**SPRING FRAMEWORK**

**INTRODUCTION:**

The Spring Framework is a Java Platform that provides comprehensive infrastructure support for developing Java applications. ”Spring” handles the infrastructure.

Examples, as an application developer can benefit from the Spring platform:

* Make a Java method execute in a database transaction without having to deal with transaction APIs
* Make a local Java method an HTTP endpoint without having to deal with the servlet API

It typically consists of objects that collaborate to form the application proper. Thus the objects in an application have dependencies on each other.

WHY IS SPRING POPULAR?

* Enables Testable code
* No plumbing code
* Flexible Architecture
* Staying current

**FRAMEWORK MODULES:**

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP, Instrumentation, Messaging, and Test.

The following sections list is the available modules for each feature along with their artifact names. Artifact names correlate to artifact IDs used in “Dependency Management Tools”.

1. **Core Container**:

The Core Container consists of the spring-core, spring-context, spring-beans, spring-context-support, spring-expression.

The ‘spring-core and spring-beans’ modules provides the fundamental parts of the framework. “The BeanFactory” is a sophisticated implementation of the factory pattern.

The Context (spring-context) module builds on the solid base provided by the **Core and Beans** modules.

The “ApplicationContext” interface is the focal point of the context module.

1. **AOP and Instrumentation:**

The spring-aop module provides an AOP Alliance-complaint aspect-oriented programming implementation.

The separate spring-aspects module provides integration with AspectJ.

Spring-instrument-tomcat module contains Spring’s instrumentation agent for Tomcat.

1. **Data Access/Integration**:

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and the Transaction modules.

The spring-jdbc module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.

1. **WEB:**

The “Web layer” consists of the spring-web, spring-webmvc, spring-websocket and spring-webmvc-portlet modules.

The “spring-web” module provides basic web-oriented integration features. It also contains an HTTP client and the web-related parts of the Spring’s remoting support.

The “spring-webmvc” module (also known as web-servlet module) contains Spring’s model-view-controller(MVC) and REST Web services implementation for web-application. Spring’s MVC framework provides a clean separation between domain model code and web forms and integrates with all of the other features of the Spring Framework.

1. **Test:**

The spring-test module supports the [unit testing](https://docs.spring.io/spring-framework/docs/4.3.12.RELEASE/spring-framework-reference/html/unit-testing.html) and [integration testing](https://docs.spring.io/spring-framework/docs/4.3.12.RELEASE/spring-framework-reference/html/integration-testing.html) of Spring components with JUnit or TestNG. It provides consistent [loading](https://docs.spring.io/spring-framework/docs/4.3.12.RELEASE/spring-framework-reference/html/integration-testing.html#testcontext-ctx-management) of Spring ApplicationContext’s and [caching](https://docs.spring.io/spring-framework/docs/4.3.12.RELEASE/spring-framework-reference/html/integration-testing.html#testcontext-ctx-management-caching) of those contexts. It also provides [mock objects](https://docs.spring.io/spring-framework/docs/4.3.12.RELEASE/spring-framework-reference/html/unit-testing.html#mock-objects) that you can use to test your code in isolation.

**Dependency Management and Naming Conventions:**

Dependency management and dependency injection are different things. To get those nice features of Spring into your application (like dependency injection) you need to assemble all the libraries needed (jar files) and get them onto your classpath at runtime, and possibly at compile time. These dependencies are not virtual components that are injected, but physical resources in a file system (typically). The process of dependency management involves locating those resources, storing them and adding them to classpaths.

**Maven Dependency Management:**

If you are using [Maven](https://maven.apache.org/) for dependency management you don’t even need to supply the logging dependency explicitly.

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.3.12.RELEASE</version>

<scope>runtime</scope>

</dependency>

</dependencies>

Note the scope can be declared as runtime if you don’t need to compile against Spring APIs, which is typically the case for basic dependency injection use cases.

**Spring in Depth:**

* Autowiring types and Qualifiers
* Bean scope and Life Cycle
* IOC container and Application context
* XML and Java Application Context
* Component Scan
* External Properties
* Container and Dependency Injection(CD)

**Spring Bean Life Cycle:**

Spring Framework also support @PostConstruct and @PreDestroy annotations for defining post-init and pre-destroy methods. These annotations are part of javax.annotation package. However for these annotations to work, we need to configure our spring application to look for annotations. We can do this either by defining bean of org.springframework.context.annotation.CommonAnnotationBeanPostProcessor or by context:annotation-config element in spring bean configuration file.

**Container and Dependency Injection:**

Spring supports most annotations

@Inject (@Autowired), @Named (@Component and @Qualifier) and @Singleton (Defines a scope of singleton).

**HIBERNATE**

**INTRODUCTION:**

Hibernate ORM (Hibernate in short) is an object-relational mapping framework, facilitating the conversion of an object-oriented domain model to a traditional relational database. Hibernate solves the object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions.

Hibernate consists of three different components:

**• Entities**:

The classes that are mapped by Hibernate to the tables of a relational database system are simple Java classes.

• **Object-relational metadata**:

The information how to map the entities to the relational database is either provided by annotations (since Java 1.5) or by legacy XML-based configuration files. The information in these files is used at runtime to perform the mapping to the data store and back to the Java objects.

• **Hibernate Query Language (HQL)**:

When using Hibernate, queries send to the database do not have to be formulated in native SQL but can be specified using Hibernate’s query language. As these queries are translated at runtime into the currently used dialect of the chose product, queries formulated in HQL are independent from the SQL dialect of a specific vendor.

To implement a code we need the following Dependencies:

H2 dependency:

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>h2 database version</version>

<scope>test</scope>

Hibernate Dependency:

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>hibernate version </version>

</dependency>

**RELATIONSHIPS:**

Up to now the only relationship between two tables we have seen was the "extends" one. Next to the mere inheritance Hibernate can also map relationships that are based on lists where the one entity has a list of instances of another entity. The following types of relationships are distinguished:

• **One to one**:

This denotes a simple relationship in which one entity of type A belongs exactly to one entity of type B.

• **Many to one**:

As the name indicates, this relationship encompasses the case that an entity of type A has many child entities of type B.

• **Many to many**:

In this case there can be many entities of type A that belong to many entities of type B.

**USER DEFINED DATATYPES:**

When working for example with a legacy database it can happen that certain columns are modelled in a different way than Hibernate would map them. The Boolean data type for example is mapped on an H2 database to the type Boolean. If the original development team has decided to map Boolean values using a string with the value "0" and "1", Hibernate allows to implement user-defined types that are used for the mapping.