




High-Level Program Design

---

# Java Developer Immersive

# Java Developer Immersive

 Instructor-Led  Onsite or Remote  ~480 Hours

## Overview:

Expand your Java development workforce with experiential training in the most in-demand skills. Participants will build core skills in Java, Spring Boot, test-driven development, troubleshooting, and Agile development.

## Business Outcomes:

- Build engineering capacity to tackle enterprise technology projects.
- Transform junior talent into job-ready Java developers.
- Solve real engineering problems through capstone projects.

## Prerequisites:

Students should have comfort with another programming language such as Python, Ruby, or JavaScript.

### ~480 Hours of Java Developer Training

Foundational Java	Spring Boot	Expand into DevOps	Cloud Infrastructure	Capstone Project
<b>Understand foundational Java topics</b> and apply design patterns to write maintainable object-oriented code.	<b>Leverage the popular framework Spring Boot</b> to spend less time configuring code and more time writing logic.	<b>Extend Agile learnings into DevOps</b> with microservice architecture, containerization, and continuous integration.	<b>Explore the benefits of using the cloud</b> to host application infrastructure.	<b>Build, test, and deploy</b> a microservices application for your final project.

# Why Java Developer Immersive (JDI)?

- **Validated tools and approaches** for Java and Spring Boot development in an Agile environment, created in partnership with top organizations.
- **Built with subject matter experts** with experience in Java, Spring Boot, cloud-native application development, test-driven development, and more.
- 480 hours (12 weeks) of expert-led, hands-on learning, including:
  - **Projects, labs, and assignments** that mimic real-world workflows.
  - **Examples** that demonstrate how businesses use the concepts.
  - **Regular feedback and touchpoints** with instructors and peers to ensure students meeting learning goals.



# Learner Personas for JDI

This product is specifically designed for the following audience:

- **Early-career or junior software engineers** who are well suited to work at an enterprise-level business where Java, DevOps, and/or building applications for Cloud is a technical requirement.
- Students should have **1-2 years of object-oriented programming experience** in a language like Ruby, Python, or JavaScript.

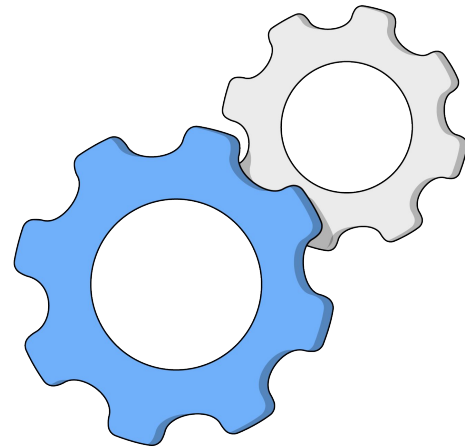


# Projects in JDI

Students complete a project at the end of each unit, which culminate in a capstone project in which students apply everything they've learned in the course.

Each project builds on the previous project as students add more complexity and features to their applications based on what they learn in each unit.

Projects are completed in pair programming.



# Anatomy of Java Developer Immersive

Unit	What's Covered	Unit Project
<b>Unit 1:</b> <b>Foundational Java</b> <b>(3 weeks)</b> Apply Java design patterns to write maintainable object-oriented code.	<ul style="list-style-type: none"><li>• Java programming syntax and basics</li><li>• Object-oriented programming</li><li>• Functional programming</li><li>• Agile and extreme programming methodologies</li></ul>	Create a version of a rock–paper–scissors game that allows users to play against the computer in the console. The game should consist of a few main features: <ul style="list-style-type: none"><li>• Play rock–paper–scissors against a computer player.</li><li>• Play rock–paper–scissors against a human player.</li></ul>
<b>Unit 2:</b> <b>Building Applications With Spring Boot</b> <b>(3 weeks)</b> Leverage the Spring Boot framework to spend less time configuring code and more time writing business logic for applications.	<ul style="list-style-type: none"><li>• Spring Boot</li><li>• Spring Data with Postgres</li><li>• Spring Testing</li><li>• Spring Security</li><li>• SQL and relational databases</li><li>• Spring design patterns</li></ul>	In pairs, build a back-end for a Reddit-like application, where a user can create a post that will show up on every user's feed and on which any user can comment. Use Spring Boot to create a monolithic back-end that exposes APIs and work in PostgreSQL to create a database. Demonstrate the ability to build a back-end application using Spring Boot, Hibernate or JDBC, and PostgreSQL; document how a database works; and use Tomcat to run an app locally.

# Anatomy of Java Developer Immersive

Unit	What's Covered	Unit Project
<b>Unit 3:</b> <b>Microservices Architecture</b> <b>(2 weeks)</b> Develop microservices and describe the benefits of microservice architecture.	<ul style="list-style-type: none"><li>• Microservice architecture</li><li>• Docker</li><li>• Domain-driven design</li><li>• NoSQL and MongoDB</li></ul>	In pairs, create a new microservices application using the monolithic application built in Unit 2 as reference. Use Spring Boot with Hibernate or JDBC to create multiple microservices that expose APIs, and demonstrate proficiency with a relational database. Demonstrate the ability to create microservices and, more broadly, understand when to choose a microservice over a monolith. Deploy the app on the cloud using AWS.
<b>Unit 4:</b> <b>DevOps and Cloud Infrastructure</b> <b>(2.5 weeks)</b> Extend Agile learnings into DevOps and describe the benefits of using the cloud to host application infrastructure.	<ul style="list-style-type: none"><li>• Kubernetes</li><li>• Jenkins</li><li>• Twelve-factor design</li><li>• Cloud infrastructure</li><li>• Deploying apps on a PaaS</li><li>• RabbitMQ</li><li>• Swagger</li><li>• Application monitoring</li></ul>	No unit project; students apply learnings in the Capstone.

# Anatomy of Java Developer Immersive

Unit	What's Covered	Unit Project
<b>Capstone Project and Interview Prep (1.5 weeks)</b>	<ul style="list-style-type: none"><li>• Capstone project</li><li>• Computer science foundations: data structures and algorithms</li><li>• Interview prep</li></ul>	<p>In pairs, add more complexity to the full-stack application built in Unit 3 by integrating email notifications to a user when another user comments on their post. Have the application fully tested. Demonstrate the ability to implement messaging queues, incorporate unit tests with mocking, work with different databases, configure the build, and deploy an app on AWS.</p>



# Tools Used in the Course



JDK 8



The course is written for MacOS; if students or instructors use a Windows or Linux system, some modifications to the curriculum may have to be made.



Git and Github Enterprise



IntelliJ IDEA Community Edition



Zoom, Slack, and Google Chrome

Java Developer Immersive



# What You'll Learn



## Foundational Java

3 weeks



### Why?

Java is the most popular coding language among engineers and is popular with enterprises due to its robustness, ease of use, multi-platform capabilities, and security features. In this unit, students will dive into the basics of Java, including programming basics, OOP, and functional programming.

### Topics Covered

- Data types and variables
- Control flow
- Methods and scope
- Data collections
- Debugging and exception handling
- JUnit
- Classes, subclasses, and inheritance
- Abstract classes and interfaces
- Lambdas and stream processing
- Functional interfaces
- Agile methodologies
- Extreme programming



# Hello World in Java

We'll complete this activity using a text editor and the command line.

Let's start with the all-time classic function, `Hello World`. To begin, create a file called `HelloWorld.java` and save it.

All Java files must be defined as a `class`, so let's begin with a `class` definition. This `class` definition must match the name of the file, so we'll call ours `HelloWorld`:

```
public class HelloWorld {  
}
```

Then, all Java programs require a `main` method representing the entry point of the program. This method will automatically be invoked when we run our Java file. The following function must be placed **inside** the `class` definition:

```
public static void main(String[] args) {  
}
```

What's going on here?

## `public`

First, the `public` keyword declares this method to be available anywhere. On the other hand, a `private` method is only available within the `class`. You can find more details [here](#).

## `static`

Next, the `static` keyword indicates that this method belongs to the `class` itself. The opposite of a `static` method would be an `instance` method, where the method belongs to the objects the `class` creates. To run an instance method, you would have to create a new instance of the `class` with the `new` keyword and use it to run that method.

# Applying Inheritance

In this lab, you'll be designing an app that creates and maintains households. All households earn income and pay 20% of their income in taxes. All households also have a pet. All pets can be fed, groomed, and played with.

You can express the state or action of something by printing messages to the command line, such as "Household sells X to gain income" or "Play fetch."

---

## Exercise

### Requirements

- Create at least one abstract class.
- Use at least one interface.
- Create at least two households, each with a different kind of pet.
- Use polymorphism to have each household earn income, pay its taxes, and care for its pet (feed, play with, and groom) in one loop.

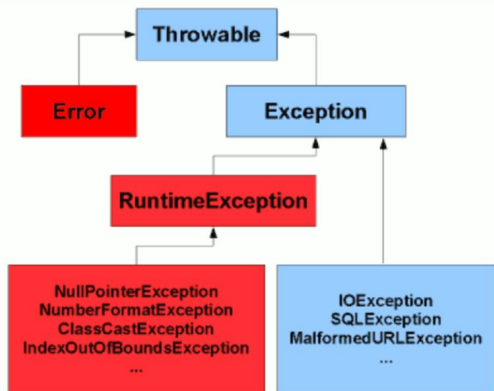
**Bonus:** Add additional classes or subclasses.

# The Basics of Exceptions

Before we start talking about the `try-catch` block, we need to talk about exceptions. **Exceptions** are events that occur while a program is running and interrupt the normal flow of the code. These can be null-pointer exceptions, divide-by-zero exceptions, index-out-of-bounds exceptions, and more. You can review many of the built-in exceptions in the [Java documentation](#).

There are two types of exceptions: checked and unchecked. A **checked exception** occurs at compile time, which means a programmer is forced to handle these exceptions; otherwise, the program won't compile. Checked exceptions are subclasses of the `Exception` class. An **unchecked exception** — also known as a **runtime exception** — occurs at the time a program executes. You don't have to handle them, but you can if you'd like.

Below is an illustration of the exception hierarchy. Red denotes unchecked exceptions, while blue denotes checked ones:



[Source](#)

# Project 1: Rock, Paper, Scissors in Java

## Overview

It's time to put your Java knowledge to use!

Create a version of a [rock-paper-scissors game](#) that allows users to play against the computer in the console. The game should consist of a few main features:

- Play rock-paper-scissors against a computer player.
- Play rock-paper-scissors against a human player.

## Project Requirements

### Feature Requirements

Your game must:

- Have a main menu with options to enter `2 players` or `vs. computer`.
- If the user enters `2 players`, they should be able to play rock-paper-scissors against a human competitor.
- If the user enters `vs. computer`, they should be able to play against the computer.
- When the game is over, the winner should be declared.

Hint: Use a [random number generator](#) to make the computer's choice.



## Building Applications With Spring Boot

### Why?

Development frameworks like Spring Boot allow developers to spend less time dealing with configuration and troubleshooting, and more time thinking about the logic and architecture of an application. In this unit, students will build applications using Spring Boot and its related projects.

### Topics Covered

- SQL
- Relational databases
- The Spring framework
- Spring Boot
- Spring Data with PostgreSQL
- Spring Security
- Spring Testing
- Design patterns: MVC, inversion of control, dependency injection, Singleton, and more
- Development tools: Maven, Gradle, Tomcat, and Postman





# Introduction to SQL

## What Is SQL?

SQL stands for **Structured Query Language**, and it's a language that's universally used and adapted to interact with relational databases. When you use a SQL client and connect to a relational database that contains tables with data, the scope of what you can do with SQL commands includes:

- Inserting data.
- Querying or retrieving data.
- Updating or deleting data.
- Creating new tables and entire databases.
- Controlling permissions of who can access to our data.

Note that all of these actions depend on what the database administrator sets for user permissions. If you're an analyst, for example, you'll only have access to retrieving company data. But, as a developer, you could access all of these commands and be in charge of setting the database permissions for your web or mobile application.

## Why Is SQL Important?

A database is just a repository for storing data, and you need to use systems to dictate how that data will be stored and how clients will interact with it. We call these systems **database management systems**, and they come in *many* forms:

- MySQL
- SQLite
- PostgreSQL (what we'll be using!)

All of these management systems use SQL (or some adaptation of it) as a language for managing data in the system.

# Creating an API With Spring Boot

Now that our basic set up is done and our app is running, let's create a simple route.

Spring Boot, like many web frameworks, is built on top of the ubiquitous MVC pattern. APIs built in Spring Boot need to model and persist data from a database and receive and respond to client requests.

## The Controller

The first thing we need is a Controller. We need controllers to manage our traffic. This is where we define our routes.

Let's create a new `UserController` to start serving up our `User` data. To get started, we'll need to create a new `controller` package. Inside of this new package, we'll create a class called `UserController.java`.

Spring Boot Controllers are simply Java classes with an annotation at the top that adds special functionality. In our `UserController`, let's set up a new class with the `@RestController` annotation.

### UserController

```
package com.example.springbootmonolith.controller

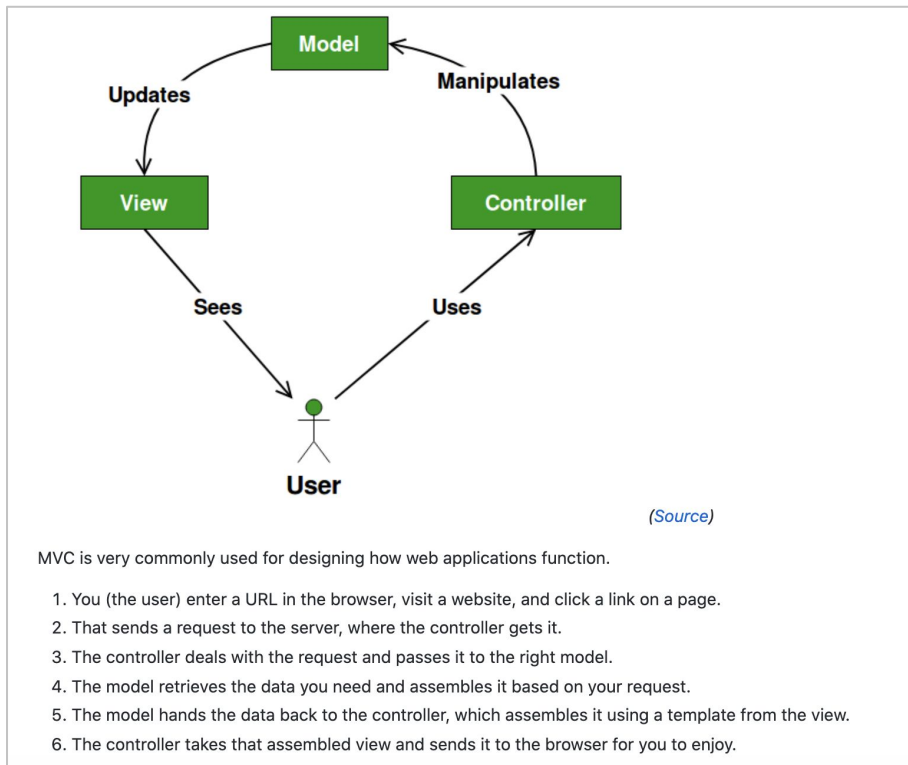
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

}
```

Tip: To automatically import the package containing `RestController`, place the cursor on it and Press `⌘ ↵` to accept the suggestion.

# The MVC Pattern



# Project 2: Build a Monolithic Application in Spring Boot

In pairs, build a back-end for a Reddit-like application, where a user — once logged in — can create a post that will show up on every user's feed and on which any user can comment.

You will use Spring Boot to create a monolithic back-end application that exposes APIs. You will work in PostgreSQL to create your database.

You will demonstrate your ability to build a back-end application using Spring Boot, Hibernate or JDBC, and PostgreSQL; document how a database works; and use Tomcat to run your app locally. You'll also demonstrate your ability to successfully program in pairs.

## Project Requirements

The application must allow a user to:

- Create an account.
- Log-in.
- Update their profile.
- Create and delete their own posts.
- View and comment on others' posts.

Your app must:

- Have at least 80 percent unit test coverage.
- Persist at least three models (User, Post, Comment) to a PostgreSQL database.
- Expose APIs for users to perform login, sign-up, and profile updates.
- Expose APIs for users to perform CRUD on posts.
- Expose APIs for users to create a comment on a post by another user.
- Expose CRUD routes that were built using the REST convention.
- Stick with the KISS (keep it simple, stupid) and DRY (don't repeat yourself) principles.



## Microservices Architecture

### Why?

In a microservice architecture, even the most complex applications can be delivered quickly and reliably by being built as loosely coupled individual services. In this unit, students will use Docker to develop microservices, as well as explore the tradeoffs between microservices and monolithic architecture.

### Topics Covered

- NoSQL and MongoDB
- Spring Data with MongoDB
- Monoliths vs. microservices: pros and cons
- Microservices architecture patterns: timeouts, circuit breaker, and bulkhead
- Docker
- Domain-driven design
- API gateway and service registry with Eureka and Zuul



# How Docker Works

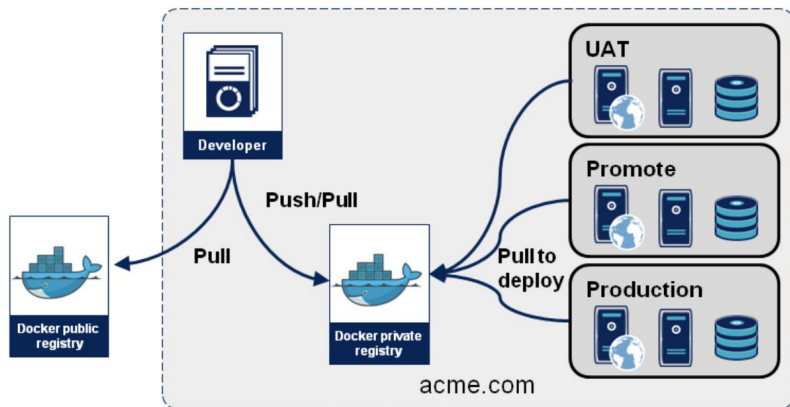
Docker is an open-source platform (the most popular one) for building applications using containers.

However, Docker isn't a completely new technology. Many of the components and principles existed previously. Docker is written in Go and takes advantage of several features of the Linux kernel to deliver its functionality, including namespaces and cgroups (more on those in the Additional Resources section).

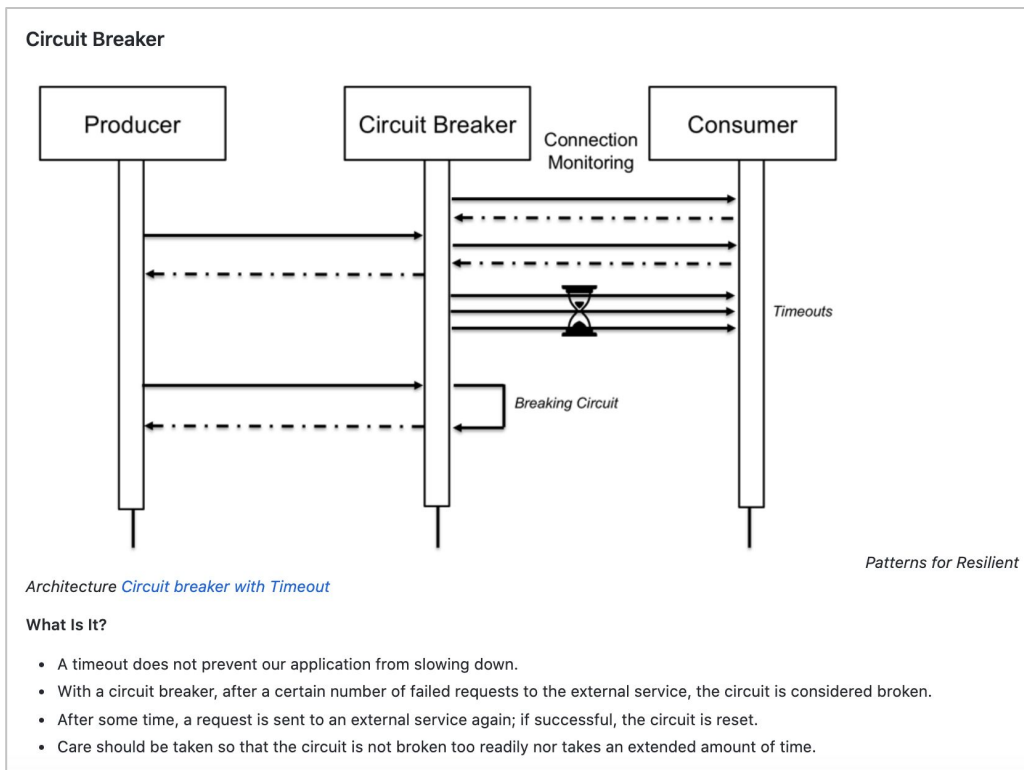
The ultimate goal of Docker is to mirror our dev environment with our production environment. This is mostly useful for your back-end but can be applied to your front-end applications as well! It's cool to run Docker locally, but its real benefit comes into play while running it on some production machine.

Take a look at this very simplified version of how Docker works.

**Knowledge Check:** Can someone explain what's going on here?



# Avoiding Failure 101



# Building a Microservice Application

## Requirements

Use the `starter-code` directory to grab the starter code for the lab.

Set up a basic service that returns some songs at [localhost:8080/songs](http://localhost:8080/songs).

Remember:

- The microservice will be a basic Spring Boot REST application, with some extra dependencies and annotations that tell it to register with our service registry (Eureka).
- You'll need to allocate a new port for each service to run on. The API gateway is already set to run on `8080`.
- We'll want the app to register at the `/songs` URL through the Zuul API gateway.

## Adding Song Functionality

Once the service is wired up to our API gateway and service registry, we'll want to add some functionality to each. Both the `User` and `Song` microservices will share the same database.

2. Song service should have Flyway migrations that add `SONGS` table to the database.
  - The `SONGS` table should have at least `title` and `length` (in milliseconds) columns.
3. Song service should be able to perform CRUD (create, read, update, and delete data).

The only catch to adding Flyway migration file is, as you know all the services by docker are run parallelly so there is no way for Flyway to know which migration file to run first. That can really mess up it's head. There is no elegant way to combat this issue. So what you can do is put all migration files in any one service starting with different versions.



# Project 3: Build a Microservice Application in Spring Boot

## Overview

In pairs, create a new microservices application using the monolithic application built in Project 2 as reference.

You will use Spring Boot with Hibernate or JDBC to create multiple microservices that expose APIs. You'll demonstrate proficiency with a relational database. You will also demonstrate your ability to create microservices and, more broadly, understand when to choose a microservice over a monolith.

You will deploy your app on the cloud using AWS. Additionally, you'll demonstrate proficiency in pair programming. For this project, you will exchange pairs. One original partner will remain with the previously built monolith and one will move to another project. The student new to the previously built monolith will need to demonstrate competence in working with an existing codebase.

## Project Requirements

Your app must:

- Have three different microservices for User, Post, and Comment.
- Have PostgreSQL database(s) set up.
- Be deployed on AWS.
- Stick with the KISS (keep it simple, stupid) principle.



## DevOps and Cloud Infrastructure

### Why?

More and more enterprises are relying on cloud infrastructure and DevOps techniques to deploy and maintain software efficiently, as well as to scale its delivery to customers as needed. In this unit, students will learn key cloud and DevOps concepts and tools that they'll use to manage their applications.

### Topics Covered

- Kubernetes
- Basics of DevOps, continuous integration, and continuous delivery
- Jenkins
- Swagger
- 12 factor application design
- Cloud infrastructure and service models
- Deployment on AWS
- Monitoring application logs with ELK
- Systems design patterns
- RabbitMQ



# Creating a Jenkins Pipeline

A [pipeline](#) is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins.

To start, we will build a sample Java application with Maven.

Fork [this repository](#) into your own GitHub account and clone the code.

Once the application is set up:

1. Go back to Jenkins homepage <http://localhost:8080> and click **create new jobs** under **Welcome to Jenkins!**.
2. Specify a name for your pipeline (e.g. `simple-java-maven-app`).
3. Scroll down and click **Pipeline**, then click OK at the end of the page.
4. Scroll down to the Pipeline section and choose **Pipeline script from SCM** option.
5. From the **SCM** field, choose **Git**.
6. In the **Repository URL** field, specify the directory path where you locally cloned the `simple-java-maven-app` repository.
7. Click **Save**.

After creating a pipeline project, follow the steps below to create a Jenkinsfile and automate building the `simple-java-maven-app` application. This file will then be checked-in and committed to our locally cloned Git repository. In this way, our pipeline becomes part of the application and is reviewed and versioned like any other code in the application. Refer to the [Using the Jenkinsfile](#) section for more information.

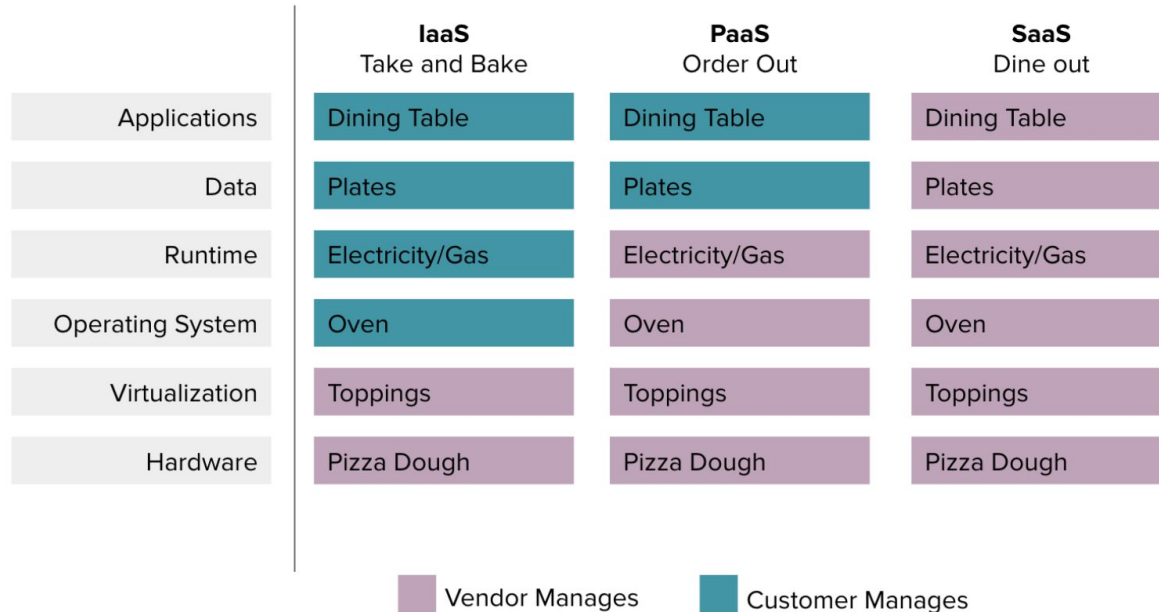
1. Open terminal and create a new file `Jenkinsfile` at the root of the `simple-java-maven-app` repository.

```
touch Jenkinsfile
```

2. Copy the following code and paste it into the empty Jenkinsfile.

# IaaS, PaaS, SaaS: What's the Difference?

Let's dive into each of the three service models: IaaS, PaaS, and SaaS.



# Deploying an Application on AWS

## The Deployment Process

### Sign Up for an AWS Account

You should already have created an AWS account; if not, sign up [here](#).

### Create Amazon EC2 Key Pairs

You'll need these to SSH into your EC2 instance.

**Secure Shell (SSH)** is a cryptographic network protocol for operating network services securely over an unsecured network. Typical applications include remote command line, login, and remote command execution, but any network service can be secured with SSH.

To create the SSH, follow the instructions [here](#).

### Launch an Instance

Step-by-step instructions are provided [here](#).

A couple of things to look out for:

- Make sure that "type" is Amazon Linux AML.
- The configuration should be marked as "free tier eligible."
- The instance type should be "micro."



## Capstone Project and Interview Prep

### Why?

In the final weeks of the course, everything will come together for students as they work to complete a capstone project and prepare for software engineering job interviews.

Students will work in pairs to complete their capstone project with the expert guidance and support of their instructional team.

They will also explore core computer science topics, including data structure and algorithms, and practice the whiteboarding challenges that they are likely to encounter while interviewing.



# Capstone Project

## Overview

In pairs, add more complexity to the full-stack application built in Project 3 by integrating email notifications to a user when another user comments on their post. Have the application fully tested.

This project builds on the work you completed in Project 3. Through this project, you will demonstrate your ability to implement messaging queues, incorporate unit tests with mocking, work with different databases, configure the build, and deploy your app on AWS. As with the other projects in this course, this will require proficiency in pair programming.

## Project Requirements

Your app must:

- Use the SQL database we covered in class.
- Be a complete product with multiple relationships and CRUD functionality for all three models.
- Have a messaging queue using RabbitMQ to send email notifications to a user if another user comments on their post.
- Be bootstrapped with Spring Boot.
- Be monitored using tools covered in class.
- Have mocked unit tests and integration tests incorporated with the build process using Jenkins.
- Have test coverage (SonarQube) greater than 80 percent.
- Be deployed on AWS.
- Have Swagger documentation of all the RESTful routes in their app.
- Stick with the KISS (keep it simple, stupid) principle.

# Interview Prep

## Introduction

Whiteboarding is a common practice in technical interviews. The goal is to convey how you think through a challenging problem, and what steps you'd take to get started. You don't need to arrive at a solution or write perfect code. In fact, think of it as writing detailed pseudocode.

## Instructions

Each group member will take turns being the candidate (the one whiteboarding) and the interviewers (those reading and assessing the prompt).

If you're the candidate, stand in front of the whiteboard without looking at your prompt. One of the interviewers should read out the prompt and set a timer for 10 minutes. The candidate should work through the prompt until the timer goes off. If you feel like you've solved the problem before the timer goes off, start to talk and think through ways to make your code faster or more efficient.

## Candidate Instructions

The interviewer will read out the prompt. Ask clarifying questions if something seems unclear, then start whiteboarding. It's a good idea to take notes about the prompt on the whiteboard as the interviewer reads it. Sometimes, the questions are intentionally vague, and the interviewers want to see how you deal with that, so **ask questions**.

As you work on the solution, talk it out. The goal is not to arrive at the correct answer but rather to convey how you would approach solving the problem. If you get stuck, ask if you can do a quick Google search. Don't search for the answer to the problem, but rather for the small piece of information you need. Note that your interviewers have every right to deny your request.

When the timer goes off, take a seat in front of your interviewers so you can debrief.



