Project 5 - Adventure Game

Due Nov 18 by 12pm **Points** 80

Resources

- **cs5044-f2019-adventure-lib.jar (https://drive.google.com/file/d/166AdedVRRH3xPtZ0d3lpYBCxgdb73FmE/view?usp=sharing)**
- **cs5044-f2019-adventure-doc.zip (https://drive.google.com/file/d/1LXtdWp4jTDRVw-BYecc5f-jz3PPng_xJ/view?usp=sharing)**
- **Message.java (https://drive.google.com/file/d/1M2G1mNgzJTmrEPurs2TLdo-wKdDo5YY7/view?usp=sharing)**
- **MyGame.java (https://drive.google.com/file/d/1zwwQY9HLglpd6QxYefZDNxookjv7OHdH/view?usp=sharing)**
- **MyGameTest.java (https://drive.google.com/file/d/1djW3ifPJH5JMlk2i9p88nyUjoddl_wPv/view?usp=sharing)**

Recent Updates to Project 5

1. We have removed *close* and *hide* from the required commands to implement since they are not absolutely necessary for the scenario. If you have already implemented them, then they count toward the two extra command you need to implement.
2. We are adding alternative options for listing room contents so that all three of the examples below are valid ways of presenting room contents. Regardless of which you chose, the player and the player's contents will not be listed. Also, examining and searching containers (as well as successfully opening them) will still require that you list their contents.
   - You see a screwdriver and a shoebox (containing a light-bulb) here. [original]
   - You see a screwdriver and a shoebox here. [no items in containers are listed]
   - You see a screwdriver, a shoebox and a light-bulb here. [items in containers are included in the main list]
3. Dr. K has updated a revised version of the Message.java class for you to use. The layout should be more understandable than the old file. There are some name changes, so if you have already made progress using the old file, you may have to make a few edits, but it should be obvious what to do. If not, ask on Piazza.

We'll talk more about project 5 at this week's Q&A.

Goal

In this assignment, you will be implementing a text adventure game. You will be working with a number of classes, many of which will be subclasses of existing classes provided for you in the CS5044AdventureLibrary.jar library.

Adventure games have been around for a long time, dating back to the *Colossal Cave Adventure*, (see **Colossal Cave Adventure page** (http://www.rickadams.org/adventure/), or **DG Jerz's Colossal Cave Adventure page** (http://jerz.setonhill.edu/if/canon/Adventure.htm)), but there is more to the history of adventure gaming than this initial spark. In particular, **Infocom** (https://en.wikipedia.org/wiki/Infocom) was famous as a computer game producer who made text (and later, graphic) adventure games. Zork, their oldest adventure game, is considered by many to be the most famous of all text adventures, or "interactive fiction games". You can even **download and play** (http://www.infocom-if.org/downloads/downloads.html) Zork I, II, and III, or read its **entry in Wikipedia** (http://en.wikipedia.org/wiki/Zork).

This assignment offers you a large amount of creative freedom, so do not hesitate to have fun with it!

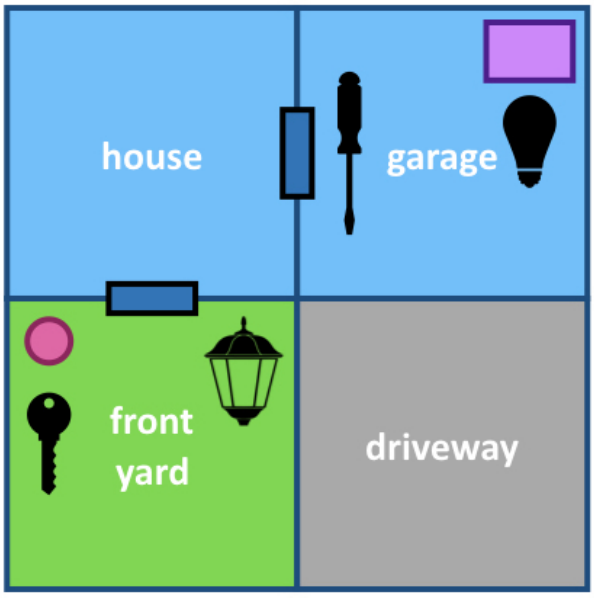
Learning Objectives

- Exposure to the Composite design pattern
- Exposure to object-oriented dispatch
- Familiarity with creating subclasses
- Familiarity with overriding methods
- Familiarity with polymorphism
- Familiarity with using library classes

Locked Out of Your House

The game starts off with you outside in your yard, and your first goal is to get into your house. The front door to your house is locked and you don't have the key. You can go east to your driveway and then north to your garage, but the side door to your house (in the garage) is also locked.

Game Objects	Game Actions
front-door	inventory
side-door	examine
shoebox	search
flower-pot	take
wall-lantern	go
house-key	open
screwdriver	close
light-bulb	unlock
	replace
	hide



The rooms, objects, and actions (commands) you must create are shown in the figure above. Note that the hyphens in the object names are mandatory, as our simple parser only handles two word commands. The pink circle in the front yard is the flower-pot. It contains the key, which is not visible until you "search flower-pot" (simply examining it won't suffice). However, the wall-lantern is broken, so it is dark outside -- and you

cannot search for things in the dark. Therefore you have to replace the light-bulb in the wall-lantern. To do this, you have to first open the wall-lantern with a screwdriver and then you can "replace light-bulb". The screwdriver is in the garage, and so it the light-bulb. However, the light-bulb is inside the shoebox, which is closed. Therefore, you don't see it until you "open shoebox". Once you have the key, you can unlock the door and solve the puzzle. Once you do that, you can explore the other rooms in your game and interact with other objects. You can even create other people to talk to. You're only limited by your imagination!

Support Classes Provided for You (DO NOT WRITE THESE!)

Just as in the previous assignments, you have a number of support classes available that will help in constructing an adventure game. These classes are all located in the adventure package, which is provided in the CS5044AdventureLibrary.jar file that is linked to at the top of this page (be sure to add it to your project).

The are two main classes for creating adventure games in the package adventure:

- Game represents the main class of a game. It sets the game up, and then enters a loop to read and execute player commands. Note that Game is an *abstract class*: that means you cannot create an object of this class directly, because some of its methods have not been given definitions. Instead, you must create your own subclass (that extends Game) that defines the missing pieces.
- Command is an abstract class that defines the common properties of all command objects. In particular, it requires every command object to implement an execute method. There are three classes that extend this class that are also provided for you in the adventure package: GoCommand, HelpCommand, and QuitCommand. You can use these three commands, and create any new ones you like.
- Room represents a location in the game. The Room class provides minimal features. However, you can create/extend your own custom subclass of Room that adds any new features you wish.
- Player represents you, the player. Every game must have a player that can move from room to room and interact with the game world.

Study their javadoc documentation of these classes well so that you understand what their methods do.

The adventure package also contains two other classes to help you implement a game:

- CommandWords represents a dictionary of known commands in a game. This class uses a Map to associate words with Command objects. This makes it easy to add new commands without affecting the structure of this class.
- Parser reads command lines and breaks them up into words, looking the command word up using a CommandWords object. The parser implemented here understands one- and two-word input lines, where a one-word command is a verb (like "quit"), and a two-word command is a verb followed by an object (like "go east" or "take wand").

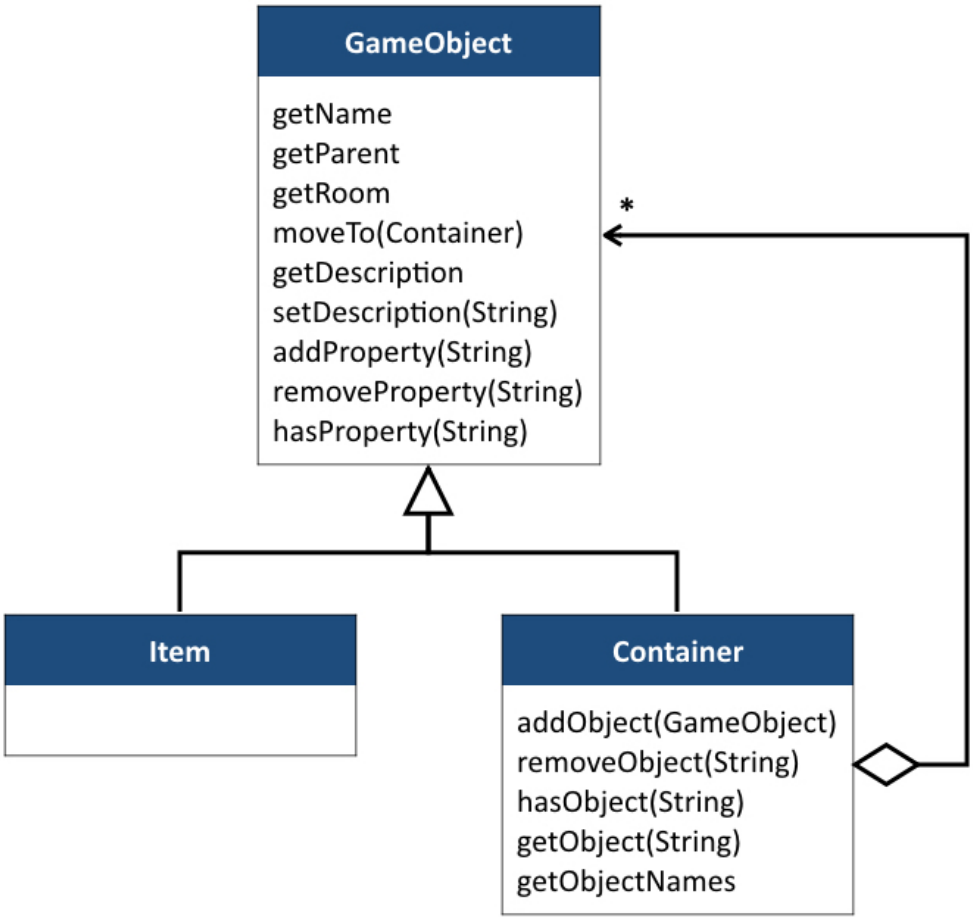
In addition to these classes, AdventureGUI is a graphical user interface that contains a large rectangular area for displaying text. There is a smaller rectangle below it where you can enter commands, and a "Move"

button that you can press to have the command execute (you should also just be able to hit the [return] key).

The classes GameObject, Container, and Item are used to implement a composite design pattern and will be discussed in the next section.

Composite Design Pattern

The adventure package uses the **Composite Design pattern** for game objects. The composite design pattern is used when you want to arrange your objects in a tree-like structure. The game objects in our adventure game are arranged in a tree-like structure because we have some game objects (like Room and Player objects) that can contain other game objects. We may also have boxes and vases and garbage cans in our game world, which can also hold other game objects. But we can also have things like keys and lamps and screwdrivers that cannot hold other game objects. The composite design pattern handles this well.



The Container class consolidates code that is common to composite game objects, and the GameObject class holds code that is common to all game objects. Strictly speaking, we don't need the Item class, but it helps use distinguish between composite objects and leaf objects.

Requirements for Your Game Implementation

You will need to extend several classes to implement a base level of required features in your game. Your game implementation must do the following:

- Include your own custom subclass of Game. It must be called MyGame.
- Include your own custom subclass of Room. The four mandatory rooms are "your house", "your garage", "the front yard", and "the driveway", as shown in the figure above.
- Make sure you implement the requirements described in the section "Locked Out of Your House" above.
- Include a variety of locations/rooms (a minimum of 8: the 4 starting rooms + 4 others). Include a minimum of 3 extra objects (besides the 8 above) and make one of those a container, also implement a minimum of 2 extra commands (besides those given above).
- Create a Door class that extends Item and automatically adds properties common to all doors when a new door is constructed.
- It must be possible for a player to reach each of the locations/rooms.
- The player can win. There has to be some situation that is recognized as the end of the game where the player is informed that he/she has won.
- In addition to the *go* command (which is already implemented for you in the GoCommand class), you must also support the following one-word movement commands: *north*, *south, east, west, up,* and *down,* together with one-letter abbreviations for each: n, s, e, w, u, d. Note: you can support all of these with a single command class that extends GoCommand, so you will lose points if you use six (or twelve!).
- In addition to the commands provided in the adventure package and the movement commands described above, your game should support the following commands:
    - *examine* (also x) – Allows you to see the description of an item. This command should work as long as the item is in scope. An item is in scope if it is the same room as you (including in your inventory or some other container), and it is not hidden.
    - *search* – Allows you to search a game object (typically a container). For example, you must search the flower-pot before the house-key is revealed.
    - *take* – Allows you to take an item from the room and place it in your inventory.
    - *inventory* (also i) – Allows you to view the list of items in your inventory. If there are no items in your inventory, you should get the empty inventory message. If there is at least one item in your inventory, you should get a comma-separated list of items (use the appropriate methods from the Message.java and Formatter.java).
    - *open* – Allows you to open a closed container or other openable object (like a door).
    - *close* – [you do not need to implement this] Allows you to close an open container or other closed object (like a door).
    - *unlock* – Allows you to unlock a locked item (like a door). If you want to implement a "lock" method, feel free to do so, but it is not required.

- *replace* – A command that is solely used with the light-bulb. "replace light-bulb" will work when you are in the yard when the wall-lantern is open and has not been fixed yet.
- *hide* – [you do not need to implement this] A command that is solely used with the key. After you find the key by searching the flower-pot, you can "hide key" again when you are in the front yard.

To make it easier to test your game, you must make use of Message.java file. Read the descriptions of the methods and make sure that each message is used in the appropriate place in your game. This file is NOT included in the library, so you will have to create a file named "Message.java" in your project and cut-and-paste the code below into your class. Make sure you do not modify the names of the methods in the file. You may be able to make *small* changes to the text strings without causing problems with the test cases (but use caution if you attempt this!).

Beyond these requirements, you are free to explore any other game features you wish to provide. Feel free to take advantage of this flexibility to have fun while you complete the assignment.

Modify MyGame

To see how to start creating your own game, look at MyGame. It should already have the rooms and some of the objects required in the example scenario and allow you to move between them.

Run it (with the library) and see what happens. Remember you have 3 commands "go", "help", and "quit".

Use Properties to Help You Implements Your Commands

Make generous use of properties to help you implement the execute methods in your actions. Properties are just strings, so feel free to come up with your own properties, but here are some properties that other adventure game writers have used in the Inform game design system:

- concealed – the object is here but can't be seen
- door – the object is a door
- lockable – the object can be locked and unlocked
- locked – the object is locked
- moved – the object has been moved from its original location
- open – the object is open
- openable – the object can be opened and closed
- static – the object is fixed in place (cannot be taken)
- visited – the player character has visited this location

Feel free to make your own. For example, you might have one called "solved" for an object whose puzzle has been solved.

Modify the Default Long Description of Room

This is probably one of the last tasks that you want to perform. The default long description of Room prints something like this:

```
Your Garage
You are in your garage.
Exits: south west
Items: Player screwdriver shoebox side-door
```

It gives you good information (almost), but it does not look very nice. For example, it even lists the "Player" as an item in the room. Also, even if the shoebox is open, it does not tell us what is inside of it (it should). Change the description so that it prints something like this:

```
Your Garage
You are in your garage.
You can go south to the driveway or west to your house.
You see a screwdriver and a shoebox (containing a light-bulb) here.
```

You will need to override the Room class to do this and make use of the supplied Formatter class. You will also likely need to use properties to determine which items are concealed so that they are not included in the list (like the player and the side-door).

Testing Your Game

As with other work in this course, you are responsible for writing tests for all the code you write. In this case, because we have separated the GUI interface from the command processing, we are going to ignore the GUI completely and just test that the commands give you the results in your game that you need. The JUnit test starter file MyGameTest.java sets up your tests so that you can pass multiple commands to a private method (*executeMoves*), and then check if the resulting room description and message are what you expect them to be. Use the long room description when you want to check whether a room has something in it.

Finally, note that you are expected to (or required to) test the *main* method of your game subclass. However, if you've incrementally tested all your other methods individually, you really only need one very simple test for the main method – its so short that it shouldn't need anything more than that. You also must write appropriate test cases for all other public methods in all classes that **you** write, of course.