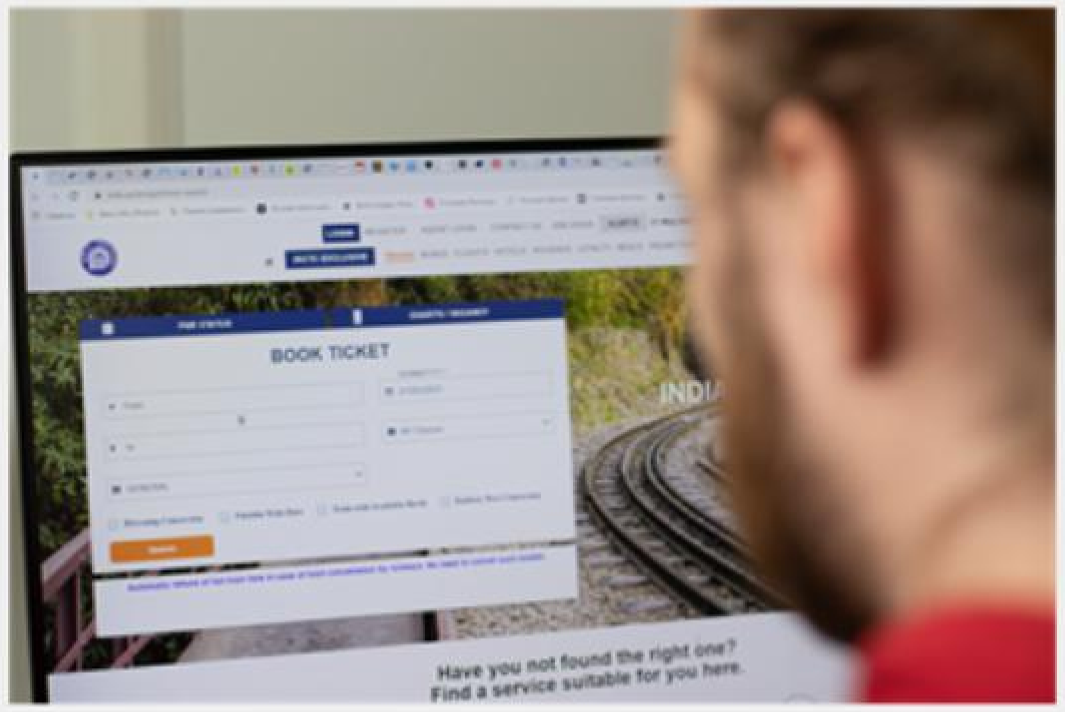
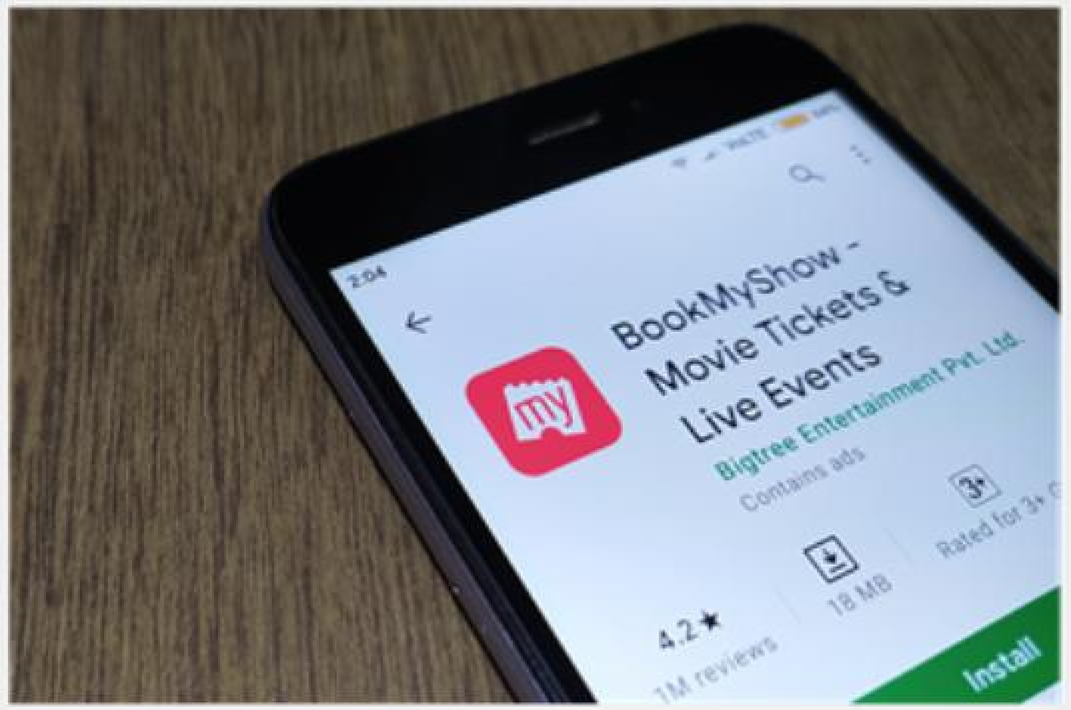
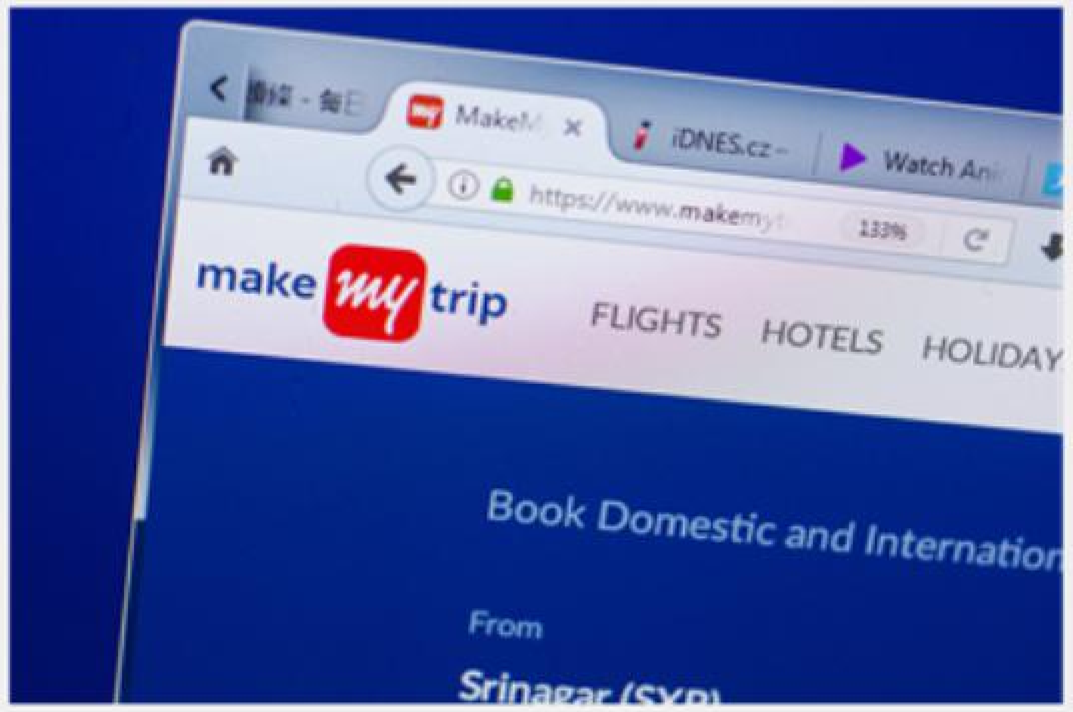
Slide Note

What Do These Websites Have in Common?


BOOK TICKET
INDIA
Have you not found the right one?
Find a service suitable for you here.


BookMyShow - Movie Tickets & Live Events
Bigtree Entertainment Pvt. Ltd.
Contains ads
4.2★
1M reviews
18 MB
Rated for 3+
Install



Menu

Slide Note

Think and Tell!

- You must sign up on the sites to use their services.
- Each site provides a payment option.
- Do you think these are common features among similar sites?
- Do you use these websites regularly to book a flight, hotel, movie, or dinner?

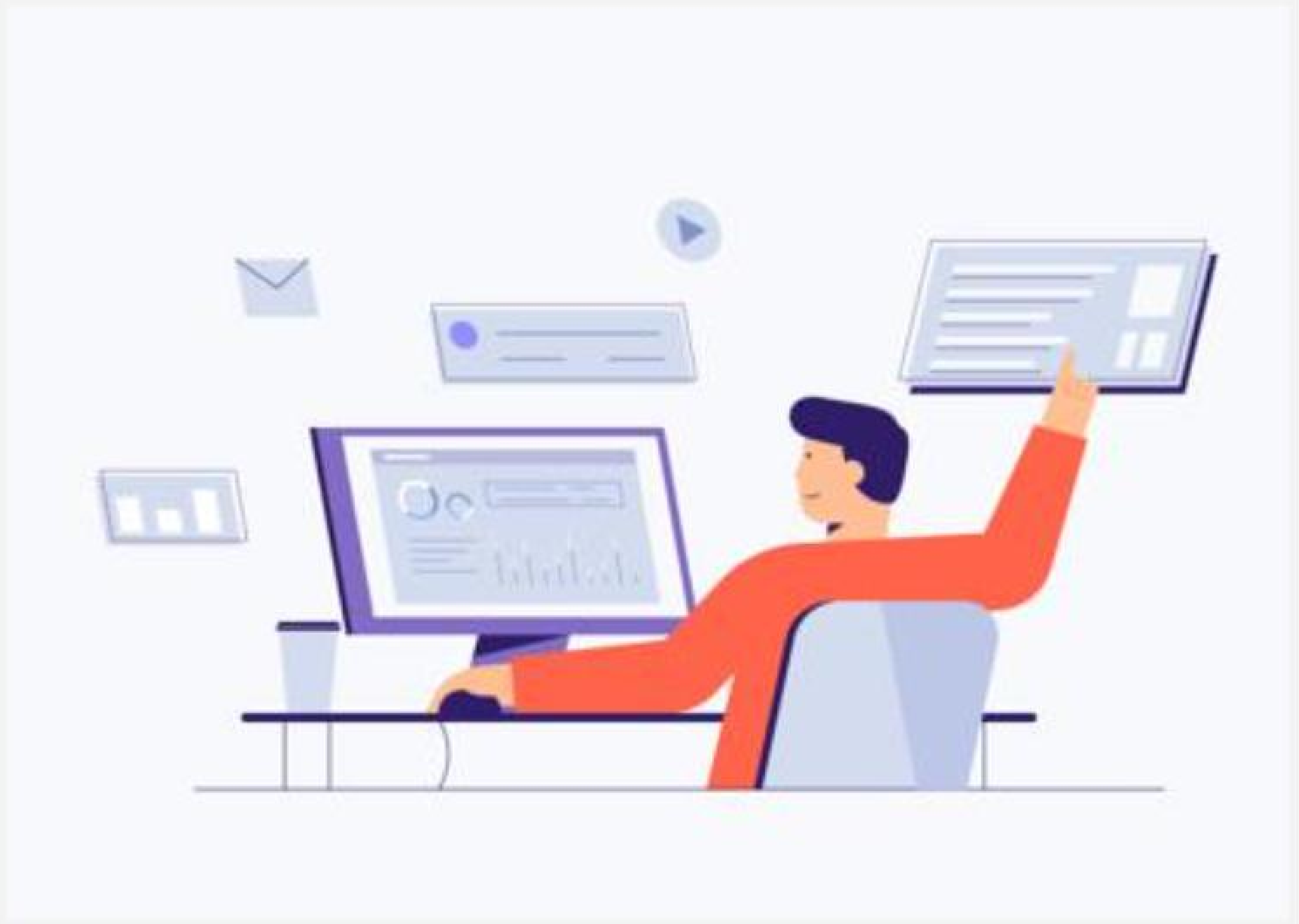
Menu

00:02 00:00:31 02/ 31

Slide Note

Think and Tell!

- Should a developer use one software to develop all the modules of an application, such as login, payment, etc., or should they use different software for each function?
- Should the developer use different software to test and maintain the application?
- If multiple software applications are used, will the application become heavy, affecting performance?
- If a new feature must be added, how should it be done?
- Should each feature of the application depend on the others or be loosely coupled (i.e., independent of one another)?



Menu

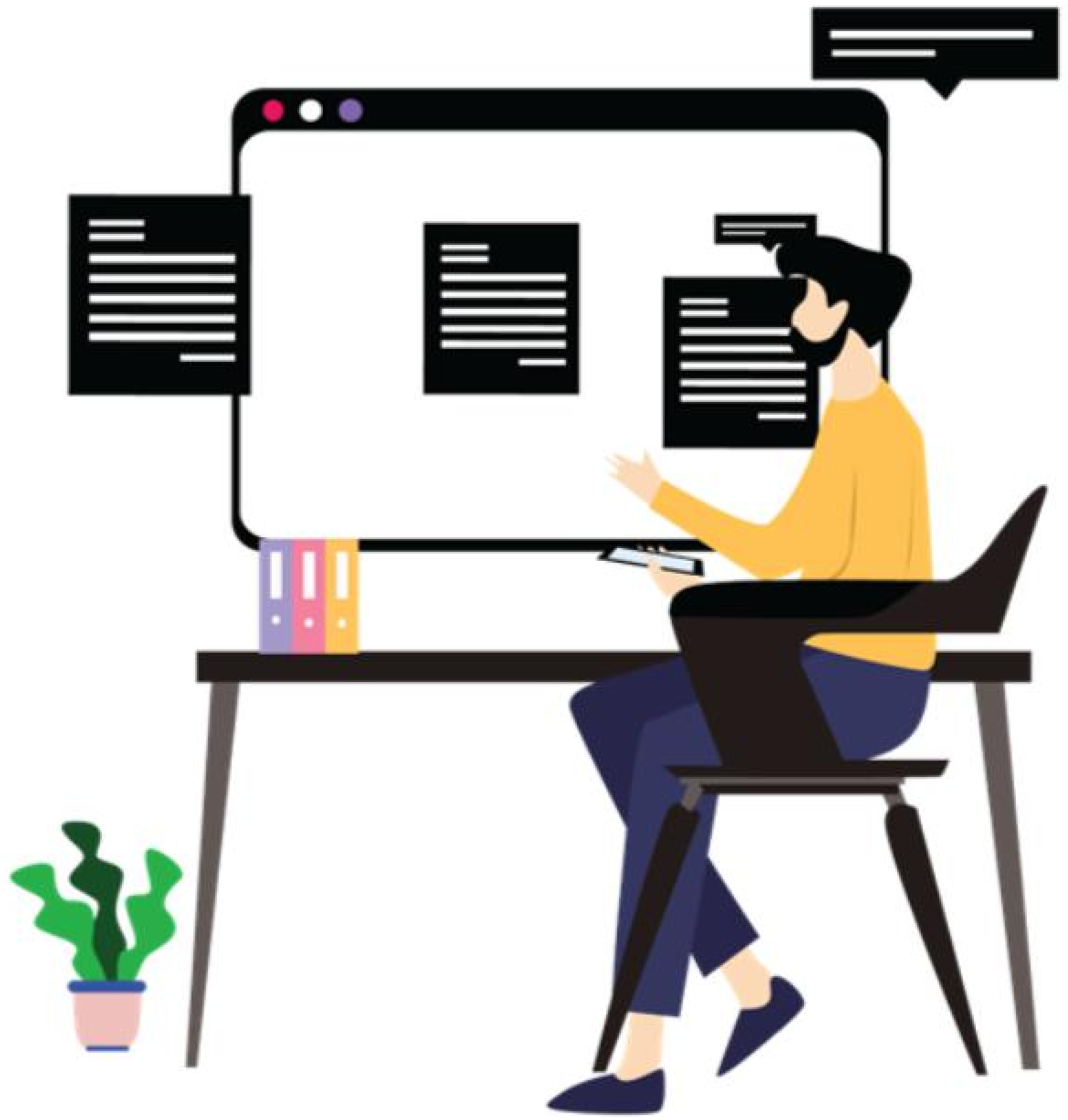
Slide Note

Object Lifecycle

- When creating objects for all the classes in Java, do you use the new keyword multiple times as below?
  - `Employee empSam = new Employee();`
  - `Employee empTom = new Employee();`
  - `Employee empDan = new Employee();`
- Can the complexity of handling multiple objects be reduced?
- Can the handling and lifecycle of the objects be done by an independent framework or system?

Slide Note

Develop Backend Application by Using Spring Framework



Menu

00:05 00:00:31 05/ 31

Slide Note

Learning Objectives

- Explore the Spring Framework
- Configure the Spring Core Container
- Define Beans in the Spring Core Container

Menu

Slide Note

Exploring the Spring Framework

Menu

00:07 00:00:31 07/ 31

Slide Note

Framework

- A framework is a large body of predefined code to which we can add our own code to solve a problem in a specific domain.
- Frameworks are based on architecture and design, and they force users to follow them.
- The classes, or application programming interfaces (APIs), that are provided by the framework are efficient, secure, less expensive, and provide good support for the developer who is going to use them.
- Beginners may not know certain design principles and architectures, but framework guidelines help to code automatically with chosen design principles.

Menu

Slide Note

The Spring Framework

- Spring is a powerful, flexible, fast, secure, and lightweight framework used for building Java web applications.
- The Spring Framework is a well-defined tool that supports several web applications using Java as a programming language.
- Spring is an open-source project and has a large and active community.
- The Spring framework is divided into modules.
- Modules are a set or package of classes that provide functionality for the Spring framework.
- Applications can choose which modules they need.

Slide Note

● Core Container

The *Core Container* consists of the Core, Beans, Context, and Expression Language modules.

The *Core and Beans* modules provide the fundamental parts of the framework, including the including the Inversion of Control and Dependency Injection features.

● Data Access/Integration

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

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

● Web

The *Web* layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

Spring's *Web* module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.

● AOP (Aspect-Oriented Programming)

Spring's *AOP* module provides an *AOP Alliance*-compliant, aspect-oriented programming implementation.

Instrumentation

The *Instrumentation* module provides class instrumentation support and classloader implementations to be used in certain application servers.

Test

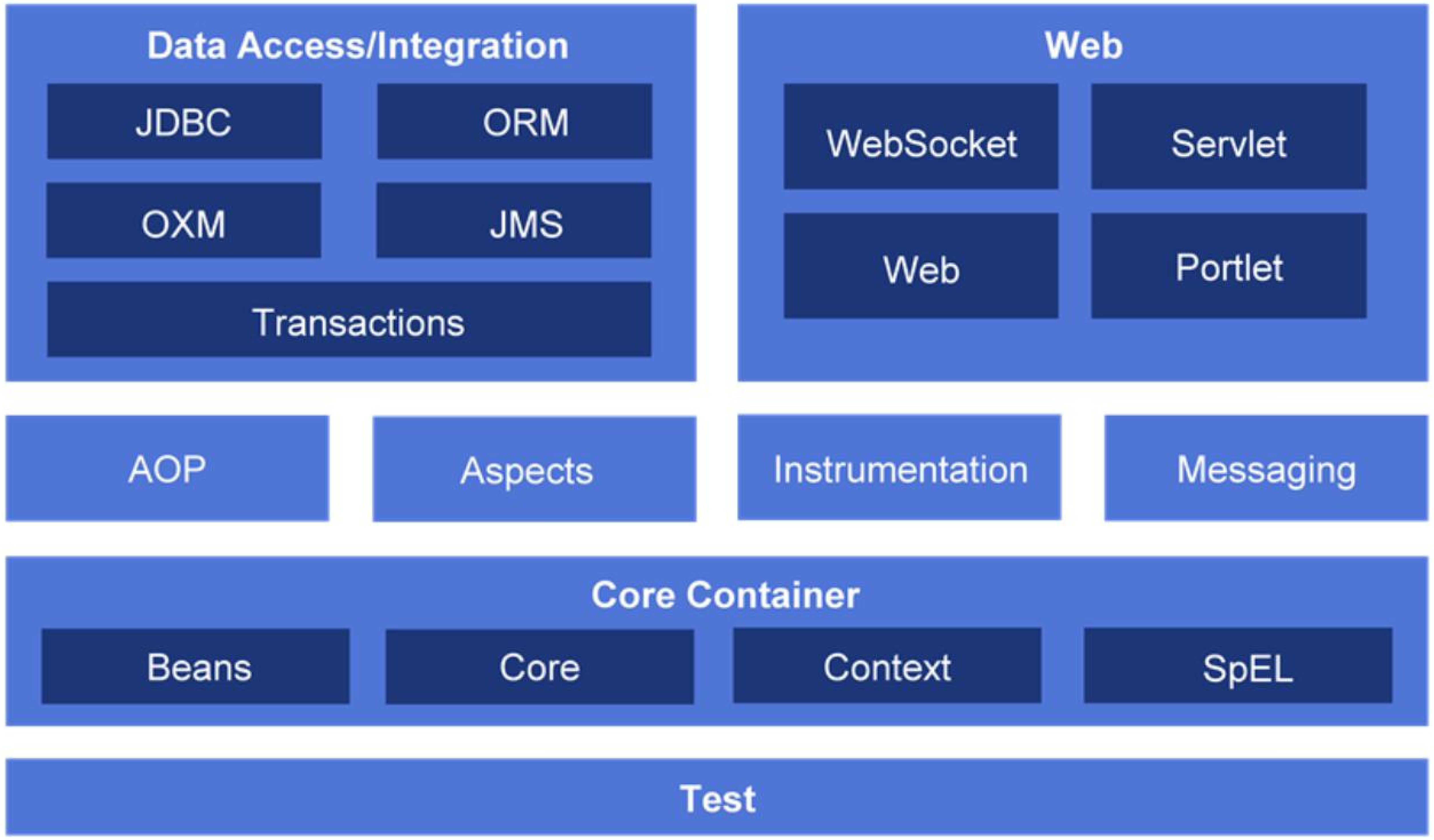
The *Test* module supports the testing of Spring components with JUnit or TestNG.

Components of Spring

- The Spring framework consists of features organized into about 20 modules, or independent functionalities.
- These modules are grouped into:
  - Core Container
  - Data Access/Integration
  - Web
  - AOP (Aspect-Oriented Programming)
  - Instrumentation
  - Test

Slide Note

Spring Architecture



Note: Only Spring core container will be discussed in this session

Slide Note

Configuring the Spring Core Container

Menu

00:12 00:00:31

12/ 31

Spring Core Container

- The Spring core container is the core module of the Spring framework.
- The container is responsible for:
  - creating application objects – The core container creates all the objects of the application; the objects are also called beans.
  - wiring the objects together – The core container manages how the beans must be used. If a class requires a bean, then the container injects the bean into the class.
  - configuring the objects – The beans must be configured using a configuration file, which informs the core container that all beans in the file can be managed by Spring.
  - managing the complete lifecycle of the objects – Beans are managed by Spring from creation to destruction.

Slide Note

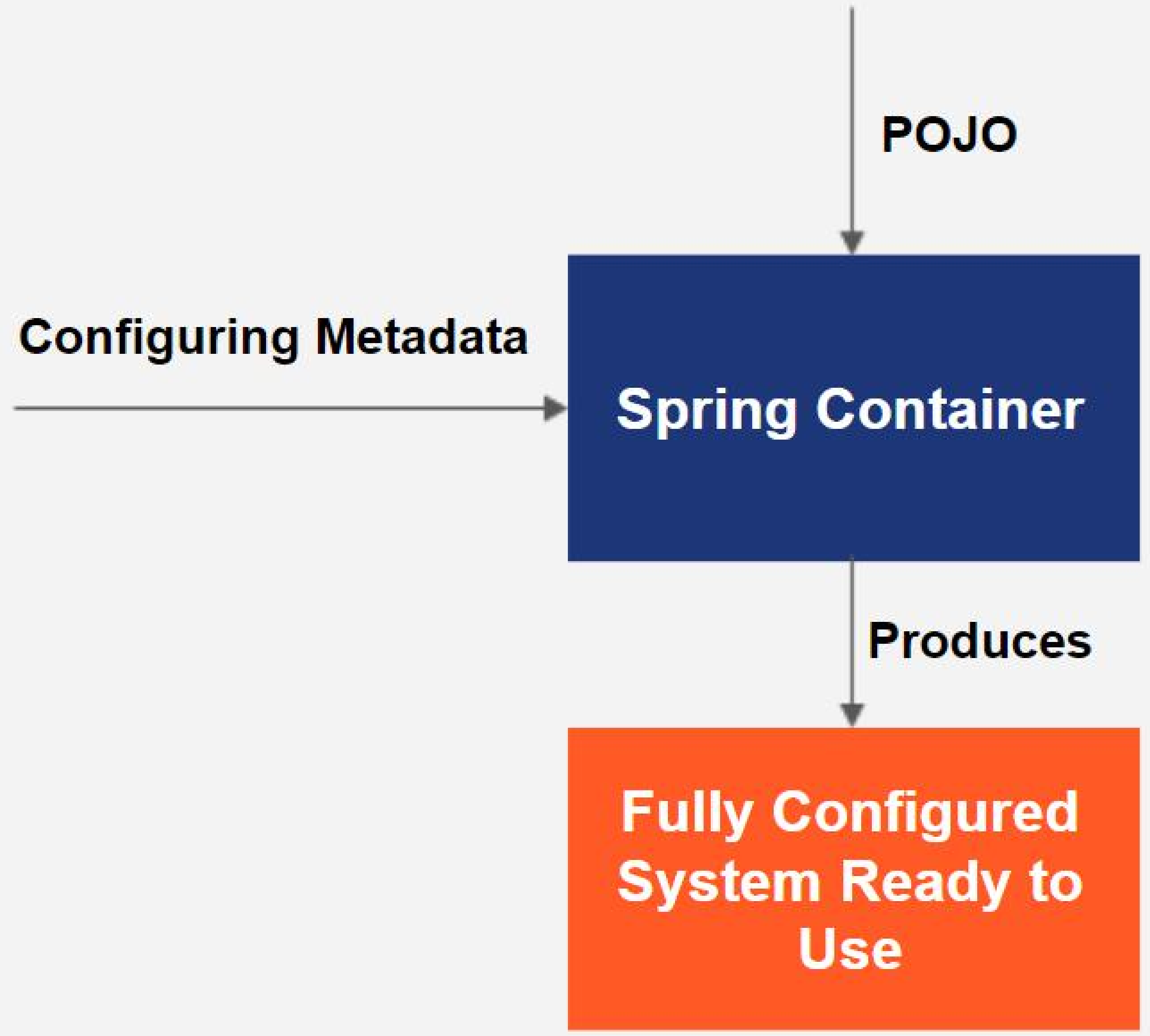
IoC and DI

- The two most important aspects of the core container are:
    - Inversion of Control (IoC)
    - Dependency Injection (DI)
- IoC is a mechanism by which the control of objects is transferred to a container or framework.
- It is used in the context of object-oriented programming.
- DI is a pattern that can be used to implement IoC, where the control is inverted and given to the framework.
- The framework itself does connect objects with other objects or inject them into other objects.

Menu

Inversion of Control (IoC)

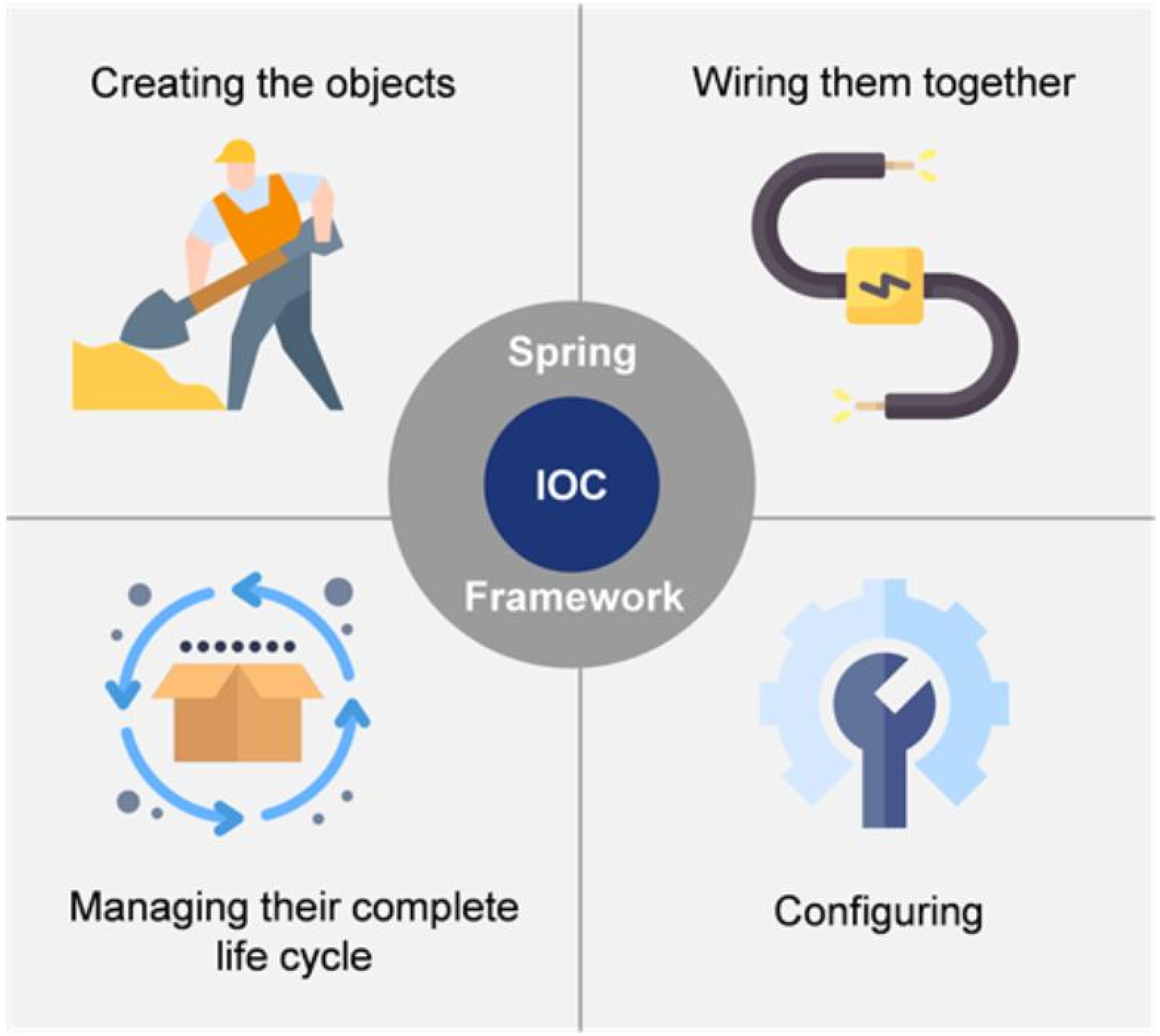
- In large applications, programmers must:
  - programmatically create all objects.
  - manage to map relations between the objects.
  - handle any lifecycle events to have complete control.
- While this is suitable for small applications, for large applications with hundreds and thousands of objects, it's tedious and error-prone.
- With IoC, control is inverted; instead of the programmer, the Spring framework has control.
- We simply instruct Spring where we need the objects, and Spring will create, manage, and inject them at the point requested.



POJO classes are the objects or beans of the application.

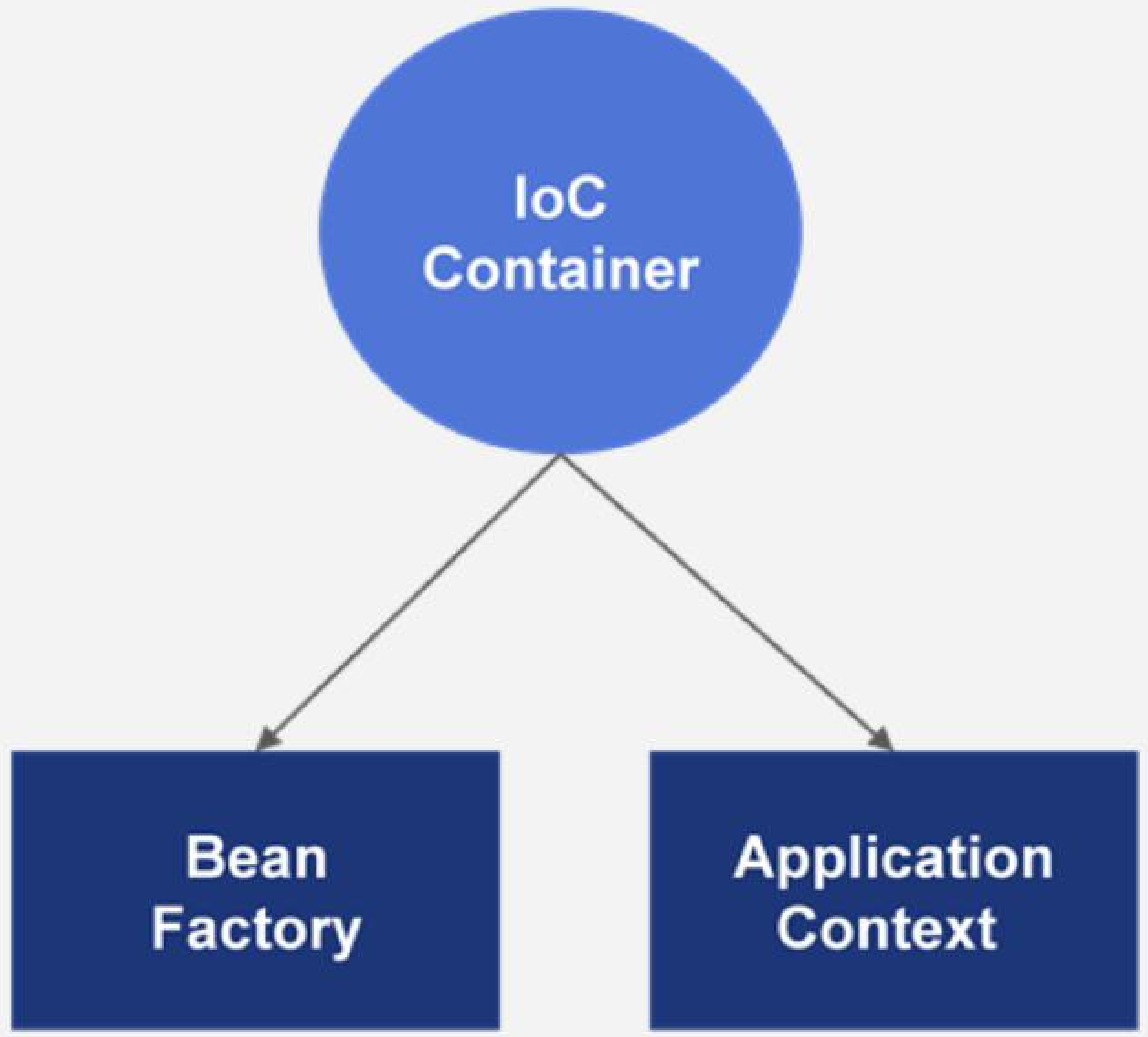
Slide Note

Inversion of Control (IoC) Explained



- Creating Objects – control over object creation is given to the Spring container.
- Configuring – reading configuration files that will be used in the application.
- Wiring – when a class is dependent on another class, the object of the other class is also created.
- The Spring IoC Container – uses Java POJO classes and configuration metadata to produce a fully configured and executable system or application.

Slide Note



IoC Container

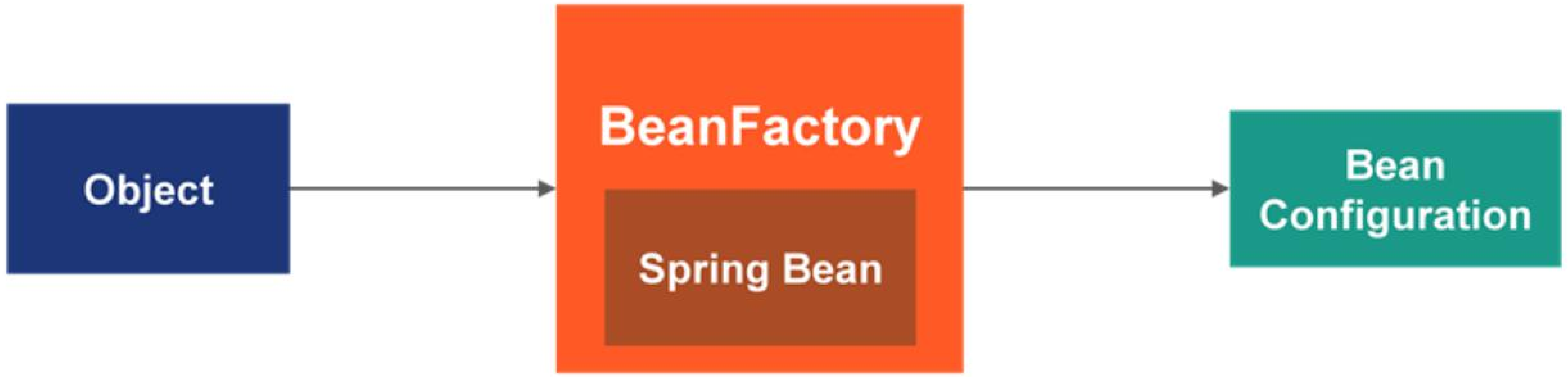
- There are two types of IoC containers:
  - `BeanFactory` – Provides a configuration framework and basic functionality to manage objects.
  - `ApplicationContext` – A sub-interface of the `BeanFactory` that provides more enterprise-specific functionality.

Menu

Slide Note

Spring-Managed Objects

- Spring has a `BeanFactory` to create new objects.
- Instead of a hard-coded new object, Spring `BeanFactory` creates an object.
- To create objects, or beans, `BeanFactory` reads from the configuration file that contains the bean definition.
- Since the new bean is created in the `BeanFactory`, Spring knows and manages the lifecycle of this bean. Spring acts as a container for this new bean or object created.



Menu

Slide Note

Dependency Injection

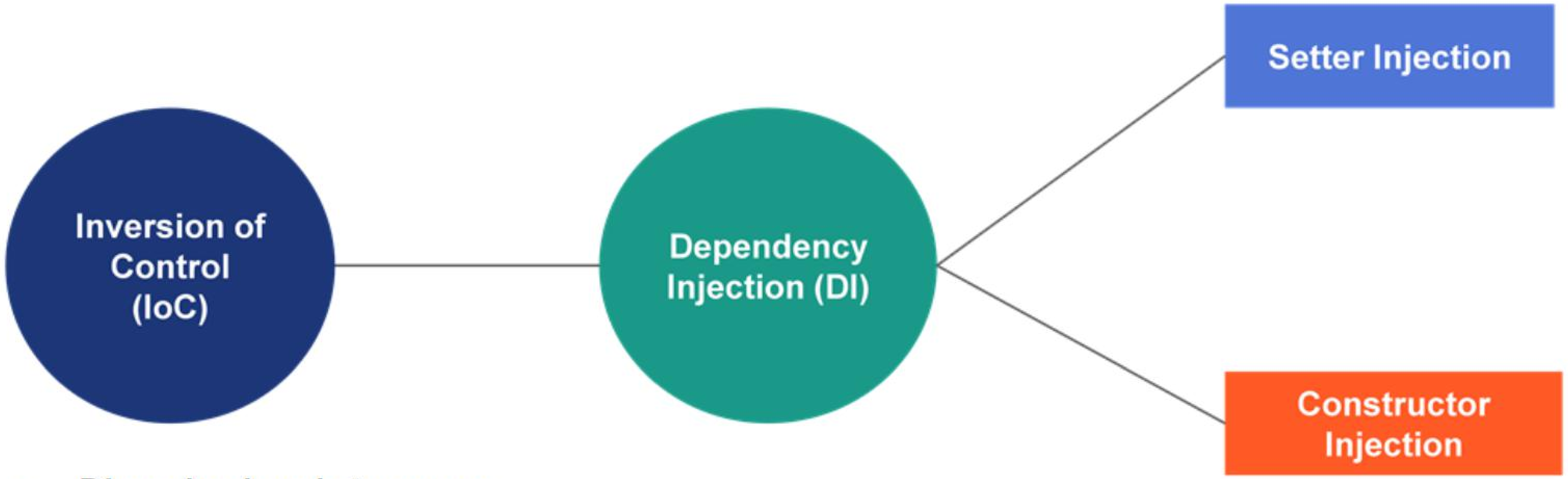
- The beans are now created and managed by Spring. Other objects of the application can use these objects or beans.
- DI is a fundamental aspect of the Spring framework, through which the Spring container injects objects into other objects or dependencies of the application.
- It is a design pattern used to implement IoC.
- It allows the creation of dependent objects outside of a class and provides those objects to a class differently.
- Using DI, we move the creation and binding of the dependent objects outside the class that depends on them.

Menu

00:19:00:00:31 19/ 31

Slide Note

Dependency Injection – Injecting Objects



- DI can be done in two ways:
    - Setter – Si is accomplished by the container calling the setter methods on the beans after invoking a no-argument constructor to instantiate the bean.
    - Constructor – Constructor-based DI is accomplished when the container invokes a class constructor with the specified number of arguments, each representing a dependency on the other class.

Slide Note

Quick Check

Identify the IoC containers in Spring.

1. BeanFactory, ApplicationContext
2. BeanFactory, ApplicationContext, and IocContextFactory
3. BeanFactory, BeanContext, and IocContextFactory
4. BeanFactory, ApplicationContext, and BeanContext

Menu

00:21:00:00:31 21/ 31

Slide Note

Quick Check: Solution

Identify the IoC containers in Spring.

1. BeanFactory, ApplicationContext
2. BeanFactory, ApplicationContext, and IocContextFactory
3. BeanFactory, BeanContext, and IocContextFactory
4. BeanFactory, ApplicationContext, and BeanContext

Menu

Slide Note

Defining Beans in the Spring Core Container

Menu

00:23 00:00:31 23/ 31

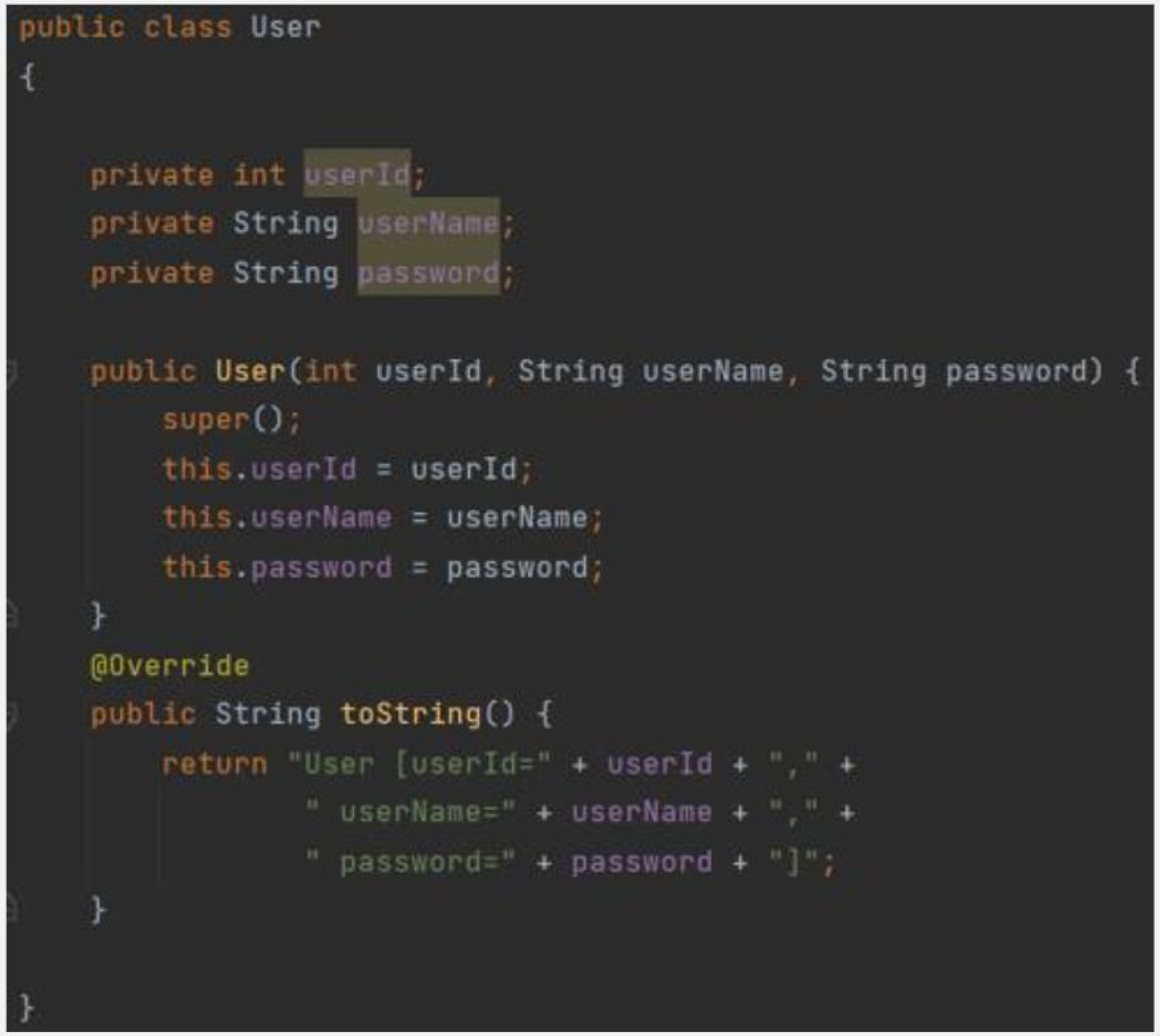
Steps for Configuration

1. Create a Java Maven project with `archetype` as `quickstart` and add the dependency below in `pom.xml;` this will set up the `BeanFactory` and `ApplicationContext`.

```xml
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.22</version>
</dependency>
```

2. Create the domain classes whose objects will be handled within the Spring container.
3. Define a configuration class that will have all bean definitions for the domain class objects. The `@Bean` will be used here.
4. Use the `AnnotationConfigApplicationContext` class to access the beans declared within the container.

Slide Note



```
public class User
{

    private int userId;
    private String userName;
    private String password;

    public User(int userId, String userName, String password) {
        super();
        this.userId = userId;
        this.userName = userName;
        this.password = password;
    }
    @Override
    public String toString() {
        return "User [userId=" + userId + "," +
                " userName=" + userName + "," +
                " password=" + password + "]";
    }


}
```

Domain Class

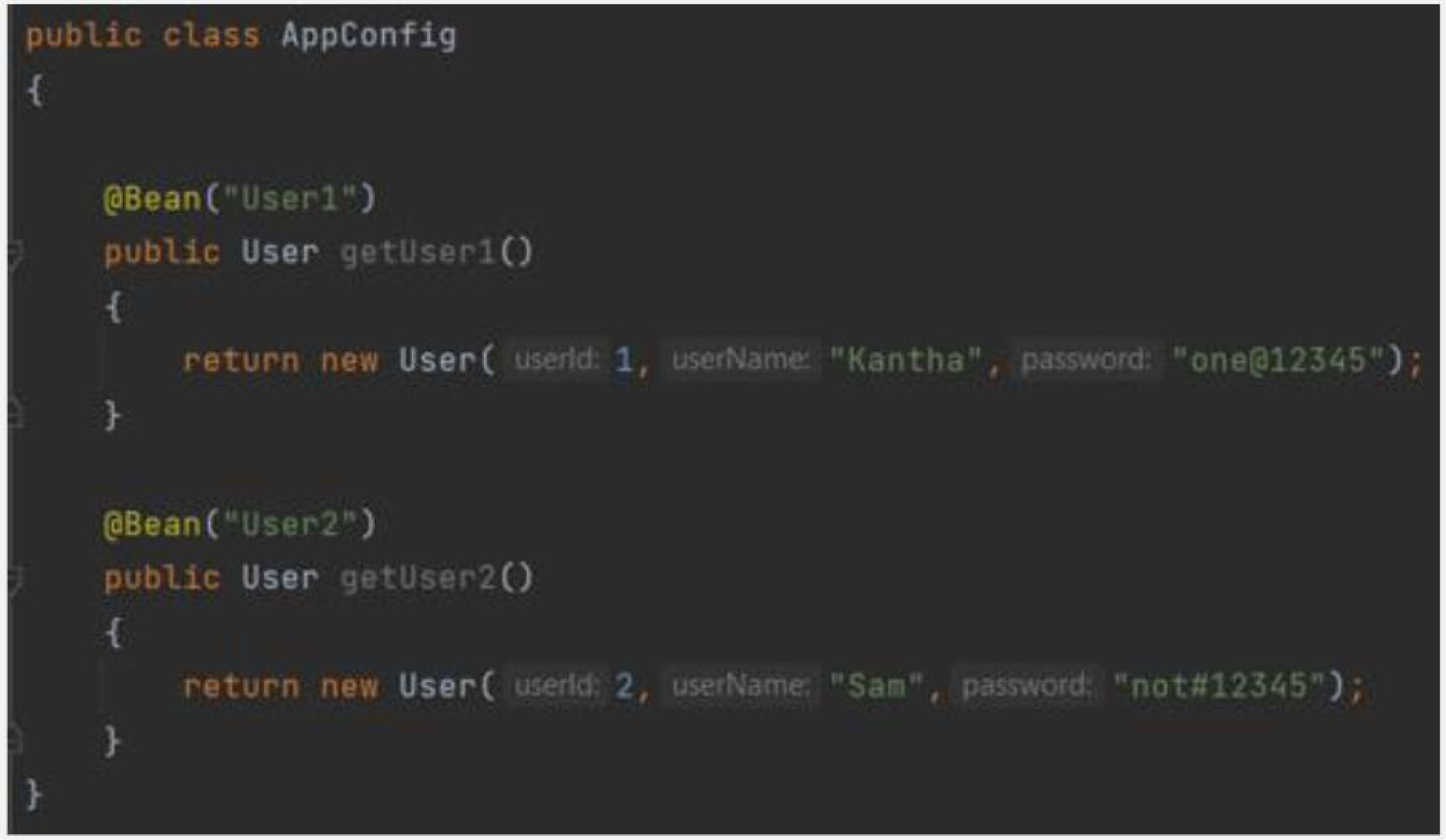
- Create a domain class that the application will use.
- User is a class that has userId, username, and password as attributes.

Menu

Slide Note

Configure the Beans

- In a configuration class, define the beans or objects along with values assigned for their attributes.
- The `@Bean` annotation tells Spring that the core container must handle the object or bean.
- The bean name can also be defined in the annotation.
- `User1` and `User2` are the two beans defined with different values for their attributes.



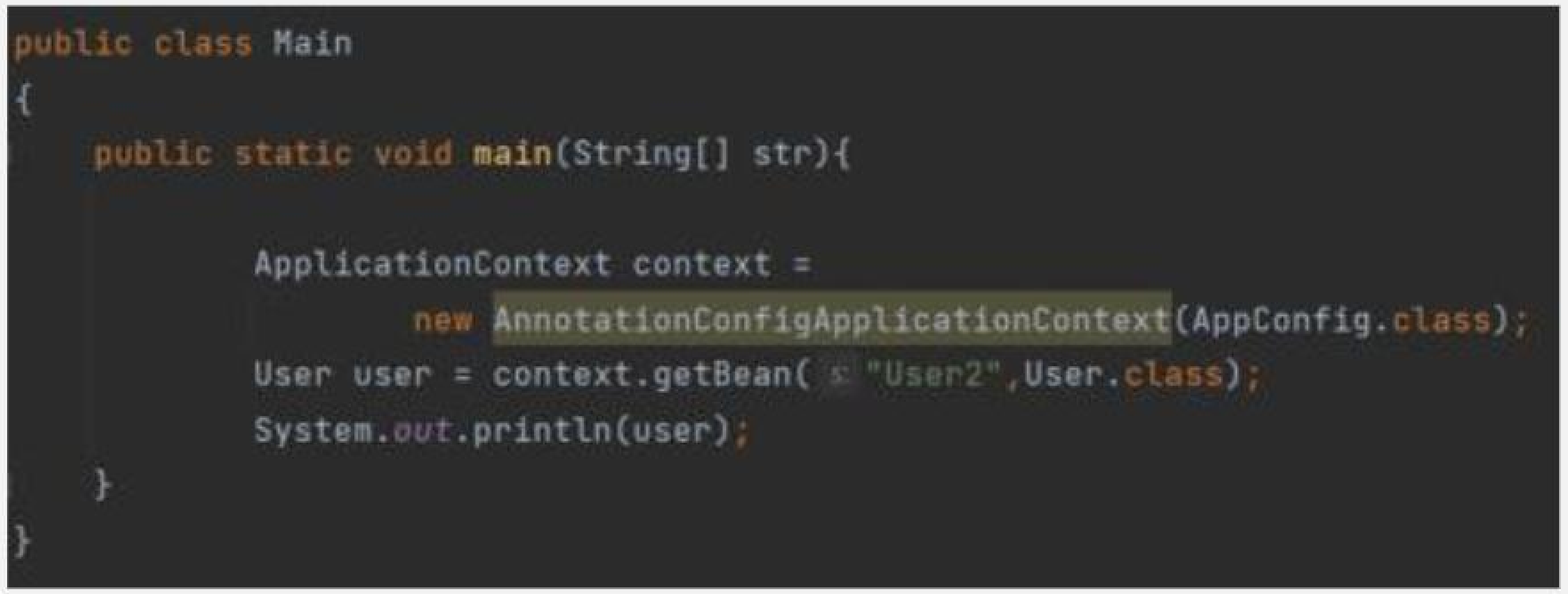
```
public class AppConfig
{

    @Bean("User1")
    public User getUser1()
    {
        return new User( userId: 1, userName: "Kantha", password: "one@12345");
    }

    @Bean("User2")
    public User getUser2()
    {
        return new User( userId: 2, userName: "Sam", password: "not#12345");
    }
}
```

Menu

00:26:00:00:31 26/ 31

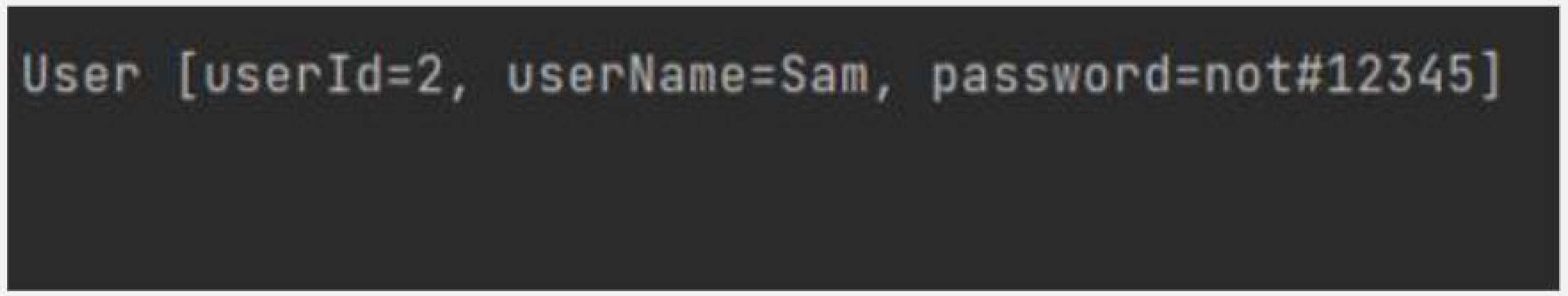
Slide Note

Use the Beans in the Application



```java
public class Main
{
    public static void main(String[] str){

            ApplicationContext context =
                    new AnnotationConfigApplicationContext(AppConfig.class);
            User user = context.getBean( s: "User2",User.class);
            System.out.println(user);
    }
}
```

Output



```
User [userId=2, userName=Sam, password=not#12345]
```

- **In a Main class, call the beans through the** `ApplicationContext` **and display the values.**
- **The** `AnnotationApplicationContext` **implements the** `ApplicationContext` **interface .**
- **The configuration file, i.e., the** `AppConfig.class`, **is passed as a parameter.**
- **Through the** `AnnotationApplicationContext`**, the bean with the name** `User2` **can be accessed.**
- **The output is the** `User2` **object that is printed on the console.**

Slide Note

Quick Check

Which one of the following statements is correct about Dependency Injection?

1. It helps decouple application objects from each other.
2. It helps in determining the dependencies between objects.
3. It stores object states in the database.
4. It stores object states in the file system.

Menu

Quick Check: Solution

Which one of the following statements is correct about Dependency Injection?

1. **It helps decouple application objects from each other.**
2. It helps in determining the dependencies between objects.
3. It stores object states in the database.
4. It stores object states in the file system.

Slide Note

User Entity

The user credentials for logging into an e-commerce application need to be captured. Manage the beans of the application in the Spring container.

- Add appropriate dependencies.
- Use the annotation-based configuration to manage the beans.
- Display the username on the console.
- Click here to see the solution.

DEMO

Menu

00:30 00:00:31 30/ 31