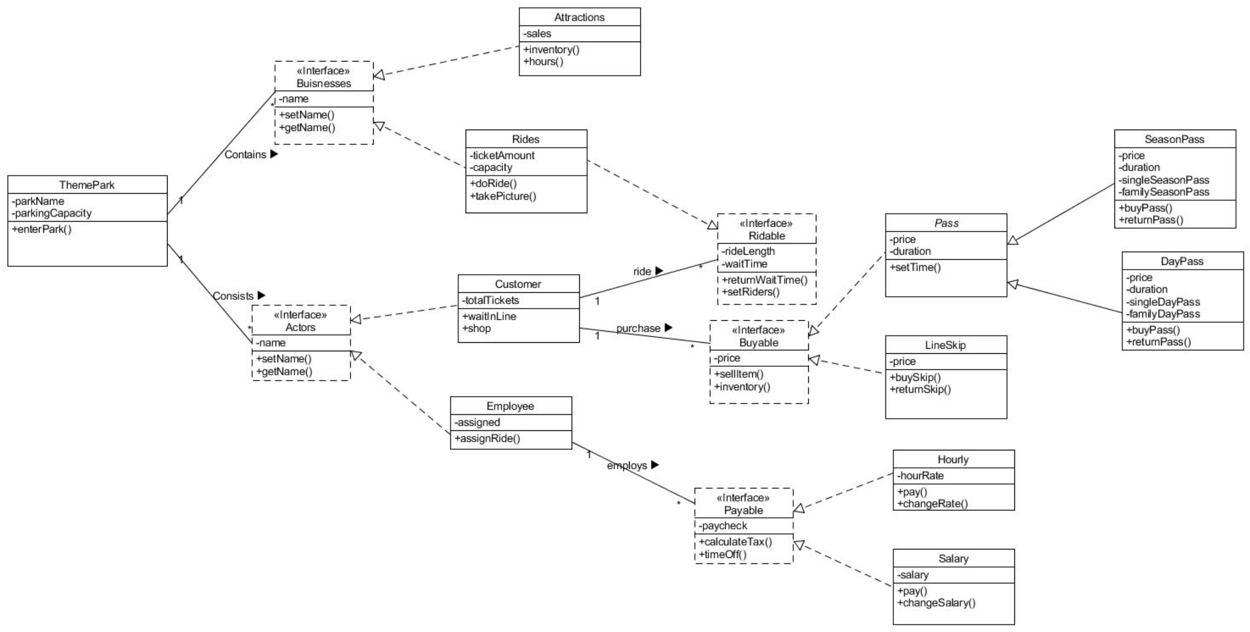
Jared Jamieson, Jordan Vaglivelo, Casey Brenner

Lab 8 - UML

March 13, 2020



Lab 8 Write Up

Interfaces:

We chose to make Businesses, Actors, Ridable, Buyable, and Payable into interfaces. We chose these attributes to be interfaces because of how general their scope is. There are many different things that could potentially implement these interfaces that are not related to ThemePark. For example a restaurant could use Buyable for food and gift cards, and any company could use Businesses.

Classes:

The first class we have is ThemePark and this is the class that holds everything in our theme park. We have an Attractions class that implements from Businesses.

Attractions represents shops, games, etc. (everything that is not a ride). The next class is Rides which implements Businesses and is implemented by Rideable. The Customer class is implemented by Actors and is associated with Rideable and Buyable. Each Customer holds N amount of Rideable and Buyable. The Employee class is implemented by Actors and is associated with Payable. The LineSkip class is implemented by Buyable and holds data relating to line skip passes. Hourly and Salary are implemented by Payable and deal with employee’s pay. SeasonPass and DayPass extend *Pass* and stores data based off of the type of pass.

Abstract Classes:

We chose to turn *Pass* into an abstract class. We chose this because Pass would most likely not be widely used. The interfaces we used can be applied to any other program because they are broad, but this class is specific to amusement parks.

Rejection:

Our first few designs for the Theme Park Lab were rejected for most of the same reasons. These were that the classes were too coupled to each other and could be more separated to allow for future edits. We implemented a few interfaces to separate some classes from interacting directly with other classes, so if we were to edit the behaviors of these classes after implementing, then we would edit the behavior classes instead of the classes relying on these behaviors directly. After that it was an issue of deciding the correct relationships to connect the classes and what methods and attributes they should contain.