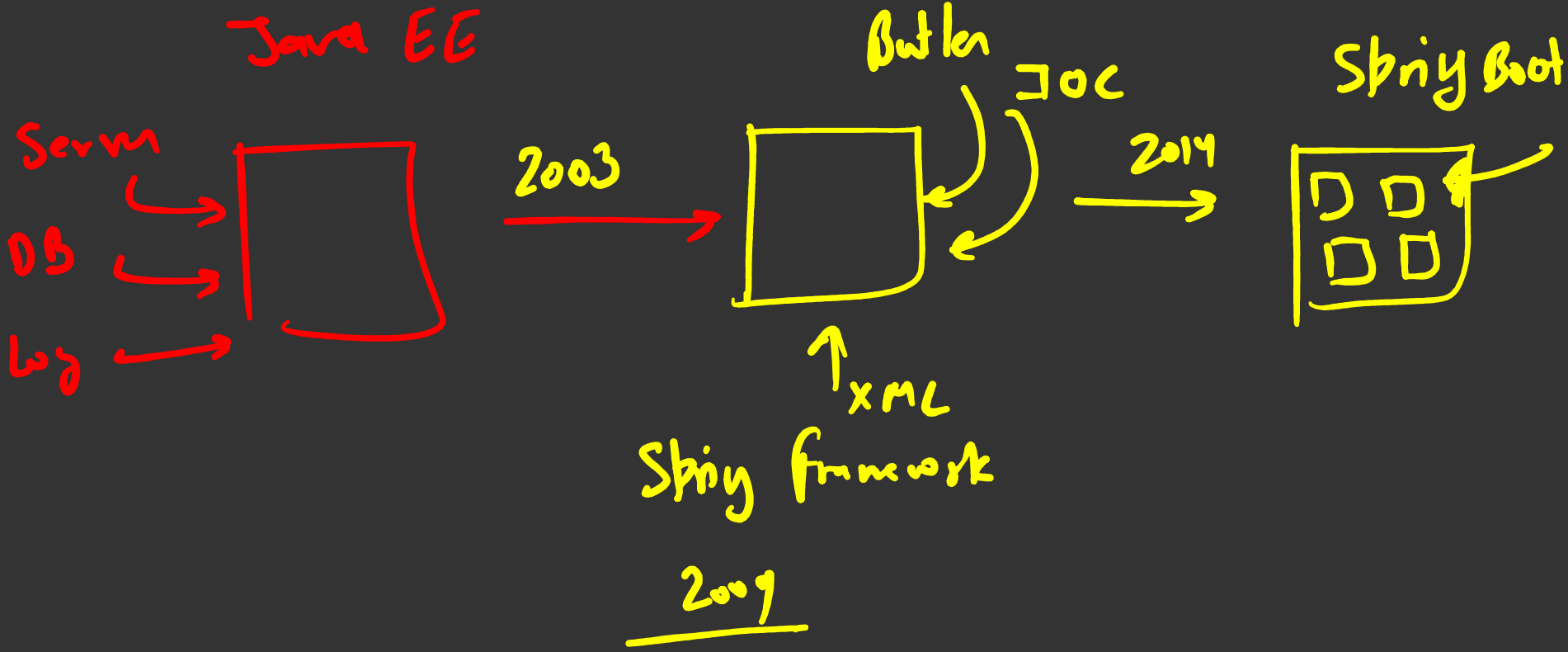
CODING SHUTTLE

1.1

Introduction to Spring Framework

Spring Framework



CODING SHUTTLE

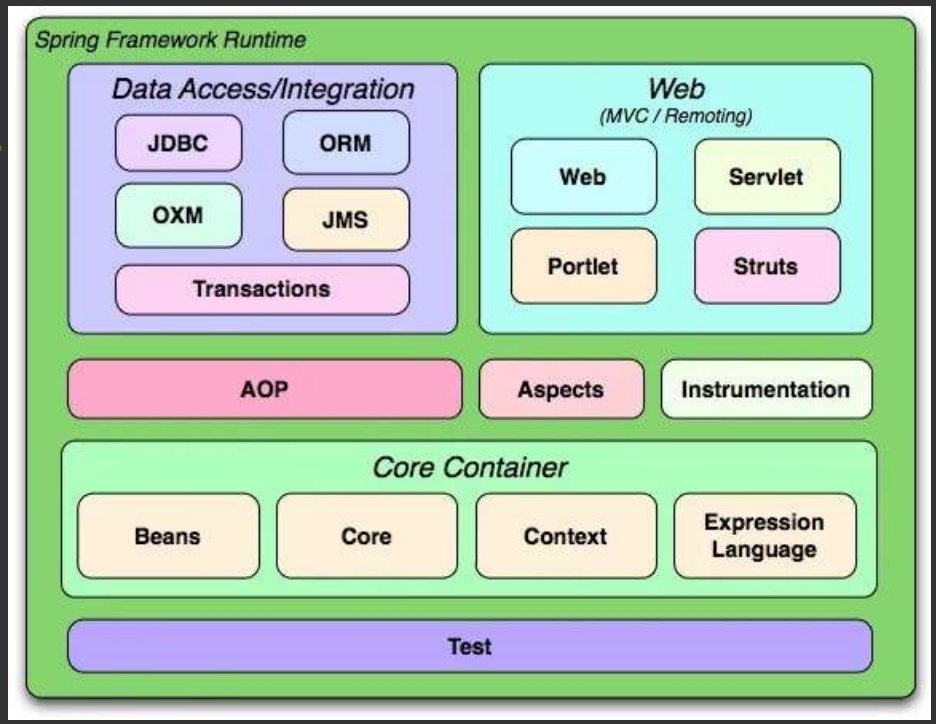
Spring Framework

- Spring is a Dependency Injection framework to make java applications loosely coupled.
- Spring framework makes the development process easy for JavaEE applications.
- Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs.
- Spring was developed by Rod Johnson in 2003.

Spring Framework

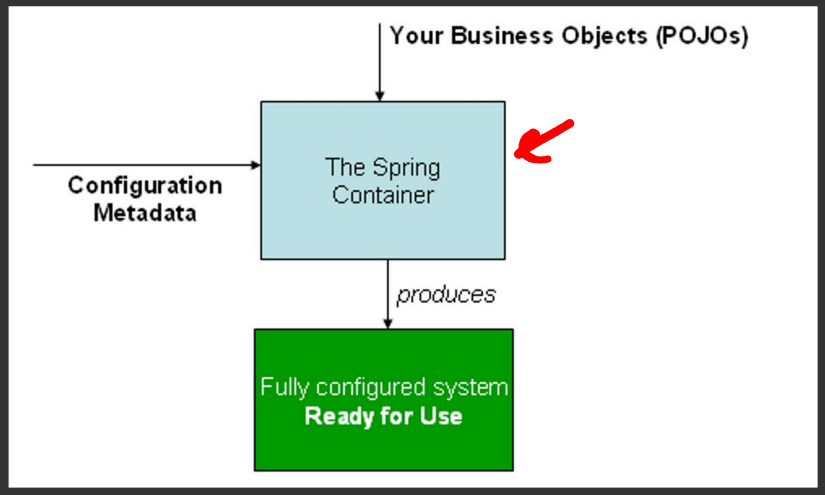
Important Components:

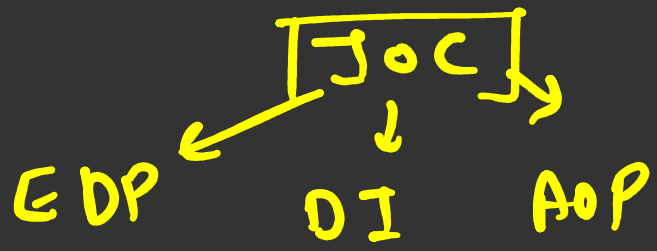
- Core Container
- AOP
- JDBC
- Web
- Testing



IoC Container

In the Spring Framework, the IoC container is responsible for managing the components of an application and injecting dependencies into them. The container creates the objects (beans), wires them together, configures them, and manages their complete lifecycle.





CODING SHUTTLE

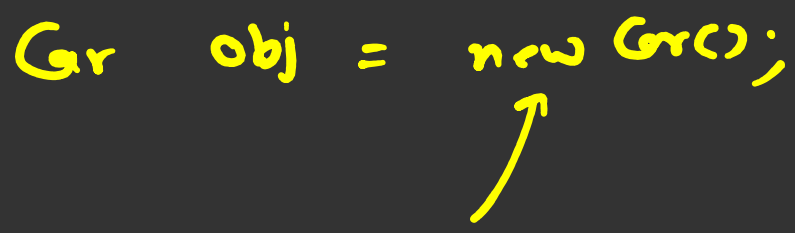
1.3

Beans

Beans

A "bean" is a managed object that is instantiated, assembled, and managed by the Spring IoC container.

Beans form the backbone of a Spring application and are the core building blocks that are wired together to create the application.



Spring Annotations

- Traditionally, Spring allows a developer to manage bean dependencies by using XML-based configuration.
- There is an alternative way to define beans and their dependencies. This method is a Java-based configuration.
- Unlike the XML approach, Java-based configuration allows you to manage bean components programmatically. That's why Spring annotations were introduced.

Defining Beans

1. Using Stereotype Annotations

Annotate your class with one of the stereotype annotations (@Component, @Service, @Repository, @Controller). These annotations inform Spring that the class should be managed as a bean.

2. Explicit Bean Declaration in Configuration Class

Create a configuration class and annotate it with @Configuration. This class will contain methods to define and configure beans.

CODING SHUTTLE

Beans Lifecycle

Bean Created	Dependency Injected	Bean Initialized	Bean is Used	Bean Destroyed
The bean instance is created by invoking a static factory method or an instance factory method (for annotation-based configuration).	Spring sets the bean's properties and dependencies, either through setter injection, constructor injection, or field injection.	If the bean implements the InitializingBean interface or defines a custom initialization method annotated with @PostConstruct, Spring invokes the initialization method after the bean has been configured.	The bean is now fully initialized and ready to be used by the application.	Spring invokes the destruction method when the bean is no longer needed or when the application context is being shut down.

Bean Lifecycle Hooks

The **@PostConstruct** annotation is used to mark a method that should be invoked immediately after a bean has been constructed and all of its dependencies have been injected.

The **@PreDestroy** annotation is used to mark a method that should be invoked just before a bean is destroyed by the container. This method can perform any necessary cleanup or resource releasing tasks.

CODING SHUTTLE

Scope of Beans

Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance for each Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request. That is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
websocket	Scopes a single bean definition to the lifecycle of a WebSocket. Only valid in the context of a web-aware Spring ApplicationContext.

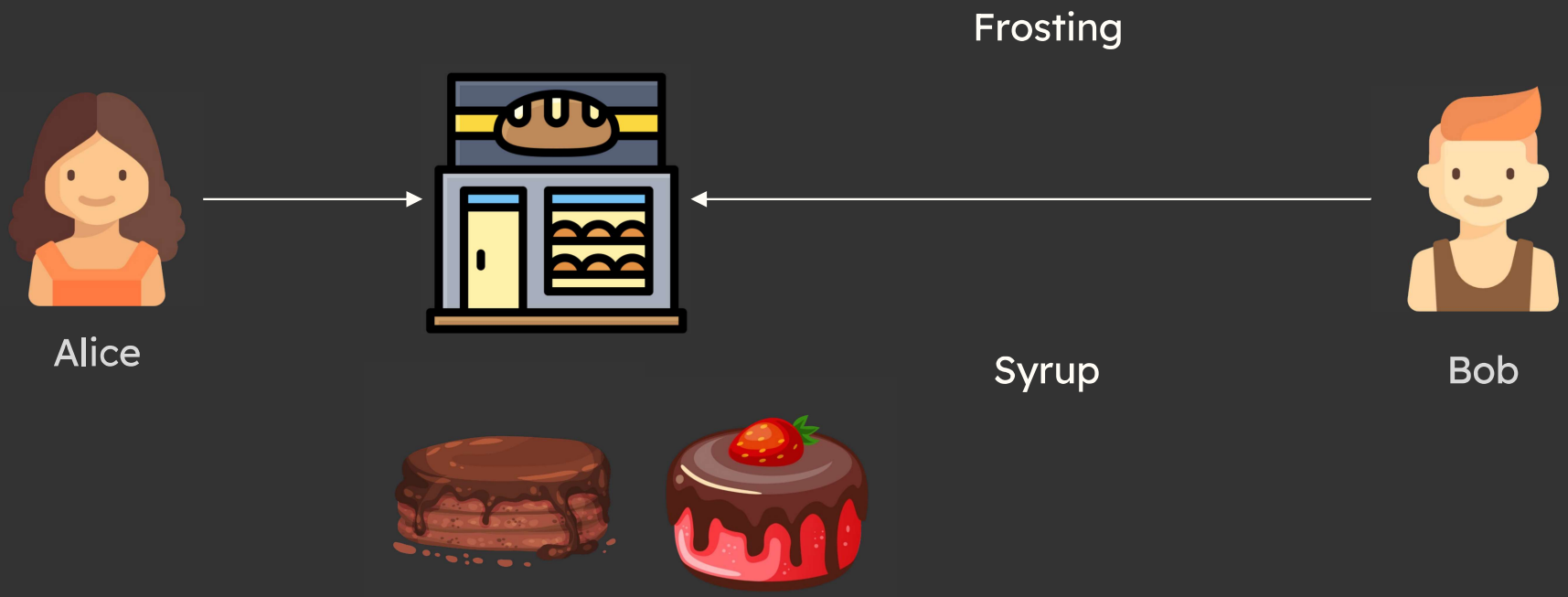
CODING SHUTTLE

1.4

Dependency Injection

CODING SHUTTLE

Dependency Injection



CODING SHUTTLE

Dependency Injection

Dependency Injection (DI) in the context of the Spring Framework is a design pattern and technique used to achieve loose coupling between components in a software application. In a DI scenario, instead of a component creating its dependencies directly, the dependencies are injected into the component from an external source, typically managed by a framework like Spring.

Benefits of Dependency Injection

Loose Coupling: Components are decoupled from their dependencies, making them easier to maintain and test.

Flexible Configuration: Dependencies can be configured externally, allowing for easier customization and swapping of components.

Improved Testability: Components can be easily mocked or replaced during testing, allowing for more thorough and isolated unit tests.

CODING SHUTTLE

How to Inject Dependencies

Constructor Injection

Dependencies are provided through a class constructor.

Field Injection

Dependencies are provided directly into the fields of a class using @Autowired

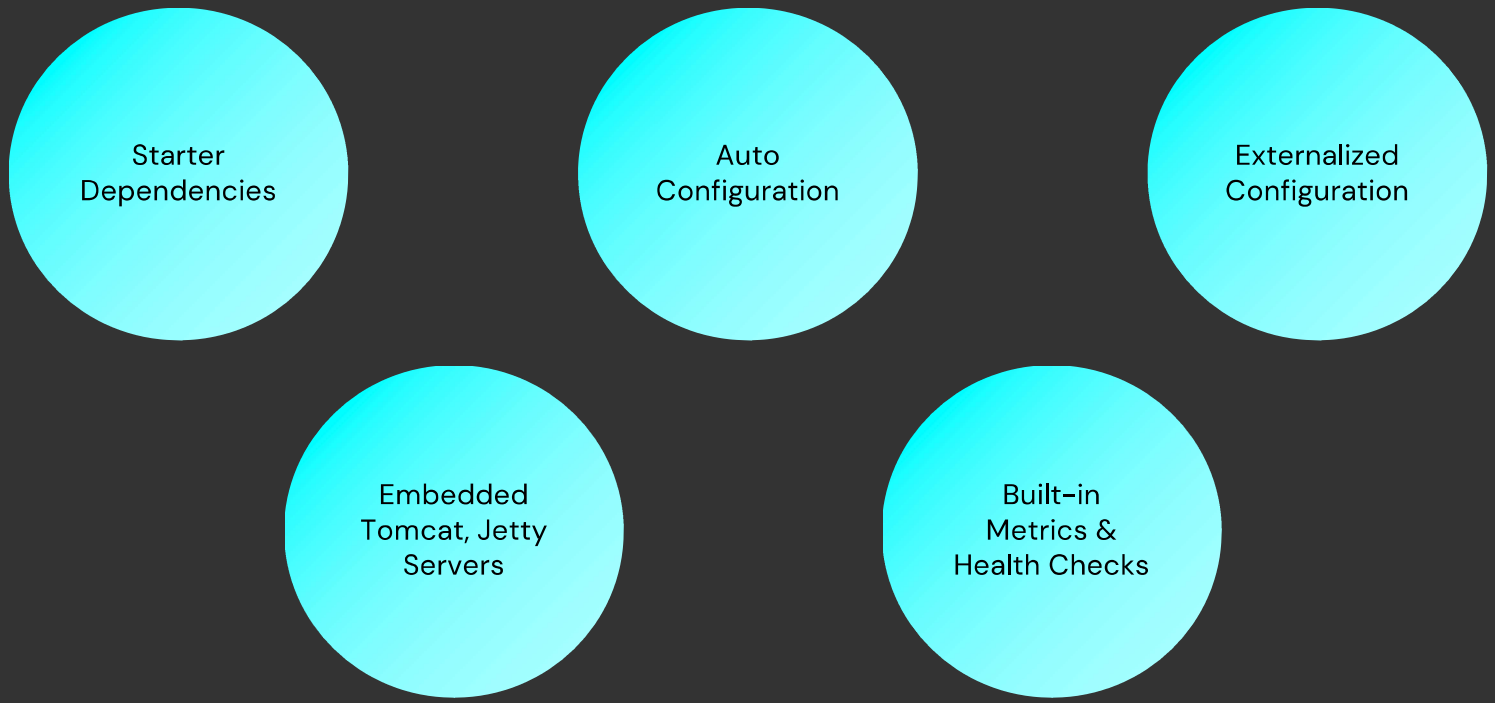
CODING SHUTTLE

1.5

Spring Boot vs Spring Framework

CODING SHUTTLE

Spring Boot vs Spring Framework



CODING SHUTTLE

1.6

Auto-Configuration and Spring Boot Internal flow

pom.xml

- Maven is a popular build automation tool used in many Java projects. In a Spring Boot project, dependencies are specified in the pom.xml file. Maven then resolves these dependencies and includes them in the classpath.

- Starters like **spring-boot-starter-parent** include a ton of third-party libraries into your project - by default. Its AutoConfigurations use these dependencies to setup and preconfigure these libraries automatically.

- The spring-boot-dependencies pom.xml contains every 3rd party library (and version) that Spring Boot knows. These libraries are predefined in a dependenciesManagement section, so you do not need to specify the version numbers in your own project, anymore.

CODING SHUTTLE

What is Auto Configuration

Autoconfiguration refers to the mechanism that automatically configures Spring applications based on the dependencies present on the classpath and other application-specific settings.

This feature simplifies the setup and development process, allowing developers to focus more on writing business logic rather than configuring the framework.

CODING SHUTTLE

How Autoconfiguration Works

Classpath Scanning	Configuration Classes	Conditional Beans
Spring Boot scans the classpath for the presence of certain libraries and classes. Based on what it finds, it applies corresponding configurations	Spring Boot contains numerous autoconfiguration classes, each responsible for configuring a specific part of the application.	Each autoconfiguration class uses conditional checks to decide if it should be applied. These conditions include the presence of specific classes, the absence of user-defined beans, and specific property settings.

CODING SHUTTLE

Core Features of AutoConfiguration

- **@PropertySources Auto-registration**
  When you run the main method of your Spring Boot Application, Spring Boot will automatically register 17 of the PropertySources for you. LINK

- **META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports**
  Every Spring Boot project has a dependency on the following library: org.springframework.boot:spring-boot-autoconfigure. It is a simple .jar file containing pretty much all of Spring Boot's magic.

CODING SHUTTLE

Core Features of AutoConfiguration

- **Enhanced Conditional Support**
  Spring Boot comes with its own set of additional @Conditional annotations, which make developers' lives easier.

  - **@ConditionalOnBean(DataSource.class)**. The condition is true only if the user specified a DataSource @Bean in a @Configuration.
  - **@ConditionalOnClass(DataSource.class)**. The condition is true if the DataSource class is on the classpath.
  - **@ConditionalOnProperty("my.property")**. The condition is true if my.property is set.

CODING SHUTTLE

Hold Up!

So basically, Spring Boot is just a bunch of AutoConfigurations classes (== normal Spring @Configurations), that create @Beans for you if certain @Conditions are met!

CODING SHUTTLE

Spring Boot Internal Flow

1. **Initialization**: When you start a Spring Boot application, the main entry point is typically a class annotated with @SpringBootApplication (or its meta-annotations). This annotation combines several other annotations such as @Configuration, @EnableAutoConfiguration, and @ComponentScan.

2. **Spring Application Context Creation**: Spring Boot creates an application context, which serves as the container for managing beans and their dependencies. It scans the classpath for components, configurations, and auto-configurations, and initializes the application context based on the detected classes and dependencies.

Spring Boot Internal Flow

3. Auto-Configuration: Spring Boot auto-configures beans and components based on the classpath and detected dependencies. It uses conditional annotations (@ConditionalOnClass, @ConditionalOnBean, etc.) to conditionally configure beans only if certain conditions are met.

4. Externalized Configuration: Spring Boot loads configuration properties from various sources, such as property files, YAML files, environment variables, and command-line arguments. It provides sensible default values for configuration properties and allows them to be easily overridden or customized.

Spring Boot Internal Flow

5. Embedded Web Server Initialization: If the application is a web application, Spring Boot initializes the embedded web server (such as Tomcat, Jetty, or Undertow) based on the application's dependencies and configurations. It configures the server with sensible defaults and starts it to listen for incoming requests.

6. Application Startup: Spring Boot invokes lifecycle callbacks such as @PostConstruct methods and initialization callbacks on beans as the application context is being initialized. Beans are instantiated, dependencies are injected, and any necessary initialization logic is executed.

Spring Boot Internal Flow

7. Application Ready: Once the initialization process is complete, the application context is fully initialized and ready to handle requests. The embedded web server is up and running, and the application is ready to serve incoming HTTP requests.

CODING SHUTTLE

1.7

Maven

CODING SHUTTLE

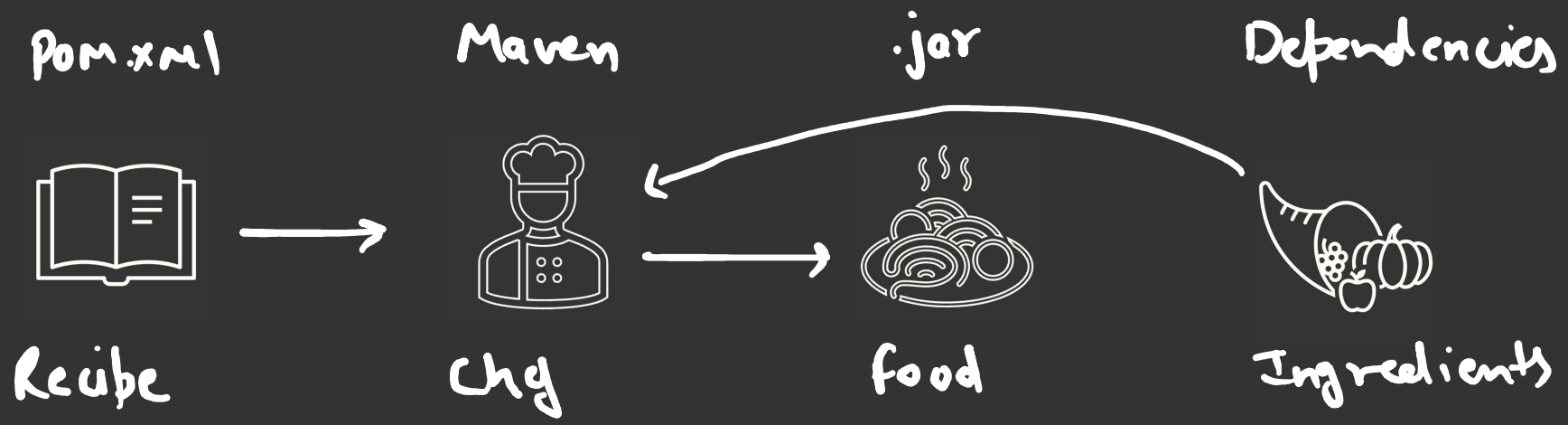
Maven

Maven is a build automation tool and project management tool primarily used for Java projects. It plays a crucial role in the development, build, and dependency management of Spring applications, including those using Spring Framework's component scanning and annotation-driven configuration.

In the upcoming slides, let’s learn how Maven fits into the Spring Framework ecosystem and the development process:

CODING SHUTTLE

Maven As the Chef

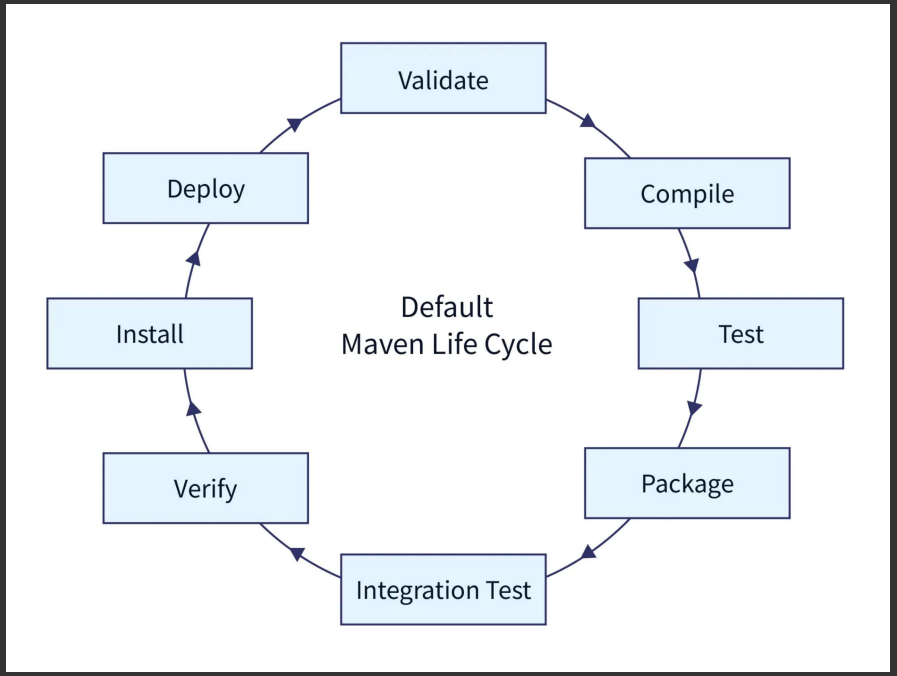


Project & Dependency Management

- Maven provides a standardized way to manage Java projects by defining project structure, dependencies, and build configurations using a declarative XML-based format (pom.xml).
- Developers use Maven to specify project metadata, dependencies, plugins, repositories, and other project-related configurations.
- Spring Framework and its various modules (e.g., Spring Core, Spring MVC, Spring Boot) are managed as dependencies in Maven projects. Developers specify the Spring dependencies in the pom.xml, and Maven handles the rest.

Build Automation

- Maven automates the build process, including compilation, testing, packaging, and deployment, using predefined build lifecycle phases (e.g., clean, compile, test, package, install).
- Maven facilitates the building and packaging of Spring applications into deployable artifacts (e.g., JAR files, WAR files) for deployment in production environments.



CODING SHUTTLE

Maven Commands

Maven Commands	Description
mvn compile	We compile the project's source code Using the mvn compile command.
mvn clean	Using the mvn clean command, All previous-build files are removed from the project.
mvn test	We execute project testing steps with the mvn test command.
mvn install	The mvn install command aids in deploying packaged WAR or JAR files by storing them in the local repository as classes.
mvn package	The mvn package command generates a WAR or JAR file for the project so that it can be distributed.
mvn deploy	The mvn deploy command is used after compilation, project testing, and project building. The packaged WAR or JAR files are copied to the remote repository so other developers can use them.

CODING SHUTTLE

Maven Commands

Maven Commands	Description
mvn spring-boot:run	Runs a Spring Boot application directly from the source code without packaging it into a JAR or WAR file.
mvn spring-boot:build-image	Builds a Docker image of the Spring Boot application using the Spring Boot Maven plugin.

CODING SHUTTLE

Week 1

Homework

CODING SHUTTLE

Recap

1. Beans
2. Annotations
3. Dependency Injection
4. IoC Container
5. Maven
6. Spring Boot Autoconfiguration
7. Internal Flow of Spring Boot

CODING SHUTTLE

Homework

1. Make a list of all the annotations you have learned so far.
2. Make a list of scenarios where you feel Spring Framework can be very useful over NodeJS.
3. Make a list of scenarios where you feel Spring Boot can be very useful over Spring Framework.

CODING SHUTTLE

Homework

4. Alice and her Bakery

- Create a class called CakeBaker, that is dependent on two other classes called Frosting and Syrup. This class has a function called bakeCake().
- Create two interfaces of type Frosting and Syrup with a function called getFrostingType and getSyrupType respectively.
- Create two implementations of these two interfaces (so total 4 classes) for Chocolate and Strawberry flavors.
- Use Dependency injection to inject the Frosting and Syrup dependencies into CakeBaker and also to call the bakeCake function of the CakeBaker class.