Spring Start Here

Chapter-1: Spring in the Real World

Upcode Software Engineer Team

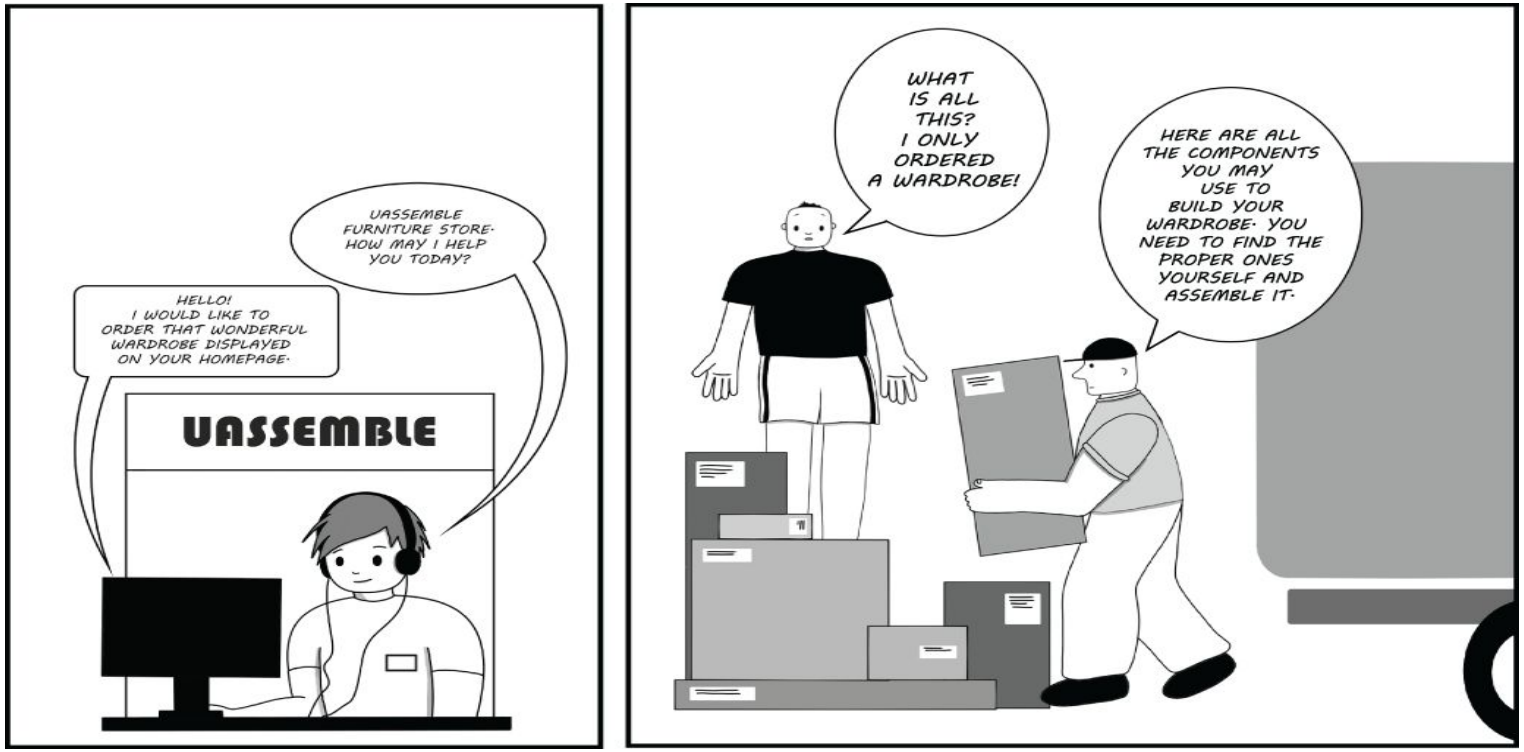
-2024-

CONTENT

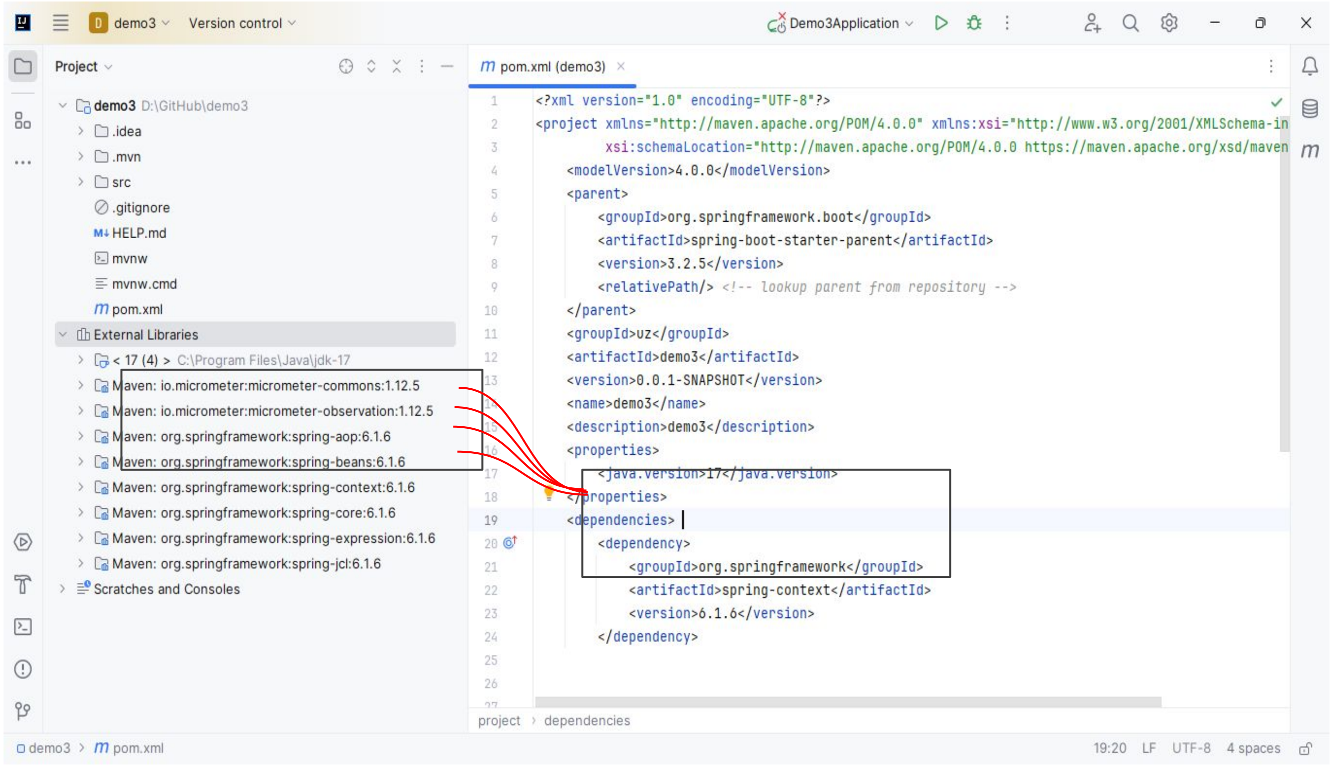
Why should we use frameworks

- **An application framework** is a set of functionalities on top of which we build applications.
- The application framework provides us a broad set of tools and functionalities that you can use to build apps.
- You don’t need to use all the features the framework offers. you’ll choose the right parts of the framework to use.

Why should we use frameworks


UASSEMBLE FURNITURE STORE. HOW MAY I HELP YOU TODAY?
HELLO! I WOULD LIKE TO ORDER THAT WONDERFUL WARDROBE DISPLAYED ON YOUR HOMEPAGE.
UASSEMBLE
WHAT IS ALL THIS? I ONLY ORDERED A WARDROBE!
HERE ARE ALL THE COMPONENTS YOU MAY USE TO BUILD YOUR WARDROBE. YOU NEED TO FIND THE PROPER ONES YOURSELF AND ASSEMBLE IT.

When we add dependency

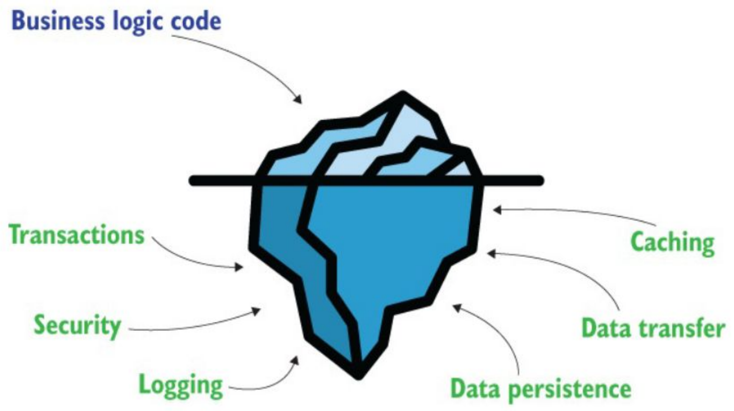


```
demo3 D:\GitHub\demo3
  .idea
  .mvn
  src
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
External Libraries
  < 17 (4) > C:\Program Files\Java\jdk-17
  Maven: io.micrometer:micrometer-commons:1.12.5
  Maven: io.micrometer:micrometer-observation:1.12.5
  Maven: org.springframework:spring-aop:6.1.6
  Maven: org.springframework:spring-beans:6.1.6
  Maven: org.springframework:spring-context:6.1.6
  Maven: org.springframework:spring-core:6.1.6
  Maven: org.springframework:spring-expression:6.1.6
  Maven: org.springframework:spring-jcl:6.1.6
Scratches and Consoles
```

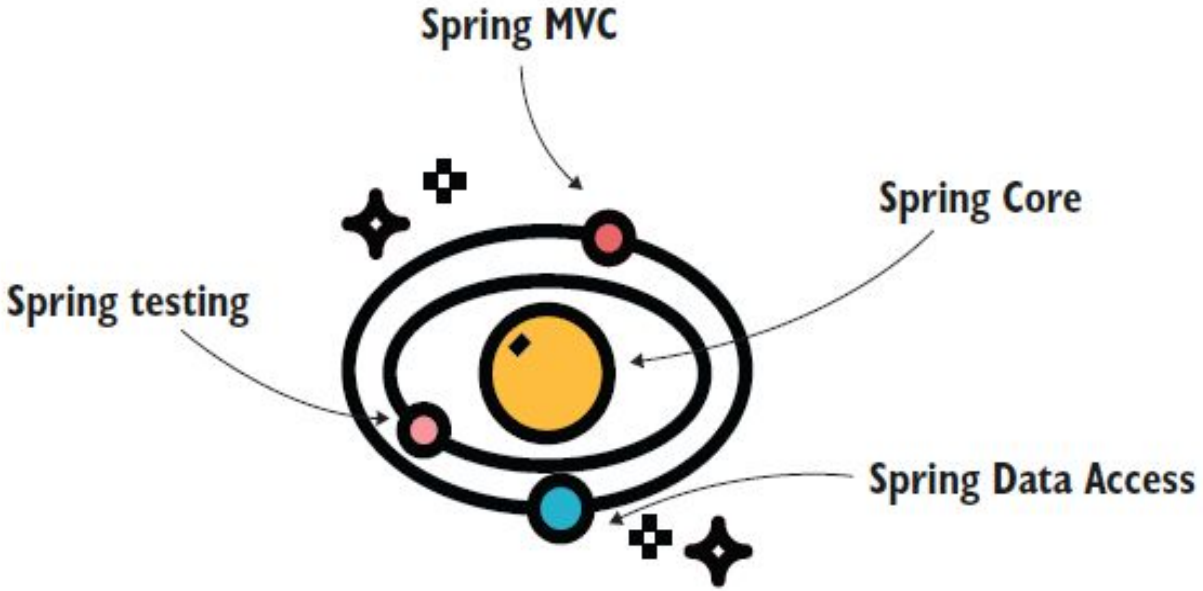
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>uz</groupId>
    <artifactId>demo3</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo3</name>
    <description>demo3</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>6.1.6</version>
        </dependency>
```

Business logic code

- The user’s perspective is like an iceberg. The users mainly observe the results of the business logic code, but this is only a small part of what builds the app’s complete functionality. Like an iceberg has most of it underwater, in an enterprise app .
- we don’t see most of the code because it’s provided by dependencies



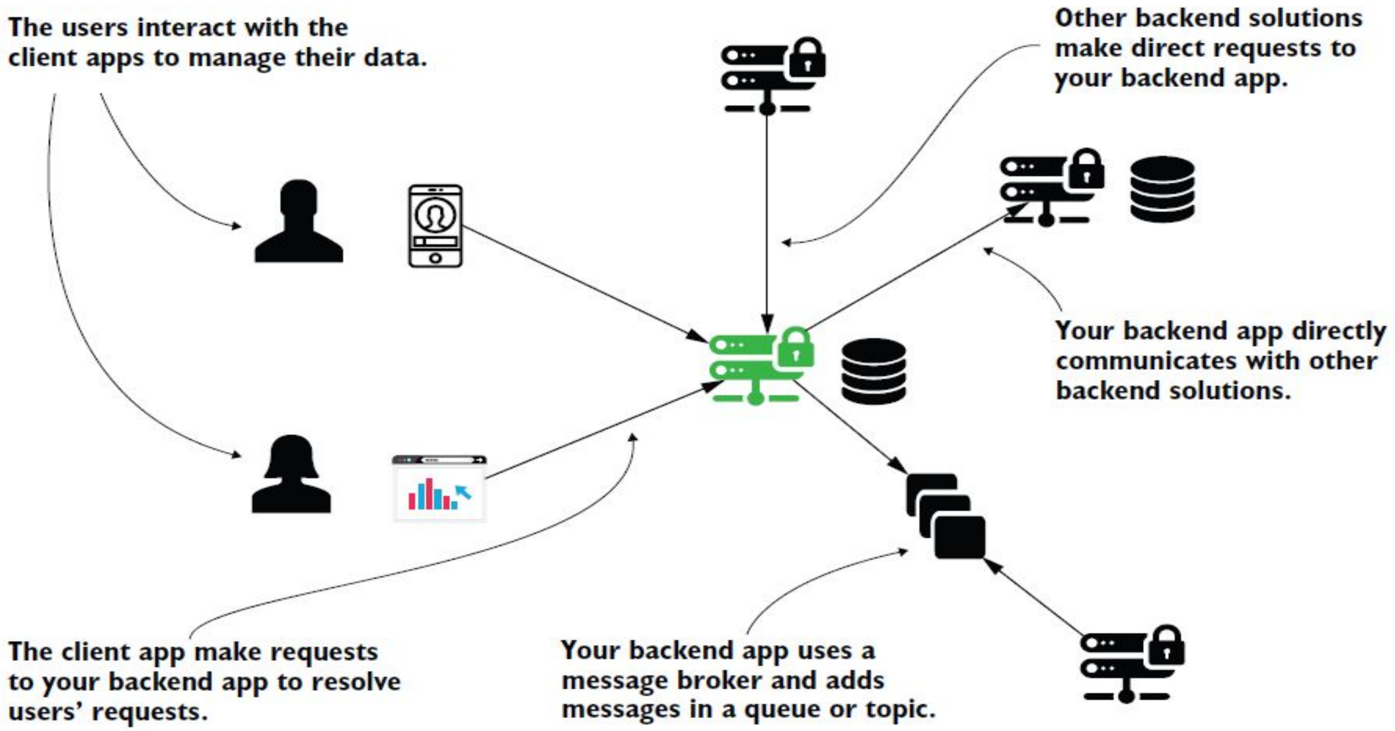
The Spring ecosystem



Spring in real-world scenarios

1. The development of a backend app
2. The development of an automation testing framework
3. The development of a desktop app
4. The development of a mobile app

Spring in real-world scenarios


The users interact with the client apps to manage their data.
Other backend solutions make direct requests to your backend app.
Your backend app directly communicates with other backend solutions.
The client app make requests to your backend app to resolve users' requests.
Your backend app uses a message broker and adds messages in a queue or topic.

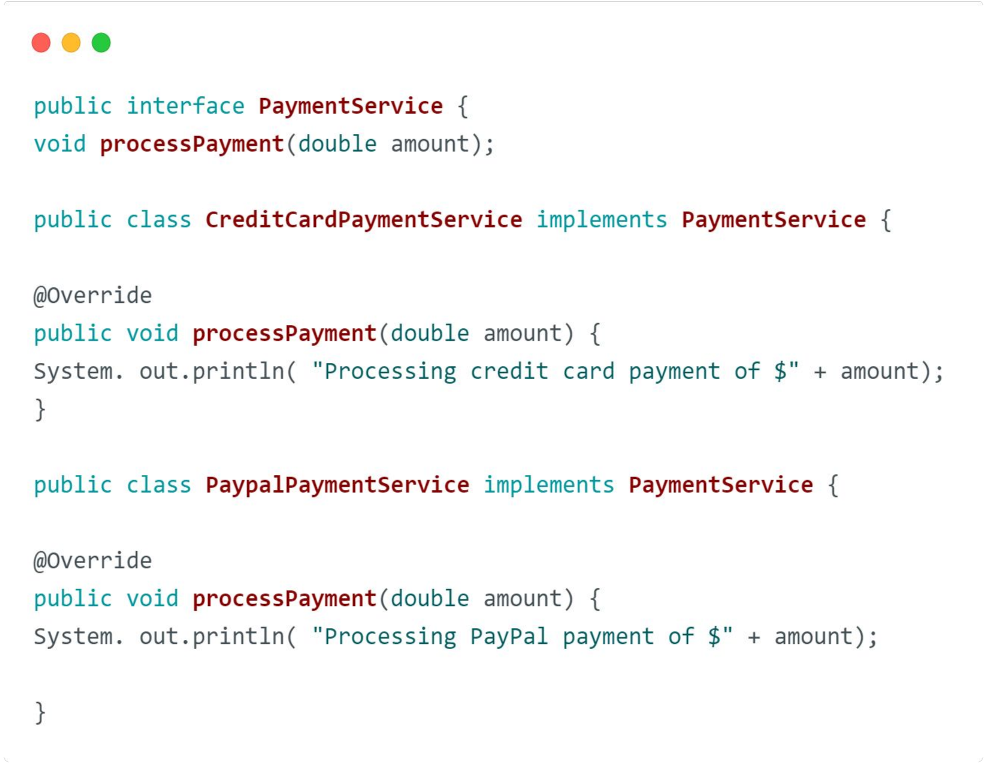
Components of the Java Spring Framework

- Inversion of Control (IoC)
- Dependency Injection (DI)
- Aspect-Oriented Programming (AOP)
- Model-View-Controller (MVC)
- Database Integration
- Security
- Testing Support
- Spring Boot
- Spring Data
- Spring Integration

Inversion of Control (IoC)

- **Spring- Inversion of Control** (IoC). The **Inversion of Control** (IoC) is a process where the objects define their dependencies, that is, the dependencies of other objects with they are working.
- It is done simply by an argument-constructor, argument to a factory method, or by setting properties of the objects when they are being constructed. The dependencies are injected by the container when the bean is created.

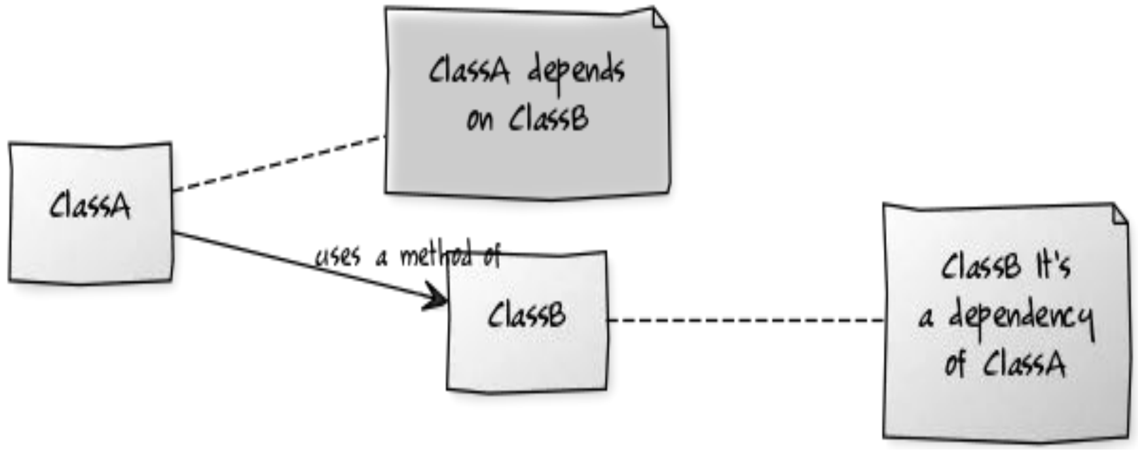
Inversion of Control (IoC)



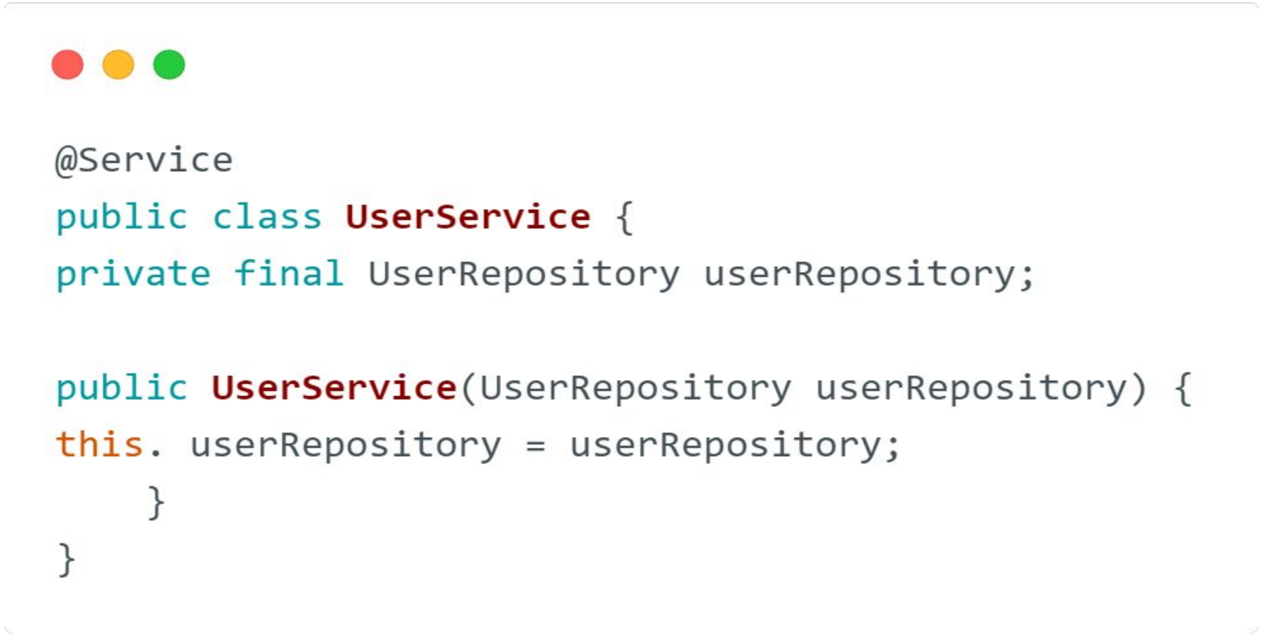
```
public interface PaymentService {
void processPayment(double amount);

public class CreditCardPaymentService implements PaymentService {

@Override
public void processPayment(double amount) {
System. out.println( "Processing credit card payment of $" + amount);
}

public class PaypalPaymentService implements PaymentService {

@Override
public void processPayment(double amount) {
System. out.println( "Processing PayPal payment of $" + amount);

}
```

Dependency Injection (DI)

When a class `ClassA` uses any method of another class `ClassB`, we can say that `ClassB` is a dependency of `ClassA`

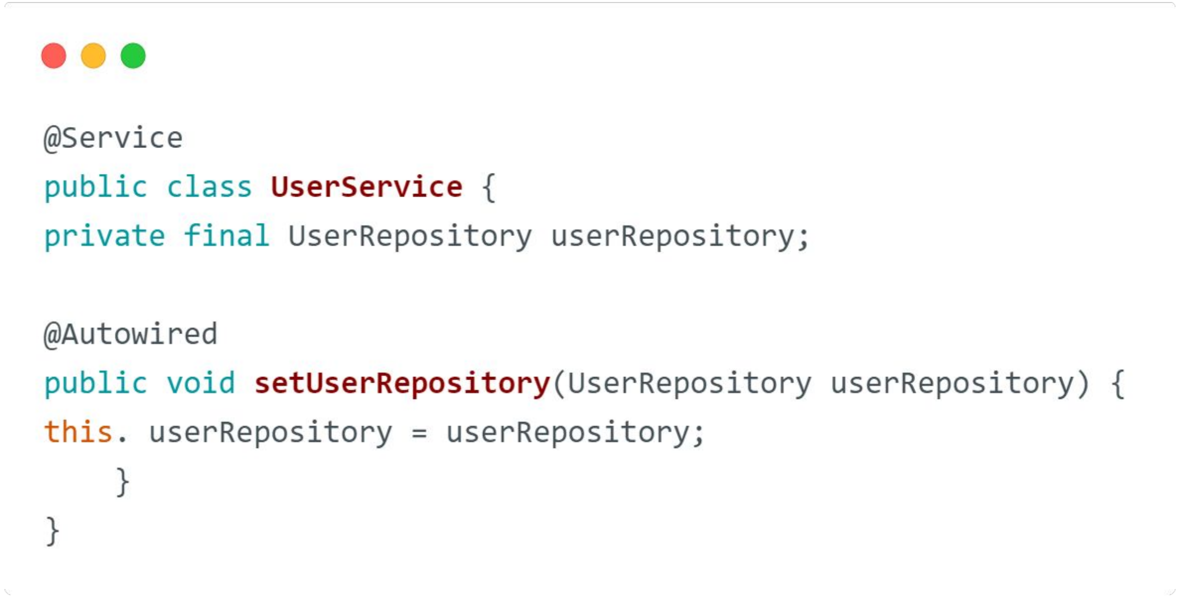


Constructor Injection



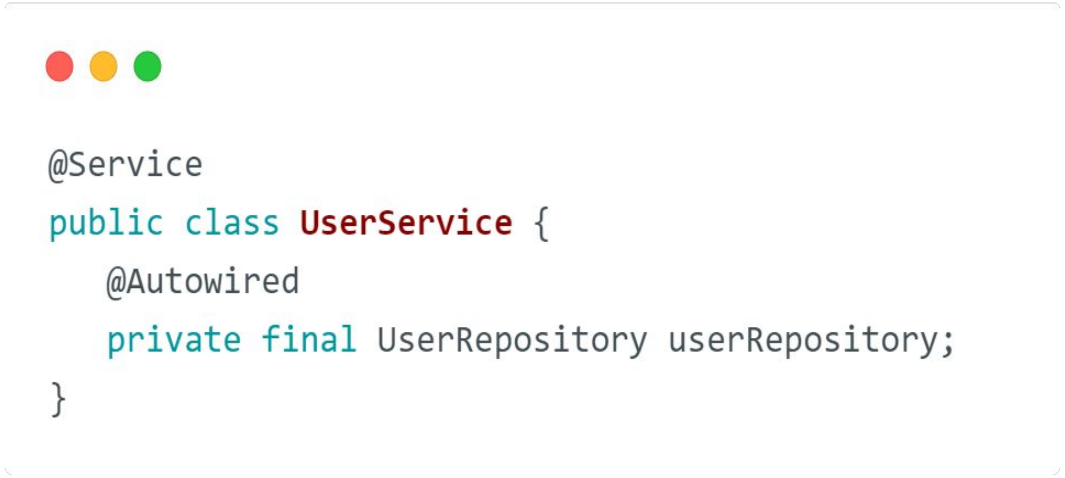
```
@Service
public class UserService {
private final UserRepository userRepository;

public UserService(UserRepository userRepository) {
this. userRepository = userRepository;
    }
}
```

Setter Injection



```
@Service
public class UserService {
private final UserRepository userRepository;

@Autowired
public void setUserRepository(UserRepository userRepository) {
this. userRepository = userRepository;
    }
}
```

Injection with annotation

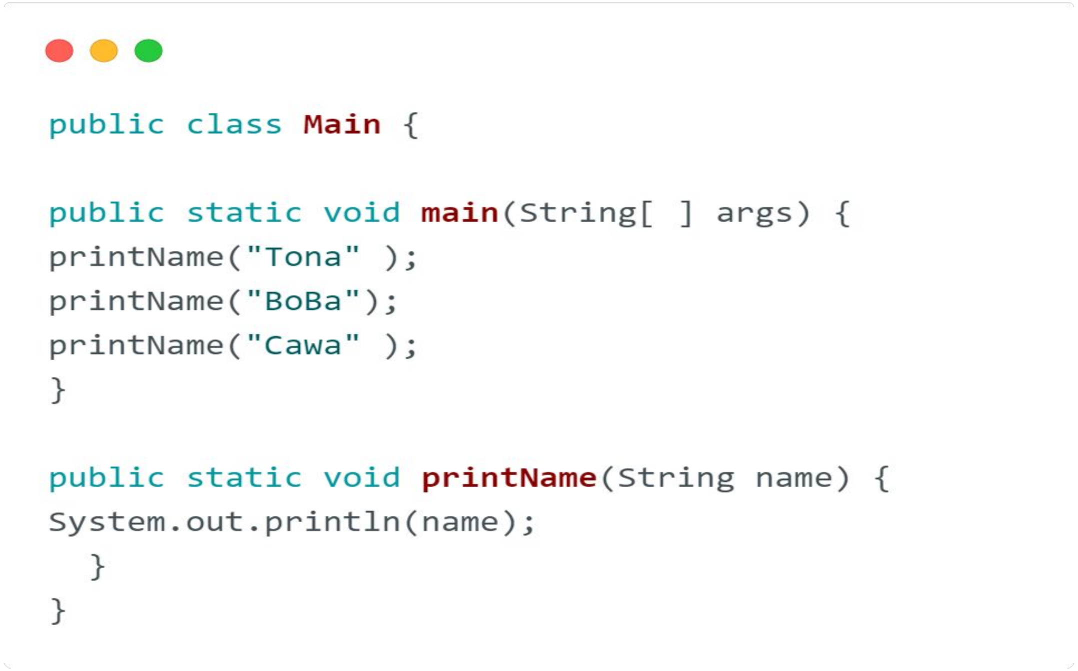


```java
@Service
public class UserService {
    @Autowired
    private final UserRepository userRepository;
}
```

Aspect-Oriented Programming (AOP)

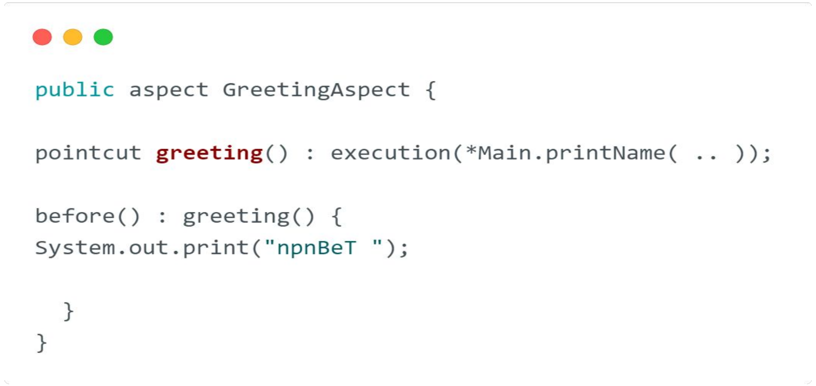
- Aspect-oriented programming (AOP) is a programming paradigm that aims to modularize cross-cutting concerns in software systems.
- In traditional programming, concerns such as logging, error handling, and security are typically scattered across different modules or classes, leading to code tangling and low maintainability.
- AOP provides a solution to this problem by allowing developers to separate these concerns from the core logic of the program.

Example AOP



```
public class Main {

public static void main(String[ ] args) {
printName("Tona" );
printName("BoBa");
printName("Cawa" );
}

public static void printName(String name) {
System.out.println(name);
  }
}
```

Example AOP



```
public aspect GreetingAspect {

pointcut greeting() : execution(*Main.printName( .. ));

before() : greeting() {
System.out.print("npnBeT ");

   }
}
```

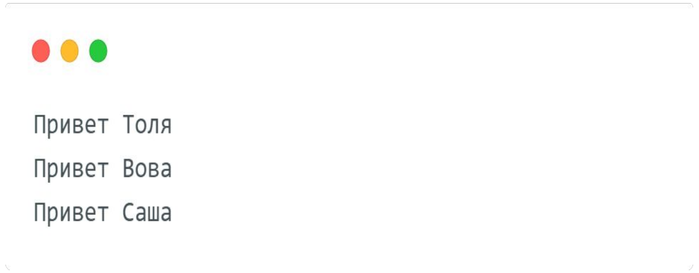
pointcut -> slice or set of connection points

greeting( ) -> the name of this slice

: execution - when executing * - all, call - Main.printName(..) - this method.

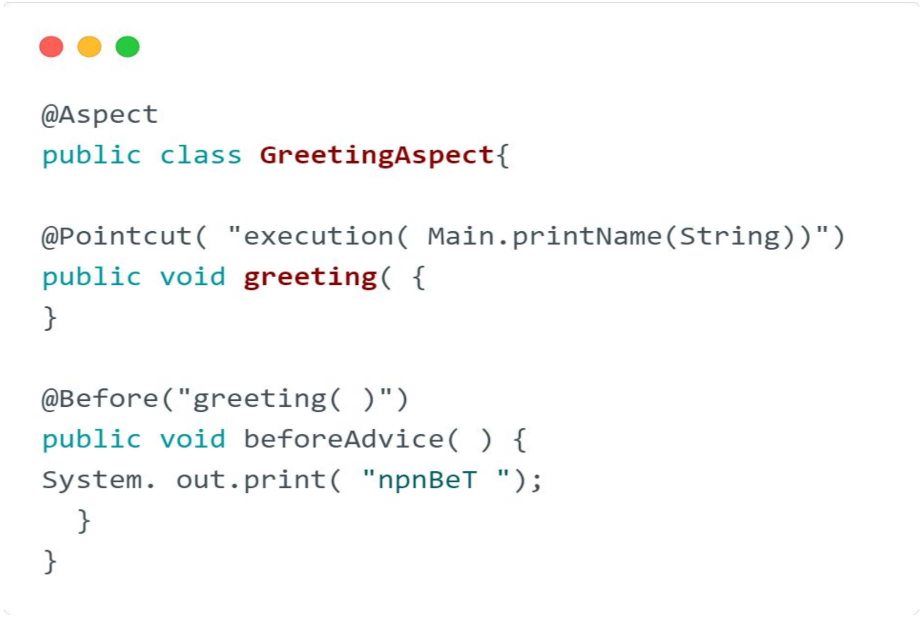
Example AOP

Next comes the specific advice - before() - which is executed before the target method is called, : greeting() - the slice to which this advice reacts, and below we see the body of the method itself, which is written in the Java language we understand.



```
Привет Толя
Привет Вова
Привет Саша
```

Example AOP



```
@Aspect
public class GreetingAspect{

@Pointcut( "execution( Main.printName(String))")
public void greeting( {
}

@Before("greeting( )")
public void beforeAdvice( ) {
System. out.print( "npnBeT ");
  }
}
```

Example AOP

@Aspect denotes that the given class is an aspect

@Pointcut("execution(* Main.printNane(String])") - a cut point that fires on all calls

to

Main. printNane with an incoming argument of type String;

@Before("greeting( )") - advice that is applied before calling the code described at

the **greeting( )**

cut()

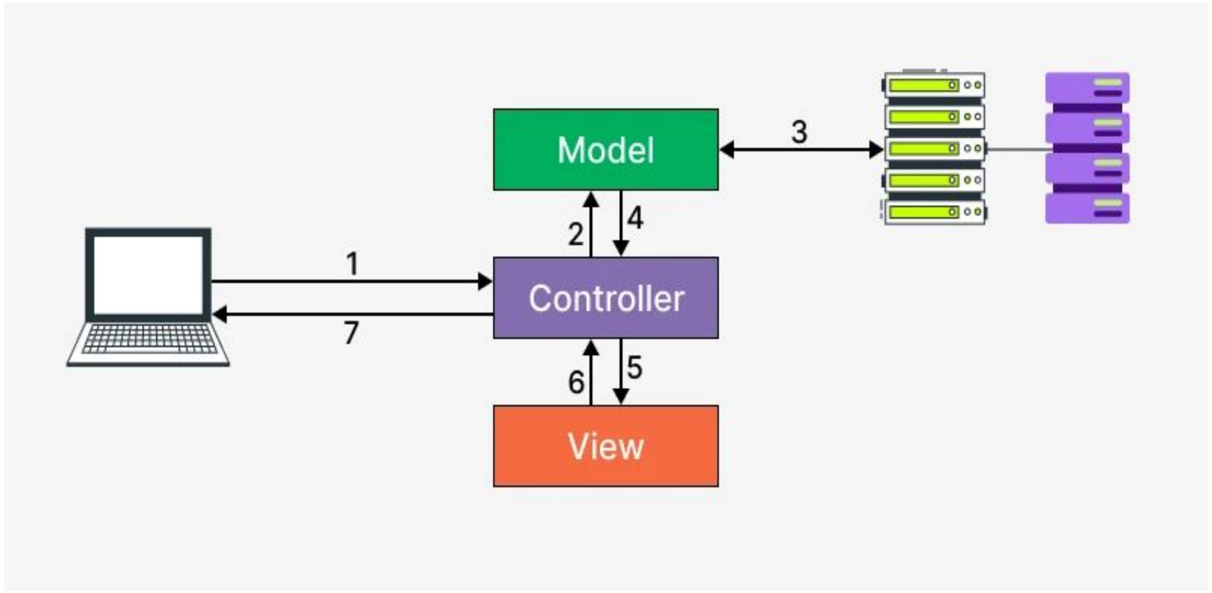
point().

Model-View-Controller (MVC)

Model–view–controller (**MVC**) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements. These elements are : the model, the internal representations of information. the view, the interface that presents information to and accepts it from the user.

Model. Receives data from the controller, performs the necessary operations and passes them to the view.
View (view or presentation). Receives data from the model and displays it to the user.
Controller (controller). Processes user actions, checks the received data and transfers it to the model.

MVC Architecture



When not to use frameworks

- Simple or Small-Scale Projects(Простые или маломасштабные проекты)
- Tight Project Deadlines (Сжатые сроки проекта)
- Specific or Unique Requirements (Особые или уникальные требования)

REFERENCE

1. Spring Start Here
2. GeekforGeeks(AOP)

Thank you!

Presented by

Tokhirjon Sadullaev

(tohirjonsadullayev387@gmail.com)