Assignment 12
CSc 210 Fall 2017
Due December 6th, 8:00 pm MST

Introduction

In this final project, we will incorporate many ideas learned from this class into one program. Using your skills for decomposing a problem, you will create a fairly large dynamic adventure game.

You will turn in your code via github classroom. Here is the link to receive your repository: https://classroom.github.com/a/w9VM_84M

For each Java program assigned, you will turn in a .java file for that program. We should be able to run "javac *file*.java" and then "java *file*" on the test machine and your program should compile and run successfully.

Your Java programs should adhere to our style guidelines we will give to you. Here is the link: https://www2.cs.arizona.edu/classes/cs210/fall17/StyleGuidelines.pdf

The Adventure (150 points)

You will be creating a text based dungeon-crawler that is read in from a file and dynamically created. The game will consist of having a player that traverses through different rooms. Rooms will contain treasure that must be gathered by the player. Your score will be based on how much treasure you gather and moves made.

Part I: The Game Play

The game has a simple set of rules to play. First, the goal of the game is to gather as much treasure as possible and leave it in your stash room in the dungeon. Each piece of treasure has a value associated with it. Additionally, each item picked up by the player has weight associated with it, and our adventurer can only hold so much weight. The player can pick up treasure as they traverse the dungeon and drop it off at the stash room to relieve themselves of the weight and add the value to the final score. Rooms can also have a mob living in them. Each mob has a power level associated with it. In order to kill a mob, you most use a weapon on it that has a higher power level. A player does not know the power level of a mob until it is destroyed. You cannot gather treasure from a room until the mob has been destroyed. When the player finishes, they will be given a score based on the amount of treasure value they have and the amount of moves they made.

When the game starts you will give the player the following prompt:

```
Welcome to the adventure!
What class would you like to play as (Warrior, Dwarf, or Ranger)?
```

Valid input is Warrior, Dwarf, or Ranger (case insensitive). If the user enters an invalid choice, print to stdout “Invalid choice.” on its own line and keep waiting for input.

You may notice that there are several player classes. Each one has a unique benefit.

First, consider a Warrior. We mentioned earlier that weapons have power level associated with them. Power levels are represented as an integer. Warriors have the benefit of automatically adding +20 to the power level of any weapon used.

Next, consider a Dwarf. Each class has a maximum weight cary limit of 20. The Dwarf class gets an additional ten units of weight added to their carrying ability. So, a Dwarf may carry 30 units of weight.

Finally, consider a Ranger. It was said before that the final score is calculated by the amount of treasure value a player has and the amount of moves they made. A move is defined as moving from one room to another. For a Ranger, two moves count as one. In the final score, the moves for a Ranger should be calculated as floor(numMoves / 2).

Once the player chooses a class, you should print out:

```
Starting the adventure...
```

The game will consist of having a REPL to read in commands from the player. For every command, print the string “> “ at the start of the line. You may read in the following commands:

- **Look -** This command will print out a name and description of a room, its connections, and the mobs and items contained within. You should have the following format:

```
Name: <room_name>
Description: <room_description>
There are connections in the following directions: <dir_list>
Items: <item_list>
Mob: <mob_name>
```

Where <room_name> is the name of the room you are in, <room_description> is its description, <dir_list> is an alphabetically sorted list of directions that have a connection separated by commas, <item_list> is a list of sorted item names in the room separated by commas, and <mob_name> is the name of the mob in the room. Both <item_list> and <mob_name> should be “none” if it doesn't exist.

- **Examine *TYPE NAME* -** This command allows you to examine mobs and items in a room. TYPE may be either “mob” or “item” (case insensitive). NAME is then the name of the mob or item we would like to examine. When examining a mob, printto stdout:

  ```
  Name: <mob_name>
  Description: <mob_description>
  ```

  Note: you do not print out a mob's power level. That is not revealed until it is defeated. If the player would like to examine an item, use the following format:

  ```
  Name: <item_name>
  Weight: <item_weight>
  Value: <item_value>
  ```

  Or if it is a weapon:

  ```
  Name: <item_name>
  Weight: <item_weight>
  Power: <item_power>
  ```

  If any piece of the Examine command is incorrect, print out “Invalid command.” and continue reading commands.

- **Move *DIRECTION*** - This command will move your player to the room in the direction specified. If a direction is specified that does not have a room (or a valid direction is not given), print to stdout “Invalid direction.” on its own line and continue reading input.

- **Pickup *ITEM_NAME* -** This command is used to remove an item from a room and place it into your inventory where *ITEM_NAME* is the name of the item to be picked up. If the item is too heavy to be picked up, print to stdout “I cannot carry this item.” and continue reading commands. If the item specified is not found in the current room, print to stdout “Invalid item.” on its own line and continue reading commands. If the mob in the room has not been defeated yet, print to stdout “You must destroy the mob first.” and continue reading commands.

- **Drop *ITEM_NAME*** - This will remove an item from your current inventory and place it into the current room. If the item is not found in the player's inventory, print to stdout “Invalid item.” on its own line and continue reading commands.

- **Stash *ITEM_NAME*** - This command may only be used in your stash room. If this command is given in a room that is not the stash room, print to stdout “Invalid command.” on its own line and continue reading input. If the item is not found in the player's inventory, print to stdout “Invalid item.” on its own line and continue reading commands.

- **Fight *ITEM_NAME*** - This command fights the mob in the current room with the weapon specified. If this command is given in a room that does not have a mob, print to stdout “No mob found.” on its own line and continue reading input. If the weapon is not found in the player's inventory or the item is not a weapon, print to stdout “Invalid weapon.” on its own line and continue reading commands. If the player is able to defeat the mob, print to stdout:

  ```
  You destroyed <mob_name> with power level <mob_power>!
  ```

  Where <mob_name> was the name of the mob you destroyed and <mob_power> was their power level. Once defeated, the mob should no longer exist in the room.

  If the mob and item exist, but the item is not powerful enough to defeat the mob, the following should be printed to stdout:

  ```
  <mob_name>  defeated you! Try a different weapon.
  ```

- **Inventory** - This will print out all items currently found in the player's inventory in ascending order based on the items names. For each treasure item, use the following format if it is treasure:

  ```
  Name: <item_name>
  Weight: <item_weight>
  Value: <item_value>
  (blank line)
  ```

  Or if it is a weapon:

  ```
  Name: <item_name>
  Weight: <item_weight>
  Power: <item_power>
  (blank line)
  ```

Where blank line is just a line of empty space.

- **Quit** - This will end the game and print out the score. When received, print out:

```
Finishing game...
Final score: <game_score>
```

Where <game_score> is $\frac{\sum \textit{item values in stash}}{\#\ \textit{player moves}}$ and a player move is the count of valid fight or move commands (remember the special case for a Ranger).

When reading commands, ignore the case, recognize it as long as it is the same sequence of characters as defined in this spec. Commands will be given one per line. If an invalid command is given, print to stdout "Invalid command." on its own line and continue reading input.

Part II: The Game File

The first thing you should do is read in a "game file" which will define the dungeon you will play in. This file will be given as the first command line argument. If the first command line argument is not given or it is an invalid file name, you should print a message to stderr and exit your program with a status of 1. This game file will have three sections:

1. Definition of mobs, items, and weapons.
2. Definition of any rooms.
3. Stating any connections between rooms.

Definitions will have the following format:

```
define <definition type>
      ⋮
end
```

Where <definition type> is either Room, Mob, Treasure, or Weapon.

An Item definition will have the following format:

```
define Treasure
      name: <item name>
      weight: <float value>
      value: <int value>
```

```
end
```

Whatever string follows “name:” until the end of the line is the name of the treasure. The float following weight will be how heavy the treasure is. Finally, the int following value will be the value of the treasure.

A Weapon definition will have the following format:

```
define Weapon
      name: <weapon name>
      weight: <float value>
      damage: <int value>
end
```

Whatever string follows “name:” until the end of the line is the name of the weapon. The float following weight will be how heavy the weapon is. Finally, the int following damage will be the power level of it.

A mob definition will have the following format:

```
define Mob
      name: <mob name>
      description: <mob description>
      power: <int>
end
```

Whatever string follows “name:” until the end of the line is the name of the mob. Similarly, whatever string follows “description:” until the end of the line is the description of the mob. After power will be an integer representing the power level of the mob.

A Room definition will have the following format:

```
define Room
      name: <room name>
      description: <room description>
      items: <list of items>
      mob: <mob name>
end
```

Whatever string follows “name:” until the end of the line is the name of the room. Similarly, whatever string follows “description:” until the end of the line is the description of the room. Following “items” will be a list of strings separated by comma that are item names to be found in that room. Finally, there will be a mob type that appears in the room (none if no mob).

If you receive a definition of an item name or mob name that has been already been defined earlier in the file, print a message to stderr and exit with a status of 1.

The final section of the game file will be stating connections between rooms. They will take the following format:

```
<start room name> DIRECTION <end room name>
```

Where <start room name> is the name of the room the connection is starting at and <end room name> is the name of the room the connection is ending at. DIRECTION will be one of the following: NORTH, SOUTH, EAST, WEST. It tells us which direction our connection is travelling along. Note that a connection is directional.

If you ever have a second definition for a path that has already been defined, this is an error. An example of this would be:

```
Room1 SOUTH Room2
          ⋮
Room1 SOUTH AnotherRoom
```

If this happens, print a message to stderr and exit with a status of 1.

The start room (which is also the stash room) will be the first room defined in the game file. Note that it is fine for items or mobs to be in the start room.

You are required to turn in a file called Game.java that contains your main method. You should turn in at least 10 class files (including the Game class). You are highly encouraged to use concepts discussed in class such as inheritance, graphs, hash maps, linked lists, sorting, and others. You are allowed to use any Java defined classes as long as your code compiles on lectura using javac.

Before beginning to write any code, you should use the methods of decomposing a problem discussed in class to reduce the program into smaller pieces, more approachable. This would be a great exercise using these tools and should make starting the program easier.

Miscellaneous

In order to receive a grade, your repo must only contain your source java files in the root directory of your repo.

This assignment will be submitted through github classroom. Make sure all of your code you would like to submit is in your repository when the due date arrives.

Note: Your output must match what is defined here in the spec. We will give you a small selection of test cases so you can make sure you have the right format. Do not print extraneous output, you may lose a lot of points (This includes prompts when reading in input!). **These test cases can be found in your repo.**

Remember, do not cheat! Refer to the syllabus and first lecture for more information.