**CoasterReviewer 🎢**

A platform for discovering and rating roller coasters.

Summary

CoasterReviewer is, as the name suggests, an application or business centered around coaster enthusiasts as a customer base. There are many roller coasters that exist worldwide, and it would be useful to know nearby locations of roller coasters in a specific area and the popularity or rating of those roller coasters. This will enhance the experience of users looking for a thrill during their upcoming vacation or those looking to contribute to their passion.

For the first phase of this project, the scope will be limited to a subset of roller coasters among the most popular in the United States. Further iterations of the full-scale application should include almost all major roller coasters and theme parks worldwide as data points while using a web-based client. The user base will also be limited to a few participants in the United States in order to test demand and availability of system resources in a controlled environment. The first prioritized use case that is being developed in the first phase of the project is the ability to search and view a list of roller coasters with a variety of filters and sorting including by alphabet or by overall rating. The second prioritized use case is the function of allowing users to leave a number rating on a roller coaster’s page (the page also will include details about the roller coaster). The application will be constructed in the Eclipse Integrated Development Environment using Java and will have thorough unit testing through JUnit. Unit tests will be outlined in detail once development begins, however some examples can be found later in the document under each use case section. Finally, our application will be compiled and built with the help of Apache Ant.

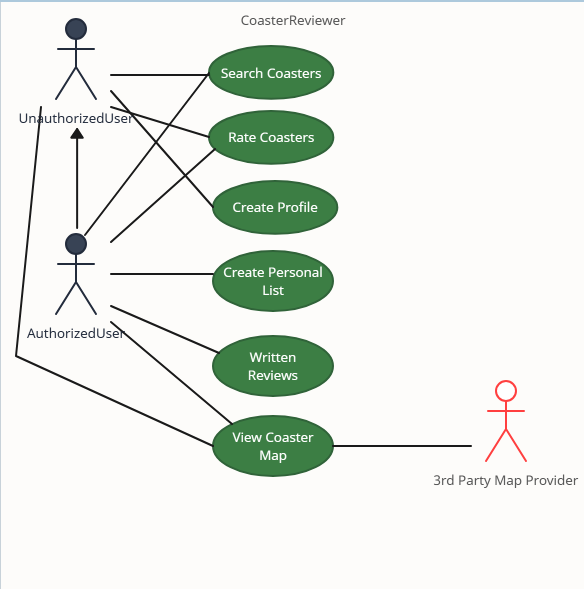
The main dashboard for this application will include a GUI containing a search functionality for roller coasters. Once users search via text, a list of search results will appear with each result being a card that is hyperlinked to the appropriate page for that specific roller coaster result. Here, a User can use most of the app functionality including rating, reviewing, and saving roller coasters to a “favorites” list.

**CoasterReviewer Use Cases and Diagram**

The following are the list of complete use cases organized by priority:

1. **Search Coaster**
2. **Numerically Rate Coaster**
3. Create Profile
4. Create Personal Lists
5. Written Reviews
6. Map Integration

Use Case Diagram



**Fully Dressed Use Cases**

**Search Coaster Use Case**

**Primary Actor:** User

**Assumptions:** User is an unauthorized or authorized user of the application.

**Stakeholders:** CoasterReviewer wants to ensure basic functionality of the application for the enjoyment of enthusiasts.

**Pre-conditions:** User has the application installed locally with all required dependencies and has run the application.

**Post-conditions:** User is displayed a list of all roller coaster results that meet the input criteria. Alternatively, if no results fit the criteria, then a message will be displayed to indicate that no matching results have been found.

**Event Flow:**

1. User runs the application and is presented with the main dashboard for the application.
2. User inputs text into a text field displayed on the top or center of the screen to search.
   1. If no results are found for the current inputted text, a message will be displayed on the screen indicating that no results have been found for the input criteria.
3. Upon successful search (indicating a result has been found for input criteria), results will be displayed on the GUI to the user in the format of a list of cards.
4. User then scrolls through the search result list for desired information.

**Numerically Rate Coaster Use Case**

**Primary Actor:** User

**Assumptions:** User is an authorized or authorized user of the application.

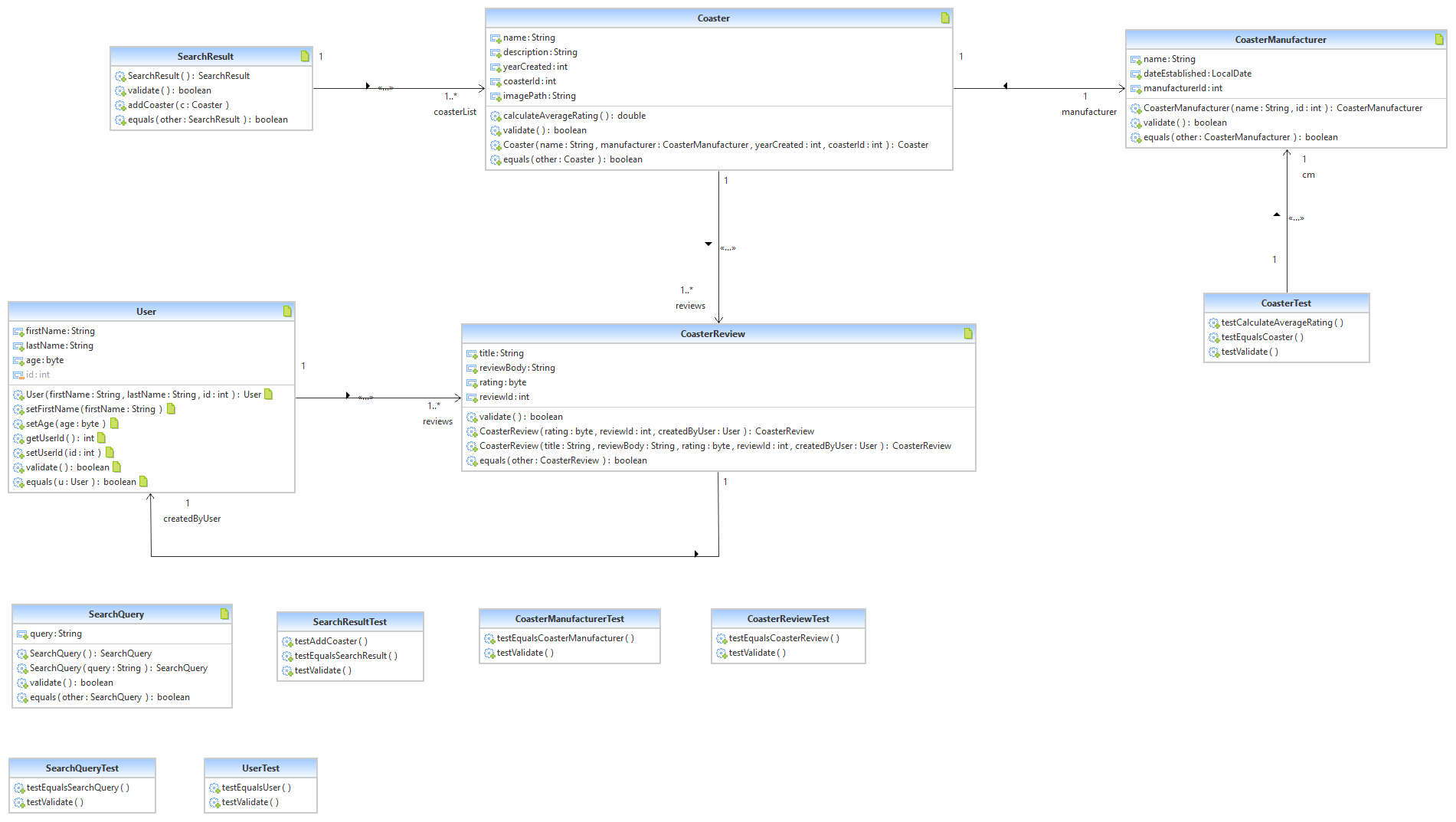
**Stakeholders:** CoasterReviewer wants to highlight notable and outstanding roller coasters for enthusiasts to try and visit

**Pre-conditions:** User has successfully searched and navigated to a roller coaster’s page containing details about said roller coaster.

**Post-conditions:** User has successfully logged their rating and has affected the average rating for the specified roller coaster with their input (User can still make edits to their rating of the coaster post-submission).

**Event Flow:**

1. User has navigated to and is currently viewing a roller coaster page after searching and clicking on a search result.
2. User navigates to the section on the bottom of the GUI and selects a number between 1 and 5 to describe their rating of the roller coaster they are viewing.
3. User presses the submit button to lock in their rating, at which point the average rating statistic for the roller coaster should update.
4. User continues to browse roller coaster details.
   1. ALT FLOW: User presses the edit rating button and updates their numerical rating of the roller coaster after having already submitted a previous rating.
   2. After 4a, user repeats event flow step 3.

**Domain Layer Class Diagram in UML**

**Unique Methods**

* **Coaster.calculateAverageRating()** - this class method returns the average rating calculated by iterating through all reviews of this.Coaster.
* **SearchResult.testAddCoaster(Coaster c)** - this class method is of void type and simply adds Coaster c to the data field containing the list of SearchResult.
* **validate() and equals(other)** - present in all domain classes, this returns a boolean type dependent on if the data fields are validated, and if the other Object is equal to this.Object.

**Service Layer Classes**

**ServiceFactory.java**

This provides methods that return the two implementations present in the service layer including the Search Service and the Rate Service.

**ISearchService.java**

This is the interface for the Search Service for the application. It includes the following 5 method stubs for implementing the Search Service:

* SearchQuery createQuery();
* SearchQuery updateQuery(String s, SearchQuery search);
* SearchResult getSearchResults(SearchQuery sq, List<Coaster> coasters);
* void writeCoasterToFile(Coaster c, String directoryPath);
* List<Coaster> loadAllCoasters(String coasterDirectoryPath);

**SearchServiceImpl.java**

This is the implementation of the ISearchService interface from which this class extends.

**IRateService.java**

This is the interface for the Rate Service for the application. It includes the following 3 method stubs for implementing the Rate Service:

* void addReview(Coaster c, CoasterReview review, String directoryPath);
* List<CoasterReview> getReviews(Coaster c);

UPDATE: User objects and related methods have been removed from this interface due to no authentication privileges needed for the scope of this build.

**RateServiceImpl.java**

This is the implementation of the IRateService interface from which this class extends.

**Business Layer Classes**

**CoasterManager.java**

This is the general manager facade for all managers to extend from in the business layer classes. It contains a singular method for getting services from the service factory.

**SearchManager.java**

This is the manager facade class for all things related to the Search Coaster use case which is the highest priority use case. It contains the following useful methods:

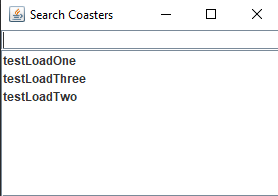
* void initializeSearch(String dataPath) - initializes the Search functionality to be used by the presentation layer including loading all coaster data from the desired dataPath.
* void updateSearch(String input) - takes a user input and updates the search by filtering the data based on input. Updates the query and result fields as necessary.

**RateManager.java**

This class will, in the future, be a manager facade class for the Rate Coaster use case which is the 2nd highest priority use case.

**Presentation Layer**

Version 1 of Search Use Case



Search functions as normal, with event handling after release of every keystroke to update the search results.

Rate Use Case coming soon.