```
package osu.cse2123;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;

/**
 * A program that implements the bare bones of a text adventure game
 * - a map that the player can move through and the ability to pick up
 * and drop items in different rooms.
 * @author Asia Fortuna
 * @author Sara Fortuna
 * @version 7/30/2022
 */

public class Game {

	/**
	* This method changes the direction given by the text file to the actual
direction name.
	* @param  direction    direction string to change.
	* @return new direction string.
	*/
	public static String changeDirection(String direction){
		String newDirect = direction;
		//If-else statement to change to appropriate
		if(direction.equals("N")) {
			newDirect = "north";
		}else if(direction.equals("S")) {
			newDirect = "south";
		}else if(direction.equals("E")) {
			newDirect = "east";
		}else if(direction.equals("W")) {
			newDirect = "west";
		}else if(direction.equals("U")) {
			newDirect = "up";
		}else if(direction.equals("D")) {
			newDirect = "down";
		}
		return newDirect;
	}

	/**
	* This method sets up a room and its attributes.
	* @param  roomAttributes     attributes to fill in room with.
	* @param  roomDirections     contains all directions in each room.
	* @return room that has all appropriate attributes associated with it.
	*/
	public static Room setRoom(List<String> roomAttributes,
Map<String,List<String>> roomDirections){
		Room room = new SimpleRoom();
		//Set up room name
		room.setName(roomAttributes.get(0));
		//Set up room directions and desc.
```

```
            String[] directions = roomAttributes.get(1).split(",");
            List<String> collectionOfDirections = new LinkedList<String>();
            //For loop to set room with its exits and the exits to the list of
exits
            for(int i = 0; i < directions.length; i++) {
                  //Splits the string so that the direction and desc is separated.
                  String[] mapAttr = directions[i].split(":");
                  //Change shorten direction string to actual direction string
                  mapAttr[0] = changeDirection(mapAttr[0]);
                  collectionOfDirections.add(mapAttr[0]);
                  room.setExit(mapAttr[0], mapAttr[1]);
            }
            //Put all possible directions in room direction collection
            roomDirections.put(room.getName(), collectionOfDirections);
            //for loop to fill in description
            List<String> desc = new LinkedList<String>();
            for(int j = 2; j < roomAttributes.size(); j++) {
                  desc.add(roomAttributes.get(j));
            }
            room.setDesc(desc);
            return room;
      }

      /**
      * This method prints the directions in a room in the correct order.
      * @param  orderedDirections   contains all the directions of room to print
      */
      public static void printCorrectOrderedDirection(List<String>
orderedDirections){
            List<String> copy = orderedDirections;
            //While loop to print each direction in list until 1 is left.
            //Loop is to print direction in correct order.
            while(copy.size() > 1) {
                  if(copy.contains("east")) {
                        System.out.print("east" + ", ");
                        copy.remove("east");
                  }
                  else if(copy.contains("north")) {
                        System.out.print("north" + ", ");
                        copy.remove("north");
                  }
                  else if(copy.contains("south")) {
                        System.out.print("south" + ", ");
                        copy.remove("south");
                  }
                  else if(copy.contains("west")) {
                        System.out.print("west" + ", ");
                        copy.remove("west");
                  }
                  else if(copy.contains("up")) {
                        System.out.print("up" + ", ");
                        copy.remove("up");
                  }
            }
            //Prints the last direction contained in the room
            System.out.println(copy.get(0));
            copy.remove(0);
      }
      /**
```

```
      * This method displays all the qualities of the room.
      * @param  currentRoom   room that user is currently in
      * @param  roomDirections     contains all directions in each room.
      * @param  items   collection of item names and the actual items.
      */
      public static void displayRoom(Room currentRoom, Map<String,List<String>>
roomDirections, Map<String,Item> items){
            //Print name and description
            System.out.println(currentRoom.getName());
            System.out.println(currentRoom.getDesc());
            //Print exits
            System.out.print("There are exits in the following directions: ");
            List<String> orderedDirections = new LinkedList<String>();
            //Set variable to list of available directions
            for(int i = 0; i < roomDirections.get(currentRoom.getName()).size(); i+
+) {

      orderedDirections.add(roomDirections.get(currentRoom.getName()).get(i));
            }
            //Print all directions of room in correct order
            printCorrectOrderedDirection(orderedDirections);
            //Create an array variable to contain all item names
            Set<String> itemNames = items.keySet();
            String [] itemList = new String[itemNames.size()];
            itemList = itemNames.toArray(itemList);
            //For loop to print each item in the room.
            for(int j = 0; j < itemList.length; j++) {
                  if(currentRoom.hasItem(itemList[j])) {
                        System.out.println("There is " +
items.get(itemList[j]).getDesc() + " here.");
                  }
            }
            System.out.println();
      }
      /**
      * This method displays the user's inventory.
      * @param  inventory    user's inventory.
      * @param  items   collection of item names and the actual items.
      */
      public static void displayInventory(List<String> inventory,Map<String,Item>
items){
            System.out.println("You are currently carrying:");
            //Prints either the time or nothing if inventory is empty
            if(inventory.isEmpty()) {
                  System.out.println("   nothing");
            }
            else {
                  for(int i = 0; i < inventory.size(); i++) {
                        System.out.println("   " +
items.get(inventory.get(i)).getDesc());
                  }
            }
            System.out.println();
      }
      /**
      * This method fills inventory when user wants to get an item.
      * @param  inventory    user's inventory.
      * @param  currentRoom   room that user is currently in.
      * @param  items   collection of item names and the actual items.
```

```
	* @param  item    item that user wants to get.
	*/
	public static void fillInventory(List<String> inventory, Room currentRoom,
Map<String,Item> items, String item){
		//If-else statement to check if current Room has item
		if(currentRoom.hasItem(item)) {
			inventory.add(item);
			currentRoom.removeItem(item);
			System.out.println("You pick up " + items.get(item).getDesc() +
".");
		}
		else {
			System.out.println("There is no " + item + " here.");
		}
		System.out.println();
	}

	/**
	* This method changes the current Room.
	* @param  currentRoom   room that user is currently in.
	* @param  rooms   all the rooms in the game.
	* @param  direction    direction user wants to go to.
	*/
	public static Room changeRooms(Room currentRoom, Map<String,Room> rooms,
String direction){
		Room newRoom = currentRoom;
		//If-else statement to check if direction is possible
		if(currentRoom.hasExit(direction)) {
			newRoom = rooms.get(currentRoom.getExit(direction));
		}
		else {
			System.out.println("There is no exit in that direction.");
		}
		return newRoom;
	}

	/**
	* This method drops an item from inventory.
	* @param  inventory    user's inventory.
	* @param  currentRoom   room that user is currently in.
	* @param  items   collection of items and its description.
	* @param  item    item that user wants to drop.
	*/
	public static void dropInventory(List<String> inventory, Room currentRoom,
Map<String,Item> items, Item item){
		//If-else statement to check if user has item to remove
		if(inventory.contains(item.getName())) {
			inventory.remove(item.getName());
			currentRoom.addItem(item);
			System.out.println("You drop " +
items.get(item.getName()).getDesc() + ".");
		}
		else {
			System.out.println("Sorry, item to be dropped is not in
inventory");
		}
		System.out.println();
	}
```

```
	/**
	* This method reads the room file and sets up the rooms and room directions
	* @param  Rooms   map that contains room names and rooms
	* @param  roomDirections     contains all directions in each room.
	*/
	public static void readRoomFile (Map<String,Room> rooms,
Map<String,List<String>> roomDirections) {
		try {
			File roomFile = new File("rooms.txt");
		    Scanner roomReader = new Scanner(roomFile);
		    //Creates a list to contain all room attributes and current line
		List<String> roomAttributes = new LinkedList<String>();
		    //While loop to read each line in the room text file
		    while (roomReader.hasNextLine()) {
			String currentLine = roomReader.nextLine();
			//If else statement to either put room in map or add an attribute
to list.
			if(currentLine.equals("---")) {
				Room room = setRoom(roomAttributes,roomDirections);
				rooms.put(room.getName(), room);
				roomAttributes.clear();
			}
			else {
				roomAttributes.add(currentLine);
			}
		    }
		    roomReader.close();
		} catch(FileNotFoundException e) {
			System.out.println("An error reading file.");
		}
	}

	/**
	* This method reads the item file and sets up the items and rooms items
	* @param  Rooms   map that contains room names and rooms
	* @param  items   collection of item names and the actual items.
	*/
	public static void readItemFile (Map<String,Item> items, Map<String,Room>
rooms) {
		try {
			File itemFile = new File("items.txt");
		    Scanner itemReader = new Scanner(itemFile);
		    //Creates a list to contain all item attributes
		List<String> itemAttributes = new LinkedList<String>();
		//While loop to read each item in text file
		    while(itemReader.hasNextLine()) {
			String currentLine = itemReader.nextLine();
			//If else statement to either put item in room or add item
attributes.
			if(currentLine.equals("---")) {
				//Create an item variable to set name and description
				Item item = new SimpleItem();
				item.setName(itemAttributes.get(0).toLowerCase());
				item.setDesc(itemAttributes.get(1));
				items.put(item.getName().toLowerCase(), item);
				//Get room name item belongs to, then put item in
				//appropriate room
				String roomWithItem = itemAttributes.get(2);
				rooms.get(roomWithItem).addItem(item);
```

```
                        itemAttributes.clear();
                    }
                    else {
                        itemAttributes.add(currentLine);
                    }
                }
                itemReader.close();
            } catch(FileNotFoundException e) {
                System.out.println("An error reading file.");
            }
      }

      public static void main(String[] args)  {
            //Scanner variable
            Scanner scn = new Scanner(System.in);
            //Map for rooms and directions within room.
            //Set these variables up with a function.
            Map<String,Room> rooms = new HashMap<String,Room>();
            Map<String,List<String>> roomDirections = new
HashMap<String,List<String>>();
            readRoomFile(rooms,roomDirections);
            //Map of item names and items. Call function to fill up items
            Map<String,Item> items = new HashMap<String,Item>();
            readItemFile(items,rooms);
            //While loop to run until user quits. Variable to hold current room.
            String prompt = "test";
            List<String> inventory = new LinkedList<String>();
            Room currentRoom = rooms.get("Entry Room");
            while(!prompt.toLowerCase().equals("quit")) {
                  //If user wants to see their inventory
                  if(prompt.toLowerCase().equals("inventory")) {
                        displayInventory(inventory, items);
                  }
                  //If user wants to pick up an item.
                  else if(prompt.toLowerCase().substring(0,3).equals("get")) {
                        fillInventory(inventory, currentRoom, items,
prompt.toLowerCase().substring(4,prompt.length()));
                  }
                  //If user wants to go to another room
                  else if(prompt.toLowerCase().substring(0,2).equals("go")) {
                        currentRoom =
changeRooms(currentRoom,rooms,prompt.toLowerCase().substring(3,prompt.length()));
                  }
                  //If user wants to pick up an item.
                  else if(prompt.toLowerCase().substring(0,4).equals("drop")) {
                        dropInventory(inventory, currentRoom, items,
items.get(prompt.toLowerCase().substring(5,prompt.length())));
                  }
                  //Display Room
                  displayRoom(currentRoom, roomDirections,items);
                  //Check for user prompt
                  System.out.print("> ");
                  prompt = scn.nextLine();
                  //If statement for user to be able to quit.
                  if(prompt.toLowerCase().equals("quit")) {
                        System.out.println("Are you sure you want to quit?");
                        System.out.print("> ");
                        String answer = scn.nextLine();
                        if(!(answer.toLowerCase().equals("y"))) {
```

```
                        prompt = "test";
                    }
                }
            }
            scn.close();
        }
}
```