Sri Lanka Institute of Information Technology

APPLICATION FRAMEWORKS

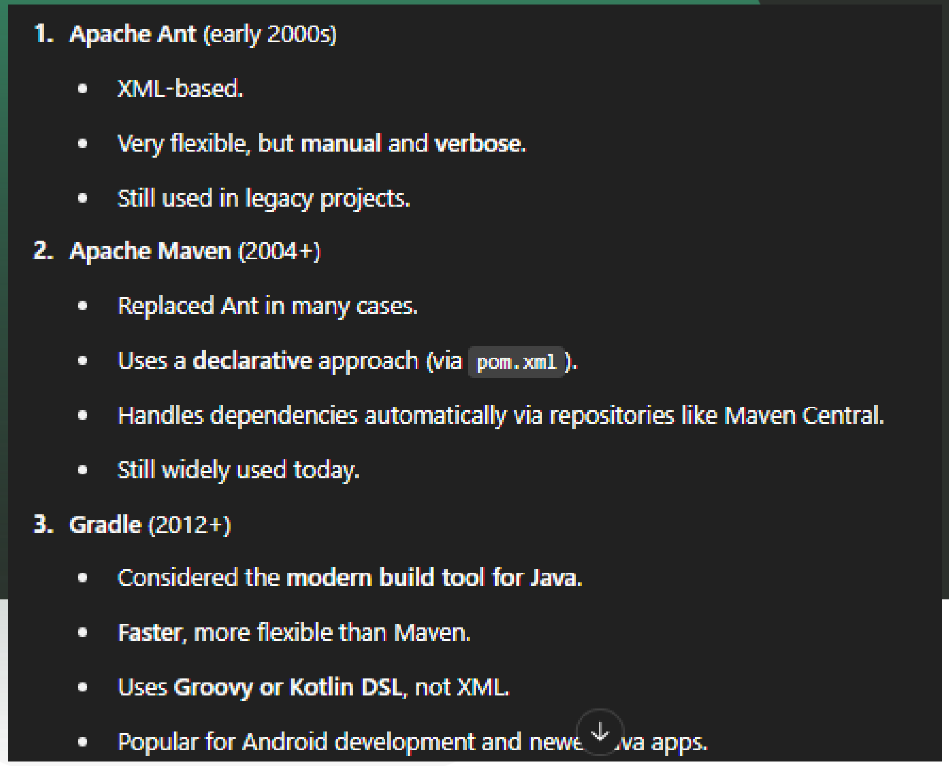
SPRING BOOT

LECTURE 07

Faculty of Computing
Department of Computer Science and Software Engineering
Module Code: SE3040

Agenda

1. Maven
2. Spring Boot



1. **Apache Ant** (early 2000s)
   - XML-based.
   - Very flexible, but **manual** and **verbose**.
   - Still used in legacy projects.
2. **Apache Maven** (2004+)
   - Replaced Ant in many cases.
   - Uses a **declarative** approach (via `pom.xml`).
   - Handles dependencies automatically via repositories like Maven Central.
   - Still widely used today.
3. **Gradle** (2012+)
   - Considered the **modern build tool for Java**.
   - **Faster**, more flexible than Maven.
   - Uses **Groovy or Kotlin DSL**, not XML.
   - Popular for Android development and newe va apps.

MAVEN

Node -> npm dependancy managemnt tool
.NET -> NuGet dependancy managemnt tool
RUST -> Cargo

- Maven is a popular **build tool** for Java-based projects.

XML libraries

- Maven simplifies the process of managing project dependencies and building applications.
- Maven provides a wide range of plugins for performing common build tasks.
- Maven supports the concept of repositories, which are centralized locations for storing and sharing project artifacts.
- Maven makes it easy to share and reuse code across projects by providing a standardized way to manage dependencies and build settings.

BUILT TOOLS

- Built tools are software programs that automate the process of building and packaging software applications.
- Built tools help developers to manage dependencies, compile code, run tests, and package applications into distributable artifacts.
- Maven is a popular built tool for Java-based projects that simplifies the process of managing project dependencies and building applications.
- Maven uses a declarative XML-based configuration file called pom.xml to manage project dependencies and build settings.

BUILT TOOLS...CNT

new

- Other popular built tools for Java-based projects include Gradle, Ant, and Ivy, each with their own strengths and weaknesses.

- Choosing the right built tool for a project depends on factors such as project complexity, team preferences, and community support.

SPRING FRAMEWORK

SPRING FRAMEWORK

- Spring is a widely-adopted open-source framework for building enterprise applications
- Spring Boot features and Spring framework offer a robust, lightweight infrastructure for Java applications
- Comprehensive programming and configuration model for web and non-web application parts
- Spring framework provides many APIs to boost developer productivity, including transaction management and integration, data access and security, server-side technology abstraction, etc.
- One of the most versatile and powerful frameworks in Java

SPRING FRAMEWORK...CNT

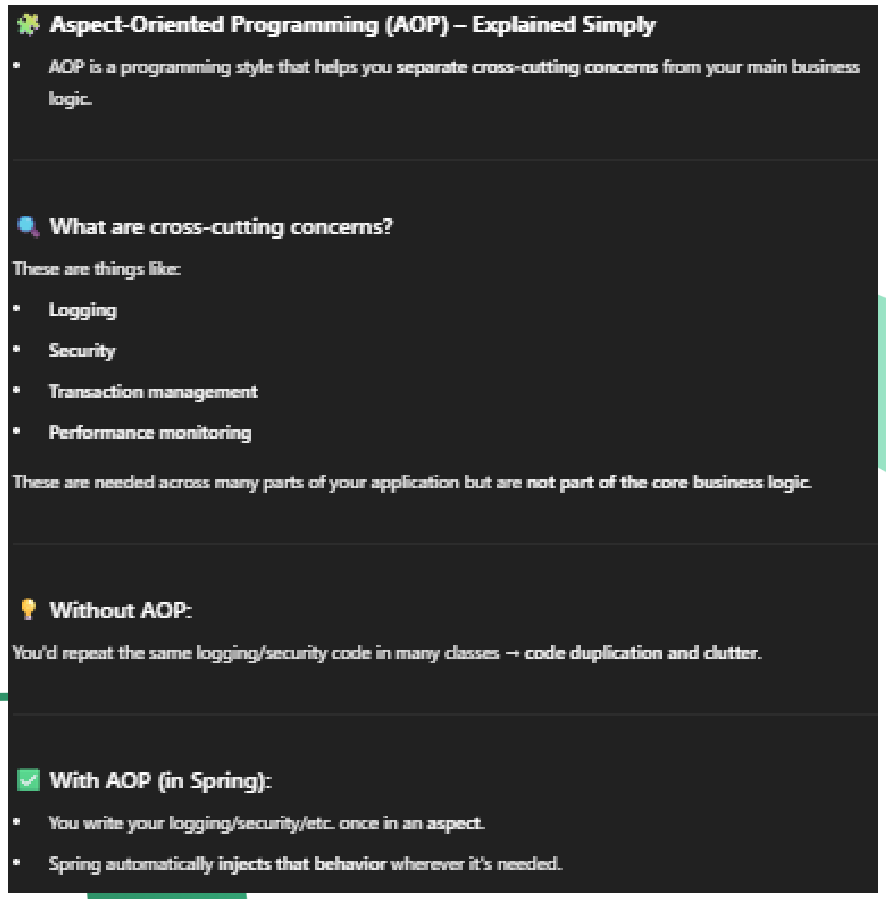
- Focuses on several areas of application development to simplify Java EE development and help developers be more productive It reduces boilerplate code and makes enterprise app development faster and easier

Plain Old Java Objects

- Introduces a paradigm for building applications with POJOs so that business objects are not tied to any specific framework or runtime environment You can build apps using regular Java classes — no need to extend framework-specific classes.

- Most famous for its inversion of controller container for dependency injection.

Instead of creating dependencies manually, Spring injects them for you.



🧩 Aspect-Oriented Programming (AOP) – Explained Simply

- AOP is a programming style that helps you separate cross-cutting concerns from your main business logic.

🔍 What are cross-cutting concerns?

These are things like:

- Logging
- Security
- Transaction management
- Performance monitoring

These are needed across many parts of your application but are **not part of the core business logic**.

💡 Without AOP:

You'd repeat the same logging/security code in many classes → **code duplication and clutter.**

✅ With AOP (in Spring):

- You write your logging/security/etc. once in an **aspect**.
- Spring automatically injects that behavior wherever it's needed.

WHAT ARE THE MAIN FEATURES OF SPRING?

- The most fundamental aspect of Spring and Spring Boot is **Dependency Injection (DI)** or **Inversion of Control (IoC)**

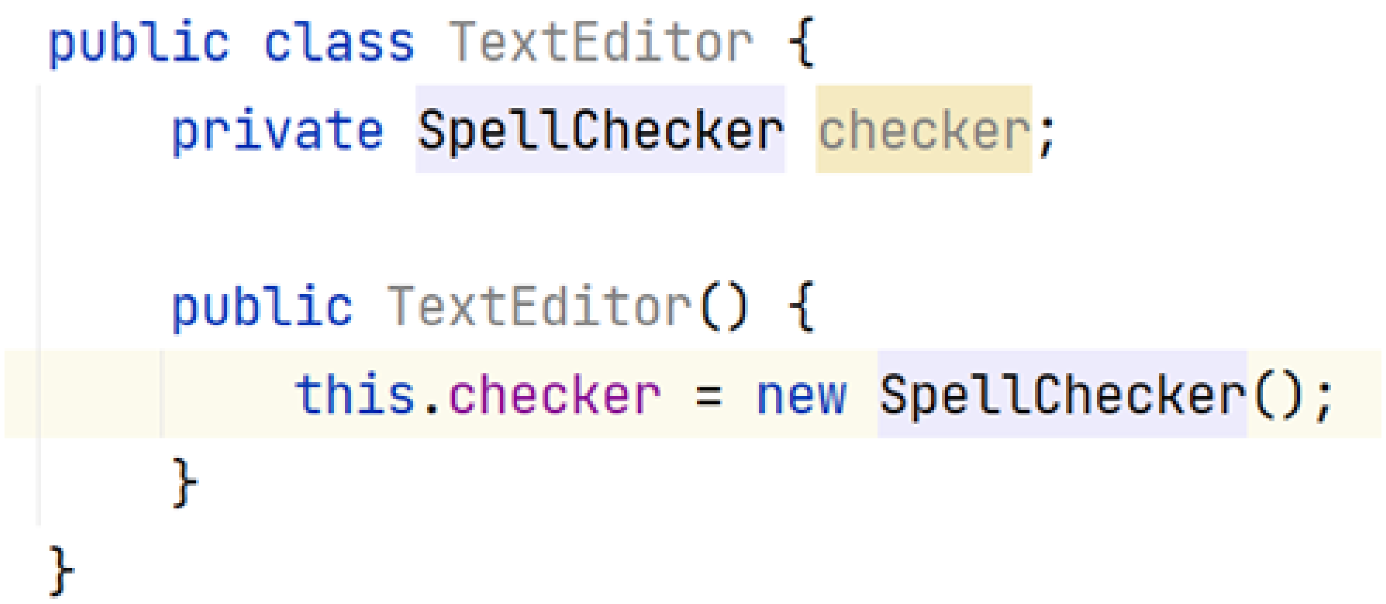
Instead of creating dependencies manually, Spring injects them for

- We can create loosely coupled applications that can be easily tested and maintained using these design patterns. The Spring framework also includes several out-of-the-box modules, namely:
  - Spring MVC Separates logic (Model), UI (View), and flow (Controller)
  - Spring Security Adds authentication and authorization to apps.
  - Spring ORM Integrates with Hibernate, JPA, and other ORM tools. mapping of Java objects to database tables.
  - Spring Test Provides tools for unit testing and integration testing. TDD Support
  - Spring AOP Helps separate cross-cutting concerns like logging, security, and transactions.
  - Spring Web Flow Manages complex user interactions (like multi-page forms or wizards).
  - Spring JDBC. Simplifies direct database access using plain SQL.

- These modules make web applications more functional and reduce development time significantly.

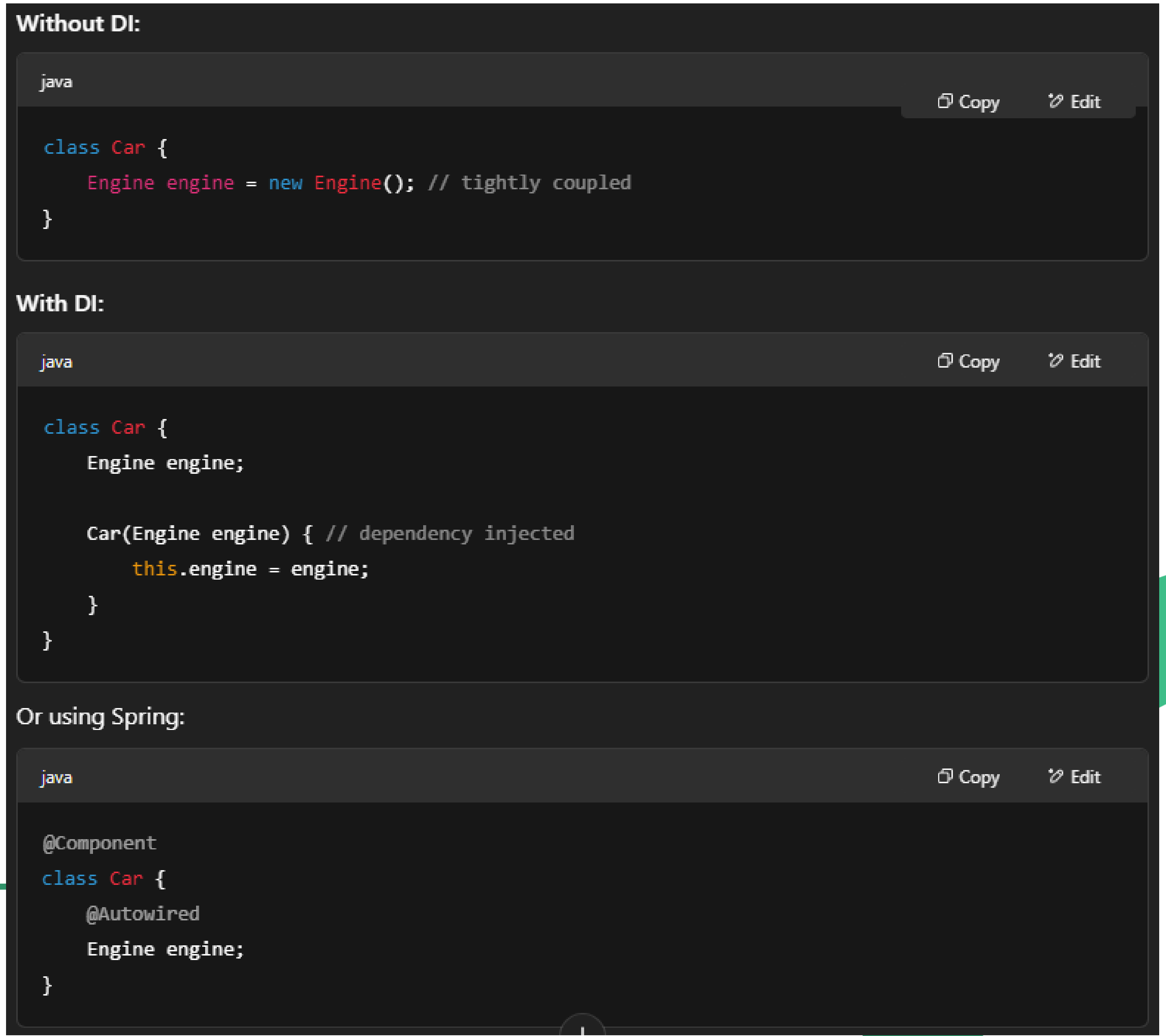
DEPENDENCY INJECTION (DI)

- A type of Inversion of control.
- Passing the dependency at runtime (mostly) into the class without concreate dependencies.
- Resulting context is low coupling between classes.



```java
public class TextEditor {
    private SpellChecker checker;

    public TextEditor() {
        this.checker = new SpellChecker();
    }
}
```



Without DI:

```java
class Car {
    Engine engine = new Engine(); // tightly coupled
}
```

With DI:

```java
class Car {
    Engine engine;

    Car(Engine engine) { // dependency injected
        this.engine = engine;
    }
}
```

Or using Spring:

```java
@Component
class Car {
    @Autowired
    Engine engine;
}
```

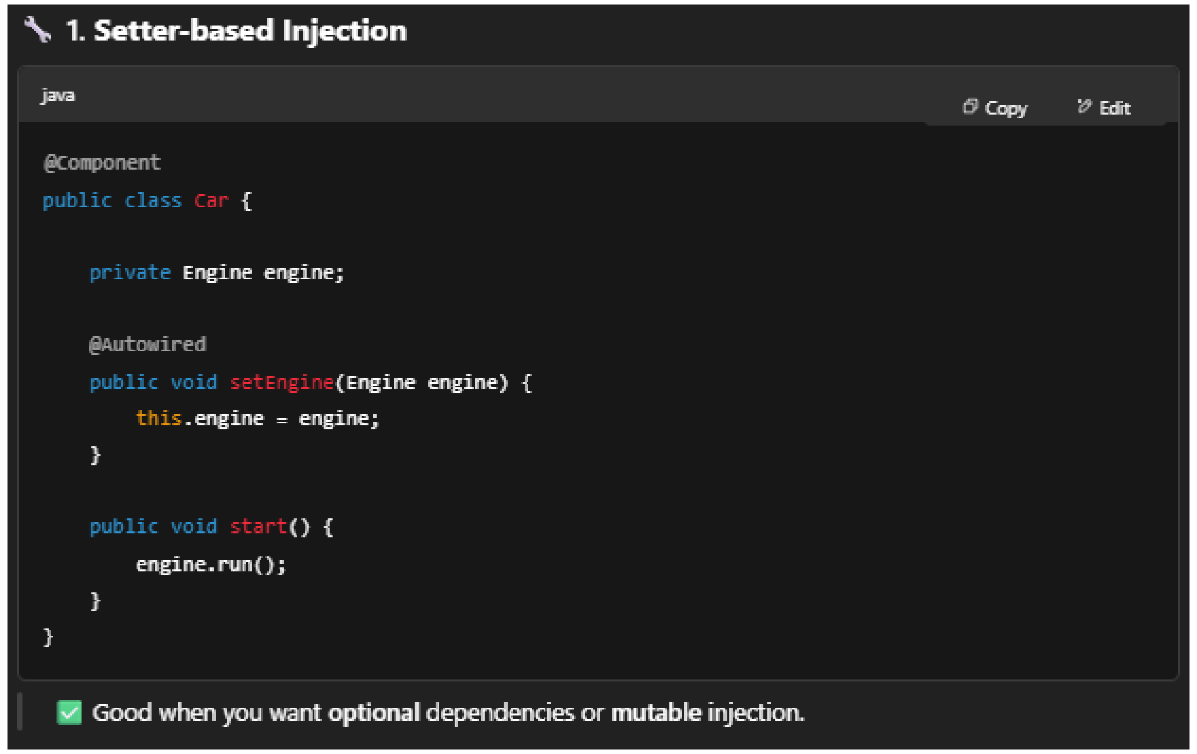
DEPENDENCY INJECTION (DI)... CNT

Maven dependency

```
<dependency>
 <groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
 <version>4.0.0.RELEASE</version>
</dependency>
```

Injections

- Setter based - @Autowired on top of the setter
- Constructor based - @Autowired on top of the constructor
- Field based - @Autowired on top of the field (highly discouraged)



1. Setter-based Injection

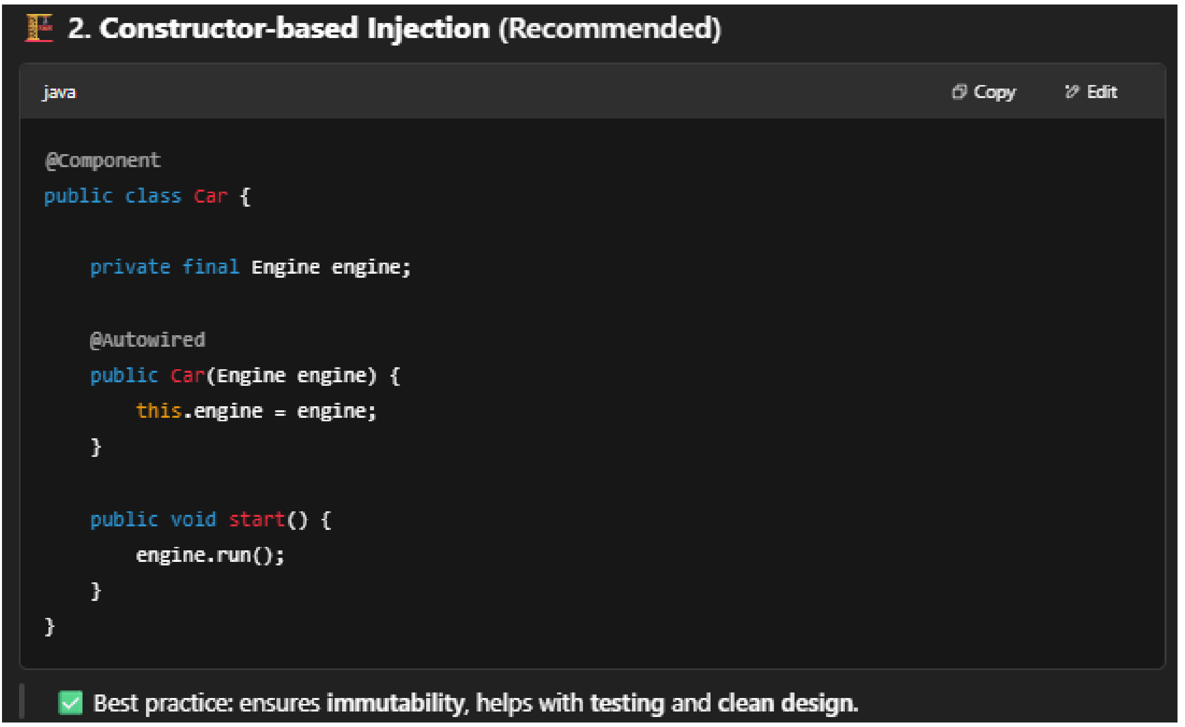
```java
@Component
public class Car {

    private Engine engine;

    @Autowired
    public void setEngine(Engine engine) {
        this.engine = engine;
    }

    public void start() {
        engine.run();
    }
}
```

✅ Good when you want **optional** dependencies or **mutable** injection.



2. Constructor-based Injection (Recommended)

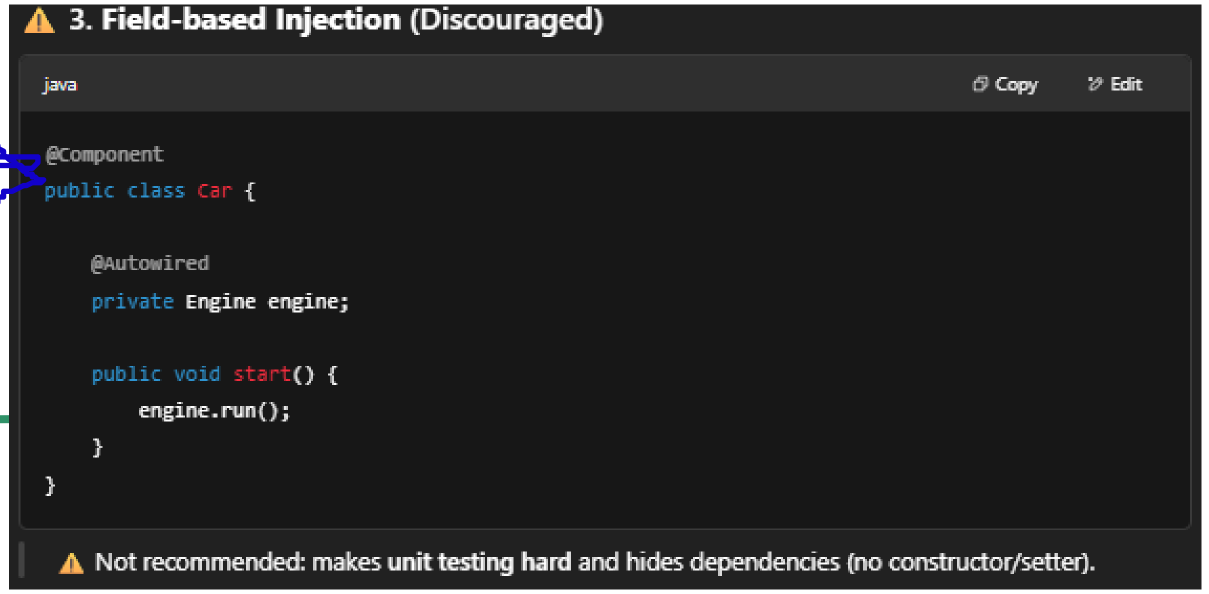
```java
@Component
public class Car {

    private final Engine engine;

    @Autowired
    public Car(Engine engine) {
        this.engine = engine;
    }

    public void start() {
        engine.run();
    }
}
```

✅ Best practice: ensures **immutability**, helps with **testing** and **clean design**.



3. Field-based Injection (Discouraged)

```java
@Component
public class Car {

    @Autowired
    private Engine engine;

    public void start() {
        engine.run();
    }
}
```

⚠ Not recommended: makes **unit testing hard** and hides dependencies (no constructor/setter).

DEPENDENCY INJECTION (DI)... CNT

Some more annotations

@Component

- Making class Spring container aware as a Component.

@Service

- Making class Spring container aware as a Service.

@Repository

- Making class Spring container aware as a DAO.

@RestController

- Making class Spring container aware as a REST controller.

@Configurations

- Spring aware configuration class.

@Autowired

Injects a bean automatically

SPRING USE CASES:

Spring framework can be used for several tasks, including

- Developing serverless applications
- Building scalable microservices
- Securing the server-side of your application
- Asynchronous application development
- Automating tasks by creating batches
- An event-driven architecture

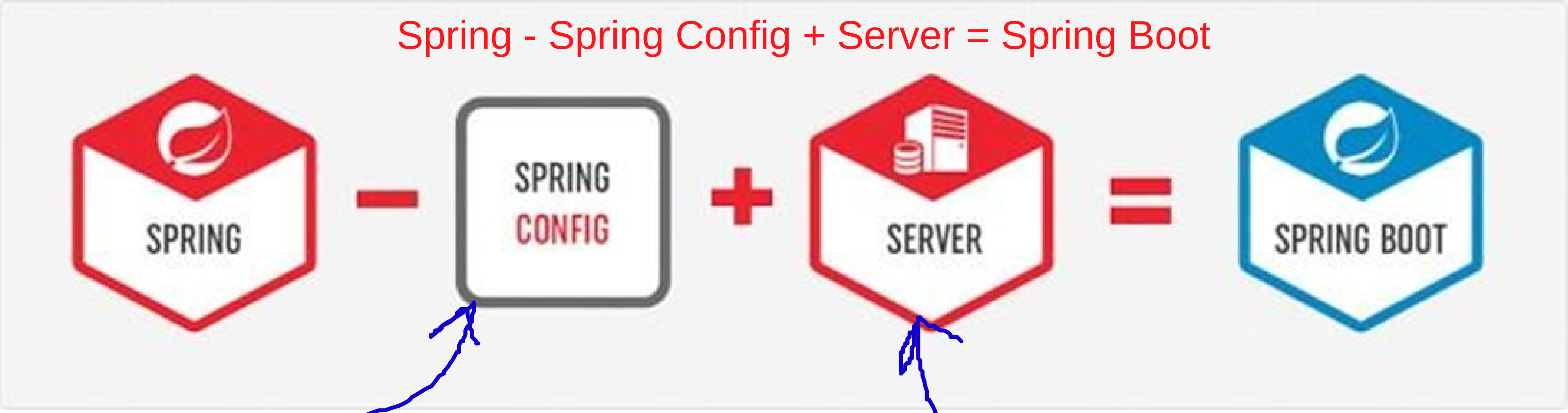
Nice framework but the amount of configuration it has, made it cumbersome to use for rapid application development

SPRING BOOT

- Fully featured robust framework mainly targeted for Microservices application development.

complex

- A solution for cumbersome configuration Spring Framework has.
- Support for microservices.
- Easy integration with multiple other libraries and frameworks (Cloud, Circuit breakers)
- Embedded server for development and deployments.


Spring - Spring Config + Server = Spring Boot

Traditional Spring requires a lot of manual configuration (XML or Java-based) for setting up beans, data sources, etc.

Spring Boot includes an embedded server (like Tomcat or Jetty), so there's no need to deploy WAR files to an external server.

MAIN FEATURES OF SPRING BOOT?

- Embedded server eliminates the need for complex application development
- Starter dependencies that facilitate building and configuring apps
- Automated Spring configuration
- Metrics, health check, and other reports
- Support for microservices.
- Everything in Spring Boot is pre-configured. We simply need to use the proper configuration to use a specific functionality. If we want to create a REST API, we can use Spring Boot.

  Spring Boot comes with smart defaults and auto-configuration, so developers don’t have to set up everything manually.

MICROSERVICES WITH SPRING BOOT

- Spring Boot is a popular framework that simplifies the development and deployment of microservices
- It provides a suite of tools and features that address the challenges of microservices, including:
  - Embedded web server
  - Auto-configuration
  - Health checks
  - Distributed tracing
  - Service discovery and registration
  - Load balancing
  - Configuration management

THAT'S ALL FOLKS !

ANY QUESTIONS ?