**SPRING FRAMEWORK AND SPRING BOOT**

Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as [Struts](https://www.javatpoint.com/struts-2-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), Tapestry, [EJB](https://www.javatpoint.com/ejb-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc. We will learn these modules in next page. Let's understand the IOC and Dependency Injection first.

**Advantages of Spring Framework**

There are many advantages of Spring Framework. They are as follows:

1) **Predefined Templates**

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

**2) Loose Coupling**

The Spring applications are loosely coupled because of dependency injection.

**3) Easy to test**

The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

**4) Lightweight**

Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.

**5) Fast Development**

The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

**6) Powerful abstraction**

It provides powerful abstraction to JavaEE specifications such as [JMS](https://www.javatpoint.com/jms-tutorial), [JDBC](https://www.javatpoint.com/java-jdbc), JPA and JTA.

**7) Declarative support**

It provides declarative support for caching, validation, transactions and formatting.

## **Steps to create spring application**

Simple steps to create the first spring application. To run this application, we are not using any IDE. We are simply using the command prompt. Let's see the simple steps to create the spring application

* **create the class**
* **create the xml file to provide the values**
* **create the test class**
* **Load the spring jar files**
* **Run the test class**

## **Steps to create spring application using Spring Initlizr**

**Spring Initializr** is a **web-based tool** provided by the Pivotal Web Service. With the help of **Spring Initializr**, we can easily generate the structure of the **Spring Boot Project**. It offers extensible API for creating JVM-based projects

1. Navigate to [https://start.spring.io](https://start.spring.io/). This service pulls in all the dependencies you need for an application and does most of the setup for you.
2. Choose either Gradle or Maven and the language you want to use. This guide assumes that you chose Java.
3. Click **Dependencies** and select **Spring Web**.
4. Click **Generate**.
5. Download the resulting ZIP file, which is an archive of a web application that is configured with your choices.

# **Spring JdbcTemplate**

Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API

## **Problems of JDBC API**

The problems of JDBC API are as follows:

* We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
* We need to perform exception handling code on the database logic.
* We need to handle transaction.
* Repetition of all these codes from one to another database logic is a time consuming task.

## **Advantage of Spring JdbcTemplate**

Spring JdbcTemplate eliminates all the above mentioned problems of JDBC API. It provides you methods to write the queries directly, so it saves a lot of work and time.

## **Spring Jdbc Approaches**

Spring framework provides following approaches for JDBC database access:

* JdbcTemplate
* NamedParameterJdbcTemplate
* SimpleJdbcTemplate
* SimpleJdbcInsert and SimpleJdbcCall

# **Spring with ORM Frameworks**

Spring provides API to easily integrate Spring with ORM frameworks such as Hibernate, JPA(Java Persistence API), JDO(Java Data Objects), Oracle Toplink and iBATIS.

**Advantage of ORM Frameworks with Spring**

There are a lot of advantage of Spring framework in respect to ORM frameworks. There are as follows:

* **Less coding is required**
* **Easy to test**:
* **Better exception handling**
* **Integrated transaction management**

# **Hibernate and Spring Integration**

We can simply integrate **hibernate application with spring application**.In hibernate framework, we provide all the database information hibernate.cfg.xml file.

But if we are going to integrate the hibernate application with spring, we don't need to create the hibernate.cfg.xml file. We can provide all the information in the applicationContext.xml file.

### Advantage of Spring framework with hibernate

## The Spring framework provides **HibernateTemplate** class, so you don't need to follow so many steps like create Configuration, BuildSessionFactory, Session, beginning and committing transaction etc. So it saves a lot of code.

## **Steps**

Let's see what are the simple steps for hibernate and spring integration:

1. **create table in the database** It is optional.
2. **create applicationContext.xml file** It contains information of DataSource, SessionFactory etc.
3. **create Employee.java file** It is the persistent class
4. **create employee.hbm.xml file** It is the mapping file.
5. **create EmployeeDao.java file** It is the dao class that uses HibernateTemplate.
6. **create InsertTest.java file** It calls methods of EmployeeDao class.

Methods of HibernateTemplate class

Let's see a list of commonly used methods of HibernateTemplate class.

1) **void persist(Object entity)**: persists the given object.

2) **Serializable save(Object entity)**: persists the given object and returns id.

3) **void saveOrUpdate(Object entity)**: persists or updates the given object. If id is found, it updates the record otherwise saves the record.

4) **void update(Object entity):** updates the given object.

5) **void delete(Object entity)**: deletes the given object on the basis of id.

6) **Object get(Class entityClass, Serializable id):** returns the persistent object on the basis of given id.

7) **Object load(Class entityClass, Serializable id)**: returns the persistent object on the basis of given id.

8) **List loadAll(Class entityClass)**: returns the all the persistent objects.

# **Spring Data JPA**

Spring Data JPA API provides JpaTemplate class to integrate spring application with JPA.

JPA (Java Persistent API) is the sun specification for persisting objects in the enterprise application. It is currently used as the replacement for complex entity beans

## **Advantage of Spring JpaTemplate**

You don't need to write the before and after code for persisting, updating, deleting or searching object such as creating Persistence instance, creating EntityManagerFactory instance, creating EntityTransaction instance, creating EntityManager instance, commiting EntityTransaction instance and closing EntityManager.

# **Spring MVC**

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**. Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
* **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

## **Advantages of Spring MVC Framework**

Let's see some of the advantages of Spring MVC Framework:-

* **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
* **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
* **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
* **Rapid development** - The Spring MVC facilitates fast and parallel development.
* **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
* **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
* **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

Steps for Spring Web MVC framework:

* Load the spring jar files or add dependencies in the case of Maven
* Create the controller class
* Provide the entry of controller in the web.xml file
* Define the bean in the separate XML file
* Display the message in the JSP page
* Start the server and deploy the project

## **Spring Boot**

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications. It is a Spring module that provides the **RAD (Rapid Application Development)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.

In short, Spring Boot is the combination of **Spring Framework** and **Embedded Servers**.

In Spring Boot, there is no requirement for XML configuration (deployment descriptor). It uses convention over configuration software design paradigm that means it decreases the effort of the developer.

We can use Spring **STS IDE** or **Spring Initializr** to develop Spring Boot Java applications.

We should use Spring Boot Framework because:

* The dependency injection approach is used in Spring Boot.
* It contains powerful database transaction management capabilities.
* It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
* It reduces the cost and development time of the application

## **Advantages of Spring Boot**

* It creates **stand-alone** Spring applications that can be started using Java **-jar**.
* It tests web applications easily with the help of different **Embedded** HTTP servers such as **Tomcat, Jetty,** etc. We don't need to deploy WAR files.
* It provides opinionated '**starter**' POMs to simplify our Maven configuration.
* It provides **production-ready** features such as **metrics, health checks,** and **externalized configuration**.
* There is no requirement for **XML** configuration.
* It offers a **CLI** tool for developing and testing the Spring Boot application.
* It offers the number of **plug-ins**.
* It also minimizes writing multiple **boilerplate codes** (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
* It **increases productivity** and reduces development time.

## **Spring Boot Features**

* Web Development
* SpringApplication
* Application events and listeners
* Admin features
* Externalized Configuration
* Properties Files
* YAML Support
* Type-safe Configuration
* Logging
* Security

# **Spring vs. Spring Boot vs. Spring MVC**

## **Spring vs. Spring Boot**

**Spring:** Spring Framework is the most popular application development framework of Java. The main feature of the Spring Framework is **dependency Injection** or **Inversion of Control** (IoC). With the help of Spring Framework, we can develop a **loosely** coupled application. It is better to use if application type or characteristics are purely defined.

**Spring Boot:** Spring Boot is a module of Spring Framework. It allows us to build a stand-alone application with minimal or zero configurations. It is better to use if we want to develop a simple Spring-based application or RESTful services.

## **Spring Boot vs. Spring MVC**

**Spring Boot:** Spring Boot makes it easy to quickly bootstrap and start developing a Spring-based application. It avoids a lot of boilerplate code. It hides a lot of complexity behind the scene so that the developer can quickly get started and develop Spring-based applications easily.

**Spring MVC:** Spring MVC is a Web MVC Framework for building web applications. It contains a lot of configuration files for various capabilities. It is an HTTP oriented web application development framework.

# **Spring Boot Annotations**

Spring Boot Annotations is a form of metadata that provides data about a program. In other words, annotations are used to provide **supplemental** information about a program. It is not a part of the application that we develop. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.

**Spring Framework Annotations**

**@Required:** It applies to the **bean** setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception **BeanInitilizationException**

**@Autowired:** Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

**@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

**@ComponentScan:** It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

**@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

**@Component:** It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.

**@Controller:** The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation.

**@Service:** It is also used at class level. It tells the Spring that class contains the **business logic**.

**@Repository:** It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

# **Spring Boot Dependency Management**

Spring Boot manages dependencies and configuration automatically. Each release of Spring Boot provides a list of dependencies that it supports. The list of dependencies is available as a part of the **Bills of Materials** (spring-boot-dependencies) that can be used with **Maven**. So, we need not to specify the version of the dependencies in our configuration. Spring Boot manages itself. Spring Boot upgrades all dependencies automatically in a consistent way when we update the Spring Boot version.

## **Advantages of Dependency Management**

* It provides the centralization of dependency information by specifying the Spring Boot version in one place. It helps when we switch from one version to another.
* It avoids mismatch of different versions of Spring Boot libraries.
* We only need to write a library name with specifying the version. It is helpful in multi-module projects.

Spring Boot provides various properties that can be configured in the **application.properties**file. The properties have default values. We can set a property(s) for the Spring Boot application. Spring Boot also allows us to define our own property if required.

### Example of application.properties

1. #configuring application name
2. spring.application.name = demoApplication
3. #configuring port
4. server.port = 8081

In the above example, we have configured the **application name** and **port**. The port 8081 denotes that the application runs on port **8081**.

# **Spring Boot JPA**

**Spring Boot JPA**is a Java specification for managing **relational** data in Java applications. It allows us to access and persist data between Java object/ class and relational database. JPA follows **Object-Relation Mapping**(ORM). It is a set of interfaces. It also provides a runtime **EntityManager** API for processing queries and transactions on the objects against the database. It uses a platform-independent object-oriented query language JPQL (Java Persistent Query Language).

In the context of persistence, it covers three areas:

* The Java Persistence API
* **Object-Relational** metadata
* The API itself, defined in the **persistence** package

## **JPA Architecture**

JPA is a source to store business entities as relational entities. It shows how to define a POJO as an entity and how to manage entities with relation.

The following figure describes the class-level architecture of JPA that describes the core classes and interfaces of JPA that is defined in the **javax persistence** package. The JPA architecture contains the following units:

* **Persistence:** It is a class that contains static methods to obtain an EntityManagerFactory instance.
* **EntityManagerFactory:** It is a factory class of EntityManager. It creates and manages multiple instances of EntityManager.
* **EntityManager:** It is an interface. It controls the persistence operations on objects. It works for the Query instance.
* **Entity:** The entities are the persistence objects stores as a record in the database.
* **Persistence Unit:** It defines a set of all entity classes. In an application, EntityManager instances manage it. The set of entity classes represents the data contained within a single data store.
* **EntityTransaction:** It has a **one-to-one** relationship with the EntityManager class. For each EntityManager, operations are maintained by EntityTransaction class.
* **Query:** It is an interface that is implemented by each JPA vendor to obtain relation objects that meet the criteria.

# **Spring Boot JDBC**

**Spring Boot JDBC** provides starter and libraries for connecting an application with JDBC.

In Spring Boot JDBC, the database related beans such as **DataSource, JdbcTemplate,** and **NamedParameterJdbcTemplate** auto-configures and created during the startup. We can autowire these classes if we want to use it. For example:

1. @Autowired
2. JdbcTemplate jdbcTemplate;
3. @Autowired
4. **private** NamedParameterJdbcTemplate jdbcTemplate;

In **application.properties** file, we configure **DataSource** and **connection pooling**. [Spring Boot](https://www.javatpoint.com/spring-boot-tutorial)

chooses **tomcat** pooling by default.

## **JDBC Connection Pooling**

**JDBC connection pooling** is a mechanism that manages **multiple** database connection requests. In other words, it facilitates connection reuse, a memory cache of database connections, called a **connection pool.** A connection pooling module maintains it as a layer on top of any standard JDBC driver product.

# **Spring Boot H2 Database**

## **In-memory database**

In-memory database relies on system memory as oppose to disk space for storage of data. Because memory access is faster than disk access. We use the in-memory database when we do not need to persist the data. The in-memory database is an embedded database. The in-memory databases are volatile, by default, and all stored data loss when we restart the application.

The widely used in-memory databases are **H2, HSQLDB**(HyperSQL Database)**,**and**Apache Derby.**It creates the configuration automatically.

## **Persistence vs. In-memory Database**

The persistent database persists the data in physical memory. The data will be available even if the database server is bounced. Some popular persistence databases are **[Oracle](https://www.javatpoint.com/oracle-tutorial)**

**, [MySQL](https://www.javatpoint.com/mysql-tutorial)**

**, [Postgres](https://www.javatpoint.com/postgresql-tutorial)**

In the case of the **in-memory database,** data store in the **system memory**. It lost the data when the program is closed. It is helpful for **POC**s (Proof of Concepts), not for a production application. The widely used in-memory database is **H2.**

## **H2 Database**

**H2** is an **embedded, open-source,**and**in-memory** database. It is a relational database management system written in [Java](https://www.javatpoint.com/java-tutorial)

. It is a **client/server** application. It is generally used in **unit testing**. It stores data in memory, not persist the data on disk.

**Advantages**

* Zero configuration
* It is easy to use.
* It is lightweight and fast.
* It provides simple Configuration to switch between a real database and in-memory database.
* It supports standard SQL and JDBC API.
* It provides a web console to maintain in the database.

## **Spring Boot H2 Project Creation Example using Spring Initilizr**

**Step 1:** Open Spring Initializr [http://start.spring.io](https://start.spring.io/" \t "_blank)

.**Step 2:** Select the Spring Boot latest version**.**

**Step 2:** Provide the **Group** name. We have provided **com.Ru.Spring-Boot**

**Step 3:** Provide the **Artifact** Id. We have provided **demo-spring-boot-h2-database**

**Step 5:** Add the dependencies **Spring Web, Spring Data [JPA](https://www.javatpoint.com/jpa-tutorial)**

**,**and**H2 Database.**

**Step 6:** Click on the **Generate** button. When we click on the Generate button, it wraps the project in a **Jar** file and downloads it to the local system.

**Step 7:** **Extract** the Jar file and paste it into the eclipse workspace.

**Step 8:** **Import** the project folder into Eclipse.