eazy bytes

Spring, SpringBoot, JPA, Hibernate

Zero To Master

Agenda of the course

eazy bytes

Intro to Spring framework

- *What is Spring?*
- *Spring Vs Java EE*
- *Evolution of Spring*
- *Spring release timeline*
- *Different projects inside Spring*

Spring Core

- *Inversion of Control (IoC)*
- *Dependency Injection (DI)*
- *Different approaches for Beans creation*
- *Autowiring, Bean Scopes*
- *Aspect-Oriented Programming (AOP)*

Spring MVC

- *Introduction to MVC pattern*
- *Overview of Web Apps*
- *How Web Apps works ?*
- *Spring MVC internal flow*
- *Create a web App using Spring MVC*
- *Spring MVC validations*

Agenda of the course

eazy bytes

Thymeleaf

- *How to build dynamic web apps using Thymeleaf & Spring*
- *Thymeleaf integration with Spring, Spring MVC, Spring Security*

Spring Boot

- *Why Spring Boot?*
- *Auto-configuration*
- *Build Web Apps using SpringBoot, Thymeleaf, Spring MVC*
- *Spring Boot Dev Tools*
- *Spring Boot H2 Database*
- *Connecting to AWS MYSQL DB*

Spring Security

- *Authentication & Authorization*
- *Securing web Apps/REST APIs using Spring Security*
- *Role based access*
- *Cross-Site Request Forgery (CSRF)*
- *Cross-Origin Resource Sharing (CORS)*

Agenda of the course

eazy bytes

Spring JDBC

- *Problems with Core Java JDBC*
- *Advantages with Spring JDBC*
- *Performing CRUD operations with Spring JDBC*
- *Details about JdbcTemplate*
- *RowMapper*

Spring Data

- *Why do we need ORM frameworks?*
- *Introduction to JPA*
- *Derived Query methods in JPA*
- *OneToOne, OneToMany, ManyToOne, ManyToMany mappings inside JPA/Hibernate*
- *Sorting, Pagination, JPQL*

Spring REST

- *Building Rest Services*
- *Consuming Rest Services using OpenFeign, Web Client, RestTemplate*
- *Securing Rest Services*
- *Spring Data Rest*
- *HAL Explorer*

Agenda of the course

eazy bytes

Logging

- *Logging severities*
- *How to make logging configurations inside Spring Boot*
- *Writing logs into a file*

Properties & Profiles

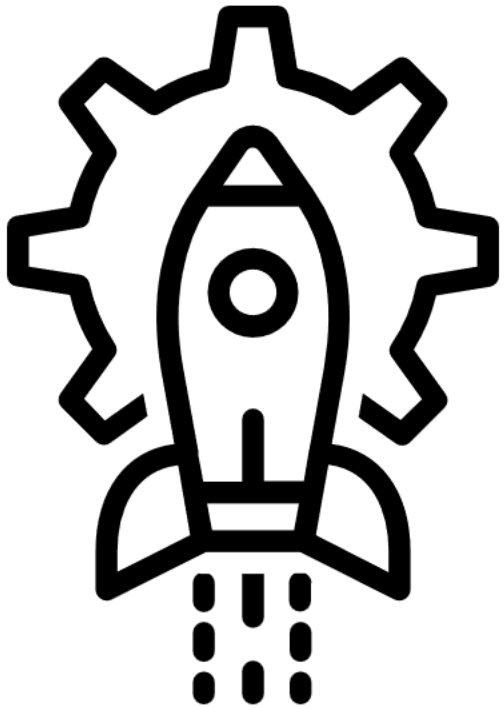
- *How to define custom properties*
- *How to read properties in Java code*
- *Deep dive on Spring Profiles*
- *Activating a Profile*
- *Conditional Bean creation using Profile*

Actuator

- *Introduction to SpringBoot Actuator*
- *Exploring the APIs of Actuator*
- *Securing Actuator*
- *Viewing Actuator Data in Spring Boot Admin*

Agenda of the course

eazy bytes



DEPLOYING SPRINGBOOT WEBAPP INTO CLOUD (AWS)

aws

WHAT IS SPRING?

eazy bytes

The Spring Framework (shortly, Spring) is a mature, powerful and highly flexible framework focused on building web applications in Java.

Spring makes programming Java quicker, easier, and safer for everybody. It's focus on speed, simplicity, and productivity has made it the world's most popular Java framework.

Whether you're building secure, reactive, cloud-based microservices for the web, or complex streaming data flows for the enterprise, Spring has the tools to help.

Born as an alternative to EJBs in the early 2000s, the Spring framework quickly overtook its opponent with its simplicity, variety of features, and its third-party library integrations.

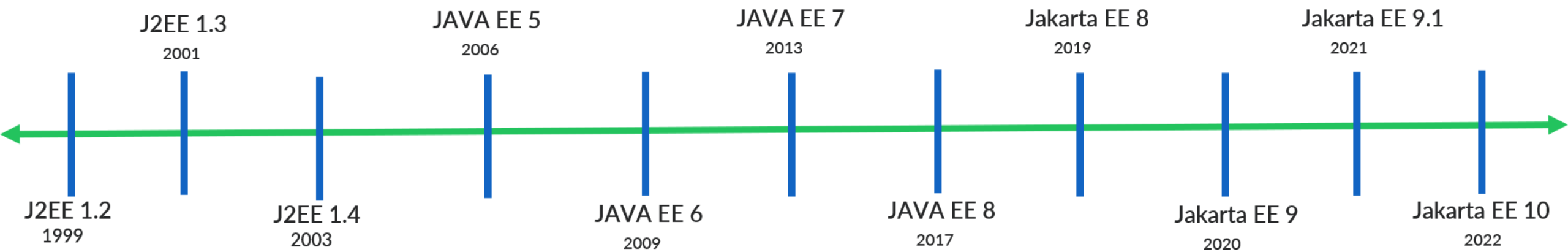
It is so popular, that its main competitor quit the race when Oracle stopped the evolution of Java EE 8, and the community took over its maintenance via Jakarta EE.

The main reason of Spring framework success is, it regularly introduces features/projects based on the latest market trends, needs of the Dev community. For ex: SpringBoot

Spring is open source. It has a large and active community that provides continuous feedback based on a diverse range of real-world use cases.

JAVA EE RELEASE TIMELINE

eazy bytes



- *Java/Jakarta Enterprise Edition (EE) contains Servlets, JSPs, EJB, JMS, RMI, JPA, JSF, JAXB, JAX-WS, Web Sockets etc.*
- *Components of Java/Jakarta Enterprise Edition (EE) like EJB, Servlets are complex in nature due to which everyone adapted Spring framework for web applications development.*
- *Java EE quit the race against the Spring framework, when Oracle stopped the evolution of Java EE 8, and the community took over its maintenance via Jakarta EE.*
- *Since Oracle owns the trademark for the name "Java", Java EE renamed to Jakarta EE. All the packages are updated with javax.* to jakarta.* namespace change.*

SPRING RELEASE TIMELINE

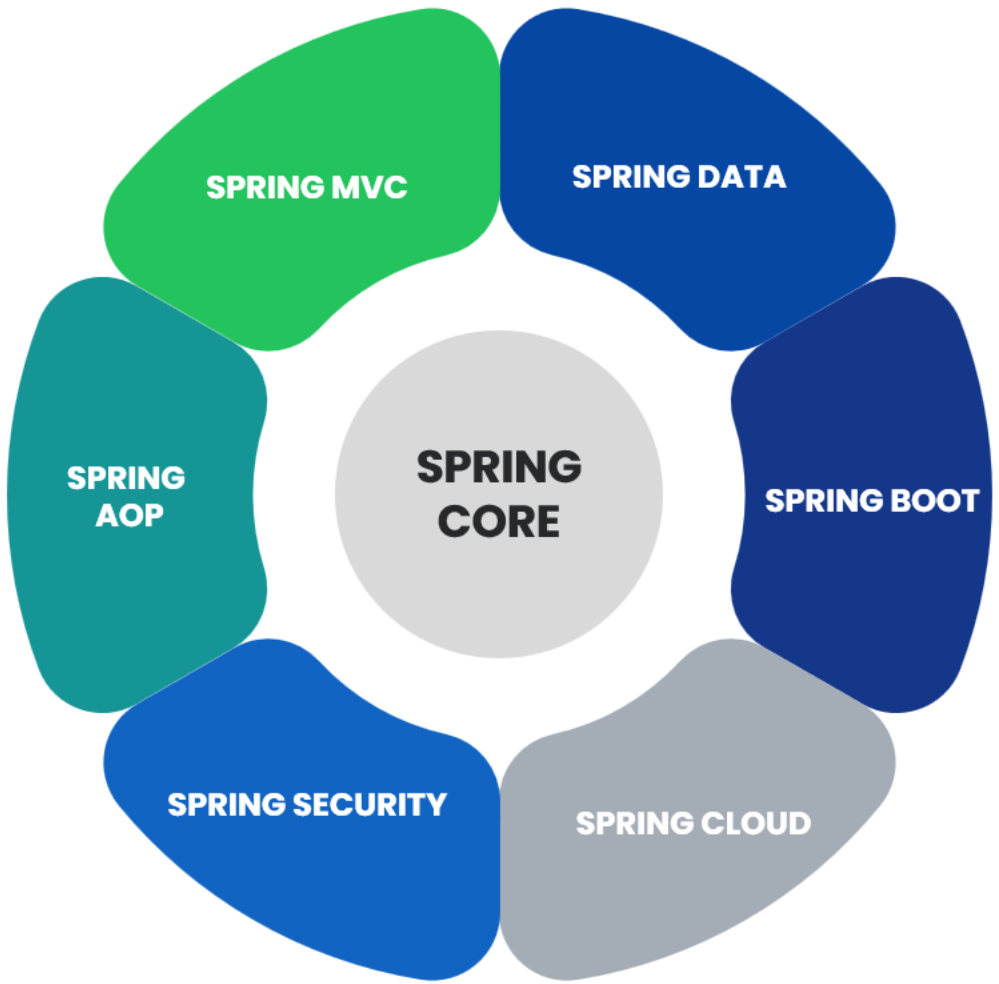
eazy bytes

SPRING VERSION HISTORY

- *The first version of Spring was written by Rod Johnson, who released the framework with the publication of his book Expert One-on-One J2EE Design and Development in October 2002*
- *Spring came into being in 2003 as a response to the complexity of the early J2EE specifications. While some consider Java EE and Spring to be in competition, Spring is, in fact, complementary to Java EE. The Spring programming model does not embrace the Java EE platform specification; rather, it integrates with carefully selected individual specifications from the EE umbrella*
- *Spring continues to innovate and to evolve. Beyond the Spring Framework, there are other projects, such as Spring Boot, Spring Security, Spring Data, Spring Cloud, Spring Batch, among others.*

SPRING CORE

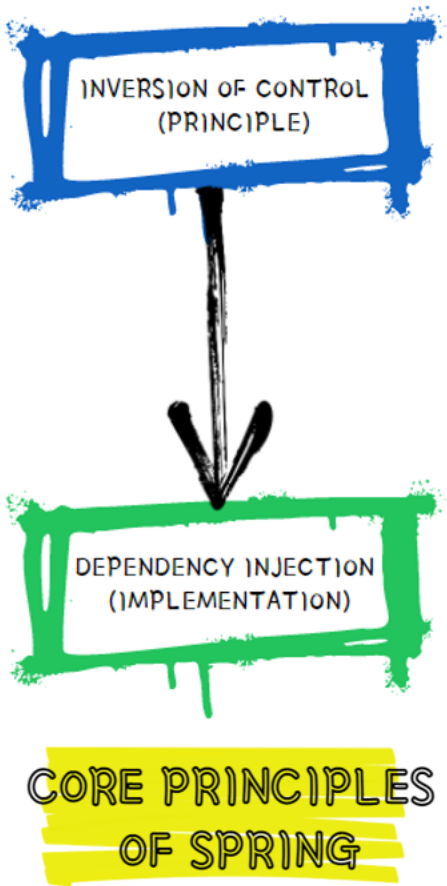
eazy bytes



- Spring Core is the heart of entire Spring. It contains some base framework classes, principles and mechanisms.
- The entire Spring Framework and other projects of Spring are developed on top of the Spring Core.
- Spring Core contains following important components,
  - ✓ IoC (Inversion of Control)
  - ✓ DI (Dependency Injection)
  - ✓ Beans
  - ✓ Context
  - ✓ SpEL (Spring Expression Language)
  - ✓ IoC Container

INVERSION OF CONTROL & DEPENDENCY INJECTION

eazy bytes

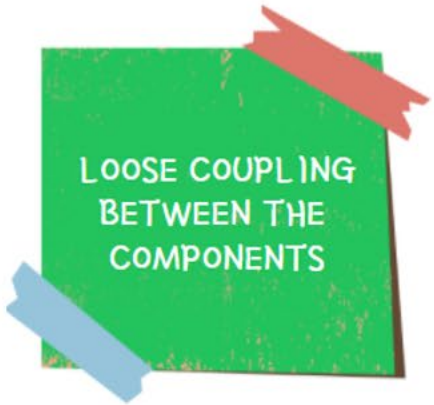


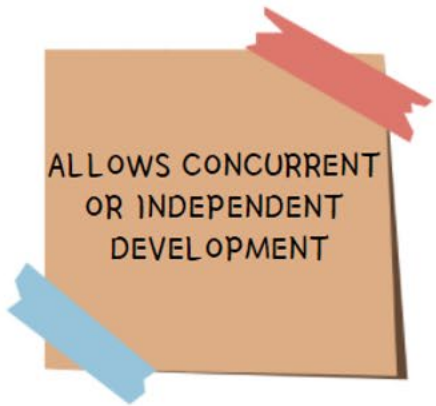
CORE PRINCIPLES OF SPRING

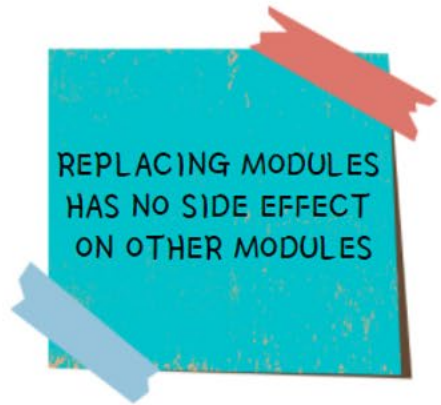
- Inversion of Control (IoC) is a Software Design Principle, independent of language, which does not actually create the objects but describes the way in which object is being created.
- IoC is the principle, where the control flow of a program is inverted: instead of the programmer controlling the flow of a program, the framework or service takes control of the program flow.
- Dependency Injection is the pattern through which Inversion of Control achieved.
- Through Dependency Injection, the responsibility of creating objects is shifted from the application to the Spring IoC container. It reduces coupling between multiple objects as it is dynamically injected by the framework.

ADVANTAGES OF IoC & DI

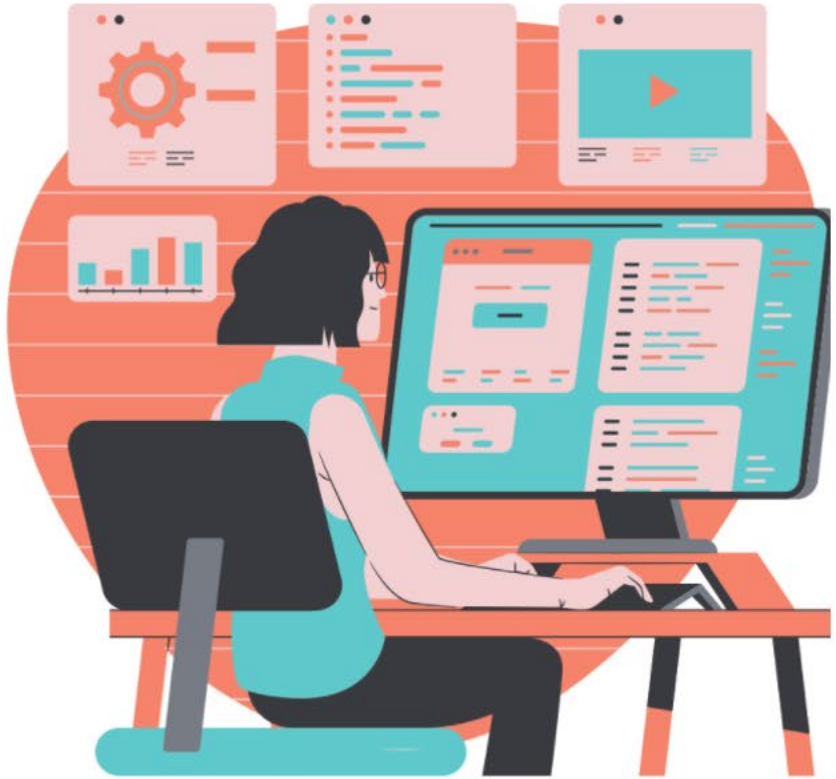
eazy bytes







eazy bytes



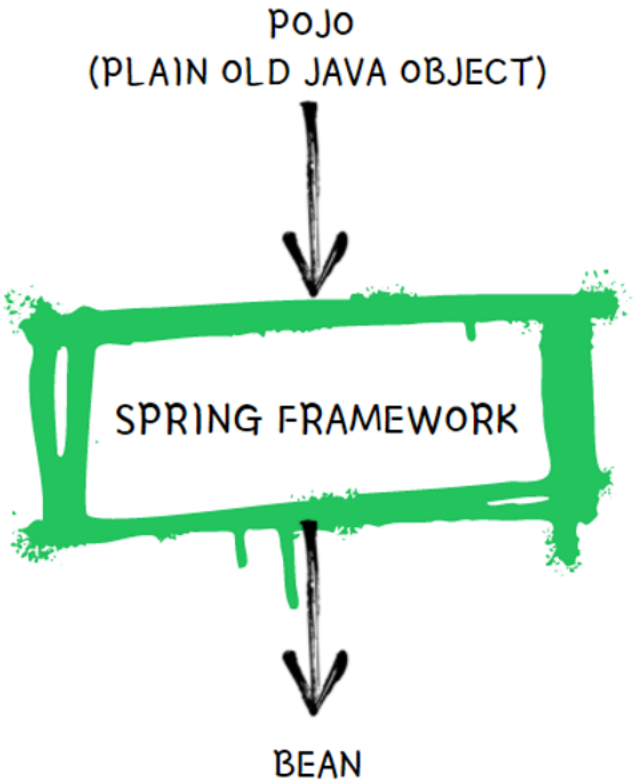
REAL LIFE LOOSE COUPLING EXAMPLE

TIGHT COUPLING SCENARIO

LOOSE COUPLING SCENARIO

SPRING BEANS, CONTEXT, SpEL

eazy bytes



- Any normal Java class that is instantiated, assembled, and otherwise managed by a Spring IoC container is called Spring Bean.
- These beans are created with the configuration metadata that you supply to the container either in the form of XML configs and Annotations.
- Spring IoC Container manages the lifecycle of Spring Bean scope and injecting any required dependencies in the bean.
- Context is like a memory location of your app in which we add all the object instances that we want the framework to manage. By default, Spring doesn't know any of the objects you define in your application. To enable Spring to see your objects, you need to add them to the context.
- The SpEL provides a powerful expression language for querying and manipulating an object graph at runtime like setting and getting property values, property assignment, method invocation etc.

SPRING IoC CONTAINER

eazy bytes

Spring IoC Container

- The IoC container is responsible
  - ✓ to instantiate the application class
  - ✓ to configure the object
  - ✓ to assemble the dependencies between the objects

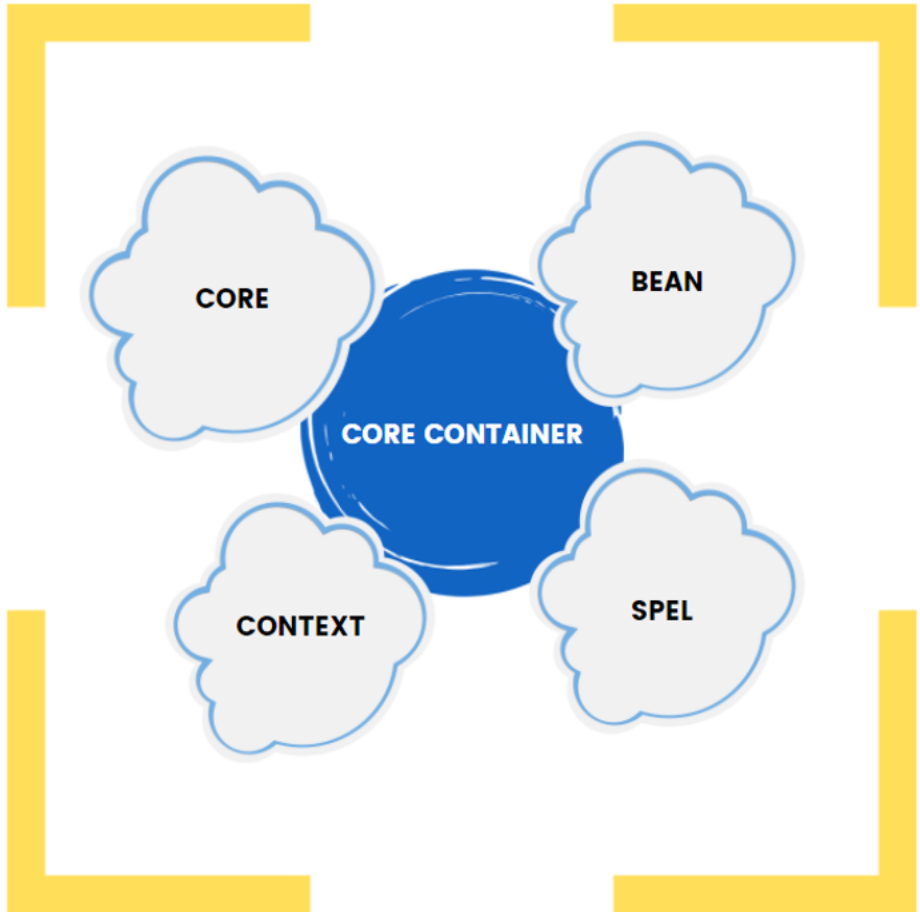
- There are two types of IoC containers. They are:

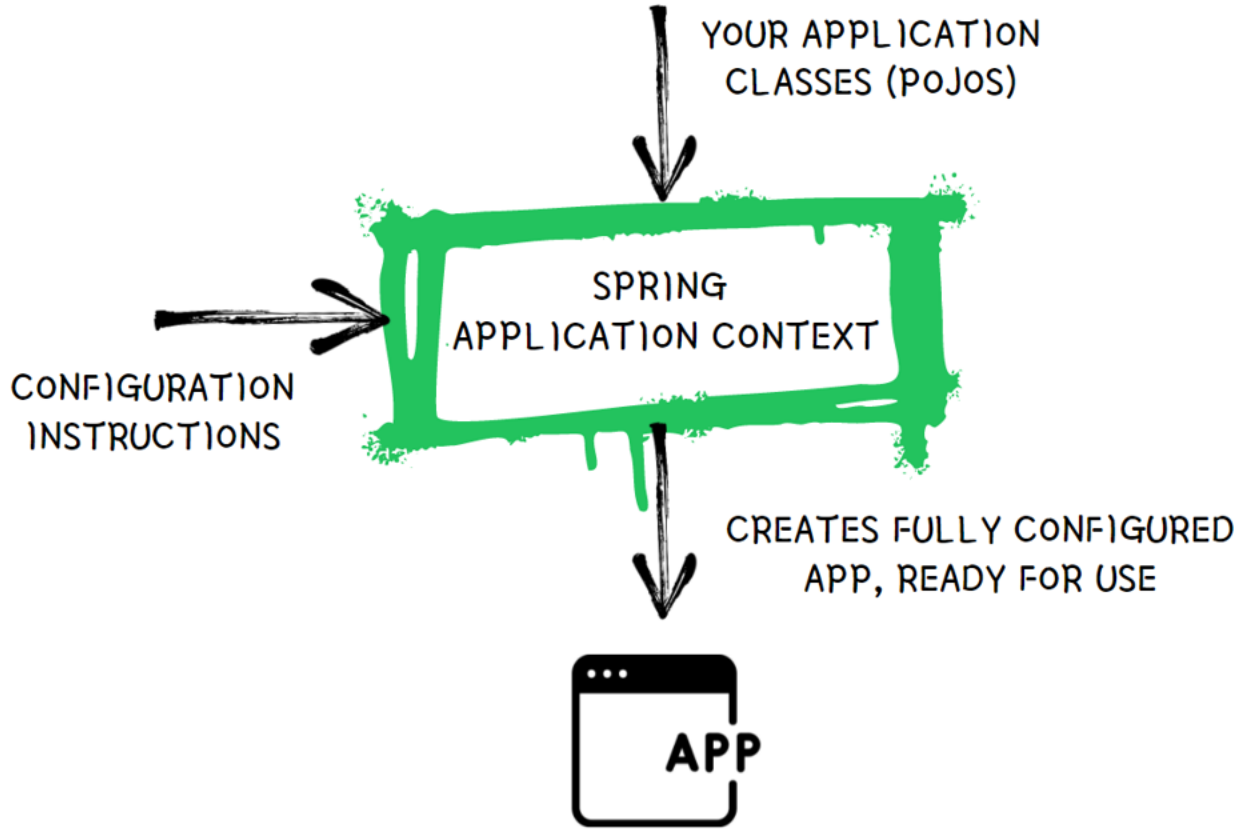
  - ✓ org.springframework.beans.factory.BeanFactory
  - ✓ org.springframework.context.ApplicationContext

- The Spring container uses dependency injection (DI) to manage the components/objects that make up an application.

SPRING IoC CONTAINER

eazy bytes





eazy bytes

MAVEN

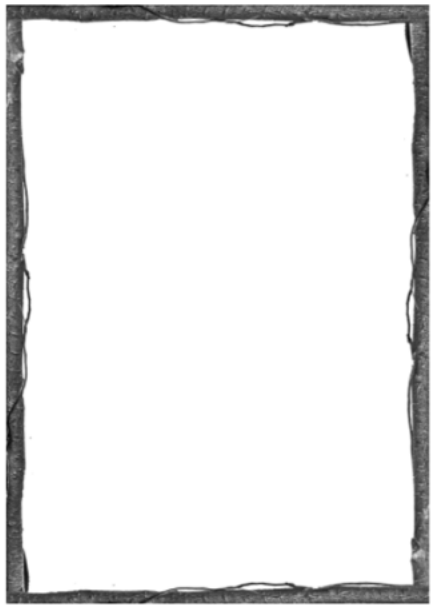
ADDING NEW BEANS TO SPRING CONTEXT

eazy bytes

When we create an java object with new () operator directly as shown below, then your Spring Context/Spring IoC Container will not have any clue of the object.

```
Vehicle vehicle = new Vehicle();
```

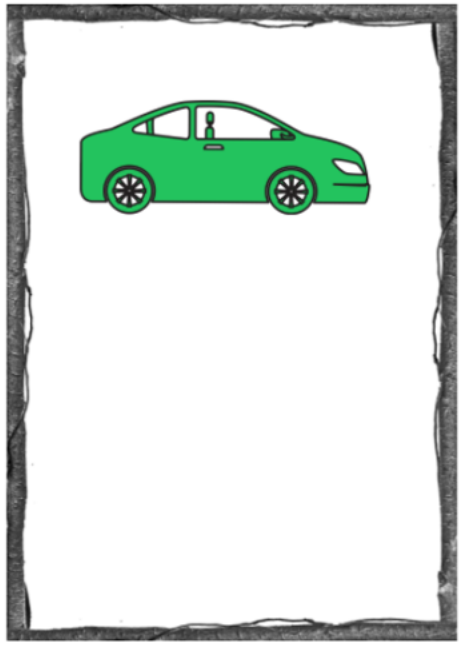
SPRING CONTEXT



@Bean annotation lets Spring know that it needs to call this method when it initializes its context and adds the returned object/value to the Spring context/Spring IoC Container.

```
@Bean
Vehicle vehicle() {
    var veh = new Vehicle();
    veh.setName("Audi 8");
    return veh;
}
```

SPRING CONTEXT

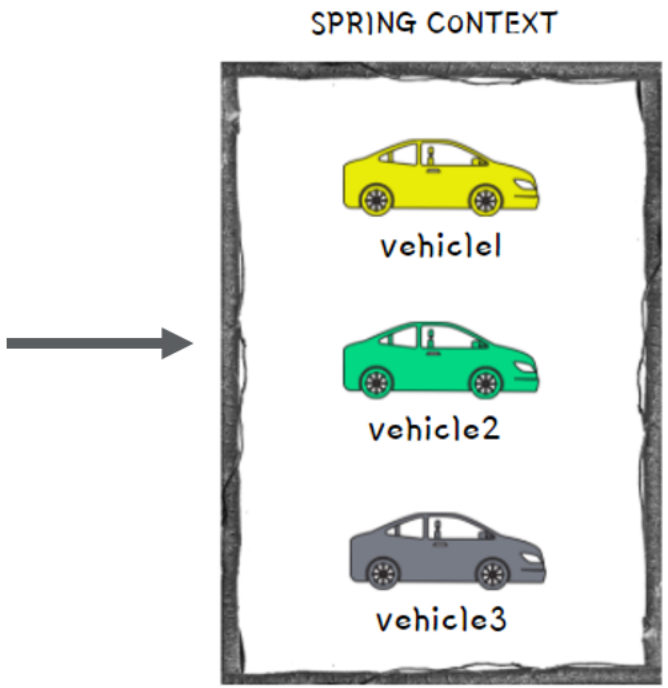


NoUniqueBeanDefinitionException

eazy bytes

When we create multiple objects of same type and try to fetch the bean from context by type, then Spring cannot guess which instance you've declared you refer to. This will lead to NoUniqueBeanDefinitionException like shown below,

```java
@Bean
Vehicle vehicle1() {
    var veh = new Vehicle();
    veh.setName("Audi");
    return veh;
}

@Bean
Vehicle vehicle2() {
    var veh = new Vehicle();
    veh.setName("Honda");
    return veh;
}

@Bean
Vehicle vehicle3() {
    var veh = new Vehicle();
    veh.setName("Ferrari");
    return veh;
}
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Vehicle veh = context.getBean(Vehicle.class);
```

NoUniqueBeanDefinitionException

Output on Console

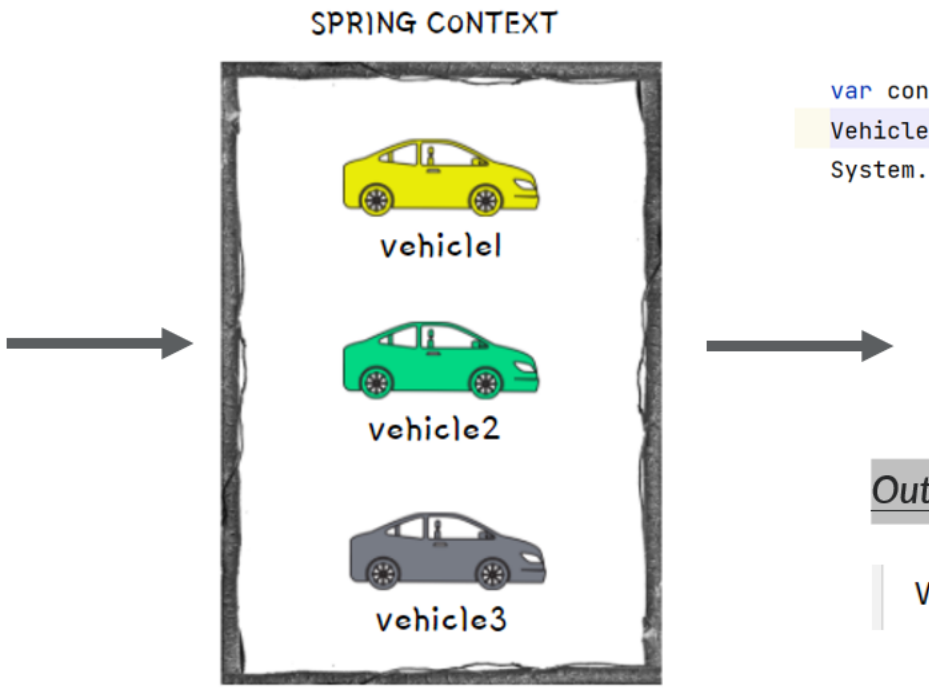
```
Exception in thread "main" org.springframework.beans.factory.NoUniqueBeanDefinitionException Create breakpoint : No qualifying bean of type
 'com.example.beans.Vehicle' available: expected single matching bean but found 3: vehicle1,vehicle2,vehicle3
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveNamedBean(DefaultListableBeanFactory.java:1262)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveBean(DefaultListableBeanFactory.java:494)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBeanFactory.java:349)
    at org.springframework.beans.factory.support.DefaultListableBeanFactory.getBean(DefaultListableBeanFactory.java:342)
    at org.springframework.context.support.AbstractApplicationContext.getBean(AbstractApplicationContext.java:1172)
    at com.example.main.Example2.main(Example2.java:18)
```

NoUniqueBeanDefinitionException

eazy bytes

To avoid NoUniqueBeanDefinitionException in these kind of scenarios, we can fetch the bean from the context by mentioning it's name like shown below,

```java
@Bean
Vehicle vehicle1() {
    var veh = new Vehicle();
    veh.setName("Audi");
    return veh;
}

@Bean
Vehicle vehicle2() {
    var veh = new Vehicle();
    veh.setName("Honda");
    return veh;
}

@Bean
Vehicle vehicle3() {
    var veh = new Vehicle();
    veh.setName("Ferrari");
    return veh;
}
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Vehicle veh = context.getBean( name: "vehicle1",Vehicle.class);
System.out.println("Vehicle name from Spring Context is: " + veh.getName());
```

Output on Console

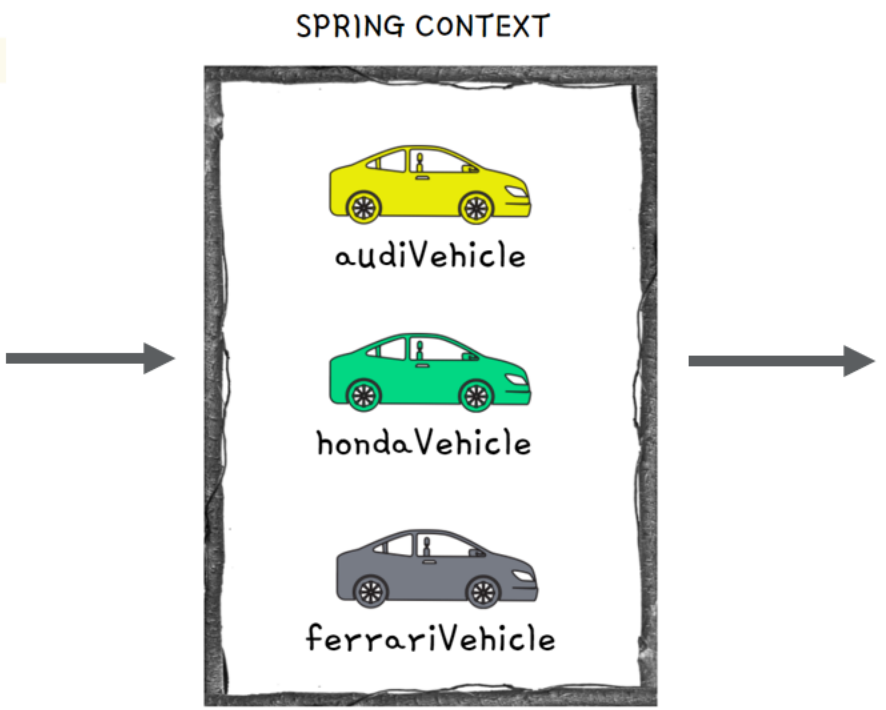
```
Vehicle name from Spring Context is: Audi
```

DIFFERENT WAYS TO NAME A BEAN

eazy bytes

By default, Spring will consider the method name as the bean name. But if we have a custom requirement to define a separate bean name, then we can use any of the below approach with the help of @Bean annotation,

```java
@Bean(name="audiVehicle")
Vehicle vehicle1() {
    var veh = new Vehicle();
    veh.setName("Audi");
    return veh;
}

@Bean(value="hondaVehicle")
Vehicle vehicle2() {
    var veh = new Vehicle();
    veh.setName("Honda");
    return veh;
}

@Bean("ferrariVehicle")
Vehicle vehicle3() {
    var veh = new Vehicle();
    veh.setName("Ferrari");
    return veh;
}
```




```java
Vehicle veh1 = context.getBean( name: "audiVehicle",Vehicle.class);
System.out.println("Vehicle name from Spring Context is: " + veh1.getName());

Vehicle veh2 = context.getBean( name: "hondaVehicle",Vehicle.class);
System.out.println("Vehicle name from Spring Context is: " + veh2.getName());

Vehicle veh3 = context.getBean( name: "ferrariVehicle",Vehicle.class);
System.out.println("Vehicle name from Spring Context is: " + veh3.getName());
```

Output on Console

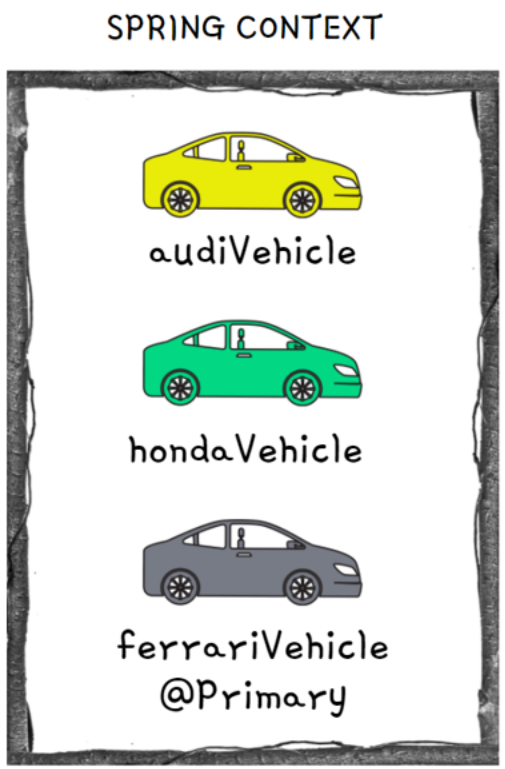
```
Vehicle name from Spring Context is: Audi
Vehicle name from Spring Context is: Honda
Vehicle name from Spring Context is: Ferrari
```

@Primary Annotation

eazy bytes

When you have multiple beans of the same kind inside the Spring context, you can make one of them primary by using @Primary annotation. Primary bean is the one which Spring will choose if it has multiple options and you don't specify a name. In other words, it is the default bean that Spring Context will consider in case of confusion due to multiple beans present of same type.

```java
@Bean(name="audiVehicle")
Vehicle vehicle1() {
    var veh = new Vehicle();
    veh.setName("Audi");
    return veh;
}

@Bean(value="hondaVehicle")
Vehicle vehicle2() {
    var veh = new Vehicle();
    veh.setName("Honda");
    return veh;
}

@Primary
@Bean("ferrariVehicle")
Vehicle vehicle3() {
    var veh = new Vehicle();
    veh.setName("Ferrari");
    return veh;
}
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Primary Vehicle name from Spring Context is: " + vehicle.getName());
```

Output on Console

```
Primary Vehicle name from Spring Context is: Ferrari
```

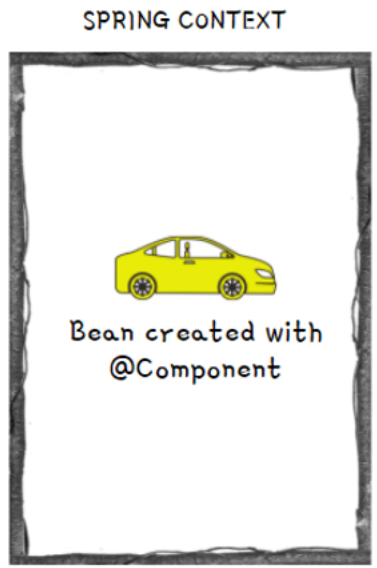
@Component Annotation

eazy bytes

@Component is one of the most commonly used stereotype annotation by developers. Using this we can easily create and add a bean to the Spring context by writing less code compared to the @Bean option. With stereotype annotations, we need to add the annotation above the class for which we need to have an instance in the Spring context.

Using @ComponentScan annotation over the configuration class, instruct Spring on where to find the classes you marked with stereotype annotations.

```java
@Component
public class Vehicle {

    private String name;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public void printHello(){
        System.out.println(
            "Printing Hello from Component Vehicle Bean");
    }
}
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Component Vehicle name from Spring Context is: " + vehicle.getName());
vehicle.printHello();
```

```java
@Configuration
@ComponentScan(basePackages = "com.example.beans")
public class ProjectConfig {

}
```

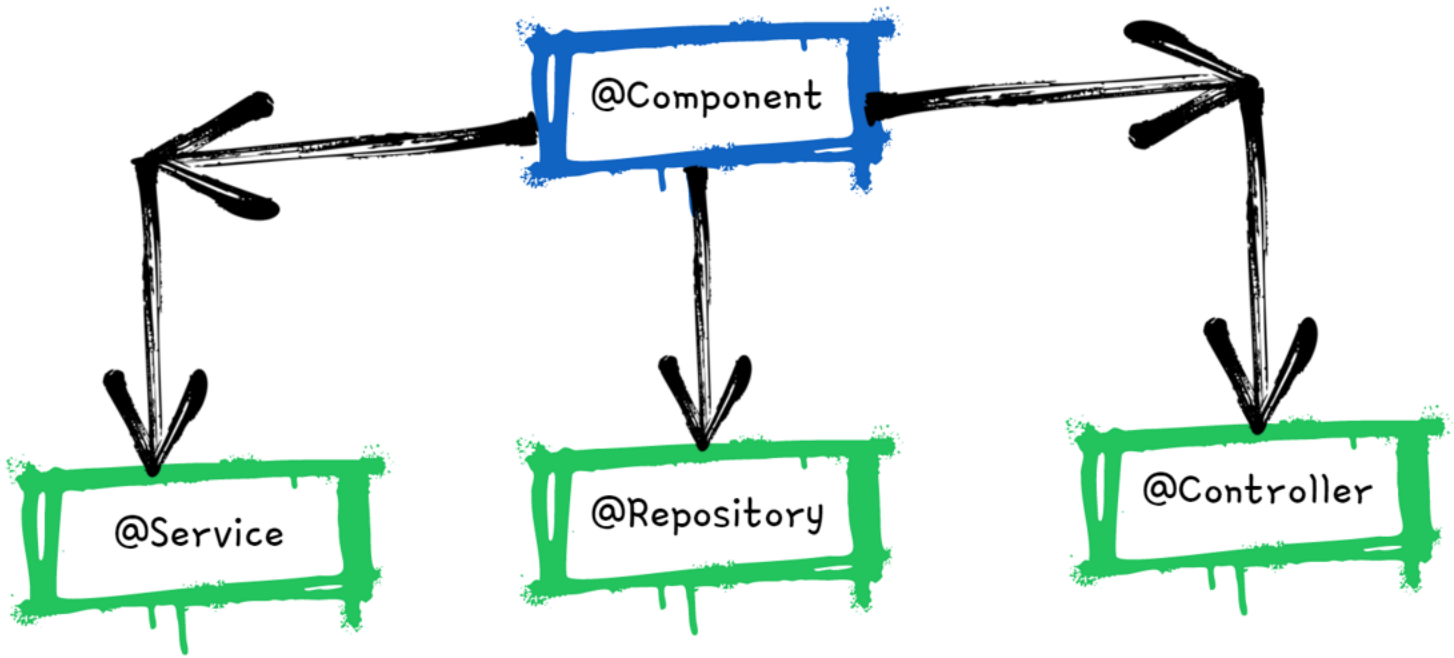
Output on Console

```
Component Vehicle name from Spring Context is: null
Printing Hello from Component Vehicle Bean
```

Spring Stereotype Annotations

eazy bytes

- *Spring provides special annotations called Stereotype annotations which will help to create the Spring beans automatically in the application context.*
- *The stereotype annotations in spring are @Component, @Service, @Repository and @Controller*



@Component is used as general on top of any Java class. It is the base for other annotations.

@Service can be used on top of the classes inside the service layer especially where we write business logic and make external API calls.

@Repostiory can be used on top of the classes which handles the code related to Database access related operations like Insert, Update, Delete etc.

@Controller can be used on top of the classes inside the Controller layer of MVC applications.

@Bean Vs @Component

eazy bytes

@Bean

- One or more instances of the class can be added to the Spring Context
- We can create an object instance of any type of class including present inside libraries like String etc.
- Usually we need to write more code like separate methods to create beans instances
- Developer will have full control in creating and configuring the bean
- Spring framework creates the bean based on the instructions and values provided by Developer

@Component

- Only one instance of the class can be added to the Spring context
- We can create an object instance for application class only which are created by Dev team
- Bean instances can be created with very less code like using @Component on top of the class
- Developer will not have any control in creating and configuring the bean
- Spring framework takes charge of creating the bean and post that Developer will have access to it

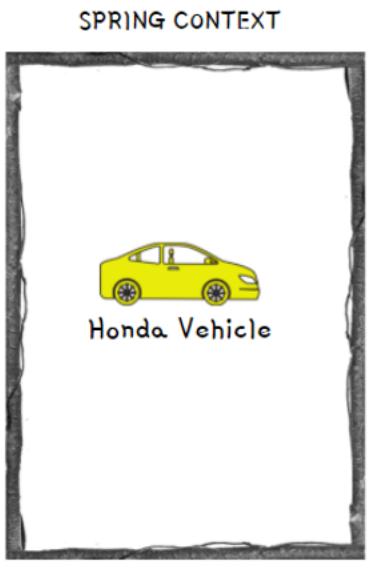
@PostConstruct Annotation

eazy bytes

- *We have seen that when we are using stereotype annotations, we don't have control while creating a bean. But what if we want to execute some instructions post Spring creates the bean. For the same, we can use @PostConstruct annotation.*
- *We can define a method in the component class and annotate that method with @PostConstruct, which instructs Spring to execute that method after it finishes creating the bean.*
- *Spring borrows the @PostConstruct annotation from Java EE.*

```java
@Component
public class Vehicle {

    private String name;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    @PostConstruct
    public void initialize() {
        this.name = "Honda";
    }

    public void printHello(){...}
}
```

```java
@Configuration
@ComponentScan(basePackages = "com.example.beans")
public class ProjectConfig {

}
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Component Vehicle name from Spring Context is: " + vehicle.getName());
vehicle.printHello();
```

Output on Console

```
Component Vehicle name from Spring Context is: Honda
Printing Hello from Component Vehicle Bean
```

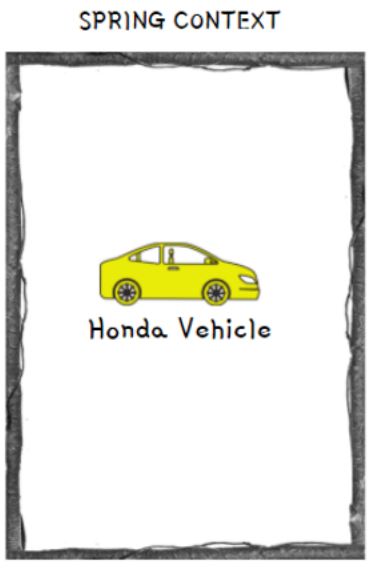
@PreDestroy Annotation

eazy bytes

- *@PreDestory annotation can be used on top of the methods and Spring will make sure to call this method just before clearing and destroying the context.*
- *This can be used in the scenarios where we want to close any IO resources, Database connections etc.*
- *Spring borrows the @PreDestory annotation also from Java EE.*

```java
@Component
public class Vehicle {

    private String name;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    @PreDestroy
    public void destroy() {
        System.out.println(
                "Destroying Vehicle Bean");
    }
}
```

```java
@Configuration
@ComponentScan(basePackages = "com.example.beans")
public class ProjectConfig {

}
```




```java
var context = new AnnotationConfigApplicationContext
                (ProjectConfig.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Component Vehicle name from " +
        "Spring Context is: " + vehicle.getName());
vehicle.printHello();
context.close();
```

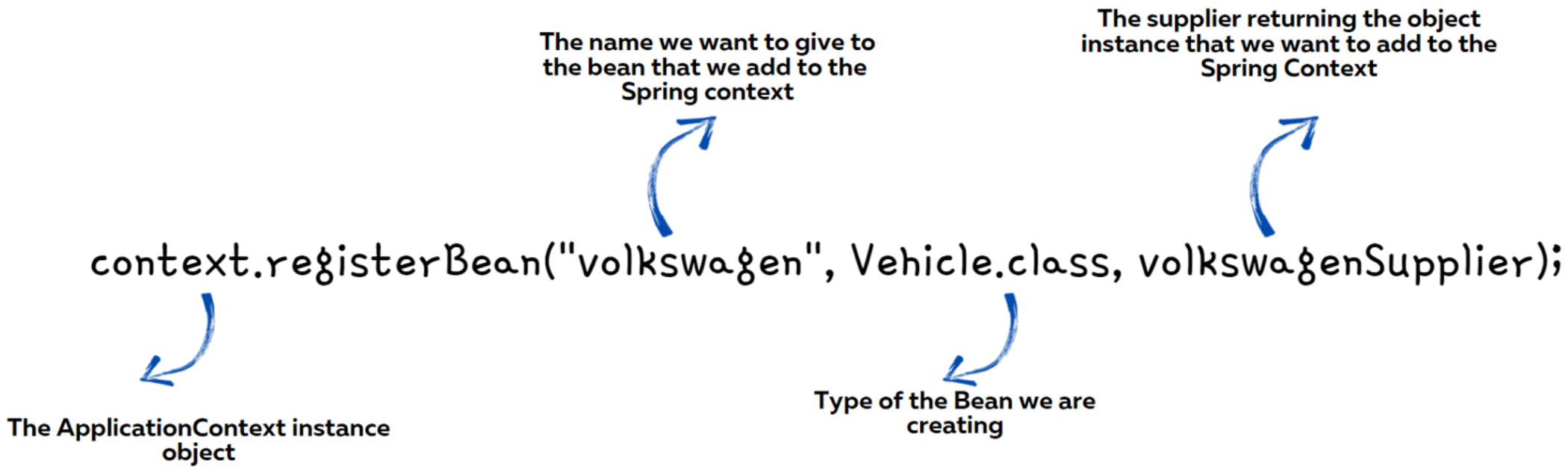
Output on Console

```
Component Vehicle name from Spring Context is: Honda
Printing Hello from Component Vehicle Bean
Destroying Vehicle Bean
```

ADDING NEW BEANS PROGRAMMATICALLY

eazy bytes

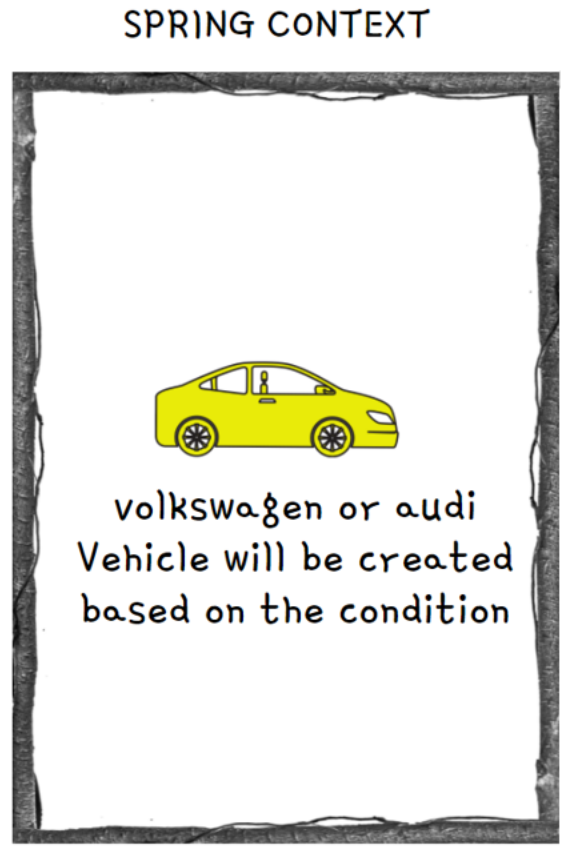
- *Sometimes we want to create new instances of an object and add them into the Spring context based on a programming condition. For the same, from Spring 5 version, a new approach is provided to create the beans programmatically by invoking the registerBean() method present inside the context object.*



ADDING NEW BEANS PROGRAMMATICALLY

eazy bytes

```
if((randomNumber% 2) == 0){
    context.registerBean( beanName: "volkswagen",
            Vehicle.class,volkswagenSupplier);
}else{
    context.registerBean( beanName: "audi",
            Vehicle.class,audiSupplier);
}
```

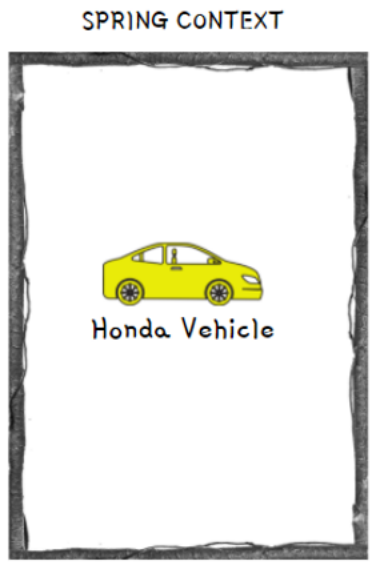


ADDING NEW BEANS USING XML CONFIGs

eazy bytes

- *In the initial versions of Spring, the bean and other configurations used to be done using XML. But over the time, Spring team brings annotation based configurations to make developers life easy. Today we can see XML configurations only in the older applications built based on initial versions of Spring.*
- *It is good to understand on how to create a bean inside Spring context using XML style configurations. So that, it will be useful if ever there is a scenario where you need to work in a project based on initial versions of Spring.*

```xml
<bean id="vehicle" class="com.example.beans.Vehicle">
    <property name="name" value="Honda" />
</bean>
```



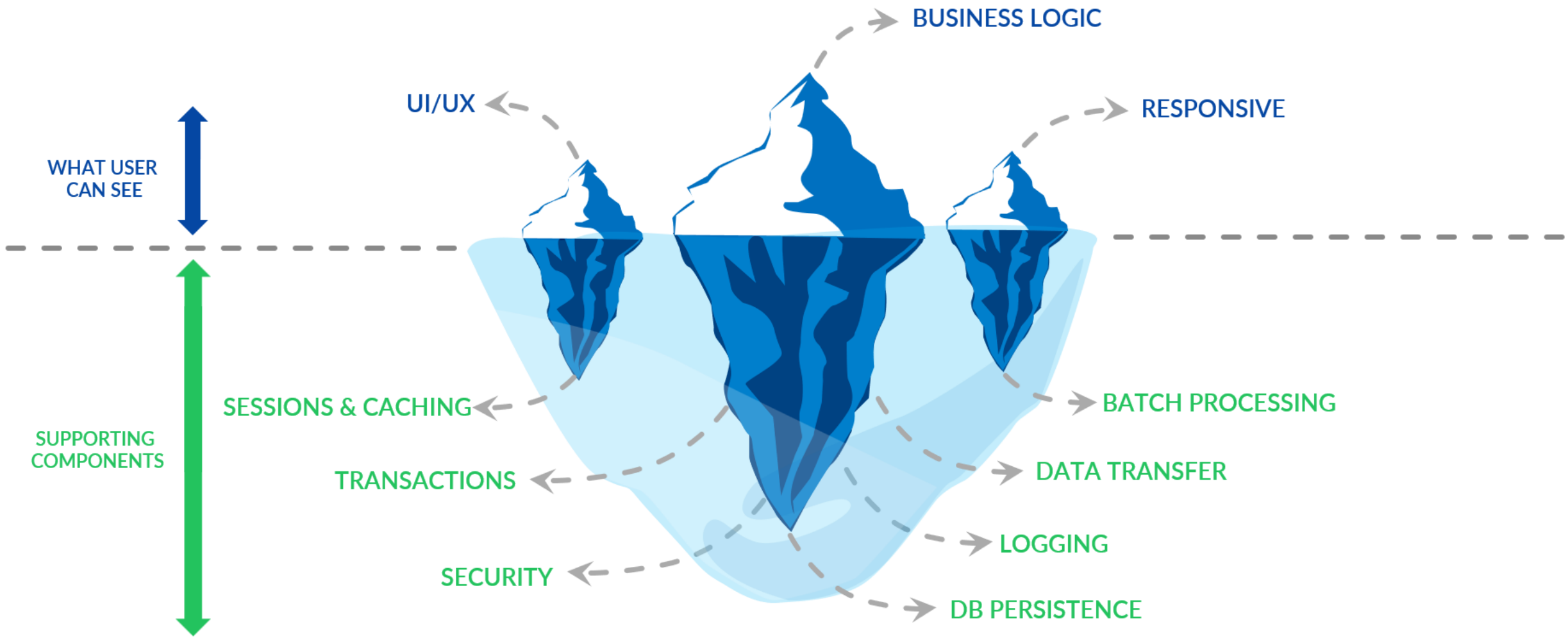

```java
var context = new ClassPathXmlApplicationContext( configLocation: "beans.xml");
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Vehicle name from Spring Context is: " + vehicle.getName());
```

Output on Console

```
Vehicle name from Spring Context is: Honda
```

BEHIND THE SCENES OF A WEB APP

eazy bytes



WHY SHOULD WE USE FRAMEWORKS?

eazy bytes

CHEF VICKY

Uses best readily available best ingredients like Cheese, Pizza Dough etc. to prepare Pizza

- Pizza preparation time is less
- Can easily scale his restaurant pizza orders
- Gets consistent taste for his pizzas
- Focus more on the pizza preparation
- Less efforts and more results/revenue

CHEF SANJEEV

Prepare all the ingredients like Cheese, Pizza Dough etc. by himself to prepare Pizza

- Pizza preparation time is more
- Scaling his restaurant pizza orders is not an option
- May not get a consistent taste for his pizzas
- Focus more on the raw material & ingredients
- More efforts and less results/revenue

WHY SHOULD WE USE FRAMEWORKS?

eazy bytes

DEV SANJEEV

Uses best readily available best frameworks like Spring, Angular etc. to build a web app

- Leverage Security, Logging etc. from frameworks
- Can easily scale his application
- App will work in an predictable manner
- Focus more on the business logic
- Less efforts and more results/revenue

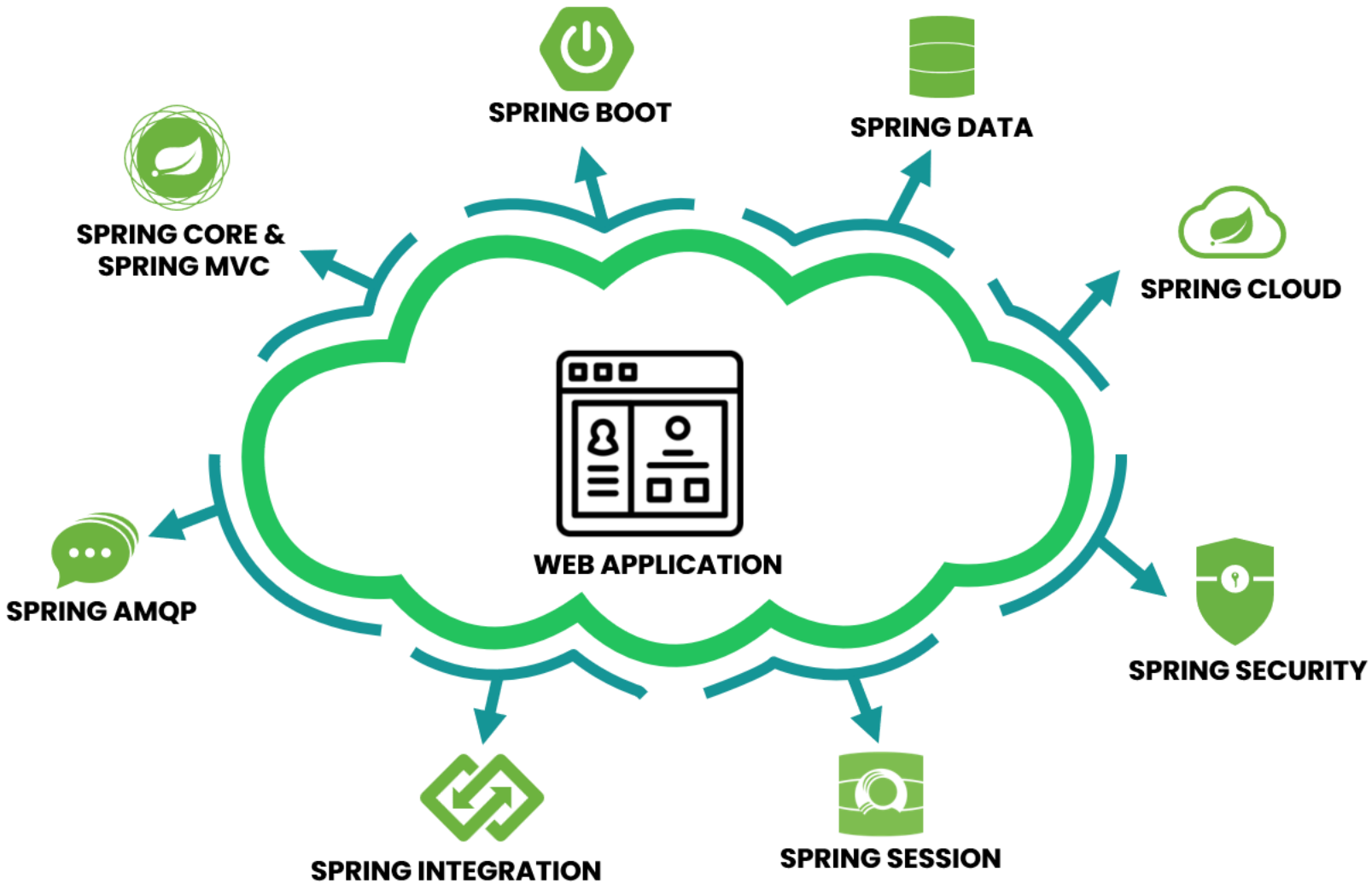
DEV VICKY

Build his own code by himself to build a web app

- Need to build code for Security, Logging etc.
- Scaling his is not an option till he test everything
- App may not work in an predictable manner
- Focus more on the supporting components
- More efforts and less results/revenue

SPRING PROJECTS

eazy bytes



* *These projects are few important projects from Spring but not a complete list.*

SPRING PROJECTS

eazy bytes

JSP/JSF | HTML/CSS | Servlets | JDBC | SOAP | J2EE

JSP/JSF | HTML/CSS | jQuery/ Bootstrap | SOAP/REST | MVC | ORM

Angular | React JS | HTML/CSS | REST | ORM/JPA | Cloud

Angular | React JS | HTML/CSS | REST | ORM/JPA | Microservices | Cloud | Docker/K8s

EVOLUTION OF APPLICATION DEVELOPMENT OVER YEARS

SPRING CORE & SPRING MVC

SPRING BOOT

SPRING DATA

SPRING SECURITY

INTEGRATION

SPRING CLOUD

Spring for Apache Kafka

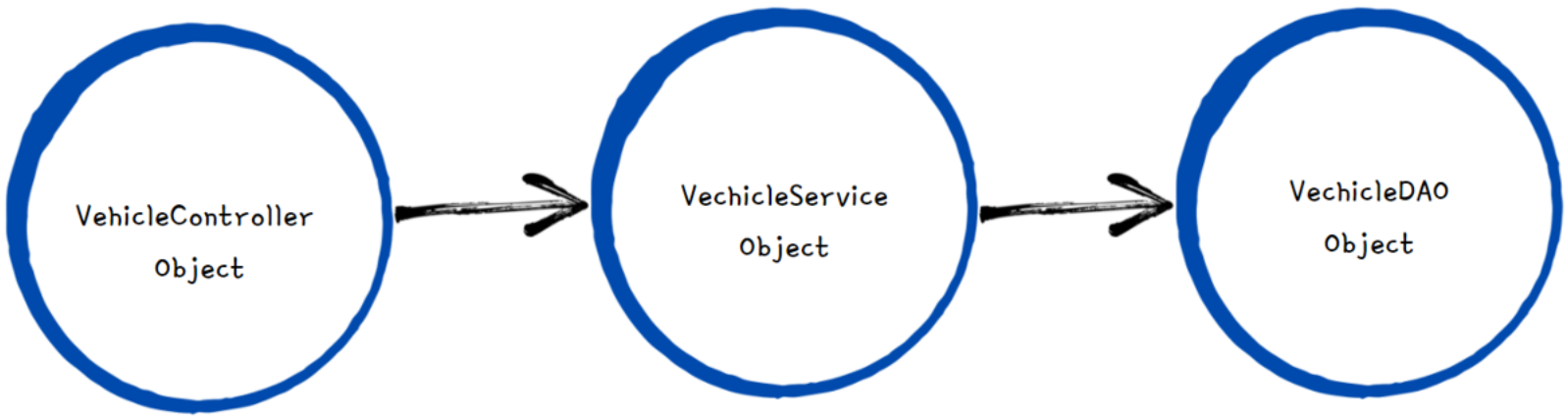
SPRING CLOUD DATA FLOW

EVOLUTION OF SPRING FRAMEWORK OVER YEARS

INTRODUCTION TO BEANS WIRING INSIDE SPRING

eazy bytes

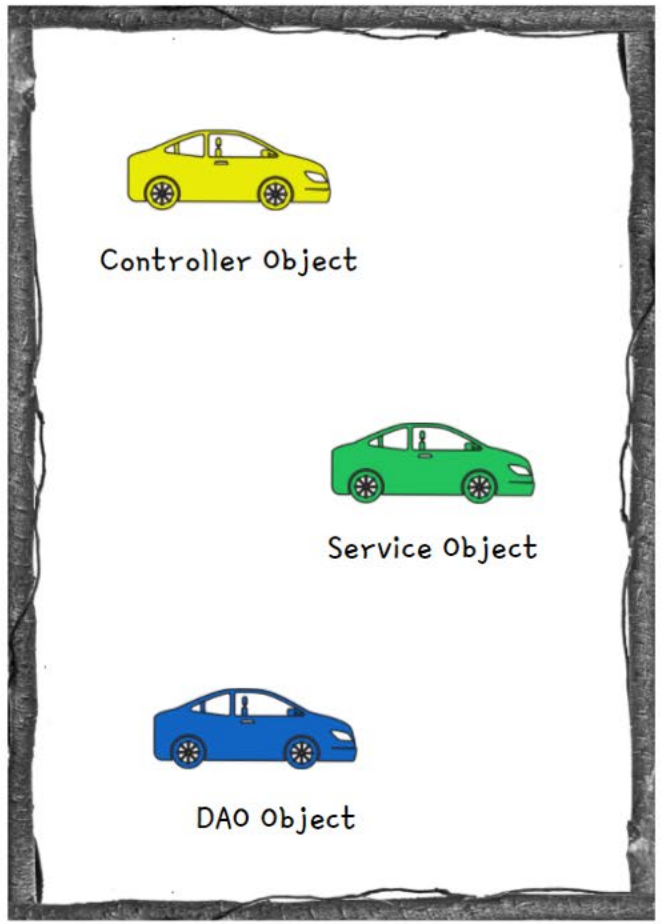
- *Inside Java web applications, usually the objects delegate certain responsibilities to other objects. So in this scenarios, objects will have dependency on others.*
- *In very similar lines when we create various beans using Spring, it our responsibility to understand the dependencies that beans have and wire them. This concept inside is called Wiring/Autowiring.*



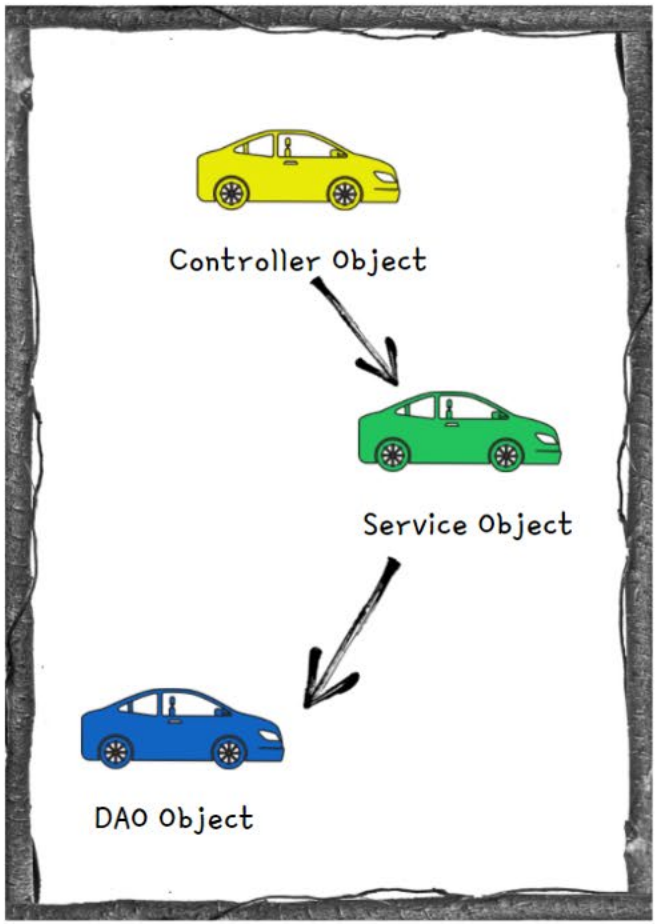
INTRODUCTION TO BEANS WIRING INSIDE SPRING

eazy bytes

SPRING CONTEXT WITH OUT WIRING



SPRING CONTEXT WITH WIRING & DI



NO WIRING SCENARIO INSIDE SPRING

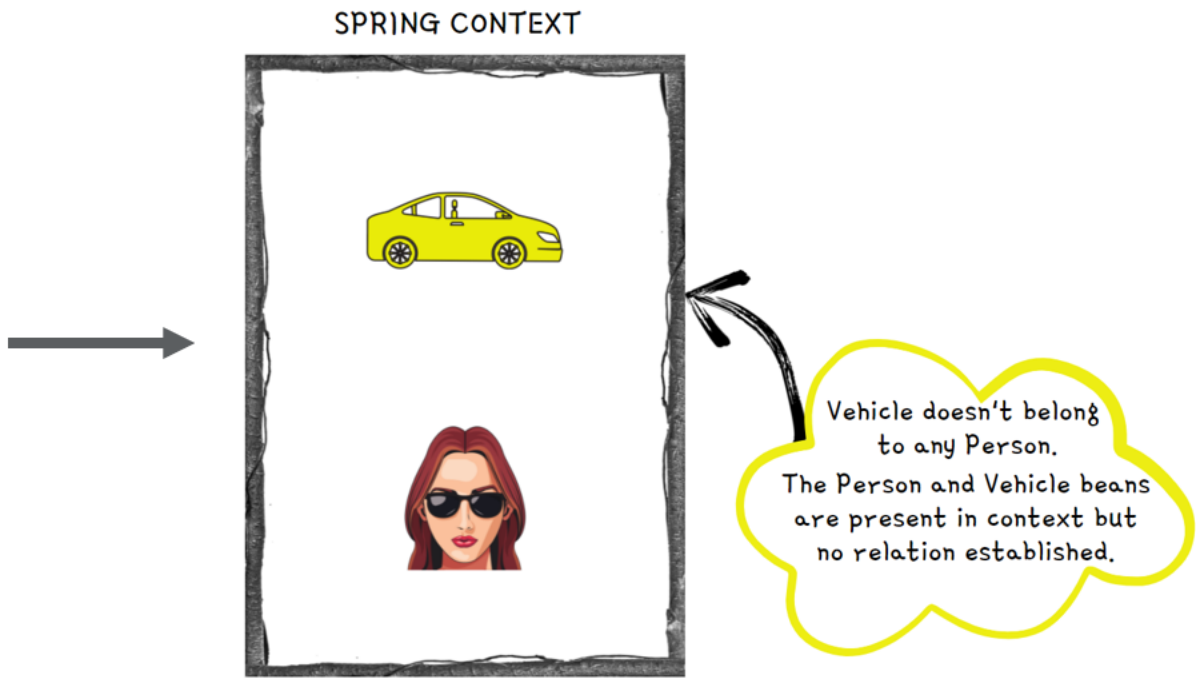
eazy bytes

Consider a scenario where we have two java classes Person and Vehicle. The Person class has a dependency on the Vehicle. Based on the below code, we are only creating the beans inside the Spring Context and no wiring will be done. Due to this both this beans present inside the Spring context with out knowing about each other.

```java
public class Vehicle {

    private String name;
```

```java
public class Person {

    private String name;
    private Vehicle vehicle;
```

```java
@Bean
public Vehicle vehicle() {
    Vehicle vehicle = new Vehicle();
    vehicle.setName("Toyota");
    return vehicle;
}

@Bean
public Person person() {
    Person person = new Person();
    person.setName("Lucy");
    return person;
}
```

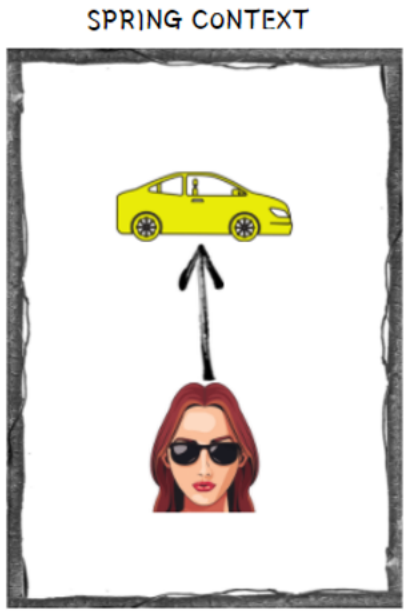


WIRING BEANS USING METHOD CALL

eazy bytes

- *Here in the below code, we are trying to wire or establish a relationship between Person and Vehicle, by invoking the vehicle() bean method from person() bean method. Now inside Sprint Context, person owns the vehicle.*
- *Spring will make sure to have only 1 vehicle bean is created and also vehicle bean will be created first always as person bean has dependency on it.*

```java
@Bean
public Vehicle vehicle() {
    Vehicle vehicle = new Vehicle();
    vehicle.setName("Toyota");
    return vehicle;
}

@Bean
public Person person() {
    Person person = new Person();
    person.setName("Lucy");
    person.setVehicle(vehicle());
    return person;
}
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Person person = context.getBean(Person.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Person name from Spring Context is: " + person.getName());
System.out.println("Vehicle name from Spring Context is: " + vehicle.getName());
System.out.println("Vehicle that Person own is: " + person.getVehicle());
```

Output on Console

```
Vehicle bean created by Spring
Person bean created by Spring
Person name from Spring Context is: Lucy
Vehicle name from Spring Context is: Toyota
Vehicle that Person own is: Vehicle name is - Toyota
```

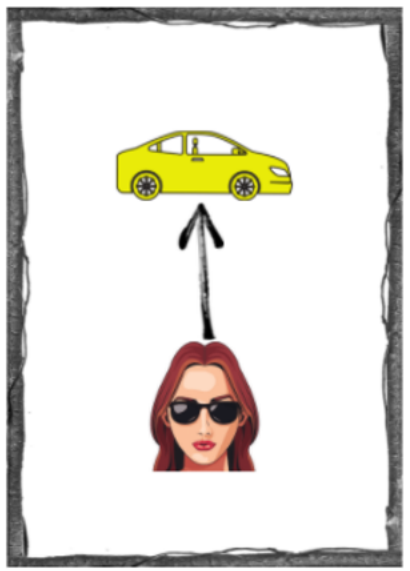
WIRING BEANS USING METHOD PARAMETERS

eazy bytes

- *Here in the below code, we are trying to wire or establish a relationship between Person and Vehicle, by passing the vehicle as a method parameter to the person() bean method. Now inside Sprint Context, person owns the vehicle.*
- *Spring injects the vehicle bean to the person bean using Dependency Injection*
- *Spring will make sure to have only 1 vehicle bean is created and also vehicle bean will be created first always as person bean has dependency on it.*

```java
@Bean
public Vehicle vehicle() {
    Vehicle vehicle = new Vehicle();
    vehicle.setName("Toyota");
    return vehicle;
}

/*...*/
@Bean
public Person person(Vehicle vehicle) {
    Person person = new Person();
    person.setName("Lucy");
    person.setVehicle(vehicle);
    return person;
}
```

SPRING CONTEXT



```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Person person = context.getBean(Person.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Person name from Spring Context is: " + person.getName());
System.out.println("Vehicle name from Spring Context is: " + vehicle.getName());
System.out.println("Vehicle that Person own is: " + person.getVehicle());
```

Output on Console

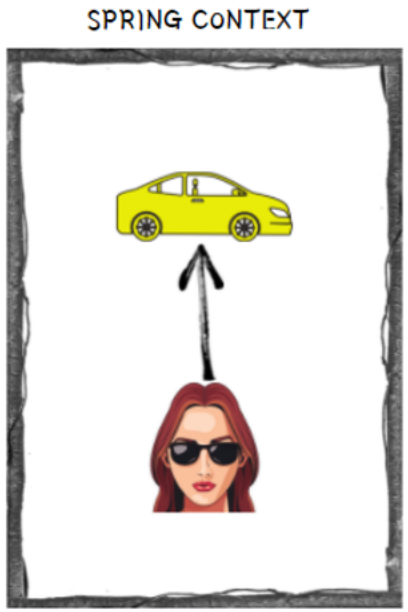
```
Vehicle bean created by Spring
Person bean created by Spring
Person name from Spring Context is: Lucy
Vehicle name from Spring Context is: Toyota
Vehicle that Person own is: Vehicle name is - Toyota
```

Inject Beans using @Autowired on class fields

eazy bytes

- *The @Autowired annotation marks on a field, setter method, constructor is used to auto-wire the beans that is 'injecting beans'(Objects) at runtime by Spring Dependency Injection mechanism.*
- *With the below code, Spring injects/auto-wire the vehicle bean to the person bean through a class field and dependency injection.*
- *The below style is not recommended for production usage as we can't mark the fields as final.*

```java
@Component
public class Person {

    private String name="Lucy";

    /*...*/
    @Autowired
    private Vehicle vehicle;
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Person person = context.getBean(Person.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Person name from Spring Context is: " + person.getName());
System.out.println("Vehicle name from Spring Context is: " + vehicle.getName());
System.out.println("Vehicle that Person own is: " + person.getVehicle());
```

Output on Console

```
Vehicle bean created by Spring
Person bean created by Spring
Person name from Spring Context is: Lucy
Vehicle name from Spring Context is: Toyota
Vehicle that Person own is: Vehicle name is - Toyota
```

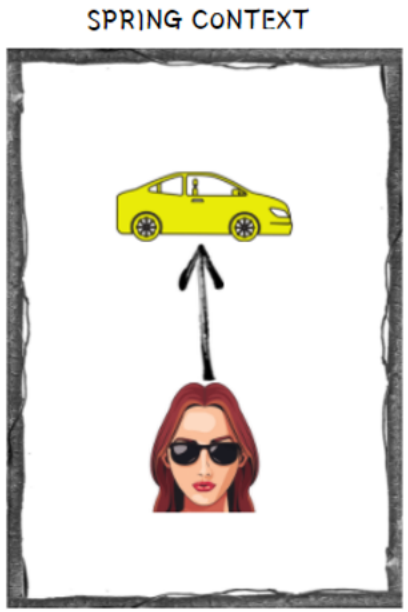
@Autowired(required = false) will help to avoid the NoSuchBeanDefinitionException if the bean is not available during Autowiring process.

Inject Beans using @Autowired on setter method

eazy bytes

- *The @Autowired annotation marks on a field, setter method, constructor is used to auto-wire the beans that is 'injecting beans'(Objects) at runtime by Spring Dependency Injection mechanism.*
- *With the below code, Spring injects/auto-wire the vehicle bean to the person bean through a setter method and dependency injection.*
- *The below style is not recommended for production usage as we can't mark the fields as final and not readable friendly.*

```java
@Component
public class Person {

    private String name="Lucy";
    private Vehicle vehicle;

    @Autowired
    public void setVehicle(Vehicle vehicle) {
        this.vehicle = vehicle;
    }
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Person person = context.getBean(Person.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Person name from Spring Context is: " + person.getName());
System.out.println("Vehicle name from Spring Context is: " + vehicle.getName());
System.out.println("Vehicle that Person own is: " + person.getVehicle());
```

Output on Console

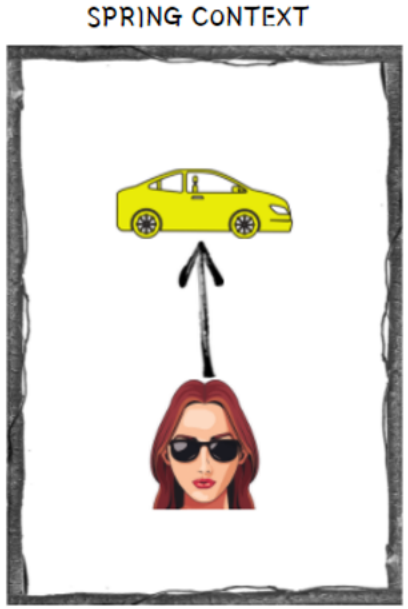
```
Vehicle bean created by Spring
Person bean created by Spring
Person name from Spring Context is: Lucy
Vehicle name from Spring Context is: Toyota
Vehicle that Person own is: Vehicle name is - Toyota
```

Inject Beans using @Autowired with constructor

eazy bytes

- *The @Autowired annotation marks on a field, setter method, constructor is used to auto-wire the beans that is 'injecting beans'(Objects) at runtime by Spring Dependency Injection mechanism.*
- *With the below code, Spring injects/auto-wire the vehicle bean to the person bean through a constructor and dependency injection.*
- *From Spring version 4.3, when we only have one constructor in the class, writing the @Autowired annotation is optional*

```java
@Component
public class Person {

    private String name="Lucy";
    private final Vehicle vehicle;

    @Autowired
    public Person(Vehicle vehicle){
        System.out.println("Person bean created by Spring");
        this.vehicle = vehicle;
    }
```




```java
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
Person person = context.getBean(Person.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Person name from Spring Context is: " + person.getName());
System.out.println("Vehicle name from Spring Context is: " + vehicle.getName());
System.out.println("Vehicle that Person own is: " + person.getVehicle());
```

Output on Console

```
Vehicle bean created by Spring
Person bean created by Spring
Person name from Spring Context is: Lucy
Vehicle name from Spring Context is: Toyota
Vehicle that Person own is: Vehicle name is - Toyota
```

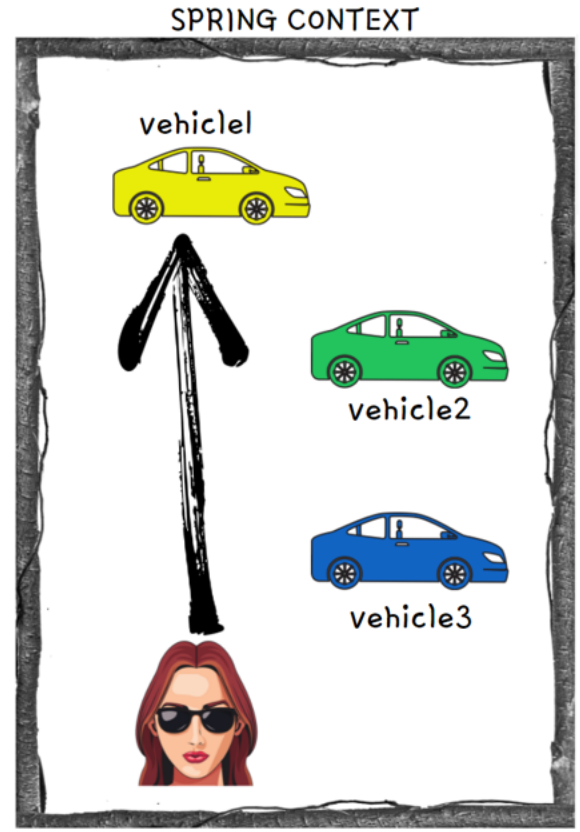
How Autowiring works with multiple Beans of same type

eazy bytes

- *By default Spring tries autowiring with class type. But this approach will fail if the same class type has multiple beans.*
- *If the Spring context has multiple beans of same class type like below, then Spring will try to auto-wire based on the parameter name/field name that we use while configuring autowiring annotation.*
- *In the below scenario, we used 'vehicle1' as constructor parameter. Spring will try to auto-wire with the bean which has same name like shown in the image below.*

```java
@Component
public class Person {

    private String name="Lucy";
    private final Vehicle vehicle;

    @Autowired
    public Person(Vehicle vehicle1){
        System.out.println("Person bean created by Spring");
        this.vehicle = vehicle1;
    }
```

STEP 1



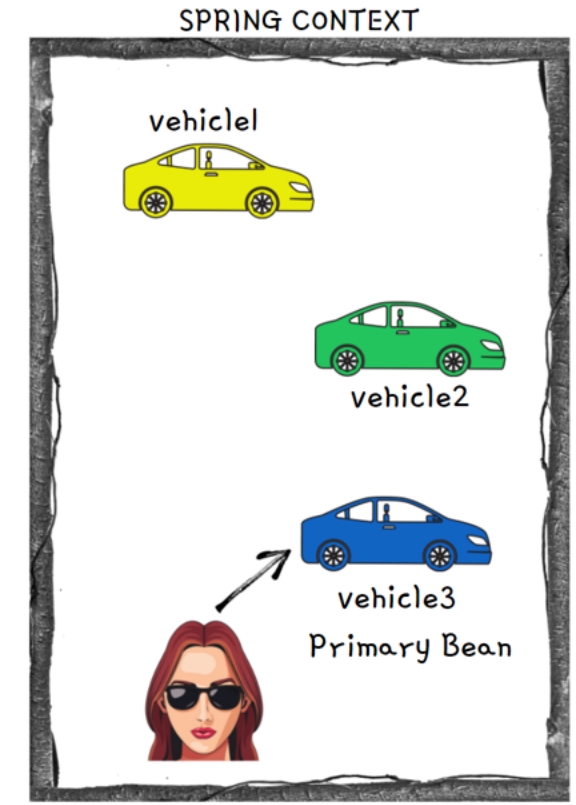
How Autowiring works with multiple Beans of same type

eazy bytes

- *If the parameter name/field name that we use while configuring autowiring annotation is not matching with any of the bean names, then Spring will look for the bean which has @Primary configured.*
- *In the below scenario, we used 'vehicle' as constructor parameter. Spring will try to auto-wire with the bean which has same name and since it can't find a bean with the same name, it will look for the bean with @Primary configured like shown in the image below.*

```java
@Component
public class Person {

    private String name="Lucy";
    private final Vehicle vehicle;

    @Autowired
    public Person(Vehicle vehicle){
        System.out.println("Person bean created by Spring");
        this.vehicle = vehicle;
    }
```

STEP 2

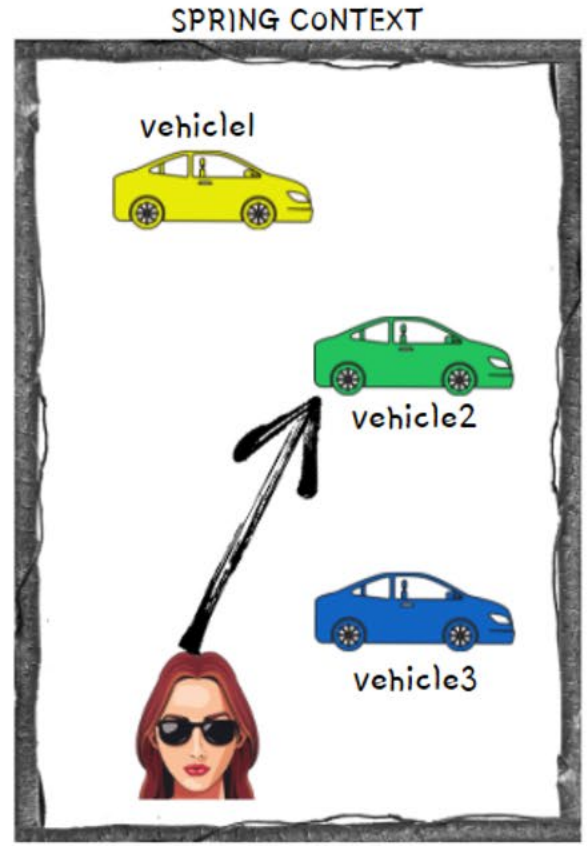


How Autowiring works with multiple Beans of same type

eazy bytes

- *If the parameter name/field name that we use while configuring autowiring annotation is not matching with any of the bean names and even Primary bean is not configured, then Spring will look if @Qualifier annotation is used with the bean name matching with Spring context bean names.*
- *In the below scenario, we used 'vehicle2' with @Qualifier annotation. Spring will try to auto-wire with the bean which has same name like shown in the image below.*

```
@Component
public class Person {

    private String name="Lucy";
    private final Vehicle vehicle;

    @Autowired
    public Person(@Qualifier("vehicle2") Vehicle vehicle){
        System.out.println("Person bean created by Spring");
        this.vehicle = vehicle;
    }
```



STEP 3

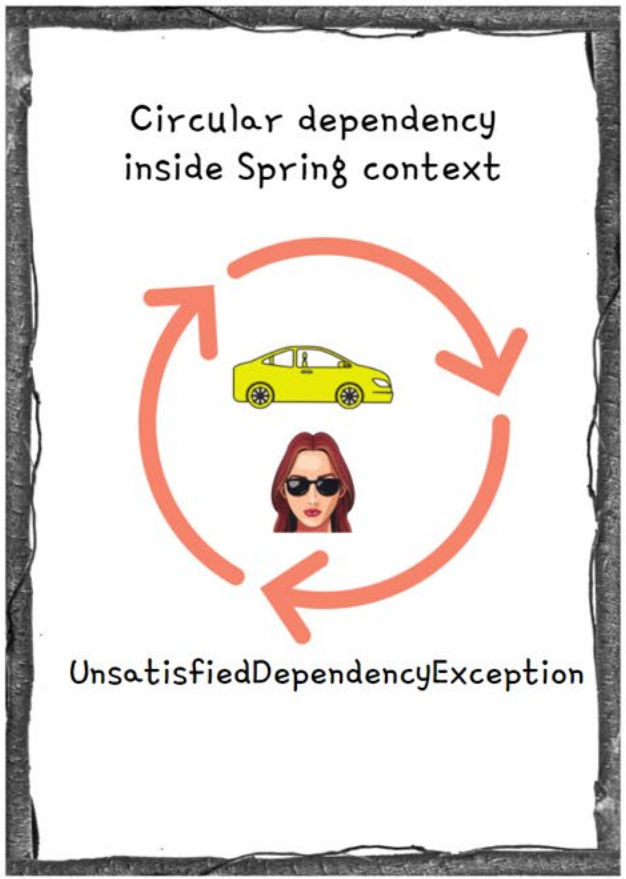
Understanding & Avoiding Circular dependencies

eazy bytes

- *A Circular dependency will happen if 2 beans are waiting for each to create inside the Spring context in order to do auto-wiring.*
- *Consider the below scenario, where Person has a dependency on Vehicle and Vehicle has a dependency on Person. In such scenarios, Spring will throw UnsatisfiedDependencyException due to circular reference.*
- *As a developer, it is our responsibility to make sure we are defining the configurations/dependencies that will result in circular dependencies.*

```java
@Component
public class Person {

    private String name="Lucy";
    private Vehicle vehicle;

    @Autowired
    public void setVehicle(Vehicle vehicle) {
        this.vehicle = vehicle;
    }
```

```java
public class Vehicle {

    private String name;
    @Autowired
    private Person person;
```



ASSIGNMENT RELATED TO BEANS, AUTOWIRING, DI

eazy bytes

Speakers interface with makeSound() method

SonySpeakers Bean implementation of Speakers

BoseSpeakers Bean implementation of Speakers

Vehicle Bean has a dependency on VehicleServices Bean, to play music and move the vehicle

VehicleServices bean depend on the implementations of Speakers and Tyres to serve vehicle bean requests.

Person Bean has a dependency on Vehicle Bean

Your application should play music from one of the Speakers implementations & move using one of the Tyres implementation. It should also give flexibility to switch between the implementations easily.

BridgeStoneTyres Bean implementation of Tyres

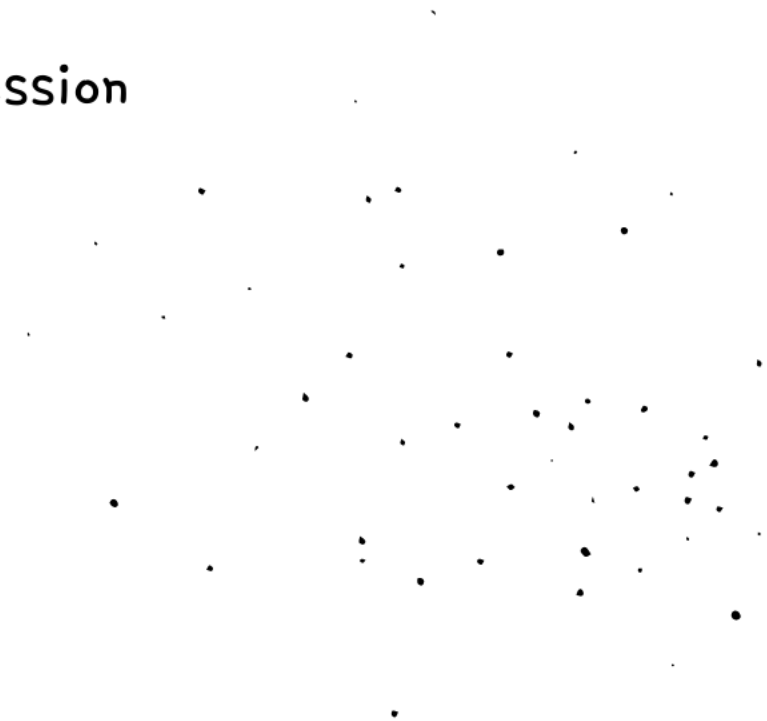
MichelinTyres Bean implementation of Tyres

Tyres interface with rotate() method

Bean Scopes inside Spring

eazy bytes

1. Singleton
2. Prototype
3. Request
4. Session
5. Application



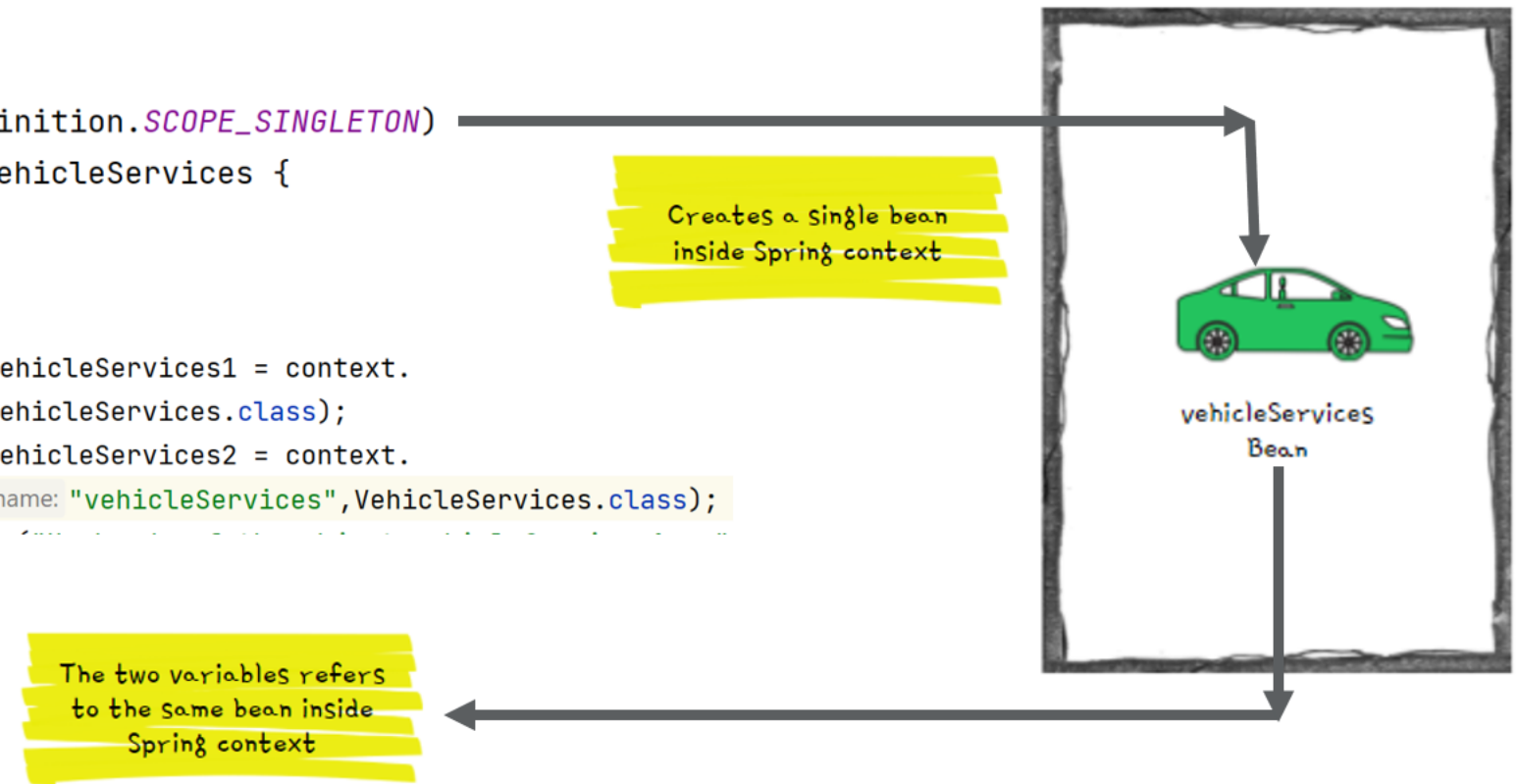
SINGLETON BEAN SCOPE

eazy bytes

- *Singleton is the default scope of a bean in Spring. In this scope, for a single bean we always get a same instance when you refer or autowire inside your application.*
- *Unlike Singleton design pattern where we have only 1 instance in entire app, inside Singleton scope Spring will make sure to have only 1 instance per unique bean. For example, if you have multiple beans of same type, then Spring Singleton scope will maintain 1 instance per each bean declared of same type.*

```
@Component
@Scope(BeanDefinition.SCOPE_SINGLETON)
public class VehicleServices {
```

```
VehicleServices vehicleServices1 = context.
        getBean(VehicleServices.class);
VehicleServices vehicleServices2 = context.
        getBean( name: "vehicleServices",VehicleServices.class);
```

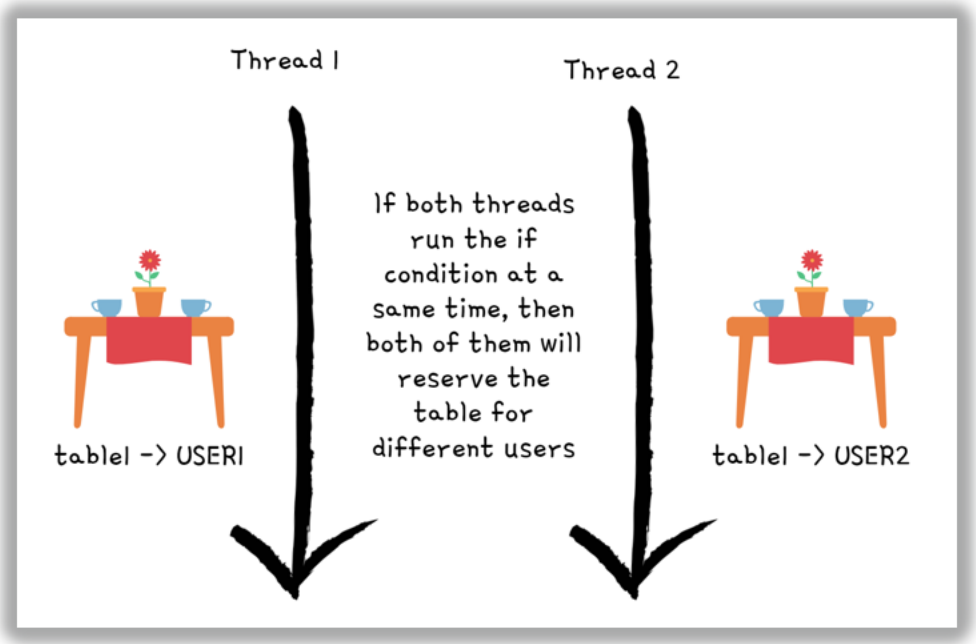


RACE CONDITION

eazy bytes

- *A race condition occurs when two threads access a shared variable at the same time. The first thread reads the variable, and the second thread reads the same value from the variable. Then the first thread and second thread perform their operations on the value, and they race to see which thread can write the value last to the shared variable. The value of the thread that writes its value last is preserved, because the thread is writing over the value that the previous thread wrote.*

```
// Shared Value inside a object
Map<String, String> reservedTables = new HashMap<>();

// thread -1
if (!reservedTables.containsKey("table1")){
    reservedTables.put("table1","USER1");
}
// thread - 2
if (!reservedTables.containsKey("table1")){
    reservedTables.put("table1","USER2");
}
```



USE CASES OF SINGLETON BEANS

eazy bytes

Singleton Beans use cases

- ✓ Since the same instance of singleton bean will be used by multiple threads inside your application, it is very important that these beans are immutable.
- ✓ This scope is more suitable for beans which handles service layer, repository layer business logics.

1. Building mutable singleton beans, will result in the race conditions inside multi thread environments.
2. There are ways to avoid race conditions due to mutable singleton beans with the help of synchronization.
3. But it is not recommended, since it brings lot of complexity and performance issues inside your app. So please don't try to build mutable singleton beans.

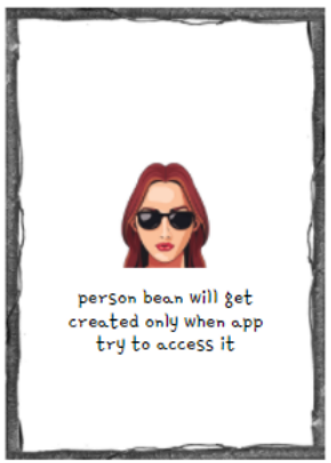
EAGER & LAZY INSTANTIATION

eazy bytes

- *By default Spring will create all the singleton beans eagerly during the startup of the application itself. This is called Eager instantiation.*
- *We can change the default behavior to initialize the singleton beans lazily only when the application is trying to refer to the bean. This approach is called Lazy instantiation.*

Sample demo of Lazy instantiation

```
@Component(value="personBean")
@Lazy
public class Person {
```



Output on Console

```
var context = new AnnotationConfigApplicationContext(ProjectConfig.class);
System.out.println("Before retrieving the Person bean from the Spring Context");
Person person = context.getBean(Person.class);
System.out.println("After retrieving the Person bean from the Spring Context");
```

```
Before retrieving the Person bean from the Spring Context
Person bean created by Spring
After retrieving the Person bean from the Spring Context
```

Eager Vs Lazy

eazy bytes

Eager instantiation

This is the default behavior inside Spring framework

The singleton bean will be created during the startup of the application

The server will not start if bean is not able to create due to any dependent exceptions

Spring context will occupy lot of memory if we try to use eager for all beans inside a application

Eager can be followed for all the beans which are required very commonly inside a application

Lazy instantiation

This is not a default behavior and need to configure explicitly using @Lazy

The singleton bean will be created when the app is trying to refer the bean for the first time

Application will throw an exception runtime if bean creation is failed due to any dependent exceptions

The performance will be impacted if we try to use lazy for all beans inside a application

Lazy can be followed for the beans that are used in a very remote scenario inside a application

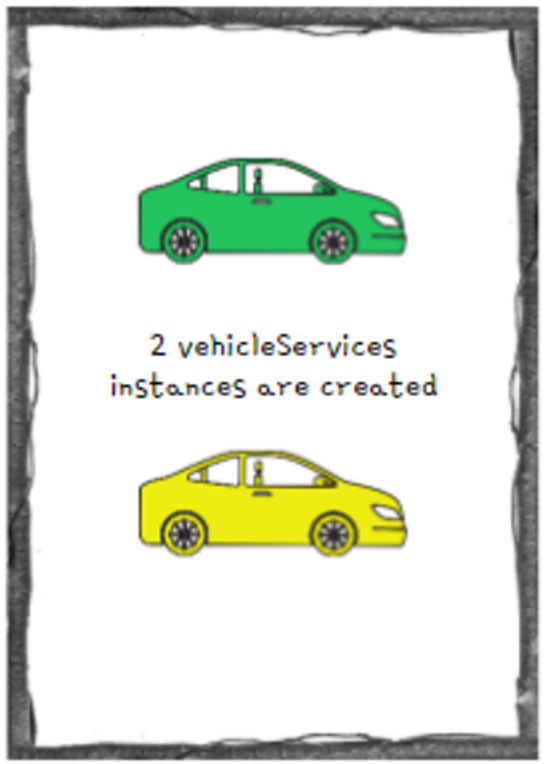
PROTOTYPE BEAN SCOPE

eazy bytes

- *With prototype scope, every time we request a reference of a bean, Spring will create a new object instance and provide the same.*
- *Prototype scope is rarely used inside the applications and we can use this scope only in the scenarios where your bean will frequently change the state of the data which will result race conditions inside multi thread environment. Using prototype scope will not create any race conditions.*

```
@Component
@Scope(BeanDefinition.SCOPE_PROTOTYPE)
public class VehicleServices {
```

```
VehicleServices vehicleServices1 = context.
        getBean(VehicleServices.class);
VehicleServices vehicleServices2 = context.
        getBean( name: "vehicleServices",VehicleServices.class);
```



Singleton Vs Prototype

eazy bytes

Singleton Scope

- This is the default scope inside Spring framework
- The same object instance will be returned every time we refer a bean inside the code
- We can configure to create the beans during the start up or when the first time referred
- Immutable objects can be idle for Singleton scope
- Most commonly used scope

Prototype Scope

- Need to explicitly configure using @Scope(BeanDefinition.SCOPE_PROTOTYPE)
- New object instance will be returned every time we refer a bean inside the code
- Spring always creates the new object when we try to refer the bean. No eager instantiation is possible
- Mutable objects can be idle for prototype scope
- Very rarely used scope

ASPECT-ORIENTED PROGRAMMING (AOP)

eazy bytes

Aspect-Oriented Programming

- ✓ **An aspect is simply a piece of code the Spring framework executes when you call specific methods inside your app.**
- ✓ **Spring AOP enables Aspect-Oriented Programming in spring applications. In AOP, aspects enable the modularization of concerns such as transaction management, logging or security that cut across multiple types and objects (often termed crosscutting concerns).**

1. AOP provides the way to dynamically add the cross-cutting concern before, after or around the actual logic using simple pluggable configurations.
2. AOP helps in separating and maintaining many non-business logic related code like logging, auditing, security, transaction management.
3. AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does this by adding additional behavior to existing code without modifying the code itself.

More details : https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#aop-ataspectj

ASPECT-ORIENTED PROGRAMMING (AOP)

eazy bytes

```java
public String moveVehicle(boolean started){
    Instant start = Instant.now();
    logger.info( msg: "method execution start");
    String status = null;
    if(started){
        status = tyres.rotate();
    }else{
        logger.log(Level.SEVERE, msg: "Vehicle not started to perform the" +
                " operation");
    }
    logger.info( msg: "method execution end");
    Instant finish = Instant.now();
    long timeElapsed = Duration.between(start, finish).toMillis();
    logger.info( msg: "Time took to execute the method : "+timeElapsed);
    return status;
}
```

There is so much of non business logic code along with the main business logic

AOP MAGIC

```java
public String moveVehicle(boolean started){
    return tyres.rotate();
}
```

With AOP magic, all the non business logic is moved to different location which will make method clean & clear

ASPECT-ORIENTED PROGRAMMING (AOP)

eazy bytes

AOP Jargons

- ✓ **When we define an Aspect or doing configurations, we need to follow WWW (3 Ws)**
  - **WHAT -> Aspect**
  - **WHEN -> Advice**
  - **WHICH -> Pointcut**

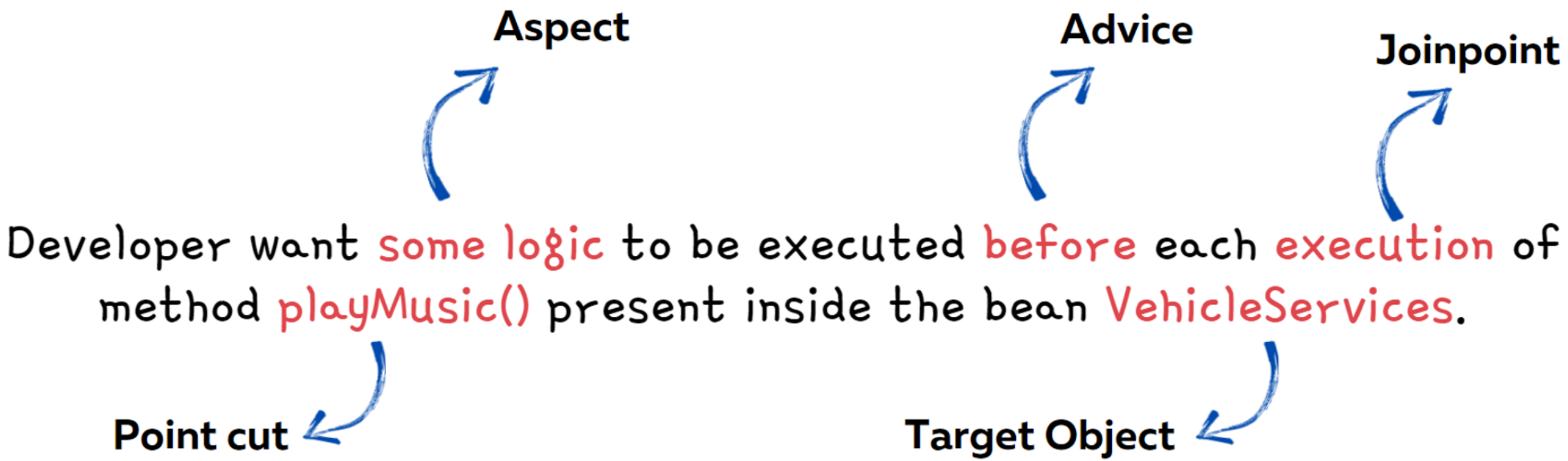
1. WHAT code or logic we want the Spring to execute when you call a specific method. This is called as Aspect.
2. WHEN the Spring need to execute the given Aspect. For example is it before or after the method call. This is called as Advice.
3. WHICH method inside App that framework needs to intercept and execute the given Aspect. This is called as a Pointcut.

- Join point which defines the event that triggers the execution of an aspect. Inside Spring, this event is always a method call.
- Target object is the bean that declares the method/pointcut which is intercepted by an aspect.

ASPECT-ORIENTED PROGRAMMING (AOP)

eazy bytes

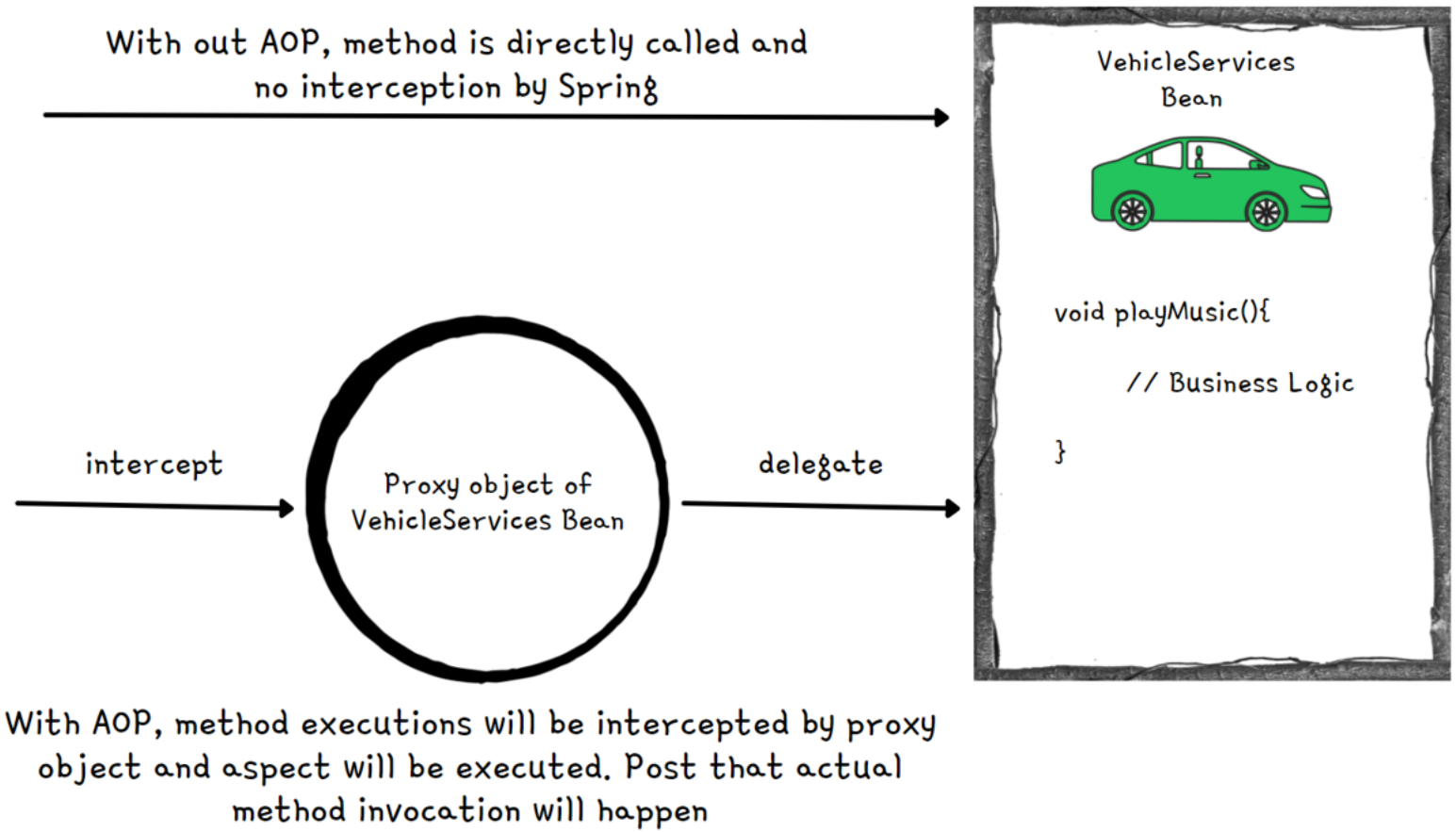
Typical Scenario of AOP implementation



WEAVING INSIDE AOP

eazy bytes

- *When we are implementing AOP inside our App using Spring framework, it will intercept each method call and apply the logic defined in the Aspect.*
- *But how does this works ? Spring does this with the help of proxy object. So we try to invoke a method inside a bean, Spring instead of directly giving reference of the bean instead it will give a proxy object that will manage the each call to a method and apply the aspect logic. This process is called Weaving.*



ADVICE TYPES INSIDE AOP

eazy bytes

Type of Advices in Spring AOP

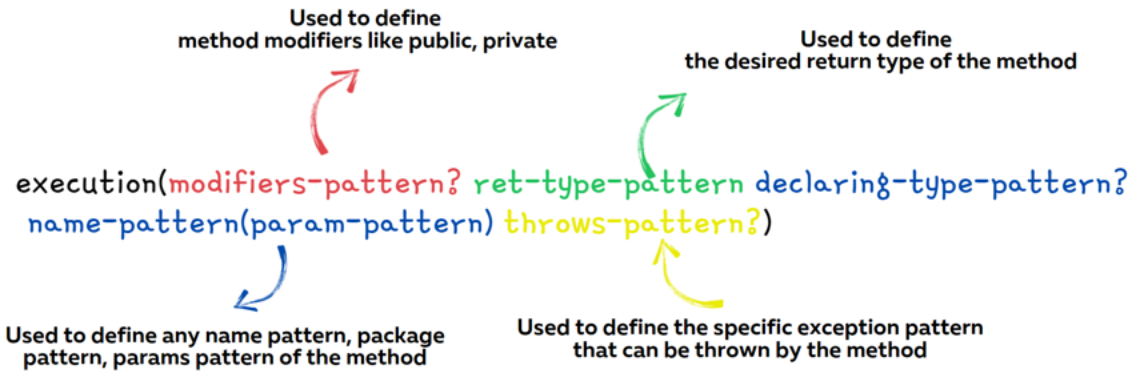
- ✓ @Before
- ✓ @AfterReturning
- ✓ @AfterThrowing
- ✓ @After
- ✓ @Around

1. Before advice runs before a matched method execution.
2. After returning advice runs when a matched method execution completes normally.
3. After throwing advice runs when a matched method execution exits by throwing an exception
4. After (finally) advice runs no matter how a matched method execution exits.
5. Around advice runs "around" a matched method execution. It has the opportunity to do work both before and after the method runs and to determine when, how, and even if the method actually gets to run at all.

CONFIGURING ADVICES INSIDE AOP

eazy bytes

- *We can use the AspectJ pointcut expression to provide details to Spring about what kind of methods it needs to intercept by mentioning details around modifier, return type, name pattern, package name pattern, params pattern, exceptions pattern etc.*
- *Below is the format of the same,*



Execution expression approach

- *Like shown below, we can mention the pointcut expressions as an input after advise annotations that we use,*

```
@Configuration
@ComponentScan(basePackages = {"com.example.implementation",
        "com.example.services", "com.example.aspects"})
@EnableAspectJAutoProxy
public class ProjectConfig {

}
```

```
@Aspect
@Component
public class LoggerAspect {

    @Around("execution(* com.example.services.*.*(..))")
    public void log(ProceedingJoinPoint joinPoint) throws Throwable {
        // Aspect Logic
    }

}
```

CONFIGURING ADVICES INSIDE AOP

eazy bytes

- *Alternatively, we can use Annotation style of configuring Advices inside AOP. Below are the three steps that we follow for the same,*

```java
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface LogAspect {
}
```

Step 1: Create an annotation type

Annotations approach

```java
@LogAspect
public String playMusic(boolean started, Song song){
    // Business Logic
}
```

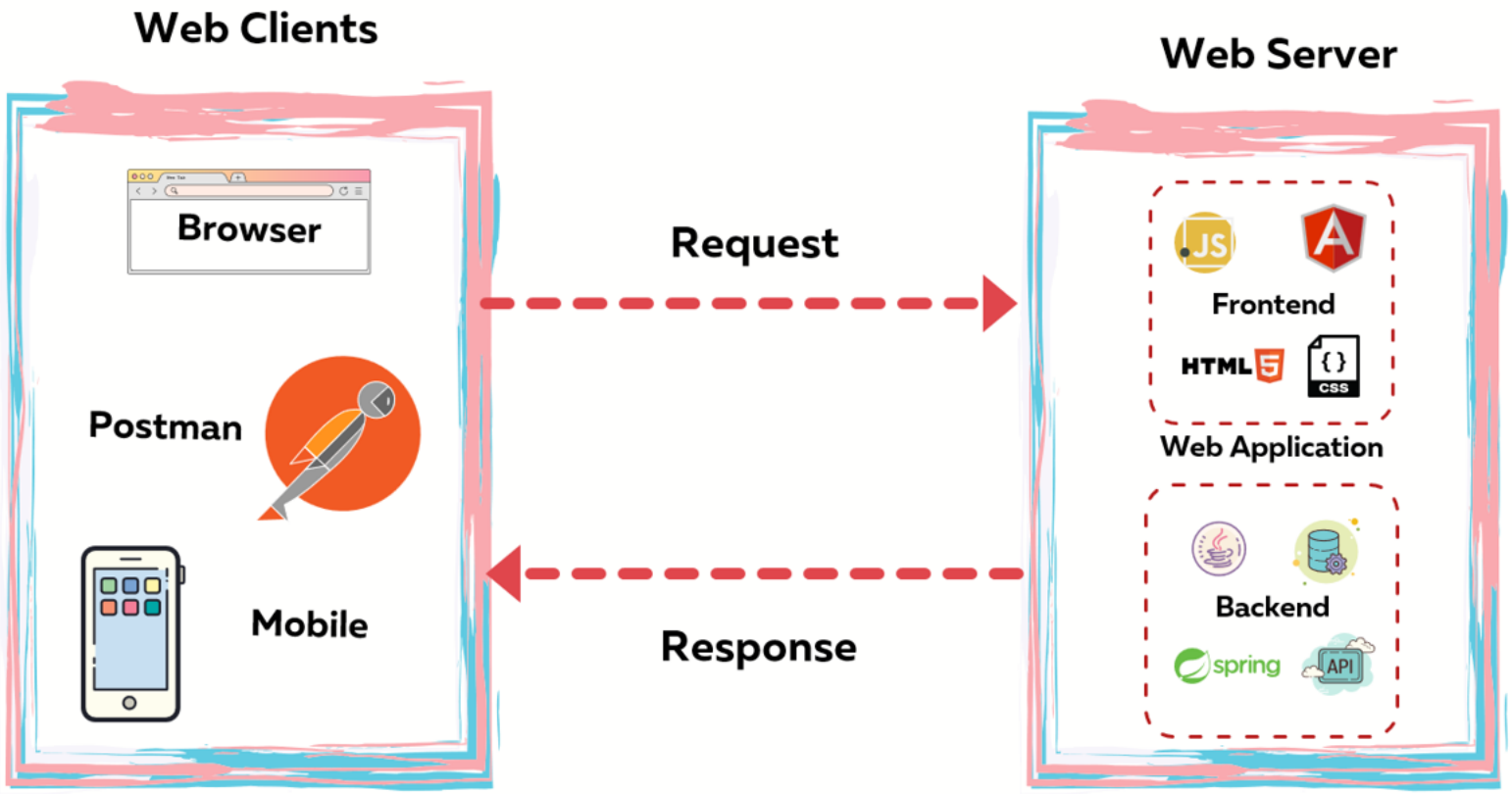
Step 2: Mention the same annotation on top of the method which we want to intercept using AOP

```java
@Around("@annotation(com.example.interfaces.LogAspect)")
public void logWithAnnotation(ProceedingJoinPoint joinPoint) throws Throwable {
    // Aspect Logic
}
```

Step 3: Use the annotation details to configure on top of the aspect method to advice

OVERVIEW OF A WEB APP

eazy bytes



1. The Web clients sends a request using protocols like HTTP to Web Application asking some data
like list of images, videos, text etc.

2. The web server where web app is deployed receives the client requests and process the data it receives. Post that it will respond to the client's request in the format of HTML, JSON etc.

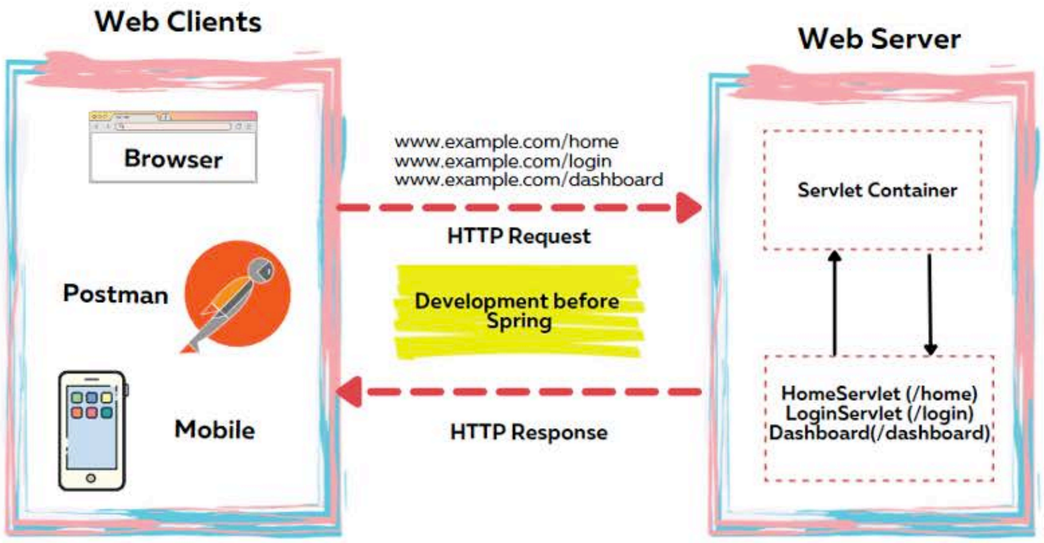
3. In Java web apps, Servlet Container (Web Server) takes care of translating the HTTP messages for Java code to understand. One of the mostly used servlet container is Apache Tomcat.

4. Servlet Container converts the HTTP messages into ServletRequest and hand over to Servlet method as a parameter. Similarly, ServletResponse returns as an output to Servlet Container from Servlet.

Usually Web Applications can be,
1) Only Frontend (Static Web Apps)
2) Only Backend (APIs)
3) Frontend + Backend (Ecommerce Apps)

ROLE OF SERVLETS INSIDE WEB APPs

eazy bytes



Before Spring

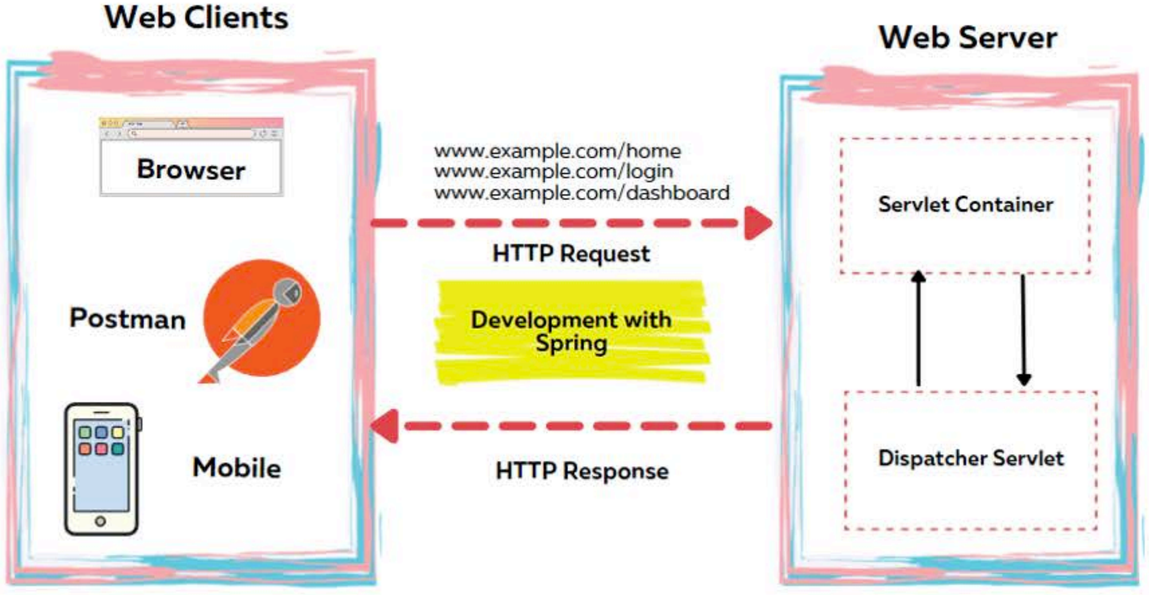
1. Before Spring, developer has to create a new servlet instance, configure it in the servlet container, and assign it to a specific URL path

2. When the client sends a request, Tomcat calls a method of the servlet associated with the path the client requested. The servlet gets the values on the request and builds the response that Tomcat sends back to the client.

With Spring

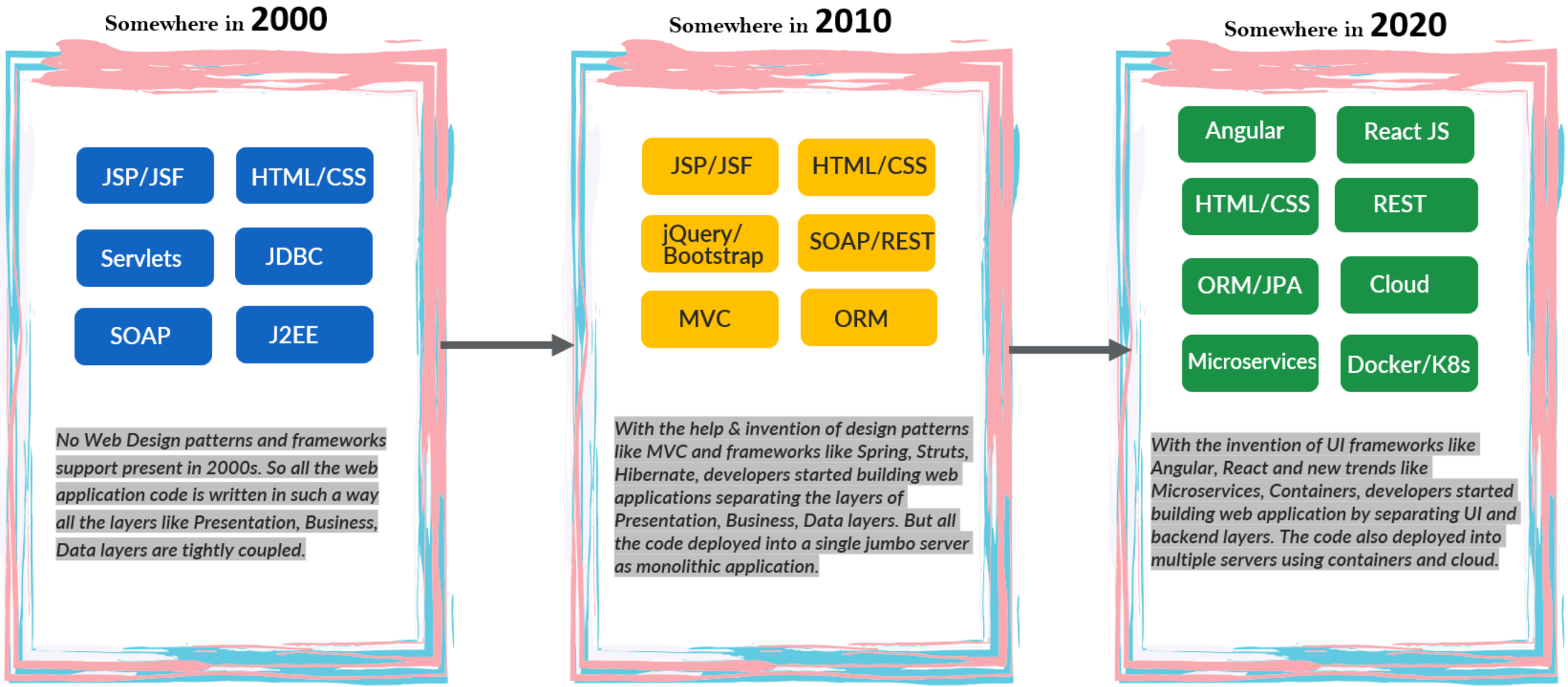
1. With Spring, it defines a servlet called Dispatcher Servlet which maintain all the URL mapping inside a web application.

2. The servlet container calls this Dispatcher Servlet for any client request, allowing the servlet to manage the request and the response. This way Spring internally does all the magic for Developers with out the need of defining the servlets inside a Web app.



EVOLUTION OF WEB APPs INSIDE JAVA ECO YSYTEM

eazy bytes

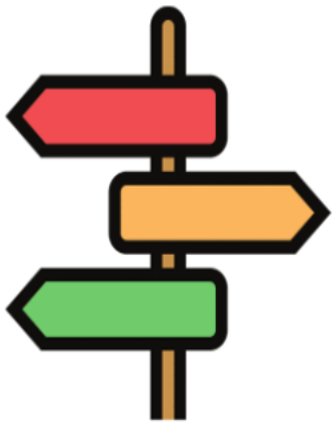


DEVELOPING WEB APPLICATIONS USING SPRING

eazy bytes

APPROACH 1

- Web Apps which holds UI elements like HTML, CSS, JS and backend logic
- Here the App is responsible to fully prepare the view along with data in response to a client request
- Spring Core, Spring MVC, SpringBoot, Spring Data, Spring Rest, Spring Security will be used



Approaches to build web applications using Spring framework

APPROACH 2

- Web Apps which holds only backend logic. These Apps send data like JSON to separate UI Apps built based on Angular, React etc.
- Here the App is responsible to only process the request and respond with only data ignoring view.
- Spring Core, SpringBoot, Spring Data, Spring Rest, Spring Security will be used

Spring MVC is the key differentiator between these two approaches.

SPRINGBOOT : THE HERO OF SPRING FRAMEWORK

eazy bytes

1. Spring Boot was introduced in April 2014 to reduce some of the burdens while developing a Java web application.

2. Before SpringBoot, Developer need to configure a servlet container, establish link b/w Tomcat and Dispatcher servlet, deploy into a server, define lot of dependencies.....

3. But with SpringBoot, we can create Web Apps skeletons with in seconds or at least in 1-2 mins ☺. It helps eliminating all the configurations we need to do.

4. Spring Boot is now one of the most appreciated projects in the Spring ecosystem. It helps us to create Spring apps more efficiently and focus on the business code.

5. SpringBoot is a mandatory skill now due to the latest trends like Full Stack Development, Microservices, Serverless, Containers, Docker etc.

BEFORE & AFTER SPRINGBOOT

eazy bytes

- Configure a Maven/Gradle project with all the dependencies needed
- Understand how servlets work & configure the DispatcherServlet inside web.xml
- Package the web application into a WAR file. Deploy the same into a server
- Deal with complicated class loading strategies, application monitoring and management

- Spring boot automatically configures the bare minimum components of a Spring application.
- Spring Boot applications embed a web server so that we do not require an external application server.
- Spring Boot provides several useful production-ready features out of the box to monitor and manage the application

eazy bytes

SpringBoot important features

SpringBoot Starters

Spring Boot groups related dependencies used for a specific purpose as starter projects. We don't need to figure out all the must-have dependencies you need to add to your project for one particular purpose nor which versions you should use for compatibility.

Example : spring-boot-starter-web

Autoconfiguration

Based on the dependencies present in the classpath, Spring Boot guess and auto configure the spring beans, property configurations etc. However, auto-configuration backs away from the default configuration if it detects user-configured beans with custom configurations

To achieve autoconfiguration SpringBoot follows the convention-over-configuration principle.

Actuator & DevTools

Spring Boot provides a pre-defined list of actuator endpoints. Using this production ready endpoints, we can monitor app health, metrics etc.

DevTools includes features such as automatic detection of application code changes, LiveReload server to automatically refresh any HTML changes to the browser all w/o server restart.

eazy bytes

Quick Recap

1. https://start.spring.io/ is the website which can be used to generate web projects skeleton based on the dependencies required for an application.

2. We can identify the Spring Boot main class by looking for an annotation @SpringBootApplication.

3. A single @SpringBootApplication annotation can be used to enable those three features, that is:

   - @EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism
   - @ComponentScan: enable @Component scan on the package where the application is located
   - @SpringBootConfiguration: enable registration of extra beans in the context or the import of additional configuration classes. An alternative to Spring's standard @Configuration annotation

eazy bytes

Quick Recap

4 The @RequestMapping annotation provides “routing” information. It tells Spring that any HTTP
request with the given path should be mapped to the corresponding method. It is a Spring MVC annotation and not specific to Spring Boot.

5 server.port and server.servlet.context-path properties can be mentioned inside the application.properties to change the default port number and context path of a web application.

6 Mentioning server.port=0 will start the web application at a random port number every time.

7 Mentioning debug=true will print the Autoconfiguration report on the console. We can mention the
exclusion list as well for SpringBoot auto-configuration by using the below config,

```
@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })
```

QUICK TIP

eazy bytes

Do you know, we can configure multiple paths against a single method using Spring MVC annotations ?

```
@Controller
public class HomeController {

    @RequestMapping(value={"", "/", "home"})
    public String displayHomePage(Model model) {
        // Business Logic
    }

}
```

INTRODUCTION TO THYMELEAF

eazy bytes

- *Thymeleaf is a modern server-side Java template engine for both web and standalone environments. This allow developers to build dynamic content inside the web applications.*
- *Thymeleaf has great integration with Spring especially with Spring MVC, Spring Security etc.*
- *The Thymeleaf + Spring integration packages offer a SpringResourceTemplateResolver implementation which uses all the Spring infrastructure for accessing and reading resources in applications, and which is the recommended implementation in Spring-enabled applications.*

```
<table>
    <thead>
    <tr>
        <th th:text="#{msgs.headers.name}">Name</th>
        <th th:text="#{msgs.headers.price}">Price</th>
    </tr>
    </thead>
    <tbody>
    <tr th:each="prod: ${allProducts}">
        <td th:text="${prod.name}">Oranges</td>
        <td th:text="${#numbers.formatDecimal(prod.price, 1, 2)}">0.99</td>
    </tr>
    </tbody>
</table>
```

Other famous template engines supported by Spring:

1. Jakarta Server Pages (JSP)
2. Jakarta Server Faces (JSF)
3. Apache FreeMarker
4. Groovy

For more details, pls refer https://www.thymeleaf.org/

SPRINGBOOT DEVTOOLS

eazy bytes

- *The Spring Boot DevTools provides features like Automatic restart & LiveReload that make the application development experience a little more pleasant for developers.*
- *It can be added into any of the SpringBoot project by adding the below maven dependency,*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

DevTools maintains 2 class loaders. one with classes that doesn't change and other one with classes that change. When restart needed it only reload the second class loader which makes restarts faster as well.

DevTools includes an embedded LiveReload server that can be used to trigger a browser refresh when a resource is changed. LiveReload related browser extensions are freely available for Chrome, Firefox

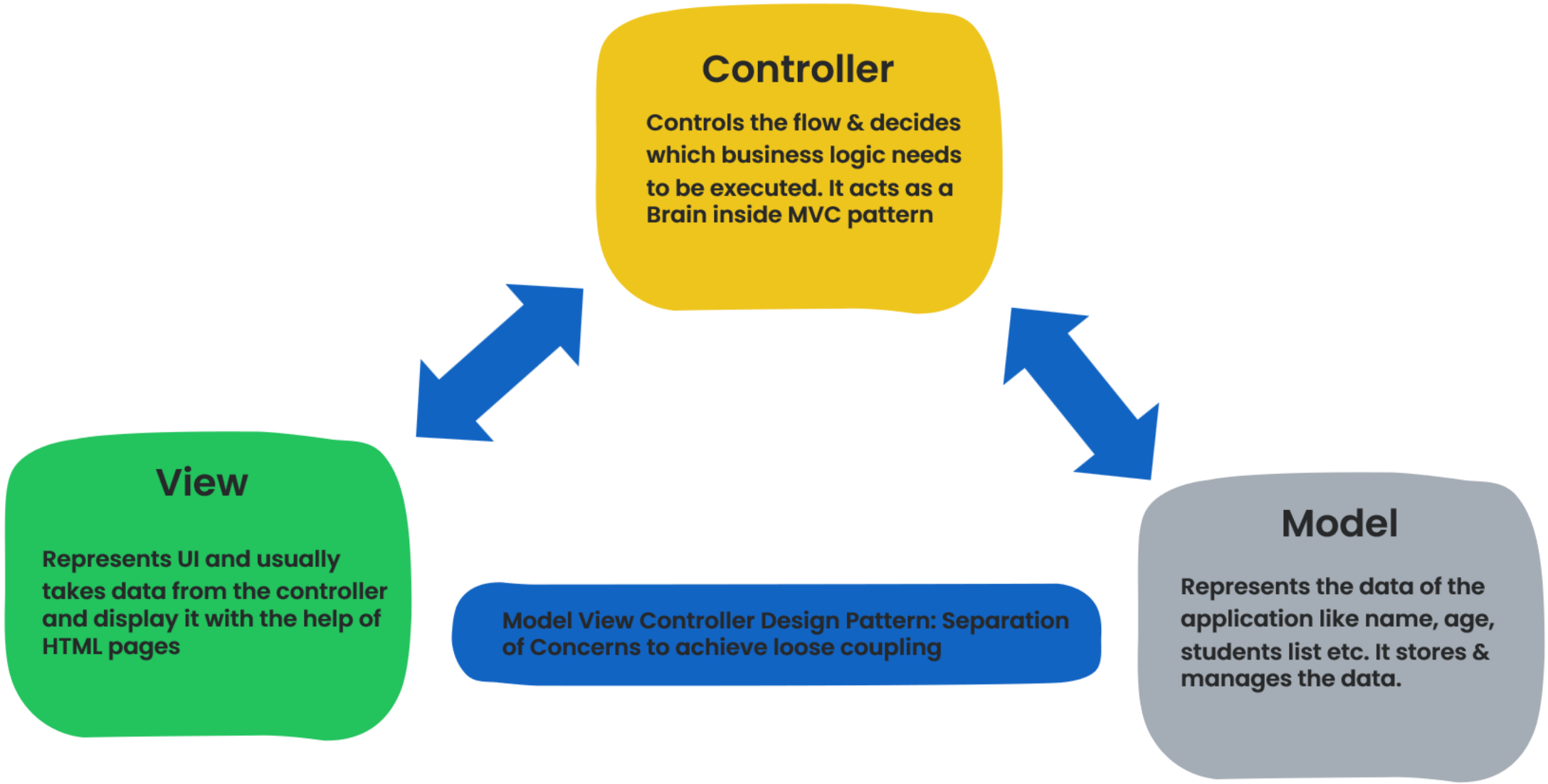
DevTools triggers a restart when ever a build is triggered through IDE or by maven commands etc.

DevTools disables the caching options by default during development.

Repackaged archives do not contain DevTools by default.

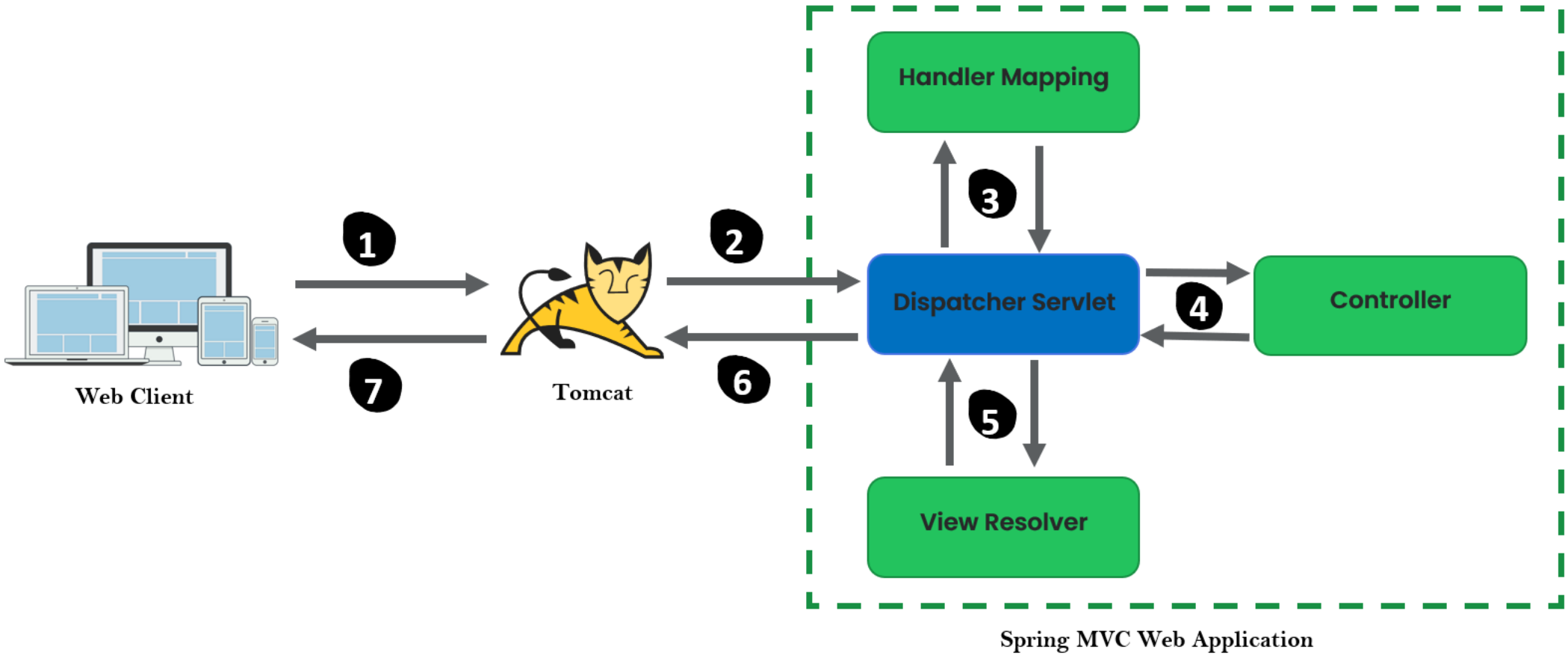
INTRODUCTION TO MVC PATTERN

eazy bytes



SPRING MVC ARCHITECTURE & INTERNAL FLOW

eazy bytes



SPRING MVC ARCHITECTURE & INTERNAL FLOW

eazy bytes

1. Web Client makes HTTP request

2. Servlet Containers like Tomcat accepts the HTTP requests and handovers the Servlet Request to Dispatcher Servlet inside Spring Web App.

3. The Dispatcher Servlet will check with the Handler Mapping to identify the controller and method names to invoke based on the HTTP method, path etc.

4. The Dispatcher Servlet will invoke the corresponding controller & method. After execution, the controller will provide a view name and data that needs to be rendered in the view.

5. The Dispatcher Servlet with the help of a component called View Resolver finds the view and render it with the data provided by the controller.

6. The Servlet Container or Tomcat accepts the Servlet Response from the Dispatcher servlet and convert the same to HTTP response before returning to the client.

7. The browser or client intercepts the HTTP response and display the view, data etc.

QUICK TIP

eazy bytes

Do you know, we can register view controllers that create a direct mapping between the URL and the view name using the ViewControllerRegistry. This way, there's no need for any Controller between the two.

```java
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController( urlPathOrPattern: "/courses").setViewName("courses");
        registry.addViewController( urlPathOrPattern: "/about").setViewName("about");
    }

}
```

REDUCING BOILERPLATE CODE WITH LOMBOK

eazy bytes

- *Java expects lot of boilerplate code inside POJO classes like getters and setters.*
- *Lombok, which is a Java library provides you with several annotations aimed at avoiding writing Java code known to be repetitive and/or boilerplate.*
- *It can be added into any of the Java project by adding the below maven dependency,*

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

Project Lombok works by plugging into your build process. Then, it will auto-generate the Java bytecode into your .class files required to implement the desired behavior, based on the annotations you used.

Most commonly used Lombok annotations,

@Getter, @Setter
@NoArgsConstructor
@RequiredArgsConstructor
@AllArgsConstructor
@ToString, @EqualsAndHashCode
@Data

@Data is a shortcut annotation that combines the features of below annotations together,

@ToString
@EqualsAndHashCode
@Getter, @Setter
@RequiredArgsConstructor

REDUCING BOILERPLATE CODE WITH LOMBOK

eazy bytes

```java
/*
@Data annotation is provided by Lombok library which generates getter, setter,
equals(), hashCode(), toString() methods & Constructor at compile time.
This makes our code short and clean.
* */
@Data
public class Contact {

    private String name;
    private String mobileNum;
    private String email;
    private String subject;
    private String message;
}
```

Contact.java

Inherited members (Ctrl+F12) Anonymous Classes (Ctrl+I) Lambdas (Ctrl+L)

- Contact
  - Contact()
  - canEqual(Object): boolean
  - equals(Object): boolean ↑Object
  - getEmail(): String
  - getMessage(): String
  - getMobileNum(): String
  - getName(): String
  - getSubject(): String
  - hashCode(): int ↑Object
  - setEmail(String): void
  - setMessage(String): void
  - setMobileNum(String): void
  - setName(String): void
  - setSubject(String): void
  - toString(): String ↑Object
  - email: String
  - message: String
  - mobileNum: String
  - name: String
  - subject: String

A sample outline of a Java POJO class with @Data annotation used. The source code will not have boilerplate code but the compiled byte code will have.

@RequestParam Annotation

eazy bytes

- *In Spring @RequestParam annotation is used to map either query parameters or form data.*
- *For example, if we want to get parameters value from a HTTP GET requested URL then we can use @RequestParam annotation like in below example.*

http://localhost:8080/holidays?festival=true&federal=true

```
@GetMapping("/holidays")
public String displayHolidays(@RequestParam(required = false) boolean festival,
                              @RequestParam(required = false) boolean federal) {
    // Business Logic
    return "holidays.html";
}
```

✓ The @RequestParam annotation supports attributes like name, required, value, defaultvalue. We can use them in our application based on the requirements.

✓ The name attribute indicates the name of the request parameter to bind to.

✓ The required attribute is used to make a field either optional or mandatory. If it is mandatory, an exception will throw in case of missing fields.

✓ The value attribute is similar to name elements and can be used as an alias.

✓ defaultValue for the parameter is to handle missing values or null values. If the parameter does not contain any value then this default value will be considered.

@PathVariable Annotation

eazy bytes

- *The @PathVariable annotation is used to extract the value from the URI. It is most suitable for the RESTful web service where the URL contains some value. Spring MVC allows us to use multiple @PathVariable annotations in the same method.*
- *For example, if we want to get the value from a requested URI path, then we can use @PathVariable annotation like in below example.*

http://localhost:8080/holidays/all
http://localhost:8080/holidays/federal
http://localhost:8080/holidays/festival

```
@GetMapping("/holidays/{display}")
public String displayHolidays(@PathVariable String display) {
    //Business Logic
    return "holidays.html";
}
```

✓ **The @PathVariable annotation supports attributes like name, required, value similar to @RequestParam. We can use them in our application based on the requirements.**

VALIDATION WITH SPRING BOOT

eazy bytes

- *Bean Validation (https://beanvalidation.org/) is the standard for implementing validations in the Java ecosystem. It's well integrated with Spring and Spring Boot.*
- *Below is the maven dependency that we can add to implement Bean validations in any Spring/SpringBoot project,*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Bean Validation works by defining constraints to the fields of a class by annotating them with certain annotations.

We can put the @Valid annotation on method parameters and fields to tell Spring that we want a method parameter or field to be validated.

Below are the important packages where validations related annotations can be identified,

- ✓ jakarta.validation.constraints.*
- ✓ org.hibernate.validator.constraints.*

eazy bytes

Important Validation Annotations

jakarta.validation.constraints.*

- ✓ @Digits
- ✓ @Email
- ✓ @Max
- ✓ @Min
- ✓ @NotBlank
- ✓ @NotEmpty
- ✓ @NotNull
- ✓ @Pattern
- ✓ @Size

org.hibernate.validator.constraints.*

- ✓ @CreditCardNumber
- ✓ @Length
- ✓ @Currency
- ✓ @Range
- ✓ @URL
- ✓ @UniqueElements
- ✓ @EAN
- ✓ @ISBN

VALIDATION WITH SPRING BOOT

eazy bytes

- *Sample validations declaration inside a java POJO class,*

```java
@Data
public class Contact {

    @NotBlank(message="Email must not be blank")
    @Email(message = "Please provide a valid email address" )
    private String email;

    @NotBlank(message="Subject must not be blank")
    @Size(min=5, message="Subject must be at least 5 characters long")
    private String subject;

    @NotBlank(message="Message must not be blank")
    @Size(min=10, message="Message must be at least 10 characters long")
    private String message;
}
```

VALIDATION WITH SPRING BOOT

eazy bytes

- *We can put the @Valid annotation on method parameters to tell Spring framework that we want a particular POJO object needs to be validated based on the validation annotation configurations. For any issues, framework populates the error details inside the Errors object. The errors can be used to display on the UI to the user. Sample example code is below,*

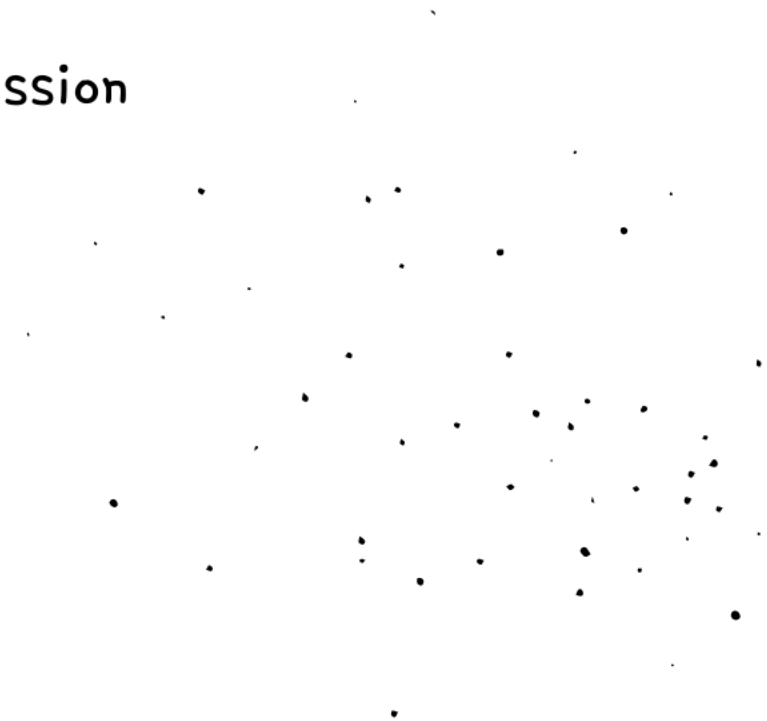
```
@RequestMapping(value = "/saveMsg",method = POST)
public String saveMessage(@Valid @ModelAttribute("contact") Contact contact, Errors errors) {

    if(errors.hasErrors()){
        log.error("Contact form validation failed due to : " + errors.toString());
        return "contact.html";
    }
    contactService.saveMessageDetails(contact);
    return "redirect:/contact";
}
```

```
<ul>
    <li class="alert alert-danger" role="alert" th:each="error : ${#fields.errors('contact.*')}"
        th:text="${error}" />
</ul>
```

eazy bytes

Bean Scopes inside Spring

1. Singleton
2. Prototype
3. Request
4. Session
5. Application



SPRING WEB SCOPES

eazy bytes

Web Scopes inside Spring

- ✓ **Request (@RequestScope)**
- ✓ **Session (@SessionScope)**
- ✓ **Application (@ApplicationScope)**

1. **Request Scope** - Spring creates an instance of the bean class for every HTTP request. The instance exists only for that specific HTTP request.
2. **Session Scope** - Spring creates an instance and keeps the instance in the server's memory for the full HTTP session. Spring links the instance in the context with the client's session.
3. **Application Scope** - The instance is unique in the app's context, and it's available while the app is running.

eazy bytes

Key points of Spring Web scopes

Request Scope

- ✓ Spring creates a lot of instances of this bean in the app's memory for each HTTP request. So these type of beans are short lived.
- ✓ Since Spring creates a lot of instances, please make sure to avoid time consuming logic while creating the instance.
- ✓ Can be considered for the scenarios where the data needs to be reset after new request or page refresh etc..

Session Scope

- ✓ Session scoped beans have longer life & they are less frequently garbage collected.
- ✓ Avoid keeping too much information inside session data as it impacts performance. Never store sensitive information as well.
- ✓ Can be considered for the scenarios where the same data needs to be accessed across multiple pages like user information.

Application Scope

- ✓ In the application scope, Spring creates a bean instance per web application runtime.
- ✓ It is similar to singleton scope, with one major difference. Singleton scoped bean is singleton per ApplicationContext where application scoped bean is singleton per ServletContext.
- ✓ Can be considered for the scenarios where we want to store Drop Down values, Reference table values which won't change for all the users.

SPRING SECURITY

eazy bytes

- *Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.*
- *Below is the maven dependency that we can add to implement security using Spring Security project in any of the SpringBoot projects,*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Spring Security is a framework that provides authentication, authorization, and protection against common attacks.

Spring Security helps developers with easier configurations to secure a web application by using standard username/password authentication mechanism.

Spring Security provides out of the box features to handle common security attacks like CSRF, CORs. It also has good integration with security standards like JWT, OAUTH2 etc.

AUTHENTICATION & AUTHORIZATION

eazy bytes

AUTHENTICATION

- In authentication, the identity of users are checked for providing the access to the system.
- Authentication done before authorization
- It needs usually user's login details
- If authentication fails usually we will get 401 error response
- For example as a Bank customer/employee in order to perform actions in the app, we need to prove our identity

AUTHORIZATION

- In authorization, person's or user's authorities are checked for accessing the resources.
- Authorization always happens after authentication.
- it needs user's privilege or roles
- If authorization fails usually we will get 403 error response
- Once logged into the application, my roles, authorities will decide what kind of actions I can do

QUICK TIP

eazy bytes

Do you know,

- ✓ **As soon as we add spring security dependency to a web application, be default it protects all the pages/APIs inside it. It will redirect to the inbuilt login page to enter credentials.**
- ✓ **The default credentials are user and password is randomly generated & printed on the console.**
- ✓ **We can configure custom credentials using the below properties to get started for POCs etc. But for PROD applications, Spring Security supports user credentials configuration inside DB, LDAP, OAUTH2 Server etc.**

```
spring.security.user.name = eazybytes
spring.security.user.password = 12345
```

DEFAULT SECURITY CONFIGURATIONS IN SPRING SECURITY

eazy bytes

By default, Spring Security framework protects all the paths present inside the web application. This behaviour is due to the code present inside the method defaultSecurityFilterChain(HttpSecurity http) of class SpringBootWebSecurityConfiguration

```
@Bean
@Order(SecurityProperties.BASIC_AUTH_ORDER)
SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
    http.authorizeHttpRequests((requests) -> requests.anyRequest().authenticated());
    http.formLogin(withDefaults());
    http.httpBasic(withDefaults());
    return http.build();
}
```

CONFIGURE permitAll() WITH SPRING SECURITY

eazy bytes

- *Using permitAll() configurations we can allow full/public access to a specific resource/path or all the resources/paths inside a web application.*
- *Below is the sample configuration that we can do in order to allow any requests in a Web application with out security,*

```
@Configuration
public class ProjectSecurityConfig {

  @Bean
  SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests((requests) -> requests.anyRequest().permitAll())
        .formLogin(Customizer.withDefaults())
        .httpBasic(Customizer.withDefaults());
    return http.build();

  }
```

- Form Login provides support for username and password being provided through an html form
- HTTP Basic Auth uses an HTTP header in order to provide the username and password when making a request to a server.

CONFIGURE denyAll() WITH SPRING SECURITY

eazy bytes

- *Using denyAll() configurations we can deny access to a specific resource/path or all the resources/paths inside a web application regardless of user authentication.*
- *Below is the sample configuration that we can do in order to deny any requests that is coming into a web application,*

```
@Configuration
public class ProjectSecurityConfig {

  @Bean
  SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {

    http.authorizeHttpRequests((requests) -> requests.anyRequest().denyAll())
        .formLogin(Customizer.withDefaults())
        .httpBasic(Customizer.withDefaults());
    return http.build();

  }
```

- Usually denyAll() is used to retire a specific API temporarily with out removing the code
- permitAll() is used to allow public access to public APIs, paths, CSS, images, JS files etc.

CONFIGURE CUSTOM SECURITY CONFIGs & CSRF DISABLE

eazy bytes

- *We can apply custom security configurations based on our requirements for each API/URL like below.*
- *permitAll() can be used to allow access w/o security and authenticated() can be used to protect a web page/API.*
- *By default any requests with HTTP methods that can update data like POST, PUT will be stopped with 403 error due to CSRF protection. We can disable the same for now and enable it in the coming sections when we started generating CSRF tokens.*
- *Below is the sample configuration that we can do to implement custom security configs and disable CSRF,*

```
@Configuration
public class ProjectSecurityConfig {

   @Bean
   SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
      http.csrf((csrf) -> csrf.disable())
            .authorizeHttpRequests((requests) -> requests
               .requestMatchers("", "/", "/home").permitAll()
               .requestMatchers("/holidays/**").permitAll()
               .requestMatchers("/contact").permitAll()
               .requestMatchers("/saveMsg").permitAll()
               .requestMatchers("/courses").permitAll()
               .requestMatchers("/about").permitAll()
               .requestMatchers("/assets/**").permitAll())
            .formLogin(Customizer.withDefaults())
            .httpBasic(Customizer.withDefaults());
      return http.build();
   }
```

IN-MEMORY AUTHENTICATION IN SPRING SECURITY

eazy bytes

- *Spring Security provide support for username/password based authentication based on the users stored in application memory.*
- *Like mentioned below, we can configure any number of users & their roles, passwords using in-memory authentication,*

```java
@Configuration
public class ProjectSecurityConfig {

  @Bean
  SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {...}

  @Bean
  public InMemoryUserDetailsManager userDetailsService() {

    UserDetails user = User.withDefaultPasswordEncoder()
        .username("user").password("12345").roles("USER").build();

    UserDetails admin = User.withDefaultPasswordEncoder()
        .username("admin").password("54321").roles("USER", "ADMIN").build();
    return new InMemoryUserDetailsManager(user, admin);
  }

}
```

in-memory authentication is idle for POC Web Apps or any internal Web App that get used only in non-prod environments. NEVER EVER use in-memory authentication for PROD web applications.

CONFIGURING LOGIN & LOGOUT PAGE

eazy bytes

- *Spring Security allows us to configure a custom login page to our web application instead of using the Spring Security default provided login page.*
- *Similarly we can configure logout page as well.*
- *Below is the sample configuration that we can follow,*

```
@Bean
SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
    http.csrf((csrf) -> csrf.disable())
            .authorizeHttpRequests((requests) -> requests.requestMatchers("/dashboard").authenticated()
            .requestMatchers("", "/", "/home").permitAll()
            .requestMatchers("/holidays/**").permitAll()
            .requestMatchers("/contact").permitAll()
            .requestMatchers("/saveMsg").permitAll()
            .requestMatchers("/courses").permitAll()
            .requestMatchers("/about").permitAll()
            .requestMatchers("/login").permitAll()
            .requestMatchers("/assets/**").permitAll())
            .formLogin(loginConfigurer -> loginConfigurer.loginPage("/login")
                    .defaultSuccessUrl("/dashboard").failureUrl("/login?error=true").permitAll())
            .logout(logoutConfigurer -> logoutConfigurer.logoutSuccessUrl("/login?logout=true")
                    .invalidateHttpSession(true).permitAll())
            .httpBasic(Customizer.withDefaults());
    return http.build();
}
```

Configured login page will be shown, if the user tries to access the secured page/resource with out a valid authenticated session. The same behavior applies for default login page provided by Spring Security

QUICK TIP

eazy bytes

Do you know, Thymeleaf has a great integration with Spring Security. More details can be found at https://www.thymeleaf.org/doc/articles/springsecurity.html

Step 1 : Add the below dependency in the pom.xml

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
```

Step 2 : Add the below XML name space which enable us to use Thymeleaf Security related tags

```
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">
```

Step 3 : Use any of the Thymeleaf Security tags inside HTML code based on Authentication details,

sec:authorize="isAnonymous()" , sec:authorize="isAuthenticated()" , sec:authorize="hasRole('ROLE_ADMIN')"
sec:authentication="name" , sec:authentication="principal.authorities"

@ControllerAdvice & @ExceptionHandler

eazy bytes

- *@ControllerAdvice is a specialization of the @Component annotation which allows to handle exceptions across the whole application in one global handling component. You can think of it as an interceptor of exceptions thrown by methods annotated with @RequestMapping or one of the shortcuts like @GetMapping etc.*
- *We can define the exception handle logic inside a method and annotate it with @ExceptionHandler*
- *Below is the sample configuration that we can follow,*

```java
@ControllerAdvice
public class GlobalExceptionController {

    @ExceptionHandler(Exception.class)
    public ModelAndView exceptionHandler(Exception exception){
        ModelAndView errorPage = new ModelAndView();
        errorPage.setViewName("error");
        errorPage.addObject("errormsg", exception.getMessage());
        return errorPage;
    }

}
```

Combination of @ControllerAdvice & @ExceptionHandler can handle the exceptions across all the controllers inside a web application globally.

QUICK TIP

eazy bytes

Do you know,

- ✓ If a method annotated with `@ExceptionHandler` present inside a `@Controller` class, then the exception handling logic will be applicable for any exceptions occurred in that specific controller class.
- ✓ If the same `@ExceptionHandler` annotated method present inside a `@ControllerAdvice` class, then the exception handling logic will be applicable for any exceptions occurred across all the controller classes.
- ✓ Using `@ExceptionHandler` annotation, we can handle any number of exceptions like below sample code,

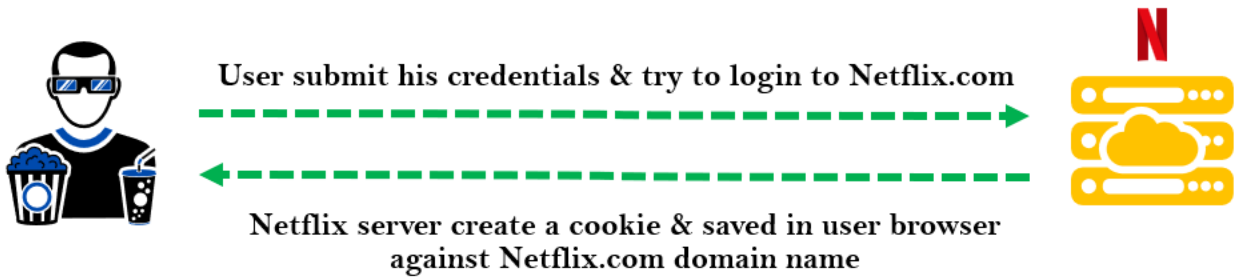
```
@ExceptionHandler({NullPointerException.class,
                   ArrayIndexOutOfBoundsException.class,
                   IOException.class})
public ModelAndView handleException(RuntimeException ex)
{
    //Exception handling logic
}
```

CROSS-SITE REQUEST FORGERY (CSRF)

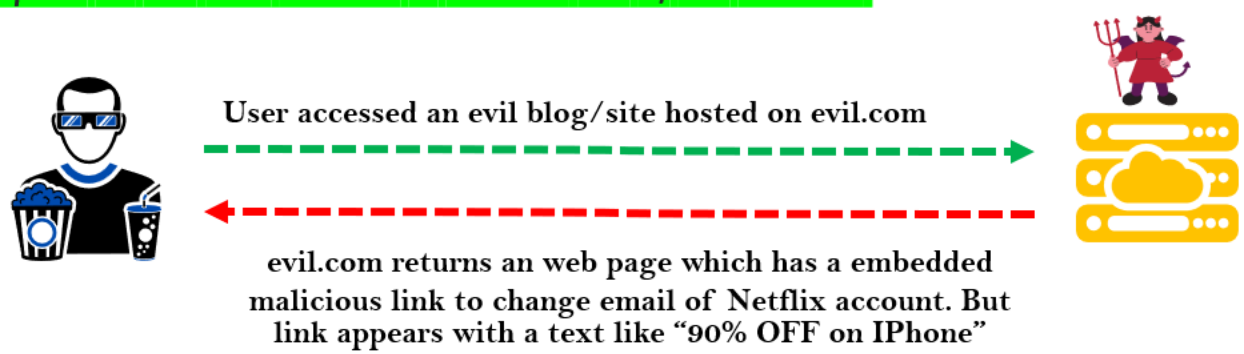
eazy bytes

- *A typical Cross-Site Request Forgery (CSRF or XSRF) attack aims to perform an operation in a web application on behalf of a user without their explicit consent. In general, it doesn't directly steal the user's identity, but it exploits the user to carry out an action without their will.*
- *Consider you are using a website netflix.com and the attacker's website evil.com.*

Step 1 : The Netflix user login to Netflix.com and the backend server of Netflix will provide a cookie which will store in the browser against the domain name Netflix.com



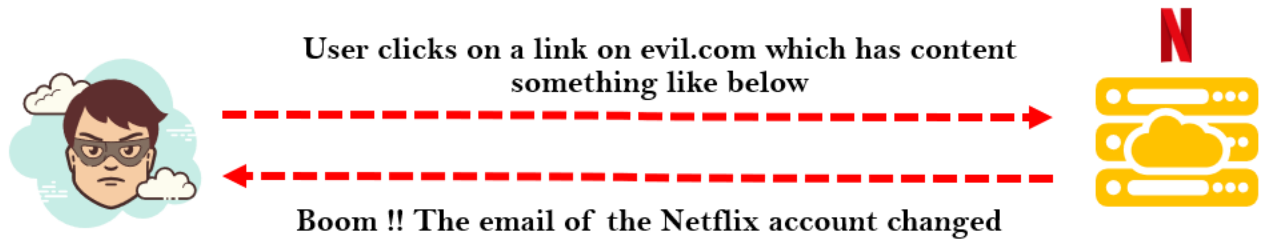
Step 2 : The same Netflix user opens an evil.com website in another tab of the browser.



CROSS-SITE REQUEST FORGERY (CSRF)

eazy bytes

Step 3 : User tempted and clicked on the malicious link which makes a request to Netflix.com. And since the login cookie already present in the same browser and the request to change email is being made to the same domain Netflix.com, the backend server of Netflix.com can't differentiate from where the request came. So here the evil.com forged the request as if it is coming from a Netflix.com UI page.




```
<form action="https://netflix.com/changeEmail"
  method="POST" id="form">
    <input type="hidden" name="email" value="user@evil.com">
<form>

<script>
    document.getElementById('form').submit()
</script>
```

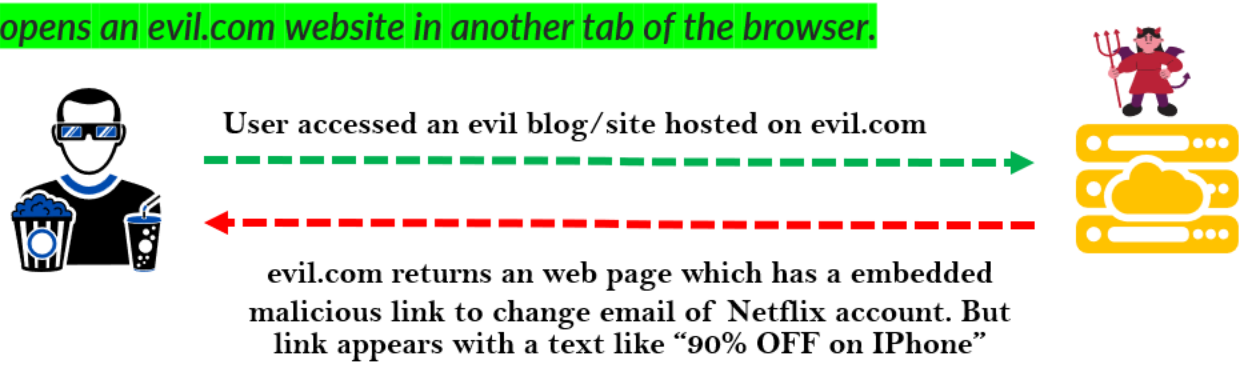
SOLUTION TO CSRF

eazy bytes

- *To defeat a CSRF attack, applications need a way to determine if the HTTP request is legitimately generated via the application's user interface. The best way to achieve this is through a CSRF token. A CSRF token is a secure random token that is used to prevent CSRF attacks. The token needs to be unique per user session and should be of large random value to make it difficult to guess.*
- *Let's see how this solve CSRF attack by taking the previous Netflix example again,*

Step 1 : The Netflix user login to Netflix.com and the backend server of Netflix will provide a cookie which will store in the browser against the domain name Netflix.com along with a randomly generated unique CSRF token for this particular user session. CSRF token is inserted within hidden parameters of HTML forms to avoid exposure to session cookies.

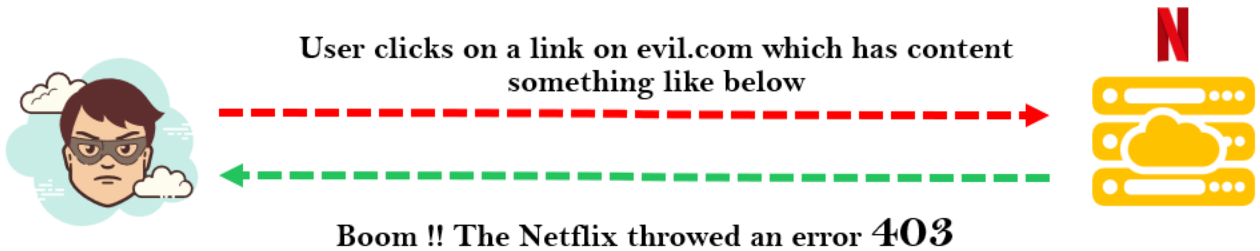
Step 2 : The same Netflix user opens an evil.com website in another tab of the browser.



SOLUTION TO CSRF

eazy bytes

Step 3 : User tempted and clicked on the malicious link which makes a request to Netflix.com. And since the login cookie already present in the same browser and the request to change email is being made to the same domain Netflix.com. This time the Netflix.com backend server expects CSRF token along with the cookie. The CSRF token must be same as initial value generated during login operation



The CSRF token will be used by the application server to verify the legitimacy of the end-user request if it is coming from the same App UI or not. The application server rejects the request if the CSRF token fails to match the test.

QUICK TIP

eazy bytes

Do you know,

- ✓ **By default, Spring Security enables CSRF fix for all the HTTP methods which results in data change like POST, DELETE etc. But not for GET.**

- ✓ **Using Spring Security configurations we can disable the CSRF protection for complete application or for only few paths based on our requirements like below.**

  - ***http.csrf((csrf) -> csrf.disable())***
  - ***http.csrf((csrf) -> csrf.ignoringRequestMatchers("/saveMsg"))***

- ✓ **Thymeleaf has great integration & support with Spring Security to generate a CSRF token. We just need to add the below code in login html form code and Thymeleaf will automatically appends the CSRF token for the remaining pages/forms inside the web application,**

```
<input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
```

SPRING BOOT & H2 DATABASE

eazy bytes

- *H2 is an embedded, open-source, and in-memory database. SpringBoot supports integration with H2 DB which can be used for POC applications and excellent for examples/testing.*
- *Below is the maven dependency that we can add to any SpringBoot projects in order to use internal memory H2 Database,*

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

Since it is a internal memory DB, we need to create the schema and data that is needed during startup of the App. Any updates to the data will be lost after restarting the server.

To create schema & data for the H2 DB, we can add schema.sql & data.sql inside the maven project's resources folder. Any table creation scripts and DB records scripts can be present inside schema.sql and data.sql respectively.

By default, the H2 web console is available at /h2-console. You can customize the console's path by using the spring.h2.console.path property.

The default credentials of H2 DB are username is sa and password is "" (Empty)