

MODULAR PROGRAMME

COURSEWORK ASSESSMENT SPECIFICATION

Module Details

Module Code UFCFC3-30-1	Run 16SEP/1	Module Title INTRODUCTION TO OBJECT-ORIENTED SYSTEMS DEVELOPMENT
Module Leader Benedict R. Gaster, Julia Dawson	Module Coordinator	Module Tutors Stewart Green, Abdullahi Arabo, Shancang Li
Component and Element Number B: CW1		Weighting: (% of the Module's assessment) 50%
Element Description DEMONSTRATION AND PRESENTATION		Total Assignment time

Dates

Date Issued to Students 03/01/2017	Date to be Returned to Students See inside the assignment
Submission Place Blackboard	Submission Date 02/03/2017
	Submission Time 2.00 pm

Deliverables

See inside the assignment.

Module Leader Signature

Benedict R. Gaster

UFCFC3-30-1 Introduction to Object-oriented Software Development Assignment (2016-2017)

Contents

COURSEWORK ASSESSMENT SPECIFICATION.....	1
Module Details	1
Dates	1
Deliverables.....	1
Module Leader Signature	1
1 Your tasks and deliverables.....	3
2 Submission and presentation schedules	3
3 Marking criteria	4
4 Zombie Text Adventure.....	5
4.1 Overview	5
4.2 Videos demoing how the game is played	5
4.3 Information for the implementation	5
4.3.1 GUI front end.....	6
4.3.2 Using the provided Jar and Java files.....	6
Appendix A	9
Appendix B	9

Please note that this assignment specification includes links to videos on Youtube. In order to see the videos you need to have access to Youtube.

1 Your tasks and deliverables

The assignment is a group assignment, with group size minimum of 2 and maximum of 3. You are asked to produce the following deliverables for the Zombie Text Adventure described Section 4.

1. Produce a UML class diagram to meet all the requirements and corresponding to your implementation.
2. Implement the system in Java.

All UML diagrams must be done in Astah. All Java code must be done in Netbeans.

You are required to submit and to present each of the above. Submission will be done on Blackboard. Demonstrations will take place in lab sessions, following the submission.

Groups are self assigned and you must sign up via Blackboard to a specific group.

Presentations slots will be allocated on a first come basis and marks will only be assigned to group members attending the specified demonstration.

2 Submission and presentation schedules

There will be the following submission and presentations:

- *2nd March 2017: Submit the UML class diagram and corresponding implementation on Blackboard*
- *The weeks following the submission will be for demonstrations.*
- Work submitted after the final deadline is subject to the standard university rules.
- For students with extenuating circumstances, please contact student advisors and the tutors to make alternative arrangements.

3 Marking criteria

The assignment will be marked using the following marking scheme.

Elements:	Key Point:	Criterion:
Class diagram (35%)	Correct use of notations	<i>20% - Matches House style.</i>
	Appropriate identification of classes	<i>40% - Maps back to the problem domain and not single class does everything approach</i>
	Appropriate identification of attributions and operations	<i>40% - Demonstration of encapsulations, e.g. private attributes and publics methods, and separation of concerns. Good use of methods and constructors.</i>
Implementation (35%)	Consistent with the class diagram design	<i>Matching to the class diagram: Class names, attributes, operations and class relationships</i>
	Implements Zombie Server.	<p><i>There are three basic sets of functionality:</i></p> <p><i>0-40% - Interacts with web front end to allow the user to enter "look" command and displays information for current room.</i></p> <p><i>41-65% - Provides ability to move around the world, i.e. between rooms, pickup and drop items, and score is maintained and displayed by the frontend.</i></p> <p><i>66%-100% - Support for Zombies is provided, frontend displays count down timer, and player can issue kill command and other actions corresponding to Zombies. If counter reaches zero, then player dies.</i></p>
Demonstration (30%)	<p>Group demonstration, showing the Zombie server in action, along with a discussion of UML class diagram, and Netbeans project and Java source code.</p> <p>It is a requirement that each student in the group is able to present and discuss some aspect of the final solution. In particular, each individual member needs to be able to demonstrate that they have contributed to a reasonable percentage of the solution. For example, if the group has two members, then each group member must be able to show ownership of at least 40% of the final solution.</p> <p>Individual group members will be assigned marks and thus it is possible that members will obtain different marks.</p>	

4 Zombie Text Adventure

4.1 Overview

Polly is the CTO of a games company, TextGames, which is a small local company offering retro games, in particular their focus is on text based adventures, glossed up with a modern look and feel.

Polly has been asked to develop a text adventure for UWE, based on the notion that the campus has been invaded by Zombies. The game should be playable from any device, without the need to install non-standard software, and so the frontend, i.e. the user interface, it has been decided will be run from within a web browser. The game logic does not need to run on the player's machine and thus is not limited to web technologies.

Luckily the frontend is standard and with some HTML/CSS magic Polly already has a working system that meets UWE's requirements. Additionally, TextGames already has a standard format for representing game worlds and Java support libraries to load a world and connect to the web based frontend.

Thus, what is left is to provide the application logic of the game itself.

Hint: this implementation consists of two main parts:

- Building a representation of the game world that provides rooms, items, and so on. This world should be dynamic, i.e. the game should be able to move around rooms, pick up and drop items, and so on. The provided package world (see the Javadocs detailed in section 4.4.2.) loads the provided world into a class that simply returns strings for each of the different parts of the world. You will need to use this package to generate your world.
- ZombieBot is an interface that is used to drive the dynamic aspects of the game. It provides 6 methods and one constructor that are used by the provided ZombieServer to check if the game is over, to report a message displayed by the client at the start of the game, to enable zombie timers, get the current score, and to process commands entered by the user playing the game. The provided Javadocs provides details of this interface and the provide ZombieStarter project includes a very basic implementation to get you started.

4.2 Videos

The following video provides details of how to download the necessary material from Blackboard and get the starter project running in Neatbeans:

<https://youtu.be/ckGPRvt29LM>

A run through of playing the game can be found here:

<https://youtu.be/TaFUwNs-Ay4>

4.3 Information for the implementation

4.3.1 GUI front end

The GUI front end is provided as a web application. The frontend has been tested in Google Chrome and Firefox, including the lab machines for OOSD1, but is likely to work with others too, e.g. IE. You must use the provided GUI front end.

To use simply point your browser of choice to the URL:

<http://www.cems.uwe.ac.uk/~br-gaster/courses/2016-2017/OOSD1/>

Once loaded you will be presented with the connect screen shown in Figure 1.

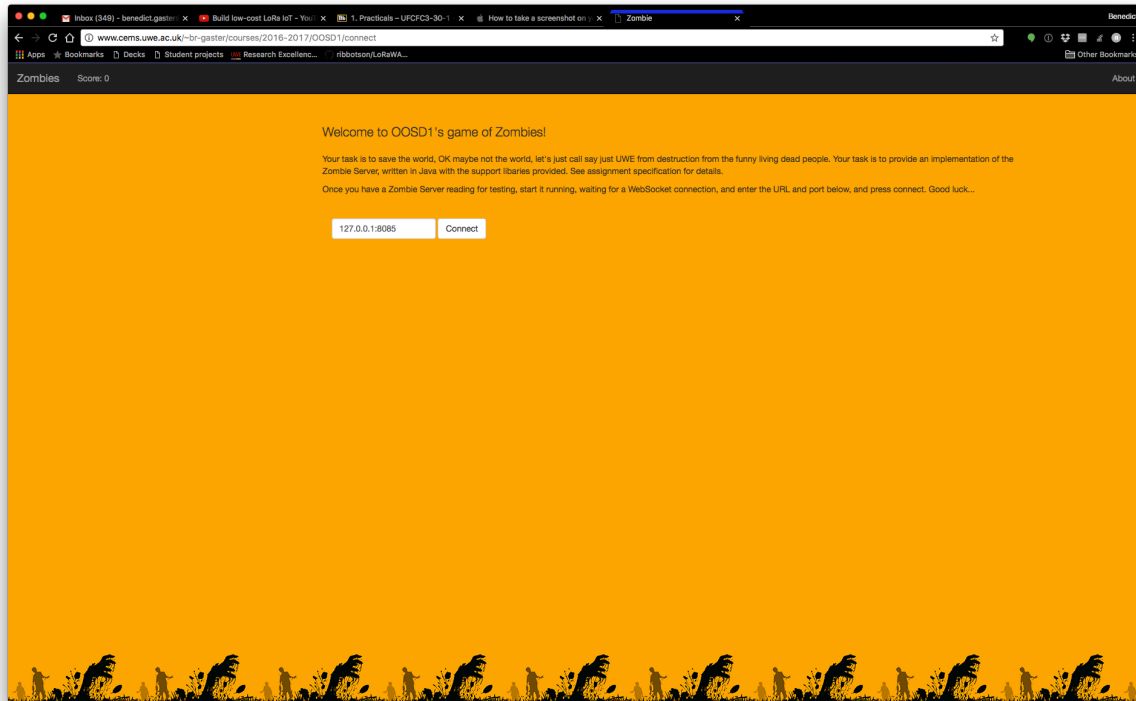


Figure 1: Webclient connect screen

At this point you need to run your Zombie server and once it is ready to receive connections, simply press connect. You can test this with provided ZombieStarter project.

There is no requirement to access the source for the frontend, however, if you are interested in this, fancy adding your own images, and so on, then the implementation can be accessed from UWE's Gitlab:

<https://gitlab.uwe.ac.uk/br-gaster/zombies-web-frontend.git>

If you are new to Git, then some starter videos can be found here:

<http://www.cems.uwe.ac.uk/~br-gaster/courses/GIT/>

Details on how to setup the frontend and build it are provided on the Git page for the project.

4.3.2 Using the provided Netbeans project and Jar files

To support the assignment a selection of Jar and Java files are provided. The Jar files are required to load the game world and provide a connection between your server and the web frontend. The Java files are provided to provide examples of how to connect to the frontend and also a template for the Zombie Bot interface that is provided to interact with the game.

The supplied resources can be found on Blackboard, supporting material for assignment, and are as follows:

-

- **world.zip** – Supporting library that loads the game world and provides a simple interface, returning strings, for each type of element in the world, e.g. Room, Items (key, gold, etc.), and so on. Hint: You will need to use this class to build a "real" world representation for playing the game. You might expect to have a set of classes similar in name to the ones in world.jar, but that support moving around the world, picking up and dropping items, and so on. Also provides assets need for the world, e.g. the world map.
- **jars.zip** – Contains JAR files necessary for the ZombieStarter project to build and work correctly. The following JAR files are included:
 - **util.jar** – Supporting library that creates a connection between your server and the web frontend. It expects the server to provide a Zombie bot class that plays the game. Hint: if you use the ZombieStarter project, then the access to classes from util.jar is already in place and you should concentrate on the package in world.jar.
 - External 3rd party libraries:
 - **ava_websocket.jar** – Simple wrapper for HTML Websockets
 - **jool-0.9.6.jar** – Java 8 functional stream library.
 - **json-20160810.jar** – Google JSON parser library.
 - **rxjava-1.2.2.jar** – Reactive stream extensions for Java 8.
- **Javadocs.zip** – A zip file containing Javadoc HTML for packages found in world.jar and util.jar.
- **ZombieStarter.zip** – A zip file containing a Netbeans project to get you started. It contains two Java classes, ZombieStarter and ZombieBot, the former is simply a main class that opens a websocket and tries to connect to the frontend. The other class provides a starting implementation for the Zombie bot; it provides default implementations for each of the methods you will need to provide to allow the client to play the game with the server. See Appendix A, for more information on this class.

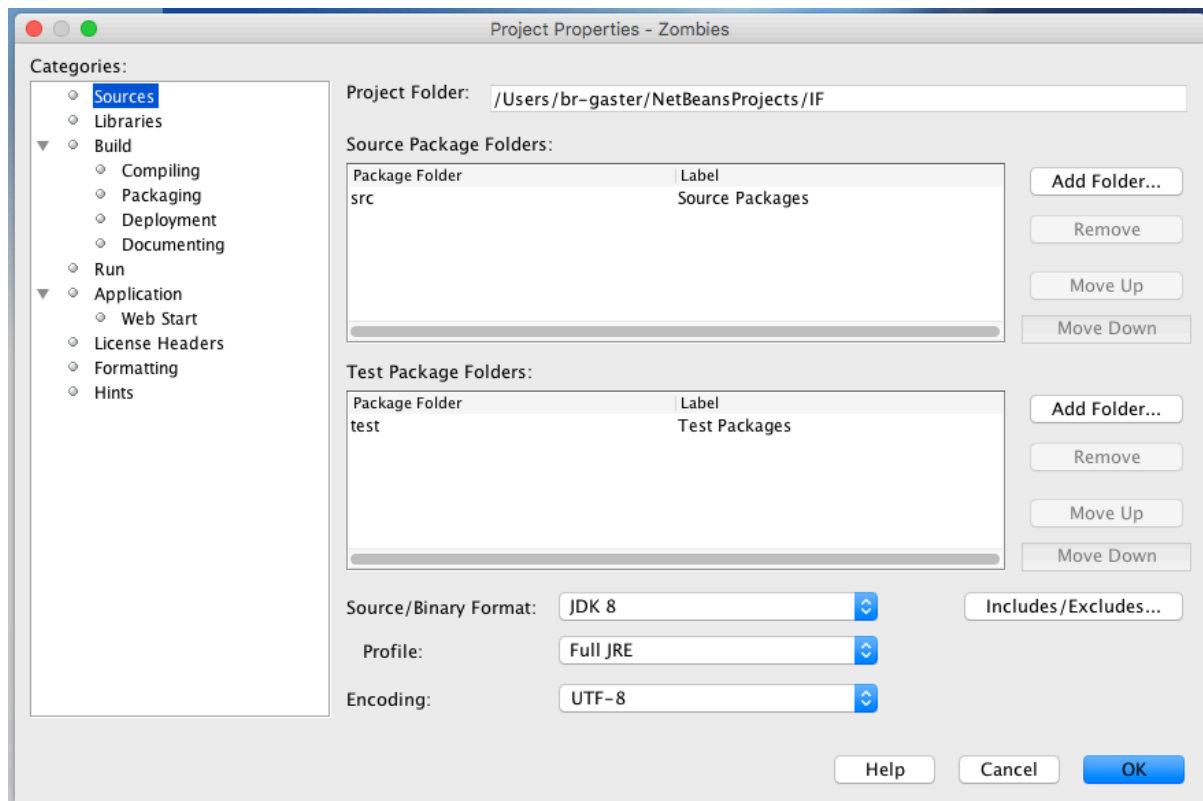
Download the zip files into a directory (folder) that you can work from. Unzip and you will see the JAR files in the directory, plus three other directories:

- **zombie-starter-master-953df7fe2b6e60ae252b9f964602e0187e93afa1** – the Netbeans project.
- **oosd1-assignment-javadocs** – Javadocs for world.jar and util.jar.
- **world** – source code for the supporting library that loads the game.

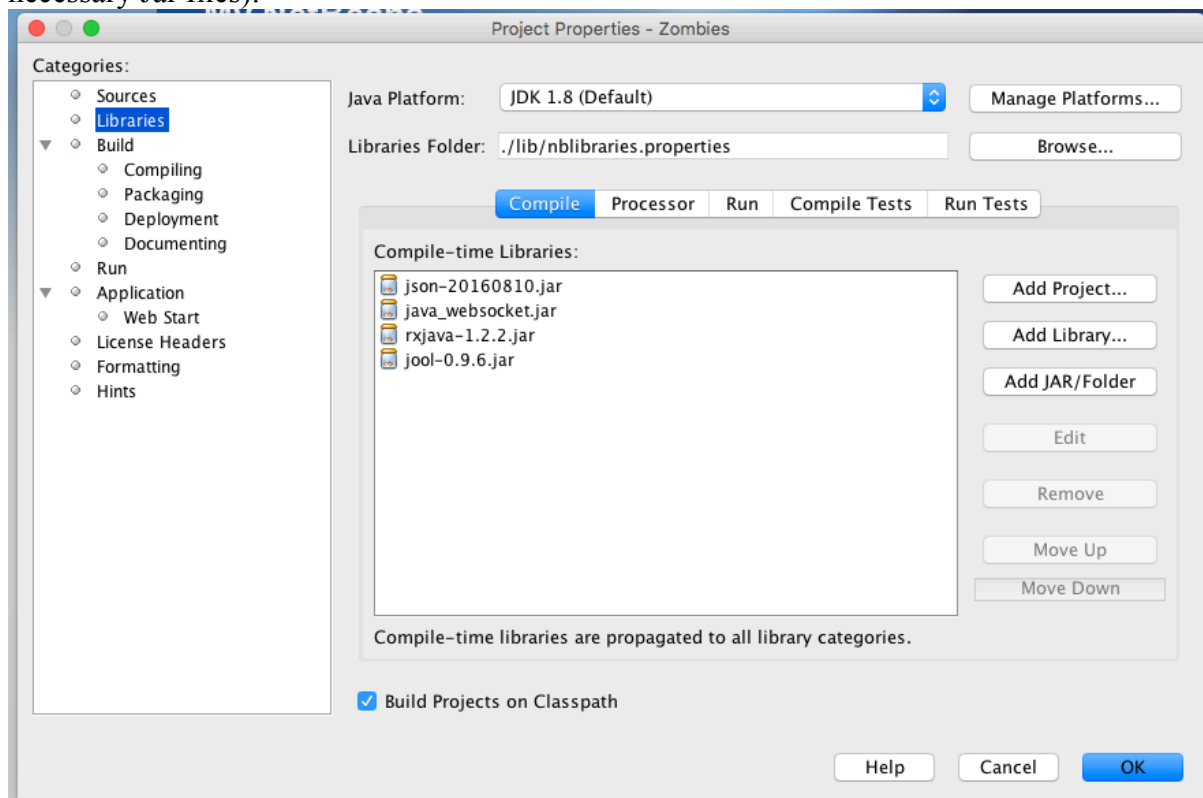
To open the Javadocs, simple double-click or open in a browser the file index.html in directory **oosd1-assignment-javadocs**.

To open the Netbeans project simply choose File->Open Project, in Netbeans, and navigate to the directory **zombie-starter-master-953df7fe2b6e60ae252b9f964602e0187e93afa1** and click open.

To use the supporting Jar files within your own project (starting from the ZombieStarter.zip), you need to load them into the Netbeans project. To do this right-click on the project and select **Properties**, you should see a window similar to:



Click the option **Libraries** and you should now see a screen similar to (but with out the necessary Jar files):



Now select **Add/JAR/Folder** and you should be able to browse to the downloaded Jar file and simply select it. It should appear as a dependency in the middle window. Repeat this for each Jar file.

Appendix A

In this appendix we provide an overview of the ZombieBot interface, which forms the heart of the game server. There are six methods that must be implemented to allow the game server to act as a go-between between the client and your implementation of the game world. These are documented in the Javadocs provided. Here we focus on just one of the methods **processCmd()**¹, which processes commands enter by the player, i.e. sent from the client, produces changes to the internal state of the game, and returns a list of HTML output messages that will be sent to the client for display.

The set of the commands that must be supported are:

- **info** - return the info string for the world.
- **look** - return the HTML representing the view of the current room. this should include entrances, items, and so on.
- **move dir** - try and move in the direction dir. if there is an entrance in that direction, then if it is not locked, move through to the connected room, return message about new room. if can't move, because no entrance in that direction, or entrance is locked, then return a message to that effect.
On entry to a room if it contains zombies, then the next time the method **enableTimer()** is called it should return true, just the one time. This will cause the client to start the Zombie kill timer event, displaying a Zombie and timer, to show the player that they will die if the zombie(s) is not killed in time.
- **pickup item** - pick up item from room, if item exists then this should cause it to be removed from room and get added to player's inventory, and return message to say picked up. if it does not exist, then return message to that effect.
The users score should be increased by one, each time an item is picked up.
- **kill** - if the room contains at least one zombie and players inventory contains either a Daisy or Chainsaw, then one should be killed (i.e. zombie count drops by one), return a message to inform the user that a zombie has died. if kill is successful, then the player's inventory should have one less Daisy or Chainsaw.
If no zombies, then nothing happens.
If the zombie timer was running, i.e. a call to **enableTimer()** had returned true for the current room, then if the zombie count is now zero, then **disableTimer()** should return true, one time, to tell the client to stop the zombie timer.
- **drop item** - drop an item into room from inventory. if the item is not in inventory, then nothing happens, otherwise it should be removed, when added to room. return a message to say item dropped.
- **timerexpired** - the client's zombie timer expired before all zombies in the current room were killed and so the player is dead. **shouldQuit()** should now return true.
- **quit** - **shouldQuit()** should now return true.
- **inventory** - return HTML to display the players current inventory.
- **anything else** - return HTML saying "That's not a verb I recognize."

Note some commands, such as **pickup** and **move**, take an argument. Additionally, commands are case insensitive, i.e. **move ne** and **MOVE NE**, or even **move NE**, are considered the same.

Items are specified as part of the world and they come with corresponding HTML that can be displayed. These are detailed in Appendix B.


Appendix B

¹ The documentation presented here is also present in the Javadocs for the world package.

The game world is described as a JSON file, which is loaded by the class **world.WorldLoader**. The world consists of the following components:

- **info** – HTML to be displayed when use enters the command "info".
- **startHTML** – HTML to be displayed when the player connects to the server
- **inventoryHtml** – HTML to be displayed along side the player inventory, when player enters command "inventory".
- **items** – Array of items, mapping item name to HTML to be displayed for that item. See below for item types.
- **rooms** – Array of rooms, with entrances, and initial items
- **start** – starting room
- **finish** – end room

Each of these are accessible via methods and iterators exposed via **world.WorldLoader**.

Both rooms and the player's inventory () can contain items. Commands exist for picking up and dropping items. The loaded world contains a list of items and corresponding HTML that provides icons (see images below) for displaying a given item in the client output window.

The following items may appear in the world:

- **Gold** - 
- **Key** - 
- **Chainsaw** 
- **Daisy** 

The corresponding HTML string that can be sent to the client for displaying an item's icon is exposed via the **items** list in the **world.WorldLoader**.

An example world, in JSON form, is provided in Figure 2.

```
{
  "info" : "<p class=\\\\"lead\\\\">Welcome to OOSD1 Zombies</p><p>Zombies are
real, no matter what you mum told you! UWE's campus has been invaded and you must
kill the zombies and get to the exit room, to save yourself and the University.
Good luck!</p>",
```

```

"startHtml" : "let's play",

"inventoryHtml" : "<img src=\\\\"../assets/backpack.png\\\\">",

"items" : [
  { "name" : "GOLD", "html" : "<img src=\\\\"../assets/gold.png\\\\">" },
  { "name" : "KEY", "html" : "<img src=\\\\"../assets/key.png\\\\">" },
  { "name" : "CHAINSAW", "html" : "<img src=\\\\"../assets/chainsaw.png\\\\">"
},
  { "name" : "DAISY", "html" : "<img src=\\\\"../assets/daisy.png\\\\">" }
],

"rooms" : [
  {
    "name" : "Toilet",
    "description": "Level Toilet desc",
    "entrances" : [
      { "direction" : "NE", "to" : "Exit", "locked": true },
      { "direction" : "SE", "to" : "R3", "locked": false },
      { "direction" : "SW", "to" : "R4", "locked": false }
    ],
    "items" : [ { "item" : "GOLD" }, { "item" : "CHAINSAW" }, { "item" :
"DAISY" } ],
    "zombies" : 0
  },
  {
    "name" : "Exit",
    "description": "This is the final room...",
    "entrances" : [ ],
    "items" : [ { "item" : "GOLD" } ],
    "zombies" : 1
  },
  {
    "name" : "R3",
    "description": "Level R3",
    "entrances" : [ { "direction" : "NW", "to" : "Toilet", "locked": false } ],
    "items" : [ { "item" : "KEY" }, { "item" : "CHAINSAW" } ],
    "zombies" : 2
  },
  {
    "name" : "R4",
    "description": "Level R4",
    "entrances" : [ { "direction" : "NE", "to" : "Toilet", "locked": false } ],
    "items" : [ { "item" : "GOLD" }, { "item" : "MACHETE" } ],
    "zombies" : 0
  }
],
"start" : "Toilet",
"finish": "Exit"
}

```

Figure 2: Example World in JSON