CODING SHUTTLE

INTRODUCTION TO THE

# SPRING FRAMEWORK

```java
class OrderService {
    private PaymentService paymentService = new PaymentService(); // tightly coupled
}
```

## 3 Major Problems with this code:

### 1. Tight coupling

If you want to switch to StripePaymentService, you need to edit and recompile OrderService.

```
class OrderService {
    private PaymentService paymentService = new PaymentService(); // tightly coupled
}
```

## 3 Major Problems with this code:

1. Tight coupling
2. Hard to test

   You can’t easily replace PaymentService with a mock for testing.

```java
class OrderService {
    private PaymentService paymentService = new PaymentService(); // tightly coupled
}
```

3 Major Problems with this code:

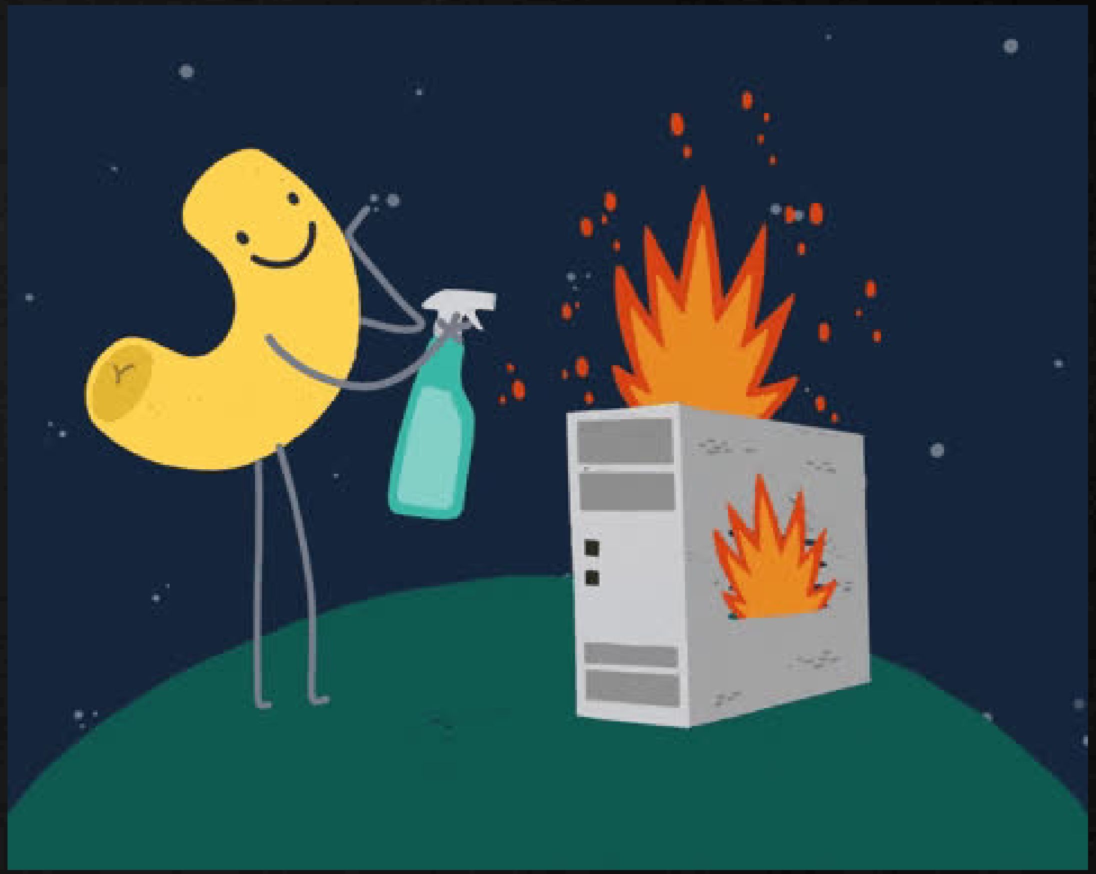
1. Tight coupling
2. Hard to test
3. Scattered Object Creation

   Every class creates its own objects using new, spreading object lifecycle and config logic throughout the app.

## Tight Coupling Problem is just the beginning

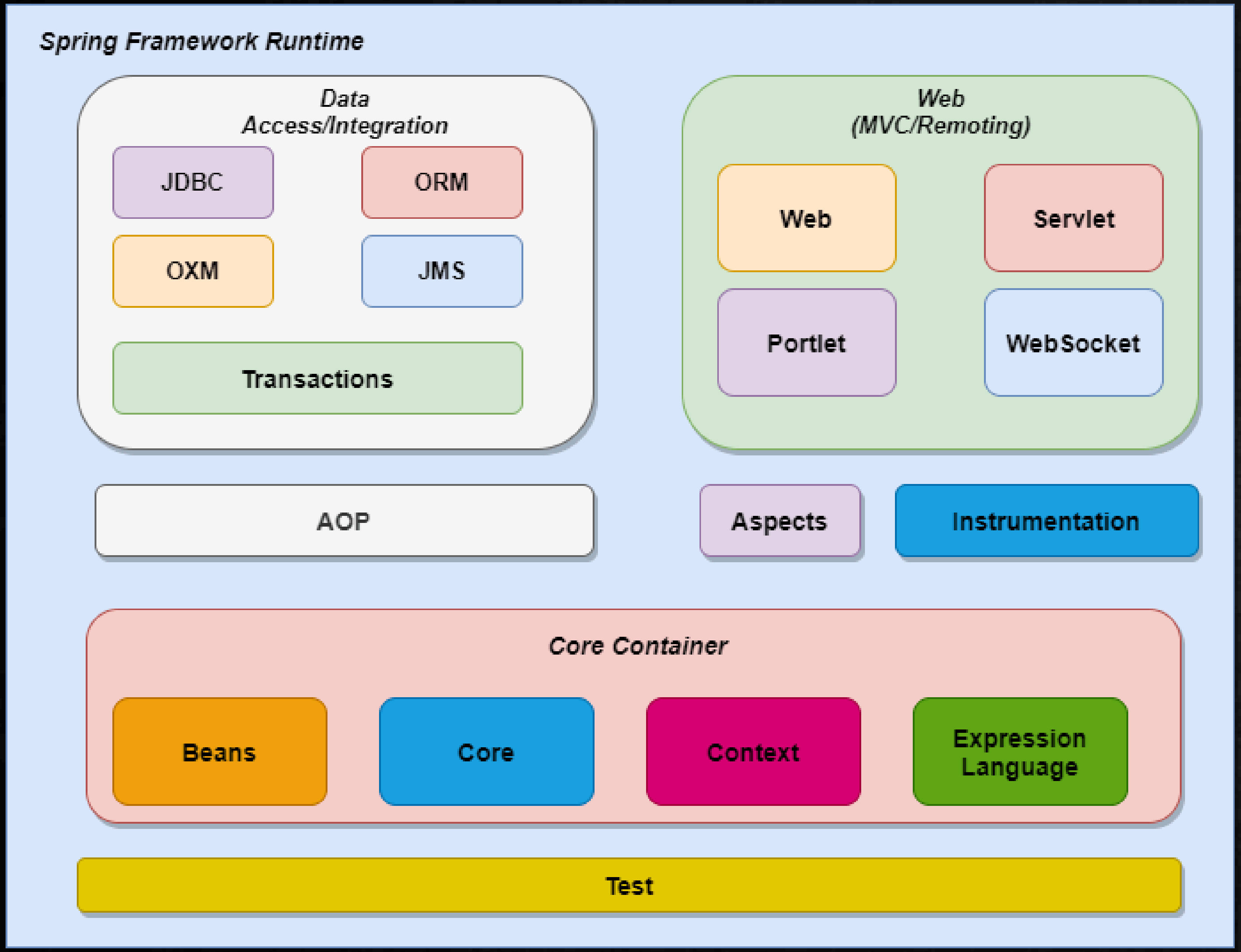
Earlier Java Enterprise Edition code had more issues like:

- Inefficient JDBC and ORM Integration
- Scattered Configuration
- Deployment Complexity
- Verbose and Repetitive Code
- Developers manually wired dependencies, making testing and maintenance hard.



Spring Framework to the Rescue 🚀

# Spring Framework Modules



The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

## Spring Ecosystem

Spring Framework

Spring Boot

Spring Cloud

Spring Data

Spring Authentication

Spring Kafka

Spring Integration

Spring Security

Spring Statemachine

# History of Spring Boot

In 2014, **Spring Boot** was released with:
- Auto-configuration
- Starter dependencies
- Embedded servers (Tomcat, Jetty)
- Production-ready features (health checks, metrics)

Early 2000s, Java EE Apps
- Complex
- Heavyweight
- Difficult to test and maintain

2004-2017, Spring Framework Evolved:
- Added Java-based Configuration
- Added support for Java 8
- Embraced microservices, reactive programming

In 2004, Rod Johnson open-sourced the Spring Framework
- Added Inversion of Control Container
- Dependency Injection (DI)
- Rapidly adopted by enterprise developers

**$420 M**

in 2009, sold it to VM Ware

$$\boxed{\begin{array}{c}\text{Spring Framework}\\ +\\ \text{Auto configuration}\\ +\\ \text{Conventions}\\ +\\ \text{Tools}\end{array}} \longrightarrow \textit{\textbf{Spring Boot}}$$

## Let’s go into Technicals 🥸

| **Spring Framework** | **Spring Boot** |
|---|---|
| The primary feature was Dependency Injection | The primary feature is Auto-configuration |
| We setup a server explicitly. | Spring Boot provides embedded servers such as Tomcat and Jetty etc |
| Lots of Boilerplate and repetitive code in Spring Framework | We compressed a whole bunch of code **into a Jar** and we added into the starter dependency packages. |
| Spring doesn’t provide an in-memory database | Spring Boot provides support for the in-memory database such as H2. The in-memory database relies on **system memory** |
| Spring requires a lot of dependencies to create a web app. Developers define dependencies inside pom.xml | Spring Boot, on the other hand, can get an application working with just one starter dependency which is a package of multiple JARs. |

## Should you learn Spring First?

Minimum Spring Knowledge to Learn First:

- IoC and Dependency Injection: @Component, @Autowired, constructor injection
- Bean Management: @Configuration, @Bean
- Spring Context: What is ApplicationContext?
- Spring MVC Basics: @Controller, @RestController, @RequestMapping

## Now start Spring Boot

Once done with Spring, move to these Spring Boot Topics:

- Spring Boot Auto-Configuration
- Starter Dependencies
- Properties and YAML config
- Spring Boot CLI (optional)
- Spring Boot DevTools and Actuator