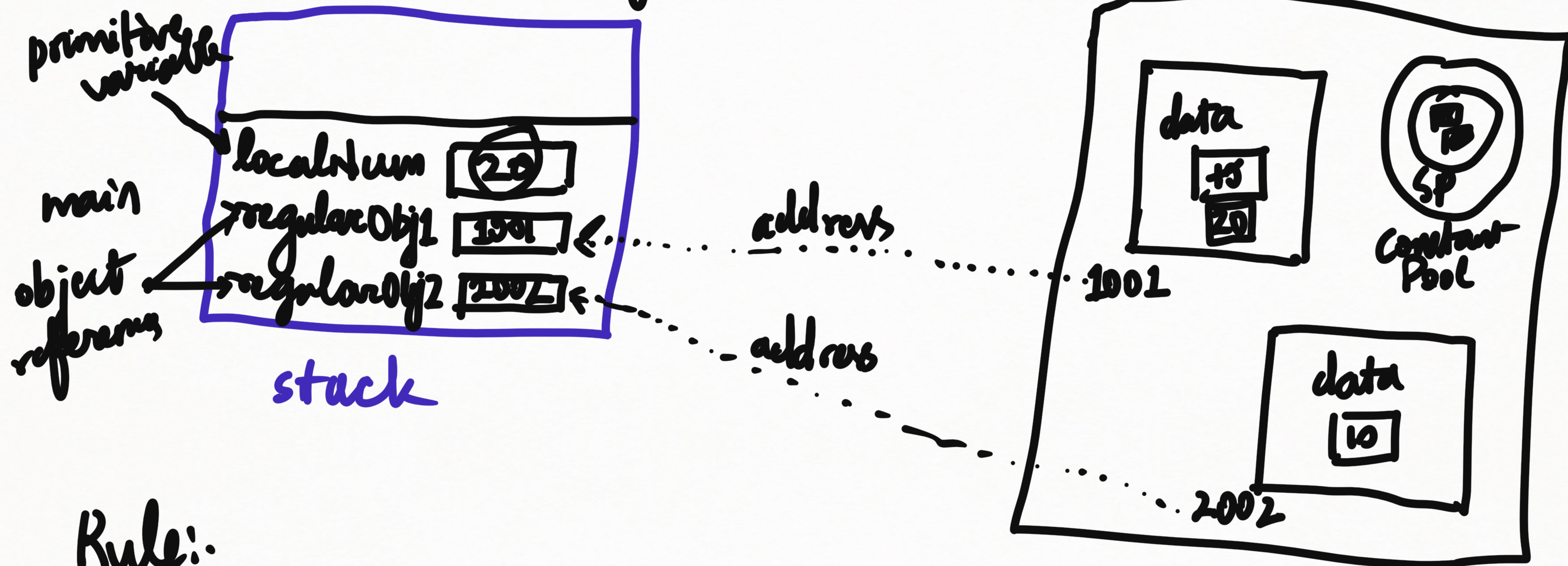


Regular class



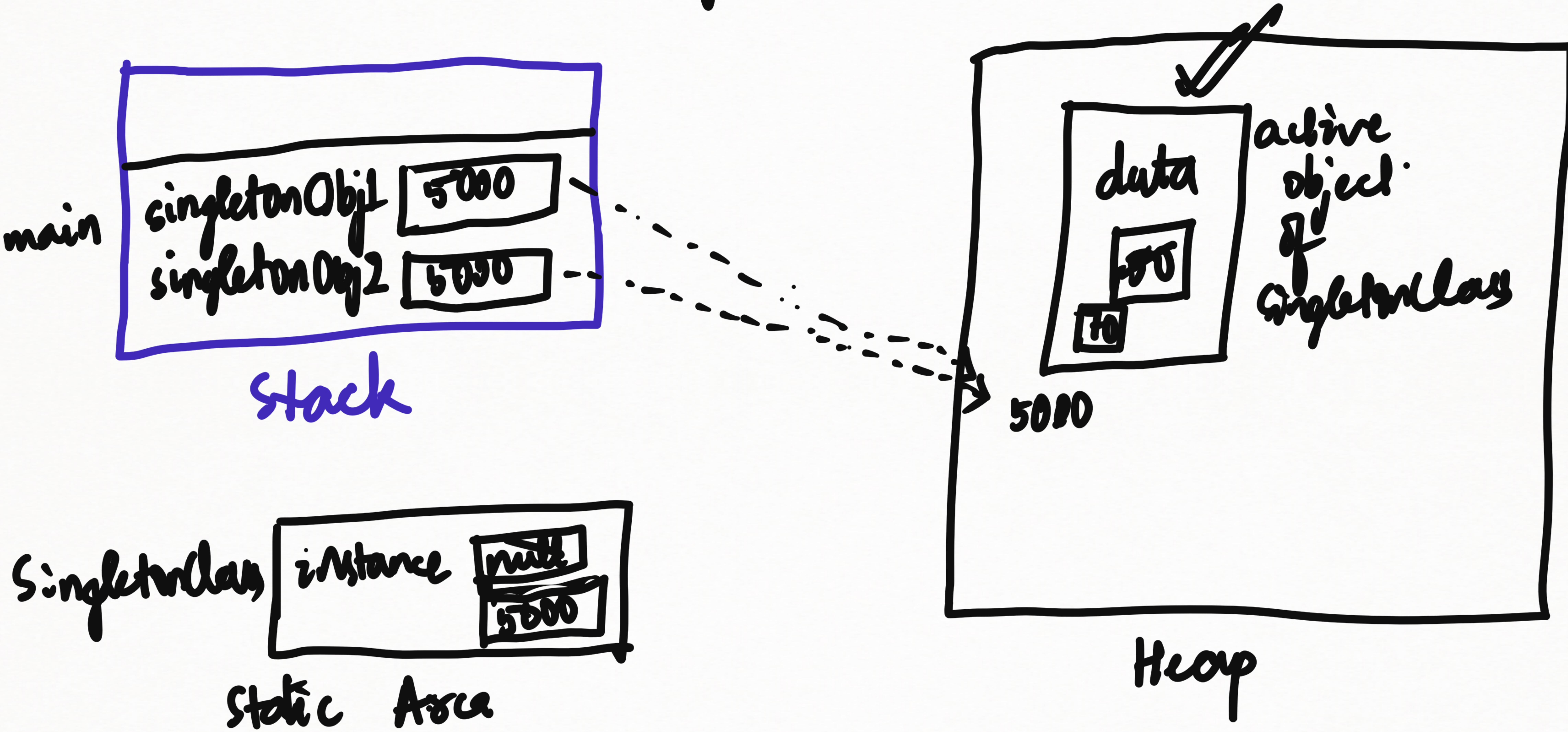
Rule:

Local variables are created in stack.

heap

instance variables reside in the heap memory (in objects)

Singleton Class



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

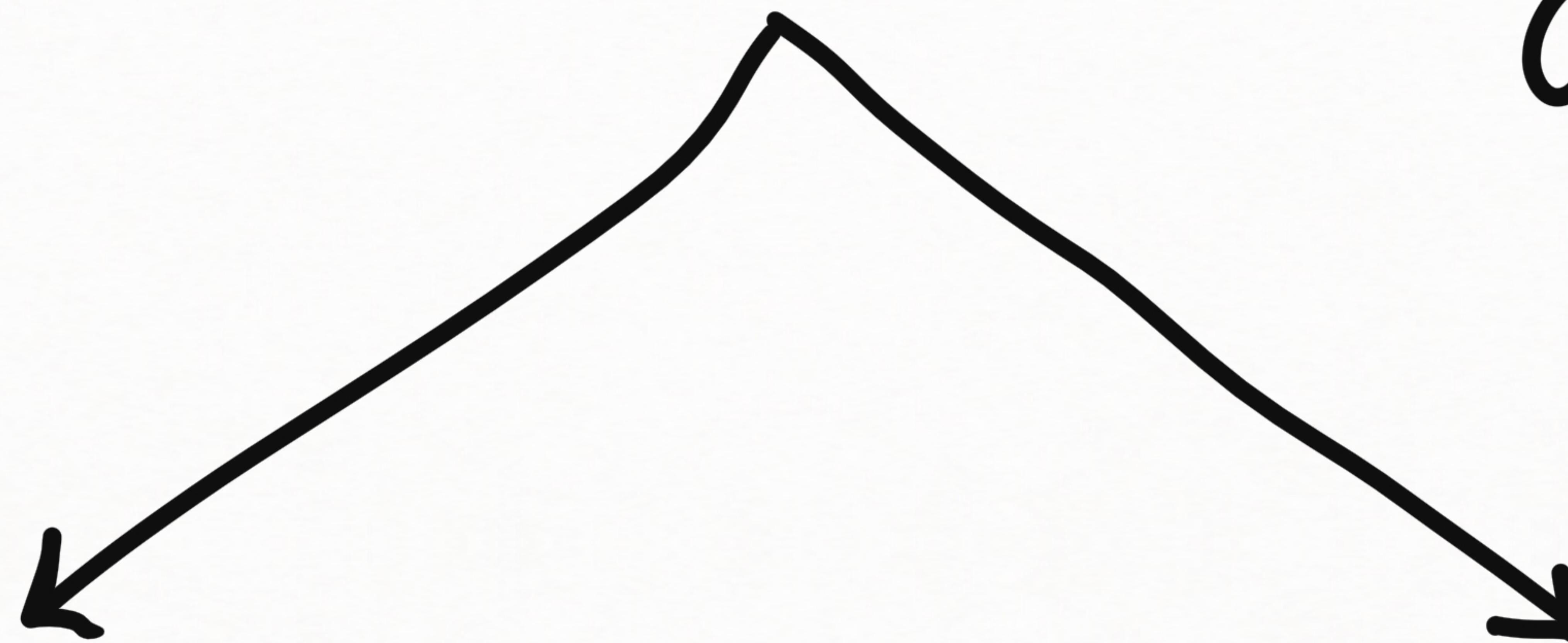
→ A set of tools to effectively and efficiently do a complex task with minimum effort.

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

ii) Creating a Java web application using Spring framework.

Framework Learning

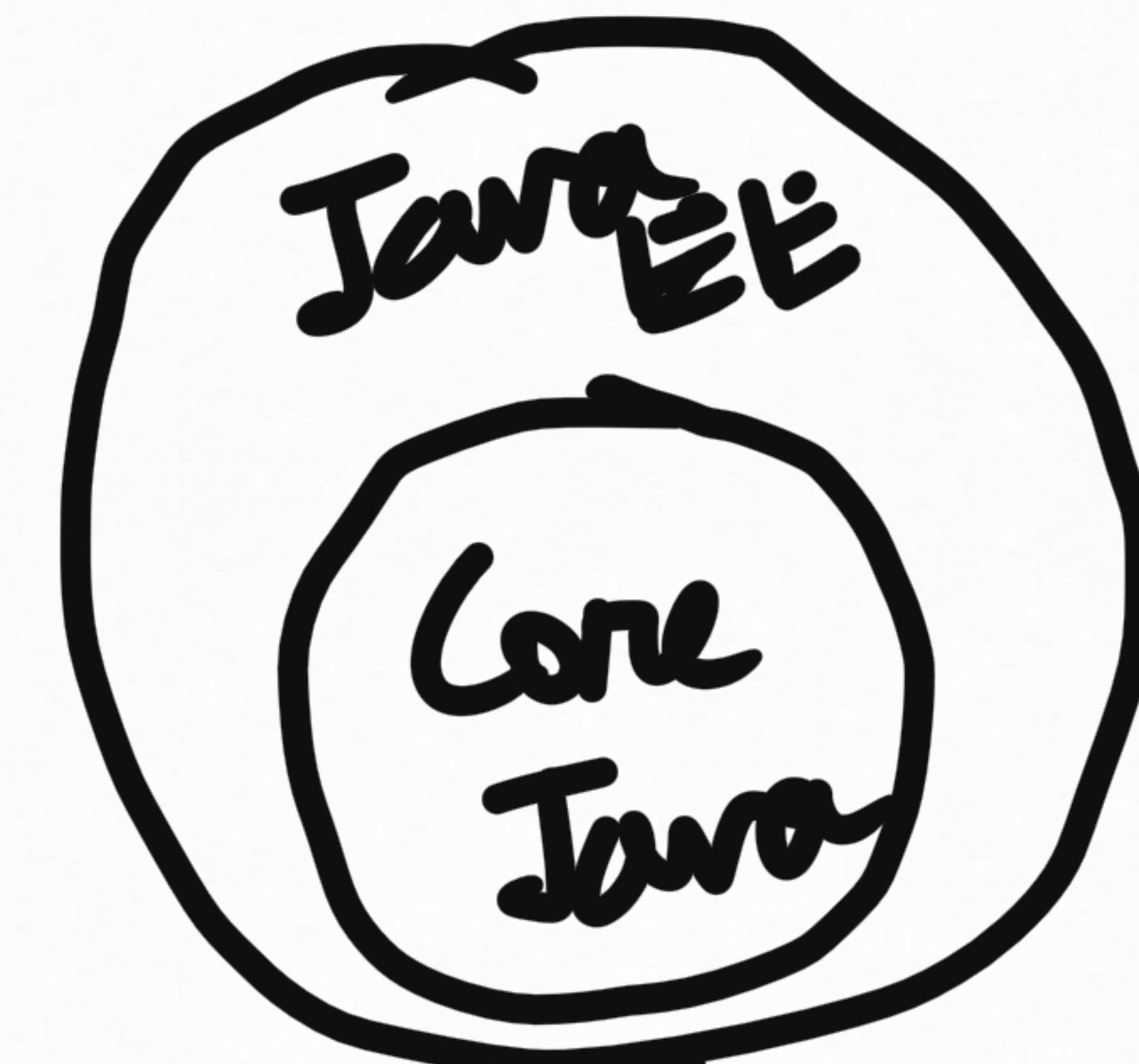
How to use the
tools in the
framework



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;  
    }  
}
```

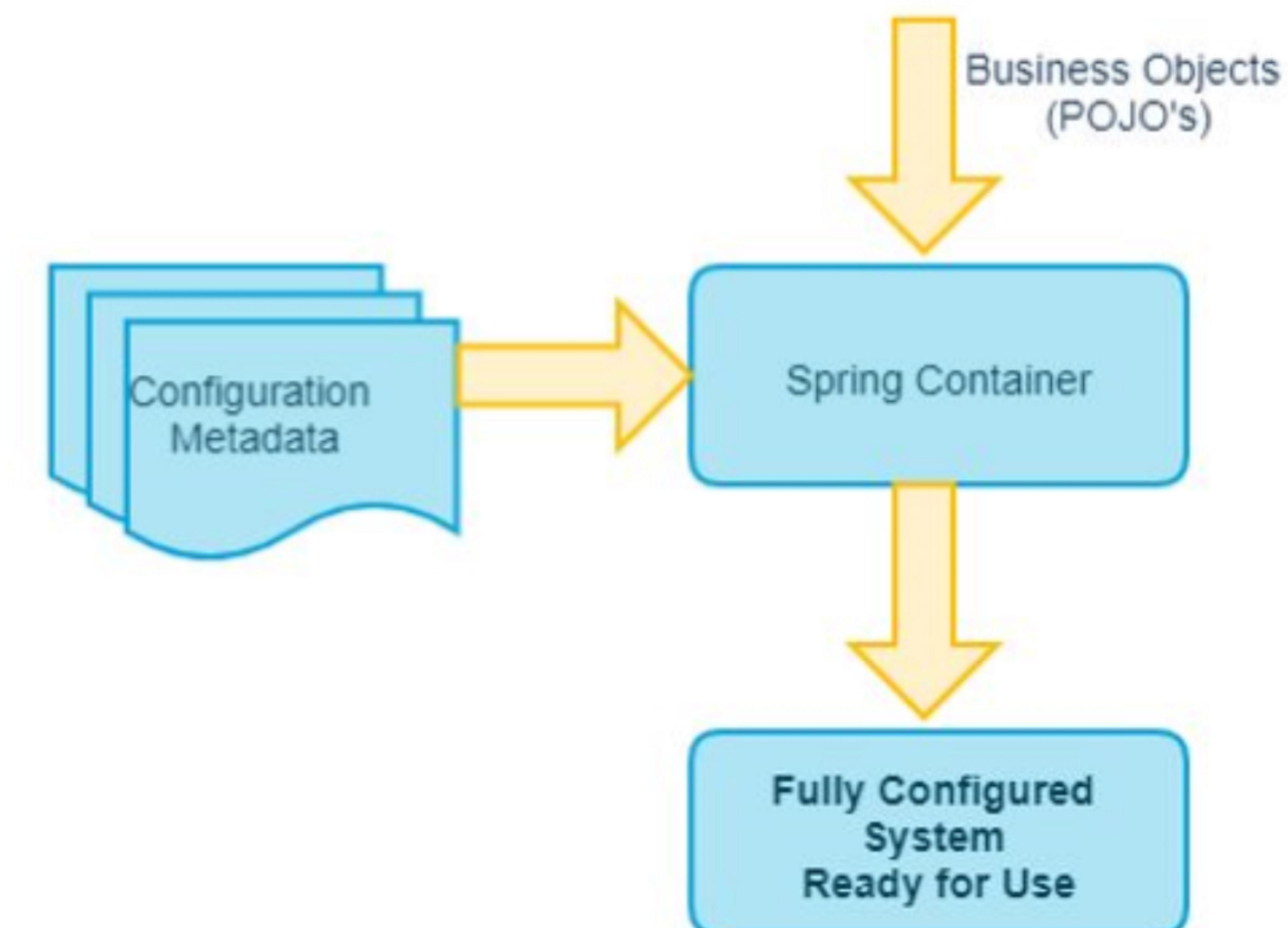
Spring IoC container - Manages DI
Configurations Metadata



POJO = Plain Old Java Object.

- ↳ In Spring framework, “`ApplicationContext`” represents the Spring IoC container.
- ↳ `ApplicationContext` manages all the beans by using the configuration metadata.
- ↳ `ApplicationContext` is an interface.
- ↳ Spring framework provides several implementations of the `ApplicationContext` interface.

- ↳ Some of the implementations are:-
 - ↳ ClassPathXMLApplicationContent
 - ↳ AnnotationConfigApplicationContext
- etc.



Types of DI

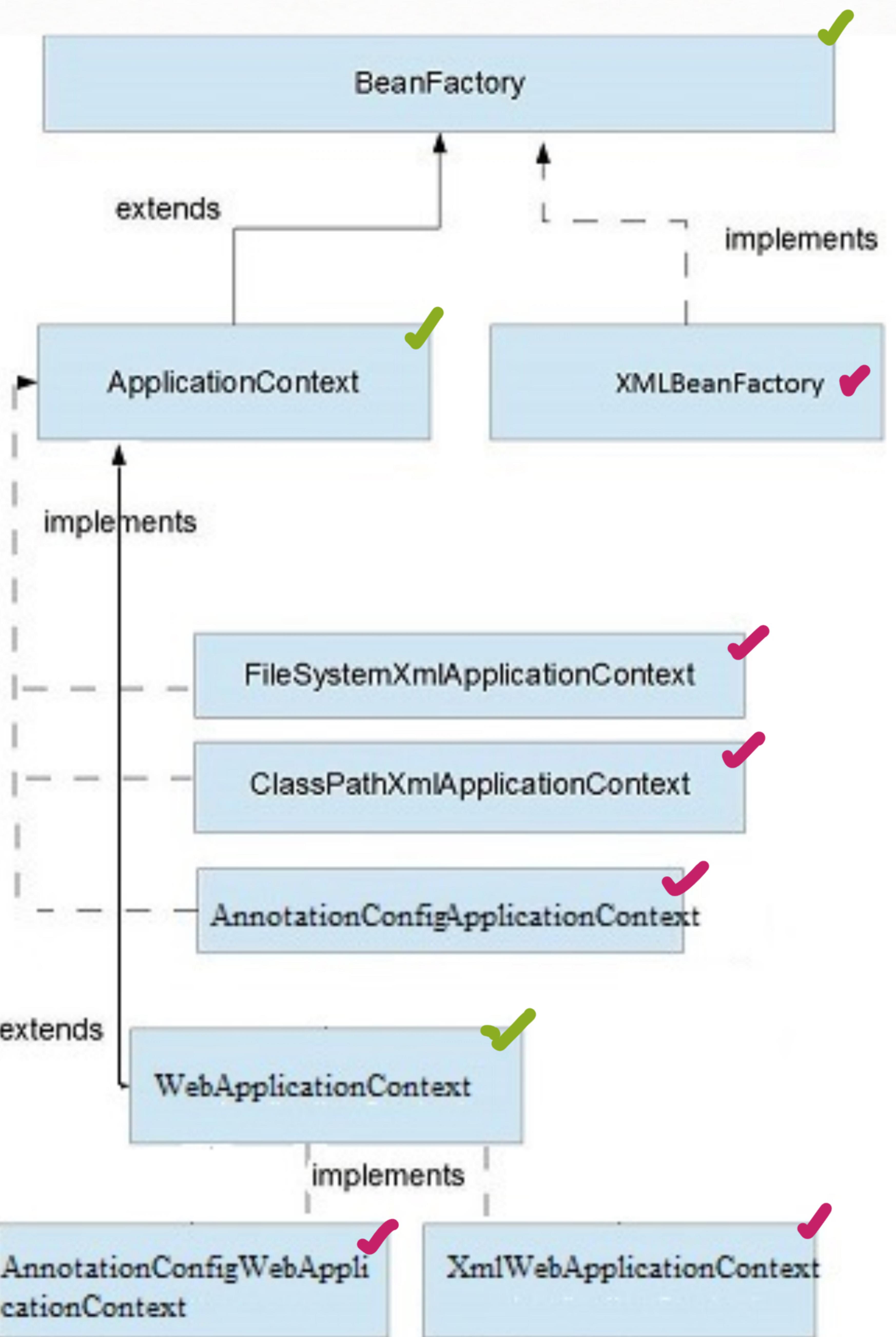
- i) Constructor Injection ✓
- ii) Property (Field) Injection ✗
- iii) Setter (Method) Injection ✓ (default)

✓ = 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 Contexts
- b) Annotation-Configured Application Contexts

Events associated with ApplicationContent :-

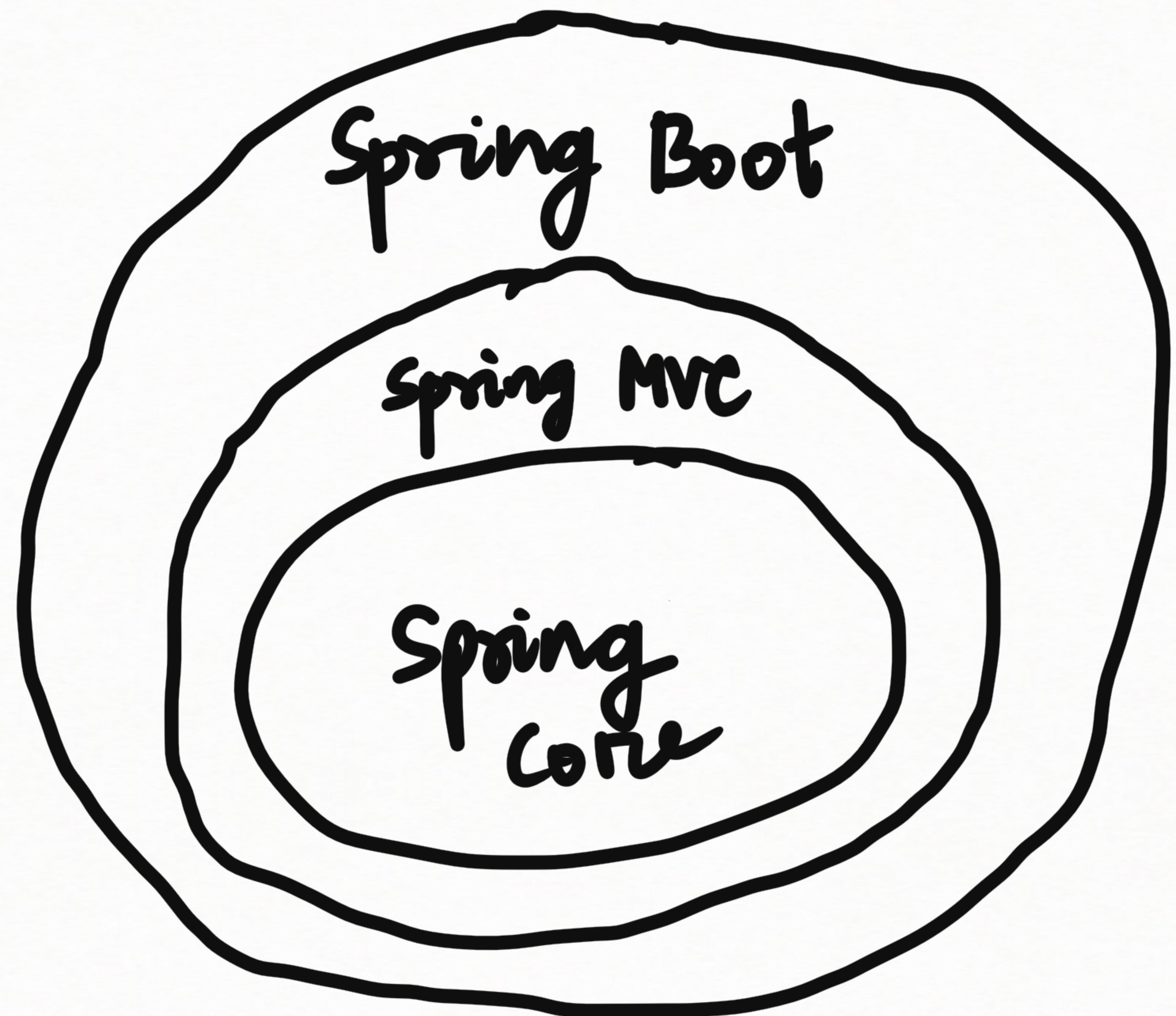
The following predefined/built-in container events are associated with the context object of every Spring application:-

- 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



XML Schema-Based Configuration

Example

```
1 package basicinjection;
2
3 public class Foo {
4     private String name;
5
6     public Foo() { }
7
8     public Foo(String name) {
9         this.name = name;
10    }
11
12    public void setName(String name) {
13        this.name = name;
14    }
15
16    public String getName() {
17        return name;
18    }
19
20 }
```

```
1 package basicinjection;
2
3 public class Bar {
4     private String name;
5     private int age;
6     private Foo foo;
7
8     public Bar() {}
9
10    public Bar(String name,int age) {
11        this.name = name;
12        this.age = age;
13    }
14    public void setFoo(Foo foo) {
15        this.foo = foo;
16    }
17
18    public void processFooName(){
19        System.out.println("Name in Injected Foo is: "+foo.getName());
20    }
21
22    @Override
23    public String toString() {
24        return "Bar has name = "+this.name+" and age = "+this.age;
25    }
26 }
```

To provide configuration metadata for the object reference "foo" in Bar 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 defⁿ
for Foo

bean
definition
for Bar

```
{   <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>
```

↳ constructor injection

↳ settu injection

→ Foo f = new Foo("Cleopatra");

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

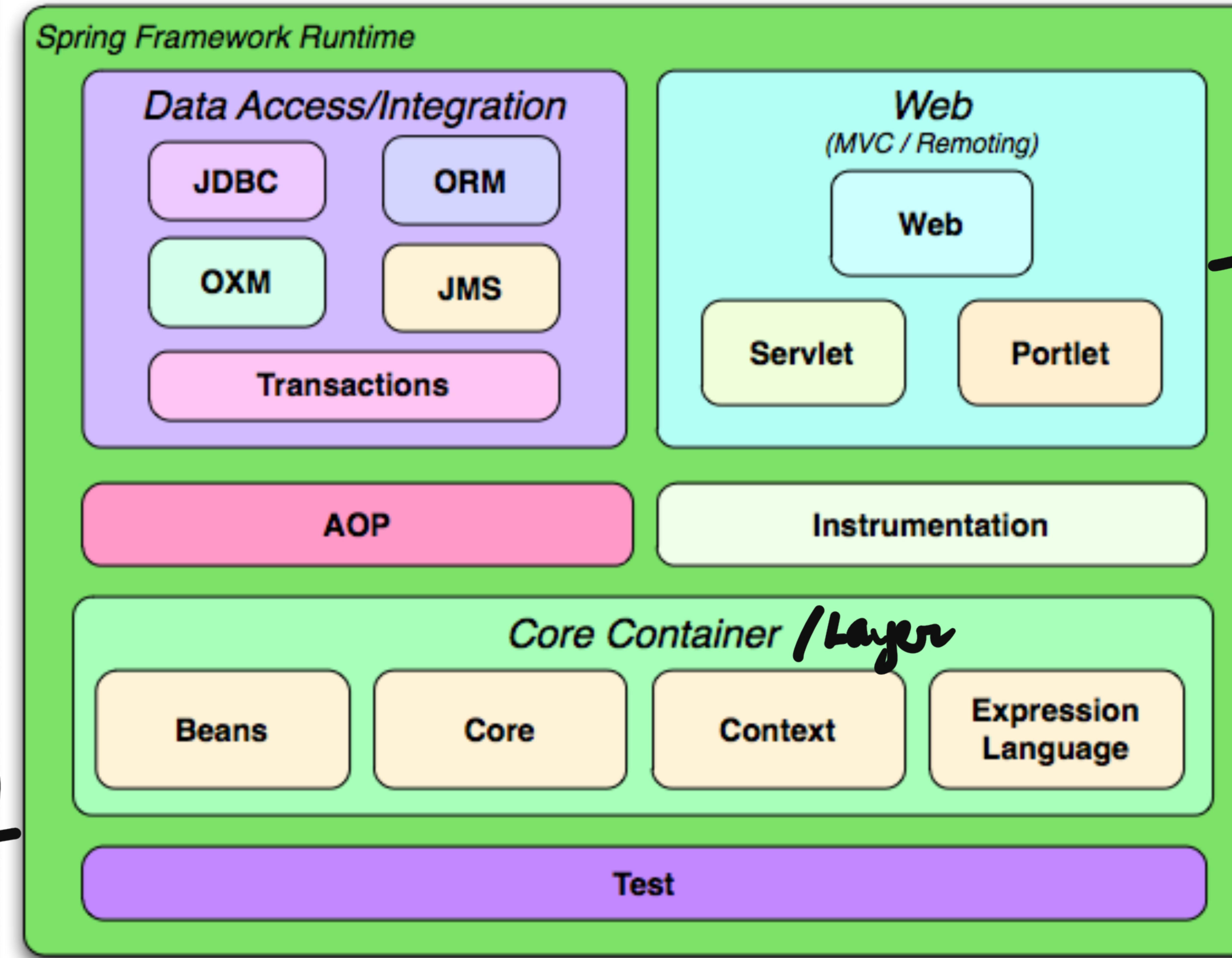
beans.xml or applicationContext.xml

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-expression)
= 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:-
 - ① Right-click on the project folder (in explorer)
 - ② Hover on Build path & choose Configure build path.
 - ③ Select "Classpath" under the Libraries tab
 - ④ 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.