Shiho Numakura

3/9/2018

CS 150

Professor Liew

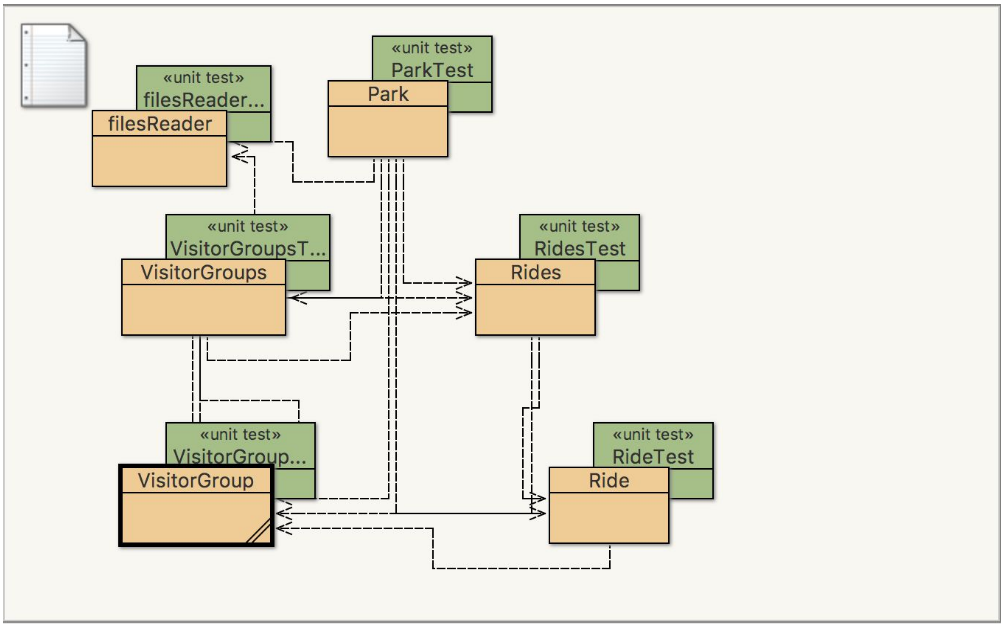
Project 1: Amusement Park Configuration

Introduction:

This project will simulate an amusement park to determine the ideal combination of rides and visitors. It will read a configuration file which determines the rides in that park with specific characteristics to it. Each characteristics of these rides will establish its precise limitations to certain age groups, average wait time, space for the ride, and more. Some of the assumptions made for this project includes that regardless of the group's set age, they will equally desire any of the available rides decided by the random Class. The final goal for this project is to see the statistics of the amusement park after a number of fixed time steps. These results should not only give the statistics of each rides but should allow the user to find the ideal configuration of an amusement.

Approach:

Approach; Figure 1.



As shown in Figure 1, my program contains six classes. To begin coding, I created a filesReader class specifically to read the configuration file using a Scanner (Class Scanner) that was inputted in the *main()*. The class contains multiple methods that allow access to the information in this external file. Without this class, I would not be able to create any visiting groups nor rides for the amusement park for all configuration of the park is based on this file. However, I was unable to implement to read the parameter file; therefore, the program will run without accessing or using any of the information in the parameter file.

Then, I was finally able to start creating the various objects needed in the park to operate. First, I began with the VisitorGroup class which represents a single group visiting my park. These objects are created as groups arrive to the park with different characteristics generated randomly according to the configuration file such as the Gaussian distribution (Class Random). Each object will hold the characteristics of each groups including the type of group they represent, the number of people in the group and their ages as well. Although the likeliness that the group has limited time to stay is higher when there is a younger visitor, regardless of the characteristics of the group, they will all be attracted to all rides equally when creating the list of rides they want to ride. This VisitorGroup class will be stored in a container class called VisitorGroups class, which has an ArrayList (Class ArrayList<E>). to store VisitorGroup objects. From here, it will call to find a ride for all the groups in the park that are not in a ride at the moment. When a group either rides all the rides or unable to ride the ride due to age limitation, they are considered as satisfied groups and will leave the park.

Next, I created a Ride class which represents the each ride in the park. Here, I have three ArrayList (Class ArrayList) to store the three different types of groups for the ride: groups on the ride, groups on the line, and the groups that has just finished. Since the simulation of this park depends on the behavior of the groups in the park, each ride will keep in track of its state at that time step. For example, there will be a container of VisitorGroup objects that are on line to ride the ride, a container for objects in the ride, and a container for objects leaving the rides. It also updates what the specific ride's activity at the moment: "loaded", "running", and "finished." Since each ride has different characteristics, they are each created according to the expected configuration. However, to even join the line for the ride, each VisitorGroup object are required to check whether there is room in the line and whether they satisfy the age restrictions. These Ride objects are all stored in the Rides class container which will update each time step for all rides in the park.

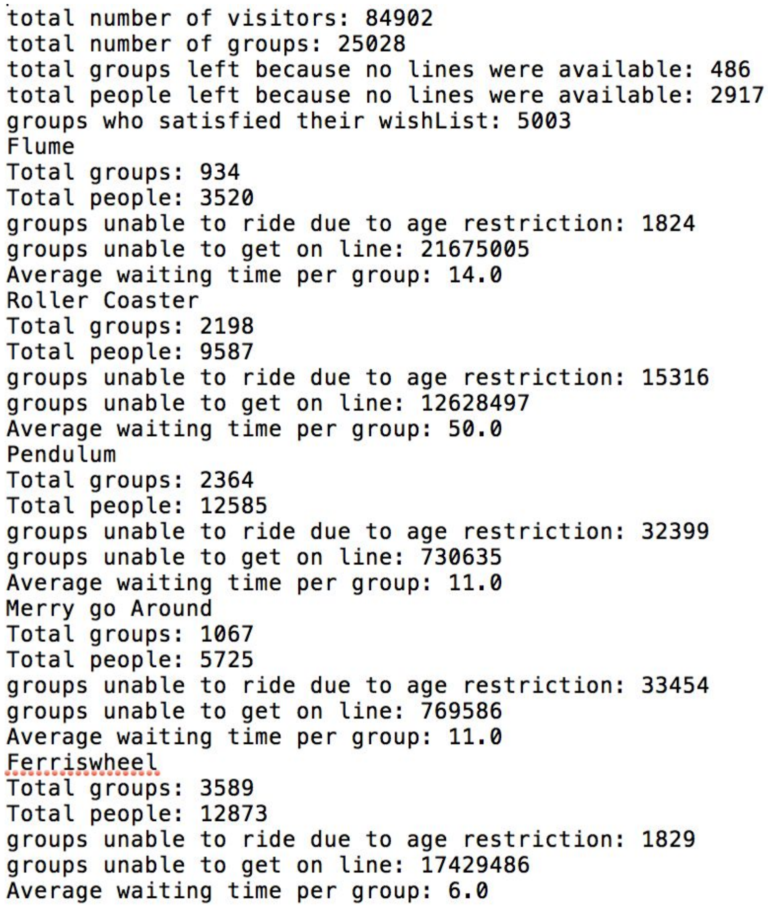
Finally, the Park class will act as the experiment controller class. This will put all objects together and take the statistics of the park at the end. Here, I purposely put all operations usually done in the experiment controller in my Park class to allow myself to generate output file of different configuration results at once. This was solely for myself to collect data and the analysis. Regardless, in the *main()* parameter, it will input the configuration file and call a constructor for my Park class, which is where my experimental set up is. When a Park object is created, it will take the configuration file then run 1000 time steps for the park. As time step increases, more VisitorGroup objects are generated and enter the park rides. Once the park goes through 1000 time steps using a for loop, it will print out various statistics of the park as a whole and for each ride. Some of these statistics collected include the number of groups which entered the park and the number of groups which rode each rides.

As I was performing my experiments, I noticed that my code wasn't working as I expected. Then, when I had gone through my code, I realized that one of my data label was wrong. Data labeled "left because no lines were available" were actually the data of the people who had left due to their given time limit to represent that not all groups have unlimited time to spend at the park. I also realized that I forgot to keep track of the people who had left due to the unavailability of line space in rides.

Methods:

To actually use this program to perform the experiment, I first called the Park constructor five times using the same configuration. To do this, I used my original configuration file, meaning that when my park has each of all the rides. This allows me to see the varying due to the different VisitorGroup objects generated throughout each run. Then, according to the average of my results, though I should've changed my parameter file, I changed the configuration of the rides in my configuration file itself. By doing this, although each group will not ride the same exact ride, they will ride the same type of ride if they wish to. I also realized that I did not take into account the space availability in the park. Though I am unable to keep in track of the space as people arrive to the park, I assume that it is ideal for an amusement park to have more people leaving the park satisfied. However, I did examine the configuration of the rides to fit into the given space for the park.

Data and Analysis:

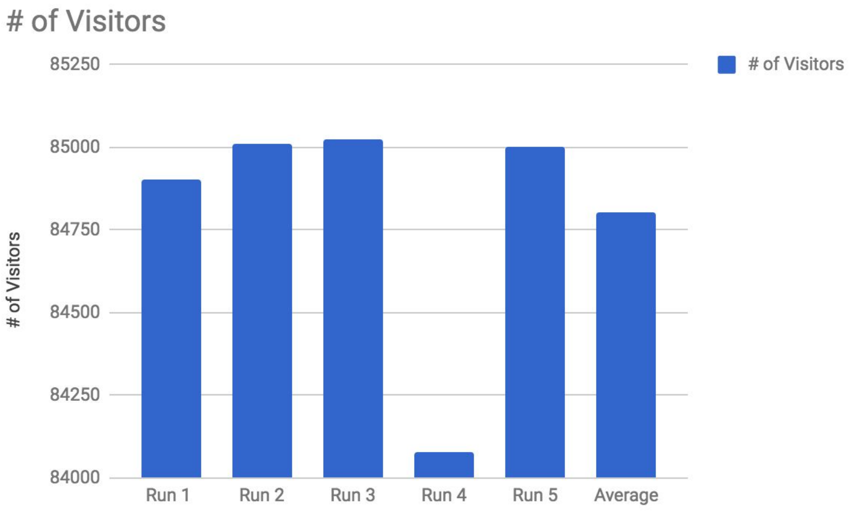


```
total number of visitors: 84902
total number of groups: 25028
total groups left because no lines were available: 486
total people left because no lines were available: 2917
groups who satisfied their wishList: 5003
Flume
Total groups: 934
Total people: 3520
groups unable to ride due to age restriction: 1824
groups unable to get on line: 21675005
Average waiting time per group: 14.0
Roller Coaster
Total groups: 2198
Total people: 9587
groups unable to ride due to age restriction: 15316
groups unable to get on line: 12628497
Average waiting time per group: 50.0
Pendulum
Total groups: 2364
Total people: 12585
groups unable to ride due to age restriction: 32399
groups unable to get on line: 730635
Average waiting time per group: 11.0
Merry go Around
Total groups: 1067
Total people: 5725
groups unable to ride due to age restriction: 33454
groups unable to get on line: 769586
Average waiting time per group: 11.0
Ferriswheel
Total groups: 3589
Total people: 12873
groups unable to ride due to age restriction: 1829
groups unable to get on line: 17429486
Average waiting time per group: 6.0
```

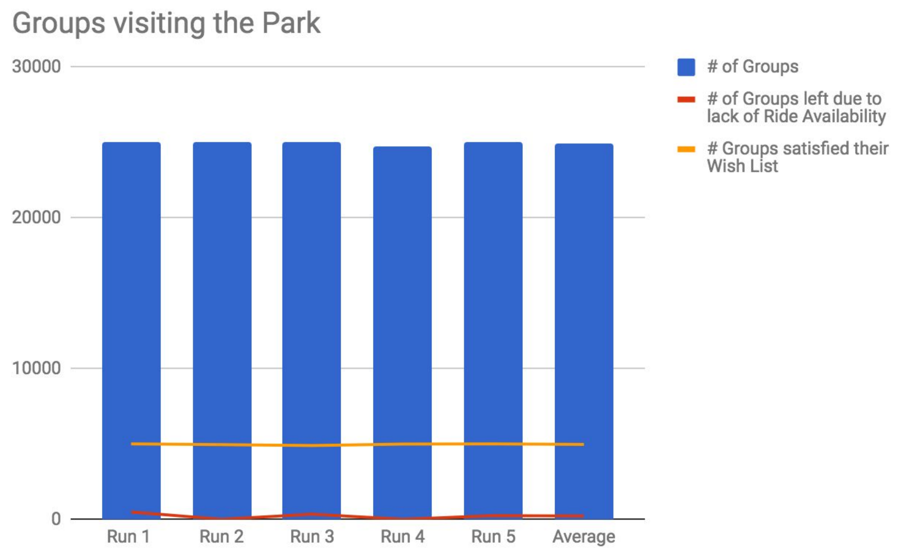
Data and Analysis; Figure 1. This is how my output file shows.

Figure 1 shows all my statistics that were take in each run of a park. First five statistics will represents the numbers taken as the whole amusement. Then, it will present the statistics of each of the rides.

Following will be the data representing all five runs using the original configuration.



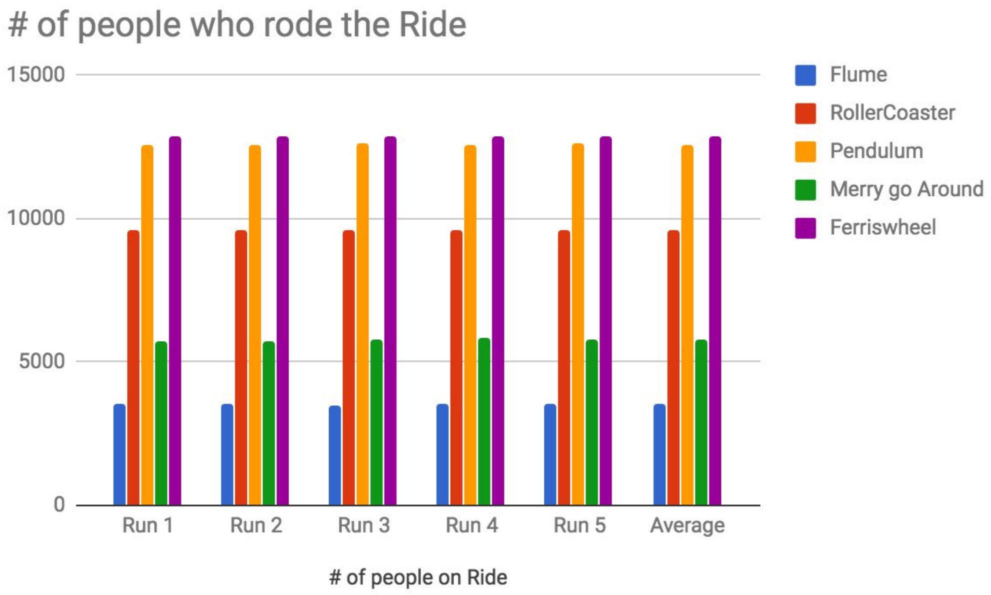
Data and Analysis; Figure 2. This shows the number of people who arrived the park during the run.



Data and Analysis; Figure 3.
This shows the number of groups that arrived to the park comparing to the number of groups that were satisfied or unsatisfied.

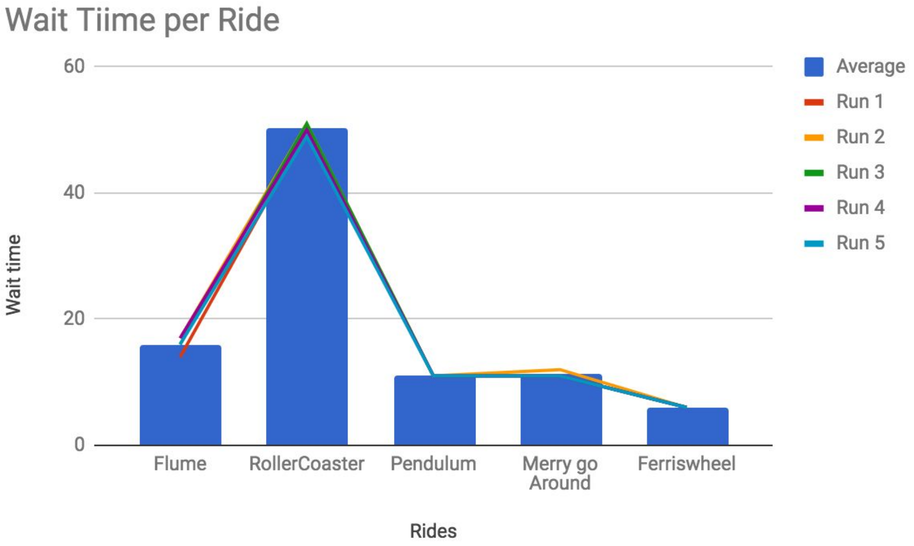
Figure 2 shows that although run 4 is significantly lower than my other runs, my arriving number of people are quite consistent. This can also be confirmed by the number of groups that arrived seen in Figure 3. This also showed that not many of these groups left satisfied. However, it is also apparent that not many groups had to leave due to the lack in space to get on line for a ride.

Following are the results for each rides throughout the same five runs.



Data and Analysis; Figure 4.
This shows the number of people who rode the ride for every run.

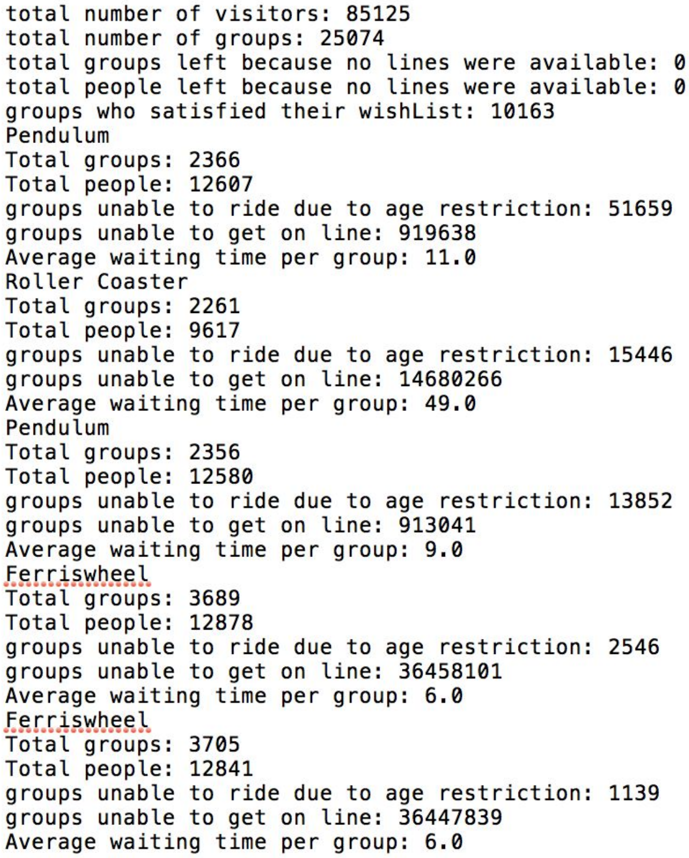
Figure 4 shows that the number of people who rode the ride were extremely consistent as well. Compare to the average of all five runs, all five runs are very similar to the average. This also shows that the Pendulum and the Ferriswheel are extremely popular followed by the RollerCoaster. However, this can be due to the greater line capacity and the capacity of the ride itself, and the age limitation can also be a cause for this wide range. For example. Flume has the least capacity available on its ride. It is only 30% of the ride capacity that the Ferriswheel can hold.



Data and Analysis; Figure 5. This shows the average wait time on line for each group.

Figure 5 shows that the RollerCoaster has the longest average waiting time per group. However, this is predictable because it has the longest line capacity compared to the other rides. Moreover, it was surprising to see that the average waiting time for the Ferriswheel was the shortest considering that it had the most people that rode the ride. Nevertheless, the line capacity is very limited, for it only takes 70 people while the Ferriswheel itself takes 40 people. Therefore, the flow of the line is fast.

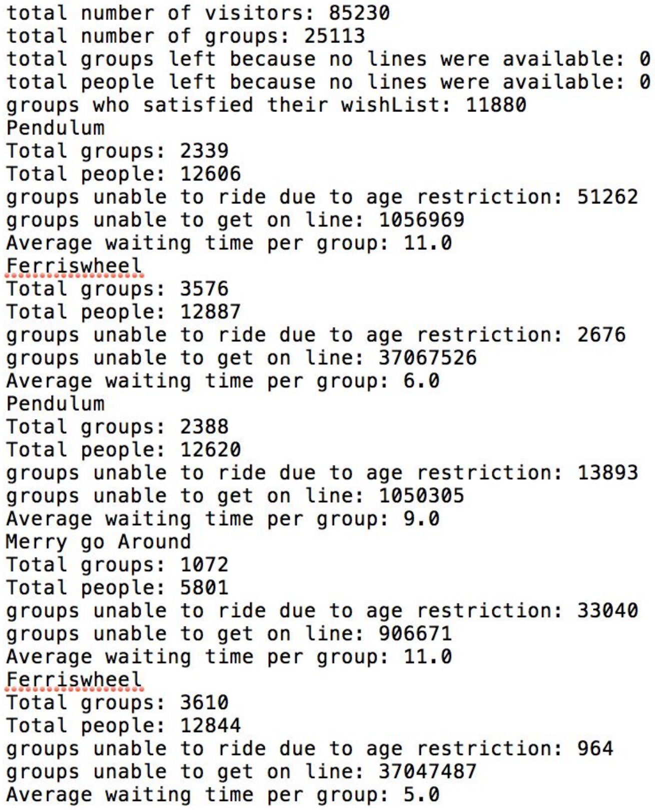
Using these data of my runs, I began to compare closer between the data for each ride to create a better configuration for the amusement park. First, I took away the Flume and the Merry go Around which had the least number of groups who rode it. Then, I added another Pendulum and a FerrisWheel because I noticed that they had the greatest number of people who rode it.



```
total number of visitors: 85125
total number of groups: 25074
total groups left because no lines were available: 0
total people left because no lines were available: 0
groups who satisfied their wishList: 10163
Pendulum
Total groups: 2366
Total people: 12607
groups unable to ride due to age restriction: 51659
groups unable to get on line: 919638
Average waiting time per group: 11.0
Roller Coaster
Total groups: 2261
Total people: 9617
groups unable to ride due to age restriction: 15446
groups unable to get on line: 14680266
Average waiting time per group: 49.0
Pendulum
Total groups: 2356
Total people: 12580
groups unable to ride due to age restriction: 13852
groups unable to get on line: 913041
Average waiting time per group: 9.0
Ferriswheel
Total groups: 3689
Total people: 12878
groups unable to ride due to age restriction: 2546
groups unable to get on line: 36458101
Average waiting time per group: 6.0
Ferriswheel
Total groups: 3705
Total people: 12841
groups unable to ride due to age restriction: 1139
groups unable to get on line: 36447839
Average waiting time per group: 6.0
```

Data and Analysis; Figure 6.
Output file with a revised configuration.

As shown in Figure 6, there were no longer groups leaving the park due to line unavailability. The number of groups satisfied with their list increased significantly as well. However, this specific configuration does not allow any groups with a toddler to ride in any of the rides which still counts them as satisfied groups, for they simply couldn’t have a ride to ride on in their wishlist. Therefore, for my next experiment, I put back the Merry go Around in my park, and took out the Roller Coaster which has the greatest waiting time.



```
total number of visitors: 85230
total number of groups: 25113
total groups left because no lines were available: 0
total people left because no lines were available: 0
groups who satisfied their wishList: 11880
Pendulum
Total groups: 2339
Total people: 12606
groups unable to ride due to age restriction: 51262
groups unable to get on line: 1056969
Average waiting time per group: 11.0
Ferriswheel
Total groups: 3576
Total people: 12887
groups unable to ride due to age restriction: 2676
groups unable to get on line: 37067526
Average waiting time per group: 6.0
Pendulum
Total groups: 2388
Total people: 12620
groups unable to ride due to age restriction: 13893
groups unable to get on line: 1050305
Average waiting time per group: 9.0
Merry go Around
Total groups: 1072
Total people: 5801
groups unable to ride due to age restriction: 33040
groups unable to get on line: 906671
Average waiting time per group: 11.0
Ferriswheel
Total groups: 3610
Total people: 12844
groups unable to ride due to age restriction: 964
groups unable to get on line: 37047487
Average waiting time per group: 5.0
```

Data and Analysis; Figure 7. Output file with a revised configuration.

As shown in Figure 7, I realized that it only made a slight change compared to the previous configuration. However, as I ran it multiple times, it showed that more groups left the park satisfied.

Conclusion

Throughout this project, I was able to realize many things in various aspects. First, although my code was able to print out data, I realize a few errors in the program. This is likely due to the lack in designing the objects before start coding. Another problem was the lack in the use of unit testing. Although I was lucky enough to find my problems, it was a major mistake to code the whole program even before running it during this process. However, throughout the process of analysing this data, I realized that any small change done in the configuration file, can make a big difference. For example, simply replacing a single ride with a different ride can cause a big change in the statistics of the data. Nevertheless, this was only possible because it was based on an external configuration file so the user wouldn’t have to touch the code itself. Overall, though this project required much time and effort, it allowed me to open up my eyes as a coder.

References

(2017, December 19). Retrieved March 22, 2018, Class Random.
https://docs.oracle.com/javase/7/docs/api/java/util/Random.html

(2017, December 19). Retrieved March 22, 2018, Class ArrayList<E>.
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

(2017, December 19). Retrieved March 22, 2018, Class Scanner.
https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Liew, C W. *Amusement Park Simulation.*