Introduction to Spring Framework

MSS

institut teknologi del
MARTUHAN-MARROHA-MARBISUK
2001

Overview

- What is Spring framework?
- Spring framework architecture.
- Benefit of using Spring framework.
- Inversion of Control.
- Dependency Injection.
- Aspect-Oriented Programming

What is Spring framework?

Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

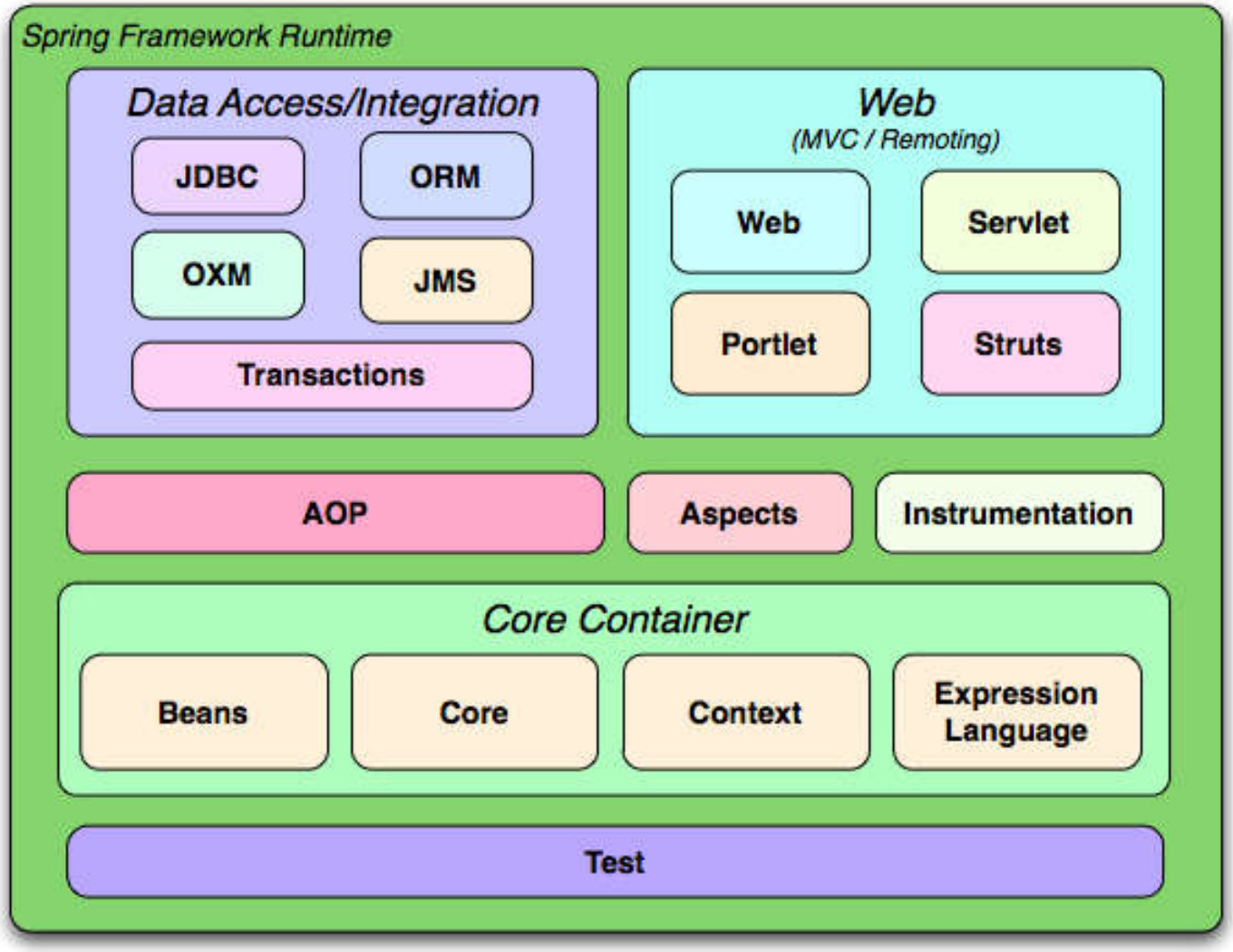
Key concepts:

Inversion of Control (IoC),

Dependency Injection (DI), and

Aspect-Oriented Programming (AOP)

The architecture



Benefit of using Spring framework

The benefit of using only POJOs is that you do not need an EJB (heavy-weight) container. Instead, robust servlet container is utilized.

By using JavaBean-style POJOs, it becomes easier to use dependency injection for injecting data or object.

Inversion of Control (IoC) [1]

The main issue is object dependency.

Normally, the dependent object will instances the independent object inside the dependent class. It is the dependent object's responsibility to decide which class should be invoked.

This tighten the relationship between objects.

Inversion of Control (IoC) [2]

To minimize this tightness, developer could let the outside world decide which class should be invoked by passing interface implemented by the independent class into the dependent.

```
// normally
class MyProgram {
  private String document = "wiro212";

  public void printDocument(){
    FilePrinter fp = new FilePrinter("document.pdf");
    fp.print(this.document);
  }
}
```

Inversion of Control (IoC) [3]

```
interface Printer{
        print(String _document);
}

class FilePrinter implements Printer{
  // ...
}

class MyProgram {
  private String document = "wiro212";
  private Printer p = null;

  public MyProgram(Printer p){
        this.p = p;
  }

  public void priintDocument(){
    p.print(this.document);
  }
}
```

Inversion of Control (IoC) [3]

```
interface Printer{
        print(String document);
}

class FilePr
  // ...
}

class MyProg
  private St
  private Pr

  public MyP
        this.p
  }

  public voi
    p.print(
  }
}
```

```
class MyProgram {
  private String document = "wiro212";
  private Printer p = null;

  public void setPrinter(Printer p){
        this.p = p;
  }

  public Printer getPrinter(){
        return(this.p);
  }

  public void priintDocument(){
    p.print(this.document);
  }
}
```

Dependency Injection (DI) [1]

Since we have loosen the relationship between objects through IoC, now the outside world could decide which object should be wired into the dependent object.

Spring framework provides two convenient ways to do dependency injection: configuration file and bean configuration annotations.

Dependency Injection (DI) [2]

An example using configuration file.

```
<?xml version = "1.0" encoding = "UTF-8"?>

<beans ...>
  <bean id = "myProgram" class = "MyProgram">
    <property name="printer">
      <bean id="filePrinter" class = "FilePrinter"/>
    </property>
  </bean>
</beans>
```

Dependency Injection (DI) [3]

An example using annotation.

```
package com.tutorialspoint;
import org.springframework.context.annotation.*;

@Configuration
public class MyProgramConfig {
  @Bean
  public Printer printer(){
    return new Printer();
  }

  @Bean
  public MyProgram myProgram(){
    return new MyProgram(this.printer());
  }
}
```

Dependency Injection (DI) [3]

An example using annotation.

```
package com.tutorialspoint;
import org.springframework.context.annotation.*;

@Confi
public
  @Bea
  publ
    re
  }

  @Bea
  publ
    re
  }
}
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.*;

public class Main {
  public static void main(String[] args) {
    ApplicationContext ctx =
      new AnnotationConfigApplicationContext(
          MyProgramConfig.class
        );

    MyProgram myProgram =
ctx.getBean(MyProgram.class);
    myProgram.printDocument();
  }
}
```

Aspect-Oriented Programming (AOP)

It helps developer to ‘separate’ the true logic and the supporting *aspect*.

The examples to this:

- Logging mechanism, and
- Authorization checking.

The aspect can be attached before, after, or around a *point-cut*.

Recaps …

Spring framework provides environment to develop either enterprise level application or desktop application.

IoC, DI and AOP are the core concepts available in the Spring framework. These concepts help developers to focus on their business logic.

E O F