Homework 4: Adventure Game

Description:

The purpose of this assignment is to build a simple text-based adventure game where the player explores the game world by travelling north, west, east, or south. The player chooses where to go and the game describes that area of the game world and what options that they have to explore. You are the designer tasked with creating your own map layout, whether it a castle, a hotel, a dungeon, or a space station. See the last page for sample output of a final build of such an adventure game.

Managing this project:

You should subdivide this assignment into the following three development phases. First, you should design your *game world* using paper and pencil. Second, you should model your *game world* into data and develop an algorithm, in pseudo code, that defines how to move within this model of your game world. Third, implement your pseudo code into a fully playable java application.

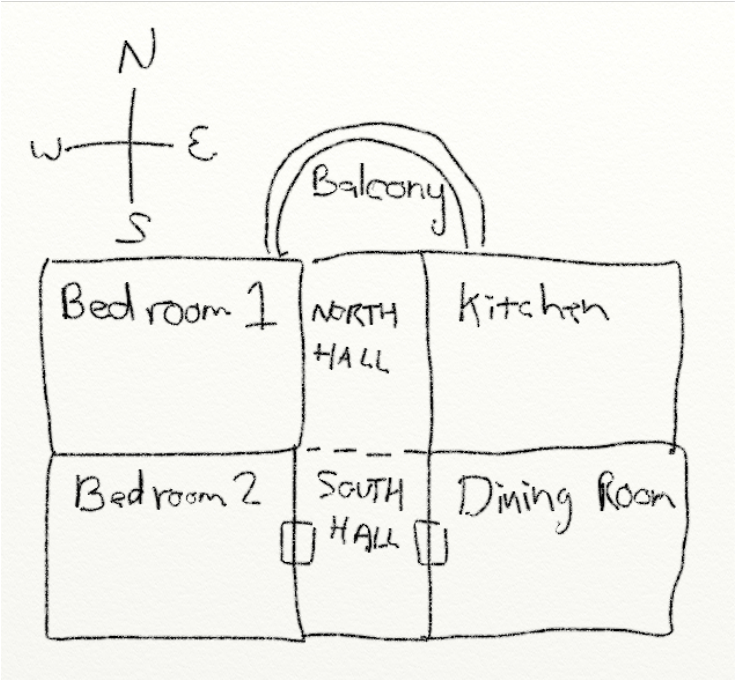
Required Concepts:

This homework requires that you implement the following concepts:

1. Array for String data (Room Descriptions)
2. Multidimensional array for integer data (Room Exits)
3. Initializer lists (Initializing Room Descriptions)
4. Multidimensional Initializer lists (Initializing Room Exits)
5. Retrieve values from array (Getting a room's description)
6. Retrieve values from multidimensional array (Getting room's exit)
7. Named constants (*Column indexers* for exit array)
8. Repetition Statement (Game loop)
9. Selection Statement (Execute the player's choice)

Phase 1: Designing Your Dungeon

Sketch out the *dungeon* that you want to create. Your only restriction is that each room can only have one exit per side: north, west, east, and south. A *room* must connect to a minimal of 1 room and to a maximum of 4 rooms. So, for example a sketch of your dungeon might look something like this:



In this example layout the rooms are connected as follows:
Bedroom2 connects: *north* to **Bedroom1** and *east* to **South Hall**.
South Hall connects: *north* to **North hall**, *east* to **Dining room**, and *west* to **Bedroom2**.
Dining Room connects: *north* to **Kitchen** and *west* to **South hall**.
Bedroom1 connects: *east* to **North Hall**, *south* to **Bedroom2**.
North Hall connects: *north* to **Balcony**, east to **Kitchen**, *west* to **Bedroom1**, and *south* to **South Hall**.
Kitchen connects: *west* to **North Hall** and *south* to **Dining Room.**
Balcony connects: *south* to **North Hall**.

Phase 2: Modeling Your Dungeon as data

Next, you must convert or *model* your dungeon's map as data. *Modeling* is essentially translating the game map into a form that the programming language can process. Learning to *model* your solutions using a programming language is a very important skillset for software developers. To do this, you must identify the important properties that define a dungeon so that it can be implemented into your game program. Since a dungeon is simply a collection of rooms then we can use **arrays** to model the dungeon. So now, we must just identify what defines a *room*?

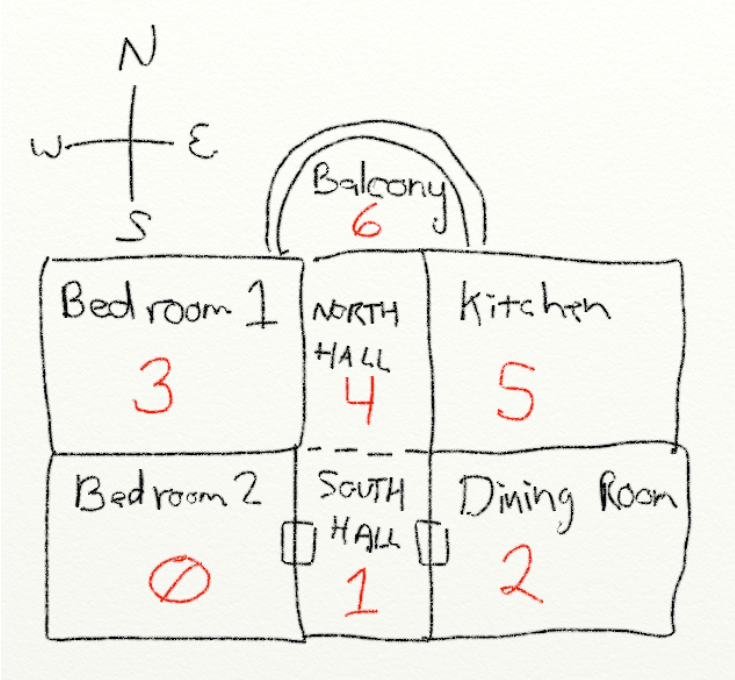
There are two properties that define a *dungeon room*:

1. A text description of the room and its contents.
2. The exits for the room.

Therefore, the dungeon will require two different arrays to properly model, one for its descriptions and the other for its exits.

Dungeon rooms as array elements:

The first step is to prepare your dungeon to be mapped into an array. Number all of the rooms in your dungeon, starting at zero:



Sample Numbering:

```
0: Bedroom2
1: South Hall
2: Dining Room
3: Bedroom1
4: North Hall
5: Kitchen
6: Balcony
```

The number for each room represents the index into the array for that room's properties.

Modeling the Room's Descriptions:

The responsibility of the *room descriptions* property is to convey to the player all relevant information regarding the room. Recall that this game contains no graphics, so everything important must be communicated to the player through the text. Our example room descriptions are pretty bland, be creative and make yours interesting.

A String array can be used to contain the dungeon's room descriptions. A graphical depiction of the array could be illustrated as follows:

Room Descriptions Array

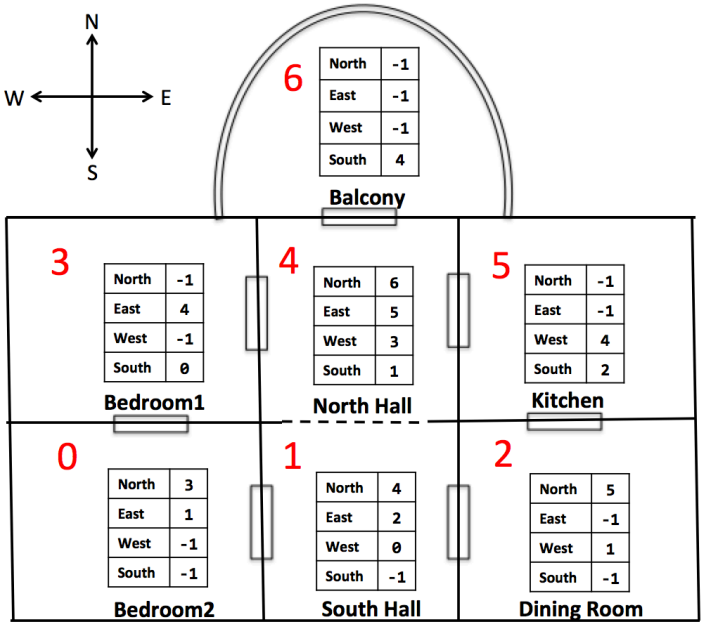
index	value
0	"You are in the guest bedroom, exits are north and east."
1	"You are in the South Hall, exits are north, east, and west."
2	"You are in the Dining Room, exits are north and west."
3	"You are in the Master Bedroom, exits are east and south"
4	"You are in the North Hall, exits are north, east, west, and south"
5	"You are in the Kitchen, exits are west and south"
6	"You are on the Balcony, Exits are south."

Notice that the index numbers matches the previous assigned room numbering.

Modeling the Room's Exits:

The responsibility of the *room exits* property is to maintain each room's corresponding set of north, east, west, and south exits. This data is critical for interconnecting the rooms together in the same order as they appear on your map.

Use your sketch to figure out how all the rooms connect. For example, room 0 in the sample map connects to room 3 to the north, room 1 to the east, and has no connections to the south and west. So 3 and 1 are room 0's "exits." Use a sentinel value of -1 for the directions with no exits. The diagram to the right shows all of the room's exits.



The data for the room's exits can be stored in the form of a table, where each row corresponds to a room. This table would have four columns: north, east, west, and south. The value held in each cell of this table would correspond to the room connecting in that direction. If there is no room in that direction, then use sentinel value. Recall that a sentinel value should not be in scope of valid values, so use -1.

Use a multidimensional integer array to model the *room exits* table. A graphical depiction of the array could be illustrated as follows:

Room Exits Array

index	North	East	West	South
0	3	1	-1	-1
1	4	2	0	-1
2	5	-1	1	-1
3	-1	4	-1	0
4	6	5	3	1
5	-1	-1	4	2
6	-1	-1	-1	4

Notice that the index numbers matches the previous assigned room numbering.

Putting the two arrays together:

We now have two collections: an array of descriptions and a multidimensional array of exits; which represent the two distinct properties that defines a room. Thus, we have everything required to properly model the dungeon data.

Room Descriptions Array

index	value
0	"You are in the guest bedroom, exits are north and east."
1	"You are in the South Hall, exits are north, east, and west."
2	"You are in the Dining Room, exits are north and west."
3	"You are in the Master Bedroom, exits are east and south"
4	"You are in the North Hall, exits are north, east, west, and south"
5	"You are in the Kitchen, exits are west and south"
6	"You are on the Balcony, exits are south."

Room Exits Array

index	NORTH	EAST	WEST	SOUTH
0	3	1	-1	-1
1	4	2	0	-1
2	5	-1	1	-1
3	-1	4	-1	0
4	6	5	3	1
5	-1	-1	4	2
6	-1	-1	-1	4

For example, let's use these two arrays to verify that they properly model the original sketch. Starting at room 0 (i.e. index 0), the room description is "You are in the guest bedroom, exits are north and east" and the room exits are room 3 to the North, room 1 to the East, with no exits to the west or south. This matches our drawing. As an exercise, see if you can verify that the other six rooms' descriptions and exits also match the original sketch.

Developing a movement algorithm:

Now that your game world has been modeled into pseudo code. Let's discuss creating an algorithm that uses these two room arrays to traverse the dungeon.

Step 1:
Setup a variable and initialize it to hold the current room number that the player is in. The value of this variable must be one of the **index values** in the arrays. Also, setup a loop control variable for a game loop and default its value to true. The game loop should contain steps 2 through 4.

Step 2:
Print the current room's description. Do this by using the current room's variable as the index into the *room's description array* and dereference the String to print it. *Note that the room's description also serves as the prompt for user input.*

Step 3:
Get the user's choice for which direction they want to go in or if they want to quit. Expected user inputs include options for: (n)orth, (e)ast, (w)est, (s)outh, and (q)uit.

Step 4:
Based on the user's choice perform their requested action using a selection statement. Also, setup a local variable that will hold the next room number to move into.

If the user chose to move in a direction (i.e. north, east, west, or south), then index into the *room exits array* to get the next room number to move the player into. Since this is a two-dimensional array, it requires two index values. The first index uses the current room value; which dereferences this current room's 4 possible exits. The second index is determined by which of the 4 exit directions to get based on player input. So if the player selected north we would use the 0^{th} column value, if east the 1^{st} column value, if west the 2^{nd} column value, and if south the 3^{rd} column value.

After we get a value for next room number we must determine if it leads to a valid room. If the next room's value is a -1 then it is invalid and report to the player that they cannot go that way and repeat step 2. Otherwise, set the current room variable to the value of next room variable, and repeat Step 2.

If the player chose to quit then set the loop control variable to false.

Any other input should report an invalid input error to the player.

Phase 3: Implementing the Adventure Game

The final phase is to implement the movement algorithm into source code. The following instructions provide basic hints on how you might do that using the material learned from chapter 6. You are free to optimize your own code provided the underling functionality is similar.

Instructions:

1) Create a `.java` file and setup a class.

2) Declare a named constant, `NUMBER_OF_ROOMS` and initialize it to the number of rooms your dungeon has (e.g. 7 in this example).

3) We will associate each possible direction (n, s, e, w) with an integer value that we will use as an index into the arrays that hold our room information. Create named constants for each of these as `NORTH=0, EAST=1, WEST=2,` and `SOUTH=3`.

3) Create a static string array (**`String[])`** to hold your room descriptions. The value of the array at a particular index will correspond to the description of the room with that number. Use an initializer to populate the array -- the Strings should be added in the order corresponding to the numbering of your rooms.

4) Create a 2-dimensional static integer array (**`int[][])`** to hold each room's exits. This array's dimension should be `NUMBER_OF_ROOMS`-by-4. The first index will be used to determine which room we are describing and the second index will be used to determine which direction we are going. The value at each index will be the room number that exit leads to. For example, in the diagram above, the value of this array at [0][NORTH] will be 3, since room 0 has an exit to room 3 to the north. If the room does not have an exit in a particular direction, use -1 as a special flag value to indicate that there is no exit. For example, the values of the array at index 0 would be {3, 1, -1, -1}, since there are exits from room 0 to the NORTH and EAST, but not to the SOUTH or WEST. Use an initializer to populate this array according to your dungeon map.

5) Declare a static class variable that stores the number of the current room the player is in. Assume the player always starts in room 0.

6) In your main method, create a game loop that allows the player to interact with the world. It should perform the following steps in order:

- Print a description of the current room the player is in. Use your **`String[]`** array you created, indexed at the current room number.

- Ask the player which direction they would like to go. The expected inputs should be:
    - **n** : go to the north
    - **s** : go to the south
    - **w** : go to the west
    - **e** : go to the east
    - **q** : quit the game and exit the program.

  The input should be case insensitive (e.g. "N" means the same thing as "n"). You can get the user input character using the **`nextChar()`** method in the Scanner class.

- If the user enters 'q' to quit, then the program should exit with an appropriate goodbye message.

- Otherwise, if the user enters a direction (n, s, e, or w), then the program should use the **`int[][]`** array you created earlier to determine what to do next:
    - If there is an exit in the desired direction from the player's current room, then the player's current room should be updated to be the number of the room where the exit leads.
    - If there is no such exit (i.e. the array value is -1), then an appropriate message should be displayed indicating that the player can't go in that direction.

- If the user enters invalid input, an appropriate message should be displayed and the prompt should be displayed again.

The game loop should repeat until the player decides to quit the game.

Of course, you may want to split up your game logic into different methods when appropriate. Remember that a good method should perform a single specific task. For instance, you might have a method `getRoomInDirection(int room, int direction)` that returns the destination room number (or -1) given a room number and a direction code.

Bonus:

Add additional functionality beyond these specifications. For example, you might encounter monsters, pick up treasures, or have interesting victory or defeat conditions. Be imaginative and have fun with this project!

Sample Run:

```
?You are in a dusty castle room.
Passages lead to the north and south.
What direction? n
You are in the armory.
There is a room off to the south.
What direction? s
You are in a dusty castle room.
Passages lead to the north and south.
What direction? s
You are in a torch-lit hallway.
There are rooms to the east and west.
What direction? e
You are in a bedroom. A window overlooks the castle courtyard.
A hallway is to the west.
What direction? w
You are in a torch-lit hallway.
There are rooms to the east and west.
What direction? w
You are in the kitchen. It looks like a roast is being made for supper.
A hallway is to the east.
What direction? w
Can't go that way.
You are in the kitchen. It looks like a roast is being made for supper.
A hallway is to the east.
What direction? q
You leave the castle and go home.
```

Submissions:

You should create a new folder named “hw4” in your gitlab project, put your source files in that folder, and add, commit, and push those files to gitlab.