Spring Workshop with Adrian

Agenda

1. Why to use any framework like Spring?
2. What is Spring Framework?
3. What is Spring Boot?
4. Inversion Of Control & Dependency Injection
5. Spring Container
6. Spring Bean
7. Bootstrapping Spring Boot App

6. Digging into code
    i. Code structure
    ii. Gradle + Gradle tasks
    iii. @SpringBootApplication
    iv. Spring Boot actuator
    v. API endpoint
    vi. Spring DevTools
    vii. App configuration – application properties
    viii. Bean types and creation
    ix. Dependency injection
    x. Controller – Service – Repository architecture
        i. Spring Data
        ii. Rest Exception Handler
        iii. Transactional service
    xi. Controller Tests

Why to use any framework like Spring?

- Helps us focus on the core task rather than the boilerplate associated with it.
- Brings together years of wisdom in the form of design patterns.
- Helps us adhere to the industry.
- Brings down the total cost of ownership for the application.

Source: https://www.baeldung.com/spring-why-to-choose

What is Spring Framework?

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

A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

Features:

- Core technologies: dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP.
- Testing: mock objects, TestContext framework, Spring MVC Test, WebTestClient.
- Data Access: transactions, DAO support, JDBC, ORM, Marshalling XML.
- Spring MVC and Spring WebFlux web frameworks.
- Integration: remoting, JMS, JCA, JMX, email, tasks, scheduling, cache.

Source: https://spring.io/projects/spring-framework

What is Spring Boot?

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

Features:

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration

Source: https://spring.io/projects/spring-boot

Inversion of Control & Dependency Injection

IoC

In contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behaviour built in. If we want to add our own behaviour, we need to extend the classes of the framework or plugin our own classes.

The advantages of this architecture are:

- decoupling the execution of a task from its implementation
- making it easier to switch between different implementations
- greater modularity of a program
- greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts

DI

Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies.

Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.

Without DI

```
public class Store {
    private Item item;

    public Store() {
        item = new ItemImpl1();
    }
}
```

With DI

```
public class Store {
    private Item item;

    public Store(Item item) {
        this.item = item;
    }
}
```

Spring Container

Spring container is at the core of the Spring Framework. The container will:

- create the objects,
- wire them together,
- configure them,
- manage their complete life cycle from creation till destruction.

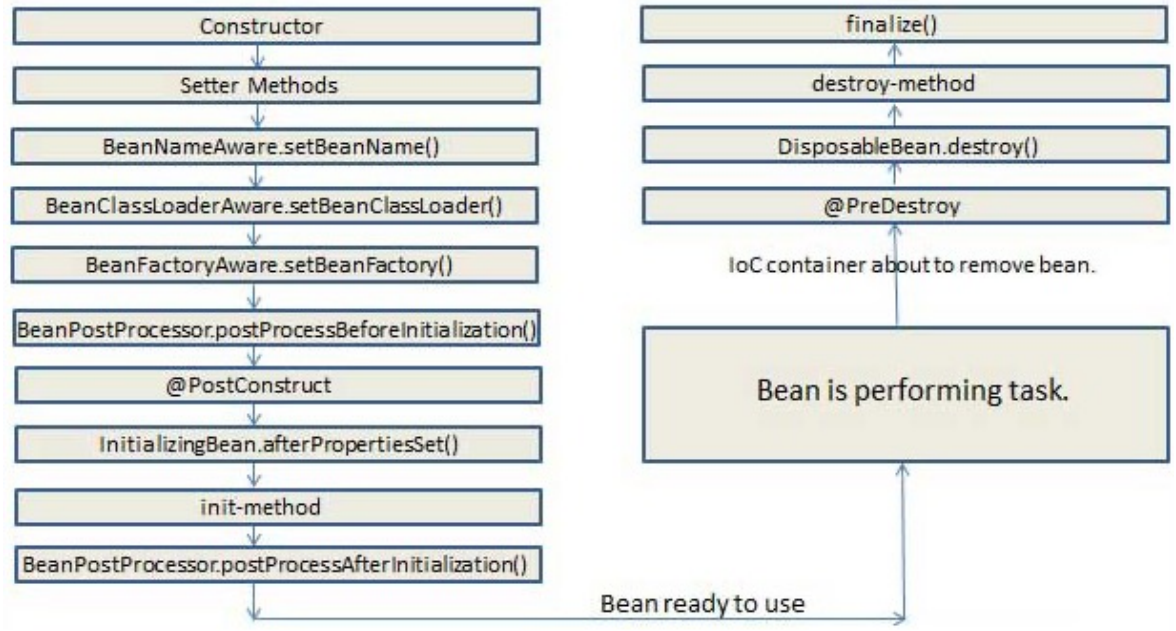
The Spring container uses DI to manage the components that make up an application. These objects are called Spring Beans.

Spring Bean

Beans are objects that form the backbone of the application. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.

Bean scopes	
singleton	Only one instance per Spring container (default).
prototype	A new instance every time bean is requested.
request	Single instance per HTTP request.
session	Single instance per HTTP session.
global-session	Single instance per global HTTP session.

Bean lifecycle



Source: https://www.concretepage.com/spring/spring-bean-life-cycle-tutorial

Spring Boot Annotations

Spring Boot and Web Annotations

Use annotations to configure your web application.

T **@SpringBootApplication** - uses @Configuration, @EnableAutoConfiguration and @ComponentScan.

T **@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

T **@Controller** - marks the class as web controller, capable of handling the requests.

T **@RestController** - a convenience annotation of a @Controller and @ResponseBody.

M T **@ResponseBody** - makes Spring bind method's return value to the web response body.

M **@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

P **@RequestParam** - bind HTTP parameters into method arguments.

P **@PathVariable** - binds placeholder from the URI to the method parameter.

Spring Framework Annotations

Spring uses dependancy injection to configure and bind your appliation together.

T **@Configuration** - used to mark a class as a source of the bean definitions.

T **@ComponentScan** - makes Spring scan the packages configured with it for the @Configuration classes.

T **@Import** - loads additional configuration. This one works even when you specify the beans in an XML file.

T **@Component** - turns the class into a Spring bean at the auto-scan time.

T **@Service** - tells Spring that it's safe to manage @Components with more freedom than regular components.

C F M **@Autowired** - wires the application parts together, on the fields, constructors, or methods in a component.

M **@Bean** - specifies a returned bean to be managed by Spring context. The returned bean has the same name as the factory method.

M **@Lookup** - tells Spring to return an instance of the method's return type when we invoke it.

T M **@Primary** - gives higher preference to a bean when there are multiple beans of the same type.

C F M **@Required** - shows that the setter method must be configured to be dependency-injected with a value at configuration time.

C F M **@Value** - used to assign values into fields in Spring-managed beans. It's compatible with the constructor, setter, and field injection.

T M **@DependsOn** - makes Spring initialize other beans before the annotated one.

T M **@Lazy** - makes beans to initialize lazily. @Lazy annotation may be used on any class directly or indirectly annotated with @Component or on methods annotated with @Bean.

T M **@Scope** - used to define the scope of a @Component class or a @Bean definition and can be either singleton, prototype, request, session, globalSession, or custom scope.

T **@Profile** - adds beans to the application only when that profile is active.

Legend: T - Class F - Field Annotation C - Constructor Annotation M - Method P - Parameter

Source: https://www.jrebel.com/sites/rebel/files/pdfs/cheatsheet-jrebel-spring-annotations.pdf

Bootstrapping Spring Boot App

Spring Boot Initializr

- https://start.spring.io

Spring Boot Starters list

- https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using.build-systems.starters