4CCS1PPA - Programming Practice and Applications

Assignment 2 – LONDON Game

Author: Hanna Hiridjee

Student No.: 1813287

Course Code: 4CCS1PPA

Due Date: 29.11.2019

TABLE OF CONTENTS

1. Introduction

For this assignment, I have decided to create a game about London called LONDON. The player arrives in London and has a mission. They must walk through London in search of the four hidden golden coins. The coins are hidden so the player must check every location for coins. Every time a coin is found, the player will receive a hint for the next one. Along the way, they encounter items that may or may not be picked up. In order to win, they must pass through at least 4 locations, collect at least 5 items (with a maximum cumulative weight of 25kg) and find the four golden coins.

2. Base Tasks

2.1 Locations

There are 9 rooms/locations in the game. Each one represents a different area/attraction in London. One of the rooms is the Magic Transporter room which I will discuss in the challenge tasks.
Each room consists of a description, multiple exits, multiple items and may or may not have a coin with a hint. An ArrayList keeps track of the items contained in this room. Throughout the game, the player can add or remove items from the rooms using the "take" and "drop" commands.
Rooms may or may not have coins. Each coin has a hint to lead the player to the next one.
Each room has at least one exit. The exits are stored in a HashMap

2.2 Items

For this task, I created a separate Item class. Each room has several items in it. The items have a name, weight and a Boolean value to represent if the item can be picked up by the player or not. In order for the player to pick up items, they must enter the command "take" followed by the item name. An item will only be picked up if it can be picked up (Boolean value set to true) and carrying it will not make the total weight carried exceed 25 kg. if an item cannot be carried, a message will be printed to tell the user why. If an item is picked up, it will be removed from the room and added to the player's bag. Players can also drop items along the way by entering the command "drop" followed by the item name. The item can be dropped in any room and will be added to the room the player is currently in.

2.3 Winning

The player wins the game if they have found the 4 coins, visited at least 4 rooms collected at least 5 items with a cumulative maximum weight not exceeding 25kg. To complete this task, I created a winCheck() method in the Game class. If the conditions are met, a message will be printed on the screen as well as a summary of the game. The game continuously checks if the player has won or not in the play() method of the Game class. Once the game is won, a message will be printed, and the game will quit automatically.

2.4 Back

If the player types in the "back" command, they will be taken back to the last room they were in. Instead of just storing the last room the player was in, I decided to make it a challenge task by using a

stack to keep track of every room visited. Each time the player enters a new room, the goRoom() method in Game class adds the room to the stack. When the player types in ”back”, the processCommands() method in Game class checks if the stack is empty. If it isn’t, the current room is changed to the element at the top of the stack. If it is empty, a message is printed out.

2.5 COMMANDS

I have added 5 new commands for the player to interact with the game.

“look” - allows the player to look around the location they are in. Calls the look() method in the Game class which prints out the location description, exits and items of the current room.

“see”- allows the player to get a reminder about what they have found so far.

> If it is followed by the word “bag”, the items in the player’s bag will be displayed by calling the showBag() method of the Player class.
>
> If it is followed by “hints”, all the hints the player has collected so far will be displayed by calling the showHints() method of the Player class.

“back”- allows the player to go back to the previous room they were in. A stack keeps track of all the rooms visited by the user so typing in “back” multiple times allows the player to go back to the first room. It also re-prints the description of the room. (See section 2.4)

“take”- allows the player to take items/coins.

> If followed by an item name, the name of each item in the current room is compared to the second word entered by the player. If the item is found, it calls the takeItem() method of the Player class. This method first checks if the item can be carried and if carrying it will not make the total weight carried exceed 25 kg. If these conditions are fulfilled, the item will be added to the list of items carried by the player (carriedItems), total weight carried is incremented and a message will be printed out. If the conditions are not met, an appropriate message will be displayed. Finally, if the item can be carried, it is removed from the room
>
> If followed by “coin”, allows the user to pick up a coin. If there is a coin in the current room, a message will be displayed, takeCoin() method of Player class will be called which returns the hint associated with this coin and the coin will be removed from the room. If there is no coin here, a message will be printed out.

“drop” - followed by the item name. Allows the player to drop item they are carrying in any room. The name of each item the player is carrying is compared to the second word entered by the player. If the item is found, it will be added to the current room and call the dropItem() method of the Player class which prints out a message, decrements the total weight the player is carrying and removes the item from the list of items carried by the player. If the item does not exist, a message will be printed out.

“check” – allows the user to check the room for coins. A message will be printed to tell the user if the room has a coin. The player can then decide to “take” it.

3.Challenge Tasks

3.1 Magic Transporter Room

The Magic Transporter Room is room which, when entered, transports the user to another (random) location in the game. In order to implement this, I did not set any exits or items for this room. Instead, I added all the other rooms to an ArrayList called rooms and created the magicTransporterRoom() method in the Game class. This method first generates a random number *between 0 and 8 (inclusive).* This random value is then used as an index to access one of the rooms in the rooms ArrayList. Finally, the value of current room changes to the random location. Whenever the user tries to enter a new room (goRoom() method in Game class), the goRoom() method in the Game class checks if the new room's description matches the Magic Transport room's description. If yes, a message is printed out announcing that the player will be transported to a new location. The magicTransporterRoom() method is called, the new room's description is printed out and the player's current room is updated to reflect the change.

3.2 Back

Instead of just storing the last room the player was in, I decided to make this task a challenge task by using a stack. (See section 2.4)

3.3 Coin

As a challenge task, I decided to create a Coin class. Only 4 locations contain a coin and all 4 of them must be found in order to win the game. Each coin has a hint that gives a clue about the location of the next coin. When a coin is taken, it is removed from the location.

4. Code Quality

4.1 Coupling

I considered coupling when creating the Item and Coin class, initially I wanted to put them in the Room class as rooms contain items and coins. However, by creating separate classes it is easier to make changes to the code. I also used encapsulation to make my code loosely coupled. For instance, the fields in the Player class are all private, however, I can still access these values from other classes by using accessor methods. For instance, the showBag() method which is called in the processCommands() method of the Game class. If I kept everything in the Game class, making changes to the Game would be difficult and my code would be hard to read.

4.2 Cohesion

In order to increase the cohesion, I decided to create a separate Item, Coin and Player class instead of keeping them in the Room or Game class. This way, each class is responsible for describing a single thing. For example, I created a separate Player class which is responsible for keeping track of the player's actions (items collected, weight, coins…). The Player class is composed of many small methods that each do a specific thing. Whenever the player enters a command, processCommands() in the Game class calls methods of the Player class. This makes my code more readable and re-useable. In addition, if the game is changed to allow for multiple players, we would just need to create multiple player objects in the Game class.

4.3 Responsibility-Driven design

I considered this when creating the Player class. Since it holds all the data related to the player (items carried, hints…) it should also be responsible for manipulating this data. This can be seen in the takeItem(), dropItem() etc… methods.

4.4 Maintainability

By creating separate classes and smaller methods, it is easier to modify/maintain the code.

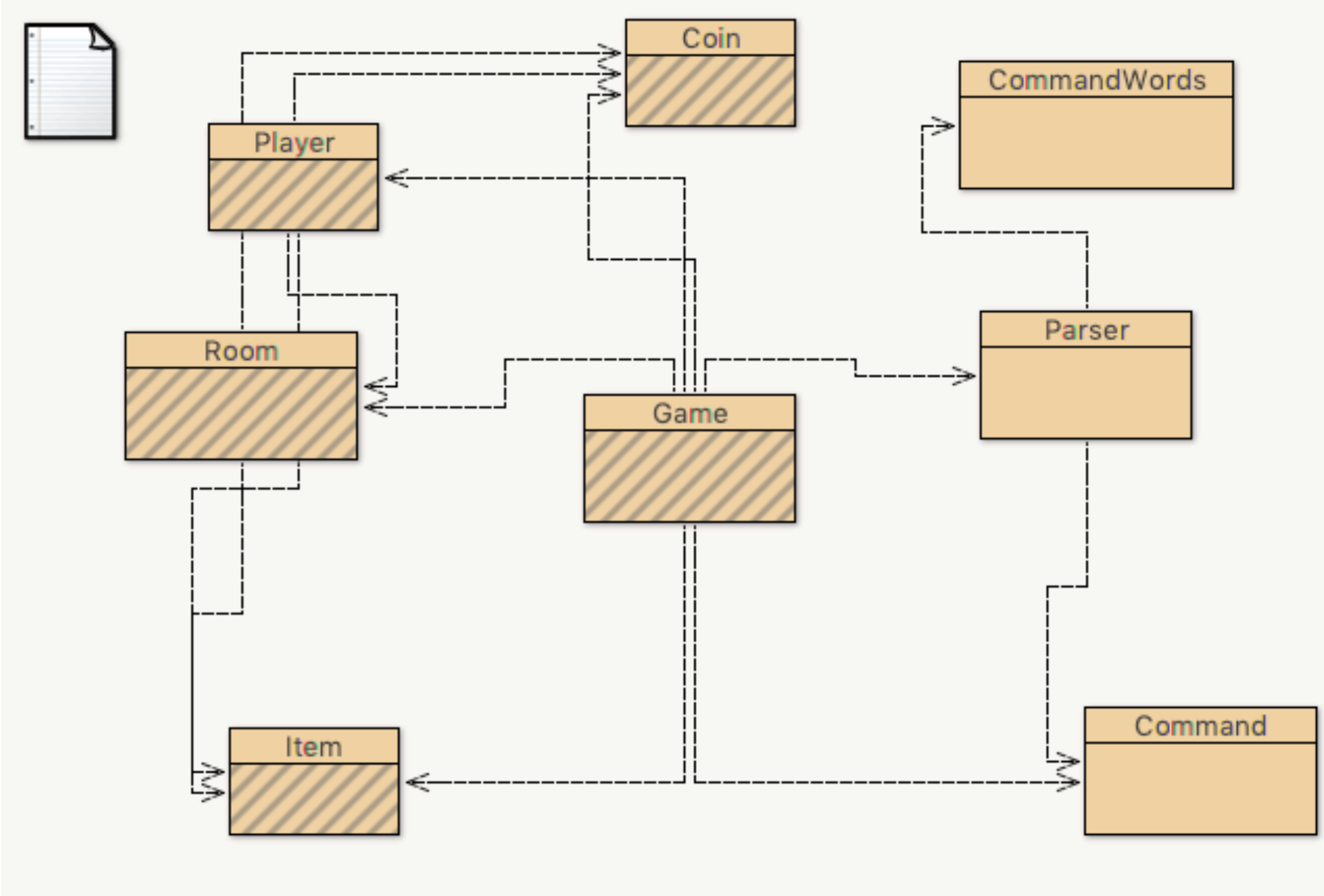
5. Walkthrough

To play the game, the player must first create an instance of the Game class and call the "play" method.

The terminal window will open and display the welcome message. Then, the description of the Mme Tussauds will be printed out as well as the exits and items it contains. The player must then enter one of the commands. Typing “take” followed by an item allows the user to pick up an item and adds it to the player’s bag (carried items). The user can then type in “see bag” which displays the contents of the bag. After taking an item, typing “look” re-prints the room’s description without the name of the item the player just took. To check the room for coins, the “check” command must be entered. If there is a coin there, the player can decide to take it by typing “take coin” which displays a hint to find the next coin. Typing in “see hints” will display the hints found by the player. To move to another room, the player types in “go” followed by the direction, if this exit exists, the new room’s description will be printed out. If the player then decides to drop an item, they must enter “drop” followed by the name of the item. Typing in “see” will display the contents of the bag without the newly dropped item and typing in “look” will display the description of the room and the dropped item will appear in the list of items of this room. If the player enters the Magic Transport room, they will be transported to a random room in the game. If the player types in “help” a message will be displayed reminding the user of their mission as well as the items in the bag, the total weight carried, the number of rooms visited, the number of coins found and their hints and the command words. If “quit” is entered, the game is terminated.

Using these commands, the player navigates through London and must fulfil the conditions required to win the game. Once the game is won, a message is displayed as well as a small summary of the game.

6.Class Sources


Coin
CommandWords
Player
Parser
Room
Game
Item
Command

```java
/**
 * This class is part of the "LONDON" game.
 * "LONDON" is a very simple, text based adventure game that takes the player around
London.
 *
 * This class represents a coin. Each coin has a hint that leads the player to the
next coin.
 *
 * @author Hanna Hiridjee (Student no. 1813287)
 * @version 29.11.2019
 */
public class Coin
{
    private String hint;    //Stores the hint for the next coin.
    /**
     * Constructor for objects of class Coin
     */
    public Coin(String hint)
    {
        this.hint = hint;
    }

    /**
     * Returns the hint of the next coin.
     * @return hint
     */
    public String getHint()
    {
        return hint;
    }
}
```

```java
/**
 * This class is part of the "LONDON" game.
 * "LONDON" is a very simple, text based adventure game that takes the player around London.
 *
 * This class represents an item. Each item has a name, weight and a boolean value to determine if it can be carried.
 *
 * @author Hanna Hiridjee (Student no. 1813287)
 * @version 29.11.2019
 */
public class Item
{
    private String name;
    private int weight;
    private boolean canBeCarried;

    /**
     * Constructor for objects of class Item.
     * Creates an item with a name, weight and whether it can be carried by the player.
     */
    public Item(String name, int weight, boolean canBeCarried)
    {
        this.name = name;
        this.weight = weight;
        this.canBeCarried = canBeCarried;
    }

    /**
     *  Returns the name of the item
     */
    public String getName(){
        return name;
    }

    /**
     * Returns the weight of the item
     */
    public int getWeight(){
        return weight;
    }

    /**
     * Returns true if the item can be picked up
     */
    public boolean getCanBeCarried(){
        return canBeCarried;
    }

}
```

```
import java.util.ArrayList;
/**
 * This class is part of the "LONDON" game.
 * "LONDON" is a very simple, text based adventure game that takes the player around
London.
 *
 * This class represents a player. Each player can carry items and hints as well as
keep track of the room, weight carried etc...
 * This class also has the methods that are called when the player performs an
action.
 *
 * @author Hanna Hiridjee (Student no. 1813287)
 * @version 29.11.2019
 */
public class Player
{
    private ArrayList<Item> carriedItems;       //stores the items carried by player
    private int weight;                         //total weight player is carrying
    private Room currentRoom;
    private int roomCount;                      //keeps track of number of rooms
visited
    private int coinCount;                      //keeps track of coins found
    private ArrayList<String> hints;            //Stores hints found
    /**
     * Constructor for objects of class Player
     * Creates a new player and initialises the items,hints and weight carried.
     */
    public Player()
    {
        carriedItems = new ArrayList<>();
        hints = new ArrayList<>();
        weight = 0;
        currentRoom = getCurrentRoom();
    }

    /**
     * Called when the user tries to take an item.
     * The total weight carried by the user cannot exceed 25kg.
     *
     * @param item the item to be taken
     */
    public void takeItem(Item item){
        if(item.getCanBeCarried() && ((weight += item.getWeight()) <= 25)){
            carriedItems.add(item);
            System.out.println("You have picked up " + item.getName());
        }
        else{
         System.out.println("You cannot pick up this item.");
         if(item.getCanBeCarried() && weight > 5){
             System.out.println("The item is too heavy or you have exceeded the
maximum weight you can carry.");
         }
         else if(item.getCanBeCarried() == false){
             System.out.println("A(n) " + item.getName()+" cannot be picked up.");
```

```
        }
    }
}

    /**
     * Called when player decides to drop an item they are carrying.
     *
     * @param item the item to be dropped in the current room.
     */
    public void dropItem(Item item){
        System.out.println("You have dropped the " + item.getName() + " item.");
        weight -= item.getWeight();
        carriedItems.remove(item);
    }

    /**
     * Sets the room the player is currently in.
     */
    public void setCurrentRoom(Room room){
        currentRoom = room;
    }

    /**
     * Returns the room the player is currently in.
     */
    public Room getCurrentRoom(){
        return currentRoom;
    }

    /**
     * Displays the items (and their weights) that the player is carrying.
     */
    public void showBag(){
        System.out.println("Items carried: " );
        for(Item item: carriedItems){
            System.out.println("- " + item.getName() + " "+ item.getWeight() +
"kgs.");
        }
    }

    /**
     * Returns the items carried by the player.
     * @return carriedItems an ArrayList of items the player is carrying.
     */
    public ArrayList<Item> showCarriedItems(){
        return carriedItems;
    }

    /**
     * Returns the total weight carried by the player.
     */
    public int getWeight(){
        return weight;
    }
```

```
    /**
     * Called when player takes a coin.
     * @return the coin's hint
     */public String takeCoin(Coin coin){
        coinCount++;
        hints.add(coin.getHint());
        return coin.getHint();
    }

    /**
     * Returns the number of coins the player has
     */
    public int getCoins(){
        return coinCount;
    }

    /**
     * Displays the hints the player has collected.
     */
    public void showHints(){
        System.out.println("Hints: ");
        for(String hint: hints){
            System.out.println("- " + hint);
        }
    }

}
```

```
import java.util.ArrayList;
import java.util.Random;
import java.util.Stack;
/**
 *  This class is the main class of the "LONDON" game.
 * "LONDON" is a very simple, text based adventure game that takes the player around
London.
 *
 *  To play this game, create an instance of this class and call the "play"
 *  method.
 *
 *  This main class creates and initialises all the others: it creates all
 *  rooms, creates the parser and player, and starts the game.  It also evaluates and
 *  executes the commands that the parser returns.
 *
 * @author  Michael Kölling and David J. Barnes
 * @edited Hanna Hiridjee (Student no. 1813287)
 * @version 29.11.2019
 */

public class Game
{
    private Parser parser;
    private Player player;
    private Room currentRoom;
    private ArrayList<Room> rooms; //stores the rooms
    private Room mmeTussauds, hydePark, oxfordSt, trafalgarSq, westminsterAbbey,
buckinghamPalace, scienceMuseum,bigBen, magicTransporter;
    private int roomCount; //keeps track of the number of rooms visited
    private Stack<Room> previousRooms; //Stack that keeps track of the rooms visited

    /**
     * Create the game and initialise its internal map.
     */
    public Game()
    {
        parser = new Parser();
        player = new Player();
        rooms = new ArrayList<Room>();
        previousRooms = new Stack<>();
        roomCount = 1;                  //first room visited is Mme Tussauds so
counter starts at 1
        createRooms();
    }

    /**
     * Creates all the rooms, links their exits together and adds items and coins to
the rooms.
     * Sets the current (starting) room.
     */
    private void createRooms()
    {
        // create the rooms
        mmeTussauds = new Room("You are now at Mme Tussauds");
```

```
hydePark = new Room("You are now at Hyde Park");
oxfordSt= new Room("You are now at Oxford Street");
trafalgarSq= new Room("You are now at Trafalgar Square");
westminsterAbbey= new Room("You are now at Westminster Abbey");
buckinghamPalace= new Room("You are now at Buckingham Palace");
scienceMuseum= new Room("You are now at the Science Museum");
bigBen= new Room("You are now at Big Ben");
magicTransporter= new Room("You are now in the Magic Transporter Room");

//Mme Tussauds
mmeTussauds.setExit("east",magicTransporter);
mmeTussauds.setExit("south", oxfordSt);

mmeTussauds.addItemToRoom(new Item("figurine1", 2, true));
mmeTussauds.addItemToRoom(new Item("statue", 50, false));

rooms.add(mmeTussauds);

//Hyde Park
hydePark.setExit("east",oxfordSt);
hydePark.setExit("south",scienceMuseum);

hydePark.addItemToRoom(new Item("picture1", 1, true));
hydePark.addItemToRoom(new Item("duck", 5, false));
hydePark.addItemToRoom(new Item("tree", 100, false));

rooms.add(hydePark);

hydePark.addCoin(new Coin("Tick tock..."));

//Oxford Street
oxfordSt.setExit("north",mmeTussauds);
oxfordSt.setExit("east",trafalgarSq);
oxfordSt.setExit("south",buckinghamPalace);
oxfordSt.setExit("west",hydePark);

oxfordSt.addItemToRoom(new Item("clothing", 3, true));
oxfordSt.addItemToRoom(new Item("pigeon1", 2, false));
oxfordSt.addItemToRoom(new Item("flag", 1, true));

rooms.add(oxfordSt);

oxfordSt.addCoin(new Coin("Fresh air..."));

//Trafalgar Square
trafalgarSq.setExit("north",magicTransporter);
trafalgarSq.setExit("south",westminsterAbbey);
trafalgarSq.setExit("west",oxfordSt);

trafalgarSq.addItemToRoom(new Item("pigeon2", 2, false));
trafalgarSq.addItemToRoom(new Item("keychain", 1, true));

rooms.add(trafalgarSq);
```

```
        //Science Museum
        scienceMuseum.setExit("east",buckinghamPalace);
        scienceMuseum.setExit("north",hydePark);

        scienceMuseum.addItemToRoom(new Item("sweater", 2, true));
        scienceMuseum.addItemToRoom(new Item("mug", 2, true));
        scienceMuseum.addItemToRoom(new Item("display", 100, false));

        rooms.add(scienceMuseum);

        scienceMuseum.addCoin(new Coin("Black Friday sales..."));

        //Buckingham Palace
        buckinghamPalace.setExit("east",westminsterAbbey);
        buckinghamPalace.setExit("north",oxfordSt);
        buckinghamPalace.setExit("west",scienceMuseum);

        buckinghamPalace.addItemToRoom(new Item("Queen", 70, false));
        buckinghamPalace.addItemToRoom(new Item("guard1", 80, false));
        buckinghamPalace.addItemToRoom(new Item("picture2", 1, true));

        rooms.add(buckinghamPalace);

        //Westminster Abbey
        westminsterAbbey.setExit("west",buckinghamPalace);
        westminsterAbbey.setExit("north",trafalgarSq);
        westminsterAbbey.setExit("south",bigBen);

        westminsterAbbey.addItemToRoom(new Item("police", 80, false));
        westminsterAbbey.addItemToRoom(new Item("picture3", 1, true));

        rooms.add(westminsterAbbey);

        //Big Ben
        bigBen.setExit("north",westminsterAbbey);

        bigBen.addItemToRoom(new Item("figurine2", 3, true));
        bigBen.addItemToRoom(new Item("picture4", 1, true));

        rooms.add(bigBen);

        bigBen.addCoin(new Coin("Let's explore space..."));


        currentRoom = mmeTussauds;  // start game at Mme Tussauds
    }

    /**
     *  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.
        // Also checks if the player has won the game yet.

        boolean finished = false;
        while (! finished) {
            Command command = parser.getCommand();
            finished = processCommand(command);
            if(winCheck()){
                finished = true;
            }
        }
        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 LONDON!");
        System.out.println("Here is your mission:");
        System.out.println("Go through various locations in London in search for the
4 golden coins. Everytime you find one, you will receive a hint for the next one.");
        System.out.println("Remember, coins are hidden so you must type in the
\"check\" command to look for them.");
        System.out.println("You must also pick up a minimum of 5 items along the way
and visit at least 4 locations. ");
        System.out.println("Don't forget, you can only carry 25kgs with you");
        System.out.println("Your commands are: " );
        parser.showCommands();
        System.out.println();
        System.out.println("Good luck!" );
        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();
```

```
        String secondWord = command.getSecondWord();
        if (commandWord.equals("help")) {
            printHelp();
        }
        else if (commandWord.equals("go")) {
            goRoom(command);
        }
        else if (commandWord.equals("quit")) {
            wantToQuit = quit(command);
        }
        else if(commandWord.equals("look")){
            look();
        }
        else if(commandWord.equals("take")){
            if(secondWord.equals("coin")){
                if(currentRoom.getCoin() != null){
                    System.out.println("You have picked up a coin. Here is a hint for
the next one:");
                    System.out.println(player.takeCoin(currentRoom.getCoin()));
                    currentRoom.removeCoin(currentRoom.getCoin());
                }
                else{
                    System.out.println("There is no coin");
                }
            }
            else{
            ArrayList<Item> items = currentRoom.showRoomItems();
            boolean found = false;
            for(int i = 0; i<items.size(); i++){
                if(items.get(i).getName().equals(secondWord)){
                    found = true;
                    player.takeItem(items.get(i));
                    if(items.get(i).getCanBeCarried()){ //Removes the item from the
room if it can be carried.
                        currentRoom.removeItem(items.get(i));
                    }
                }
            }
            if(!found){
                System.out.println("This item does not exist or is not available to
take.");
            }
          }
        }
        else if(commandWord.equals("drop")){
            for(int i =0; i<player.showCarriedItems().size(); i++){
                if(player.showCarriedItems().get(i).getName().equals(secondWord)){
//Finds which item the player is trying to drop.
                    currentRoom.addItemToRoom(player.showCarriedItems().get(i));
                    player.dropItem(player.showCarriedItems().get(i));
                }
                else
if(!(player.showCarriedItems().get(i).getName().equals(secondWord))){ //executed if
the item is not found
```

```
                System.out.println("This item does not exist.");
            }
        }
    }
    else if(commandWord.equals("see")){
        if(secondWord.equals("bag")){
            player.showBag();
        }
        else if(secondWord.equals("hints")){
            player.showHints();
        }
    }
    else if(commandWord.equals("back")){
        if(!previousRooms.empty()){
            currentRoom = previousRooms.pop();
            System.out.println(currentRoom.getLongDescription());
        }
        else
        {
            System.out.println("You cannot go back anymore!");
        }
    }
    else if(commandWord.equals("check")){
        if(currentRoom.getCoin()!=null){
            System.out.println("This location contains a coin.");
        }
        else{
            System.out.println("There are no coins here.");
        }
    }

    // else command not recognised.
    return wantToQuit;
}

/**
 * Called when the player enters the Magic Transport Room.
 * It transports the player to a random location in the game.
 */
private void magicTransporterRoom(){
    Random random = new Random();
    currentRoom = rooms.get(random.nextInt(rooms.size()));
}

/**
 * Checks if the player has won the game.
 * Prints out a win message and gives a game summary if the player won the game.
 * @return true if the game is won.
 */
private boolean winCheck(){
    boolean win = false;
    if((player.showCarriedItems().size() >= 5 ) && (roomCount > 4) &&
(player.getCoins() == 4)){
        win = true;
```

```
            System.out.println("\n" + "YOU HAVE WON THE GAME!!!" +"\n");
            System.out.println("GAME SUMMARY: ");

            System.out.println("Number of rooms visited: " + roomCount);
            System.out.println("Weight carried: " + player.getWeight() + " kgs");
            player.showBag();
            System.out.println("Coins found: " + player.getCoins());
            player.showHints();
        }
        return win;
    }

    // implementations of user commands:

    /**
     * Print out help information.
     * Including the mission and a summary of the game so far.
     * Also re-prints the commands.
     */
    private void printHelp()
    {
        System.out.println("Your mission: Visit the attractions around London and
pick up items along the way.");
        System.out.println("You must pick up a minimum of 5 items along the way and
visit at least 4 locations. ");
        System.out.println("Don't forget, you can only carry 25kgs with you");
        System.out.println();
        player.showBag();
        System.out.println();
        System.out.println("Total weight carried: " + player.getWeight() +" kgs.");
        System.out.println("Number of rooms visited: " + roomCount);
        System.out.println("Coins picked up: " + player.getCoins());
        player.showHints();
        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.
     * If the room is the Magic Transporter Room, the player will be transported to a
random room.
     *
     * @param command The command to be processed.
     */
    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);
        previousRooms.push(currentRoom);

        if (nextRoom == null) {
            System.out.println("There is no door!");
        }
        //Magic Transporter Room - Sends user to a random room.
        else if(magicTransporter.isEqual(nextRoom)){
            System.out.println();
            System.out.println("You are now in the MAGIC TRANSPORTER ROOM." + "\n"
+"You will be transported to a random location!");
            magicTransporterRoom();
            System.out.println();
            System.out.println(currentRoom.getLongDescription());
            player.setCurrentRoom(currentRoom);
            roomCount++;
        }
        else{
            currentRoom = nextRoom;
            player.setCurrentRoom(currentRoom);
            System.out.println(currentRoom.getLongDescription());
            roomCount++;
        }
    }

    /**
     * "Quit" was entered. Check the rest of the command to see
     * whether we really quit the game.
     *
     * @param command The command to be processed.
     * @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
        }
    }

    /**
     * Called when the player types in "look".
     * Allows the player to look around.
     * Gives info about the description of the room, the exits and the items it
contains.
     */
    private void look()
    {
```

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


}
```

```java
/**
 * This class is part of the "LONDON" game.
 * "LONDON" is a very simple, text based adventure game that takes the player around
London.
 *
 * This class holds information about a command that was issued by the user.
 * A command currently consists of two strings: a command word and a second
 * word (for example, if the command was "take map", then the two strings
 * obviously are "take" and "map").
 *
 * The way this is used is: Commands are already checked for being valid
 * command words. If the user entered an invalid command (a word that is not
 * known) then the command word is <null>.
 *
 * If the command had only one word, then the second word is <null>.
 *
 * @author  Michael Kölling and David J. Barnes
 * @edited Hanna Hiridjee (Student no. 1813287)
 * @version 29.11.2019
 */

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);
    }
}
```

```java
/**
 * This class is part of the "LONDON" game.
 * "LONDON" is a very simple, text based adventure game that takes the player around
London.
 *
 * This class holds an enumeration of all command words known to the game.
 * It is used to recognise commands as they are typed in.
 *
 * @author  Michael Kölling and David J. Barnes
 * @edited Hanna Hiridjee (Student no. 1813287)
 * @version 29.11.2019
 */

public class CommandWords
{
    // a constant array that holds all valid command words
    private static final String[] validCommands = {
        "go", "quit", "help", "look","take","drop","see", "back","check"
    };

    /**
     * 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();
    }
}
```

```
import java.util.Set;
import java.util.HashMap;
import java.util.ArrayList;
/**
 * Class Room - a location in "LONDON".
 *
 * This class is part of the "LONDON" game.
 * "LONDON" is a very simple, text based adventure game that takes the player around
London.
 *
 * A "Room" represents one location in the scenery of the game.  It is
 * connected to other rooms via exits.  For each existing exit, the room
 * stores a reference to the neighboring room. Rooms also contain items and coins.
 *
 * @author  Michael Kölling and David J. Barnes
 * @edited Hanna Hiridjee (Student no. 1813287)
 * @version 29.11.2019
 */

public class Room
{
    private String description;
    private HashMap<String, Room> exits;        // stores exits of this room.
    private ArrayList<Item> roomItems;          //stores the items of this current
room

    private String hint;
    private Coin coin;
    /**
     * Create a room described "description". Initially, it has
     * no exits
     * @param description The room's description.
     */
    public Room(String description)
    {
        this.description = description;
        exits = new HashMap<>();
        roomItems = new ArrayList<Item>();
    }

    /**
     * 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);
    }

    /**
     * @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 Hyde Park.
 *     Exits: east south
 *     Items: picture, 1 kg. duck, 5 kg. tree, 100 kg.
 * @return A long description of this room
 */
public String getLongDescription()
{
    return  description + ".\n" + getExitString()
            + ".\n" + getRoomItems() ;
}

/**
 * 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;
}

/**
 * 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);
}

/**
 * Adds an item to the room and an ArrayList containing the rooms items.
 * @param item the item to be added to the room
 */
public void addItemToRoom(Item item){
    roomItems.add(item);
}

/**
 * Removes an item from the room and the ArrayList containing the rooms items.
```

```
     * @param  item the item to be removed from the room
     */
    public void removeItem(Item item){
        roomItems.remove(item);
    }

    /**
     * Displays items in this room.
     * @return a string with the items of the room and their weights.
     */
    public String getRoomItems(){
        String str = "Items: ";
        for(int i = 0; i<roomItems.size();i++){
            str += roomItems.get(i).getName() + ", " + roomItems.get(i).getWeight()
            + " kg. ";
        }
        return str;
    }

    /**
     * Returns the description of the room.
     */
    public String getDescription(){
        return description;
    }

    /**
     * Checks if the room the player is in is the Magic Transport Room.
     *
     * @param room  the room the player is currently in.
     * @return true if the room is the Magic Transport Room.
     */
    public boolean isEqual(Room room){
        if(room.getDescription() == "You are now in the Magic Transporter Room"){
            return true;
        }
        return false;
    }

    /**
     * Shows items in the room.
     * @return the ArrayList containg the room's items.
     */
    public ArrayList<Item> showRoomItems(){
            return roomItems;
    }

    /**
     * Adds a coin to a room
     * @param coin the coin to be added to the room
     */
    public void addCoin(Coin coin){
        this.coin = coin;
        this.hint = coin.getHint();
```

```
    }

    /**
     * Removes a coin from a room
     * @param coin the coin to be added to the room
     */
    public void removeCoin(Coin coin){
        this.coin = null;
        hint = null;
    }

    /**
     * Returns the coin.
     * @return coin if there is a coin in this room. Otherwise, returns null
     */public Coin getCoin(){
        if(coin != null){
            return coin;
        }else{
            return null;
        }
    }

}
```

```
import java.util.Scanner;

/**
 *  This class is part of the "LONDON" game.
 * "LONDON" is a very simple, text based adventure game that takes the player around
London.
 *
 * This parser reads user input and tries to interpret it as an "Adventure"
 * command. Every time it is called it reads a line from the terminal and
 * tries to interpret the line as a two word command. It returns the command
 * as an object of class Command.
 *
 * The parser has a set of known command words. It checks user input against
 * the known commands, and if the input is not one of the known commands, it
 * returns a command object that is marked as an unknown command.
 *
 * @author  Michael Kölling and David J. Barnes
 * @edited Hanna Hiridjee (Student no. 1813287)
 * @version 29.11.2019
 */
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();
    }
}
```