# Introduction to spring framework

Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

Examples of how you, as an application developer, can use the Spring platform advantage:

* Make a Java method execute in a database transaction without having to deal with transaction APIs.
* Make a local Java method a remote procedure without having to deal with remote APIs.
* Make a local Java method a management operation without having to deal with JMX APIs.
* Make a local Java method a message handler without having to deal with JMS APIs.

# Basic tools and frameworks in JUnit

JUnit tutorial provides basic and advanced concepts of **unit testing in java** with examples. Our junit tutorial is designed for beginners and professionals.

It is an *open-source testing framework* for java programmers. The java programmer can create test cases and test his/her own code.

It is one of the unit testing framework. Current version is junit 4.

To perform unit testing, we need to create test cases. The **unit test case** is a code which ensures that the program logic works as expected.

# Basic tools and frameworks in Mockito

The mocking technique is not only used in Java but also used in any object-oriented programming language. There are many frameworks available in Java for mocking, but Mockito is the most popular framework among them.

To mock objects, you need to understand the three key concepts of mocking, i.e., stub, fake, and mock. Some of the unit tests involve only stubs, whereas some involve fake and mocks.

The brief description of the mocking concepts is given below:

1. **Stub:** Stub objects hold predefined data and provide it to answer the calls during testing. They are referred to as a dummy object with a minimum number of methods required for a test. It also provides methods to verify other methods used to access the internal state of a stub, when necessary. Stub object is generally used for **state verification**.
2. **Fake:** Fake are the objects that contain working implementations but are different from the production one. Mostly it takes shortcuts and also contains the simplified version of the production code.
3. **Mock:** Mock objects act as a dummy or clone of the real object in testing. They are generally created by an open-source library or a mocking framework like Mockito, Easy Mock, etc. Mock objects are typically used for **behavior verification**.

# Unit Testing with Spring Framework

Dependency injection should make your code less dependent on the container than it would be with traditional Java EE development. The POJOs that make up your application should be testable in JUnit or Testing tests, with objects instantiated by using the new operator, without Spring or any other container. You can use mock objects (in conjunction with other valuable testing techniques) to test your code in isolation. If you follow the architecture recommendations for Spring, the resulting clean layering and componentization of your codebase facilitate easier unit testing. For example, you can test service layer objects by stubbing or mocking DAO or repository interfaces, without needing to access persistent data while running unit tests.

True unit tests typically run extremely quickly, as there is no runtime infrastructure to set up. Emphasizing true unit tests as part of your development methodology can boost your productivity. You may not need this section of the testing chapter to help you write effective unit tests for your IoC-based applications. For certain unit testing scenarios, however, the Spring Framework provides mock objects and testing support classes.

# Spring AOP

**Aspect Oriented Programming** (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

It provides the pluggable way to dynamically add the additional concern before, after or around the actual logic. Suppose there are 10 methods in a class as given below:

6.3M

There are 5 methods that starts from m, 2 methods that starts from n and 3 methods that starts from p.

**Understanding Scenario,** I have to maintain log and send notification after calling methods that starts from m.

**Problem without AOP** We can call methods (that maintains log and sends notification) from the methods starting with m. In such scenario, we need to write the code in all the 5 methods.

But, if client says in future, I don't have to send notification, you need to change all the methods. It leads to the maintenance problem.

**Solution with AOP** We don't have to call methods from the method. Now we can define the additional concern like maintaining log, sending notification etc. in the method of a class. Its entry is given in the xml file.

In future, if client says to remove the notifier functionality, we need to change only in the xml file. So, maintenance is easy in AOP.

AOP is mostly used in following cases:

* to provide declarative enterprise services such as declarative transaction management.
* It allows users to implement custom aspects.

# Spring JDBC

The main tool that Spring JDBC uses for querying is the JdbcTemplate. The downside of using this is that it only provides the connection, everything else you need to do yourself. If you search for objects, you will need to map the results to Java objects by implementing a RowMapper. You will also need to do the exception handling by creating a ExceptionTranslator.

I will show you a simple example of how a query is created.

# Spring data JDBC

Spring Data JDBC uses a syntax that is comparable to Spring Data JPA. The biggest differences are under the hood. The management of the persistence is handled by the repository like in Spring Data JPA, but only the aggregate root has a repository. This means that if you want to insert or update data, the entire aggregate needs to be saved. You will need to call the save method of the repository of the aggregate root and this will first save the aggregate root and then all of the referenced entities get saved. If you want to insert only a part of an aggregate, for example only create a new Rental, then the whole aggregate will be updated and the referenced entities will be deleted and inserted again.

# SPRING DATA JPA

When you use the Spring Data framework, it will help you with building your queries and fetching the right data. The Spring Data JPA framework uses implementations of the JPA specifications like Hibernate. They make it possible to query the database using user friendly interfaces. When you want to query the database, instead of writing the entire query yourself, Hibernate will help you. There are multiple ways to query the database using Spring Data JPA, but they all need you to extend the repository of the entity you want to query.

Some basic queries can be written using derived queries. An example of this is findById. For these methods Spring Data will generate the SQL entirely on its own.

For example paging and sorting can be done by simply adding a parameter.

# Basic tools and Frameworks in Maven

When we develop a Java software project, such as a microservice, the major dependencies we need are:

* Spring MVC framework.
* Spring framework
* Hibernate

In addition to these 3 direct dependencies, we need to include their individual dependencies as well. So we need the dependencies of Spring MVC, Spring and Hibernate.

Let’s consider an example: REST API typically returns JSON responses. Spring MVC needs to convert Java Bean to JSON. It makes use of the Jackson framework to do the conversion. Jackson framework is a dependency of Spring MVC.

# Dependency Management With Maven

## Identifying Artifacts

In order to describe each of the artifacts stored here, Maven uses two tags - and. In addition to that, there is also a version that is maintained for each artifact. using these 3 identifiers, Maven can retrieve the correct version of the required dependency.

## Managing Transitive Dependencies

When Maven downloads a particular artifact, it also downloads those dependencies of this artifact that are needed in the project. These are known as **transitive dependencies**. This reduces the amount of effort that is spent on enumerating all the artifacts.

## Maven Does A Lot More!

The truth is in the Java world, Maven does a lot more than just dependency management.

## Building Deployable unit - Jars and Wars

Suppose we want to deploy an application to a different environment. We do not want to take the source code and build it again!

We can create an application deployable unit JAR/WAR/EAR and use it in other environments. Maven enables us to create application deployable unit in a simple way.

This entire sequence is called a **build** process, thereby making Maven a **build tool** as well.