**Project Brief**

1. Objective

**Overview: Hotel Reservation Application**

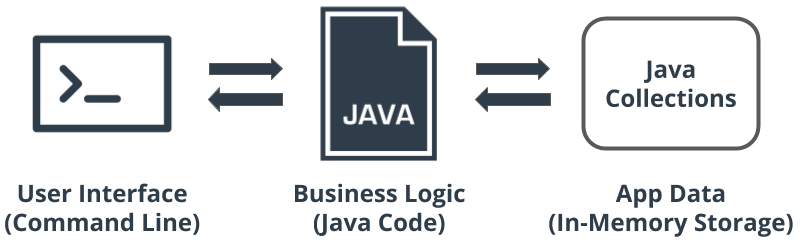
The purpose of this project is to design and implement a Java hotel reservation application. This project is one of three required projects in the Udacity Nanodegree course known as Java Programming

The hotel reservation application will allow customers to find and book a hotel room based on room availability. This project will demonstrate your abilities to design classes using OOP, organize and process data with collections, and use common Java types.

**Main Components of the App**

The major components of the Hotel Reservation Application will consist of the following:

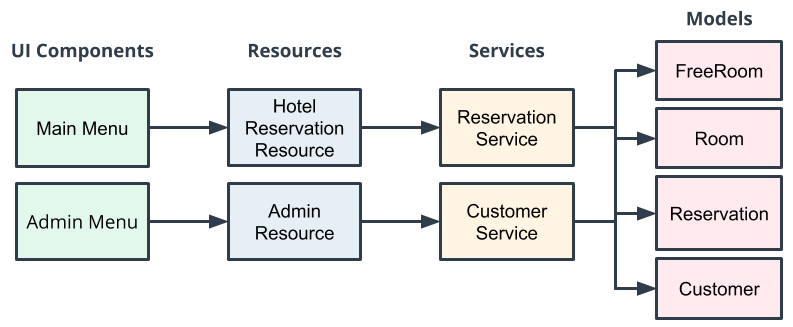
1. **CLI for the User Interface.** We'll use the *Command Line Interface* (or *CLI* for the user interface. For this, we'll need to have Java monitor the CLI for user input, so the user can enter commands to search for available rooms, book rooms, and so on.
2. **Java code.** The second main component is the Java code itself—this is where we add our business logic for the app.
3. **Java collections.** Finally, we'll use *Java collections* for in-memory storage of the data we need for the app, such as the users' names, room availability, and so on.



**Application Architecture**

Let's talk about the structure or architecture of the application. The app will be separated into the following layers:

1. **User interface (UI)**, including a *main menu* for the users who want to book a room, and an *admin menu* for administrative functions.
2. **Resources** will act as our Application Programming Interface (API) to our UI.
3. **Services** will communicate with our resources, and each other, to build the business logic necessary to provide feedback to our UI.
4. **Data models** will be used to represent the domain that we're using within the system (e.g., rooms, reservations, and customers).



**Layers**

The architecture use layers to support *modularization* and *decoupling*. For example, if we later decided to change our UI components to a webpage instead of a command-line interface, layering would support this.

Layering ensures there is no cross-communication calls from one layer to another. For example, a UI component should never communicate directly with a service. This would expose the service implementation to the UI and make it difficult for us to change it out later.