Skeleton Code Write Up

Sauhaard

Contents

Preliminary Material

Background Information

- The program can be used to play many different text adventure games because it can read different files
- Every command input into the console by the player is parsed and checked if it is an actual command or not

Commands

go

The go command moves the player from 1 location to another - Must be followed by either of these directions:

1. north
2. south
3. east
4. west
5. up
6. down

get

The get command allows the player to pick up an object

- The object has to be in their current location
- The get command must be followed by the name of the item you want to pick up
  - If the item is able to be taken, then the items location id is changed from the current location to the user's inventory

use

This allows the user to use an object that is either in their *current* location or inside their *inventory*

examine

This command provides the player with a detailed description of the object or character

- You can examine an item in the *same location* or in your *inventory*.
- It must be followed by the *name of the object* or you can write *examine inventory* which will display a list of the items in the user's inventory

say

The player will say whatever is written after the say command

quit

This quits the game

read

This will display the text written in an object that can be read, such as a book. - The item has to be in their inventory or in the same location.

move

Allows the player to move an object in their location.

- The name of the object follow the move command
- If the object is moved, then a description of what happens after moving the object is displayed in the console

open

Allows the user to open an item

- A common item to open is the door, doors are in-between 2 locations

close

Allows the user to close a closable item in the same location as them.

playdice

This command lets you play a dice game with another character.

- The name of the character you want to play with must be specified
- If you win you get to take an item from the opponent (their inventory will be displayed to you and you choose)
- But if you lose they will take a random item from you
- Both characters must have a die if you want to play

The Character

Each character has:

1. Unique character ID
2. A name
3. A description
4. A current location

Your personal ID is ***1001*** but everyone else's is ***below 2000***.

Items

Each item has:

1. Unique item ID
2. Item name
3. Description of item
4. Location ID (it's current location)
   - If the location is ***greater than 2000*** than the item is inside of another item
5. Current status of the item
   - Some examples include:
     - open, locked, close, tiny, small, medium, large, edible, fragile, usable, gettable, water, liquid, container, horrible, occupied, off, powered, lightable, weapon.
6. Command that will work when used on the item
   - It can be: get, user, move, open, close, read.
7. List of the results of what will happen when a command is used on it.
   - The results will be in the **same** order in the ArrayList that the **usable commands** for the item is in.
     - This means if 'open' is the first command in the usable commands array list, then it will also be the first thing inside of the results array list.

Please note that:

- All item id's are ***greater than 2000***
- Doors are actually two items
  - The difference between the location ID's will be ***10,000***
- Item ID's between ***10001 and 1999*** are in a **characters inventory**.
  - This can be you personally or a guard.
- An item with a "usable" status can be used as long as it is in the players location or inventory, same thing with "gettable", it can be gotten if it is in the same location as the user
- Each status value in the program variable is separated with a comma
- Some status value's are not used in the Skeleton Program.

Location

Each location has:

1. A unique location ID
2. A description of the location
3. ID's of the directions you can go to from that location
    - These id's are any of the following: north, east, south, west, up, down.
    - If there is ***no*** location in a particular direction, then the id will be ***0***.
        - So for example, if I go north from a location that you cannot go north from, then the north ID in that location will be 0.
4. All location ID's are less than ***2000***.

Constants and Sub Classes

Constant Declarations

```
    static final int INVENTORY = 1001;
    static final int MINIMUM_ID_FOR_ITEM = 2001;
    static final int ID_DIFFERENCE_FOR_OBJECT_IN_TWO_LOCATIONS = 10000;
```

Place (Sub Class)

```
    class Place {
        String description;
        int id, north, east, south, west, up, down;
    }
```

Character (Sub Class)

```
    class Character {
        String name, description;
        int id, currentLocation;
    }
```

Item (Sub Class)

```
    class Item {
        int id, location;
        String description, status, name, commands, results;
    }
```

Important Methods

boolean go(Character you, String direction, Place currentPlace)

Please note that currentPlace.[direction] are things such as currentPlace.north, or currentPlace.down, etc.

- All of these case statements checks whether you can move in that direction or not
- If you can move in that direction then the currentPlace.[direction] will NOT be 0.
- If its not 0 then your current location will be set to the location id in the currentPlace.[direction]
- It will return FALSE if the currentPlace.[direction] **EQUALS** 0.
- If moved is false however, then it sends a message aswell.

```
// Returns true when you can move in that direction
    boolean go(Character you, String direction, Place currentPlace) {

        boolean moved = true;
        // Checks whether you wrote north or south etc
        switch (direction)
        {
            case "north":
                if (currentPlace.north == 0) {
                    moved = false;
                } else {
                    you.currentLocation = currentPlace.north;
                }
                break;
            case "east":
                if (currentPlace.east == 0) {
                    moved = false;
                } else {
                    you.currentLocation = currentPlace.east;
                }
                break;
            case "south":
                if (currentPlace.south == 0) {
                    moved = false;
                } else {
                    you.currentLocation = currentPlace.south;
                }
                break;
            case "west":
                if (currentPlace.west == 0) {
                    moved = false;
                } else {
                    you.currentLocation = currentPlace.west;
                }
                break;
            case "up":
                if (currentPlace.up == 0) {
                    moved = false;
                } else {
                    you.currentLocation = currentPlace.up;
                }
                break;
            case "down":
                if (currentPlace.down == 0) {
                    moved = false;
                } else {
                    you.currentLocation = currentPlace.down;
                }
```

```
                break;
            default:
                moved = false;
        }
        if (!moved) {
            Console.writeLine("You are not able to go in that direction.");
        }
        return moved;
    }
```

examine

- This method basically takes in either a characters name or an items name.
- It finds the item/character which you specified with the name
  - "examine [insertName]"
- It then prints out the characters description to the user

```
    // Method prints out a detailed description of the item being examined
    void examine(ArrayList<Item> items, ArrayList<Character> characters, String itemToExamine, int currentLoc
        int count = 0;

        // If the user says "examine inventory"
        if (itemToExamine.equals("inventory")) {
            // Then it will just print out the inventory using displayInventory
            displayInventory(items);
        }

        // Else if the user specified an item
        else {
            // Finds the index of the item with the same name as the itemToExamine variable
            int indexOfItem = getIndexOfItem(itemToExamine, -1, items);
            // if indexOfItem is NOT -1 (-1 means it is not in the items arraylist)
            if (indexOfItem != -1) {
                // If the item ArrayList has an item with the name of the item to examine AND the item is in
                if (items.get(indexOfItem).name.equals(itemToExamine) && (items.get(indexOfItem).location ==
                        // OR the item is in the current location
                        || items.get(indexOfItem).location == currentLocation)) {
                    // then it will print out the item's description field
                    Console.writeLine(items.get(indexOfItem).description);

                    // if its a door then it will print out the status, open or closed
                    if (items.get(indexOfItem).name.contains("door")) {
                        displayDoorStatus(items.get(indexOfItem).status);
                    }
                    // if its a container then it will just print out the items contained inside
                    if(items.get(indexOfItem).status.contains("container")) {
                        displayContentsOfContainerItem(items, items.get(indexOfItem).id);
                    }
                    // method returns and ends here if you were just examining an item
                    return;
                }
            }

            // this code runs if examine was used on a character
            while (count < characters.size()) {
                if (characters.get(count).name.equals(itemToExamine)
                        && characters.get(count).currentLocation == currentLocation) {
                    // Finds the character with the mathcing name AND in the same location, then prints the d
                    Console.writeLine(characters.get(count).description);
```

```
                return;
            }

            // This while loop traverses through the characters array list til it finds a matching name
            count++;
        }

        // If all else fails!
        Console.writeLine("You cannot find " + itemToExamine + " to look at.");
    }
}
```

Utility Methods (Parsing, Etc):

getResultForCommand

getPositionOfCommand

This method is basically called to find where the "command" starts from in the commandList of an item.

The commandList of an item is just a string of all the commands you can perform on an item.

```
// This methods takes a string "commandList" of all of the acceptable commands for an item
// Then extracts the position of where the "command" specified is.
int getPositionOfCommand(String commandList, String command) {
    int position = 0, count = 0;

    while (count <= commandList.length() - command.length()) {
        String test = commandList.substring(count, count+command.length());
        // checks if the extracted string is = to the command required or not
        if (test.equals(command)) {
            // if it is then it will return the position of where the command starts from
            return position;
        } else if (commandList.charAt(count) == ',') {
            position++;
        }
        count++;
    }
    return position;
}
```

getIndexOfItem

This method traverses the item's ArrayList to find an item that matches either the itemIDToGet or the itemNameToGet - When itemIDToGet is specified, it will iterate through the items ArrayList and then return the index of the item that has the same itemID as the itemIDToGet - If an item name is specified however, then the itemIDToGet will be -1 - The iteration will just check if any items match the name that was specified into itemNameToGet - If theres a match then it will return that index

- This method is called with either itemIDToGet = ***-1***, meaning t

```
// returns int value of the item index (itemID)
int getIndexOfItem(String itemNameToGet, int itemIDToGet, ArrayList<Item> items) {
    int count = 0;
    boolean stopLoop = false;

    // If you enter the itemIDToGet as -1, then it will traverse the items ArrayList to find
    //the item index that has the name = to itemNameToGet

    //while loop runs while stopLoop = false and while count is smaller then
```

```
        // the size of the items array
        while (!stopLoop && count < items.size()) {

            // This while loop only stops when an item's name is = to itemNameToGet
          // OR when an item's ID is = to itemIDToGet
            if (itemIDToGet == -1 && items.get(count).name.equals(itemNameToGet)
            || items.get(count).id == itemIDToGet) {

                stopLoop = true;
            } else {
                count++;
            }
        }
        // We are outside the while loop now, if the stopLoop boolean = FALSE then it will
        //just return -1
        // returning -1 means it is not in the items array
        if (!stopLoop) {
            return  -1;
        } else {
            // But if stopLoop is true then it will return the counter.
            // The counter is the index where the item searched for is located in the items ArrayList
            return count;
        }
    }
```

String getInstruction()

```
    //get instruction basically gets the keyboard input data from the user
    String getInstruction() {
        String instruction;
        // writes a new line (\n) and writes a ">"
        Console.write("\n> ");
        //instruction is then read from the keyboard input
        instruction = Console.readLine().toLowerCase();
        return instruction;
    }
```

String[ ] extractCommand(String instruction)

This method will take in the instruction typed in by the user such as “go north” and then separate out the “go” and the “north”. Returns a String array, with the first index being the command and the second index being the instruction.

```
// extractCommand basically formats the instruction properly
// and seperates the command "go" etc from the item
// or direction or whatever "north" for example

    String[] extractCommand(String instruction) {
        //command is the full instruction the user specified
        String command = "";

        // if instruction doesn't contain a blank space
        if (!instruction.contains(" ")) {
             // command has no spaces (no action specified,
             // for example: "go" without having any directions
           command = instruction;
            //returns a new 2 dimensional array that
            // has the command and the instruction (ends method)
            return new String[]{command, instruction};
```

```
        }
        // while loop checks if length is bigger than 0
        // and will run if there is a blank space at the beginning
        while (instruction.length() > 0 && instruction.charAt(0) != ' ')
        {
            command += instruction.charAt(0);
            instruction = instruction.substring(1);
        }

        while (instruction.length() > 0 && instruction.charAt(0) == ' ')
        {
         //cuts the first character
            instruction = instruction.substring(1);
        }
        return new String[]{command, instruction};
    }
```

Important Information In General

Getting Characters Current Location:

```
// Returns an integer that represents the current location ID
characters.get(0).currentLocation;
```

Print Output Methods

say

Just prints out what is put into the argument.

```
    void say(String speech) {
        Console.writeLine();
        Console.writeLine(speech);
        Console.writeLine();
    }
```

displayDoorStatus

```
    // If door is being examined it uses this method
    void displayDoorStatus(String status) {
        if (status.equals("open")) {
            Console.writeLine("The door is open.");
        } else {
            Console.writeLine("The door is closed.");
        }
    }
```

displayContentsOfContainerItem

```
    // Method prints out the items inside of a container
    void displayContentsOfContainerItem(ArrayList<Item> items, int containerID) {
        Console.write("It contains: ");
        boolean containsItem = false;

        for (Item thing : items) {
            if (thing.location == containerID) {
```

```
                // containsItem is true if it has found an item beforehand
                if (containsItem) {

                    Console.write(", ");
                }
                containsItem = true;
                // prints out the item's name
                Console.write(thing.name);
            }
        }
        // If item's have been printed out it will end with a fullstop
        if (containsItem) {
            Console.writeLine(".");
        }
        else {
            // Else it will just print out nothing.
            Console.writeLine("nothing.");
        }
    }
```