Introduction to Computer Science – Honors I

CS 181 – Fall 2013

Assignment 4 (**due 11/10**)

Complete the following subjects and submit your answers/code electronically through moodle. Place all your files in a directory named using your surname, underscore, and your initial, all in letter case. For example, if your name is John Smith, place all your files in smith_j/. Each programming assignment subject should be placed in its own directory immediately under this directory. For example, subject Hamming Code should be in smith_j/hammingcode/. Answers to non-programming assignments should be placed in file answers.pdf (or answers.txt or answers.docx) under the top-level directory (ex: smith_j/). Archive the top-level directory with zip and submit as file A2_surname_initial.zip (ex: A2_smith_j.zip). DO NOT submit any .class files.

Under the Stevens Honor System, you are required to sign the Honor pledge in all material that you hand in for grading. The Stevens Honor pledge is shown below:

"I pledge my honor that I have abided by the Stevens Honor System."

Please copy the pledge to the top of your answer sheet and type your name. Programming assignments should also include the above in a comment block at the top of your source file(s).

```
/*
 * QuickSort.java
 *
 * "I pledge my honor that I have abided by the Stevens Honor System."
 *
 * John Smith (c) 2013
 *
 */
```

Maximum points = 100

Text-based Adventure Games

"A text game or text-based game is a video game that uses text characters instead of bitmapped or vector graphics. Text-based games were common from 1970 to 1990, but are still used today."

"An adventure game is a video game in which the player assumes the role of protagonist in an interactive story driven by exploration and puzzle-solving instead of physical (e.g. reflexes) challenge. The genre's focus on story allows it to draw heavily from other narrative-based media such as literature and film, encompassing a wide variety of literary genres. Nearly all adventure games (text and graphic) are designed for a single player, since this emphasis on story and character makes multi-player design difficult."

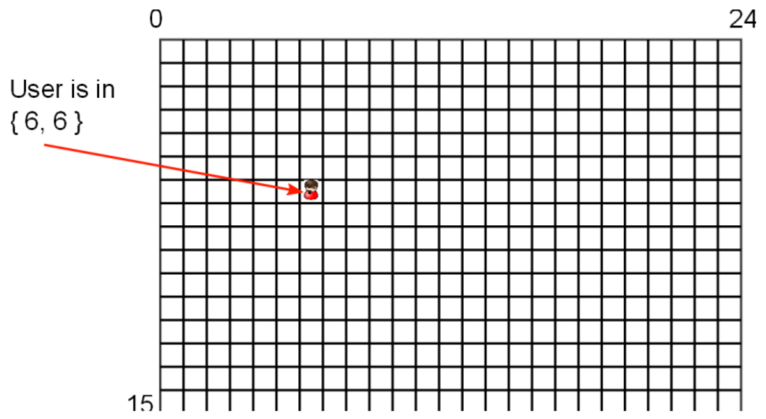
The player in text-based adventure games interacts with the game by issuing textual commands. See here for examples: http://textadventures.co.uk/

Text-based adventures games are quite simple. They (usually) have a single player (you) that moves through a virtual world, interacts with items and persons, solves puzzles, performs tasks, etc. as a storyline unfolds till the conclusion of the game.

Main concepts

The World

The world of the game can be represented as a grid, each square in the grid representing a location.



The Player

The player has an inventory of items and can move around in the world using directions such as north, east, south, west. The player could also have a state. For instance, a level of stamina.

Locations

A location can have many different "versions", but they should all support certain functionality. For example, in all locations you can look around, perform actions with the objects in the location, like using them or picking them up, and move from one location to another if an exit or path to another location exists. Going through an exit will move the player to a different location.

Items

Each location can include various items within. A player can attempt to perform certain actions with these objects. For instance, *use object_name* would "use" the object, *look at object_name* would return a description of the object, and get object_name would remove the object from the location and place it into the player's inventory.

The status of items can change. Items in different state may behave differently. For example, looking at an open drawer would also reveal its contents, which in turn can be acted upon. Finally, using one object on another could generate a new object or destroy the existing one. For example, *use water on tree*, would use up the water, and the tree could appear to produce an *apple*.

This Assignment

The goal of this assignment is to build a text-based game engine. Start with designing your class hierarchy and their methods. Start early! Don't hesitate to ask questions on moodle. Code that does not compile will not be graded! Good luck!

A basic engine and world (50 points)

Build a basic game engine that reads a world's configuration from files.

User commands

A basic game engine should support the following user commands.

Moving

north, west, east, south should result in moving the player to a new location if successful, or print a message that this is not possible. E.g., you cannot walk through walls.

Interacting with a location

look should return some information on the location, possible exit directions (e.g., "possible exits are north and east"), and should list visible items (e.g., you see: a mirror at the north wall, a comb on the dresser).

Interacting with items

look at object_name returns a description of the object

get object_name receives the item for the player and stores it in his inventory. Not all items can be put in a player's inventory

drop object_name leaves an item from the player's inventory in the current location

use object_name uses the item in some way particular to the item. Using the item should have an effect on the location, another item, or the player (e.g., change the description of a location or the possible uses or usability of another item, move a player, etc.).

Player actions

inventory list contents of player's inventory

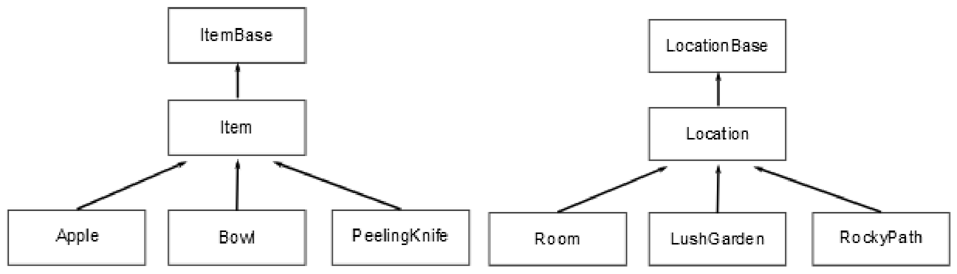
Game commands

help prints a help message

exit exits the game

Basic Classes

You can design your own classes. A sample design of the basic classes is shown below.



World

The world you create should have 4 distinct locations. These location should be connected with each other, meaning that a player should be able to move between them. NOTE: that could be only conditionally after using an item at a particular location. They do not need to be arranged to form a square. Use a larger grid to accommodate placing the locations in other configurations. At least one of these locations should change in some way based on an item being held or used by the player.

The world should contain at least 3 items. 1 item should be unmovable, that is the player should not be able to get it and store it in his inventory. 1 item should be unusable, that is the player should be able to get it, but there is no way he can use it. Finally, 1 item should be usable, that is the player should be able to get and perform an action with it that affects another item or a location. Its effect on a location or item should be one of the following: an exit appears or disappears changing the description of the location, or a new item appears expending the used item.

Configuration

The world's configuration should be read through configuration files. To keep this simple, each file configures a separate object, and every line in the configuration files is name-value pair referring to a property of the object. A name-value pair definition cannot span multiple lines. If you need to include the newline character in a value, use a special character that you can replace with a newline after reading it.

One file should include the size of the world. The file world.txt should follow this format:

world x_size, y_size

Example: world 2 2

For every grid location that is occupied in the map, you should create another file with the following name: location_xindex_yindex.txt, under a directory called *locations*. This make the relative pathname: locations/location_xindex_yindex.txt. Example location_0_0.txt corresponds to the most North West item of the world grid. Each of these files should configure a particular location. For example, location_0_0.txt can include the following:

title a nice room

item apple_1

item bowl_1

exits E S

desc This is an nice room that you never want to leave.

attr-name1 value1

…

attr-nameN valueN

This means that a location at {0, 0} has the title "a nice room", it includes 2 items with identifiers "apple_1" and "bowl_1", and two exits "S" and "E", that is south and east. The description of the room is "This is an nice room that you never want to leave.".

The remaining lines enable you to define custom attributes that your location supports. For example, if you have a wall color attribute then you can use *attr-color red* or *attr-color green*. All of your attributes should start with "attr-", followed by the name of the attribute. This way the semantics followed in the configuration files remain consistent, and you can easily differentiate your custom attributes from the rest.

For each item you should create another file under a directory called items. For instance, items/apple_1.txt and items/apple_2.txt. An item file can include the following:

class apple

canget yes

attr-name1 value1

…

attr-nameN valueN

This specifies that this item is of the class apple. Your program should create the appropriate object based on this. If you know of Java's reflection mechanism, please do not attempt to use that. Instead use switch or if..else statements to determine the subclass to instantiate. *canget* specifies if the player can get this item and place it in his inventory, and can be *yes* or *no*. Items can also have custom attributes.

Document your classes and methods using Javadoc (20 points)

Document your classes and methods using Javadoc. Your textbook lists a set of the basic Javadoc tags you can use for documenting methods, method parameters and return values, and classes. For more detailed information see http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

If you are using an IDE, it will probably generate Javadoc-based documentation for you automatically if it identifies that you have been using Javadoc-style comments. If you want to run Javadoc from the command line of a terminal, you need to first place all your files in a package. To do this place all the source files in directory with the same name as the package that you want to define, e.g., mygame. All of your source files should then begin with "package mygame;" In the parent directory of mygame, create a new directory called "doc" and run the following command: *javadoc -sourcepath . -d doc mygame.*

This should generate the documentation for the mygame package in the directory doc. Generally, the javadoc command takes the following parameters, and you can use it to generate documentation for multiple packages in one go.

javadoc -sourcepath code-path -d destination package-names

Extending your World (20 points)

Exchange classes and configuration files for new locations and items with your classmates. Integrate them into your world to create a bigger one, by establishing a Java Interface that will allow you remain compatible with the exchange files. Correctly designing a class hierarchy will help you easily integrate "foreign" code. Exchange code with (at least) 2 classmates. Get a new location type from one and a new item type from the other, and integrate them to your world. The location you obtain should be the one that can change based on using or possessing an item. The item you receive should be usable. Make changes to your code, if necessary, to accommodate the new location and item. If you prefer to work alone, you can create a new location and item yourself with the same properties mentioned above.

Saving and loading game state (10 points)

Add support for a saving and loading the state of the game by implementing the serialization interface in your classes. Add the game commands *save filename*, and *load filename* to save and load your game into and from a file. Do not forget to document these.

Supporting additional players actions (Bonus 10 points)

Support new player actions. You can be creative here. You can plant seeds, fight dragons, drive a car, etc. It’s all up to you. Do not forget to document your extensions.