School of Computing Sciences

Module: CMP-4008Y Programming 1
Assignment: Coursework Assignment 2

Set by:	Jason Lines (j.lines@uea.ac.uk)
Checked by:	Gavin Cawley (gcc@uea.ac.uk)
Date set:	Friday 7th February 2020
Value:	30%

Date due:	Friday 20th March 2020 3pm (**Week 10**)
Returned by:	Thurs 23rd April 2020
Submission:	PDF prepared with PASS submitted electronically on e:vision

Learning outcomes

The aim of this assignment is to facilitate the development of Java and object orientated programming skills by designing and implementing a program to simulate the operation of a theme park. In addition to using the fundamental concepts that were developed throughout the first assignment (such as classes, objects, instance variables and methods), this assignment will also develop skills in UML class diagrams, file I/O, inheritance, exceptions and basic enumerative types in Java.
Further general learning outcomes include: describing abstract systems using technical diagrams; following specifications when developing software; documenting code to enhance readability and reuse; increased experience of programming in Java; increased awareness of the importance of algorithm complexity; and using inheritance to model relationships between classes.

Specification

Overview

FunCorp, a new division of *GreedyJay Inc.*, have approached you to create a system for managing their new pay-as-you-go theme parks.

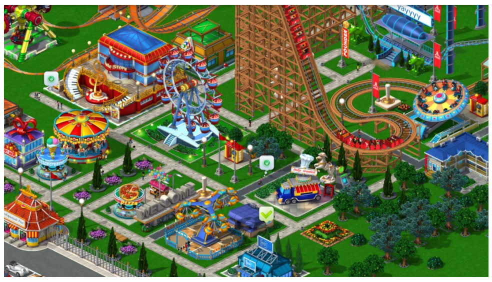

Figure 1: An artistic impression of one of *FunCorp*'s new theme park.

To model this system you will need to create a number of classes. A **theme park** class will store information for a whole park, including its name, a collection of its **attractions** and a collection of its **customers**.

Each individual **attraction** will have information associated with it, including its name and cost to ride the attraction. There will also be different discounts available for certain types of attractions during specific periods of a park‘s opening hours; this will be controlled in your application by using an abstract attraction base class, and then using inheritance to create subclasses each type of attraction (transport, gentle and roller coaster).

You will also need to create a class to store information about **customers**, including basic information such as their name, age, and account balance. Further, this class will require functionality to increase a customer’s account balance when they add funds, and decrease the balance when they ride attractions (subject to any additional customer discounts that they may have).

FunCorp would then like you to test your system using simulated data for a single park. You will therefore need to write a main method class with static methods to read in files to populate information about the park and simulate the use of the theme park (such as new customers joining, account balances being updated and customers using attractions).

A full description of the classes that you are required to create is given in the **Description** section of this document with specific implementation requirements that you must follow. A summary of the marks are also given in the **Marking Scheme** section. Informally, your tasks are to:

1. Describe this abstract system using a UML Class Diagram.
2. Implement each of the specified classes in Java
3. Create a main method class to read data from files to populate your system and simulate use of the theme park.

Please note that you are expected to provide `toString` methods for all object classes, *appropriate* comments to aid someone reading your code, and evidence of testing in all classes (i.e. simple test harnesses in your object classes).

(hint: please read the full assignment rather than diving in straight away - it will make your life a lot easier if you implement it in a logical order).

Description

1. UML Class Diagram (15%)

You are required to create a UML class diagram to describe the classes that are given in the remainder of the **Description** section for the proposed system. You should include all classes, and relationships between them, but you are not required to include accessor and mutator methods in your diagram.

Do not use specialised software to create your UML class diagrams. It is fine to use PowerPoint, Word or other simple **drawing** tools, but do not use any specialised tools that help to automate the drawing of your diagram. Marks will also be deducted due to poor presentation (e.g. make sure the text is not tiny once it is printed out). Your UML class diagram should follow the conventions given in the lectures. To avoid issues when including your diagram in PASS, make sure to save your UML class diagram as a .pdf and do not include spaces in your file name (you can use *print to pdf* or *export to pdf* in PowerPoint/Word to help get it into .pdf).

(hint: read the full specification first, but then come back and create a UML class diagram for the system before writing code. It will help you understand all of the functionality and relationships between the classes, and give you something to refer to while working on the code).

2. Object Classes

The classes that you must implement are as follows. Please note that *all* object classes should have a main method to demonstrate testing and an appropriate implementations of `toString`.

2.1 `Attraction` (10%)

Attractions are split into three different categories: transport attractions (e.g. chair lift, mono rail), gentle attractions (e.g. Ferris wheel, mini golf) and roller coasters (e.g. the corkscrew). You will model this through using an abstract base class for attractions and specific subclasses for the three types of attraction.

Each attraction has its own **name** and **base price** for using the attraction. For simplicity, prices will be stored in pence and represented as integer numbers (e.g $\pounds 1.00 = 100$).

Additionally, specific discounts are applied to each category of attraction when used during off-peak times (these should be rounded down to the nearest penny, where necessary). These off-peak discounts are as follows:

- **transport** attractions cost 50% of their base price at off-peak times.
- **gentle** attractions cost 80% of their base price during *off-peak* times.
- **roller coasters** do not have any discount during *off-peak* times.

You are required to write an abstract class called `Attraction`. This class should include fields to store information about the name and base price for using a ride. You should also include accessor and muators for both fields and a suitable `toString` method. Further, your class should include an abstract method called `getOffPeakPrice` that takes no arguments and has a return type of `int`.

2.2 `TransportAttraction`, `GentleAttraction` and `RollerCoaster` (15%)

You are required to extend `Attraction` into these three sub-classes. Each should include an appropriate implementation of `getOffPeakPrice` and the following additional functionality:

- `TransportAttraction` should have an additional field to record the distance that the journey will cover (rounded up to the nearest meter).
- `GentleAttraction` should have an additional field to store the capacity of an attraction (e.g. 4 people).
- `RollerCoaster` should have two additional fields: one for the minimum age (in years) to use the roller coaster and another for the top speed of the roller coaster (in mph). For example, the top speed of the roller coaster called *The Corkscrew* is 62.4mph.

Include accessor and mutator methods for the additional fields and ensure that these fields are represented in calls to `toString`.

2.3. `Customer` (20%)

The `Customer` class should store information about a customer‘s account. This includes basic information such as their account number, their name and their age (in years), plus two additional fields: account balance and personal discount. The account balance is the amount of money that is in their account, which should be a positive number that is reduced by the appropriate amount when using an attraction. Personal discount is an additional discount that a customer *may* have. A personal discount should be applied *after* any off-peak discount, and personal discounts can be one of the following values: **none**, **student** or **family**. Students receive an additional 10% off and family members of park employees receive 15% off.

Your class should include accessors and mutators for all fields except account balance; this should have an accessor but no mutator. Instead of a mutator, there should be a two methods: `addFunds(int amount)` and `useAttraction(int attractionPrice)`. The `addFunds` method will top up a customer‘s balance by the specified amount, while `useAttraction` will simulate using an attraction. It should take in the base price/off-peak price of the attraction that a customer wishes to ride as the `attractionPrice` argument. The method should calculate the final price by applying any personal discount, and attempt to reduce the account balance by this amount. If the customer has a sufficient balance, deduct the appropriate amount and return the price that was paid (i.e. including any personal discount). If they do not have a sufficient balance, throw an `InsufficientBalanceException` exception (you will need to create this class as a sub-class of `Exception`). Further, **overload** `useAttraction` to create a second version of the method that takes in two arguments: the attraction price *and* the age limit of the ride. This version of the method should additionally be able to raise an `AgeRestrictionException` if an age limit is violated. The version of the method that you use in `Simulation.java` should then depend on whether a ride has an age limit or not.

Finally, this class should have a static method called `getAvailableDiscountInfo` that returns a `String` including information about the discount types that are available and how much each is worth.

2.4. `ThemePark` (20%)

You will also need to create a class to model a theme park. This class should have fields for the park’s name, a collection of the attractions at this park, and a collection of the customer accounts that are registered at this park. It should have a single default constructor and methods to add individual customers and rides.

You should also provide methods for:

- `getCustomer(String accountNumber)`,
- `removeCustomer(String accountNumber)`,
- `getRide(String rideName)` and
- `removeRide(String rideName)`.

These methods should each throw either a `CustomerNotFoundException` or `RideNotFoundException` (you will need to make these). You will also need to provide get/set methods for the park‘s name.

Additionally, you should have the following methods:

- `calculateTotalTransportDistance()` which should return the sum of the distances of all transport attractions in this park,
- `calculateAverageGentleCapacity()` which should return the average capacity of all gentle attractions in this park as a decimal number, and
- `calculateMedianCoasterSpeed()` which should return the median speed of all roller coasters in this park.

It is up to you to decide how you wish to store collections of attractions and customers. The simplest solution is to use arrays/`ArrayList`, but you can use any data structure that extends the Java `Collection` class (see the Java API [1] for more information on options). A small number of additional marks will be awarded for using a more appropriate data structures than array-based collections, but only if you also justify your choice in a small comment when you declare the fields in this class (i.e. I want to know why you think your solution is better than an array/ArrayList).

3. Main method class: `Simulation` (20%)

Your main method class will simulate the use of a theme park. You are provided with three files:

- `customers.txt`. This file contains information about customer accounts that should be created for your simulation. Each line includes information for an individual customer.
- `attractions.txt`. This file contains information about the attractions that should be created for your simulation. Again, each line includes information for an individual attraction.
- `transactions.txt`. This file includes a chronological list of transaction that you should simulate using the appropriate methods that you have implemented.

You should design the following two methods in your `Simulation` class: `createThemePark` and `simulate` (you may also add any sensible helper methods in this class if you wish).

The `createThemePark` method should return a `ThemePark` that is populated with customers and attractions from the specified files (its up to you to come up with a name for your park though!). It is fine to either take in the file names as arguments for this method, or simply hard-code the file names for this assignment, but make sure that your files are in the root directory of your project, the contents are unchanged, and do not change the names of the files. Once you have returned a theme park, call and print the results of the three methods you implemented in `ThemePark` to print the total transport distance, average gentle attraction capacity and median roller coaster speed.

The `simulation` method should run through each line of the `transactions.txt` file. It should take a `ThemePark` object that has been created in your `createThemePark` method as an input and the method should then simulate the day-to-day use of your theme park; each line is a new action that should be processed and includes the relevant information that you need for that action. These events could be adding new customers to the park (`NEW_CUSTOMER`), customers using an attraction (`USE_ATTRACTION`), and customers adding funds to their account (`ADD_FUNDS`). Your method should handle all actions in the file and print out an **informative**

[1] `https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html`

summary line to the console for each transaction that is performed to describe what was processed. This should also include elegantly handling any exceptional events that happen throughout the simulation, also printing a message to the console for those. Finally, you should keep track of the money that the theme park makes throughout the course of your simulation. The last line of your simulation's output should state the total amount of money that the park made from customers riding their attractions.

You should then include a `main` method in your `Simulation` class that calls these methods without prompting for any user input (i.e. it should call `createThemePark` and pass the returned `ThemePark` into the `simulation` method without requiring any keyboard input to run when PASS calls your main method).

Relationship to formative work

Please see the following lectures and labs for background on each of the specified tasks:

- **Gavin's content from Semester 1**. In particular, the fundamentals from the first assignment are relevant here. Lecture 5 introduces how to use objects, lecture 11 introduces writing classes, lecture 14 includes using collections to store objects, and lecture 16 introduces inheritance.
- **Classes and objects**: the use case example that I went through in week 2 of semester 2 demonstrates the structure of a typical class. The lab reinforced this.
- **UML class diagrams**: these were introduced in semester 2 week 3, and further explained in week 4 with relevance and examples including inheritance.
- **File I/O**: Gavin introduced this in semester 1, and I have shown an alternative method of doing this in week 3 of semester 2 (with example code in the lecture and exercises in the associated lab).
- **Inheritance**: this was covered in more detail in week 4 of semester 2.
- **Exceptions**: this will be covered in week 5 of semester 2, 5 weeks before the deadline. I have intentionally released the coursework a week earlier than planned, before covering Exceptions, to give you extra time (since there is no teaching in week 6 due to Do Something Different week). We will cover this well in advance of the deadline however and the exercises in the week 5 labs will reinforce the use of Exceptions in Java.

Deliverables

Your solution must be formatted using PASS, which is available on all laboratory machines. You must use PASS to produce a .pdf file containing your UML class diagram, the source code of the program, the compiler messages (if any) and the output of the program. This is what will be marked and the .pdf must be submitted on e:vision. **IMPORTANT: your solution will ONLY be marked if it has been submitted on e:vision**. PASS does not submit your work for you and it is your responsibility to upload your PASS-formatted .pdf to e:vision before the deadline.

We strongly recommend that you also upload your .pdf file to BlackBoard before the deadline. **Please note that the Blackboard version will not be marked directly and the official submission must be made on e:vision - it will be treated as a non-submission if you do not submit your work on e:vision**. Further, if your Blackboard submission does not match your e:vision submission then we will ignore the Blackboard version. The reason we suggest that you also submit your work on Blackboard is that there are sometimes issues that are not easily resolved using only the printed code listing, and the markers may wish to compile and run the program for themselves.

Finally, please be advised that the Hub do not accept individual issues with using UEA computing facilities (including PASS) as a suitable reason for granting extenuating circumstances if a deadline is missed. Please make sure that you leave sufficient time to run your code through PASS before the deadline and do not leave it until the last minute. To try to avoid last-minute issues I have intentionally made the deadline for the assignment coincide with labs on this module. TAs will be there to help, but do not rely on this as they are likely to be in very high demand on the deadline day.

Plagiarism and collusion: Stackoverflow and other similar forums (such as the module discussion board) are valuable resources for learning and completing formative work. However, for assessed work such as this assignment, **you may not post questions relating to coursework solutions in such forums**. Using code from other sources/written by others without acknowledgement is plagiarism, which is not allowed (General Regulation 18).

Resources

- **Previous exercises**: If you get stuck when completing the coursework please revisit the lab exercises that are listed in the *Relationship to formative work* section during your allocated weekly lab sessions. The teaching assistants in the labs will not be able to help you with your coursework directly, but they will be more than happy to help you understand how to answer the (very) related exercises in the lab sheets. You will then be able to apply this knowledge and understanding to the new problems in this coursework assignment.
- **Discussion board**: if you have clarification questions about what is required then please ask these questions on the Blackboard discussion board. This will enable other students to also benefit from the question/answer. Please check that your question has not been asked previously before starting a new thread. **Please ask questions on the discussion board, rather than by email, so other students can also benefit from the answer.**
- **Course text:** *Java Software Solutions* by *Lewis and Loftus*. Any version of this textbook is helpful for Programming 1 and will have specific chapters on topics such as inheritance and Exceptions. You can buy your own copy, but I'd suggest doing a simple online search as many editions of the text are available online for free. The library also has a few copies of the latest version of this textbook too.

Marking Scheme

Itemised marks are provided throughout the assignment description. To summarise:

1. UML Class Diagram (15 marks)
2. Object Classes
   - 2.1. Attraction (10 marks)
   - 2.2. Attraction sub-classes (15 marks)
   - 2.3. Customer (20 marks)
   - 2.4. Theme Park (20 marks)

3. Simulation (20 marks)

Total: 100 Marks