Spring, SpringBoot, JPA, Hibernate

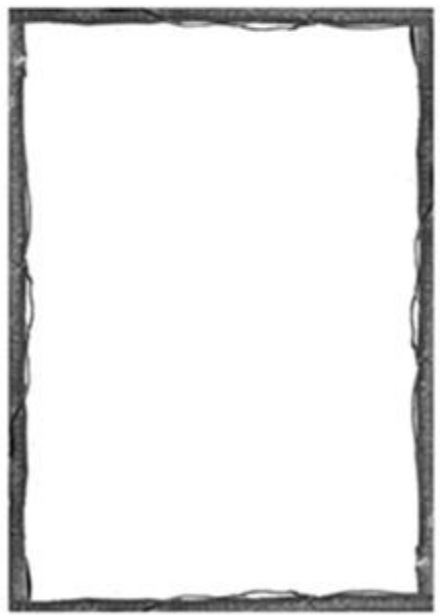
MAVEN

ADDING NEW BEANS TO SPRING CONTEXT

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();
```

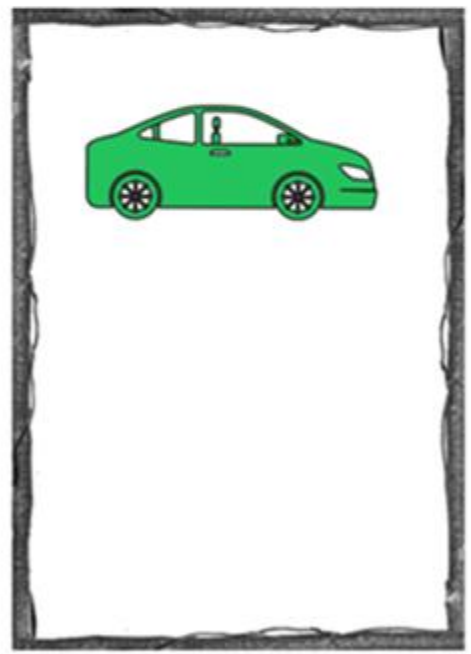
SPRING CONTEXT



@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;
}
```

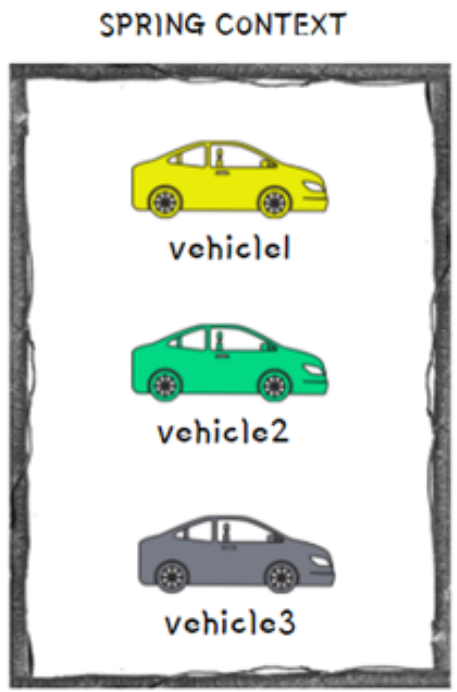
SPRING CONTEXT



NoUniqueBeanDefinitionException

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);
```

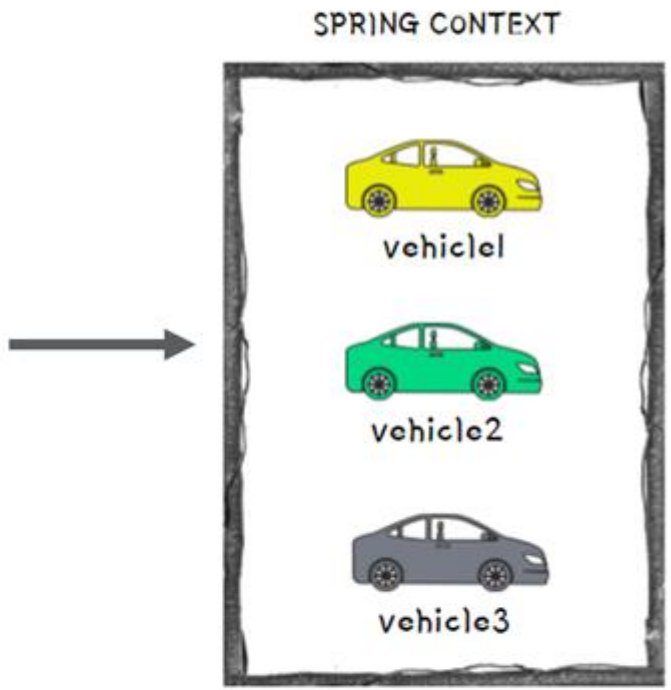
Output on Console

```
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

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());
```

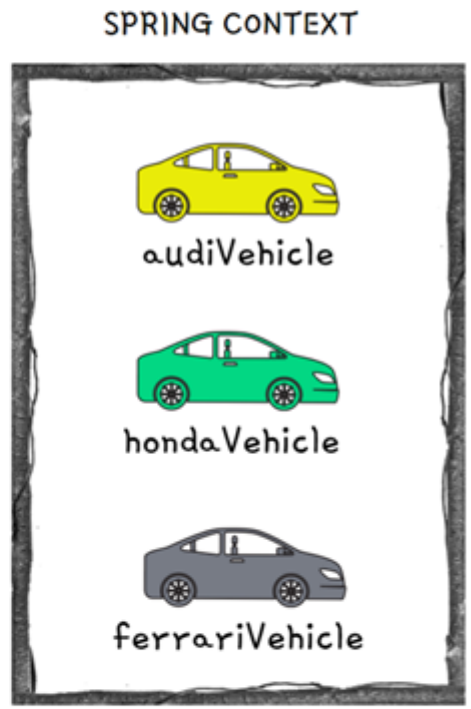
Output on Console

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

DIFFERENT WAYS TO NAME A BEAN

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());
```

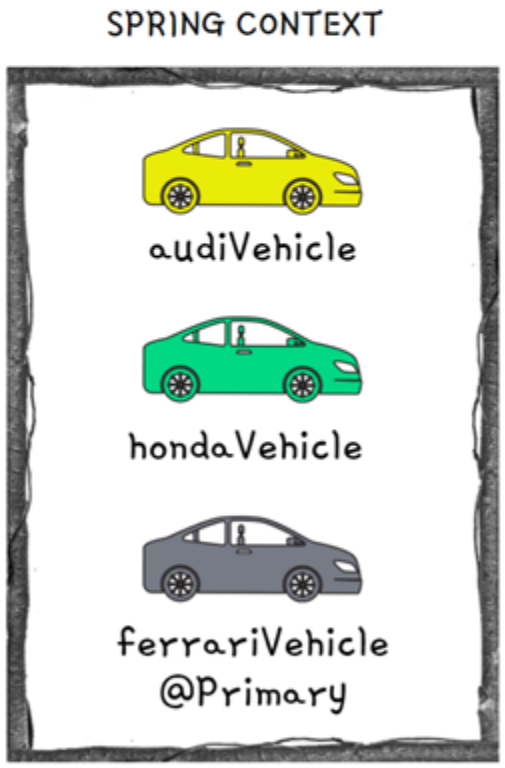
Output on Console

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

@Primary Annotation

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

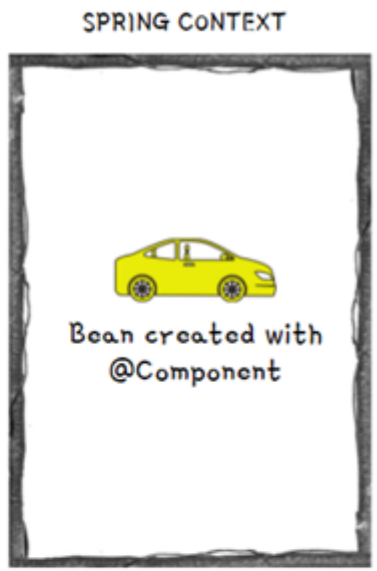
```
Primary Vehicle name from Spring Context is: Ferrari
```

@Component Annotation

@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();
```

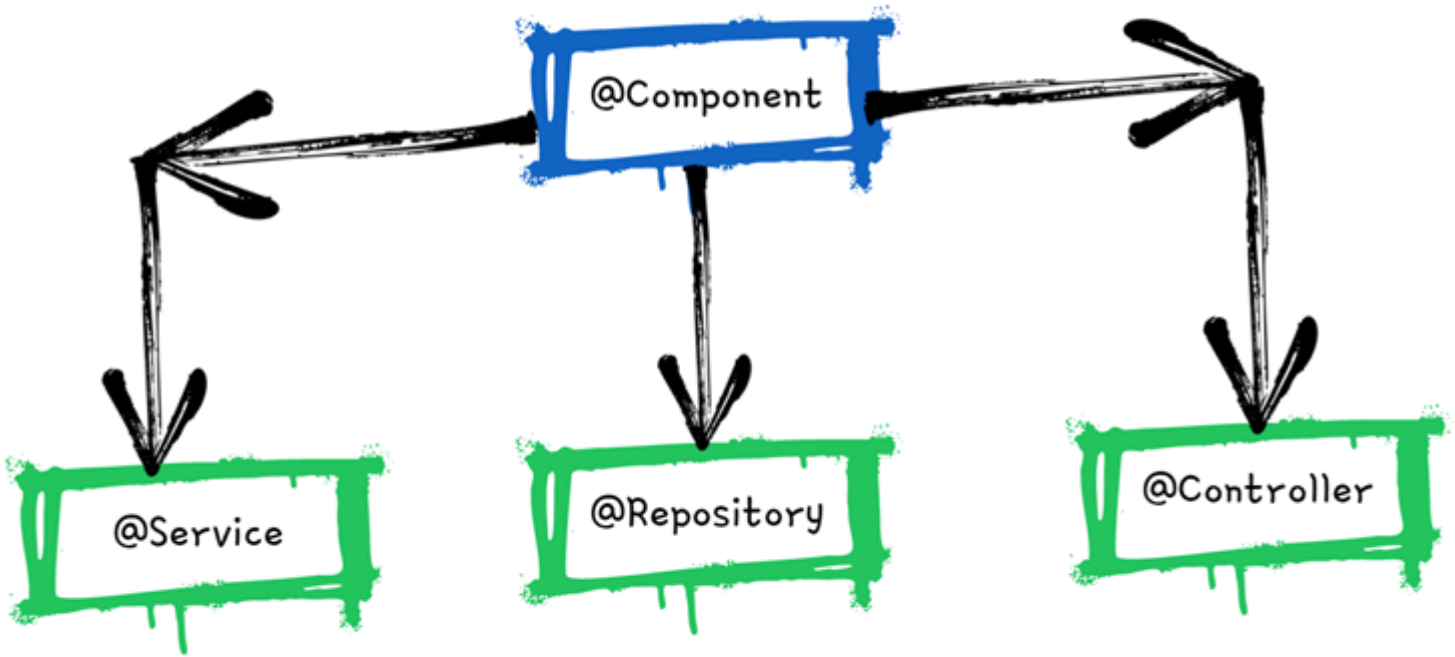
Output on Console

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

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

}
```

Spring Stereotype Annotations

- *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

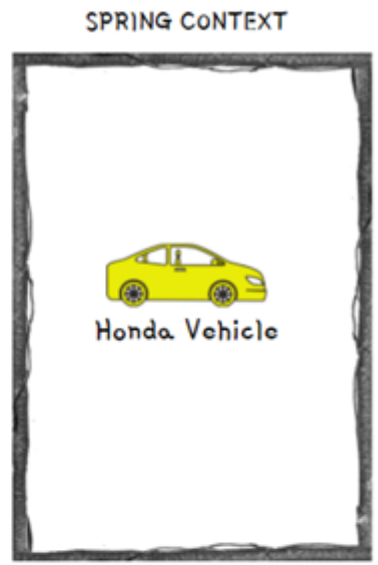
@Bean	@Component
One or more instances of the class can be added to the Spring Context	Only one instance of the class can be added to the Spring context
We can create an object instance of any type of class including present inside libraries like String etc.	We can create an object instance for application class only which are created by Dev team
Usually we need to write more code like separate methods to create beans instances	Bean instances can be created with very less code like using @Component on top of the class
Developer will have full control in creating and configuring the bean	Developer will not have any control in creating and configuring the bean
Spring framework creates the bean based on the instructions and values provided by Developer	Spring framework takes charge of creating the bean and post that Developer will have access to it

@PostConstruct Annotation

- *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

- *@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 {

}
```

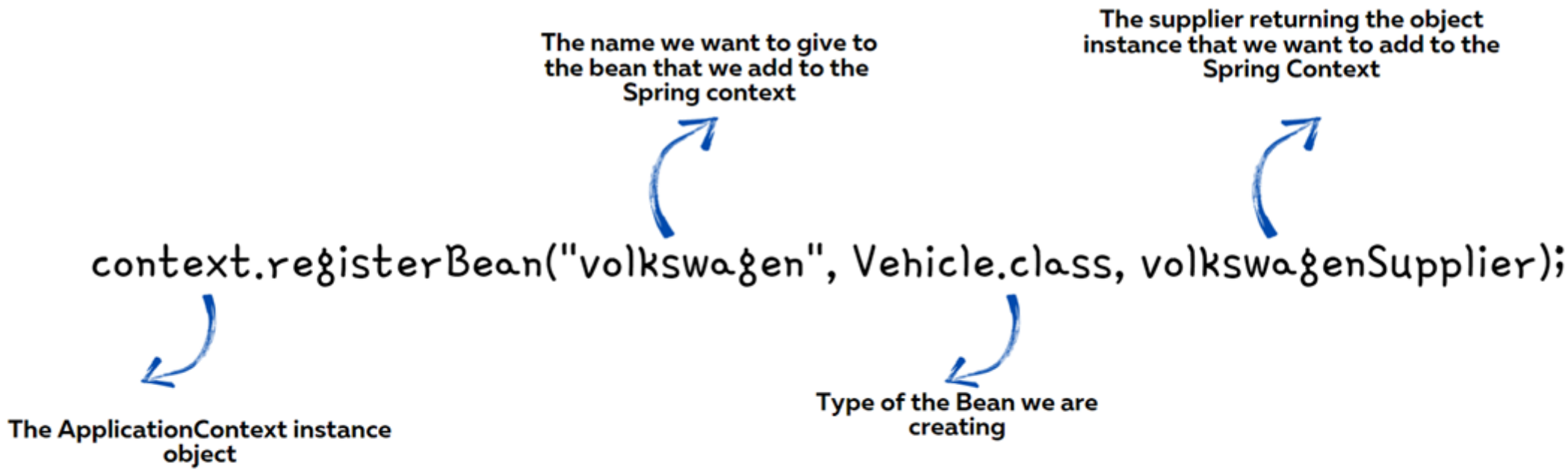

```java
var context = new AnnotationConfigApplicationContext
                (ProjectConfig.class);
Vehicle vehicle = context.getBean(Vehicle.class);
System.out.println("Component Vehicle name from " +
        "Spring Context is: " + vehicle.getName());
vehicle.printHello();
context.close();
```

Output on Console

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

ADDING NEW BEANS PROGRAMMATICALLY

- *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

```
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

- *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.*

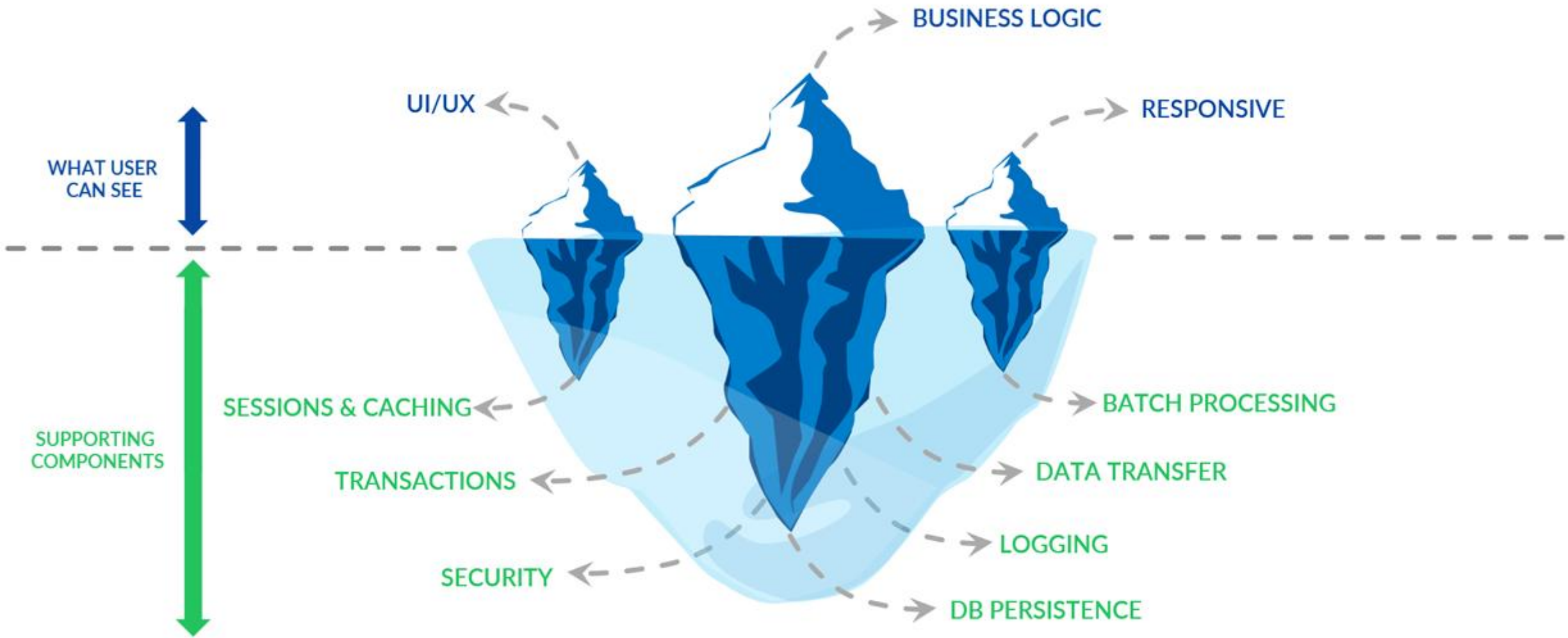
```xml
<bean id="vehicle" class="com.example.beans.Vehicle">
    <property name="name" value="Honda" />
</bean>
```


```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



WHY SHOULD WE USE FRAMEWORKS?

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?

DEV SANJEEV

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

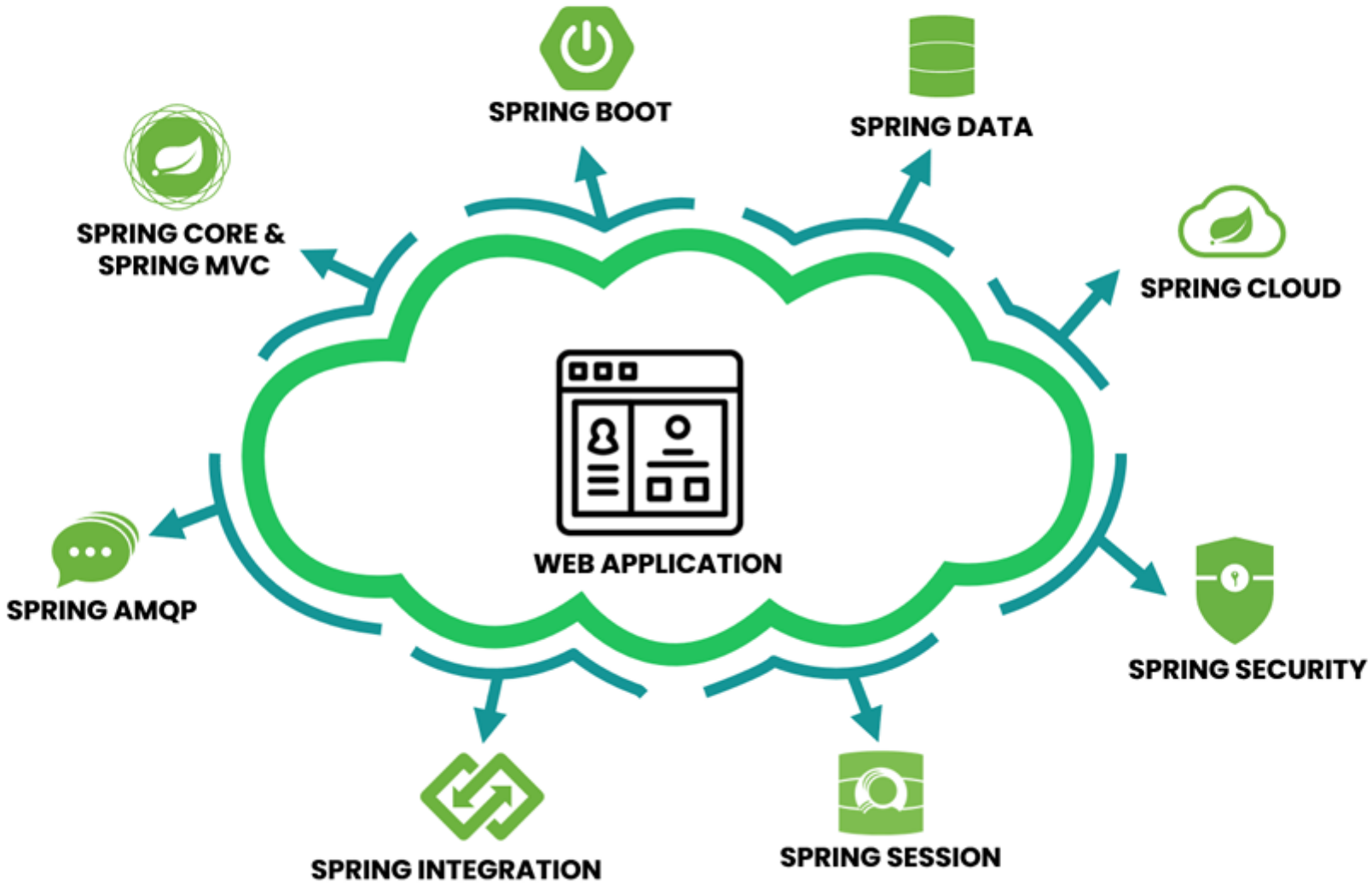
- Leverage Security, Logging etc. from frameworks
- Can easily scale his application
- App will work in an predictable manner
- Focus more on the business logic
- Less efforts and more results/revenue

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



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

SPRING PROJECTS

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