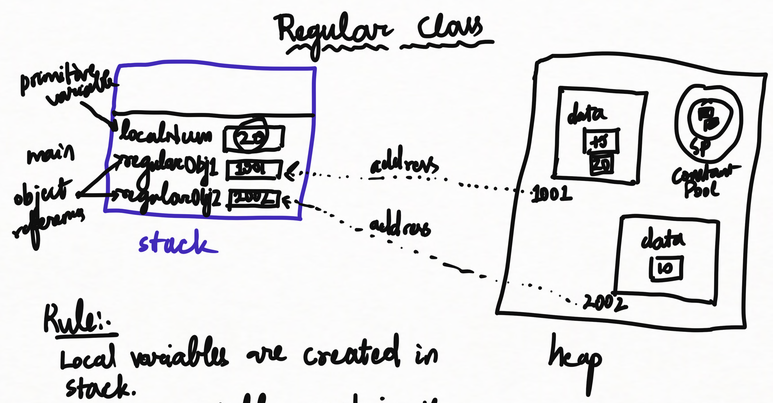
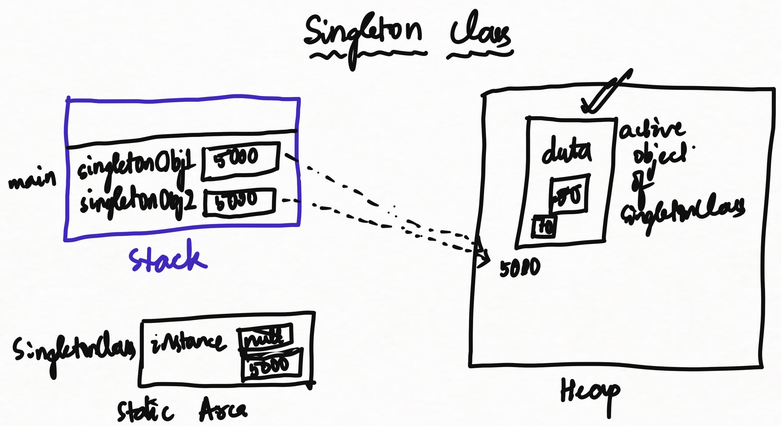


Rule:-

Local variables are created in stack.

Instance variables reside in the heap memory (in objects)


Singleton class
main
singletonObj1
5000
singletonObj2
6000
stack
SingletonClass
instance
null
5000
static Area
data
50
10
active object of singletonClass
5000
Heap

Use-cases of Singleton :-

i) Memory saving

ii) Data-sharing/synchronization

iii) can use an object repeatedly

Example

→ Database Connection

Limitations

i) can be unreliable in multi-threaded usage

ii) can be used unnecessarily to make things complex

Framework

→ A set of tools to effectively and efficiently do a complex task with minimum effort.

Example :- i) Creating a set of objects using collection framework

ii) Creating a Java web application using spring framework.

Framework Learning

How to use the tools in the framework

How the framework works internally.

Spring Framework

→ Based Java EE (Java Enterprise Edition)

→ Modules, hence loose coupling

Inversion of Control (IoC)

- → Transfers the control of objects to the container/framework.
- → Works with object-oriented programming
- → One of the ways IoC can be achieved is dependency injection (DI).

Advantages are:-

i) Loose coupling

ii) Greater modularity

iii) easier to switch implementation

iv) easier to test

Dependency Injection (DI)

- → Implementation of IoC.
- → Allows for loose coupling
- → Moves the responsibility of managing components onto the container.
- → Fundamental aspect of Spring framework.
- → The framework will inject objects into other objects.

```
class Item{ }
```

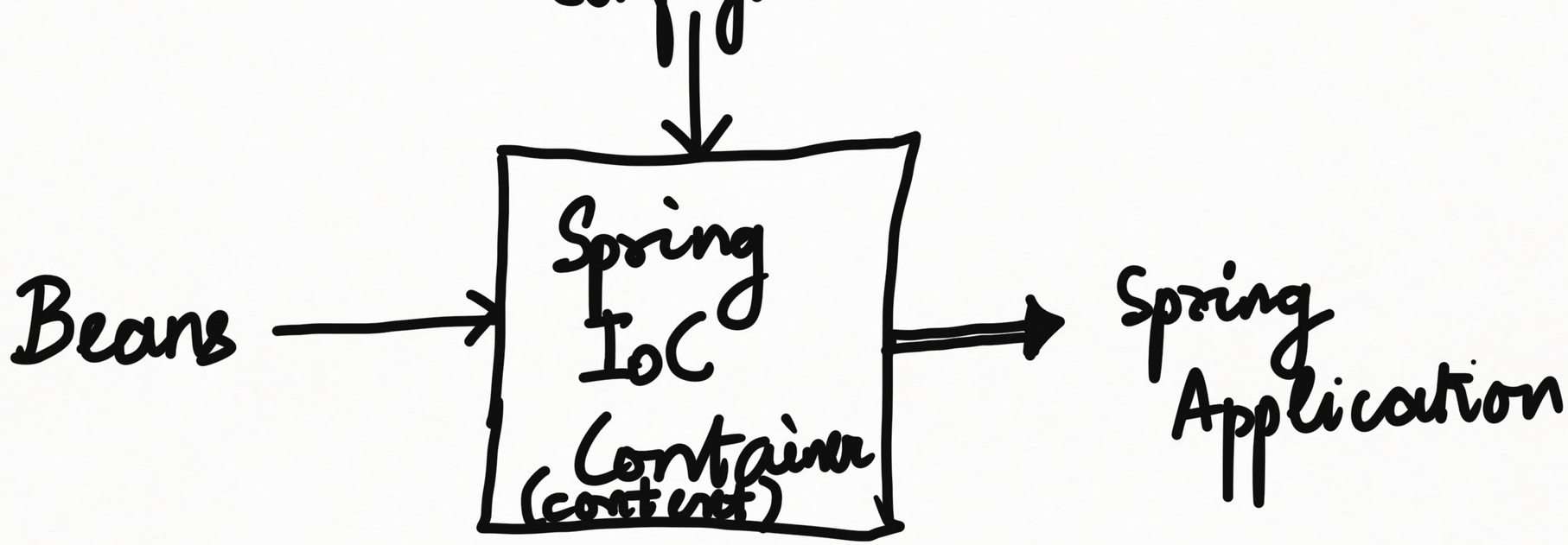
Traditional Resolution

```
class Store{
    private Item item;
    public Store(){
        item= new Item();
    }
}
```

Dependency Injection

```
class Store{
    private Item item;
    public Store(Item item){
        this.item = item;
    }
}
```

Spring IoC container - Manages DI



POJO = Plain Old Java Object.

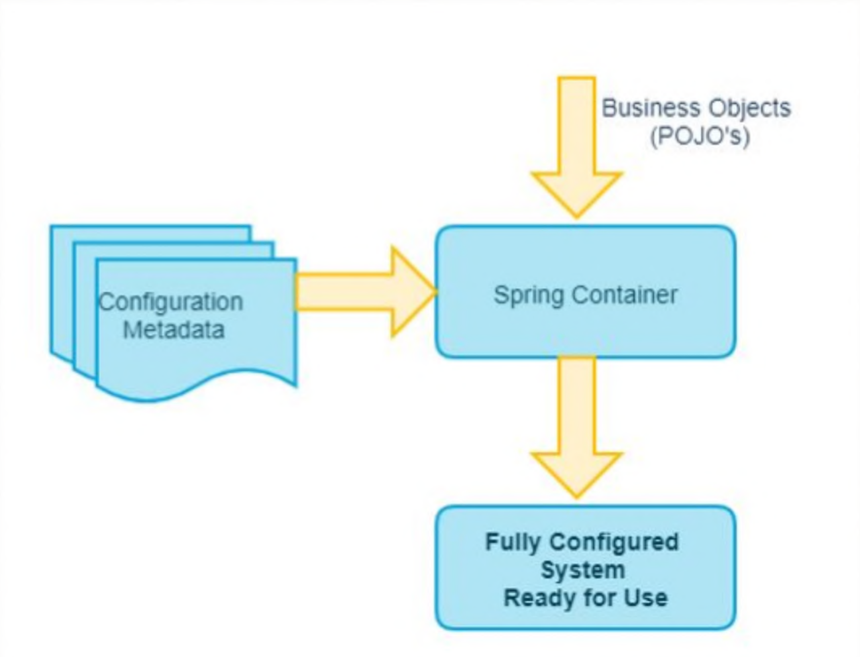
- In Spring framework, "ApplicationContent" represents the Spring IoC container.
- ApplicationContent manages all the beans by using the configuration metadata.
- ApplicationContent is an interface.
- Spring framework provides several implementations of the ApplicationContent interface.

↳ Some of the implementations are:-

↳ ClassPathXMLApplicationContext

↳ AnnotationConfigApplicationContext

etc.



Types of DI

i) Constructor Injection ✓

ii) Property (Field) Injection ✗

iii) Setter (Method Injection) ✓ (default)

✓ = supported by spring

Spring ApplicationContext

→ ApplicationContext is an interface

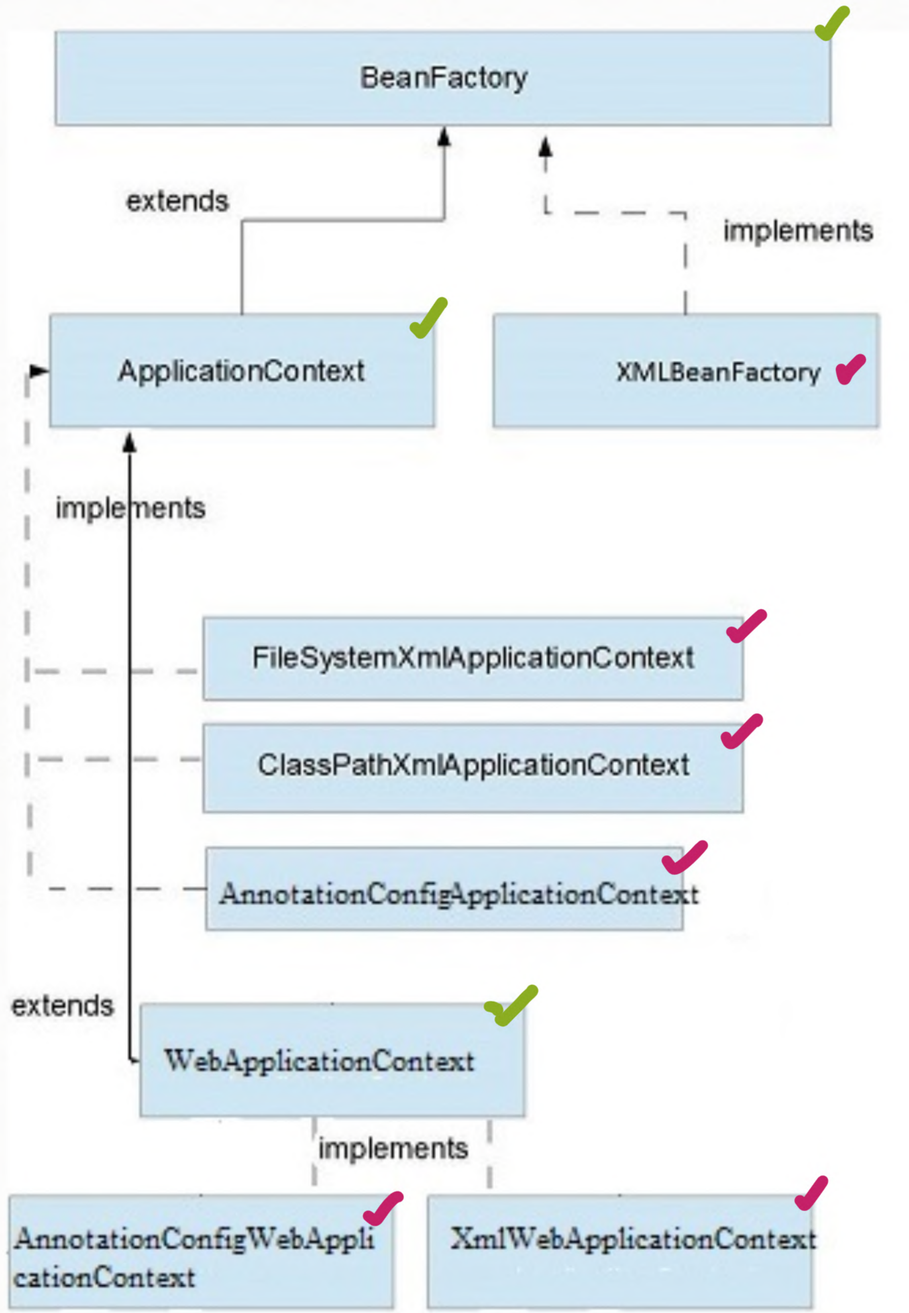
→ In Spring, ApplicationContext represents the container of the application.

→ In order to start a Spring application container, we need an instance of ApplicationContext

→ But ApplicationContext is an interface and it can't be instantiated.

→ So, we need an object of a concrete class that implements the ApplicationContext interface.

→ There are various concrete classes in Spring which implement the aforementioned interface, pertaining to different use-cases.



– interfaces

– concrete classes

→ There are 5 concrete implementations of AC :-

i) ClassPathXmlApplicationContext

ii) AnnotationConfigApplicationContext

iii) FileSystemXmlApplicationContext

iv) XmlWebApplicationContext

v) AnnotationConfigWebApplicationContext

→ The 5 concrete implementations can be broadly categorised into 2 types:-

a) Xml-Configured Application Contents

b) Annotation-Configured Application Contents

Events associated with ApplicationContext:-

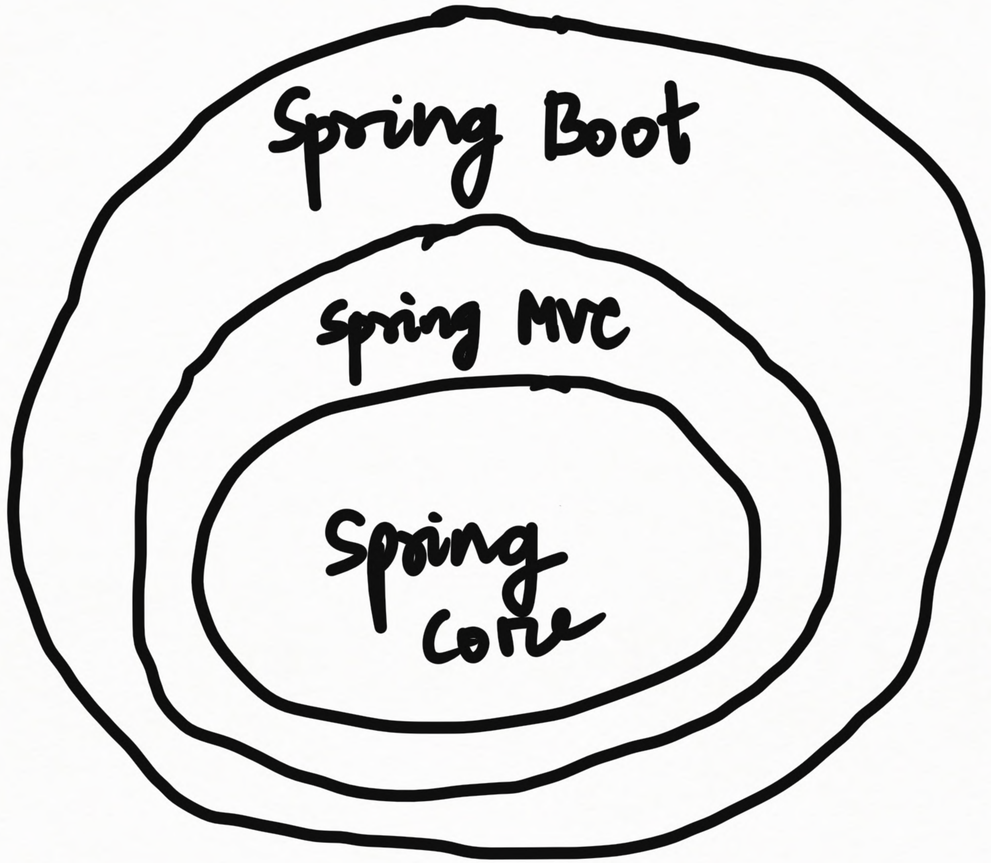
The following predefined/built-in container events are associated with the context object of every Spring application:-

i) ContextStartedEvent
ii) ContextRefreshedEvent
iii) ContextStoppedEvent
iv) ContextClosedEvent
v) RequestHandledEvent

→ Beans are managed objects (mostly POJOs). Their lifecycle is managed by the Spring IoC container. For instantiating a bean, the container may need some data, aka configuration metadata.

→ We can provide configuration metadata to the container (application context) in 3 ways:-

a) Xml-Based Configuration
b) Annotation-Based Configuration
c) Java-Based Configuration


Spring Boot
Spring MVC
Spring
Core

XML Schema-Based Configuration

Example

```java
package basicinjection;

public class Foo {
private String name;

public Foo() {  }

public Foo(String name) {
  this.name = name;
}

public void setName(String name) {
  this.name = name;
}

public String getName() {
  return name;
}

}
```

```java
package basicinjection;

public class Bar {
private String name;
private int age;
private Foo foo;

public Bar() {}

public Bar(String name,int age) {
  this.name = name;
  this.age = age;
}
public void setFoo(Foo foo) {
  this.foo = foo;
}

public void processFooName(){
  System.out.println("Name in Injected Foo is: "+foo.getName());
}

@Override
public String toString() {
  return "Bar has name = "+this.name+" and age = "+this.age;
}
}
```

To provide configuration metadata for the object reference "foo" in <u>Bar</u> class, we can use a XML file as follows:-

xml template →

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="foo" class="basicinjection.Foo" scope="prototype">
        <constructor-arg index="0" value="Cleopatra"></constructor-arg>
  </bean>
  <bean id="bar" class="basicinjection.Bar">
        <constructor-arg type="int" value="26" />
        <constructor-arg type="java.lang.String" value="Arthur" />
        <property name="foo" ref="foo"></property>
  </bean>
</beans>
```

bean defn for Foo

bean definition for Bar

→ `Foo f = new Foo("Cleopatra");`

↳ constructor injection

→ `Bar b = new Bar("Arthur",26);`
`b.setFoo(f);`

↳ setter injection

beans.xml or applicationContext.xml

There are 2 ways of creating spring application:-

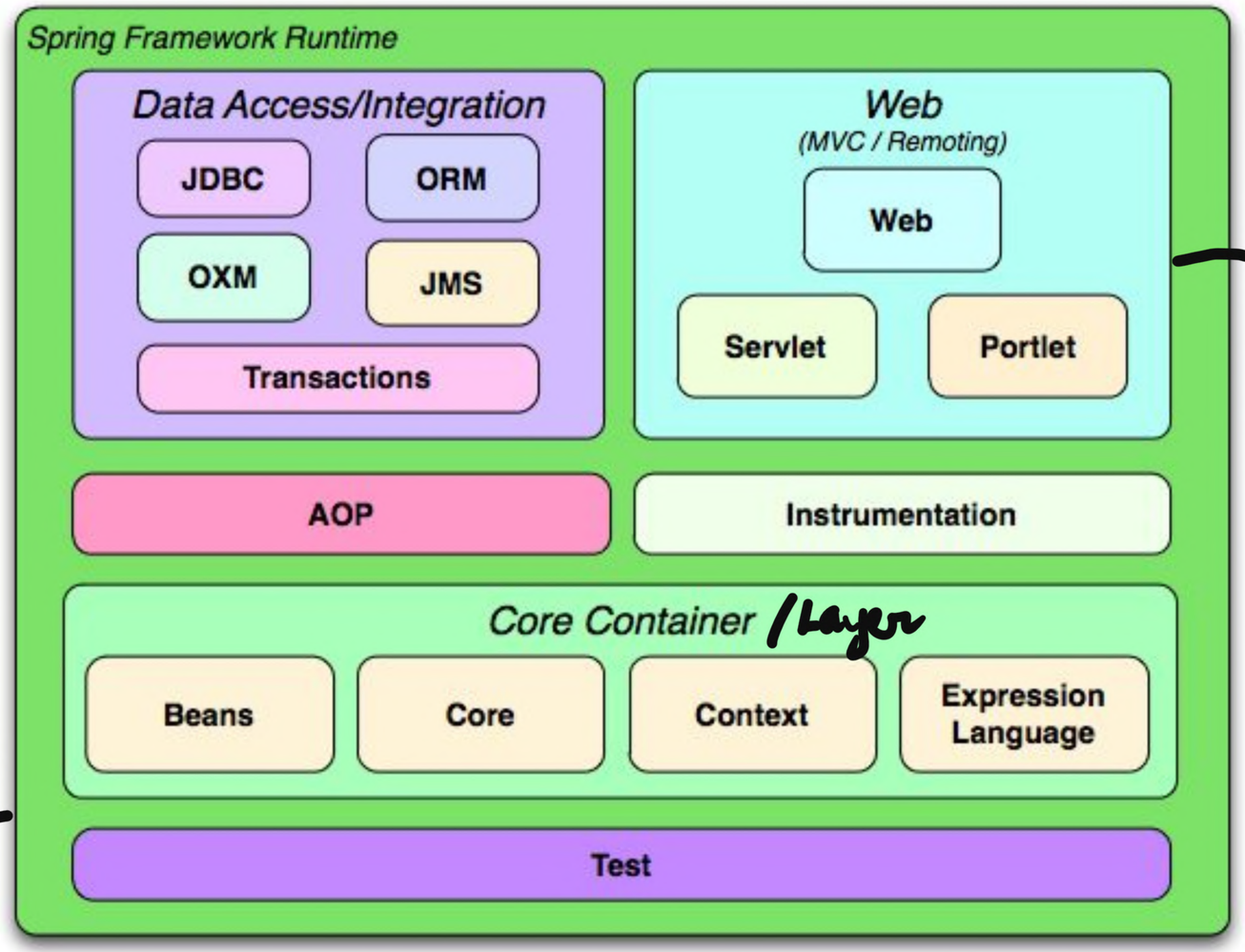
i) Manually (by adding spring dependencies to classpath)

ii) Using Maven (by defining spring dependencies in pom.xml)

Spring Modules or Spring Architecture

Java Project
+
(spring-core
+
spring-beans
+
spring-context
+
spring-expressions)

= Spring Project



20+ modules

web-enabled or web-aware Spring application

↳ Spring modules can be added as external dependencies to your Java project.

↳ External dependencies are added to the classpath.

↳ In eclipse:-

- Right-click on the project folder (in explorer)
- Hover on Build path & choose Configure build path.
- Select "Classpath" under the Libraries tab
- Click on Add External JARs button in the sidebar & add all the jar files.

→ When using Xml-Based configuration, the .xml file is kept directly under the "SRC" folder of your project.

→ XML = eXtensible Markup Language

→ In eclipse, after creating the beans.xml file or the applicationContext.xml file, right click on it and open with Generic Text Editor.

Summary for creating a basic Spring application:-

→ Create a Java project in an IDE & add the external jar files of the Spring framework

→ We need to create at least two files in our Spring app: Driver class (containing the main() method) and the XML configuration file.

→ We can create POJO classes & use them as beans by writing their bean definitions.

→ We can start our ApplicationContext instance & run our app.

Creating a Spring application using Maven

- In this approach, we don't have to manually download and add the external jar files to make it a Spring application.
- Instead, we'll enlist the required Spring modules as dependencies in the pom.xml file for our project.
- The pom.xml file will be created directly under the project folder.
- POM = Project Object Model

↳ If we use maven, we must create a "Maven project" using eclipse instead of creating a "Java project".

↳ While creating the Maven project, check the option to "Create a simple Maven project".

↳ We'll get a pom.xml file under the project folder.

↳ We need to add the dependencies to this file.

↳ Maven will download the external jars and add them to classpath of our application.

DB Operation with Spring

↳ Uses module(s) from Data Access/Integration Layer

↳ There are 3 ways of connecting and interacting with DB from Spring:-

i) Using mysql-connector + plain JDBC

ii) Using mysql-connector + Spring JDBC

iii) Using JPA (e.g.- Hibernate)

App	vs.	Web App
i) Needs installation		i) Doesn't need installation. Used in a browser app.
ii) Platform dependent		ii) Platform independent

Websites vs. Web Apps

images

Spring or Spring Boot

Java EE
J2EE

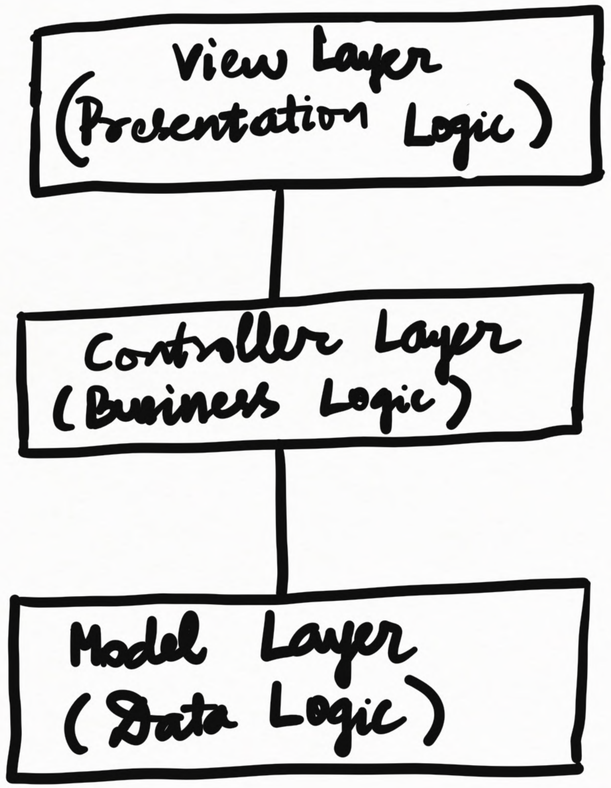
MVC
(MVC)

logic

Spring MVC	vs.	Spring Boot
→ subset		→ superset
→ can have models, views & controllers, i.e., it is used to create webapps		→ usually, it only has the models and the controllers, i.e., it is used to create APIs (mostly REST APIs)
→ Requires extensive configuration		→ Requires minimal configuration
→ We can configure a Spring MVC application by XML or annotations. But in both the cases, an XML file is needed in the project.		→ Requires almost no xml configuration. Configuration is purely Java-based annotations.
→ We need to run the project in a web-server & also structure the project		→ The server & the project structure can be pre configured.

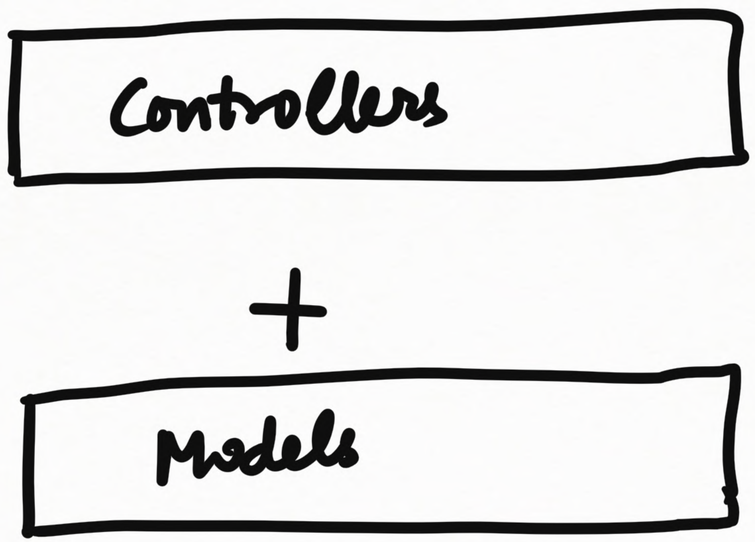
Backend of an Application

1) MVC Architecture (3-tier Architecture)

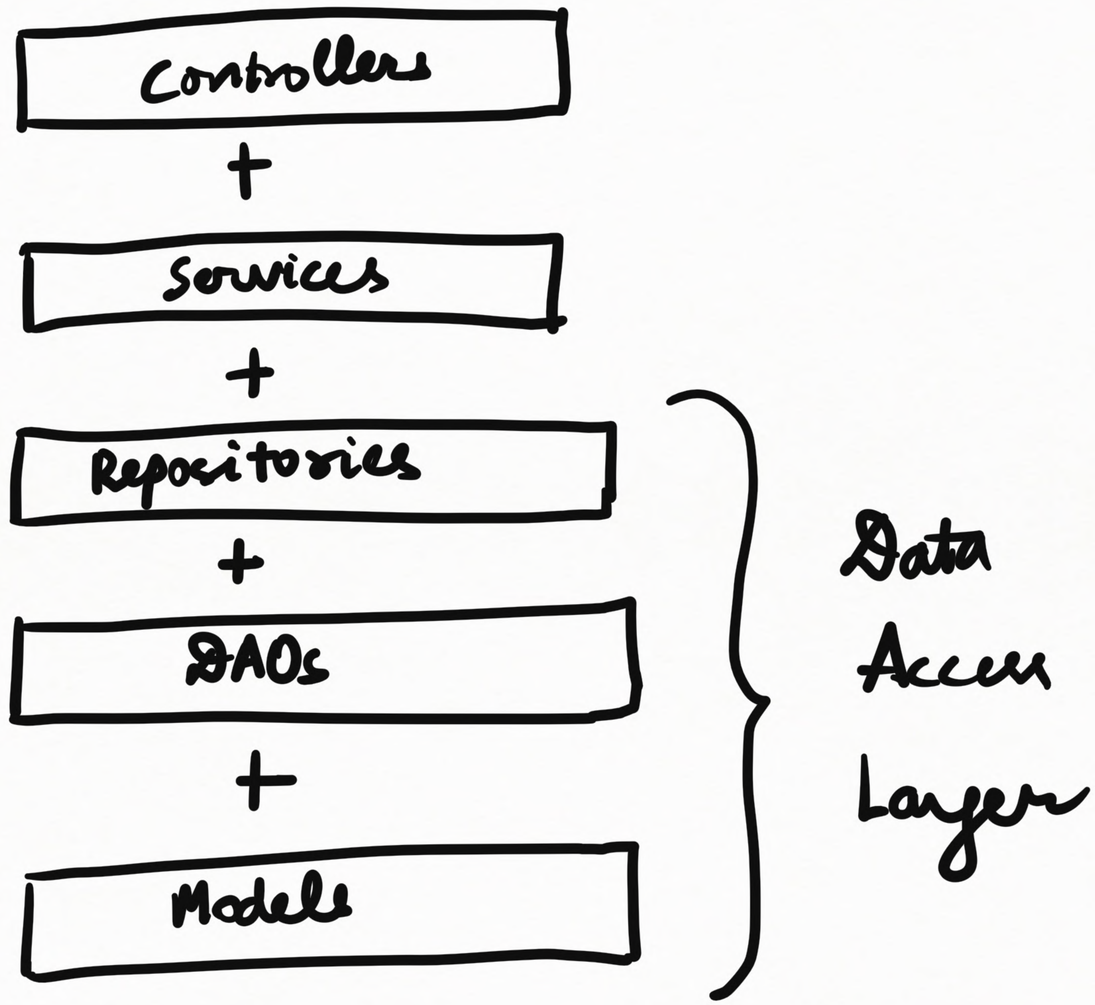


When we are talking about APIs, view layer is absent.
Thus, MVC architecture become 2-tiered.

Two-tier Architecture



N-tier Architecture



Dependency List of Tiers

→ **Models**: no dependencies

→ **DAOs**: Models

→ **Repositories**: DatabaseConnection

→ **Services**: Repositories

→ **Controllers**: Services

→ **Driver**: Controllers