



Academia Xideral

Regina Rodriguez Campo Garrido

“Decorator Pattern”

08/31/2024

## Decorator

Decorator is a structural design pattern that lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.

A spa chain offers various basic. We want to allow customers to customize these services with additional add-ons. Each add-on comes with an additional cost.

We create our client class with the attributes we want

```
1 package Decorate;
2
3 public class Client {
4
5     private String name;
6     private int cost;
7
8     public Client(String name) { //constructor
9         super();
10        this.name = name;
11    }
12
13
14    public String getName() {
15        return name;
16    }
17    public void setName(String name) {
18        this.name = name;
19    }
20    public int getCost() {
21        return cost;
22    }
23    public void setCost(int cost) {
24        this.cost = cost;
25    }
26
27
28 }
29
```

The Service interface defines a method. This method is intended to be

Implemented by classes that provide different types of services.

```
1 package Decorate;
2
3 public interface Service {
4
5     void Service(Client client);
6
7 }
8
```

The Service Decorate abstract class implements the Service Interface and serves as a base for concrete decorators.

```
Client.java x Service.java ServiceDecorate.java x
1 package Decorate;
2
3 public abstract class ServiceDecorate implements Service{
4
5     protected Service serDecorate;
6     String ExtraName;
7     int extraPrice;
8
9     public ServiceDecorate(Service serDecorate) {
10         this.serDecorate = serDecorate;
11     }
12
13     @Override
14     public void Service(Client client) {
15         client.setCost(client.getCost() + extraPrice);
16         System.out.println("Extra: " + ExtraName);
17         System.out.print("Price: " + extraPrice);
18     }
19 }
20
21 }
22
23 }
```

We create the classes that will be the spa services. Those The classes implements the Service interface and represents a concrete service.

```
1 package Decorate;
2
3 public class SwedishMassage implements Service {
4
5     int price = 450;
6
7     @Override
8     public void Service(Client client) {
9         client.setCost(client.getCost() + price);
10        System.out.println("Swedish Massage Service Price: " + price);
11        System.out.println("Cliente: " + client.getName());
12    }
13 }
14
15 }
```

```

1 package Decorate;
2
3 public class FacialTreatment implements Service{
4
5     int price = 500;
6
7     @Override
8     public void Service(Client client) {
9         client.setCost(client.getCost() + price);
10        System.out.println("Facial Treatment Service Price: " + price);
11        System.out.println("Client: " + client.getName());
12    }
13 }
14
15 }
16

```

Create extra class extends Service Decorate and represents a concrete decorator that adds extra features to the base service.

```

1 package Decorate;
2
3 public class EssentialOils extends ServiceDecorate {
4
5     public EssentialOils(Service serviceDecorate) {
6         super(serviceDecorate);
7         ExtraName = "Essential Oils";
8         extraPrice = 135;
9     }
10 }
11
12

```

```

1 package Decorate;
2
3 public class HotStone extends ServiceDecorate {
4
5     public HotStone(Service serviceDecorate) {
6         super(serviceDecorate);
7         ExtraName = "Essential Oils";
8         extraPrice = 50;
9     }
10 }

```

We create in main where the client makes the packages of the services they want.



```
1 package Decorate;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         Client client1 = new Client("Monica");
8         Service serv1 = new FacialTreatment();
9         ServiceDecorate serv2 = new EssentialOils(serv1);
10        serv1.Service(client1);
11        serv2.Service(client1);
12        System.out.println();
13        System.out.println("Total cost for " + client1.getName() + ": " + client1.getCost());
14
15        System.out.println();
16
17        Client client2 = new Client("Regina");
18        Service serv3 = new SwedishMassage();
19        ServiceDecorate serv4 = new HotStone(serv3);
20        serv3.Service(client2);
21        serv4.Service(client2);
22        System.out.println();
23        System.out.println("Total cost for " + client2.getName() + ": " + client2.getCost());
24    }
25
26 }
27 }
```