En cada uno de los siguientes programas, identifique si se usa el patrón decorator o no y justifique el porqué.

1. **Cofee**

```java
interface Coffee {
    double cost();
    String getDescription();
}

class BasicCoffee implements Coffee {
    @Override
    public double cost() {
        return 5.0;
    }

    @Override
    public String getDescription() {
        return "Basic Coffee";
    }
}

abstract class CoffeeDecorator implements Coffee {
    protected Coffee coffee;

    public CoffeeDecorator(Coffee coffee) {
        this.coffee = coffee;
    }
}
```

```java
class MilkDecorator extends CoffeeDecorator {
    public MilkDecorator(Coffee coffee) {
        super(coffee);
    }

    @Override
    public double cost() {
        return coffee.cost() + 1.5;
    }

    @Override
    public String getDescription() {
        return coffee.getDescription() + ", Milk";
    }
}

public class CoffeeShop {
    public static void main(String[] args) {
        Coffee coffee = new BasicCoffee();
        System.out.println(coffee.getDescription() + " $ " + coffee.cost());

        Coffee coffeeWithMilk = new MilkDecorator(new BasicCoffee());
        System.out.println(coffeeWithMilk.getDescription() + " $ " + coffeeWithMilk.cost());
    }
}
```

2. **Pizza**

```java
interface Pizza {
    double cost();
    String getDescription();
}

class PlainPizza implements Pizza {
    @Override
    public double cost() {
        return 8.0;
    }

    @Override
    public String getDescription() {
        return "Plain Pizza";
    }
}

class CheesePizza implements Pizza {
    @Override
    public double cost() {
        return 10.0;
    }

    @Override
    public String getDescription() {
        return "Cheese Pizza";
    }
}
```

```java
public class PizzaShop {
    public static void main(String[] args) {
        Pizza plainPizza = new PlainPizza();
        System.out.println(plainPizza.getDescription() + " $ " + plainPizza.cost());

        Pizza cheesePizza = new CheesePizza();
        System.out.println(cheesePizza.getDescription() + " $ " + cheesePizza.cost());
    }
}
```

3. **IceCream**

```java
interface IceCream {
    double cost();
    String getDescription();
}

class BasicIceCream implements IceCream {
    @Override
    public double cost() {
        return 2.0;
    }

    @Override
    public String getDescription() {
        return "Basic Ice Cream";
    }
}
```

```java
class ToppingDecorator implements IceCream {
    private final IceCream iceCream;
    private final String topping;
    private final double toppingCost;

    public ToppingDecorator(IceCream iceCream, String topping, double toppingCost) {
        this.iceCream = iceCream;
        this.topping = topping;
        this.toppingCost = toppingCost;
    }

    @Override
    public double cost() {
        return iceCream.cost() + toppingCost;
    }

    @Override
    public String getDescription() {
        return iceCream.getDescription() + " with " + topping;
    }
}

public class IceCreamShop {
    public static void main(String[] args) {
        IceCream iceCream = new BasicIceCream();
        System.out.println(iceCream.getDescription() + " $ " + iceCream.cost());

        IceCream iceCreamWithTopping = new ToppingDecorator(new BasicIceCream(), "Chocolate Chips", 1.0);
        System.out.println(iceCreamWithTopping.getDescription() + " $ " + iceCreamWithTopping.cost());
    }
}
```

4. **Message**

```java
interface Message {
    String encrypt();
    int getSize();
}

class PlainMessage implements Message {
    private String content;

    public PlainMessage(String content) {
        this.content = content;
    }

    @Override
    public String encrypt() {
        return "Encrypted(" + content + ")";
    }

    @Override
    public int getSize() {
        return content.length();
    }
}

abstract class MessageDecorator implements Message {
    protected Message message;

    public MessageDecorator(Message message) {
        this.message = message;
    }
}
```

```java
class ExtraEncryption extends MessageDecorator {
    public ExtraEncryption(Message message) {
        super(message);
    }

    @Override
    public String encrypt() {
        return "ExtraEncryption(" + message.encrypt() + ")";
    }

    @Override
    public int getSize() {
        return message.getSize() + 5;
    }
}

class Compression extends MessageDecorator {
    public Compression(Message message) {
        super(message);
    }

    @Override
    public String encrypt() {
        return "Compressed(" + message.encrypt() + ")";
    }

    @Override
    public int getSize() {
        return message.getSize() / 2;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Message plainMessage = new PlainMessage("Hello World");
        System.out.println(plainMessage.encrypt() + " Size: " + plainMessage.getSize());

        Message encryptedMessage = new ExtraEncryption(plainMessage);
        System.out.println(encryptedMessage.encrypt() + " Size: " + encryptedMessage.getSize());

        Message compressedAndEncrypted = new Compression(new ExtraEncryption(plainMessage));
        System.out.println(compressedAndEncrypted.encrypt() + " Size: " + compressedAndEncrypted.getSize());
    }
}
```