What is Spring Framework ?

-> Spring is a Dependency injection Framework is to make Loosely Coupled Application .

-> Spring framework makes the easy development of JavaEE application.

-> It was developed by Rod Johnson in 2003

Tightly Coupled

In the context of the Spring framework, tight coupling refers to a situation where two or more components are highly dependent on each other. This can make the code difficult to maintain and modify over time. In Spring, dependency injection is used to achieve loose coupling between components, where each component is independent and can be easily replaced or modified without affecting the rest of the system.

What is IOC Container ?

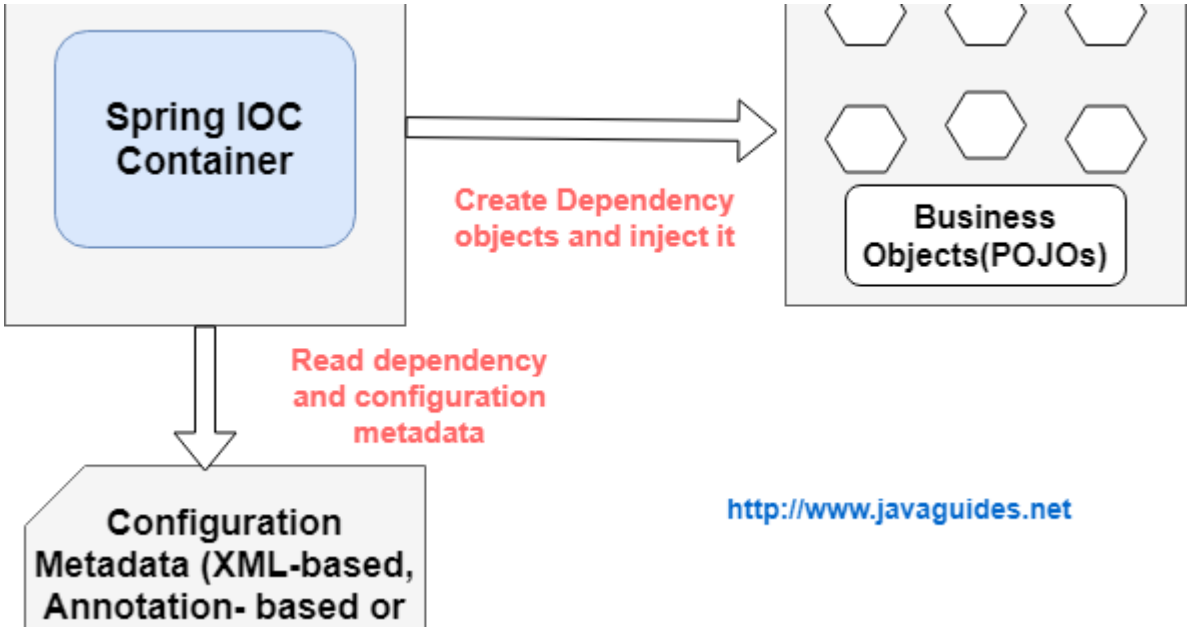
-> The IoC container is responsible to instantiate, configure and assemble the objects.

-> The IoC container gets informations 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



In Spring framework, IOC stands for Inversion of Control. IOC container is a core feature of Spring that manages the lifecycle of Java objects and their dependencies.

An IOC container is responsible for creating objects, wiring them together, configuring them, and managing their complete lifecycle from creation to destruction. The container achieves this by reading the configuration metadata from XML, Java annotations, or Java code.

There are two types of IOC containers in Spring: BeanFactory and ApplicationContext. The ApplicationContext is a more advanced container that provides additional features such as internationalization, event propagation, and integration with other frameworks.

Spring Core

The Spring Core dependency provides the basic building blocks of the framework, such as the IoC (Inversion of Control) container and the DI (Dependency Injection) mechanism. It also includes utilities for working with resources, such as files and streams, as well as support for internationalization and exception handling.

Spring Context

The Spring Context dependency builds on top of the Spring Core dependency and provides additional functionality for configuring and managing the application context. This includes support for AOP (Aspect-Oriented Programming), event handling, and integration with other frameworks and technologies, such as JPA (Java Persistence API) and JMS (Java Message Service).

Dependency Injection

-> Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application.

-> Dependency Injection makes our programming code loosely coupled.

Types of Dependency Injection ?

-> There are 2 types of dependency injection

1) Setter Injection

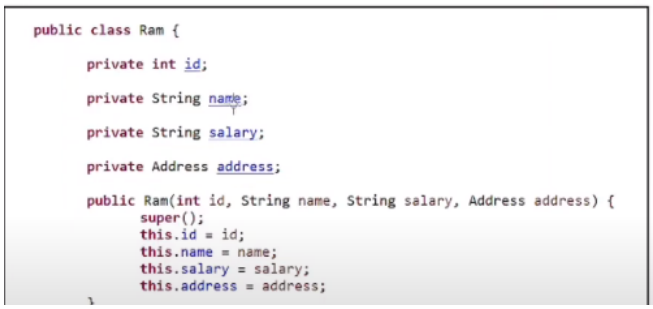
2) Constructor Injection

1) Setter Injection

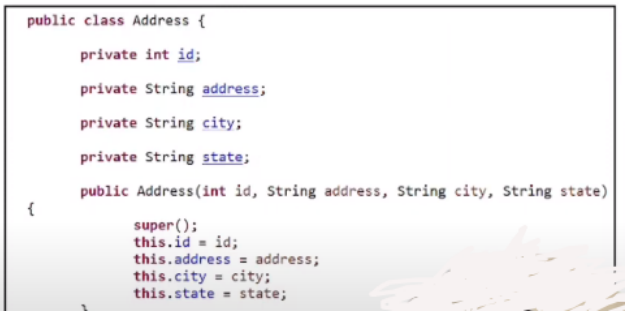
```
public class Ram {

	private int id;

	private String name;

	private String salary;

	private Address address;

	public void setId(int id) {
		this.id = id;
	}

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

	public void setSalary(String salary) {
		this.salary = salary;
	}

	public void setAddress(Address address) {
		this.address = address;
```

```
public class Address {

	private int id;

	private String address;

	private String city;

	private String state;

	public void setId(int id) {
		this.id = id;
	}

	public void setAddress(String addres
		this.address = address;
	}

	public void setCity(String city
		this.city = city;
	}

	public void setState(String st
		this.state = state;
```

2) Constructor Injection



```
public class Ram {

	private int id;

	private String name;

	private String salary;

	private Address address;

	public Ram(int id, String name, String salary, Address address) {
		super();
		this.id = id;
		this.name = name;
		this.salary = salary;
		this.address = address;
	}
```



```
public class Address {

	private int id;

	private String address;

	private String city;

	private String state;

	public Address(int id, String address, String city, String state)
{
		super();
		this.id = id;
		this.address = address;
		this.city = city;
		this.state = state;
	}
```

> Constructor Injection:

In this type of DI, dependencies are injected through a class constructor. The IoC container creates an instance of the class and injects the required dependencies into the constructor parameters. This approach ensures that the class is fully initialized before it is used.

> Setter Injection

In this type of DI, dependencies are injected through setter methods. The IoC container creates an instance of the class and calls the setter methods with the required dependencies. This approach allows for more flexibility in changing dependencies at runtime.

Data Types of Injection

Primitive type - byte, long,short, char, int,double

Collection type - list, set , map

Reference type - object

```
<bean name="st2" class="org.example.primitive.Student">
	<property name="id">
	    <value>122</value>
	</property>
	<property name="name">
	    <value>Pavy</value>
	</property>
```

```
        <property name="address">
            <value>Himachal</value>
        </property>
    </bean>
```

This is an example of XML configuration for a Spring bean named "st2" of c
"id", "name", and "address".
The property tag is used to set the values of the properties. The name att
tag contains the value to be set.
In this example, the "id" property is set to 122, the "name" property is s
"Himachal". These values will be injected into the corresponding propertie
the bean is created by the Spring IoC container.

```
<property name="name" value="aashu"/>

        <property name="address" >
            <list>
                <value>india</value>
                <value>usa</value>
                <null></null>
            </list>
        </property>

        <property name="phno">
            <set>
                <value>4354664253</value>
                <value>5634756745</value>
            </set>
        </property>
        <property name="courses">
            <map>
                <entry key="java" value="2 monts" />
                <entry key="python" value="3 monts"/>
            </map>
        </property>
```

This is an XML snippet that defines a bean named "st1" of class "org.e

The bean has four properties: "name", "address", "phno", and "courses".

```
The "name" property is a simple string value of "aashu".

The "address" property is a list of three string values: "india", "usa", and r

The "phno" property is a set of two string values: "4354664253" and "563475674

The "courses" property is a map with two key-value pairs: "java" with a value
```

> ## BEAN LIFE CYCLE

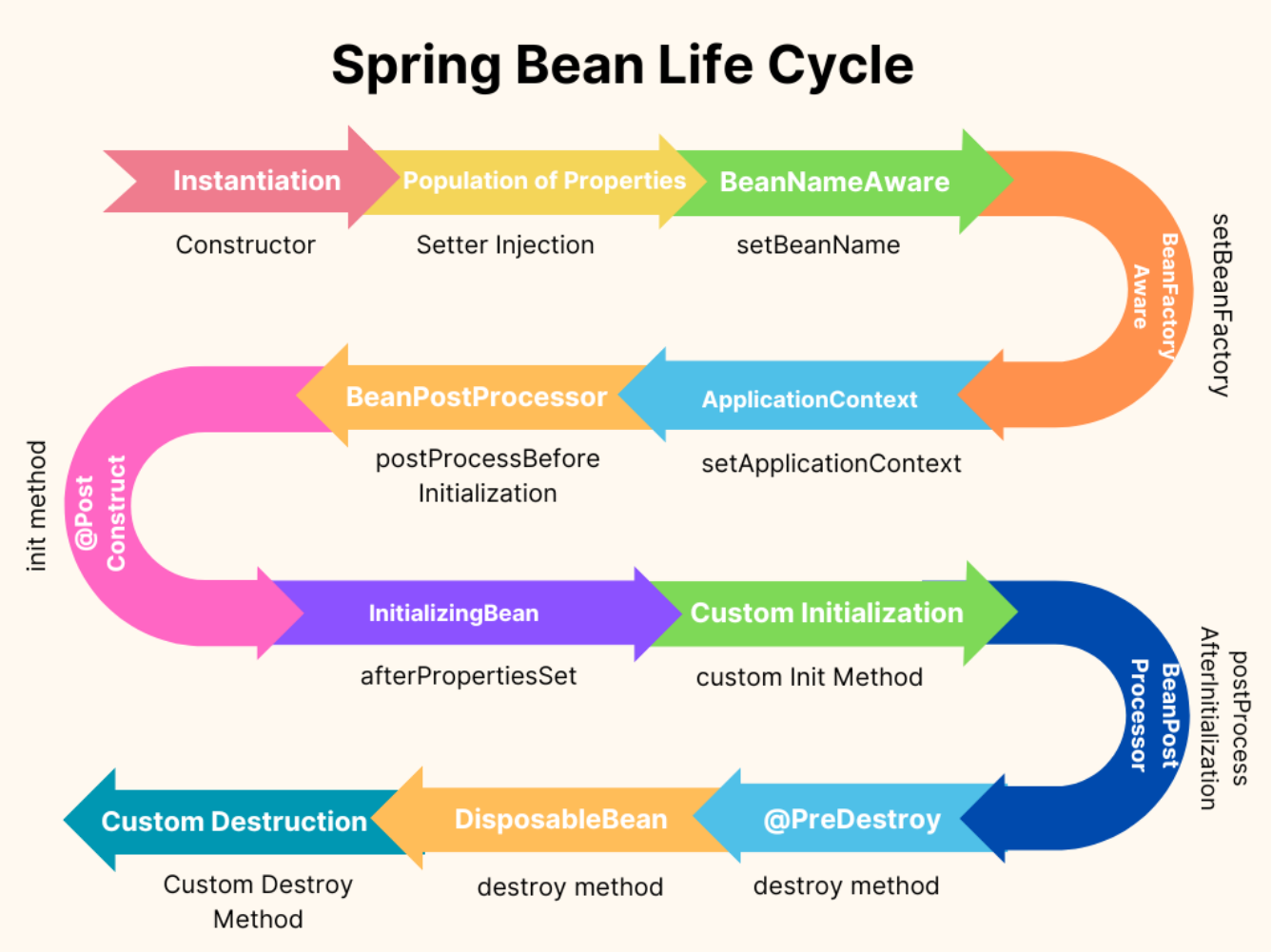
There are three common ways to implement the Spring bean lifecycle:

XML-based configuration: This approach involves defining the lifecycle methods in the XML configuration file using the init-method and destroy-method attributes in the element. This approach is simple and easy to understand, but it can be verbose and difficult to maintain for large applications.

Implementing the InitializingBean and DisposableBean interfaces: This approach involves implementing the afterPropertiesSet() method from the InitializingBean interface for initialization logic, and implementing the destroy() method from the DisposableBean interface for destruction logic. This approach is more flexible than XML-based configuration, but it requires modifying the bean class, which may not always be feasible.

Annotation-based configuration: This approach involves using annotations like @PostConstruct and @PreDestroy to define the initialization and destruction methods respectively. This approach is concise and easy to maintain, but it requires

a dependency on Spring's annotation processing.



init()

The init method is used to define the initialization logic for a bean. It is called after the bean has been instantiated and its dependencies have been injected. You can use this method to perform any necessary setup or initialization tasks for your bean.

destroy()

The destroy method is used to define the destruction logic for a bean. It is called when the container is shutting down or when the bean is being removed from the container. You can use this method to perform any necessary cleanup or finalization tasks for your bean.

registerShutdownHook()

When you call the **registerShutdownHook()** method on an instance of **AbstractApplicationContext**, it registers a shutdown hook with the JVM. This shutdown hook ensures that the Spring application context is properly closed and all resources are released when the JVM is shutting down.

By registering a shutdown hook, you ensure that the destroy methods of the beans within the application context are called, allowing them to perform any necessary cleanup or finalization tasks before the application exits.