Tuesday, 2 July 2019

Decorator Pattern - Spring-Boot

The following Describes how to implement the Decorator Pattern in a Spring-Boot Application.

We have the following Interface

```
public interface IBouquet {
    public String deliverFlowers();
}
```

The First Implementation of this Interface is the Roses class

```
public class Roses implements IBouquet {
    @Override
    public String deliverFlowers() {
        return "Deliver Roses";
    }
}
```

We now wanted to decorate the Roses with Ribbons. We have two options to do this.

Option 1 , Create a new Class which extends Roses then add the ribbons.

```
public class RibbonsOnRoses extends Roses {
    @Override
    public String deliverFlowers() {
        return super.deliverFlowers() + "and add Ribbons";
    }
```

This would be acceptable is there was only one way to decorate the Roses. A more elegant flexible alternative is to use the Decorator Pattern.

Option 2 , We use the Decorator Pattern , the following example will demonstrate the Decorator Pattern Using Spring-Boot.

First we will define the Roses Class as a Component by adding @Component(“roses”)

```
@Component ("roses")
public class Roses implements IBouquet {
    @Override
    public String deliverFlowers() {
        return "Deliver Roses";
    }
}
```

Next we will create a new Decorator Class and set it as the Primary Implementation of the IBouquet Interface by annotating with @Primary. We instantiate the Roses Class by Auto-wiring using the Qualifier @Qualifier("roses") we defined in Step one

```
@Component
@Primary
public class DecorateRosesWithRibbons implements IBouquet{
    @Autowired
    @Qualifier("roses")
    IBouquet bouquet;
    @Override
    public String deliverFlowers() {
        String flowers = bouquet.deliverFlowers();
        return flowers + " With Ribbons";
    }
}
```

To use the Decorated or Undecorated in the @SpringBootApplication , we can use the following

As we set the DecorateRosesWithRibbons as Primary with @Primary , not specifying a Qualifier will automatically load the DecorateRosesWithRibbons

```
@Autowired
private IBouquet bouquet;

System.out.println(bouquet.deliverFlowers());
```

will output “Deliver Roses With Ribbons”

If we want to use the undecorated , we can specify using the @Qualifier(“roses”).

```
@Autowired
@Qualifier("roses")
private IBouquet bouquet;

System.out.println(bouquet.deliverFlowers());
```

will output “Deliver Roses”

Lets add the Chocolates Option , we label the component @Component (“chocolates”) and Instantiate the Roses Class @Autowired Roses roses.

```
@Component ("chocolates")
public class Chocolates implements IBouquet {
    @Autowired
    Roses roses;
    @Override
    public String deliverFlowers() {
        return (roses.deliverFlowers() + " and add
chocolates");
    }
}
```

now Instantiating this in the @SpringBootApplication would be :

```
@Autowired
@Qualifier("chocolates")
private IBouquet chocolates;

System.out.println("chocolates.deliverFlowers());
```

will output “Deliver Roses and add chocolates”

next , lets Decorate the Primary(DecorateRosesWithRibbons) by creating an option of adding chocolates with the Ribbons

```
@Component("choc_ribbons")
public class ChocolateswithRibbons implements IBouquet {
    @Autowired
    DecorateRosesWithRibbons decorateRosesWithRibbons;
    @Override
    public String deliverFlowers() {
        return (decorateRosesWithRibbons.deliverFlowers() + "
add chocolates");
    }
}
```

now Instantiating this in the @SpringBootApplication would be :

```
@Autowired
@Qualifier("choc_ribbons")
private IBouquet choc_ribbon;

System.out.println("choc_ribbon.deliverFlowers());
```

will output “Deliver Roses With Ribbons and add chocolates”

The following depicts the runtime Class i.e @SpringBootApplication

```
@SpringBootApplication
public class DecoratorpatternApplication implements
CommandLineRunner {
	//Primary
	@Autowired
	private IBouquet bouquet;
	//Roses Component
	@Autowired
	@Qualifier("roses")
	private IBouquet bouquetr;
	//Tulips Component
	@Autowired
	@Qualifier("tulips")
	private IBouquet bouquett;
	//Chocolates Component
	@Autowired
	@Qualifier("chocolates")
	private IBouquet chocolates;
   //Chocolates with Ribbons Component
   @Autowired
   @Qualifier("choc_ribbons")
   private IBouquet choc_ribbon;

	public static void main(String[] args) {
	SpringApplication.run(DecoratorpatternApplication.class,
args);
	}
	@Override
	public void run(String... args) throws Exception {
		System.out.println("Primary :" +
bouquet.deliverFlowers());
		System.out.println("Roses Component :" +
bouquetr.deliverFlowers());
		System.out.println("Tulips Component :" +
bouquett.deliverFlowers());
		System.out.println("Chocolates Component :" +
chocolates.deliverFlowers());
	   System.out.println("Chocolates with Ribbons
Component :" + choc_ribbon.deliverFlowers());
   }
}
```