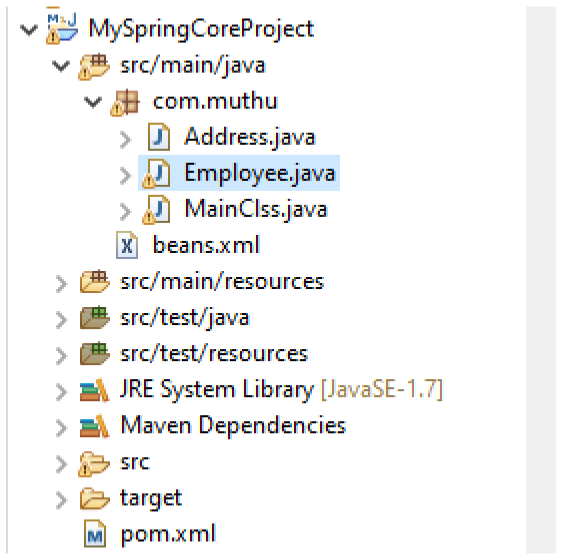
CORE SPRING:

Create the Spring Maven : Sample Example Complete Project Setup:



```
MySpringCoreProject
  src/main/java
    com.muthu
      Address.java
      Employee.java
      MainClss.java
    beans.xml
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.7]
  Maven Dependencies
  src
  target
  pom.xml
```

Building Java Projects with Maven

This guide walks you through using Maven to build a simple Java project.

What you'll build

You'll create an application that provides the time of day and then build it with Maven.

What you'll need

- **About 15 minutes**
- **A favorite text editor or IDE**
- **Java 17 or later**
- **Maven 3.5+**
- **You can also import the code straight into your IDE:**

- Spring Tool Suite (STS)
- IntelliJ IDEA
- VSCode

How to complete this guide

Like most Spring Getting Started guides, you can start from scratch and complete each step or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.
To start from scratch, move on to Set up the project.
To skip the basics, do the following:

- **Download and unzip the source repository for this guide, or clone it using Git: `git clone https://github.com/spring-guides/gs-maven.git`**
- **cd into `gs-maven/initial`**
- **Jump ahead to [initial].**

When you finish, you can check your results against the code in `gs-maven/complete`.

Set up the project

First you'll need to setup a Java project for Maven to build. To keep the focus on Maven, make the project as simple as possible for now. Create this structure in a project folder of your choosing.

Create the directory structure

In a project directory of your choosing, create the following subdirectory structure; for example, with `mkdir -p src/main/java/hello` on *nix systems:

```
└── src
    └── main
        └── java
            └── hello
```

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

1.1 Dependency Injection and Inversion of Control

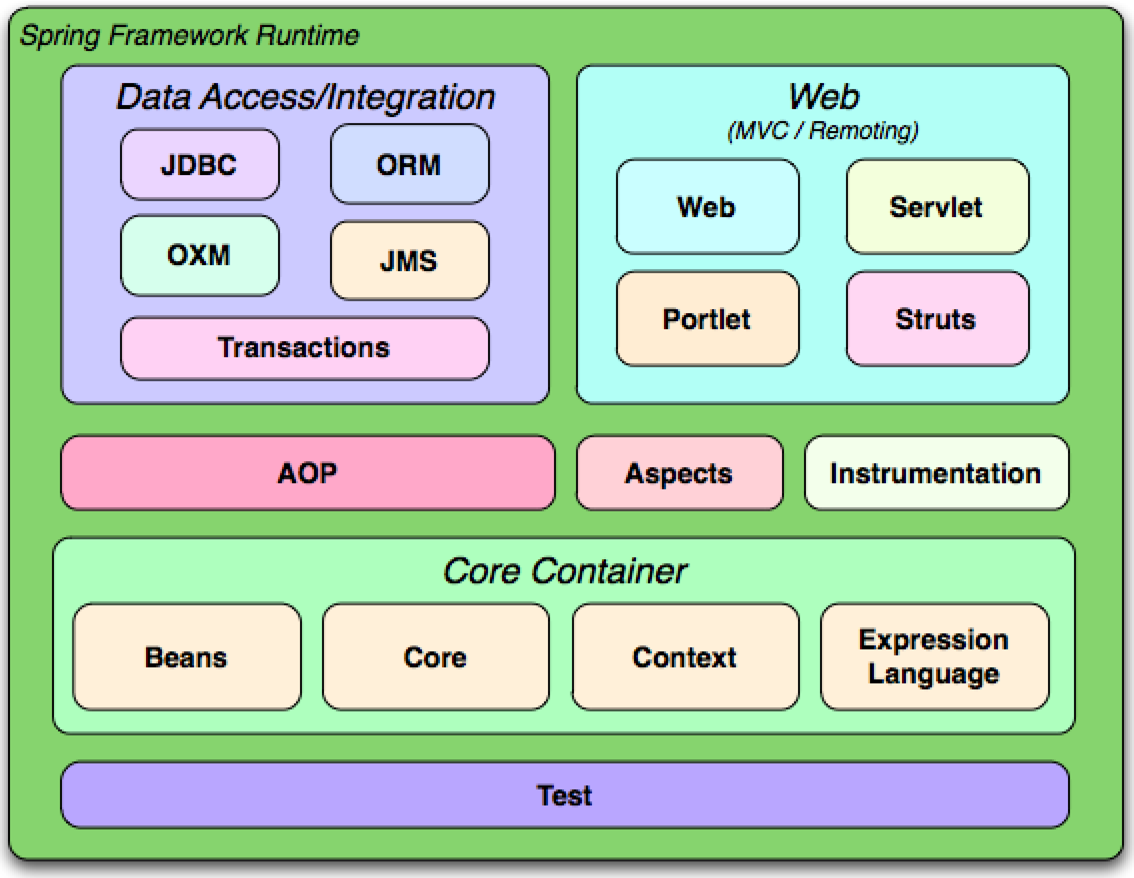
Java applications -- a loose term that runs the gamut from constrained applets to n-tier server-side enterprise applications -- typically consist of objects that collaborate to form the application proper. Thus the objects in an application have *dependencies* on each other.

Although the Java platform provides a wealth of application development functionality, it lacks the means to organize the basic building blocks into a coherent whole, leaving that task to architects and developers. True, you can use design patterns such as *Factory*, *Abstract Factory*, *Builder*, *Decorator*, and *Service Locator* to compose the various classes and object instances that make up an application. However, these patterns are simply that: best practices given a name, with a description of what the pattern does, where to apply it, the problems it addresses, and so forth. Patterns are formalized best practices that *you must implement yourself* in your application.

The Spring Framework *Inversion of Control* (IoC) component addresses this concern by providing a formalized means of composing disparate components into a fully working application ready for use. The Spring Framework codifies formalized design patterns as first-class objects that you can integrate into your own application(s). Numerous organizations and institutions use the Spring Framework in this manner to engineer robust, *maintainable* applications.

1.2 Modules

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram.



IoC Container

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets information from the XML file and works accordingly. The main tasks performed by IoC container are:

- **to instantiate the application class**
- **to configure the object**
- **to assemble the dependencies between the objects**

There are two types of IoC containers. They are:

1. **BeanFactory**
2. **ApplicationContext**

Difference between BeanFactory and the ApplicationContext

The org.springframework.beans.factory.BeanFactory and the org.springframework.context.ApplicationContext interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.