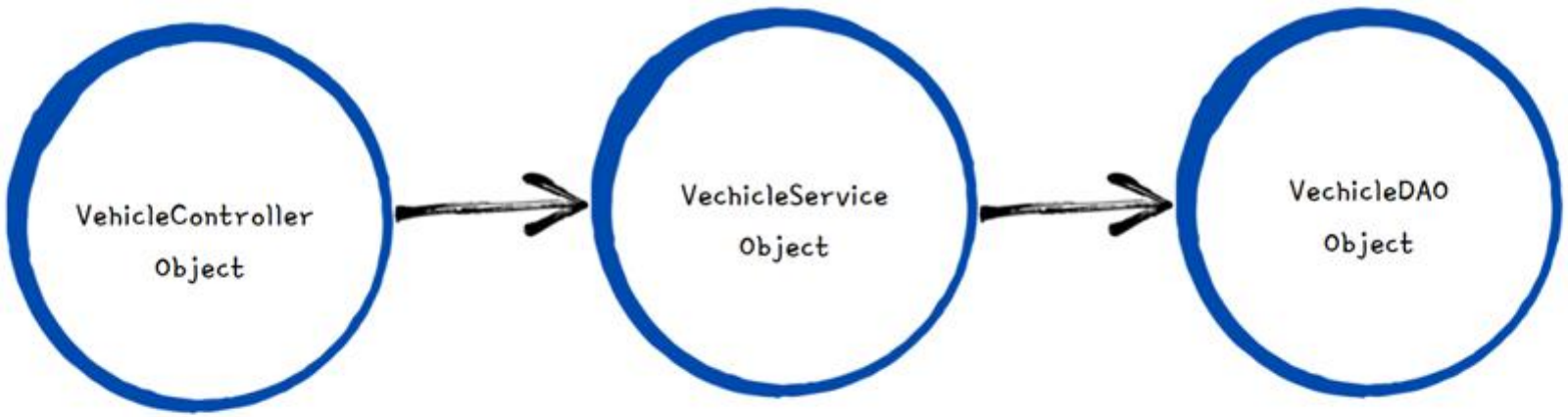
Spring, SpringBoot, JPA, Hibernate

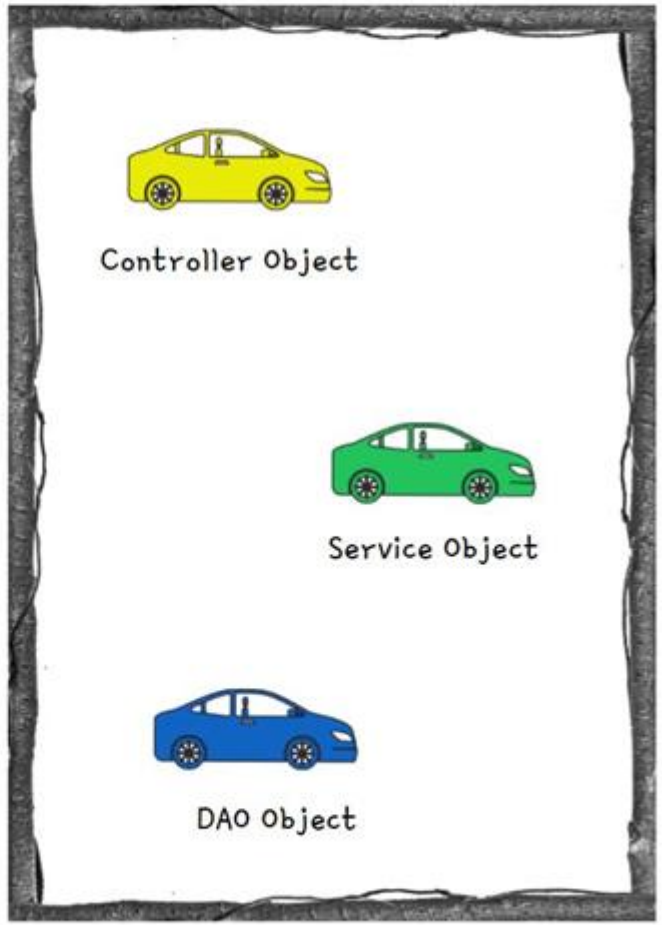
INTRODUCTION TO BEANS WIRING INSIDE SPRING

- *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.*

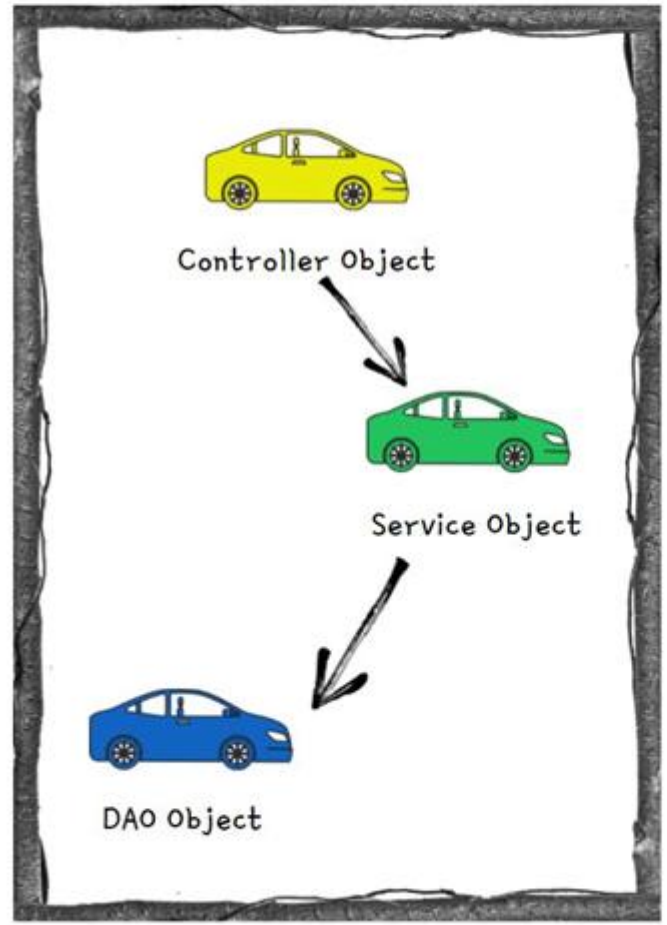


INTRODUCTION TO BEANS WIRING INSIDE SPRING

SPRING CONTEXT WITH OUT WIRING



SPRING CONTEXT WITH WIRING & DI



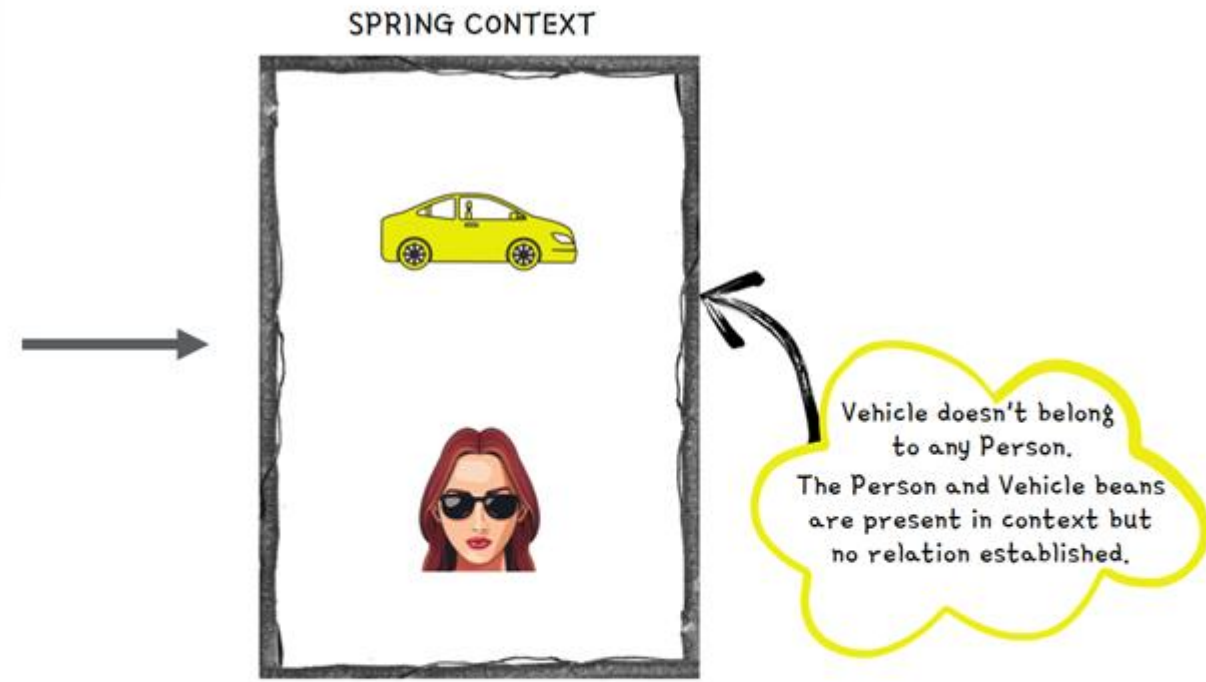
NO WIRING SCENARIO INSIDE SPRING

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;
```

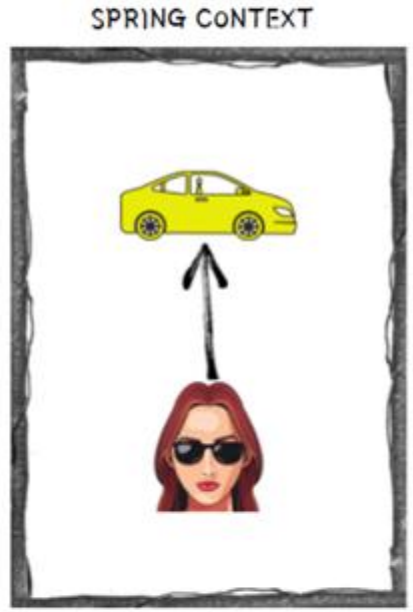
```java
@Bean
public Vehicle vehicle() {
    Vehicle vehicle = new Vehicle();
    vehicle.setName("Toyota");
    return vehicle;
}

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



WIRING BEANS USING METHOD CALL

- *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

- *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());
```

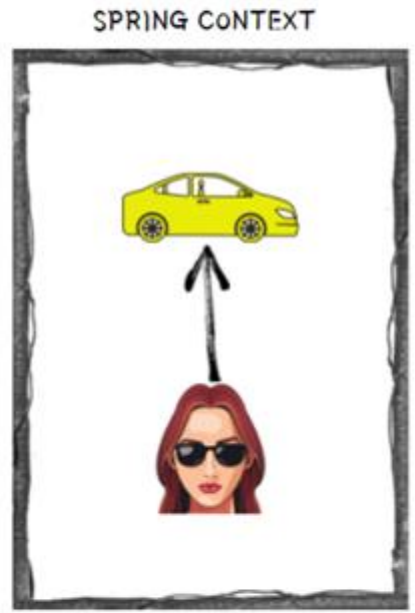
Output on Console

```
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

- *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

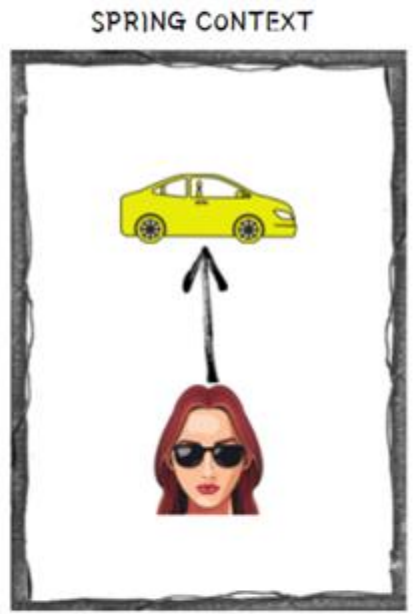
```
Vehicle bean created by Spring
Person bean created by Spring
Person name from Spring Context is: Lucy
Vehicle name from Spring Context is: Toyota
Vehicle that Person own is: Vehicle name is - Toyota
```

@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

- *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());
```

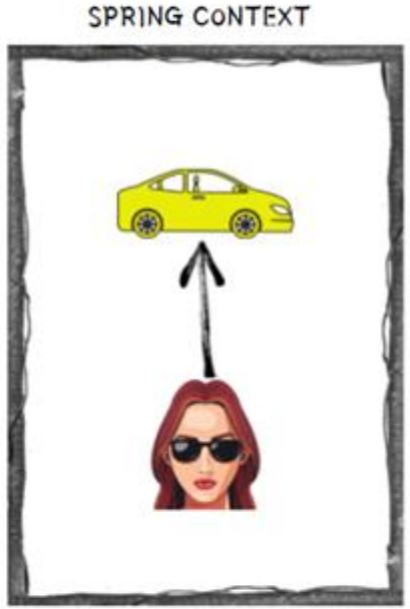
Output on Console

```
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

- *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

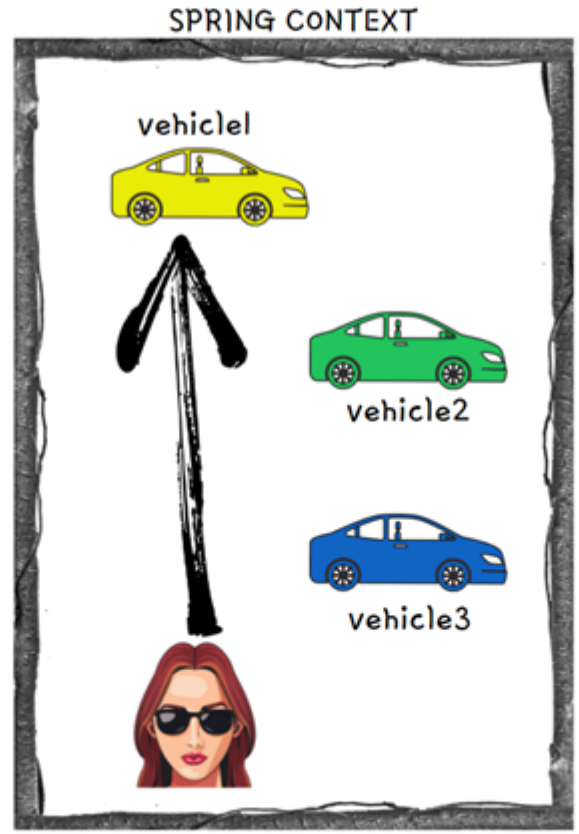
```
Vehicle bean created by Spring
Person bean created by Spring
Person name from Spring Context is: Lucy
Vehicle name from Spring Context is: Toyota
Vehicle that Person own is: Vehicle name is - Toyota
```

How Autowiring works with multiple Beans of same type

- *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

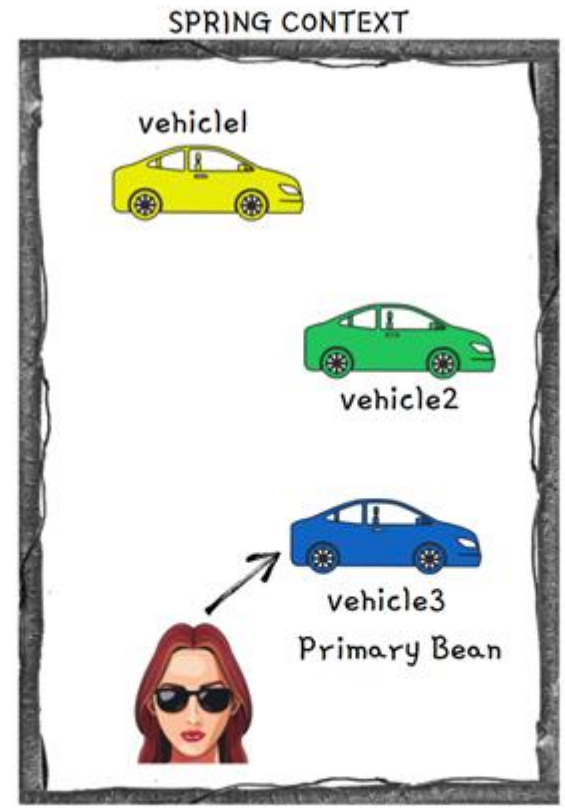


How Autowiring works with multiple Beans of same type

- *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;
    }
```

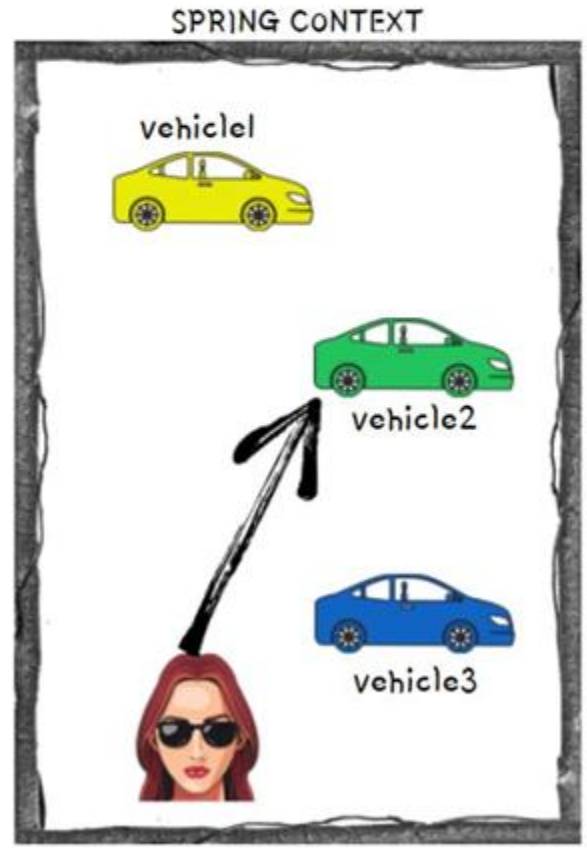
STEP 2



How Autowiring works with multiple Beans of same type

- *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.*

```java
@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;
    }
```



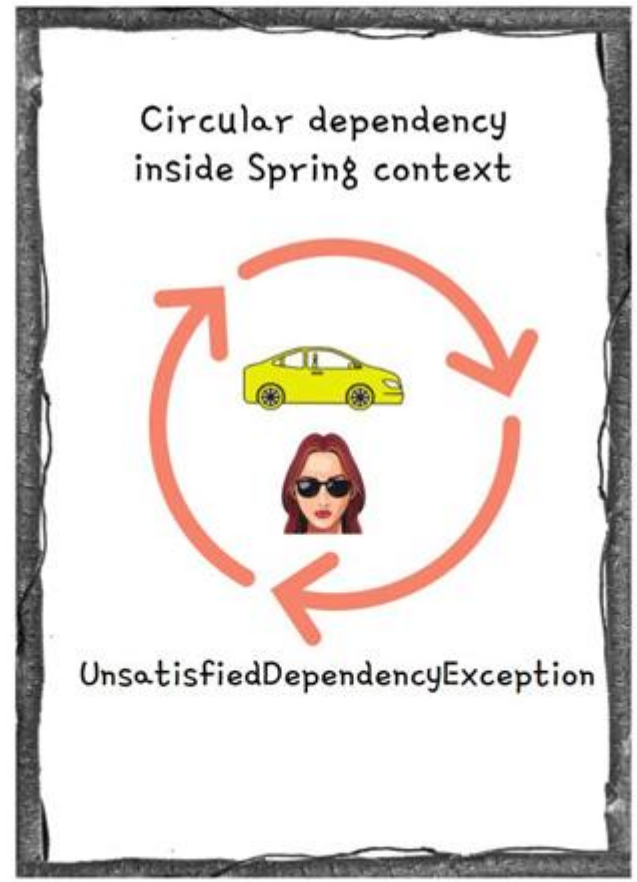
STEP 3

Understanding & Avoiding Circular dependencies

- *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

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

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