ZUUL Dokumentation

@author Noelle Senti
@version November 2019

//M226, ZUUL Projekt.

```
/**
 * Ein interaktives Spiel, welches
 * ausschliesslich auf der Konsole läuft,
 * ohne Bilder oder anderen Hilfsmittel.
 * Alles läuft über Text und simple
 * ausgeschriebene Befehle.
**/
```

Inhaltsverzeichnis

Analyse

Die Aufgabe der einzelnen Klassen

```
public class Character {
      String name;
      Thing neededThing;
      String output;
}
```

```
/**
*Die Character Klasse ist dazu da, um
verschiedene Character zu erstellen welche im
Spiel vor kommen und mit diesen auch zu
interagieren.
**/
```

```
public class Command {
    private String commandWord;
    private String secondWord;
}
```

```
/**
*Diese Klasse hält Informationen über einen Befehl,
*der vom User eingegeben wird. Die Befehle werden
*in einer anderen Klasse überprüft.
**/
```

```
public class CommandManager {
    private ArrayList<String> validCommands;
}
```

```
/**
*Hier werden gültige Befehle gespeichert
*und ein eingegebener Befehl überprüft.
**/
```

```
public class Game {
      private final Parser parser;
      private Player player;
      private Room currentRoom;
}
```

```
/**
*Das ist die Main Klasse. Hier wird das Spiel
*ausgeführt und alle Objekte initialisiert, sowie
*miteinander verbunden.
**/
```

```
public class Interactions {
      public String interaction;
      public String output;
      public Thing object;
      public boolean done;
}
```

```
/**
*In dieser Klasse sind die Informationen zu den
*Interaktionen gespeichert, die dann zu den Räumen
*hinzugefügt werden.
**/
```

```
public class Parser {
    private CommandManager command;
}
```

```
/**
*Liest die Eingabe und trennt sie in zwei Strings,
*sofern es mehr als ein Wort beinhaltet. Leitet
*ersten Teil der Eingabe an CommandManager weiter.
**/
```

```
public class Player {
      private ArrayList<Thing> inventor;
      private ArrayList<Keys> keys;
}
```

```
/**
*Repräsentiert den User. Hier werden alle
*Sachen und Schlüssel gespeichert, die der
*Spieler zu sich nimmt.
**/
```

```java
public class Room {

	private final String description;
	private String name;
	private Interactions interaction;
	private Character character;

}
```

```java
/**
*Hier werden die Informationen zu den Räumen
*gespeichert, diese Klasse gehört womöglich
*zu den mit wichtigsten Klassen.
**/
```

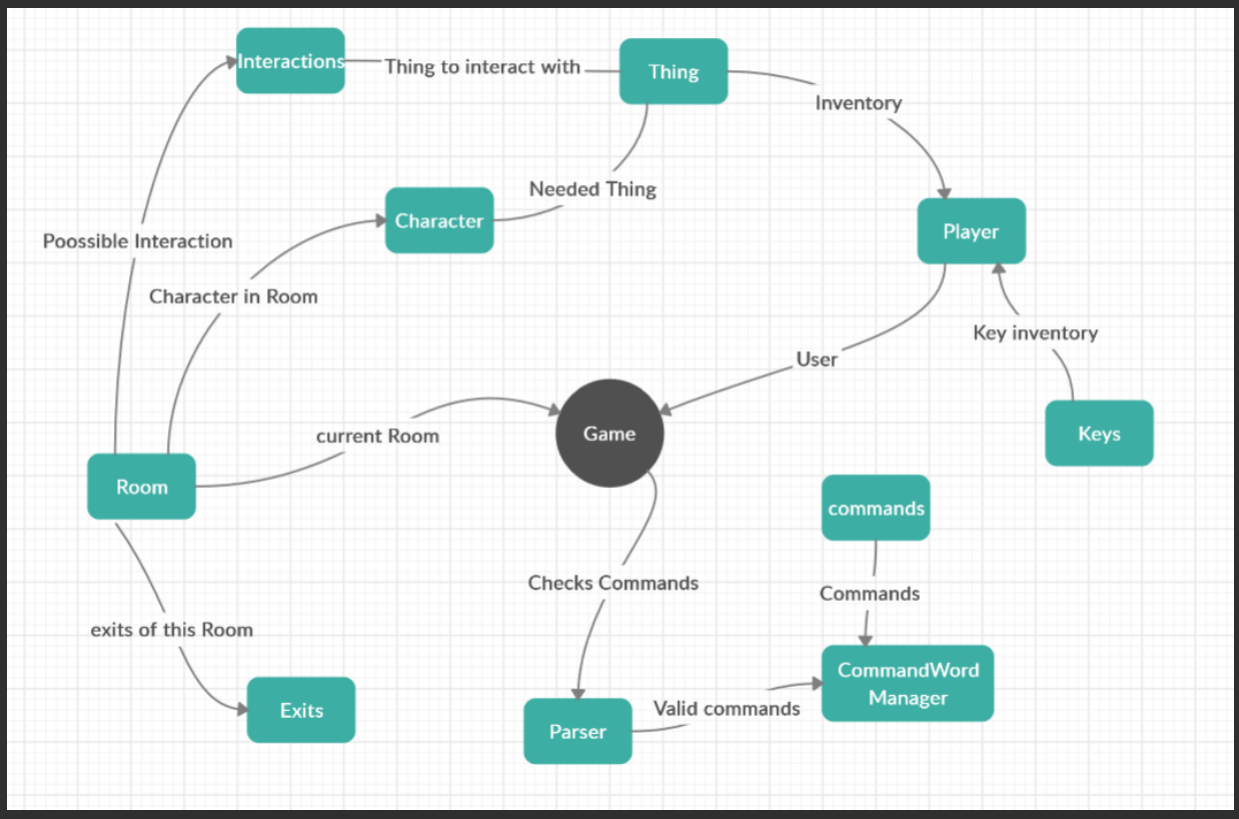
```java
public class Thing {
	private String name;
	private boolean used;

}
```

```java
/**
*Das ist die Thing Klasse, in welcher alle
*Sachen gespeichert werden, die der Spieler
*aufheben und anderweitig benutzen kann.
**/
```

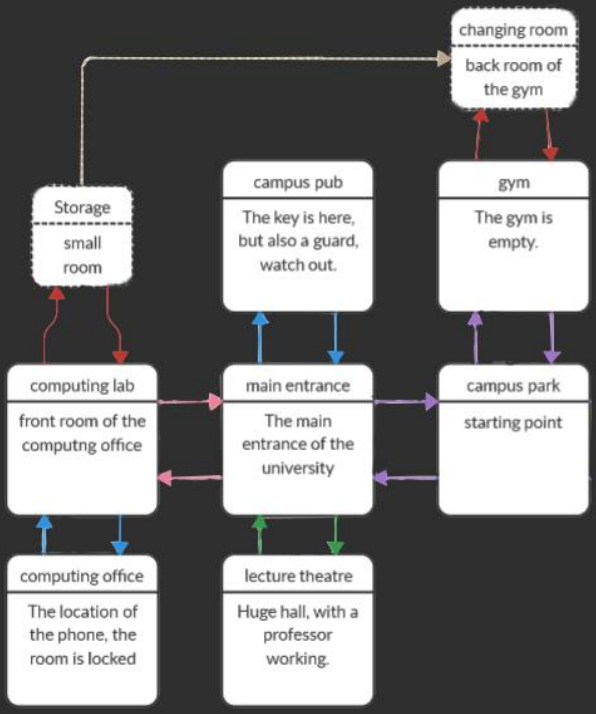
```java
public class Exits {
	String name;
	String description;
	Room room;
	Thing neededThing;
}
```

```java
/**
*Speichert die verschiedenen Ausgänge und die
*dazugehörigen Räume, sowie (optionale)
*nötige Gegenstände um den Durchgang
*freizuschalten.
**/
```

Beziehungen der Klassen



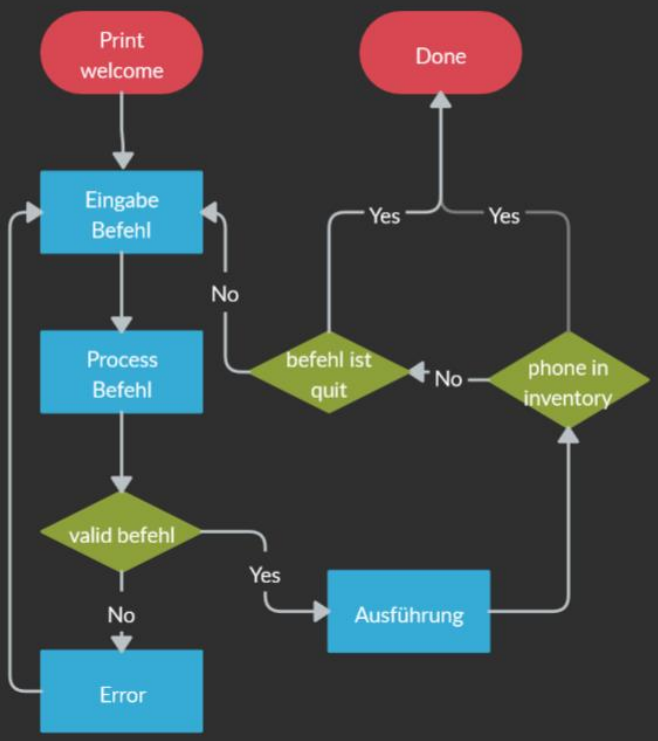
Room Map



Es hat neun Räume, in die man gehen kann. Das computing office ist vorerst abgesperrt. Die Verbindung zwischen dem storage und dem changing room ist nur in eine Richtung möglich. Verschiedene Ausgänge werden während dem Spiel auch gesperrt.

Jeder Raum muss betreten werden um das Spiel abzuschliessen, aber nicht jeder Raum hat einen Zweck.

Ablauf Skizze



Sobald man einen Befehl eingegeben hat, wird überprüft, ob er gültig ist. Ist er es nicht, wiederholt es sich wieder. Ist er gültig, läuft das Programm weiter und der Befehl wird ausgeführt. Führt der Befehl dazu, dass der Spieler sein Handy bekommt, oder ist der Befehl quit, endet das Spiel.

Ansonsten läuft es wieder von vorne weiter.

Gewonnen

Man hat gewonnen, wenn man sein Handy wiederbekommen hat. Am Anfang des Spiels wird dies als das Ziel angegeben;

```
System.out.println();
System.out.println("Welcome to Adventure!");
System.out.println(
      "Adventure is a new, incredibly thrilling adventure game.");
System.out.println("Your phone was taken by a professor at your school!");
System.out.println("Go to the computing office to get it back.");
System.out.println("Type 'help' if you need any help or clues.");
System.out.println();
```

Bis man in das Computing Office kommt, und somit auch zu seinem Handy, muss man einige Aufgaben machen.

Gegenstände

Gegenstände können aufgesammelt werden oder von Charakteren erhalten werden. Sie werden dem Spielers Inventory hinzugefügt. Die Objekte kann man benutzen, um mit Charakteren Objekte oder Informationen auszutauschen, oder Ausgänge freizuschalten, wie zum Beispiel das abgeschlossene Office mit dem Schlüssel, oder den unzugänglichen Luftschacht mit der Kiste.

```
Thing phone = new Thing("phone", false);
Thing leaves = new Thing("leaves", false);
Thing empty = new Thing("", true);
Thing key = new Thing("key", false);
Thing phoneNumber = new Thing("phone number", false);
```

To-Do-List

- Exits implementieren
- Neun Räume erstellen
- Soll Spiel gewinnen können
- Gegenstände implementieren
- Diagramme für Doku
- Beschriftung Räume
- Verbindung Raum-Exit
- Kommentare zu Klassen einfügen (im Programm)
- “Storyline” weiterführen
- Javadoc erstellen

Story

Man ist ein Student, der sein Handy zurückwill, doch es stellt sich heraus, das ist nicht so einfach wie es erst wirkt. Kommt man zum Büro, merkt man es ist abgeschlossen. Da kommt einem in den Sinn, man könne den Lab Professor fragen wo er sei. Der wird das einem sagen, gibt man ihm erst ein paar Blätter. Danach geht man in das Pub, wo der Schlüssel sein soll, wird allerdings während dem Suchen erwischt. Man rennt in einen Abschlusskammer, wo man anschliessend durch den Lüftungsschacht flieht. Man kommt im Gym wieder heraus und kann dann den Schlüssel holen. Anschliessend geht man ins Office und schnappt sich sein Handy.