The Decorator design pattern is a structural design pattern in object-oriented programming that allows behavior and functionality to be dynamically added to an object without affecting the behavior of other objects in the same class.

This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

It involves both the Is-A and Has-A relationship.

Why do you need decorator pattern ?

It is difficult to manage so many classes if they get extended from a base class for a specific feature , and it leads to the class explosion.

Example :

Base Class : Vehicle

Subclasses : Car , Car with AC , Car with Fog light , Car with power steering etc . many permutations and combinations.

UseCase : Pizza Toppings

public abstract class BasePizza{  
 public abstract int cost() ;  
}

public class VegDelight extends BasePizza {  
  
 @Override  
 public int cost() {  
 return 120;  
 }  
}

public class Margerita extends BasePizza{  
  
 @Override  
 public int cost() {  
 return 100;  
 }  
}

public class FormHouse extends BasePizza {  
  
 @Override  
 public int cost() {  
 return 200;  
 }  
}

public abstract class ToppingDecorator extends BasePizza {  
  
}

public class ExtraCheese extends ToppingDecorator {  
  
 BasePizza basePizza ;  
  
 public ExtraCheese(BasePizza basePizza ){  
 this.basePizza = basePizza ;  
 }  
  
 @Override  
 public int cost() {  
 return basePizza.cost() + 10 ;  
 }  
}

public class Mushroom extends ToppingDecorator{  
  
 BasePizza basePizza ;  
  
 public Mushroom(BasePizza basePizza){  
 this.basePizza = basePizza ;  
 }  
  
 @Override  
 public int cost() {  
 return this.basePizza.cost() + 15 ;  
 }  
}

public class DecoratorDesignPattern {  
  
 public static void main(String[] args) {  
  
 *// If we want Margerita + ExtraCheese* BasePizza pizza1 = new ExtraCheese(new Margerita()) ;  
  
 System.*out*.println(pizza1.cost());  
  
 *// If we want Margerita + ExtraCheese + Mashroom* BasePizza pizza2 = new Mushroom(new ExtraCheese(new Margerita())) ;  
  
 System.*out*.println(pizza2.cost());  
  
 }  
}