K23080184- Assignment 2 – Zuul-Better (Coraline inspired game)

Introduction

Objective of the game: To collect the items (displayed at the start of the game), the player must find them and give them to the ghost children (a character in the game). There are other items, but the player has a weight limit. There are other characters traveling around the building, but don't interact with the player. There is also an 'map' command, to allow the user to view location. The map is shown as an ascii image, with an asterisk to represent the player in the map.

The Game

The game checks the commands given by the user and carries out the corresponding methods/actions. If the aim of the game is completed, then the conditions are checked, and the game quits. The game creates rooms, and sets their exits to other different rooms, and the Items and characters are added to the rooms. The Magic room randomly teleports the player into any room in the building automatically. The characters are set into rooms, and when the player reaches that room, they start moving around the building.
The important features are the pick, drop and give methods created in the player class.

Base tasks:

- After reading through the code and understanding it, I began adding extra rooms to the game by creating a new object of Room class then giving it a description as a parameter. Then I set the exits to those rooms by calling the addRoom method in the room class.
- To add items, I created a class called items and a method, to get the name, weight and whether it can be picked up. I also created a HashMap for items in the Rooms class, then I created methods to addItems, removeItems and return a list of the keys just like creating the room exits. In the game class, I created the item objects in the createRoom method and called addItems method – to add item to a room.
- To enable the player to carry items, I used an Array List called inventory and created a pick command. First, I created a player class, then I added variables: maxWeight, currentWeight and initialised the inventory array list in the constructor. I assigned the maximum weight when I created the player object and in the game class. I added a new string in the validCommands String array 'pick'. Afterwards, I added a pickUpItem method in the player class – two parameters passed to it (item's name and the current room). I created another Array List to store the items in the current room by calling the getItems method in the Room class. I created a for each loop to go through all current room items, if the item's name is equal to the item name in the room, item can be picked, and the current weight added with the item's weight is still less than the max weight the player can carry. Then the item weight is added on to the current weight, the item is removed and a statement is printed out. Otherwise, other messages are printed out. To finish creating the pick command, in the game class I edited the process command with an else if statement to check whether command word is pick. This leads to pick method, which checks the second command word then call the pickUpItem method via the player object.
- In the play method, while condition is true it will process the player's commands but also check whether the winning conditions are true using if statement. I created a Boolean check method in the player class, and I created another array list called bag (where the character stores items given by the player). I then initialised a counter to 0 and used for each loop to iterate over all the items in the bag, an if condition checked whether the item name in the bag is equal to the item name typed. The condition uses 'or' to keep the loop iterating until all items are found in the bag list- each time the condition is true the count is incremented. When the loop ends, another if statement is checks whether the count is 3 then prints a message congratulating the player. Otherwise, other messages are printed and returns false. To check whether the visited rooms are at least two is through creating an array list of the

visited rooms and initialising in the constructor and then creating set and get methods for current room. The player class sets the current room as the starting room and adds it to the visited room array list. Everytime, set method is called in the game class, the room is added to the list. Then I created boolean checkRooms method where the size of the array is at least 2. If both these methods are true, the player wins.

- To implement a back command, I carried out the same steps as I did when making a pick command. This time I made a method goBack, in player class. First, I created a variable called lastVisitedIndex that is equal to the size of the last visited rooms array minus 2. If statement checks whether the index is greater than 0 otherwise a message is printed that player that they can't go back. Otherwise, array list get method returns the previous room and assign it currentRoom. Then I remove the room that the player is currently in, from the array list and print out the long description of the previous room.
- Inventory, drop and give are my three new commands. First, I created the show inventory method in the player class - If statement to check whether the inventory is empty. Otherwise, for each loop is used to run through the inventory array list and print the item name and weight. Then print out the total weight the player is carrying via current weight variable. I created the drop method in the player class and used an iterator, after facing "ConcurrentModificationException" error. Once the item has been removed from the inventory, I add it to the current room by calling the addItem method. The drop Item method uses a parameters to get command second word. Lastly, I created a give method in the player class (with parameters item name and string character) and another array list called bag for character to store the items given. I created a boolean variable itemFound set to false and a for each loop to iterate over the inventory items. An if statement was used to check whether the item name provided was equal to any of the items in the inventory, if so it was added to the bag list, removed from the inventory and the boolean variable was set to true. After the loop ended, I used another if statement to check the boolean variable is false and print out corresponding messages.

Challenge Tasks:

- To create the characters moving, I created a class called characters with a get method to return the name and created a HashMap for characters in the room class. I also created a getCharacters method to return an array list of the keys. Then in the create rooms method of game class, I created new objects for the Characters class and assigned them to room objects and created moveCharacters method, where I created another array list of all the characters in the current room of the player by calling getCharacters. I made a String array of all the different rooms I created and for each loop to go through the characters in the current room. Inside the loop, I used the random class object to return a random room from the string array and assign it to moveRoom variable. Then I printed out a message to make it look like the character has moved to another room. To prevent the same room being returned more than once, I looked researched (1) and used the java utilities package Collections and called the shuffle on the rooms list. The method is called in the goRoom, so every time the player reaches a room with the character the character starts to move.
- To extend the parser to understand three-word commands, I implemented another String variable (word3) and set it as null. Then added another if statement to get the third word the player had typed, in the other if statement (of the getCommand method) I added the a new parameter word3. I also edited in the command class by creating another field called thirdWord and initialised it in the constructor. Then I followed the same exact lines of code written for secondWord – creating a get method and a Boolean method.
- To create the magic room, I had to again use the random class. I decided to create another if statement in the 'else' part of the if statement, of the goRoom method. This the condition checks whether the next room's short description matches the short description of the

magic room- if the condition is true then I created a string array list of all the exit directions and use the random class to return a random direction and assign it to a variable randomRoom. Then I repeated the same lines of code when going into a new room, replacing nextRoom with randomRoom.

Code quality considerations:

Cohesion – when adding items and characters to the game, I decided to create new classes for these purposes. This represents high cohesion, as those classes have one responsibility and are well-defined (items class only stores information about the item as does the character class), other classes can access this information and make use of it. This increases the chance of reusability and maintenance, as the task is split into smaller jobs being executed by separate classes.

Coupling – for the items class, I created get methods to return the name, weight, and the Boolean value of whether it can be picked. I did the same for my character class, this enables independency between the classes. Therefore, I have shown loose coupling in my class design to improves the testability and allows for changing capabilities.

Responsibility-driven design – when adding a new functionality such as creating Items, I used a HashMap, and I decided to add this feature in the rooms class as I will be assigning the items to different rooms and list out the items in the current room. I also did the same with the characters. This will show that Room class is responsible for storing items and characters and makes it easier to modify and extend the program.

Maintainability- when implementing the pick and drop methods, I decided the create a player class. This shows both responsibility-driven design and refactoring, both are needed to maintain a good class design when the application is modified or extended. The player class makes sense to create for adding the pick, drop, and inventory methods as the player is going to be carrying out those actions. I also introduced the current room field in the player class from the game class, so if more players are they can hold their own items, current weight, and rooms.

Walk-through:

1. Type 'go east' to enter the bedroom. Enter 'pick rabbit' then type 'go west' to re-enter hall.
2. Enter 'go west' to enter the kitchen and 'go west' again to go outside. Then type 'go south' to enter the tearoom and 'pick ring' to collect the ring.
3. Type 'go north' to go back outside and 'go north' to enter the circus room. Then 'pick jack-in-the-box'.
4. Then type 'go south' to back outside and 'go east' to head back into the kitchen. 'go east' to enter the study.
5. Enter 'go south' to enter the hall again and 'go south' to enter the tunnel- or use 'back' to get all the way to bedroom. Enter 'give (each item name collected) children' to give the items to the ghost children. Once all items have been given the play loop will break.

Testing:

I conducted a lot of tests, running the game making sure all the features work. This also helped in refactoring and thinking ahead to create a better game design, such as creating a take command if the player gives the wrong item and map command to give the player an idea of their location. I also spent time trying to figure out loopholes and fixing it, such as not giving items to the right character or in the right room.

References:

(1) https://stackoverflow.com/questions/16000196/java-generating-non-repeating-random-numbers