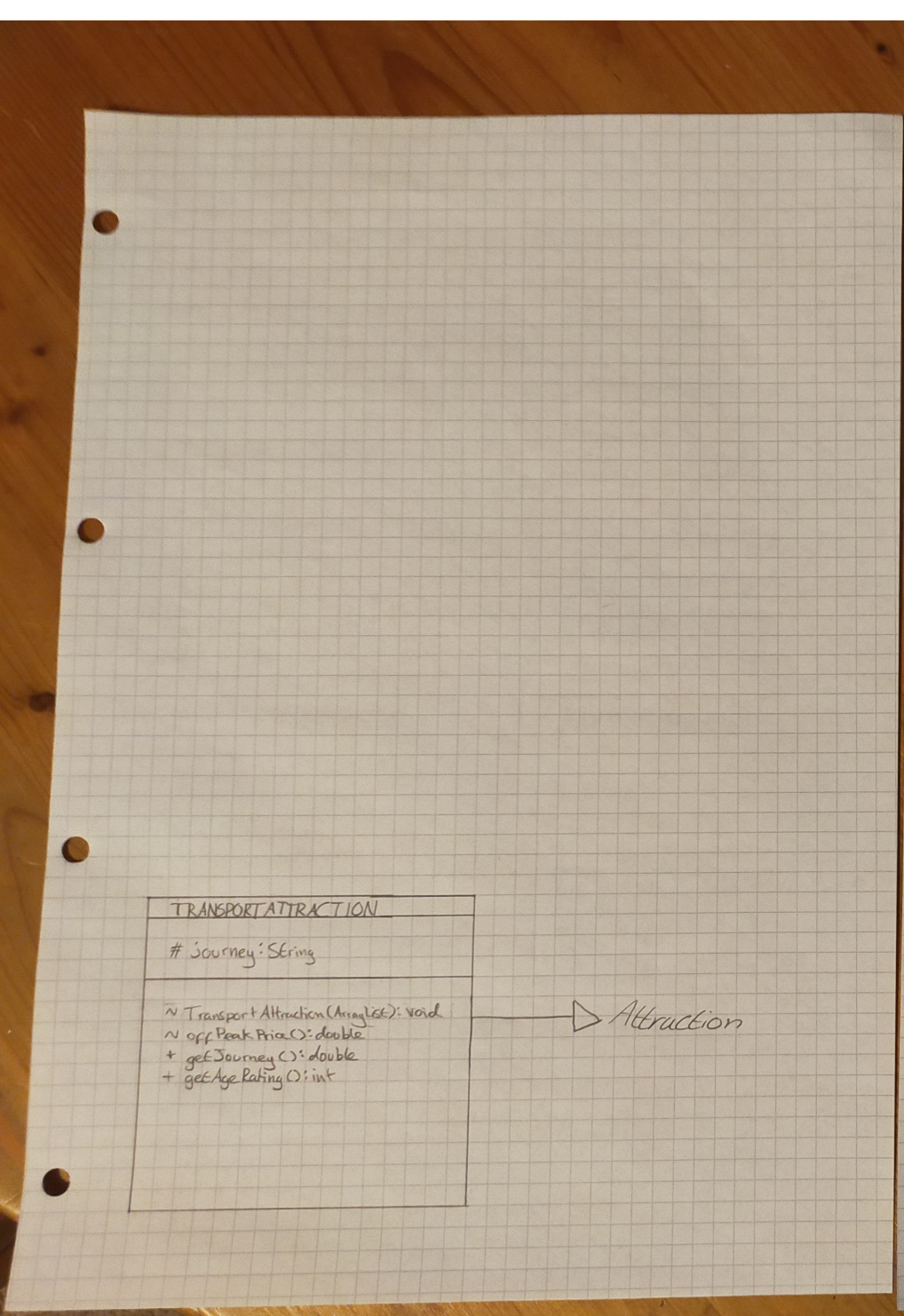
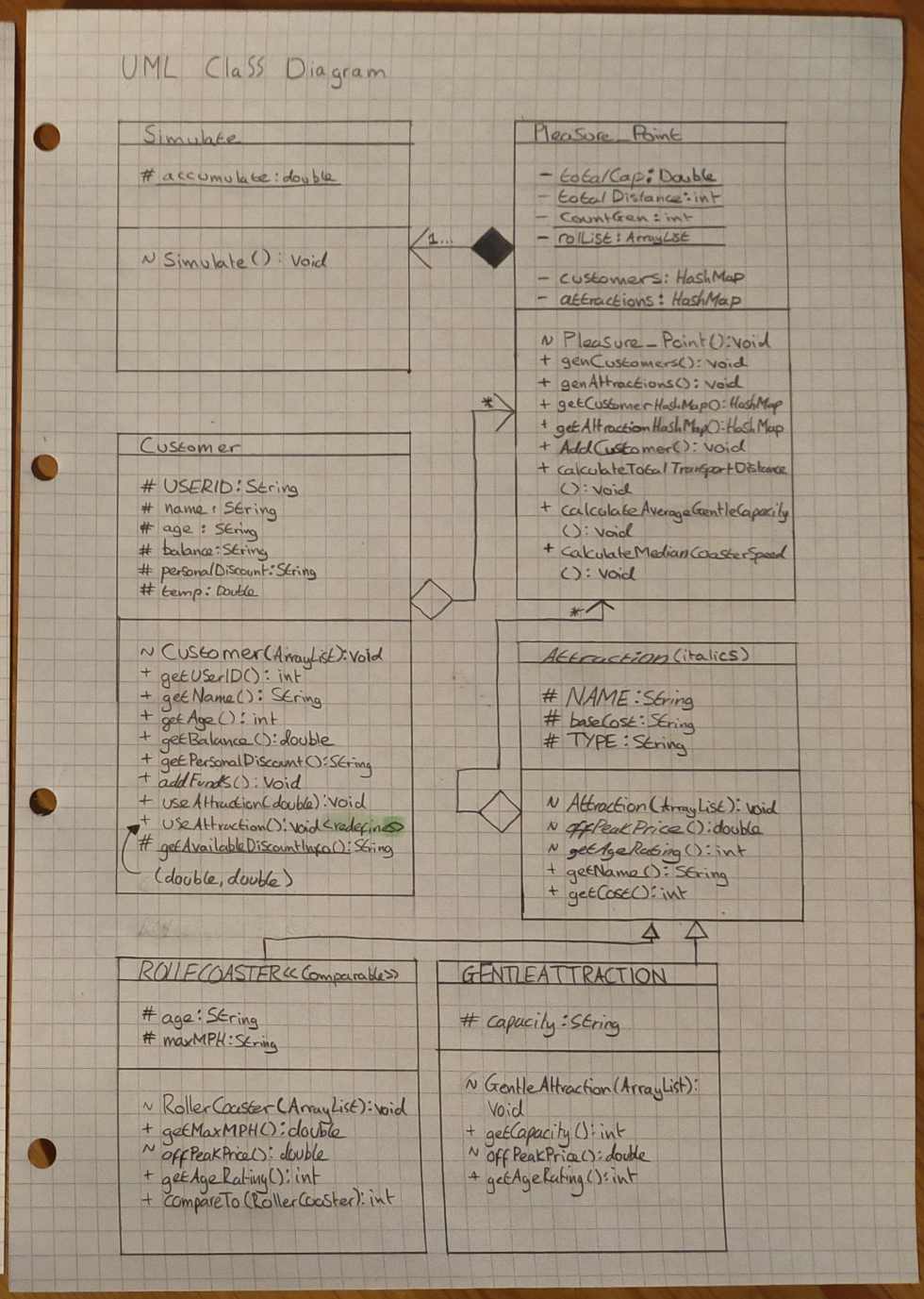
Programming 1 - Coursework Assignment 2

Student number: 100279967

Wednesday 8th April, 2020 17:14


TRANSPORTATTRACTION
Journey: String
~ TransportAttraction(ArrayList): void
~ offPeakPrice(): double
+ getJourney(): double
+ getAgeRating(): int
Attraction


UML Class Diagram
Simulate
accumulate: double
~ Simulate(): void
1...
MeaSure_Point
- totalCap: Double
- total Distance: int
- countGen: int
- rollList: ArrayList
- customers: HashMap
- attractions: HashMap
~ PleaSure_Point(): void
+ genCustomers(): void
+ genAttractions(): void
+ getCustomerHashMap(): HashMap
+ getAttractionHashMap(): HashMap
+ AddCustomer(): void
+ calculateTotalTransportDistance(): void
+ calculateAverageGentleCapacity(): void
+ calculateMedianCoasterSpeed(): void
*
Customer
USERID: String
name: String
age: String
balance: String
personalDiscount: String
temp: Double
~ Customer(ArrayList): void
+ getUserID(): int
+ getName(): String
+ getAge(): int
+ getBalance(): double
+ getPersonalDiscount(): String
+ addFunds(): void
+ useAttraction(double): void
+ useAttraction(): void <redefines>
# getAvailableDiscountHypo(): String
(double, double)
*
Attraction (italics)
NAME: String
baseCost: String
TYPE: String
~ Attraction(ArrayList): void
~ offPeakPrice(): double
~ getAgeRating(): int
+ getName(): String
+ getCost(): int
ROLLERCOASTER <<Comparable>>
age: String
maxMPH: String
~ RollerCoaster(ArrayList): void
+ getMaxMPH(): double
~ offPeakPrice(): double
+ getAgeRating(): int
+ compareTo(RollerCoaster): int
GENTLEATTRACTION
capacity: String
~ GentleAttraction(ArrayList): Void
+ getCapacity(): int
~ offPeakPrice(): double
+ getAgeRating(): int

Listing 1: Attraction.java

```
1  package com.Greedy;
2
3  // Unused imports are required for test harness.
4  import java.util.ArrayList;
5
6  abstract class Attraction {
7
8      // Final as we don't want the name to ever change (HashMap depends on name as
       ↪ primary key).
9      protected final String name;
10     protected String baseCost;
11     protected final String type;
12
13     Attraction(ArrayList<String> line) {
14         name = line.get(0);
15         baseCost = line.get(1);
16         type = line.get(2);
17     }
18
19     //Defining abstract methods
20     abstract double offPeakPrice();
21     abstract int getAgeRating();
22
23     // Accessor that retrieves name of any attraction. Not used (yet), but maybe
       ↪ useful later on.
24     public String getName() { return name; }
25
26     // parse cost as double here as this is optimal (runtime).
27     public int getCost() { return Integer.parseInt(baseCost); }
28
29     public String toString() { return name + "@" + baseCost + "@" + type; }
30
31     /* TEST HARNESS
32     public static void main(String[] args) throws FileNotFoundException,
       ↪ InsufficientBalanceException, AgeRestrictionException {
33         String x = "name@250@type";
34         ArrayList<String> y = new ArrayList<>(Arrays.asList(x.split("@")));
35         Attraction test = new Attraction(y) {
36                 @Override
37                 double offPeakPrice() { return 0; }
38
39                 @Override
40                 public int getAgeRating() { return 0; }
41             };
42         // TESTS
43         System.out.println(test);
44         System.out.println(test.getCost());
45         System.out.println(test.getName());
46         System.out.println(test.getAgeRating());
47         System.out.println(test.offPeakPrice());
48     }
49      */
50 }
```

Listing 2: TransportAttraction.java

```java
package com.Greedy;

import java.util.ArrayList;

// Subclass of baseclass (Attraction) specific too transportation attractions.
final class TransportAttraction extends Attraction {

    protected String journey;

    TransportAttraction(ArrayList<String> line) {
        super(line);
        journey = line.get(3);
    }

    @Override
    double offPeakPrice() { return Double.parseDouble(baseCost) * 0.5; }

    public String toString() {
        return name + "@" + baseCost + "@" + type + "@" + journey;
    }

    // Double because client may wish for higher degree of accuracy in future.
    public double getJourney() { return Double.parseDouble(journey); }

    @Override
    public int getAgeRating() {
        return 0;
    }

    /* TEST HARNESS
    public static void main(String[] args) {
        String x = "name@250@type@20.3";
        ArrayList<String> y = new ArrayList<>(Arrays.asList(x.split("@")));
        TransportAttraction test = new TransportAttraction(y);
        System.out.println(test);
        System.out.println(test.getJourney());
        System.out.println(test.offPeakPrice());
        System.out.println(test.getAgeRating());
    }
     */
}
```

Listing 3: GentleAttraction.java

```java
package com.Greedy;
import java.util.ArrayList;

// GentleAttraction is another subclass of Attraction, therefore, extends Attraction definition as specified
final class GentleAttraction extends Attraction {

    protected String capacity;

    GentleAttraction(ArrayList<String> line) {
        super(line);
        capacity = line.get(3);
    }

    // Return capacity of gentle attraction as an integer.
    public int getCapacity() { return Integer.parseInt(capacity); }

    // Method that returns off peak price of ride.
    @Override
    double offPeakPrice() { return Double.parseDouble(baseCost) * 0.2; }

    @Override
    public String toString() {
        return name + "@" + baseCost + "@" + type + "@" + capacity;
    }

    @Override
    public int getAgeRating() { return 0; }

    /* TEST HARNESS
    public static void main(String[] args) {
        String x = "name@250@type@4";
        ArrayList<String> y = new ArrayList<>(Arrays.asList(x.split("@")));
        GentleAttraction test = new GentleAttraction(y);
        System.out.println(test);
        System.out.println(test.getCapacity());
        System.out.println(test.getAgeRating());
        System.out.println(test.offPeakPrice());
    }
    */
}
```

Listing 4: RollerCoaster.java

```java
package com.Greedy;

import java.util.ArrayList;

// RollerCoaster is an extension on what has been defined in Attraction.
final class RollerCoaster extends Attraction implements Comparable<RollerCoaster> {

    // Protected fields for roller coaster.
    protected String age;
    protected String maxMPH;

    // ArrayList to extend definition of subclass.
    RollerCoaster(ArrayList<String> line) {
        super(line);
        age = line.get(3);
        maxMPH = line.get(4);
    }

    // returns double for maximum MPH of roller-coaster.
    public Double getMaxMPH() {
        return Double.parseDouble(maxMPH);
    }

    // offPeakPrice
    @Override
    double offPeakPrice() {
        return Double.parseDouble(baseCost);
    }

    //returns integer of the rides age-rating (e.g. 12).
    @Override
    public int getAgeRating() {
        return Integer.parseInt(age);
    }

    @Override
    public String toString() {
        return name + "@" + baseCost + "@" + type + "@" + age + "@" + maxMPH;
    }

    // Compares previous and current maxMPH to sort into list later (refer to
    ↪ Pleasure_Point class).
    @Override
    public int compareTo(RollerCoaster rollerCoaster) {
        return Double.compare(rollerCoaster.getMaxMPH(), this.getMaxMPH());
    }

    /* TEST HARNESS
    public static void main(String[] args) {
        String x = "name@250@type@20@55.2";
        ArrayList<String> y = new ArrayList<>(Arrays.asList(x.split("@")));
        RollerCoaster test = new RollerCoaster(y);
        System.out.println(test);
        System.out.println(test.getMaxMPH());
        System.out.println(test.offPeakPrice());
        System.out.println(test.getAgeRating());
    }
    */
}
```

Listing 5: Customer.java

```
1  package com.Greedy;
2
3  import java.util.ArrayList;
4
5  public class Customer {
6
7      protected final String userID;
8      protected String name;
9      protected String age;
10     protected String balance;
11     protected String personalDiscount;
12     protected Double temp;
13
14
15     // Defines 'Customer' object. Used ArrayList as size of Customer does change.
16     Customer(ArrayList<String> line) {
17         userID = line.get(0);
18         name = line.get(1);
19         age = line.get(2);
20         balance = line.get(3);
21         if (line.size() > 4)
22             personalDiscount = line.get(4);
23         else
24             personalDiscount = null;
25     }
26
27     // Accessors
28     public int getUserID() { return Integer.parseInt(userID); }
29     public String getName() { return name; }
30     public int getAge() { return Integer.parseInt(age); }
31     public double getBalance() { return Double.parseDouble(balance); }
32     public String getPersonalDiscount() { return personalDiscount; }
33
34     // Method that adds 'amount' to the customers balance.
35     public void addFunds(String amount) {
36         double temp = getBalance() + Integer.parseInt(amount);
37         balance = String.valueOf(temp);
38     }
39
40     // Method that reduced balance when using attraction (if customer has enough).
41     public void useAttraction(double attractionPrice) throws
       ↪ InsufficientBalanceException {
42
43         double price;
44
45         if (personalDiscount == null) {
46             if (getBalance() - attractionPrice < 0) {
47                 throw new InsufficientBalanceException
48                         ("Not enough funds (" + -(getBalance() - attractionPrice) +
                           ↪ "p) difference");
49             }
50             balance = Double.toString(getBalance() - attractionPrice);
51             System.out.println(attractionPrice);
52             Simulate.accumulate += attractionPrice;
53
54         }
55         else {
56             switch (personalDiscount) {
57                 case "FAMILY":
```

```
58                      price = attractionPrice * 0.85;
59                      temp = getBalance() - price;
60                      if (temp < 0) {
61                          throw new InsufficientBalanceException("Not enough funds (" +
                               ↪ -temp + "p) difference");
62                      }
63                      else {
64                          Simulate.accumulate += price;
65                          System.out.println(price);
66                          balance = temp.toString();
67                      }
68                      break;
69
70                  case "STUDENT":
71                      price = attractionPrice * 0.9;
72                      temp = getBalance() - price;
73                      if (temp < 0) {
74                          throw new InsufficientBalanceException("Not enough funds (" +
                               ↪ -temp + "p) difference");
75                      }
76                      else {
77                          Simulate.accumulate += price;
78                          System.out.println(price);
79                          balance = temp.toString();
80                      }
81                      break;
82
83                  default:
84                      //balance = Double.toString(getBalance() - attractionPrice);
85                      System.out.println("error");
86                      break;
87              }
88          }
89      }
90
91      public void useAttraction(double limit, double attractionPrice) throws
92              AgeRestrictionException, InsufficientBalanceException {
93          if (getAge() < limit) {
94              throw new AgeRestrictionException("Sorry, you are not old enough to use
                   ↪ this attraction.");
95          }
96          else
97              useAttraction(attractionPrice);
98      }
99
100     // Displays Discount information
101     protected static String getAvailableDiscountInfo() {
102         String infoTit = String.format(
103                 "%-25s %-25s %-25s", "Family Discount",
104                 "Student Discount", "Park Employees");
105         String info = String.format
106                 ("%-25s %-25s %-25s", "15% deduction",
107                         "10% deduction", "15% deduction");
108         return "\n" + infoTit + "\n" + info;
109     }
110
111     @Override
112     public String toString() {
113         String outPut = userID + "#" + name + "#" + age + "#" + balance;
114         if (personalDiscount == null || personalDiscount.equals("")) return outPut;
```

```
115         else outPut = outPut + "#" + personalDiscount;
116         return outPut;
117     }
118 
119     @Override
120     public int hashCode() {
121         return Integer.parseInt(userID);
122     }
123 
124     /* TEST HARNESS
125     public static void main(String[] args) throws InsufficientBalanceException {
126 
127         // TEST 1 - Customer object with 5 fields:
128         String record = "123456,name,23,54.2,STUDENT";
129         ArrayList<String> objectBase = new
                ↪ ArrayList<>(Arrays.asList(record.split(",")));
130         Customer test = new Customer(objectBase);
131         System.out.println(test);
132 
133         // TEST 2 - Customer object with 4 fields:
134         String record2 = "654321,name,23,23.1";
135         ArrayList<String> objectBase2 = new
                ↪ ArrayList<>(Arrays.asList(record2.split(",")));
136         Customer test2 = new Customer(objectBase2);
137         System.out.println(test2);
138 
139         System.out.println(getAvailableDiscountInfo());
140 
141         // WORKS
142         test2.useAttraction(4444);
143         test.useAttraction(4444);
144     }
145      */
146 }
```

Listing 6: PleasurePoint.java

```
1
2  package com.Greedy;
3
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.util.*;
7
8
9
10 class Pleasure_Point {
11
12     // Static data types/structure for referencing in multiple methods.
13     private static double totalCap;
14     private static int totalDistance;
15     private static int countGen = 0;
16     private static ArrayList<RollerCoaster> rolList = new ArrayList<>();
17
18     // HashMaps to store multiple customer and attraction objects (separately).
19     private static HashMap<Integer, Customer> customers = new HashMap<>();
20     private static HashMap<String, Attraction> attractions = new HashMap<>();
21
22     // Constructor automatically creates customers and attractions when
          ↪ Pleasure_Point class is created.
23     public Pleasure_Point() throws FileNotFoundException {
24         genAttractions();
25         genCustomers();
26     }
27
28     // Generates customer objects and adds them to the 'customers' HashMap.
29     public void genCustomers() throws FileNotFoundException {
30
31         Scanner scannerC = new Scanner(new File("customers.txt"));
32
33         /*
34         Loop that scans each line in file, stores as string type,
35         then splits string based off regex '#'
36         into a string[] type (called "fields").
37         With this data, we can create a 'new' instance of
38         class Customer (called "cus").
39         when then add this to CustomerList class (called "customers").
40         Repeats till encounters empty line.
41         */
42
43         while (scannerC.hasNextLine()) {
44             String record = scannerC.nextLine();
45             ArrayList<String> objectCus = new
                  ↪ ArrayList<>(Arrays.asList(record.split("#")));
46             Customer cus = new Customer(objectCus);
47             customers.put(cus.getUserID(), cus);
48         }
49         for (Integer key : customers.keySet()) {
50             System.out.println(customers.get(key));
51         }
52         System.out.println("\n");
53     }
54
55     // Generates attraction objects and adds them to the 'Attractions' HashMap.
56     public void genAttractions() throws FileNotFoundException {
57         Scanner scannerA = new Scanner(new File("attractions.txt"));
```

```
58
59          while(scannerA.hasNextLine()) {
60
61              String record = scannerA.nextLine();
62              ArrayList<String> objectBase = new
                    ↪ ArrayList<>(Arrays.asList(record.split("@")));
63              Attraction a = new Attraction(objectBase) {
64                  @Override
65                  double offPeakPrice() { return 0; }
66
67                  @Override
68                  public int getAgeRating() { return 0; }
69              };
70
71              // Switch checks what type of attraction it is, then builds object
                    ↪ accordingly.
72              switch(a.type) {
73                  case "TRA":
74                      ArrayList<String> objectT = new
                            ↪ ArrayList<>(Arrays.asList(record.split("@")));
75                      TransportAttraction tran = new TransportAttraction(objectT);
76                      totalDistance +=  tran.getJourney();
77                      attractions.put(tran.name.trim(), tran);
78                      break;
79                  case "GEN":
80                      ArrayList<String> objectG = new
                            ↪ ArrayList<>(Arrays.asList(record.split("@")));
81                      GentleAttraction gen = new GentleAttraction(objectG);
82                      totalCap += gen.getCapacity();
83                      countGen++;
84                      attractions.put(gen.name.trim(), gen);
85                      break;
86                  case "ROL":
87                      ArrayList<String> objectR = new
                            ↪ ArrayList<>(Arrays.asList(record.split("@")));
88                      RollerCoaster rol = new RollerCoaster(objectR);
89                      attractions.put(rol.name.trim(), rol);
90                      rolList.add(rol);
91                  break;
92                  default:
93                      System.out.println("Invalid attraction detected!");
94                      break;
95              }
96          }
97      }
98
99      // Accessors for HashMap(s).
100     public HashMap<Integer, Customer> getCustomerHashMap() {
101         return customers;
102     }
103     public HashMap<String, Attraction> getAttractionHashMap() { return attractions; }
104
105     // Method adds new customer (assuming request in transactions format).
106     public void addCustomer(String[] line) {
107         ArrayList<String> cus = new ArrayList<>(Arrays.asList(line));
108         if (cus.contains("ADD_CUSTOMER")) cus.remove(0);
109         Customer newCustomer = new Customer(cus);
110         customers.put(newCustomer.getUserID(), newCustomer);
111     }
112
```

```
113     // Shows result of adding all distances from transport attractions.
114     public void calculateTotalTransportDistance() {
115         System.out.println("The total transport distance of all transport
            ↪ attractions: " + totalDistance);
116     }
117
118     // Shows average amount of seats in a gentle attraction.
119     public void calculateAverageGentleCapacity() {
120         System.out.println("Average gentle attraction capacity (approx): " +
            ↪ Math.round(totalCap/countGen));
121     }
122
123     public void calculateMedianCoasterSpeed() {
124         // finds median if number of of roller coasters is odd.
125         rolList.sort(RollerCoaster::compareTo);
126         if(rolList.size()%2 != 0) {
127             System.out.println("The median roller coaster speed: " +
                ↪ rolList.get(rolList.size() / 2).maxMPH);
128         }
129         // finds median if number of of roller coasters is even.
130         else {
131             double temp =
                ↪ Math.round(Double.parseDouble(String.valueOf(rolList.size()/2)));
132             double median = ((temp - 1) + (temp + 2)) / 2;
133             System.out.println(median);
134         }
135     }
136
137     /* TEST HARNESS
138     public static void main(String[] args) throws FileNotFoundException {
139
140         genCustomers();
141         genAttractions();
142         System.out.println(customers.get(585526));
143         System.out.println(attractions.get("Tower of Midnight").getCost());
144         String[] line = {"999888", "dave", "45", "65.22", null };
145         //AddCustomer(line);
146         customers.remove(999888);
147         System.out.println(customers.get(999888));
148
149         // RETURN TO STATIC TO USE
150         //System.out.println(getCustomerHashMap().get(585526));
151         //System.out.println(getAttractionHashMap().get("Tower of Midnight"));
152
153         // MAKE STATIC TO WORK IN THIS MAIN!
154         //calculateTotalTransportDistance();
155         //calculateAverageGentleCapacity();
156         //calculateMedianCoasterSpeed();
157     }
158     */
159 }
```

Listing 7: Simulate.java

```
1
2  package com.Greedy;
3
4  import java.io.File;
5  import java.io.FileNotFoundException;
6  import java.util.Arrays;
7  import java.util.Scanner;
8
9  /**
10  BY: CALLUM CLEGG [100279967]
11
12  Data-Structures:
13  I decided to make my customer and attraction constructors store data in an ArrayList.
14  I felt this was justified as the number of fields varied depending on the type of
15  customer or attraction subclass.
16
17  Furthermore, I used hash-maps to store my all customers and all attractions because
        ↪ it is
18  capable of faster element access, I can also use hash keys which fits well with
        ↪ unique
19  fields provided (i.e. 'userID' is always unique in-addition to attraction names
        ↪ [viable primary key]).
20
21  Testing:
22  I have included the txt files I used to test different types of
        ↪ customers/attractions.
23  The customers and attractions I used were pre-existing, meaning I could use the
        ↪ original
24  'transactions.txt' in order to test if my classes were performing as expected.
25
26  All classes (except those that extend 'exception' or Simulate), have a 'TEST
        ↪ HARNESS' located
27  at the bottom of each class. Redundant imports are present due to their need when
        ↪ 'TEST HARNESS'
28  is un-commented.
29
30  LAST UPDATED: 08/04/2020
31  */
32
33 class Simulate {
34
35     // Single double variable that 'accumulates' profit from useAttraction method.
36     static double accumulate;
37
38     public static void main(String[] args) throws FileNotFoundException {
39         Simulate assignment = new Simulate();
40         System.out.println(assignment);
41     }
42
43     Simulate() throws FileNotFoundException {
44
45         Pleasure_Point pleasurePoint = new Pleasure_Point(); // Create instance of
               ↪ Pleasure_Point object.
46         Scanner transactions = new Scanner(new File("transactions.txt"));
47
48         while (transactions.hasNextLine()) {
49
50             //Break down each line of the transactions file into a number of fields.
51             String record = transactions.nextLine();
```

```
52              String[] fields = record.split(",");
53
54              // Switch checks what type of transaction the line is, acts accordingly
                    ↪ (based off index 0).
55              switch (fields[0]) {
56
57                  case "USE_ATTRACTION":
58
59                      String name = fields[3].trim();
60
61                      double price =
                            ↪ pleasurePoint.getAttractionHashMap().get(name).getCost();
62                      double priceOffPeak =
                            ↪ pleasurePoint.getAttractionHashMap().get(name).offPeakPrice();
63
64                      // Required age for ride (only relevant to roller coasters).
65                      int age =
                            ↪ pleasurePoint.getAttractionHashMap().get(fields[3]).getAgeRating()
66
67                      // Try catch to see if user has enough in balance to pay for ride
                            ↪ referenced in index 3.
68                      try {
69                          if
                                ↪ (pleasurePoint.getAttractionHashMap().get(fields[3]).type.equa
                                ↪ {
70
71                              if (fields[1].equals("STANDARD_PRICE")) {
72                                  System.out.println(pleasurePoint.getCustomerHashMap().
73                                          get(Integer.parseInt(fields[2])));
74                                  pleasurePoint.getCustomerHashMap().
75                                          get(Integer.parseInt(fields[2])).useAttraction(age,
                                                ↪ price);
76                                  System.out.println(pleasurePoint.getCustomerHashMap().
77                                          get(Integer.parseInt(fields[2])) + "\n");
78                              } else if (fields[1].equals("OFF_PEAK")) {
79                                  System.out.println(pleasurePoint.getCustomerHashMap().
80                                          get(Integer.parseInt(fields[2])));
81
82                                  pleasurePoint.getCustomerHashMap().
83                                          get(Integer.parseInt(fields[2])).useAttraction(age,
                                                ↪ priceOffPeak);
84                                  System.out.println(pleasurePoint.getCustomerHashMap().
85                                          get(Integer.parseInt(fields[2])) + "\n");
86                              } else {
87                                  System.out.println("ERROR: OFF_PEAK or STANDARD_PRICE
                                        ↪ not found!");
88                                  System.out.println(pleasurePoint.getCustomerHashMap().
89                                          get(Integer.parseInt(fields[2])));
90                                  break;
91                              }
92                          }
93                      } catch (InsufficientBalanceException | AgeRestrictionException
                            ↪ e) {
94                          System.out.println(e.toString() + " -> " + fields[2] + "\n");
95                      }
96
97                      try {
98                          if
                                ↪ (!pleasurePoint.getAttractionHashMap().get(fields[3]).type.equ
                                ↪ {
```

```
99
100                             if (fields[1].equals("STANDARD_PRICE")) {
101                                 System.out.println(pleasurePoint.getCustomerHashMap().
102                                         get(Integer.parseInt(fields[2])));
103                                 pleasurePoint.getCustomerHashMap().get(Integer.parseInt(fiel
104                                         useAttraction(Double.parseDouble(String.valueOf(pric
105                                 System.out.println(pleasurePoint.getCustomerHashMap().
106                                         get(Integer.parseInt(fields[2])) + "\n");
107                             } else if (fields[1].equals("OFF_PEAK")) {
108                                 System.out.println(pleasurePoint.getCustomerHashMap().
109                                         get(Integer.parseInt(fields[2])));
110                                 pleasurePoint.getCustomerHashMap().
111                                         get(Integer.parseInt(fields[2])).useAttraction(price
112                                 System.out.println(pleasurePoint.getCustomerHashMap().
113                                         get(Integer.parseInt(fields[2])) + "\n");
114                             } else {
115                                 System.out.println("ERROR: OFF_PEAK or STANDARD_PRICE
                                        ↪ not found!");
116                                 System.out.println(pleasurePoint.getCustomerHashMap().
117                                         get(Integer.parseInt(fields[2])) + "\n");
118                                 break;
119                             }
120                         }
121                     } catch (InsufficientBalanceException e) {
122                         System.out.println(e.toString() + " -> " + fields[2] + "\n");
123                     }
124
125                     break;
126
127                 case "NEW_CUSTOMER":
128                     fields = Arrays.copyOfRange(fields, 1, fields.length);
129                     pleasurePoint.addCustomer(fields);
130                     System.out.println("NEW CUSTOMER: " +
                            ↪ pleasurePoint.getCustomerHashMap().
131                                 get(Integer.parseInt(fields[0])) + "\n");
132                     break;
133                 case "ADD_FUNDS":
134                     System.out.println("Before: " +
                            ↪ pleasurePoint.getCustomerHashMap().
135                                 get(Integer.parseInt(fields[1])));
136
137                     System.out.println(fields[2] + "+");
138
139                     pleasurePoint.getCustomerHashMap().get(Integer.parseInt(fields[1])).addF
140
141                     System.out.println("After: " + pleasurePoint.getCustomerHashMap().
142                             get(Integer.parseInt(fields[1])) + "\n");
143                     break;
144
145                 default:
146                     System.out.println("ERROR");
147             }
148         }
149
150         System.out.println("\n");
151
152         pleasurePoint.calculateAverageGentleCapacity();
153         pleasurePoint.calculateMedianCoasterSpeed();
154         pleasurePoint.calculateTotalTransportDistance();
155         System.out.println(Customer.getAvailableDiscountInfo());
```

```
156             System.out.println("Total profit: " + accumulate);
157         }
158         @Override
159         public String toString() { return ""; }
160     }
```

Listing 8: InsufficientBalanceException.java

```java
package com.Greedy;

// Custom exception for insufficient balance
class InsufficientBalanceException extends Exception {
    InsufficientBalanceException(String message) {
        super(message);
    }
}
```

Listing 9: AgeRestrictionException.java

```java
package com.Greedy;

// Custom exception for age restriction.
public class AgeRestrictionException extends Exception {
    AgeRestrictionException(String message) {
        super(message);
    }
}
```