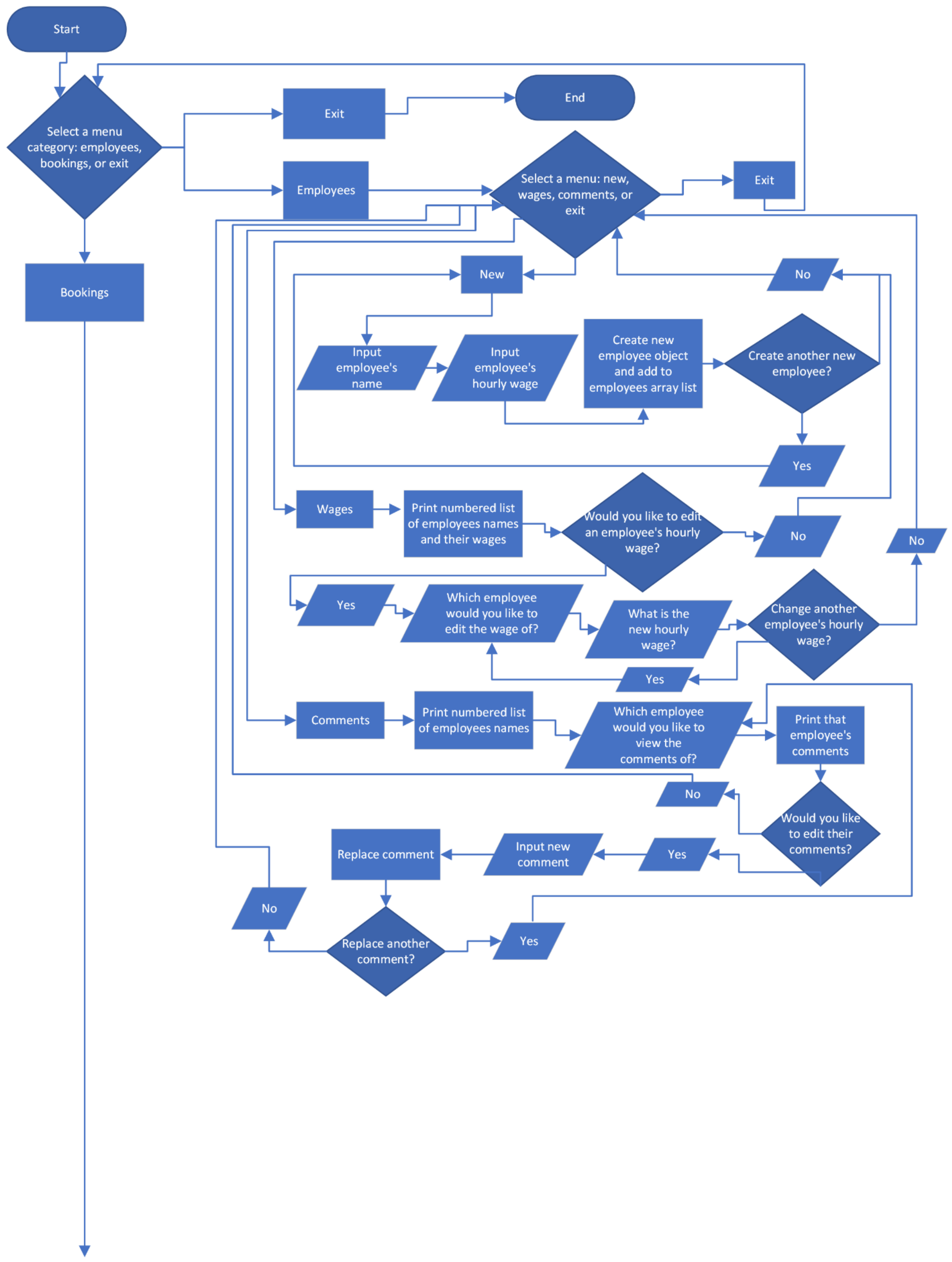
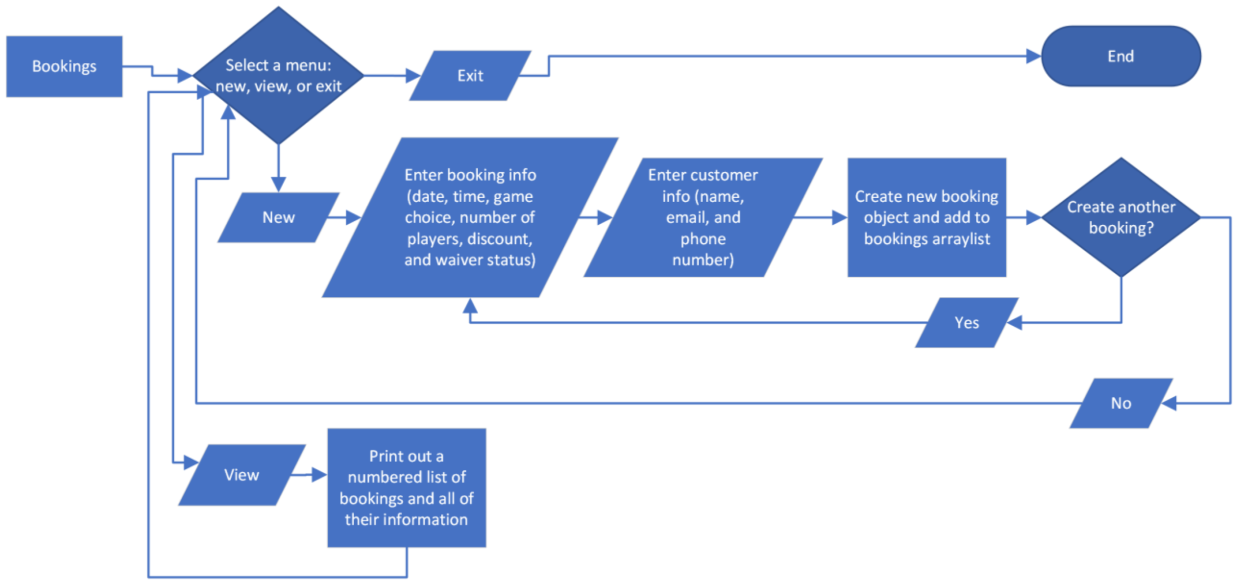
C: Development

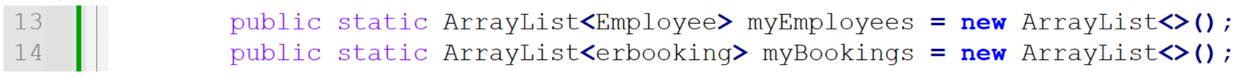
Flowchart


Start
Select a menu category: employees, bookings, or exit
Exit
End
Employees
Select a menu: new, wages, comments, or exit
Exit
Bookings
New
No
Input employee's name
Input employee's hourly wage
Create new employee object and add to employees array list
Create another new employee?
Yes
Wages
Print numbered list of employees names and their wages
Would you like to edit an employee's hourly wage?
No
No
Yes
Which employee would you like to edit the wage of?
What is the new hourly wage?
Change another employee's hourly wage?
Yes
Comments
Print numbered list of employees names
Which employee would you like to view the comments of?
Print that employee's comments
No
Would you like to edit their comments?
Replace comment
Input new comment
Yes
No
Replace another comment?
Yes


Bookings
Select a menu: new, view, or exit
Exit
End
New
Enter booking info (date, time, game choice, number of players, discount, and waiver status)
Enter customer info (name, email, and phone number)
Create new booking object and add to bookings arraylist
Create another booking?
Yes
No
View
Print out a numbered list of bookings and all of their information

Complex Code

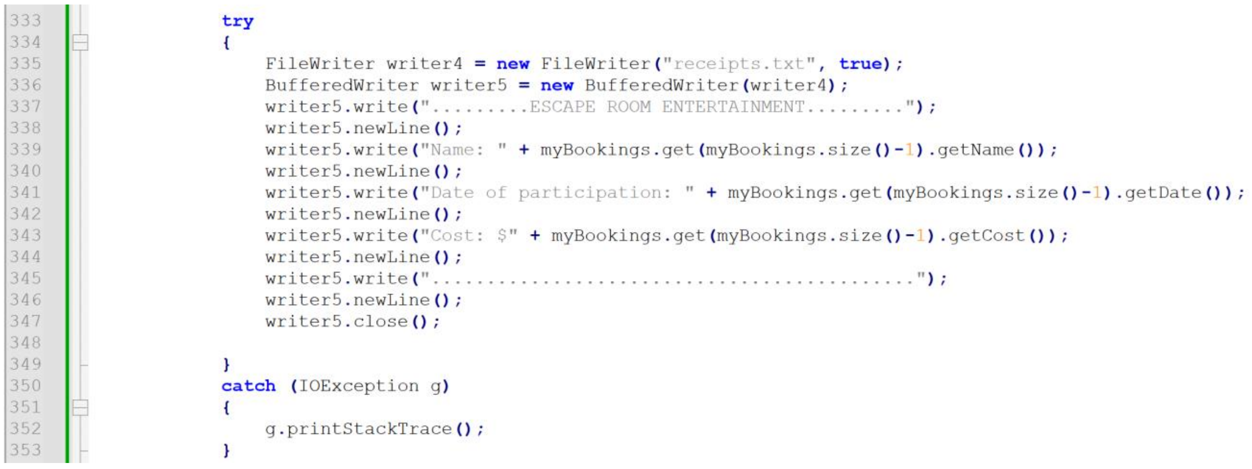
A. Array Lists



```
13          public static ArrayList<Employee> myEmployees = new ArrayList<>();
14          public static ArrayList<erbooking> myBookings = new ArrayList<>();
```

Lines 13 and 14 from the EscapeRoom class shows the declaration of two array lists of bookings and employees. These array lists are used to store all the employees or bookings that the user creates with the program. The array lists allow for easy access to all of the employees or bookings and make it easier to sort the bookings by time of participation. Each position in these arrays can allow the program to access all the attributes of the object through the object classes.

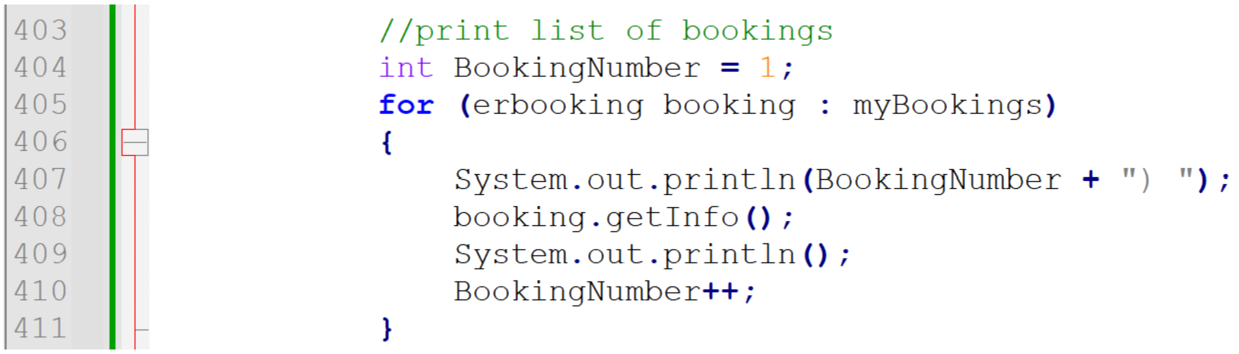
B. File IO



```
333             try
334             {
335                 FileWriter writer4 = new FileWriter("receipts.txt", true);
336                 BufferedWriter writer5 = new BufferedWriter(writer4);
337                 writer5.write("........ESCAPE ROOM ENTERTAINMENT........");
338                 writer5.newLine();
339                 writer5.write("Name: " + myBookings.get(myBookings.size()-1).getName());
340                 writer5.newLine();
341                 writer5.write("Date of participation: " + myBookings.get(myBookings.size()-1).getDate());
342                 writer5.newLine();
343                 writer5.write("Cost: $" + myBookings.get(myBookings.size()-1).getCost());
344                 writer5.newLine();
345                 writer5.write("...........................................");
346                 writer5.newLine();
347                 writer5.close();
348 
349             }
350             catch (IOException g)
351             {
352                 g.printStackTrace();
353             }
```

Lines 333 through 453 display the use of file io to write a receipt in a text file. A buffered writer object is used to write to this document. The write method writes what is passed to the document, the new line method is used to go to the next line in the document, and the close method is used when the writing to the document is complete. A catch exception is used along with file io in standard practice.

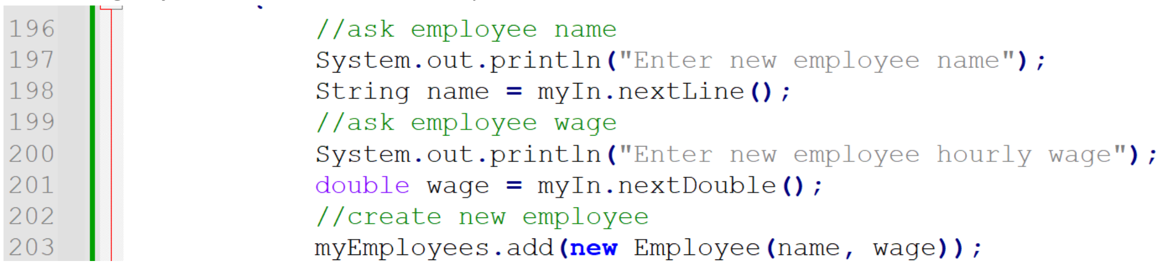
C. For each loop



```
//print list of bookings
int BookingNumber = 1;
for (erbooking booking : myBookings)
{
    System.out.println(BookingNumber + ")  ");
    booking.getInfo();
    System.out.println();
    BookingNumber++;
}
```

This for each loop is used to print out the bookings and all information regarding the booking with a number. The for each loop parses through the myBookings array list and, for each of the bookings, labels them as booking and prints the booking information out. This shows how the array list allows for easy access to each booking added by the user.

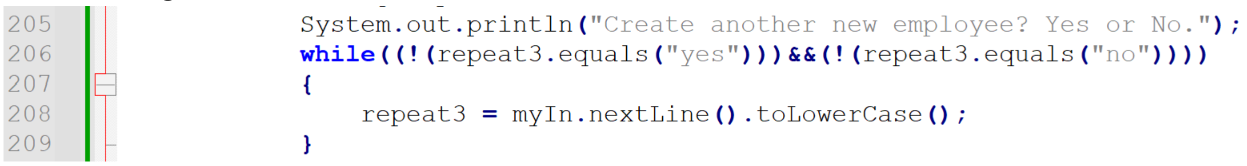
D. Initializing object instances with user input



```
//ask employee name
System.out.println("Enter new employee name");
String name = myIn.nextLine();
//ask employee wage
System.out.println("Enter new employee hourly wage");
double wage = myIn.nextDouble();
//create new employee
myEmployees.add(new Employee(name, wage));
```

By initializing the employees in the array list, this allows them to be accessed again. It would be impossible to assign a name to these employee objects as this initialization line is called multiple times and multiple objects cannot be initialized with the same name. Using the add method for the array list to create a new employee avoids this issue.

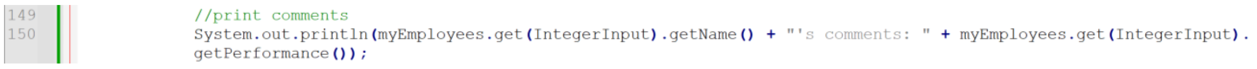
E. Error checking



```
System.out.println("Create another new employee? Yes or No.");
while((!(repeat3.equals("yes")))&&(!(repeat3.equals("no"))))
{
    repeat3 = myIn.nextLine().toLowerCase();
}
```

This while loop is a form of error checking. As long as the input value is not yes or no, it will keep asking for an input that matches either yes or no. This provides error checking if the user makes a mistake inputting their response. The response is also automatically translated to all lowercase so a user can input yes or no with or without capital letters without affecting the outcome.

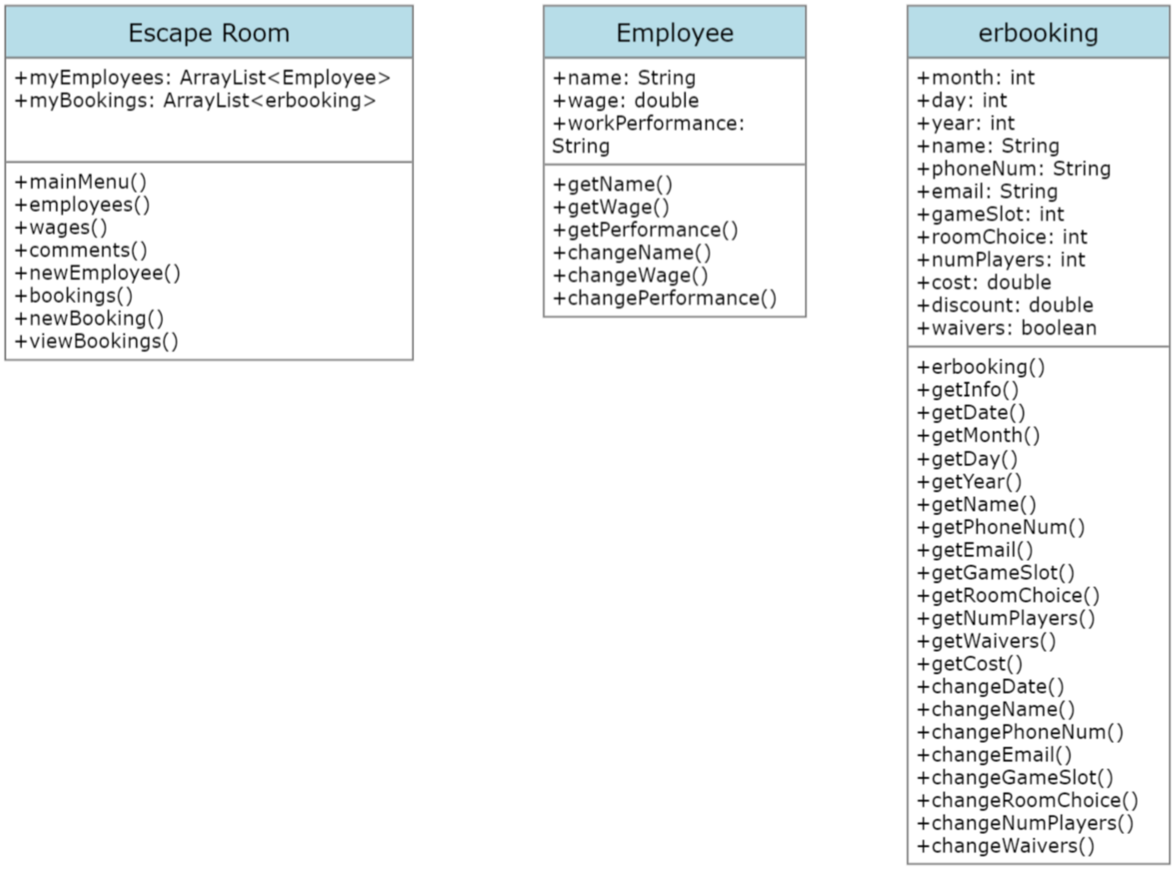
F. Calling methods in other classes



```
//print comments
System.out.println(myEmployees.get(IntegerInput).getName() + "'s comments: " + myEmployees.get(IntegerInput).
getPerformance());
```

This line shows an combination of complex coding techniques. First, the employee that the user has chosen has been assigned an index in the array list that is stored in the index integer input. The employee is retrieved from the array list with the get method and this integer input as the index. Then, a method called get name in the employee class is called to retrieve the name of the employee selected. The same is done with the get performance method to retrieve the comments that the user left under the selected employee. This information is then printed out in the command prompt.

UML Diagram


Escape Room
+myEmployees: ArrayList<Employee>
+myBookings: ArrayList<erbooking>
+mainMenu()
+employees()
+wages()
+comments()
+newEmployee()
+bookings()
+newBooking()
+viewBookings()
Employee
+name: String
+wage: double
+workPerformance: String
+getName()
+getWage()
+getPerformance()
+changeName()
+changeWage()
+changePerformance()
erbooking
+month: int
+day: int
+year: int
+name: String
+phoneNum: String
+email: String
+gameSlot: int
+roomChoice: int
+numPlayers: int
+cost: double
+discount: double
+waivers: boolean
+erbooking()
+getInfo()
+getDate()
+getMonth()
+getDay()
+getYear()
+getName()
+getPhoneNum()
+getEmail()
+getGameSlot()
+getRoomChoice()
+getNumPlayers()
+getWaivers()
+getCost()
+changeDate()
+changeName()
+changePhoneNum()
+changeEmail()
+changeGameSlot()
+changeRoomChoice()
+changeNumPlayers()
+changeWaivers()