Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

Mid-term Test (worth 10% of overall mark)

Introduction

In this test you will be working on a text-based file format for an adventure game. The adventure game world consists of Rooms connected together by Doors, and Items which are located in Rooms. An example game file is the following:

```
Room #1 { name: "Dining Room" }
Room #2 { name: "Lounge" }
Room #3 { name: "Kitchen" }
Door #4 { from: 1, to: 2 }
Door #5 { from: 2, to: 3 }
Item #6 { location: 1, description: "Candle" }
```

Each line of the file corresponds to a FileObject organised in the following manner:

```
Kind ID { field: value, ... }
```

Here, the Kind of object is either Room, Door, or Item. The ID is a unique number assigned to each object. Finally, field is the name of a field for the object in question, whilst value gives a value for that field (either a string or integer).

Download

You can download the code provided for the game file format from here:

`http://ecs.victoria.ac.nz/Courses/SWEN221_2011T1/Mt`

You will find several Java source files, including a JUnit test file.

Submission

You should submit your solutions through the usual assignment submission system. Please make sure you submit to the correct session. The URL for submission is:

`http://ecs.victoria.ac.nz/cgi-bin/auth/submit?course=SWEN221`

Late submissions will get zero marks (unless you have arranged this with us, which will only be in exceptional circumstances).

PLEASE TURN OVER

1 Debugging the Parser (worth 10%)

You should begin by importing the code provided into Eclipse and running the JUnit tests. You will find that the first three tests fail. You will find information regarding these errors is printed into the Eclipse console, which you may find helpful.

There are several bugs in FileParser.java, and your aim is to find and correct these mistakes. Upon completing this, you should find that tests validFile_1, validFile_2 and validFile_3 now pass. **NOTE:** you should also find that many or all of the other tests now fail!

2 Checking Object Fields (worth 20%)

This part concerns the following requirements for the game file format:

1. A Room object must have a "Name" field which is a String.
2. A Door object must have "from" and "to" fields which are both Integers.
3. A Door object may not connect a Room to itself (i.e. from $\neq$ to).
4. An Item object must have a "location" field (which is an integer), and a "description" field (which is a String).

You should find that some or all of the tests invalidFile_1, ..., invalidFile_6 currently fail. This is because there are a number bugs in FileChecker.java. Your aim is to fix these bugs, such that tests invalidFile_1, ..., invalidFile_6 now pass.

3 Checking for Duplicate Objects (worth 20%)

This part concerns the following requirements for the game file format:

6. No two FileObjects in a game file may have the same id.

You should find that some or all of the tests invalidFile_7, ..., invalidFile_10 currently fail. This is because the method checkForDuplicateIDs() in FileChecker.java is empty. Your aim is to implement this method appropriately, such that tests invalidFile_7, ..., invalidFile_10 now pass.

4 Checking Room References (worth 20%)

This part concerns the following requirement for the game file format:

8. Every Door object must connect two valid Room objects together (i.e. fields "from" and "to" must refer to valid Room objects). Likewise, every Item object must be located in a valid Room (i.e. the field "location" must refer to a valid Room object).

You should find that some or all of the tests invalidFile_11, ..., invalidFile_13 currently fail. This is because the method checkValidRoomReferences() in FileChecker.java is empty. Your aim is to implement this method appropriately, such that tests invalidFile_11, ..., invalidFile_13 now pass.

PLEASE TURN OVER

5 Adding Containers (worth 20%)

This part concerns an extension to the game file format to support "containers". A container is an object which may hold other objects, and is located in a room. The following illustrates:

```
Room #1 { name: "Dining Room" }
Room #2 { name: "Lounge" }
Door #3 { from: 1, to: 2 }
Container #4 { location: 1, description: "Box" }
Item #5 { location: 4, description: "Candle" }
```

Here, we see that object 4 is a Container located in Room 1. Furthermore, Item 5 is located in the container.

The following requirements are given for containers:

9. Every Container object must have a "location" field (which is an integer that identifies a room or container) and a "description" field (which is a string).

10. No Container can be located in itself (either directly, or indirectly via another container).

We additionally extend the existing requirements for Items so they may be located in a Room or Container.

You should find that some or all of the tests validFile_4, validFile_5, invalidFile_14, ..., invalidFile_17 currently fail. This is because Containers have not been implemented in FileChecker.java. Your aim is to extend FileChecker.java to support Containers, such that tests validFile_4, validFile_5, invalidFile_14, ..., invalidFile_17 now pass.

6 Checking Reachability (worth 10%)

The final (and most challenging) requirement for the game file format is the following:

11. Every room in a game file must be reachable from every other room through one or more Doors. **Note:** you should assume doors go both ways (i.e. they form an *undirected graph*).

You should find that some or all of the tests invalidFile_18, ..., invalidFile_20 currently fail. This is because the method checkAllRoomsReachable() in FileChecker.java is empty. Your aim is to implement this method appropriately, such that tests invalidFile_18, ..., invalidFile_20 now pass.