PPA assignment 2 report

Name of application: Ting’s World Trip to Paradise

* Short Description

You are in a game of Ting’s World Trip. You need to pay back the money that your parents borrowed. To win the game, pick up items and it will add up the total value and total evilness point for you at the end. If you pay back the borrowed money for you parents, you win the game! Below is the figure of the whole map.

Diagram

Description automatically generated

* What does this game do?

From the very start point of the game, the challenger will have his/her own bag to store the belongings with different money value and different evilness point from each country(room). The challenger can grab the belongings and accumulate all the value to pay back for the total amount of money that the challenger’s parents lend. The point of winning the game is to pay back the money successfully. Also, the evilness point accumulation is also being considered whether the challenger will win the game or not.

* What are the important implementation features?

This game is a two-dimension and word game. Also, this game can be considered as an easily understood game, so it can be played by people in any ages. For the programmer, it should be considered as a slightly difficult and logical project from programming beginners. This project includes much of significant concepts from java language. For instance, HashMap, array list, for loop, or if statement. Those concepts are applied directly in this project.

Basic task

* First, we need different countries for the game. I created the room in private void create countries as a new room. For example, “country’s name” = new room (“the instruction for the parameter”)
* As this task required the player to walk around each country, we need to set exits for each country(room). To set exits for each country(room), we need to use HashMap to store the direction of exit and where to go if the player exit. For example, US. set Exit (“east”, “Spain”)
* The items that can be hold will be using the command word “grab” to take the items. Some belongings in some room can’t be grabbed, so we set the belongings as not available with a true false statement.
* For the item that the player will carry, we need to set a maximum amount of weight that the player can carry. Use the if statement to set the maximum weight that the person can carry, else the person can’t handle it.
* To win the game, the player needs to get value of belongings higher than the amount money that will be set. This method will include using accumulator to add up the total value the evilness points that the player earns from the belongings. The parents (as a belonging in the last room) will be a comparator with a negative value. When the player picks up the parents, it will check whether the player wins the game or not
* To implement the going back to the previous room function, we need to first set the command word for back. Next, set the current room as previous room. In addition, it is also an if else statement. If previous room == null (there’s no previous room), system will print out there’s no room as the outcome, else current room= previous room.
* Four commands words will be first added into String command word, and we add methods below for each command word to have a clear description of the action.

Challenge task

* Attempt to do it, but the code is incorrect and didn’t work for many times.

Code Quality Considerations: coupling, cohesion, responsibility driven-design, maintainability

A high cohesion makes the method to be easier understood. For example, the private void grab method has an if else statement. Through the else part, there’s no one by one action through the method, but there’s much of use of the parameter, to each method to be continuous. Also, for coupling, the lower coupling will prevent the program to be crashed easily. Through my code source, for the avoid back section, if I change anything on this method, the “long Description” side will not be affected by fixing anything from the back section. As we can see, this can present the loose coupling in programming. In addition, the looser of coupling, the more the maintainability or stability of the function of the program. For the last point, the responsibility driven design represent one method should be responsible for or focus on one function only. To give an example, the “back” section is obvious to show that it only introduces one function for the game, and it’s the function of going back to the previous room for the player.

Known Bugs and Problems

* Weight

For the weight, the accumulated weight is still needed to be calculated. For private void grab section, the accumulation of weight, evilness point, and value don’t work because there’s a logical error for the insertion of get belongings.

* Commands

As we need to add four commands vocabularies, the “send” section cannot be doable. Since the original version is that the player will send the bread to a dog that will be created during the challenged task. However, the challenged tasks are worked, so the send function is not doable.

* Accumulation

For the accumulation of each value and winning the game by comparing cannot be calculated successfully through the game, so this game coding hasn’t fully completed all of the basic task, but attempted to finished.

A copy of source of all classes

public class Game

{

private Parser parser;

private Room currentRoom;

private Room previousRoom;

private int weight;

private int evilnessPoint;

private int value;

/\*\*

\* Create the game and initialise its internal map.

\*/

public Game()

{

createRooms();

parser = new Parser();

}

/\*\*

\* Create all the rooms and link their exits together.

\*/

private void createRooms()

{

Room US, Spain, France, Italy, Korea, Japan, Taiwan, homeUK;

// create the rooms

US = new Room("the start point of the game, you are in the US");

Spain = new Room("in Spain, discover anything here!");

France = new Room("in Paris, to get away from your parents");

Italy= new Room("in Italy, Rome!");

Korea = new Room("here to take your adventure in Korea from now on");

Japan = new Room("here. Konichiwa, Tokyo is waiting for you to take a look around ");

Taiwan= new Room("in Taiwan, and you are almost back to your home in the UK");

homeUK= new Room("in the UK. welcome back home! your parents are waiting for you");

// initialise room exits

US.setExit("east", Spain);

Spain.setExit("east", France);

Spain.setExit("west", US);

France.setExit("west", Spain);

France.setExit("east", Italy);

Italy.setExit("west", France);

Italy.setExit("northWest", Korea);

Korea.setExit("southEast",Italy );

Korea.setExit("north", Japan);

Japan.setExit("north", Taiwan);

Japan.setExit("south", Korea);

Taiwan.setExit("south", Japan);

Taiwan.setExit("north", homeUK);

homeUK.setExit("south", Taiwan);

currentRoom = US; // start game US

US.setBelongings("Tree", 0, 500, 0, false);

US.setBelongings("Lake", 0, 1000, 0, false);

Spain.setBelongings("LVWallet", 10000, 5, -1, true);

Spain.setBelongings("YourWallet", 5000,5 , 1,true);

Spain.setBelongings("destroyedWallet", 1000, 5, 2,true);

France.setBelongings("fancyWatch", 10000,5,-1,true);

France.setBelongings("yourWallet", 5000, 5,1,true);

France.setBelongings("destroyedWallet", 1000, 5, 2,true);

Italy.setBelongings("deliciousBread", 5000,2 , -1,true);

Italy.setBelongings("bread", 1000,2 , 1, true);

Italy.setBelongings("theWorstBreadYouHaveEverHad", 500, 2, 2,true);

Korea.setBelongings("bestEnergyDrink", 5000, 3, -1, true);

Korea.setBelongings("yourDrink", 1000, 3, 1, true);

Korea.setBelongings("water", 200, 3, 2, true);

Japan.setBelongings("fancyCamera", 50000, 5, -1, true);

Japan.setBelongings("yourOwnCamera", 10000, 5, 1, true);

Japan.setBelongings("fakeCamera", 1000, 5, 2, true);

Taiwan.setBelongings("deliciousBubbleTea", 5000, 3, -1,true);

Taiwan.setBelongings("anOkayBubbleTea", 2500, 3, 1, true);

Taiwan.setBelongings("Cola", 1000, 3, 2, true);

homeUK.setBelongings("yourParents", -28000, 125, -8, true);

}

/\*\*

\* Main play routine. Loops until end of play.

\*/

public void play()

{

printWelcome();

// Enter the main command loop. Here we repeatedly read commands and

// execute them until the game is over.

boolean finished = false;

while (! finished) {

Command command = parser.getCommand();

finished = processCommand(command);

}

System.out.println("Thank you for playing. Good bye.");

}

/\*\*

\* Print out the opening message for the player.

\*/

private void printWelcome()

{

System.out.println();

System.out.println("Welcome to the Ting's World Trip to Paradise!");

System.out.println("This game makes you earn money to pay back the money that your parents lended.");

System.out.println("In this game, you need to pick up enough value of things in oder to pay back the total amount of 6 million pounds.");

System.out.println("You will also earn the evilness point from grabbing things in each country.");

System.out.println("You might definitely enjoy this game while traveling around the world!");

System.out.println("Type 'help' if you need help.");

System.out.println();

System.out.println(currentRoom.getLongDescription());

}

/\*\*

\* Given a command, process (that is: execute) the command.

\* @param command The command to be processed.

\* @return true If the command ends the game, false otherwise.

\*/

private boolean processCommand(Command command)

{

boolean wantToQuit = false;

if(command.isUnknown()) {

System.out.println("I don't know what you mean...");

return false;

}

String commandWord = command.getCommandWord();

if (commandWord.equals("help")) {

printHelp();

}

else if (commandWord.equals("go")) {

goRoom(command);

}

else if (commandWord.equals("back")){

back(command);

}

else if (commandWord.equals("quit")) {

wantToQuit = quit(command);

}

else if (commandWord.equals("grab")){

grab(command);

}

else if (commandWord.equals("throwAway")){

throwAway(command);

}

else if (commandWord.equals("chat")){

chat(command);

}

else if (commandWord.equals("send")){

send(command);

}

// else command not recognised.

return wantToQuit;

}

// implementations of user commands:

/\*\*

\* Print out some help information.

\* Here we print some stupid, cryptic message and a list of the

\* command words.

\*/

private void printHelp()

{

System.out.println("You are lost. You are alone. You wander");

System.out.println("around everywhere in the world.");

System.out.println();

System.out.println("Your command words are:");

parser.showCommands();

}

/\*\*

\* Try to in to one direction. If there is an exit, enter the new

\* room, otherwise print an error message.

\*/

private void goRoom(Command command)

{

if(!command.hasSecondWord()) {

// if there is no second word, we don't know where to go...

System.out.println("Go where?");

return;

}

String direction = command.getSecondWord();

// Try to leave current room.

Room nextRoom = currentRoom.getExit(direction);

if (nextRoom == null) {

System.out.println("There is no door!");

}

else {

previousRoom= currentRoom;

currentRoom = nextRoom;

System.out.println(currentRoom.getLongDescription());

}

}

/\*\*

\* "Quit" was entered. Check the rest of the command to see

\* whether we really quit the game.

\* @return true, if this command quits the game, false otherwise.

\*/

private boolean quit(Command command)

{

if(command.hasSecondWord()) {

System.out.println("Quit what?");

return false;

}

else {

return true; // signal that we want to quit

}

}

/\*\*

\* "Back" was entered. It will lead the challenger to go back to the previous room

\*/

private void back(Command command)

{

String direction = command.getSecondWord();

// Try to leave current room.

if (previousRoom == null) {

System.out.println("There is no room!");

}

else {

currentRoom = previousRoom;

System.out.println(currentRoom.getLongDescription());

}

}

/\*\*

\* "grab" was entered. This is the section of how challenger take the belongings from current room.

\*/

private void grab(Command command)

{

if(!command.hasSecondWord()){

System.out.println("Grab what?");

}

else{

String belongingsName = command.getSecondWord();

currentRoom.removeBelonging(belongingsName);

weight = weight + getBelongings(belongingsName);

evilnessPoint= evilnessPoint + getBelongings(belongingsName);

value= value + getBelongings(belongingsName);

}

System.out.println (weight);

System.out.println (evilnessPoint);

System.out.println (value);

}

/\*\*

\* "throw"was entered. This is the section of how player throw away the belongings in current room.

\*/

private void throwAway(Command command)

{

String belongsName= command.getSecondWord();

if (!command.hasSecondWord()){

System.out.println("throwAway what?");

}

else{

String belongingName= command.getSecondWord();

currentRoom.removeBelonging(belongingName);

}

}

/\*\*

\* "throw"was entered. This is the section of how player throw away the belongings in current room.

\*/

private void chat(Command command)

{

System.out.println("Hello, How are you?");

System.out.println("How's your day?");

}

/\*\*

\* "send"was entered. This is the section of how player send the belongings to another character in current room.

\*/

private boolean send(Command command)

{

String belongsName= command.getSecondWord();

if (!command.hasSecondWord()){

System.out.println("send what?");

}

else{

String belongingName=command.getSecondWord();

currentRoom.getBelonging(belongingName);

}

}

}

public class Command

{

private String commandWord;

private String secondWord;

/\*\*

\* Create a command object. First and second word must be supplied, but

\* either one (or both) can be null.

\* @param firstWord The first word of the command. Null if the command

\* was not recognised.

\* @param secondWord The second word of the command.

\*/

public Command(String firstWord, String secondWord)

{

commandWord = firstWord;

this.secondWord = secondWord;

}

/\*\*

\* Return the command word (the first word) of this command. If the

\* command was not understood, the result is null.

\* @return The command word.

\*/

public String getCommandWord()

{

return commandWord;

}

/\*\*

\* @return The second word of this command. Returns null if there was no

\* second word.

\*/

public String getSecondWord()

{

return secondWord;

}

/\*\*

\* @return true if this command was not understood.

\*/

public boolean isUnknown()

{

return (commandWord == null);

}

/\*\*

\* @return true if the command has a second word.

\*/

public boolean hasSecondWord()

{

return (secondWord != null);

}

}

public class Room

{

private String description;

private HashMap<String, Room> exits;

private HashMap<String, Belongings> belongings;

// stores exits of this room.

/\*\*

\* Create a room described "description". Initially, it has

\* no exits. "description" is something like "a kitchen" or

\* "an open court yard".

\* @param description The room's description.

\*/

public Room(String description)

{

this.description = description;

exits = new HashMap<>();

belongings = new HashMap<String, Belongings>();

}

/\*\*

\* Define an exit from this room.

\* @param direction The direction of the exit.

\* @param neighbor The room to which the exit leads.

\*/

public void setExit(String direction, Room neighbor)

{

exits.put(direction, neighbor);

}

/\*\*

\* Define the belongings for each room.

\* @param name The name of the belonging

\* @param

\*/

public void setBelongings(String name, int value, int weight, int evilnessPoint, boolean availability)

{

belongings.put(name, new Belongings(name,value, weight, evilnessPoint,availability));

}

/\*\*

\* @return The short description of the room

\* (the one that was defined in the constructor).

\*/

public String getShortDescription()

{

return description;

}

/\*\*

\* Return a description of the room in the form:

\* You are in the kitchen.

\* Exits: north west

\* @return A long description of this room

\*/

public String getLongDescription()

{

return "You are " + description + ".\n" + getExitString()+ " " + getBelongingString();

}

/\*\*

\* Return a string describing the room's exits, for example

\* "Exits: north west".

\* @return Details of the room's exits.

\*/

private String getExitString()

{

String returnString = "Exits:";

Set<String> keys = exits.keySet();

for(String exit : keys) {

returnString += " " + exit;

}

return returnString;

}

/\*\*

\* Show the belongings in each room.

\*/

private String getBelongingString()

{

String returnString = "Belongings:";

Set<String>keys = belongings.keySet();

for (String belongings:keys){

returnString+= " "+ belongings;

}

return returnString;

}

/\*\*

\*

\*/

public void getBelonging(String belongingName)

{

belongings.get(belongingName).getWeight();

belongings.get(belongingName).getValue();

belongings.get(belongingName).getEvilnessPoint();

}

/\*\*

\* remove the belongings in each room.

\*/

public void removeBelonging(String belongingsName)

{

for (String aBelonging:belongings.keySet())

{

if (belongings.get(aBelonging).getName().equals(belongingsName)){

belongings.remove(belongingsName);

}else{

System.out.println("the belonging cannot be found!");

}

}

}

/\*\*

\* Return the room that is reached if we go from this room in direction

\* "direction". If there is no room in that direction, return null.

\* @param direction The exit's direction.

\* @return The room in the given direction.

\*/

public Room getExit(String direction)

{

return exits.get(direction);

}

}

public class Parser

{

private CommandWords commands; // holds all valid command words

private Scanner reader; // source of command input

/\*\*

\* Create a parser to read from the terminal window.

\*/

public Parser()

{

commands = new CommandWords();

reader = new Scanner(System.in);

}

/\*\*

\* @return The next command from the user.

\*/

public Command getCommand()

{

String inputLine; // will hold the full input line

String word1 = null;

String word2 = null;

System.out.print("> "); // print prompt

inputLine = reader.nextLine();

// Find up to two words on the line.

Scanner tokenizer = new Scanner(inputLine);

if(tokenizer.hasNext()) {

word1 = tokenizer.next(); // get first word

if(tokenizer.hasNext()) {

word2 = tokenizer.next(); // get second word

// note: we just ignore the rest of the input line.

}

}

// Now check whether this word is known. If so, create a command

// with it. If not, create a "null" command (for unknown command).

if(commands.isCommand(word1)) {

return new Command(word1, word2);

}

else {

return new Command(null, word2);

}

}

/\*\*

\* Print out a list of valid command words.

\*/

public void showCommands()

{

commands.showAll();

}

}

public class CommandWords

{

// a constant array that holds all valid command words

private static final String[] validCommands = {

"go", "quit", "help", "back", "grab", "sell", "attack", "restart", "send"

};

/\*\*

\* Constructor - initialise the command words.

\*/

public CommandWords()

{

// nothing to do at the moment...

}

/\*\*

\* Check whether a given String is a valid command word.

\* @return true if it is, false if it isn't.

\*/

public boolean isCommand(String aString)

{

for(int i = 0; i < validCommands.length; i++) {

if(validCommands[i].equals(aString))

return true;

}

// if we get here, the string was not found in the commands

return false;

}

/\*\*

\* Print all valid commands to System.out.

\*/

public void showAll()

{

for(String command: validCommands) {

System.out.print(command + " ");

}

System.out.println();

}

}

public class Belongings

{

// instance variables - replace the example below with your own

private String name;

private int value;

private int weight;

private int evilnessPoint;

private boolean availability;

/\*\*

\* Constructor for objects of class Belongings

\*/

public Belongings(String name, int value, int weight, int evfilnessPoint, boolean availability)

{

this.name= name;

this.value= value;

this.weight= weight;

this.evilnessPoint= evilnessPoint;

this.availability= availability;

}

/\*\*

\* This method implement the availabity of the belonging can be grab or not

\*/

private boolean availableBelongings()

{

return availability;

}

/\*\*

\* this method is the section of getting or choosing the belonging

\*/

public String getName()

{

return name;

}

/\*\*

\* this method is the section of getting the weight of belonging.

\*/

public int getWeight()

{

return weight;

}

/\*\*

\* this method is the section of getting the evilnessPoint of each belonging that a person picked.

\*/

public int getEvilnessPoint()

{

return evilnessPoint;

}

/\*\*

\* this method is the section of getting the value of the belonging.

\*/

public int getValue()

{

return value;

}

}