Name: -Aditya Raj Roll No: - 11212714 Section: - A1

Design Patterns
W11A1

Topic: - Write a program to implement Decorator Design Pattern

Definition:-

A Decorator Pattern says that just **"attach a flexible additional responsibilities to an object dynamically".**

In other words, The Decorator Pattern uses composition instead of inheritance to extend the functionality of an object at runtime.

The Decorator Pattern is also known as **Wrapper.**

Advantage of Decorator Pattern

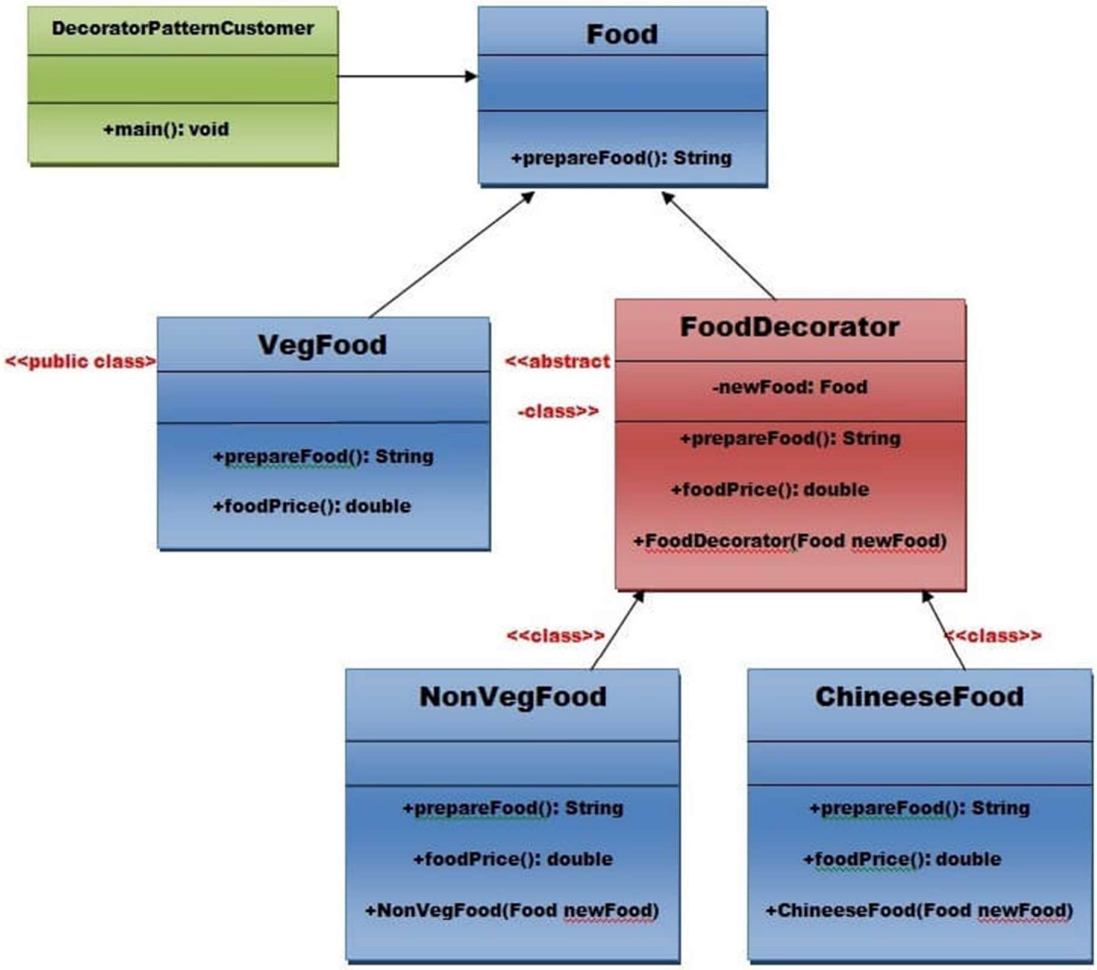
- It provides greater flexibility than static inheritance.
- It enhances the extensibility of the object, because changes are made by coding new classes.
- It simplifies the coding by allowing you to develop a series of functionality from targeted classes instead of coding all of the behavior into the object.

Usage of Decorator Pattern

It is used:

- When you want to transparently and dynamically add responsibilities to objects without affecting other objects.
- When you want to add responsibilities to an object that you may want to change in future.
- Extending functionality by sub-classing is no longer practical.

UML for Decorator Pattern:



Code: -

```
package decorator;

interface Dress {
  public void assemble();
}

class BasicDress implements Dress {
  @Override
  public void assemble() {
    System.out.println("Basic Dress Features");
```

```
  }
}

class DressDecorator implements Dress {
  protected Dress dress;

  public DressDecorator(Dress c) {
    this.dress = c;
  }

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

class CasualDress extends DressDecorator {
  public CasualDress(Dress c) {
    super(c);
  }

  @Override
  public void assemble() {
    super.assemble();
    System.out.println("Adding Casual Dress Features");
  }
}


class SportyDress extends DressDecorator {
  public SportyDress(Dress c) {
    super(c);
  }

  @Override
  public void assemble() {
    super.assemble();
    System.out.println("Adding Sporty Dress Features");
  }
}

class FancyDress extends DressDecorator {
  public FancyDress(Dress c) {
```

```
    super(c);
  }

  @Override
  public void assemble() {
    super.assemble();
    System.out.println("Adding Fancy Dress Features");
  }
}

public class DecoratorPatterTest {

  public static void main(String[] args) {

    Dress sportyDress = new SportyDress(new BasicDress());
    sportyDress.assemble();
    System.out.println();

    Dress fancyDress = new FancyDress(new BasicDress());
    fancyDress.assemble();
    System.out.println();

    Dress casualDress = new CasualDress(new BasicDress());
    casualDress.assemble();
    System.out.println();


    Dress sportyFancyDress = new SportyDress(new FancyDress(new BasicDress()));
    sportyFancyDress.assemble();
    System.out.println();

    Dress casualFancyDress = new CasualDress(new FancyDress(new BasicDress()));
    casualFancyDress.assemble();
        }
}
```

Output: -

Name: -Aditya Raj Roll No: - 11212714 Section: - A1

```
PS D:\Design-And-Pattern> cd
Basic Dress Features
Adding Sporty Dress Features

Basic Dress Features
Adding Fancy Dress Features

Basic Dress Features
Adding Casual Dress Features

Basic Dress Features
Adding Fancy Dress Features
Adding Sporty Dress Features

Basic Dress Features
Adding Fancy Dress Features
Adding Casual Dress Features
PS D:\Design-And-Pattern>
```