Bourret Emilie

Code Documentation

Project : Interactive Fiction

Poo Coo

Table of Contents:

I - Game description:

1- Game:

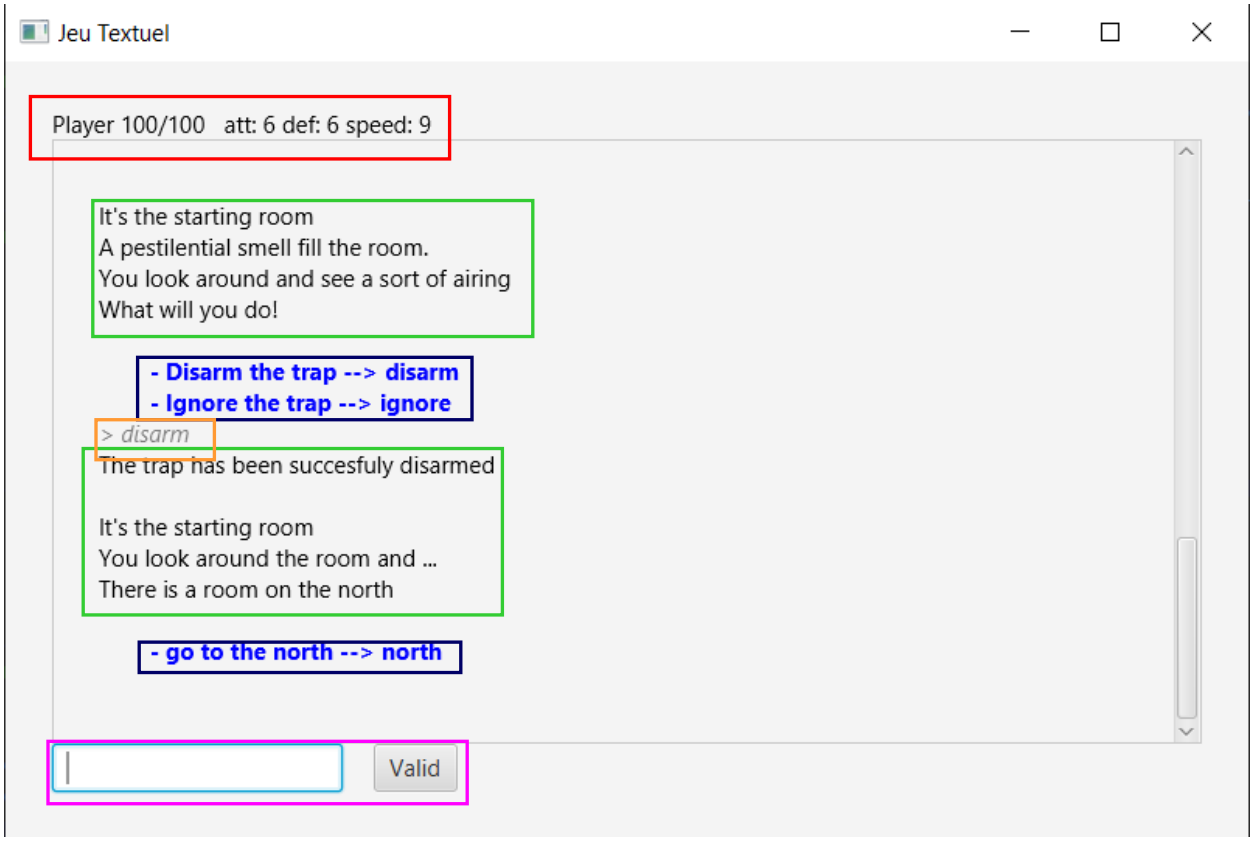
This game is a basic game where the goal is to attain the end of the game safely.
The narration is slightly based on the movie "*Tron legacy*".

To do so, the player will have to wander in a dungeon separated in rooms which contains different kind of events such as a battle, traps, chest to open and shortcut leading to other rooms.

The player will encounter enemy who will act in combat based on their health and on the player's action. The enemies have different capacity and can carry a weapon.
The player will also obtain items such as keys to open chest with lockers, potions to heal his wound and weapon to do greater damage.

2- Interface:

The application is made with javaFx which means it features a window view of the game.



There are various elements which compose the game

In red we have the player's information composed of their name ("*player*"), health, attack points, defence points and speed points

In green there is the text part which is used to describe the environment. It can contain the room's or an enemy's description or other information.

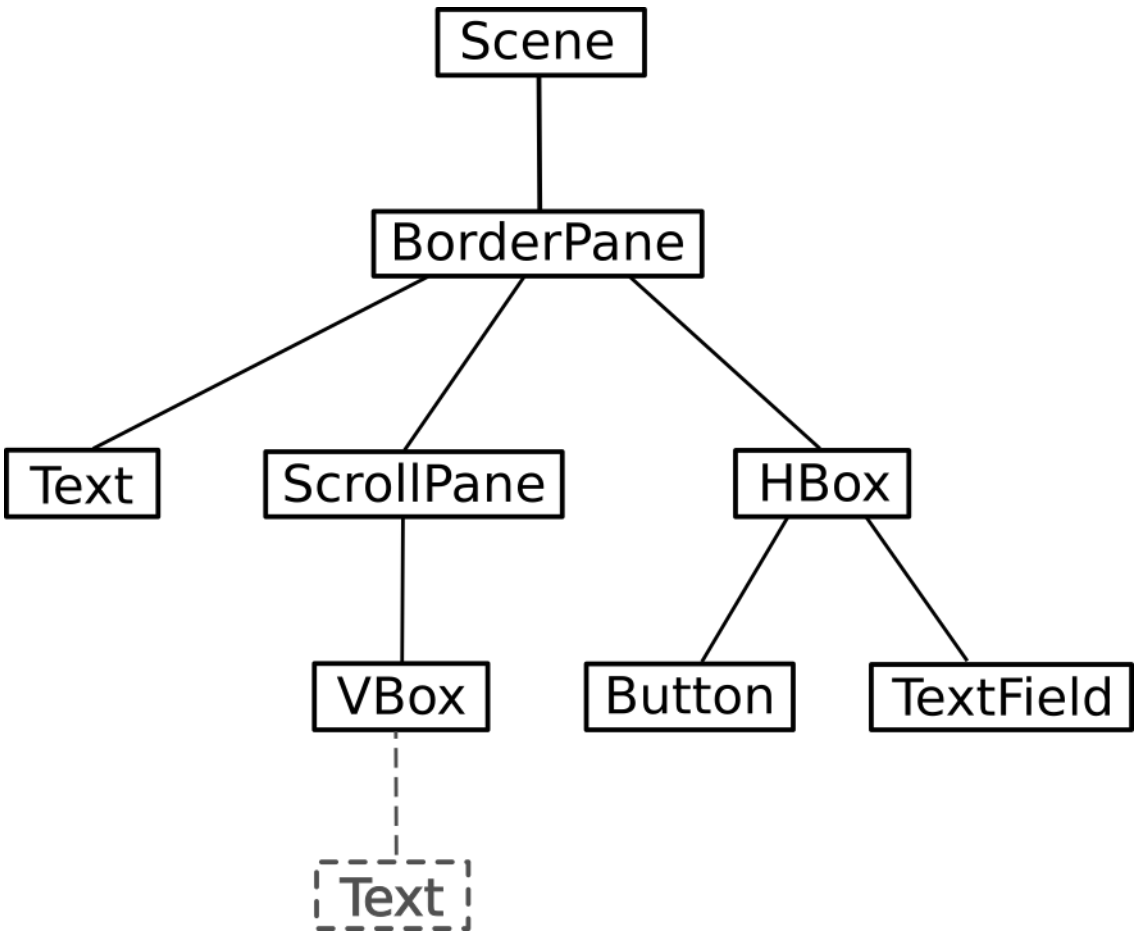
In black, we can see all the command that the player is allowed to make (there are command such as "*help*" which are always accessible)

In orange we have the player's response to the action. Only some valid terms will be written, else, it will write an error

```
Error - invalid input
Error - empty input
```

Finally, in pink, we have the text field were the player will write their answer.

Architecture



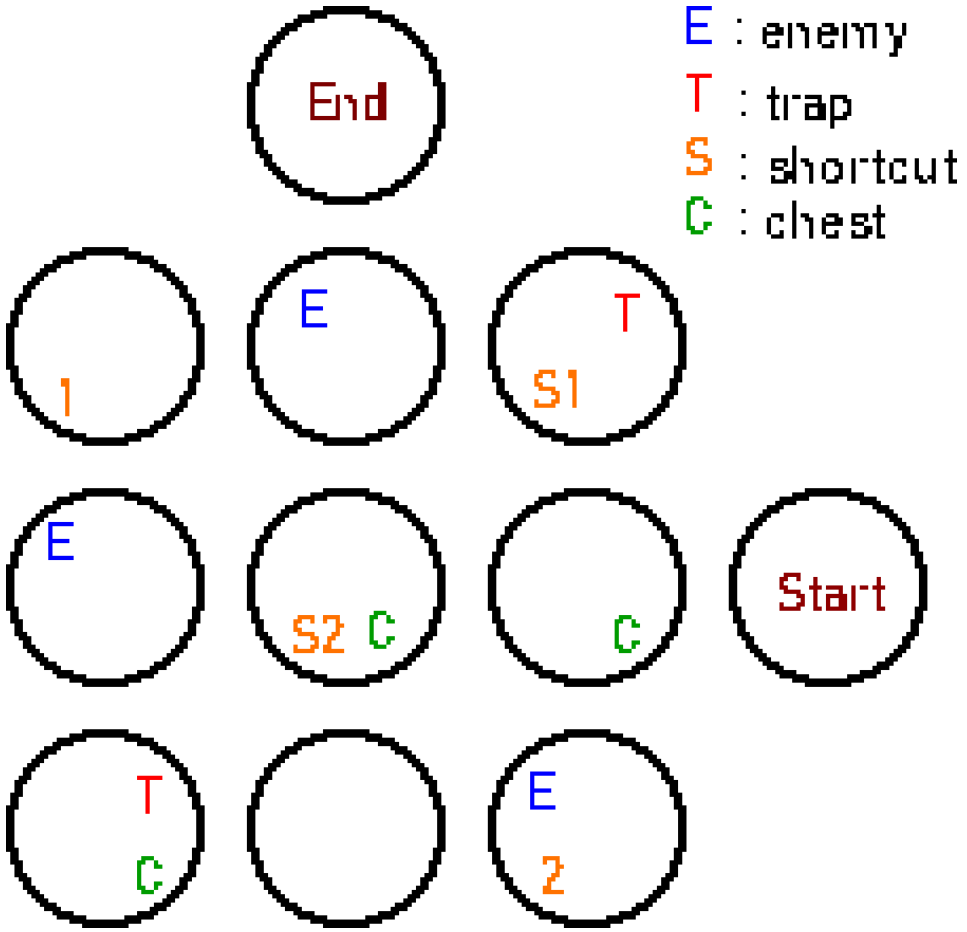
BorderPane contains the *ScrollPane* at the center of the screen. It also as a *Text* which is used to display the player's informations and which fx id is playerInfo. Plus, it has an *HBox*.

The **ScrollPane** has a *VBox* in it where the *Text* of the game is sent. Its fx id is scrollPane and the VBox one is container.

Finally, in the **HBox** we can find the *Button* to validate an input and the *textField* for the player to write this input. The TextField as input as its fx id.

3- Map:

Here is the map used for the game.



The map is constituted as a 2 dimensional array where each room as four direction. The path between these rooms is generated randomly excepted for a path going from the start to the end.

This path is here to ensure an ending to the game.

4- Function:

Name	Description	Problem	Importance
F0	**Field and generation**		
F0.0	Do a map with the four cardinal directions	*none*	max
F0.1	Implement the move between rooms	*none*	max
F0.2	Generate layout of room randomly	Layout with successful path	low
F1	**Ending of the game**		
F1.0	Game ending (win and lose)	Isn't accurate	medium
F1.1	Player can restart the game	*none*	medium
F1.2	Player can shut down the game	*none*	low
F2	**Gestion of events**		
F2.0	Create event from xml	*none*	medium
F2.1	Gestion of the player's input	*none*	max
F2.2	Player can ask to display the command available	*none*	medium
F2.3	Player can ask to display their inventory	*none*	low
F3	**Shortcut events**		
F3.0	Shortcut event (player can take a shortcut)	*none*	low
F3.1	The shortcut leads to another room	*none*	low
F4	**Chest events**		
F4.0	Chest event (player can open chest)	*none*	low
F4.1	Add items in inventory	*none*	medium
F5	**Battle events**		
F5.0	Battle event (player can enter in a fight)	*none*	medium
F5.1	Do a battle where the player make the tactical choice (attack, defence, …)	*none*	low
F5.2	Player can heal themselves	*none*	low
F5.3	Make enemy react	*none*	low
F5.4	Enemy can flee the fight	The enemy can't escape the fight	low
F6	**Trap events**		
F6.0	Trap event (player can disarm the trap)	*none*	medium
F6.1	Player can ignore the trap or enemy	*none*	low

5- Algorithm:

In the event Battle, two opponents are fighting. We have therefore the need for an enemy to fight against. The goal is, here, to create ana algorithm to make this enemy act.

We want the enemy to react to its health and to our actions.

First, we can simplify the health of the enemy by categorizing it as either low, medium or high.

Second, we can divide the action in battle between two types: offensive and defensive

Attack / special attack / defend / heal / flee

It give us a table with the different action and health:

	Player is offensive	Player is defensive
Enemy life is low	Heal / attack	Flee / defend
Enemy life is medium	Special attack / heal	Special attack / attack
Enemy life is high	Special attack	Attack

We can also see the actions as Boolean: true if the action was successful and false if it wasn't.

If we look at the actions we note that defend is the only one that has a chance of 100% to be successful.
It means that we can use it has the action executed by default (and we can remove it from the table)

The action will then, be determined in a function which will return a Boolean and if this one is false then the action defend will be executed.
There can be two action for a particular case, in this case, the action will be determined randomly with Math.random().

Thus, it means that an action will be executed by a main function and if this one fail, the enemy will defend itself.

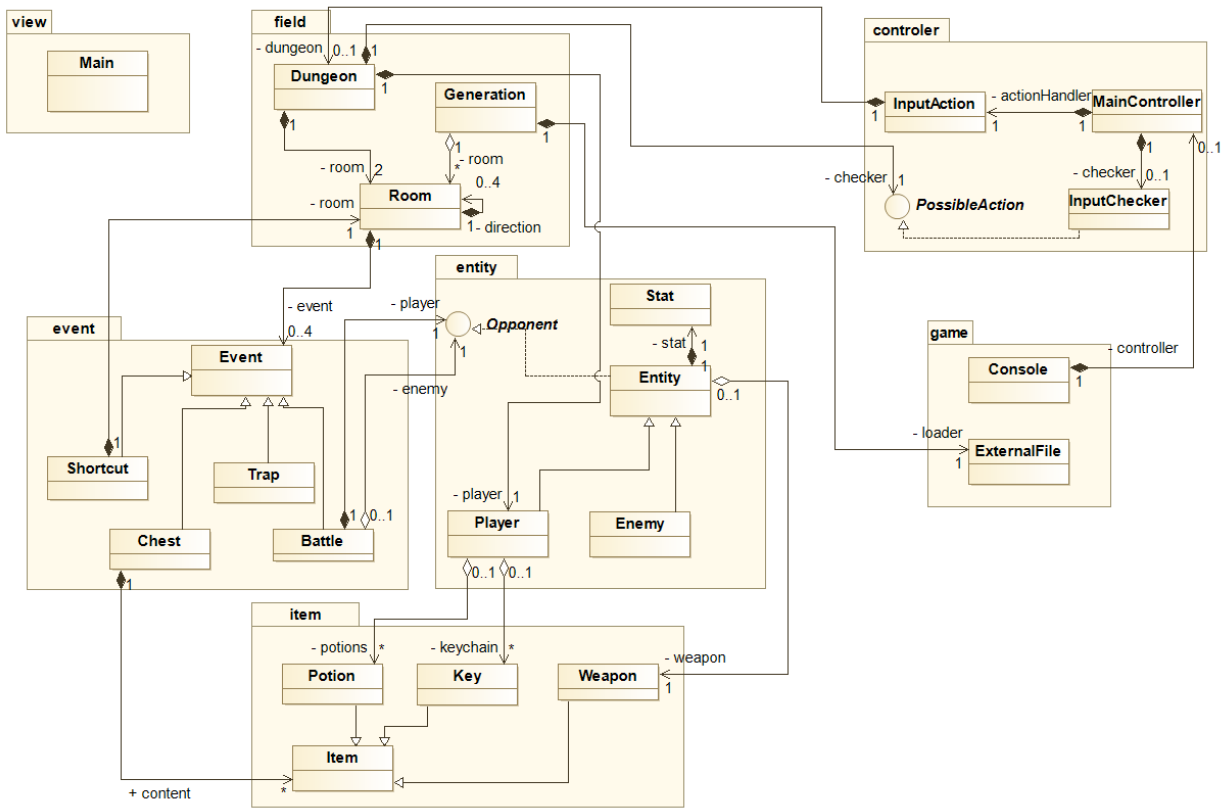
II - Code documentation:

1- Organisation:

The code is organized with the logic of the model-view-controller.

Therefore, there is seven packages composing the code: the controller, the view, and the others which five which represents the model part of the application.

2- Global view:



Here is the package diagram of the game. It was simplified to show only the important links but there are a little more association.

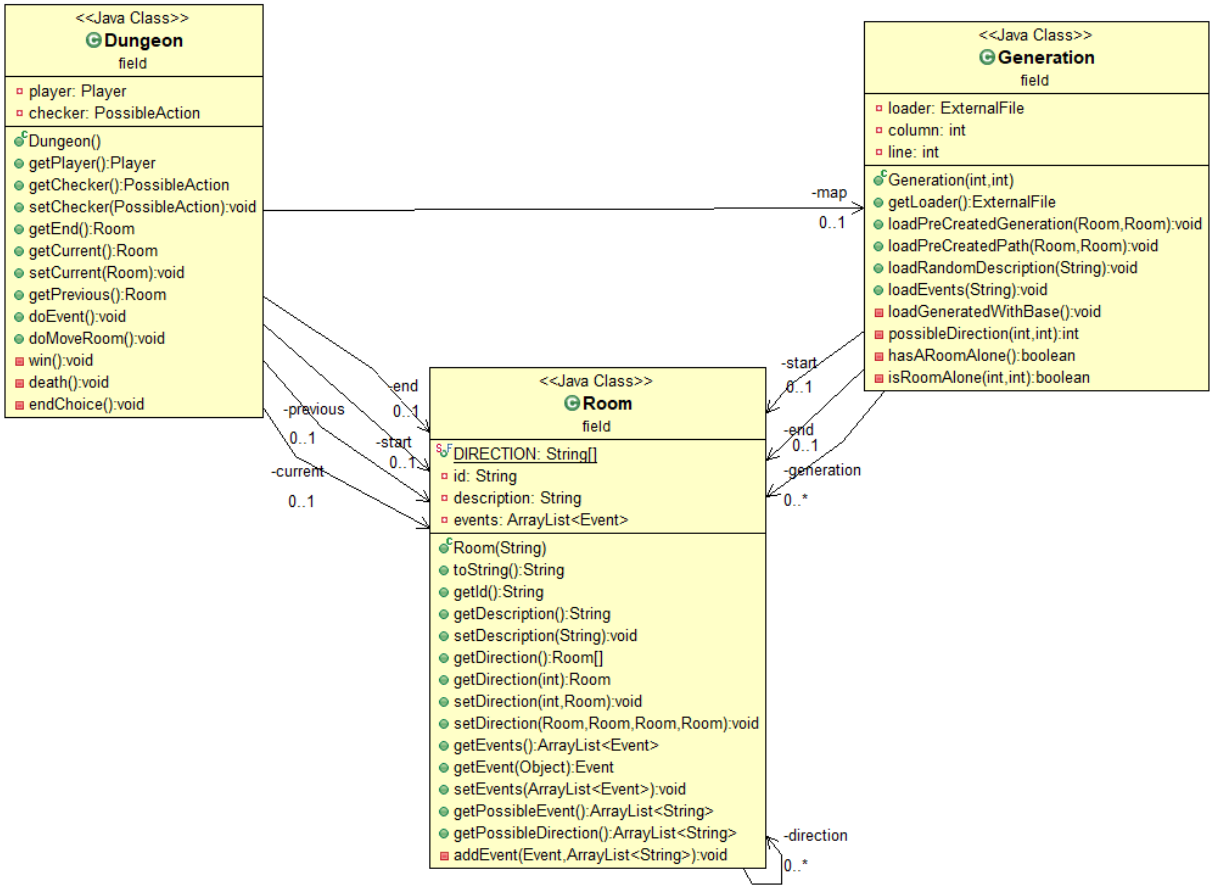
The different class and their attributes and methods will be explained in detail below.

View

The package view contains only the main class which load the interface of the application and instantiate the class MainController. It is not used in the game anymore after that.

3- Packages:

Field



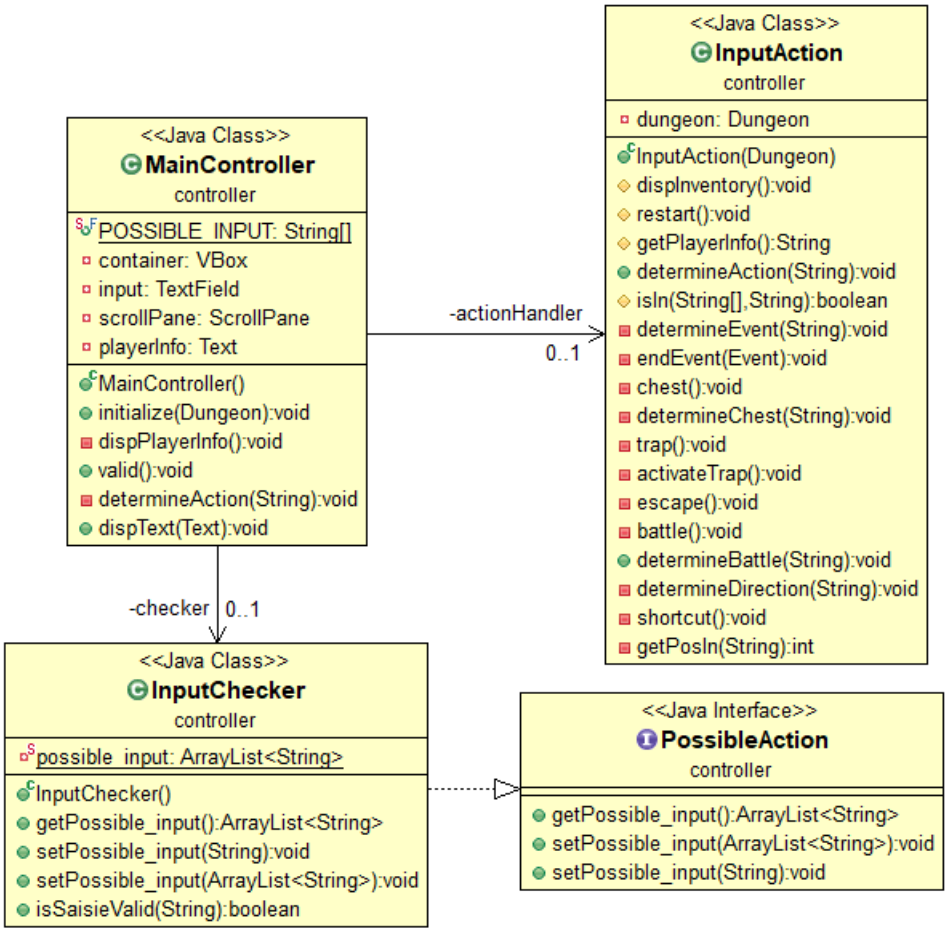
The package field contains all the class in relation with the “physical” parts of the game: where the player will physically be.
There are only three classes: Dungeon, Room and Generation.

Dungeon is the main class of the game and is composed of a *player* and a *checker* (PossibleAction). It has a *current* and a *previous* Room and is composed of a *start* and *end* Room. It also has a *map* (Generation) It define the action available for the player with its checker with the methods doMoveRoom() and doEvent().

Generation is the class used to load xml data and instantiate all the rooms. Therefore, it has a *generation* of rooms and is composed of a *start*, *end* room and a *loader* (ExternalFile)

Room is the class that define each room. They are composed of *events* (Event) and other rooms which are the *direction* where the player can go after entering a room.
There are method in this class to set the possible actions the player can do in a particular room.

Controller



This package contains all the class relative to the interface and the execution of actions.
It has 3 classes and an interface.

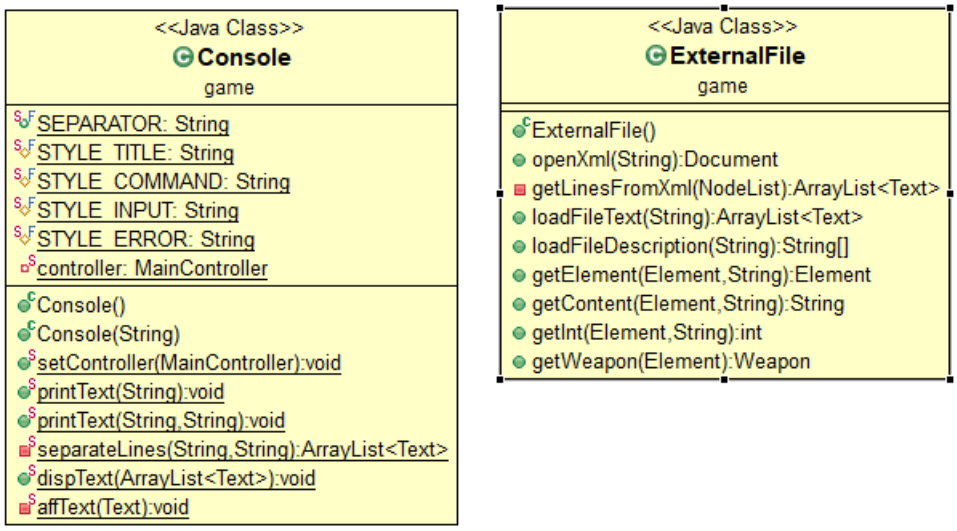
MainController is the class where the game handles the javaFx objects. It is where the method to get the input of the player is. This method needs a *checker* (InputChecker) and an *actionHandler* (InputAction) which compose this class.

InputChecker is the class which contain the possible action that are available for the player and verify if the input corresponds to one of them.

PossibleAction is the interface where InputChecker is seen only has a setter of the possible action. It is used by the Dungeon class.

InputAction take an input that has been verified and determine which action it should do. Once it has done so it will execute it and modify the *dungeon* that compose it.

Game

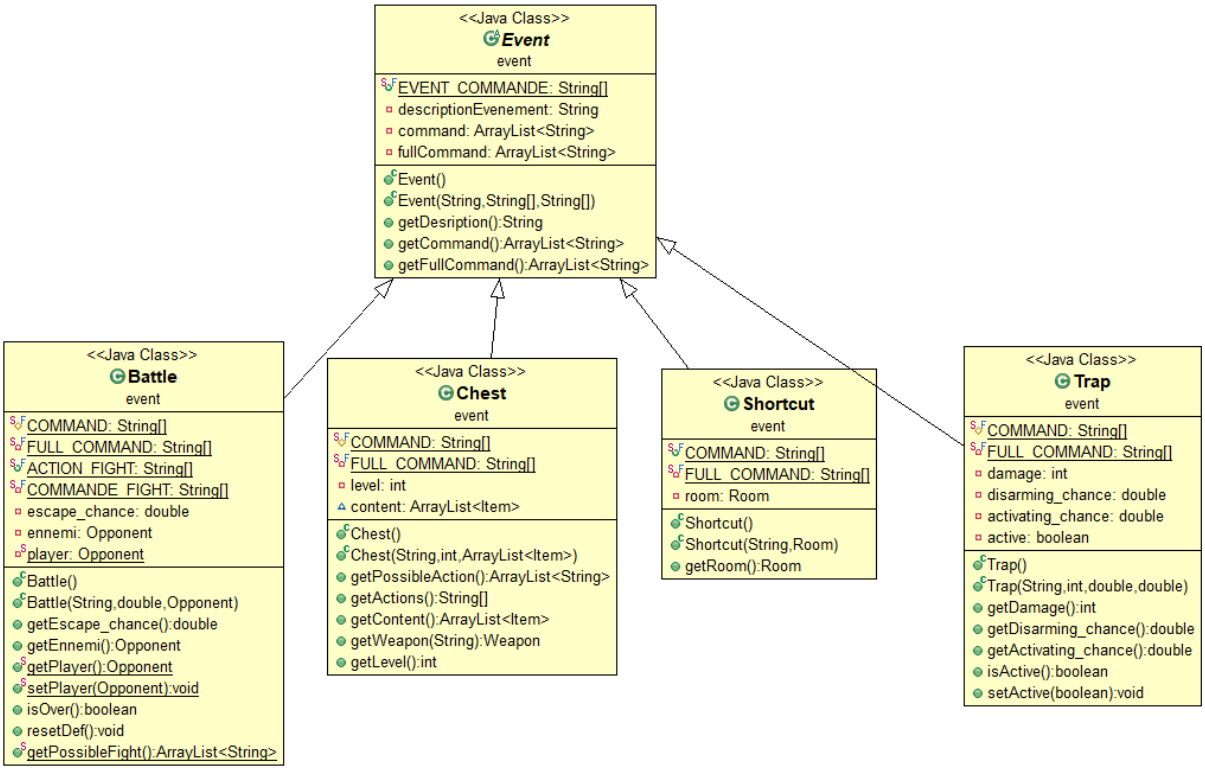


It is the package dedicated to the class that have a relation with the text displayed in the game. It contains only two classes: Console and ExternalFile.

ExternalFile is the class that load external files with method such as openXml() and loadFileText() it recognize xml formatted to display a text (with titles, paragraphs, …), to load descriptions for the rooms and to load events (this part is practically entirely in Generation).

Console is a special class where the attributes are mainly static. The reason is that it is used to display texts in the game. It also features a gestion of the style of the text. It is composed of a *controller* (MainController) where the text is sent.

Event



The package event is the package related to all the events which could happen in the game. It contains data about which actions the player can make.
It has a class Event from which every other class of the package inherit.

Event is the class Parent of the others. It contains a description of the event and the command related to it. It is used by the Room class to determinate which action are available in a room.

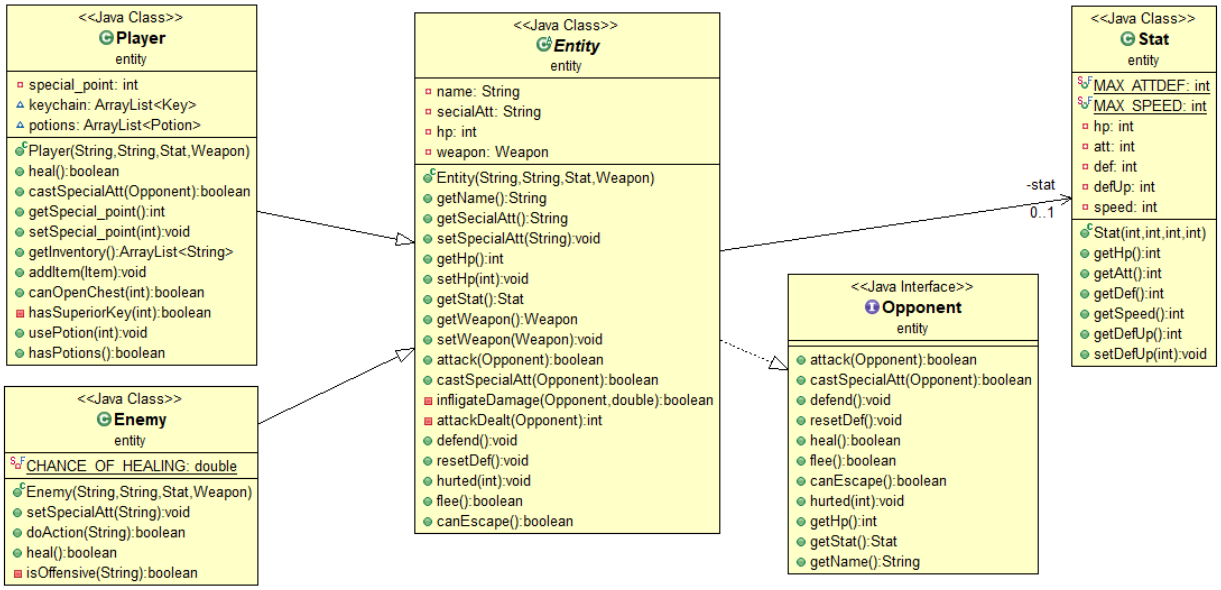
Shortcut, the simplest class of this package is composed of a *room* where the shortcut lead. The player can go down this room instead of following one of the directions of the room he is in.

Trap is the event where the player can disarm a trap or decide to ignore it. Both actions have a chance of success and if the player fail, he will get hurt.

Chest is the event where the player can gain more objects. It has a *content* (Item). The chest can be locked by a locker with a level which needs a key with at least the same level. If it contains weapons, the player will be given the choice of equipping it.

Battle is the event where the *player* (who composes the event) fight against an opponent, his *enemy*. The player can try to escape instead of fighting. If the player engages in a fight, he will be given a set of action which are defined in Battle.

Entity



This package is for the entity of the game there are only two types of theme: the player and the foes. It defines the player and its enemies.
It has 4 different classes and an interface.

Entity is the class which define an entity: their name, health, *weapon* and *stat*. An entity also has a special attack that causes more damage but cost point (that are reset at the end of a battle). It also defines the main classes of Opponent.

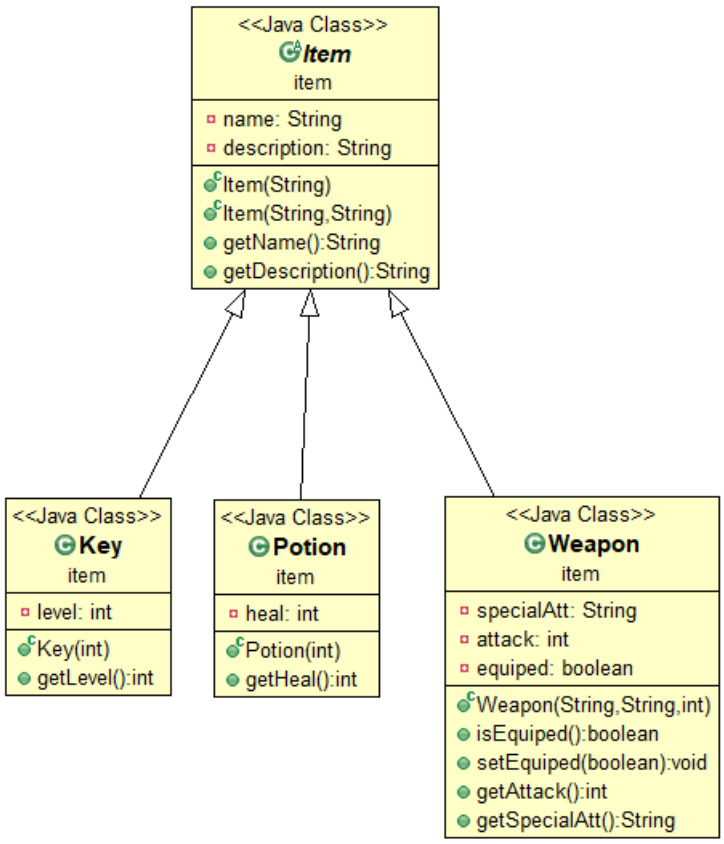
Stat is a simple class which contain the statistics of an entity: its defence, attack, speed, and total health. DefUp is the defence upgraded for a turn in a battle.

Opponent is the interface where an entity is seen as a fighter. There are method for fighting: the offensives: to attack and to do a special attack and the defensive : to defend (will up the defence for a turn), to heal and to flee the fight. These methods are mainly defined in Entity

Player is the class inheriting entity which is the avatar of the player. It has *potions* and a *keychain* to heal and to open chest. Thus, it contains method to handle the different items and it redefine the method for healing and to do a special attack.

Enemy is the class that contains the method used to determine the action of an enemy. It is the enemies the player will fight against.

Item



It's the package where the items used in the game are defined.
It's a simple package with a super class: Item and three classes which inherit it.

Item is the class that define an item as an object with a name and a description. It doesn't have a lot of method or attribute but is used to handle the keys, potions and weapons has the same type of object.

Key is the class that define the keys to open chests. It has a level to open chest with locker.

Potion is an object which can heal the player (the enemies don't need to use potion). Thus, it has an amount of point it can heal.

Weapon is the class defining the weapon of the entity. It has an increment the entity's attack by the damage it deals. It also has a special attack which supplant the one of the entities when in fight.

4- Action:

The mechanism to let the player make an action is a complex system. The game makes a decision in two times.

In the first time, the game will determine a set of action for the player depending on the context he is in. For example: if they move to another room, the class dungeon will use the method in room to get the possible actions and set them with its checker (PossibleAction) attribute.

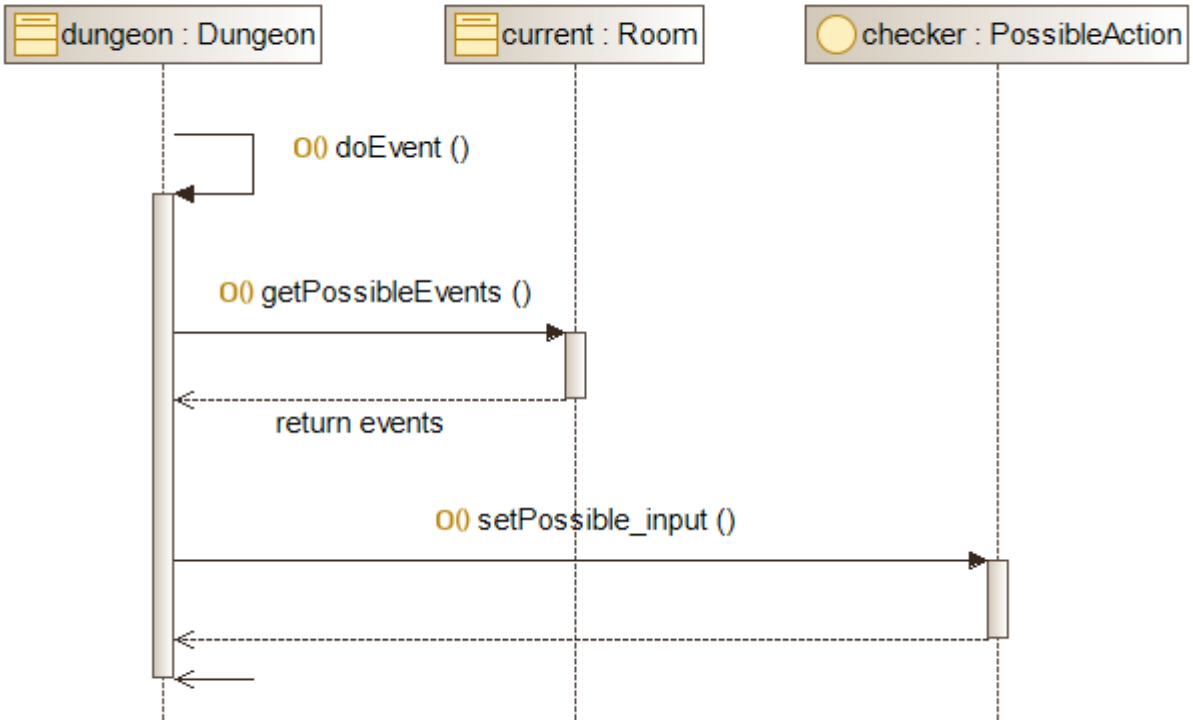
Between the two time, the player will input an action.

In the second time the game will verify the action with class MainController using the checker (InputChecker) and then send it to its actionHandler (InputAction) attribute. Then, in this class (InputAction), the action will be determined (example: if the player write "*fight*", the class will determine that it's an input referring to the battle event).

It will then execute the action.

Sequence diagram

To understand this mechanism better, we will take as example, the case of the user entering a room where there is an event Battle and where the player starts a fight.



First, the player enter as entered a room and the method **doEvent()** from dungeon is called.

This method calls **getPossibleEvents()** from the current Room which return an array of Strings containing the action that the player can write. In this case, there will be "*fight*".

Then, the method **doEvent()**, send these strings to the checker with **setPossible_input()** and the checker add theme to the list of possible actions.

The player will then write the word "*fight*" and press the "*Valid*" button of the game which calls **valid()** from the class MainController.

The method **valid()** will verify the word by calling **isSaiseValid()** from its checker. It will return true as the input is in the list of possible input in the checker.

The controller will then call **determineAction()** who will send it to **determineAction()** from the actionHandler where it will be determined that this is an action of type event. It will call **determineEvent()** where the method **battle()** will finally be called.

Thus, the process of determining the action is done. It means that the game as to set others possible actions.

The method **getChecker()** will return the checker of the dungeon and **setPossible_input()** will be called again with the action in the Battle event so that the player can start the fight.