Final Program (Text-based game)

Abstract:

This program is designed for more advanced coders, either as a final project to the intro level class or an early project in the class following the intro level. This program requires an understanding of object oriented programming, 2D arrays, and file processing, so students should have already been assigned projects dealing with these topics individually before tackling this one. This program will help further students understanding of object oriented programming by allowing them to experiment with it in a "fun" way. It may be very beneficial to a student's understanding of OOP when they think of objects as tangible objects in a game world rather than abstract data manipulation tools. This is a good project because a lot of students get into the Computer Science major in hopes of getting into game development, and this allows students to experiment with game development and respect the complexities of it.

Overview:

For this assignment you will be creating a text-based game, where the player will enter commands in the console to manipulate the game world. This program will involve object oriented program, file processing, 2D arrays, and arraylists.

Specifications:

Game World (main): Process data from a file and generate a 2D array (game board) of room objects based on information from the file.

Objects:

- *Room*: A room will only ever need a North, East, South, and West wall/pathway/door. A room's exit may require the usage of some item to progress to a previously inaccessible room, imagine a locked door requiring a key to open it. A room can only contain a maximum of 1 item.
- *Player*: The player object is manipulated by commands entered by the user. The player can pick up items and store items in there "inventory" and use items from their inventory. I recommend storing the current location of the player inside this object.
- *Item*: Stores a description of the item and what object this item can manipulate.

You can assume that the game board will always have walls surrounding it, meaning when moving you should never encounter an array out of bounds exception. You may also assume that the path through the dungeon is directed.

Input: (case insensitive)

- "Exit" - exits the game.
- "Move North" - Move one room North, if there isn't a wall in the way. *Hint: If we think of this in terms of a 2D graph, we'd be moving one row upwards.*
- "Move South" - Move one room South, if there isn't a wall in the way.
- "Move East" - Move one room East, if there isn't a wall in the way.

- “Move West”- Move one room West, if there isn’t a wall in the way.
- “Look” - print the description of the room to the console. Also print any item within the room.
- “Grab” - Removes the item from the room and places it in the player’s inventory.
- “Use [Item name]” - if the item is in the player inventory, then we use the item in the room we are currently in. Should print whether the item was used successfully or not (whether the item was needed inside this particular room). If the item was used successfully then it should be removed from the player’s inventory, it’s usage description should be printed, and the respective door should be open (stored in Path to Open). If item was used unsuccessfully the item should NOT be removed from the player’s inventory and a message should be printed that the item was used unsuccessfully.
  “Inventory” - Print a list of the items inside the player’s inventory.

Expected File Format:
Line 1: [No. of rows] [No. of columns] [start_row] [start_col]
Line 2-n: item: [Item name] [door_name] [Usage Description]
Line n-n+(no of rooms): room: [N. wall] [E. wall] [S. wall] [W. wall] [item [name] inside room] [door_name] [description of room]

Assumptions for file processing:
- Items to come before rooms in the file.
- File will always have the correct number of arguments (separated by space) per line. If there is no item/item required in a room, the name of the item will be replaced with the string “null”..
- Rooms will always start from the top left, move right towards the no. of columns, then move back to column 0 for the next row.
- Rooms and items will always have the headers “room:” and “item” in front of them.
- 2D array and play starting position information will always be the first line of the file.

Program Behavior:
Your program will read specified data from a file and create a “game world” based around it. The program will then accept input from the user to manipulate the player in the game world. For this assignment you're essentially going to have 2 different loops, one for file processing and one to represent the “game loop”. The game loop is most easily defined as a loop running in the background awaiting user input and then processing that data; the game loop should only end if the user finishes the game, or types exit. To start this program I would recommend implementing the needed objects first and designing how you want to go about moving through the rooms. After, you can begin writing the file processing loop to fill the game board (2D array) with the needed room objects and items. Finally, once the board is properly populated, you can begin writing the game loop where you will be expecting input from the player.

Error Conditions: Input file does not exist, incorrect number of lines read to fill rooms.

Grading Rubric:
External:

- **/5 - 4 move commands, look command.**
- **/3 - use command. 3 points for perfect. -1 for error. If the student made a reasonable attempt they should get at least 1 point.**
- **/2 - grab command. 2 points for perfect, 1 point for attempt.**