* **[Instructions](https://teamtreehouse.com/projects/amusement-park-pass-generator-part-1" \l "instructions)**

For this project, you’ll create a system for creating and validating access passes to an amusement park. There are three types of people who can be granted access to the park: employees, vendors, and of course, guests. As you can imagine, these different groups are allowed to enter different areas of the park, and may or may not be permitted to do certain things, like ride the rides or receive discounts in certain eateries and shops, for example.

Within each category of park access, there should be several sub-categories with varying access rights. For example, guests can be “Classic”, “VIP”, or “Free Child”, with different privileges associated with each. Details on exactly what each type of entrant is permitted to do and what type of personal information is required from them are outlined in the Business Rules Matrix. The document can be downloaded in the Project Resources area.

This project is divided into two parts. Part 1, outlined here as Project 4, will focus on building the data structures, object definitions, logic, error handling and other plumbing for the app. No user interface will be built at this stage. Your code will be built, tested and run by using temporary hard-coded “plug” values (or test cases).

For Part 2, which is Project 5, you will construct the User Interface element and workflow. They are shown in the project mockups and Onward and Upward Instruction Video. You will also add features like data entry screen and pass generation. In addition, several additional types of entrants such as contracted employees, vendors, season pass holders and senior guests, will be added. Keep in mind that you’ll want to write the code for Part I so it can be reused and extended in Part 2. (You will probably need to refactor some code, regardless, but do keep this in mind.)

In creating your code, be sure to make use of optionals, enums, protocols, data structures (such as collections) and error handling. Also, remember you can use a combination of both inheritance and composition, depending on which is best suited for the particular feature. The majority of the tools, syntax and concepts needed to complete Part I have been covered in the courses so far. However the implementation of certain elements, like dates and extra credit items will require you to seek additional resources and documentation.

Just to be clear, the app you are creating would utilized by the staff of the amusement park at the entrance gates, inside their kiosk. Actual entrants wouldn’t see the screen. They will tell the staff their relevant information and staff members would type it on the iPad screen and generate the passes. The access levels of the passes will then be tested by simulated “swiping’ on the iPad. The completed app in Project 5 will not actually print any passes, but simply creates an image of the pass, as per the mockups. Please watch the Onward and Upward Instruction Video for a walk through of the app in order to get a full understanding.

* **Before you start:**
* Make sure you have watched the Onward and Upward Instruction video and the other files which can be found in Project Resources.
* Read the instructions and grading rubric carefully.
* Download the Business Rules Matrix and make sure you are familiar with the details and clear on what’s expected.
* Download and check out the mockups for the app. This will give you an idea of the finished app, even though creating the user interface won’t be required at this stage.
* Make sure you have carefully read these instructions and the grading rubric.
* Before writing any code, you should sketch out your ideas and object modeling with pencil and paper. Even if you change your initial plans or scrap them completely, this is a good exercise. Also, given all the business rules listed below, you may want to print out this document so you have it handy as a checklist.

## **Project Instructions**

To complete this project, follow the instructions below. If you get stuck, ask a question in the community.

[8 steps](https://teamtreehouse.com/projects/amusement-park-pass-generator-part-1)

#### You can download and use the empty Swift 3 starter files template to start your project.

#### Create the required data structures and related class/structs required by the app (for example, to represent the entrant types listed, as specified in the business rules matrix). Please make sure class, struct, protocol, inheritance, composition, and enums are utilized.

#### Create initializer methods for the classes or structs. These initializer methods will be used for the creation of test objects and test cases as outlined in the next step. Note: In Part II we will instead use these initializer methods to create entrants based on user input.

#### Create a set of possible (such as missing birthday, missing first/last name) and informative errors (such as the name of the object that caused the error, and the details of the error). Make sure they can be thrown at appropriate situations (e.g, when personal information is not complete). These errors must be caught and displayed (for Part I, these can be printed in the console).

#### Create at least one swipe+ method (or more if you prefer to have different swipe methods for rides, registers, restricted areas) which can assess an entrants right to access areas within the park, ride rides, skip ride lines, or receive discounts and shops and eateries. You can think of this as the logic which would reside at a checkpoint, such as ride turnstile, restricted door or cash register, in the case of discounts. Please note, it is not an “error” when a user is denied access to a ride or area. Your swipe method should handle this as a “failing” situation and print a suitable message to the console.

⁺NOTE: By swipe we mean a user swiping their pass at a ride or cash register. This has nothing to do with the swipe gesture found on mobile devices.

#### Create a set of test cases (plug values) for each valid entrant type, and include related actions they can perform (one example is to create a VIP entrant, and have him/her swipe to get into different part of the park, and try to get discounts, etc). For at least some entrant types, deliberately have missing information such that errors will be triggered and handled. Once finished, you can comment out the various test cases so that reviewers can uncomment them and run your code for and test the various scenarios for different types of entrants.

#### It is also important to take a step back and think about how all these real world objects (e.g. entrants, passes) can be modeled in Swift using object oriented facilities (e.g. protocols and inheritance), how the objects modelling can be made flexible and versatile by leveraging object oriented techniques (hint: may be the swipe method could be a polymorphic method?), how type definitions can be made flexible using enums, and lastly, how the overall design in Part 1 can lead to the use of the MVC (Model-View-Controller) paradigm when the UI is created in Part 2.

#### As the projects are getting more involved and complex, the best way to ask questions, get clarification, and bounce ideas is the Treehouse Techdegree iOS Slack channel. Make sure you leverage this very useful resource. If you need the invite again, please email [help@teamtreehouse.com](mailto:help@teamtreehouse.com) and we’ll get you setup.

## **Extra Credit**

To get an "exceeds" rating, you can expand on the project in the following ways:

[5 steps](https://teamtreehouse.com/projects/amusement-park-pass-generator-part-1)

* **NOTE:** In order to get an overall Exceed grade for the project, you must get an Exceed in ALL of the items where an Exceed can be achieved, including ALL of the following features

#### Where it make sense (there are a few candidates in this project), utilize associated values when using Enums

#### When processing a swipe (or part of the swipe processing flow) polymorphic methods are being implemented, such that a method is able to handle (or to be applied to) multiple entrant or pass types

#### Add a feature to the swipe method(s), so that when an entrant swipes on their birthday, they receive a personalized message, if their birthday is known.

#### Add a feature to prevent a guest from swiping into the same ride twice in row within 5 seconds at the same checkpoint. This is to prevent someone from sneaking in a second person. Rejecting an entrant’s second swipe should be done gracefully with an alert message.