CIS 162 Project 5 (Part B)
Adventure Game

Full Project Due Date

- at the beginning of class, Thursday, July 30, 2015 (last class for the semester)

Before Starting the Project

- This is a BIG project! It is important that you read this entire project description and to get started right away.

Learning Objectives

After completing this project you should be able to:

- *use* ArrayLists to maintain and process collections of objects
- *develop* an application with multiple classes

Step 1: Create a New BlueJ Project

Step 2: Create a class called *Item* (5 pts)

Implement a class to maintain information about an item including: a one-word name (String), a longer description of the item (String), a weight (int) and whether it is edible (Boolean).

- provide appropriate names and data types for each of the instance variables.
- `public Item (String n, String d, int w, boolean e)` – initialize instance variables.
- provide get and set methods for each instance variable.
- `public boolean isEdible ()` – return `true` if the item is edible.
- Test this class thoroughly using a main method before moving on.

Step 3: Create a class called *Room* (15 pts)

Implement a class to maintain information about a room including: a description of the room (String), an Item, and a list of all adjacent Rooms (HashMap). Rooms can have an unlimited number of neighbors in unique directions. For example, a room could have neighbors to the "north", "upstairs", "outside", "southwest" or any other direction. Allowing an unknown number of neighbors requires a dynamic data structure. You will use a `HashMap` that is described later in this document.

- provide appropriate names and data types for each of the instance variables.
- `public Room (String d, Item i)` - a constructor that is passed the description and an optional item.
- provide get methods for the description, and item
- `public void addItem (Item i)` – add an item to the room. If an item is already in the room it gets replaced.
- `public boolean hasItem ()` – return `true` if the room has an item.

- `public void addNeighbor (String dir, Room r)` – add the provided room and corresponding direction to the `HashMap` of neighbors (see below).
- `public Room getNeighbor (String dir)` – return the adjacent room in the requested direction. Return `null` if there is no neighbor in that direction.
- `public Item removeItem ()` – remove and return the item. To remove the item, set the instance variable to `null`. Warning: this can be a bit tricky!
- `public String getLongDescription ()` – return a `String` that begins with "You are" followed by the room description. If there is an item in the room, include "You see" followed by the item description.
- Test this class thoroughly using a main method before moving on.

Step 4: Create a class called *Game* (40 pts)

This class is the most complex you have written so far! It is responsible for keeping track of the items being held by the player and the current location. The game maintains a current message that GUI is responsible for displaying. There are no `println` statements anywhere in this class except maybe the `main` method used for testing.

- Define an instance variable that holds an `ArrayList` of `Items` for what the player picks up along the way. Define instance variables for each `Room` and `Item` specified in the game design. You will need an additional `Room` instance variable for the player's current location and a `String` to maintain the current message.
- `public Game ()` – instantiate the `ArrayList` of Items. Create all of the rooms by invoking the helper method defined next. Set the current location to the starting location of the game and the message to the introduction message.
- `private void createRooms ()` – this helper method instantiates the items and rooms. Identify all of the room neighbors. See the section below for more information about creating rooms.
- `private void setIntroMessage ()` – this helper method initializes the game's message with an introduction to the game. DO NOT print the message.
- `public String getMessage ()` – return the game's message for another object to display (e.g. the GUI). DO NOT print the message. This is one line of code.
- `public void help ()` – update the game's message with hints, suggestions and reminders about the game objective. DO NOT print the message.
- `public void show ()` – update the game's message with the current room's long description. DO NOT print the message.
- `public void move (String direction)` – update the current location with the neighbor in the requested location. If not possible, the message should explain the player can not move in that direction. See the sample code below for moving from one room to another.
- `public void inventory ()` – update the game's message with a list of all items the player is holding. If holding nothing, the message should explain.
- `public void eat (String item)` – update the game's message with one of the following options: 1) "you are not holding an *item*" , 2) "*item* is not edible", or 3) "Yum,

that was a tasty *item*!"

- `public boolean gameOver ()` – determine if the game has been won or lost. If either, update the game's message with the news and return `true`. Otherwise, return `false` with no change to the message. This method is invoked in the GUI class to determine when the game is over.
- `public void pickup ()` – if appropriate, remove the item from the current room and add it to the player's inventory. Update the game message with one of the following options: 1) there is no item in the room to take, 2) the item is too heavy to take, or 3) you are now holding the item.
- `private Item searchInventory (String name)` – this helper method checks if the player is holding the requested item *name*. If found, return the `Item`. If not, return `null`.
- `public void leave (String item)` – if appropriate, remove the item from the inventory and add it to the current room. Update the game's message with one of the following options: 1) you are not holding that item, 2) the room already has an item and can not be replaced, or 3) you have successfully dropped the item in the room.
- `public void backup ()` – if possible, return to the previous room. Update the game message with one of the following options: 1) long description of the new current room, or 2) an explanation that the player cannot retreat from here. The player can only retreat one step before moving forward again. This is accomplished with an additional instance variable of type `Room` that maintains the previous location or `null` as appropriate. There is no retreat from the starting location.
- `public static void main (String args[])` – provide a main method that instantiates a `Game` object and tests all methods. Include a series of method calls that allows the player to win the game. See the next section of Software Testing for more information.

Step 5: Software Testing (5 pts)

Software developers must plan from the beginning that their solution is correct. BlueJ allows you to instantiate objects and invoke individual methods. You can carefully check each method and compare actual results with expected results. However, this gets tedious. Another approach is to write a `main` method that calls all the other methods. See Listings 4.1 and 4.3.

Main Method

Write a main method in the `Game` class to automatically play a game until the player wins and also demonstrates all game methods work. Here is a minimal example that does not test all methods as required.

```
public static void main (String args[]){
   Game g = new Game();
   System.out.println(g.getMessage());
   g.pickup();
   System.out.println(g.getMessage());
   g.move("south");
   System.out.println(g.getMessage());
   g.leave("book");
   System.out.println(g.getMessage());
   g.move("north");
   System.out.println(g.getMessage());
   g.move("south");
   System.out.println(g.getMessage());
   g.pickup();
   System.out.println(g.getMessage());
   if(g.gameOver()){
      System.out.println(g.getMessage());
   }
```

Sample Output

Your game will look different but the following example provides a flavor of what the messages should look like. Provide a blank line between each message for readability.

```
Welcome to GVSU!  Home of the Lakers. “In Search of Louie” is a new and
incredibly boring adventure game.

You are outside the main entrance of the university
You see a red Nike shoe

You are holding a red Nike shoe

You are in a computing lab

There is nothing to take.

You are in the computing admin office
You see a dusty old book

You are holding a dusty old book

You are holding:
   a red Nike shoe
   a dusty old book

You are in the magic treasure room
You see a very large treasure chest

The treasure is too heavy to pick up!
```

Step 6: Create a GUI class (15 pts)

Now that you have the `Game` working it is time to create a more interesting graphical user interface for someone to use.

- Use the GUI class from project 4 to get started.
- Define an instance variable of type `Game` and instantiate it in the constructor
- Define `JButtons` for each valid direction in your game (i.e. North, Upstairs, Inside)
- Define `JButtons` for each action
- Define a `JTextArea` to display the messages called **results**

GUI Constructor

- Instantiate each `JButton` in the constructor
- Create a `JPanel` to hold all of the action buttons and place it in the SOUTH region of the `JFrame`.
- Create a `JPanel` to hold all of the direction buttons and place it in the EAST region of the `JFrame`. Use a BoxLayout for the panel to allow the buttons to stack vertically (see Section 7.11).

```
JPanel directionPanel = new JPanel();
directionPanel.setLayout(new BoxLayout(directionPanel,
                  BoxLayout.Y_AXIS));
directionPanel.add(new JLabel("Directions"));
directionPanel.add(eastButton);
```

- Register the buttons with the `ActionListener`
- Instantiate the `JTextArea`. The following statements allow the r**esults** text area to scroll and display the text more attractively. Place the **results** text area in the CENTER region of the `JFrame`.

```
results = new JTextArea(30,60);
JScrollPane scrollPane = new JScrollPane(results);

// allow word wrap
results.setLineWrap(true);
results.setWrapStyleWord(true);

// allows auto scrolling within the JTextArea
DefaultCaret caret = (DefaultCaret) results.getCaret();
caret.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);
```

actionPerformed

- `public void actionPerformed(ActionEvent e)` – add if statements for each of the button clicks and invoke the appropriate game method. For example:

```
     if (buttonPressed == help){
          myGame.help();
     }
```

- For the eat action, prompt the player using a `JOptionPane` and then use the returned `String`. For example:

```
if (buttonPressed == eat){
      String message = "What do you want to eat?";
      String toEat = JOptionPane.showInputDialog(null, message);
      myGame.eat(toEat);
 }
```

- At the end of the method, display each game message in the **results** `JTextArea`.
  `results.append(myGame.getMessage());`
- Add an if statement for the **quitItem** `JMenuItem`. Java applications are terminated with a call to `System.exit(1)`.

Additional Methods and Requirements

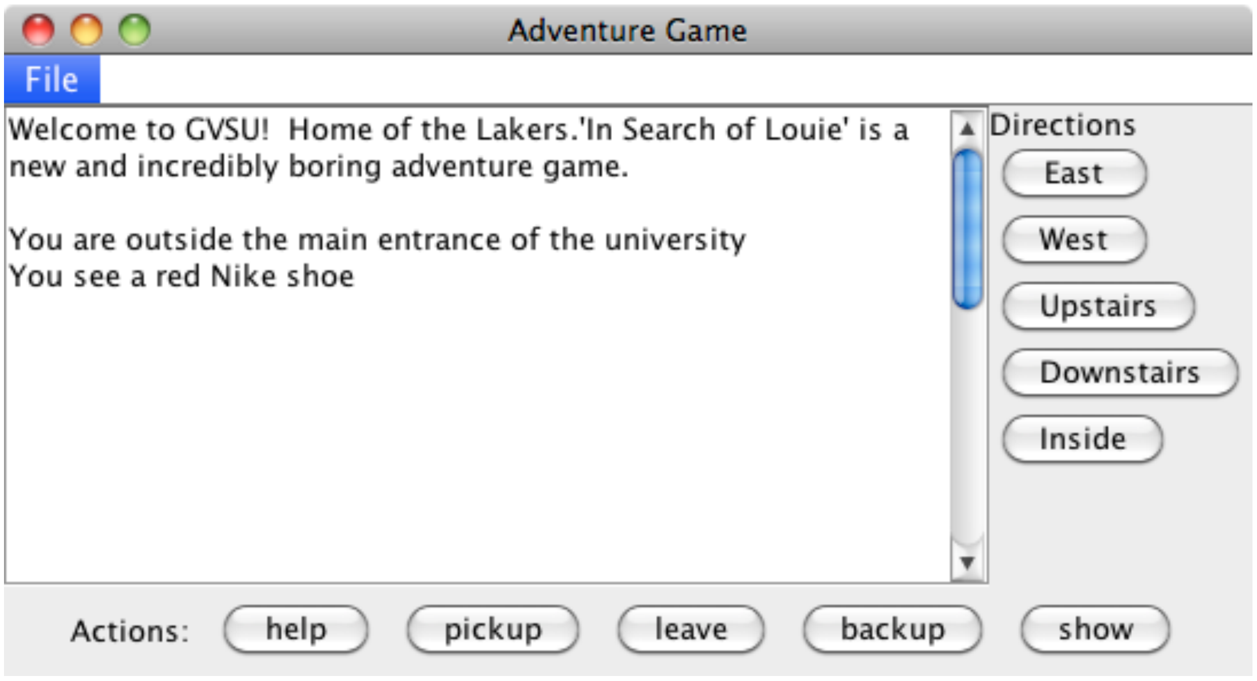
- `private void gameOver()` – disable all buttons. This helper method is called when the GUI determines the game is over by asking the Game object.
- `private void newGame()` – instantiate a new game, reset the **results** `JTextArea` with the initial game message, enable all buttons. Invoked this method when the player selects the menu item.

```
myGame = new Game();
results.setText(myGame.getMessage());
```

- Create File menu items for "New Game" and "Quit". Reuse the code from Project 4.
- Enhance the appearance of the GUI by providing color and borders to the Action and Direction panels (see Section 7.12).

Sample GUI

Your GUI will have different directions and additional actions.



Welcome to GVSU! Home of the Lakers.'In Search of Louie' is a new and incredibly boring adventure game.

You are outside the main entrance of the university
You see a red Nike shoe

Using HashMaps

A `Hashmap` is a special type of `ArrayList` that uses words, called a key, as the index instead of integers. The book does not cover HashMaps but the following code should be sufficient for a basic understanding. You must adapt these examples in the `Room` methods.

```
// define a HashMap with pairs of words and Rooms
HashMap <String, Room> myNeighbors;

// instantiate the HashMap in the Room constuctor
myNeighbors = new HashMap <String, Room> ();

// add Kitchen to the "north"
myNeighbors.put("north", Kitchen);

// Get the room (if any) found to the "east"
Room next = myNeighbors.get("east");
```

Creating Rooms

Creating the rooms, items and relationships is accomplished by instantiating objects and using their methods. For example:

```
shoe = new Item("shoe", "a red Nike shoe", 10);
outside = new Room("outside the main entrance of GVSU", shoe, null);
theater = new Room("in a lecture theater");
outside.addNeighbor("east", theater);
outside.addNeighbor("south", lab);
outside.addNeighbor("west", pub);
currentLocation = outside;
```

Moving from Room to Room

Moving from one location to another involves updating the current location if there is a neighbor in the requested direction. Here is an example for the `Game` method.

```
public void move(String direction){
      Room nextRoom = currentLocation.getNeighbor(direction);
      if (nextRoom == null){
            msg = "You can't go in that direction";
      }else{
            currentLocation = nextRoom;
            msg = currentLocation.getLongDescription();
      }
}
```

Grading Criteria

There is a 50% penalty on programming projects if your solution does not compile.

- Stapled cover page with your name and signed pledge. (-5 pts if missing)
- Project requirements as specified above. (90 pts)
- Elegant source code that follows the GVSU Java Style Guide. (10 pts)

Late Policy

Projects are due at the START of the class period. However, you are encouraged to complete a project even if you must turn it in late.

- The first 24 hours (-20 pts)
- Each subsequent weekday is an additional -10 pts
- Weekends are free days and the maximum late penalty is 50 pts.

Turn In

A professional document **is stapled** with an attractive cover page. Do not expect the lab to have a working stapler!

1. **Cover page** - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project.
2. **Signed Pledge** – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " You are responsible for understanding and adhering to the School of CIS Guidelines for Academic Honesty.
3. **Time Card** – The cover page must also include a brief statement of how much time you spent on the project. For example, "I spent 7 hours on this project from January 22-27 reading the book, designing a solution, writing code, fixing errors and putting together the printed document."
4. **Steps to winning the game** – provide a step-by-step sequence that the player can take to win the game. This will allow the instructor to grade the game more easily. For example:
    a. Move North
    b. Pickup
    c. Move North
    d. Leave "wand"
5. **Sample Output** – cut and paste the results from the Game's main method showing the series of actions that lead to victory.
6. **Source code** - a printout of your elegant source code for the Item, Room, Game and GUI classes.
7. Compress the **BlueJ project folder** and submit the compressed file to **Blackboard**.