :

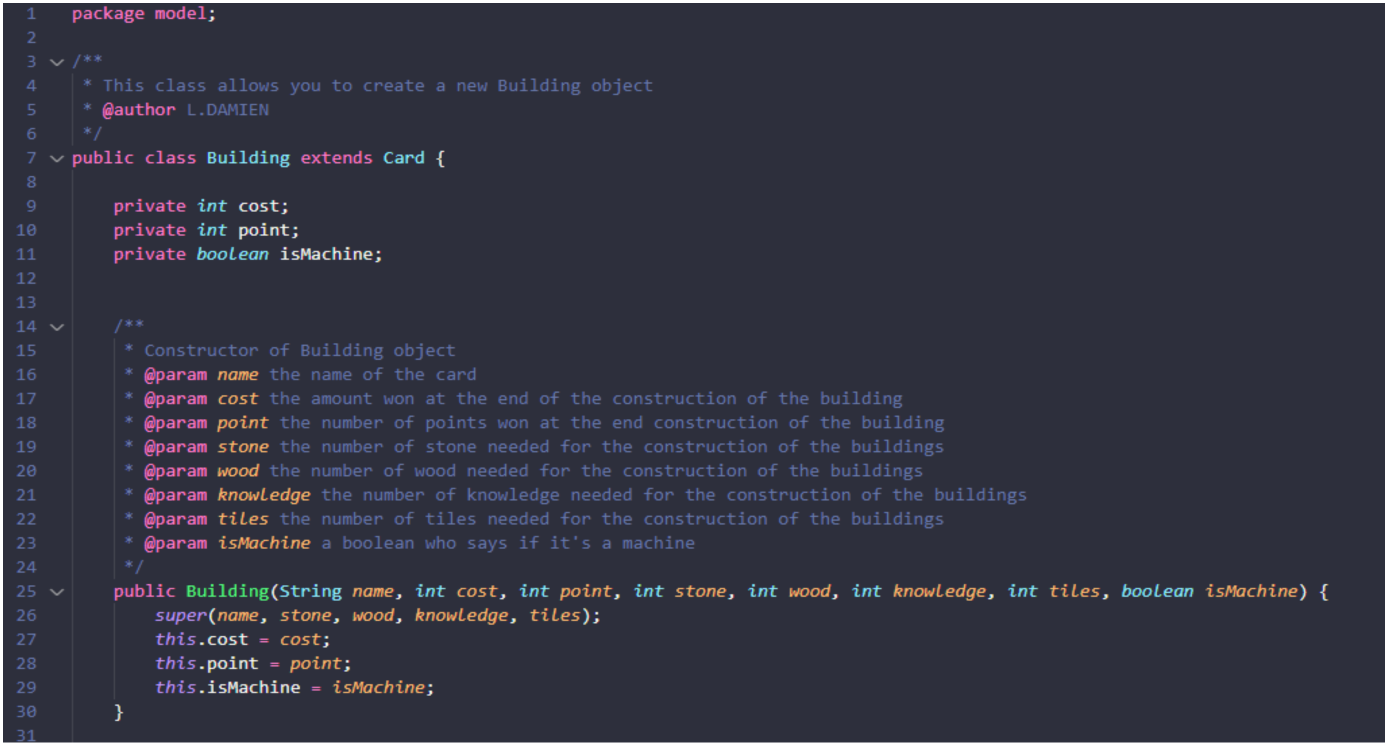


```java
    /**
     * Get the tiles of the card
     * @return the tile of the card
     */
    public int getTiles() {
        return this.tiles;
    }
}
```

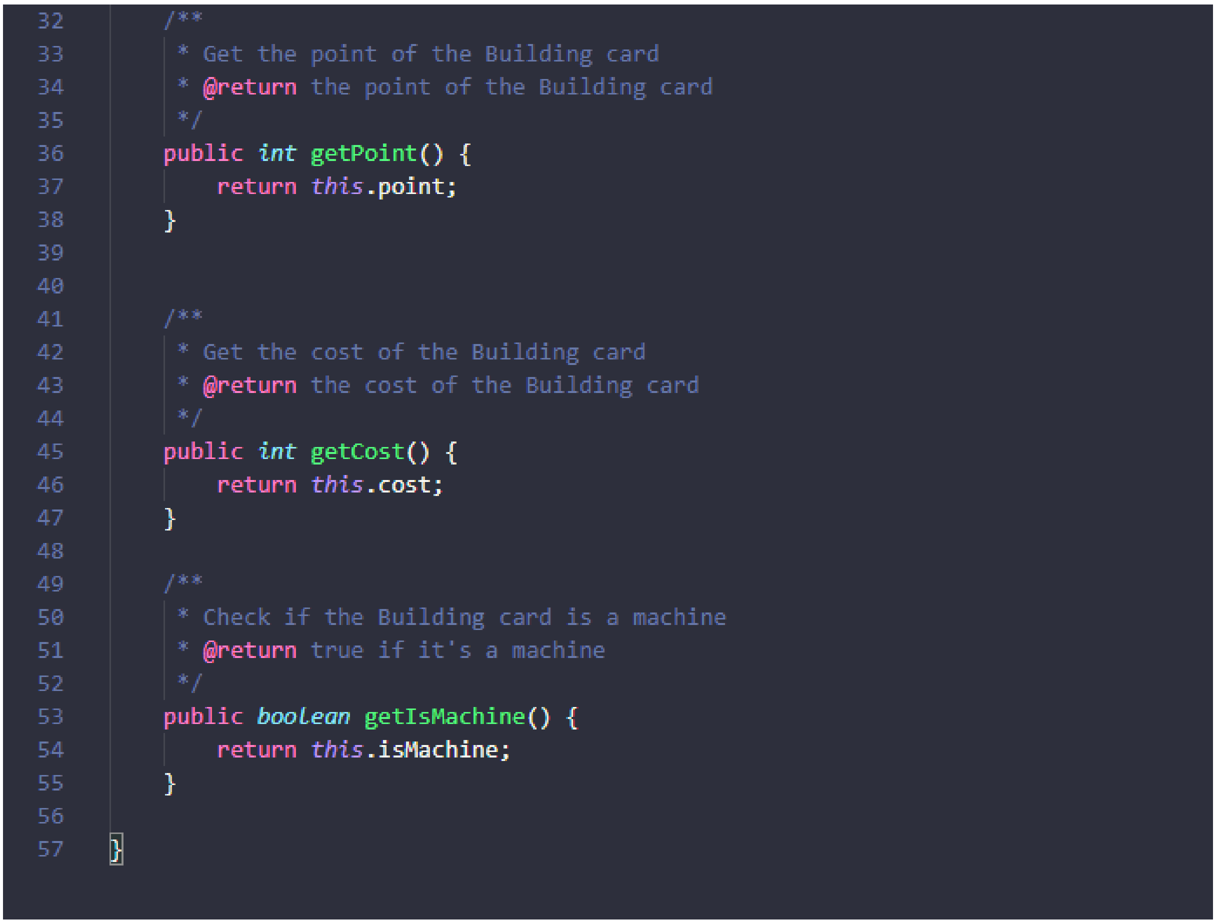


```java
package model;

/**
 * This class allows you to create a new Worker object
 * @author L.DAMIEN
 */
public class Worker extends Card {

    private int cost;

    /**
     * Constructor of Worker object
     * @param name the name of the card
     * @param cost the amount the player must pay for the worker
     * @param stone the number of stone produced by the worker
     * @param wood the number of wood produced by the worker
     * @param knowledge the number of knowledge produced by the worker
     * @param tiles the number of tiles produced by the worker
     */
    public Worker(String name, int cost, int stone, int wood, int knowledge, int tiles) {
        super(name, stone, wood, knowledge, tiles);
        this.cost = cost;
    }

    /**
     * Get the cost of the Worker card
     * @return the cost of the Worker card
     */
    public int getCost() {
        return this.cost;
    }
}
```

La classe Building :

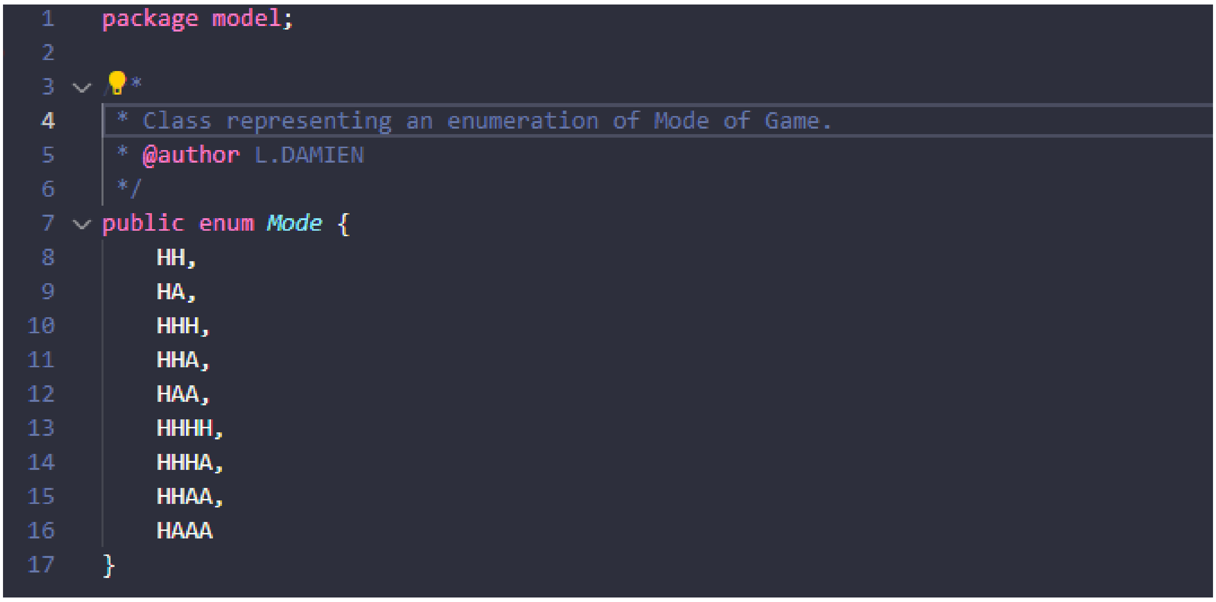


```java
package model;

/**
 * This class allows you to create a new Building object
 * @author L.DAMIEN
 */
public class Building extends Card {

    private int cost;
    private int point;
    private boolean isMachine;


    /**
     * Constructor of Building object
     * @param name the name of the card
     * @param cost the amount won at the end of the construction of the building
     * @param point the number of points won at the end construction of the building
     * @param stone the number of stone needed for the construction of the buildings
     * @param wood the number of wood needed for the construction of the buildings
     * @param knowledge the number of knowledge needed for the construction of the buildings
     * @param tiles the number of tiles needed for the construction of the buildings
     * @param isMachine a boolean who says if it's a machine
     */
    public Building(String name, int cost, int point, int stone, int wood, int knowledge, int tiles, boolean isMachine) {
        super(name, stone, wood, knowledge, tiles);
        this.cost = cost;
        this.point = point;
        this.isMachine = isMachine;
    }

```



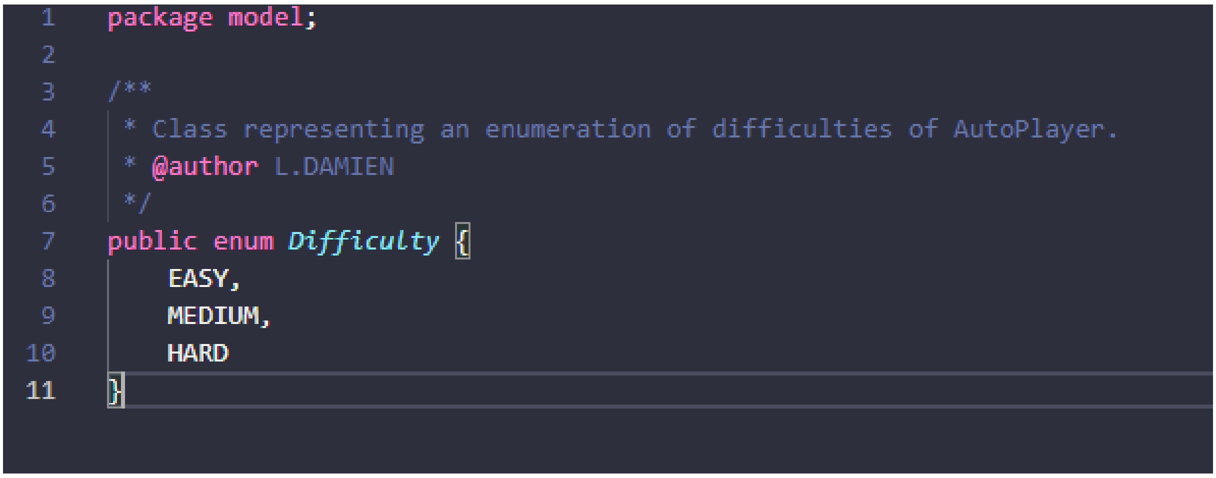
```java
    /**
     * Get the point of the Building card
     * @return the point of the Building card
     */
    public int getPoint() {
        return this.point;
    }


    /**
     * Get the cost of the Building card
     * @return the cost of the Building card
     */
    public int getCost() {
        return this.cost;
    }

    /**
     * Check if the Building card is a machine
     * @return true if it's a machine
     */
    public boolean getIsMachine() {
        return this.isMachine;
    }

}
```

Énumération Mode :



```java
package model;

/**
 * Class representing an enumeration of Mode of Game.
 * @author L.DAMIEN
 */
public enum Mode {
    HH,
    HA,
    HHH,
    HHA,
    HAA,
    HHHH,
    HHHA,
    HHAA,
    HAAA
}
```

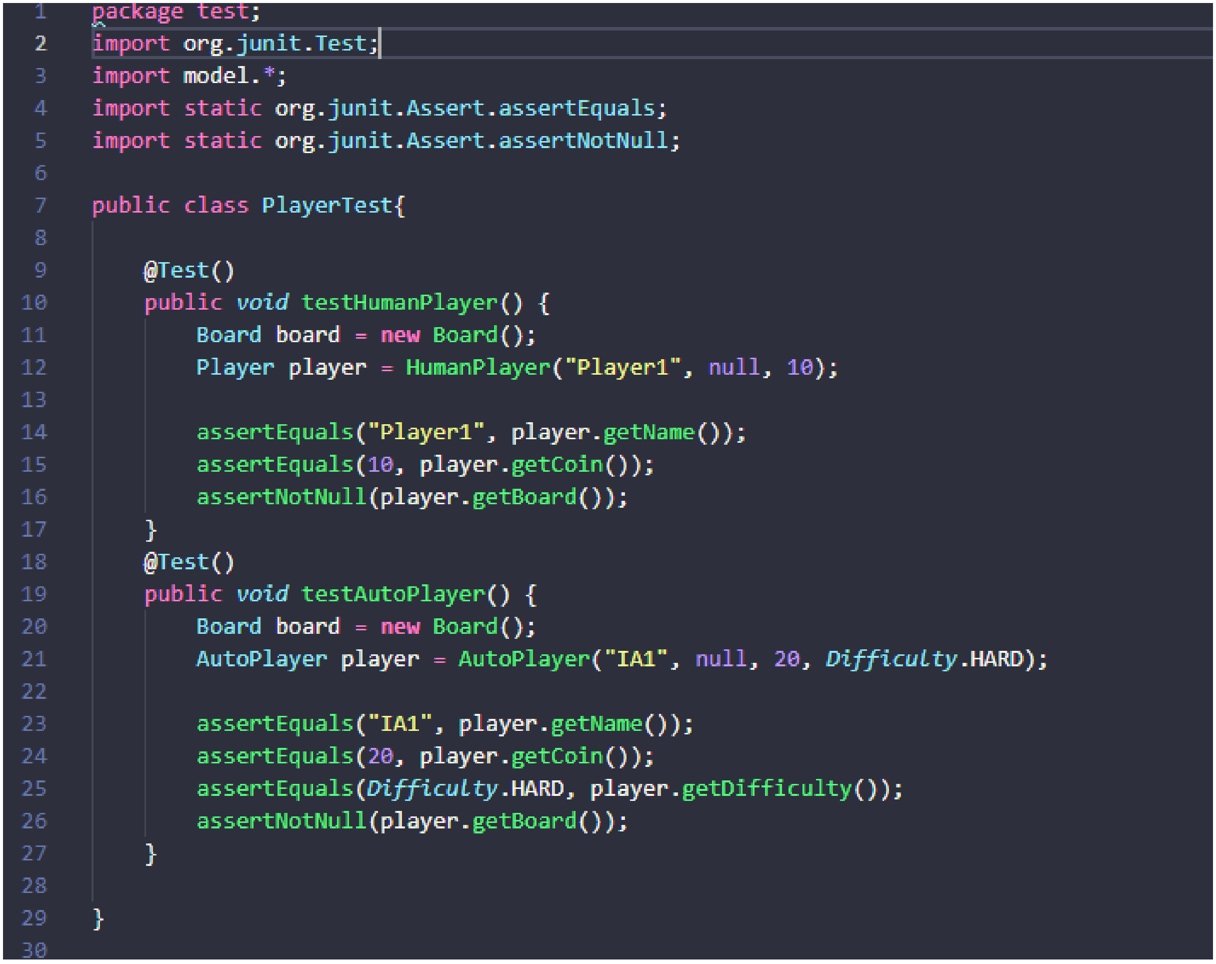
Énumération Difficulty :



```java
package model;

/**
 * Class representing an enumeration of difficulties of AutoPlayer.
 * @author L.DAMIEN
 */
public enum Difficulty {
    EASY,
    MEDIUM,
    HARD
}
```

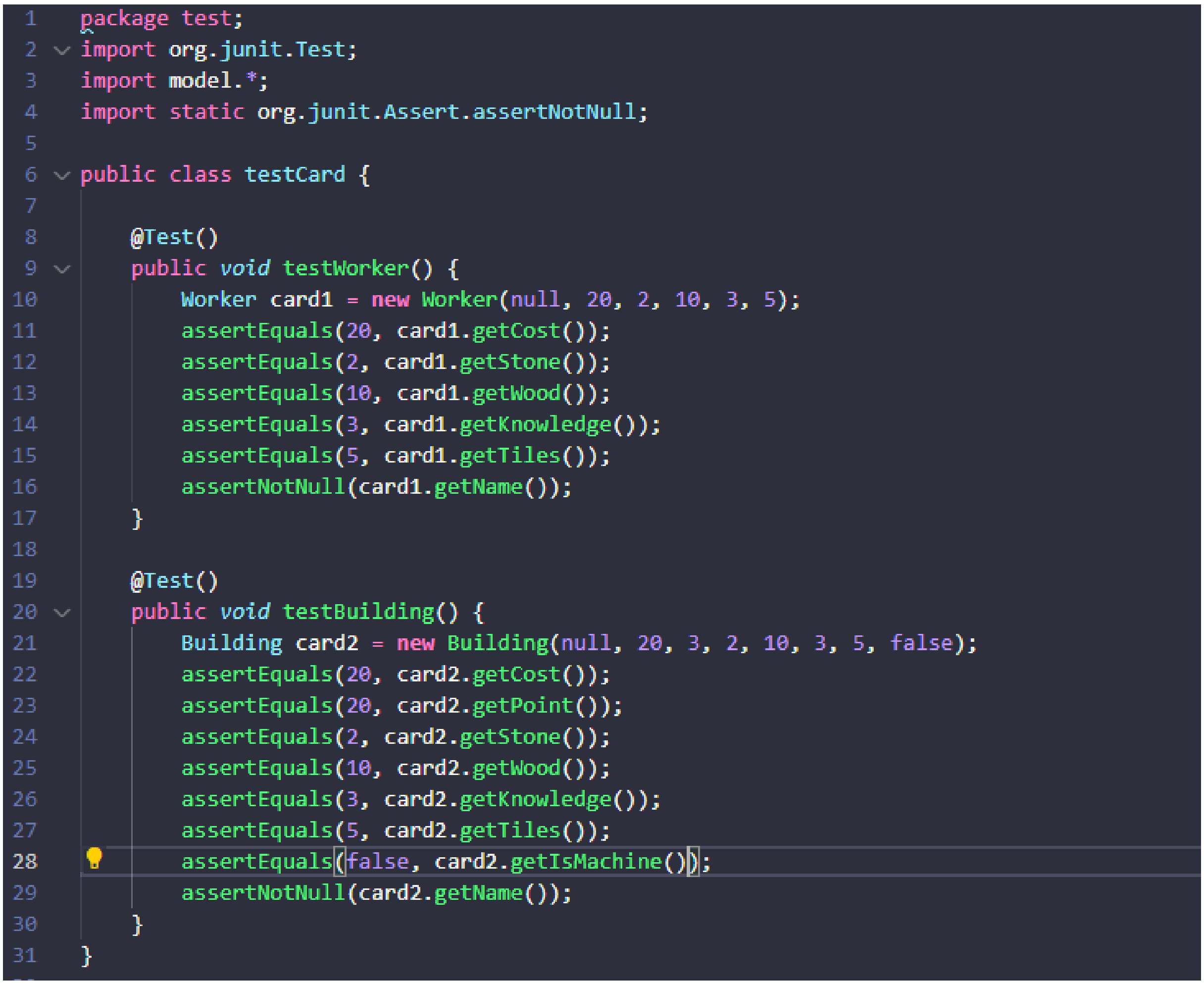
6) Tests unitaires

Test Player / AutoPlayer / HumanPlayer :



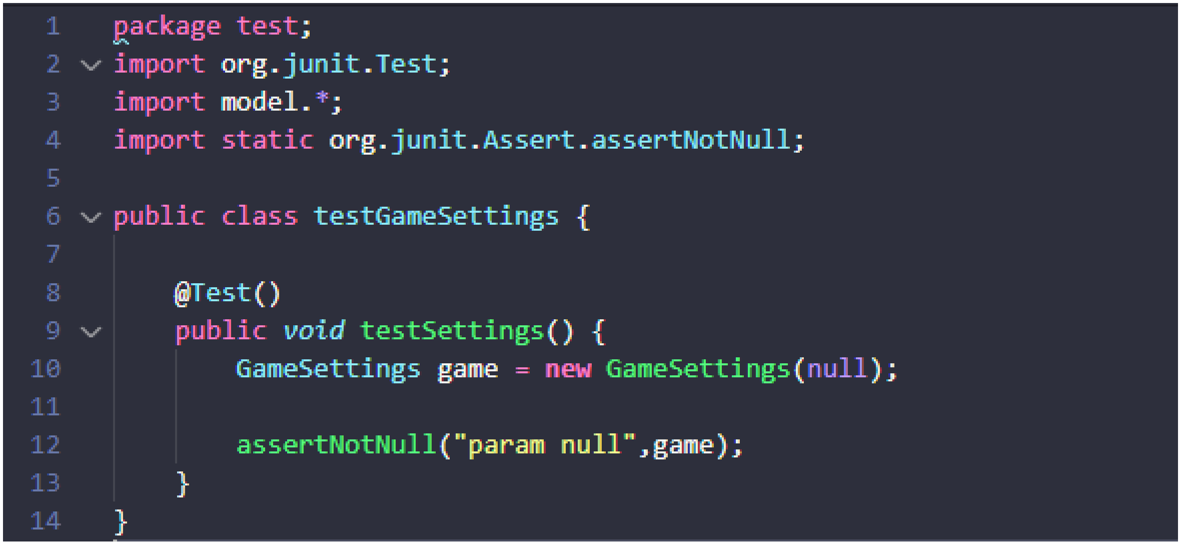
```java
package test;
import org.junit.Test;
import model.*;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;

public class PlayerTest{

    @Test()
    public void testHumanPlayer() {
        Board board = new Board();
        Player player = HumanPlayer("Player1", null, 10);

        assertEquals("Player1", player.getName());
        assertEquals(10, player.getCoin());
        assertNotNull(player.getBoard());
    }
    @Test()
    public void testAutoPlayer() {
        Board board = new Board();
        AutoPlayer player = AutoPlayer("IA1", null, 20, Difficulty.HARD);

        assertEquals("IA1", player.getName());
        assertEquals(20, player.getCoin());
        assertEquals(Difficulty.HARD, player.getDifficulty());
        assertNotNull(player.getBoard());
    }

}

```

Test Card / Worker / Building :



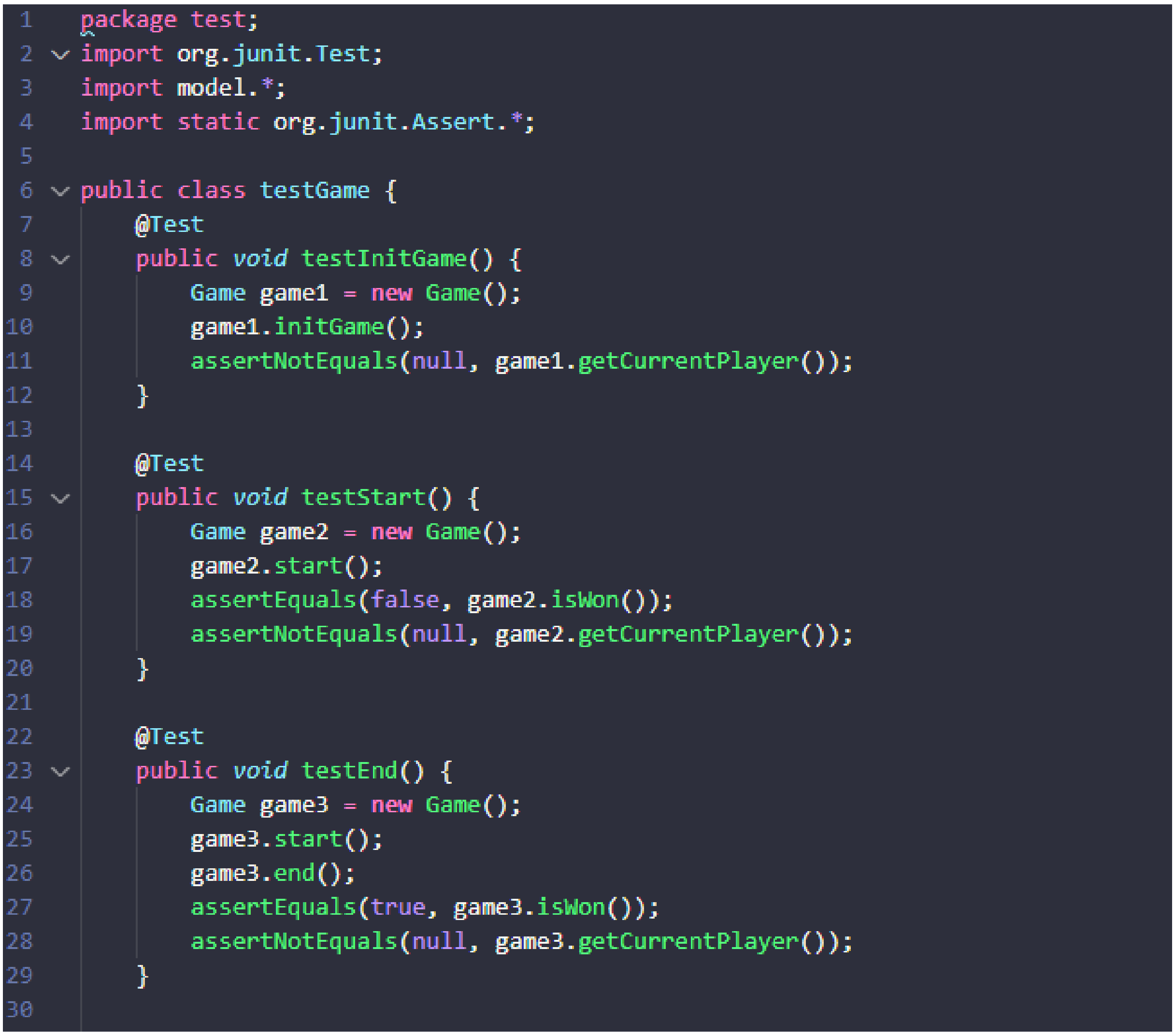
```java
package test;
import org.junit.Test;
import model.*;
import static org.junit.Assert.assertNotNull;

public class testCard {

    @Test()
    public void testWorker() {
        Worker card1 = new Worker(null, 20, 2, 10, 3, 5);
        assertEquals(20, card1.getCost());
        assertEquals(2, card1.getStone());
        assertEquals(10, card1.getWood());
        assertEquals(3, card1.getKnowledge());
        assertEquals(5, card1.getTiles());
        assertNotNull(card1.getName());
    }

    @Test()
    public void testBuilding() {
        Building card2 = new Building(null, 20, 3, 2, 10, 3, 5, false);
        assertEquals(20, card2.getCost());
        assertEquals(20, card2.getPoint());
        assertEquals(2, card2.getStone());
        assertEquals(10, card2.getWood());
        assertEquals(3, card2.getKnowledge());
        assertEquals(5, card2.getTiles());
        assertEquals(false, card2.getIsMachine());
        assertNotNull(card2.getName());
    }
}
```

Test GameSettings :



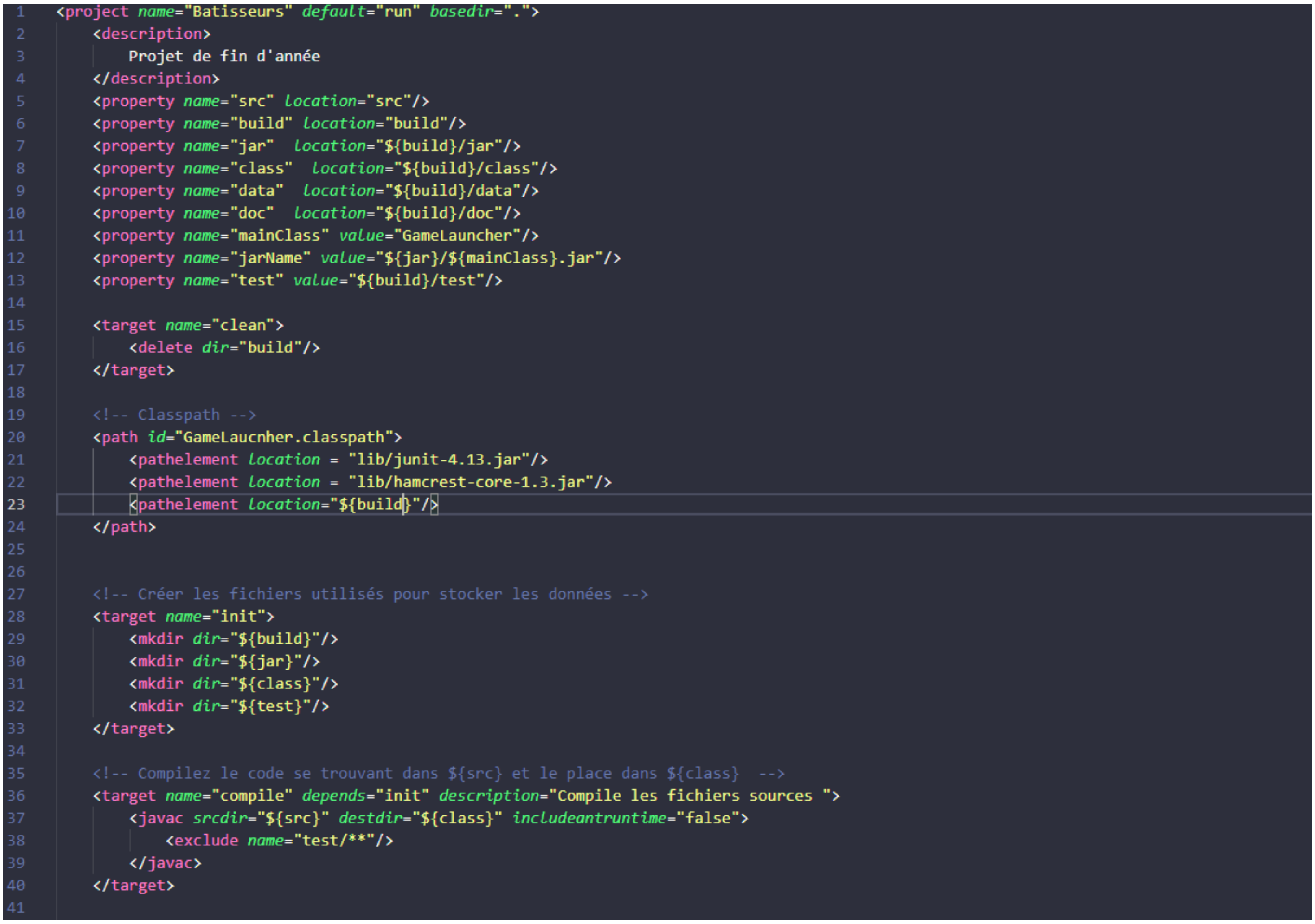
```java
package test;
import org.junit.Test;
import model.*;
import static org.junit.Assert.assertNotNull;

public class testGameSettings {

    @Test()
    public void testSettings() {
        GameSettings game = new GameSettings(null);

        assertNotNull("param null",game);
    }
}
```

Test Game :

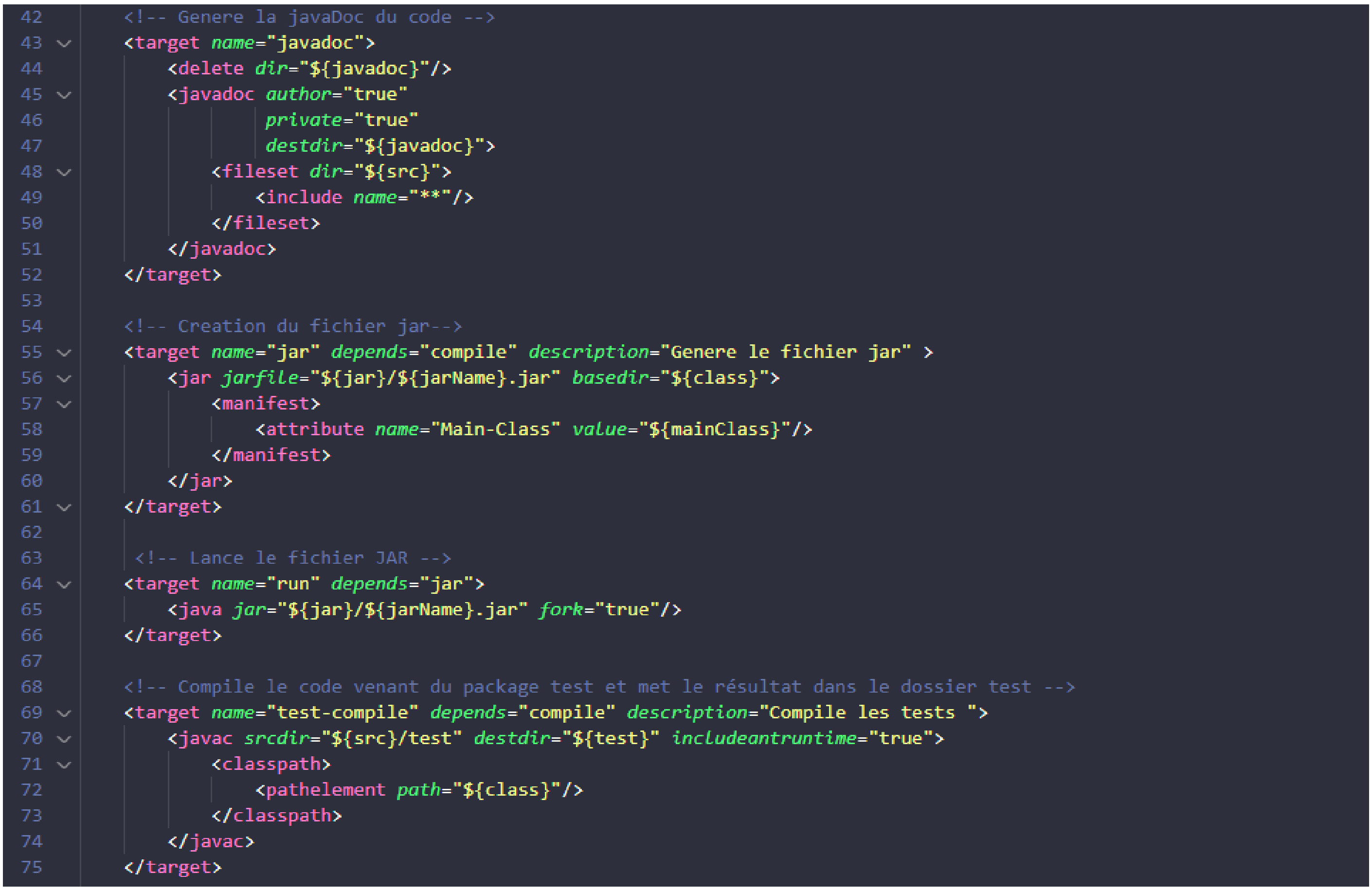


```java
package test;
import org.junit.Test;
import model.*;
import static org.junit.Assert.*;

public class testGame {
    @Test
    public void testInitGame() {
        Game game1 = new Game();
        game1.initGame();
        assertNotEquals(null, game1.getCurrentPlayer());
    }

    @Test
    public void testStart() {
        Game game2 = new Game();
        game2.start();
        assertEquals(false, game2.isWon());
        assertNotEquals(null, game2.getCurrentPlayer());
    }

    @Test
    public void testEnd() {
        Game game3 = new Game();
        game3.start();
        game3.end();
        assertEquals(true, game3.isWon());
        assertNotEquals(null, game3.getCurrentPlayer());
    }

```

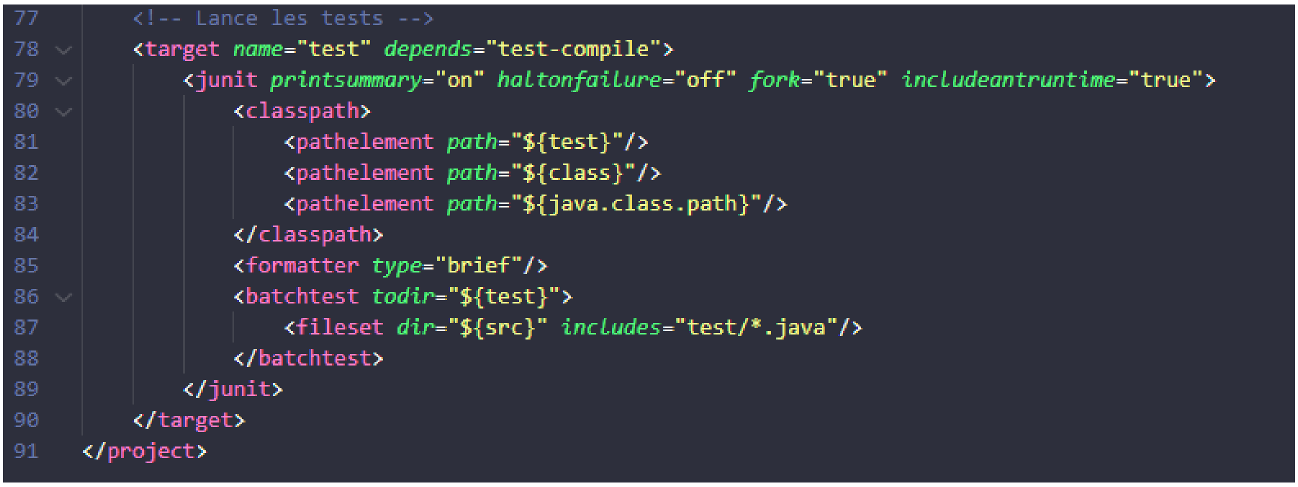
7) ANT



```xml
<project name="Batisseurs" default="run" basedir=".">
    <description>
        Projet de fin d'année
    </description>
    <property name="src" location="src"/>
    <property name="build" location="build"/>
    <property name="jar"  location="${build}/jar"/>
    <property name="class"  location="${build}/class"/>
    <property name="data"  location="${build}/data"/>
    <property name="doc"  location="${build}/doc"/>
    <property name="mainClass" value="GameLauncher"/>
    <property name="jarName" value="${jar}/${mainClass}.jar"/>
    <property name="test" value="${build}/test"/>

    <target name="clean">
        <delete dir="build"/>
    </target>

    <!-- Classpath -->
    <path id="GameLaucnher.classpath">
        <pathelement location = "lib/junit-4.13.jar"/>
        <pathelement location = "lib/hamcrest-core-1.3.jar"/>
        <pathelement location="${build}"/>
    </path>


    <!-- Créer les fichiers utilisés pour stocker les données -->
    <target name="init">
        <mkdir dir="${build}"/>
        <mkdir dir="${jar}"/>
        <mkdir dir="${class}"/>
        <mkdir dir="${test}"/>
    </target>

    <!-- Compilez le code se trouvant dans ${src} et le place dans ${class}  -->
    <target name="compile" depends="init" description="Compile les fichiers sources ">
        <javac srcdir="${src}" destdir="${class}" includeantruntime="false">
            <exclude name="test/**"/>
        </javac>
    </target>

```



```xml
    <!-- Genere la javaDoc du code -->
    <target name="javadoc">
        <delete dir="${javadoc}"/>
        <javadoc author="true"
                 private="true"
                 destdir="${javadoc}">
            <fileset dir="${src}">
                <include name="**"/>
            </fileset>
        </javadoc>
    </target>

    <!-- Creation du fichier jar-->
    <target name="jar" depends="compile" description="Genere le fichier jar" >
        <jar jarfile="${jar}/${jarName}.jar" basedir="${class}">
            <manifest>
                <attribute name="Main-Class" value="${mainClass}"/>
            </manifest>
        </jar>
    </target>

     <!-- Lance le fichier JAR -->
    <target name="run" depends="jar">
        <java jar="${jar}/${jarName}.jar" fork="true"/>
    </target>

    <!-- Compile le code venant du package test et met le résultat dans le dossier test -->
    <target name="test-compile" depends="compile" description="Compile les tests ">
        <javac srcdir="${src}/test" destdir="${test}" includeantruntime="true">
            <classpath>
                <pathelement path="${class}"/>
            </classpath>
        </javac>
    </target>
```



```xml
    <!-- Lance les tests -->
    <target name="test" depends="test-compile">
        <junit printsummary="on" haltonfailure="off" fork="true" includeantruntime="true">
            <classpath>
                <pathelement path="${test}"/>
                <pathelement path="${class}"/>
                <pathelement path="${java.class.path}"/>
            </classpath>
            <formatter type="brief"/>
            <batchtest todir="${test}">
                <fileset dir="${src}" includes="test/*.java"/>
            </batchtest>
        </junit>
    </target>
</project>
```