The Decorator Pattern is a structural design pattern that allows behavior to be added to an individual object,

either statically or dynamically, without affecting the behavior of other objects from the same class.

It provides a flexible alternative to subclassing for extending functionality.

The Decorator Pattern involves wrapping the original object with one or more decorator objects that add the desired behavior.

Let's say you have a Beverage class and you want to add optional toppings

(such as whipped cream or chocolate syrup) to a beverage without changing the original class.

Using the Decorator Pattern, you can achieve the following benefits:

Open/Closed Principle (OCP): You can add new toppings without modifying the existing code, adhering to the OCP.

Single Responsibility Principle (SRP): The Beverage class focuses on its core responsibility, and the decorators handle the additional behavior.

Flexibility: You can easily add or remove toppings dynamically without affecting the original object.

Extensibility: You can introduce new toppings by creating new decorator classes, promoting code reusability.

 *// Step 1: Component interface
interface Topping* {
 String getDescription();
}

*class* Beverage *implements Topping* {
 *private* String description = "Beverage";

 *public* String getDescription() {
 *return* description;
 }
}

*class* WhippedCream *implements Topping* {
 *private Topping* beverage;

 *public* WhippedCream(*Topping* beverage) {
 *this*.beverage = beverage;
 }

 *public* String getDescription() {
 *return* beverage.getDescription() + ", Whipped Cream";
 }
}

*class* ChocolateSyrup *implements Topping* {
 *private Topping* beverage;

 *public* ChocolateSyrup(*Topping* beverage) {
 *this*.beverage = beverage;
 }

 *public* String getDescription() {
 *return* beverage.getDescription() + ", Chocolate Syrup";
 }
}

*public class* Decorator {
 *public static void* main(String[] args) {
 Beverage beverage = *new* Beverage();
 System.*out*.println(beverage.getDescription());

 *Topping* beverageWithWhippedCream = *new* WhippedCream(beverage);
 System.*out*.println(beverageWithWhippedCream.getDescription());

 *Topping* beverageWithChocolateSyrup = *new* ChocolateSyrup(beverage);
 System.*out*.println(beverageWithChocolateSyrup.getDescription());
 }
}

Beverage

Beverage, Whipped Cream

Beverage, Chocolate Syrup

*interface Car* {
 *public void* assemble();
}

*class* BasicCar *implements Car* {

 @Override
 *public void* assemble() {
 System.*out*.print("Basic Car.");
 }

}

*class* CarDecorator *implements Car* {

 *protected Car* car;

 *public* CarDecorator(*Car* c) {
 *this*.car = c;
 }

 @Override
 *public void* assemble() {
 *this*.car.assemble();
 }

}

*class* SportsCar *extends* CarDecorator {

 *public* SportsCar(*Car* c) {
 *super*(c);
 }

 @Override
 *public void* assemble() {
 *super*.assemble();
 System.*out*.print(" Adding features of Sports Car.");
 }
}

*class* LuxuryCar *extends* CarDecorator {

 *public* LuxuryCar(*Car* c) {
 *super*(c);
 }

 @Override
 *public void* assemble() {
 *super*.assemble();
 System.*out*.print(" Adding features of Luxury Car.");
 }
}

*public class* CarDecoratorPattern {
 *public static void* main(String[] args) {
 *Car* sportsCar = *new* SportsCar(*new* BasicCar());
 sportsCar.assemble();
 System.*out*.println("\n\*\*\*\*\*");

 *Car* sportsLuxuryCar = *new* SportsCar(*new* LuxuryCar(*new* BasicCar()));
 sportsLuxuryCar.assemble();
 }
}

Basic Car. Adding features of Sports Car.

\*\*\*\*\*

Basic Car. Adding features of Luxury Car. Adding features of Sports Car.



