Factory pattern with shape factory area

🧩 Step-1: Shape Interface

```
public interface Shape {
    double getArea();
}
```

- সব shape-এর জন্য common interface
- Factory শুধু `Shape` return করবে → loose coupling

🔵 Step-2: Circle Class

```
public class Circle implements Shape {

    private double radius;

    // default constructor
    public Circle() {
        this.radius = 1.0;   // default value
    }

    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }
}
```

📌 Formula: πr^2
📌 radius = 1 (default)

🔵 Step-3: Rectangle Class

```
public class Rectangle implements Shape {

    private double width;
    private double height;

    public Rectangle() {
        this.width = 2.0;     // default
        this.height = 1.5;    // default
    }

    @Override
    public double getArea() {
        return width * height;
    }
}
```

📌 Formula: width × height

🔵 Step-4: Triangle Class

```
public class Triangle implements Shape {

    private double a, b, c;

    public Triangle() {
        this.a = 3.0;
        this.b = 4.0;
        this.c = 5.0;   // valid triangle
    }

    @Override
    public double getArea() {
        double s = (a + b + c) / 2;
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }
}
```

📌 Heron's Formula
📌 s = (a + b + c) / 2

🏭 Step-5: ShapeFactory Class (Core of Factory Pattern)

```
public class ShapeFactory {

    public Shape getShape(String shapeType) {

        if (shapeType == null)
            return null;

        if (shapeType.equalsIgnoreCase("CIRCLE")) {
            return new Circle();
        }
        else if (shapeType.equalsIgnoreCase("RECTANGLE")) {
            return new Rectangle();
        }
        else if (shapeType.equalsIgnoreCase("TRIANGLE")) {
            return new Triangle();
        }

        return null;
    }
}
```

🔑 এখানে কী হচ্ছে:

- `new` keyword client জানে না

- object creation centralized
- client শুধু shape name দেয়

Step-6: Demo / Main Class

```
public class FactoryPatternDemo {

    public static void main(String[] args) {

        ShapeFactory factory = new ShapeFactory();

        Shape circle = factory.getShape("CIRCLE");
        System.out.println("Circle Area = " + circle.getArea());

        Shape rectangle = factory.getShape("RECTANGLE");
        System.out.println("Rectangle Area = " + rectangle.getArea());

        Shape triangle = factory.getShape("TRIANGLE");
        System.out.println("Triangle Area = " + triangle.getArea());
    }
}
```

Output (Approximate)

```
Circle Area = 3.141592653589793
Rectangle Area = 3.0
Triangle Area = 6.0
```

Abstract factory with car

Step-1: Car Interface (Abstract Product)

```
public interface Car {
    void showSpecifications();
}
```

সব car-এর common behavior

Step-2: Abstract Car Classes (Optional but clean)

```
public abstract class AbstractCar implements Car {

    protected String region;
    protected String type;
    protected String engineVolume;
    protected String color;
    protected String fuelType;

    @Override
    public void showSpecifications() {
        System.out.println("Region: " + region);
        System.out.println("Car Type: " + type);
        System.out.println("Engine: " + engineVolume);
        System.out.println("Color: " + color);
        System.out.println("Fuel: " + fuelType);
        System.out.println("----------------------");
    }
}
```

Step-3: USA Cars (Concrete Products)

USA Small

```
public class USASmallCar extends AbstractCar {
    public USASmallCar() {
        region = "USA";
        type = "Small";
        engineVolume = "1.0L";
        color = "Red";
        fuelType = "Petrol";
    }
}
```

USA Sedan

```
public class USASedanCar extends AbstractCar {
    public USASedanCar() {
        region = "USA";
        type = "Sedan";
        engineVolume = "3.5L";
        color = "Blue";
```

```
        fuelType = "Petrol";
    }
}
```

USA Luxury

```
public class USALuxuryCar extends AbstractCar {
    public USALuxuryCar() {
        region = "USA";
        type = "Luxury";
        engineVolume = "5.0L";
        color = "White";
        fuelType = "Petrol";
    }
}
```

Step-4: Asia Cars (Concrete Products)

Asia Small

```
public class AsiaSmallCar extends AbstractCar {
    public AsiaSmallCar() {
        region = "Asia";
        type = "Small";
        engineVolume = "0.8L";
        color = "Silver";
        fuelType = "Electric";
    }
}
```

Asia Sedan

```
public class AsiaSedanCar extends AbstractCar {
    public AsiaSedanCar() {
        region = "Asia";
        type = "Sedan";
        engineVolume = "1.5L";
        color = "White";
        fuelType = "Hybrid";
    }
}
```

Asia Luxury

```
public class AsiaLuxuryCar extends AbstractCar {
    public AsiaLuxuryCar() {
        region = "Asia";
        type = "Luxury";
        engineVolume = "2.5L";
        color = "Champagne";
        fuelType = "Hybrid";
    }
}
```

Step-5: Abstract Factory Interface

```
public interface CarFactory {
    Car createSmallCar();
    Car createSedanCar();
```

```
    Car createLuxuryCar();
}
```

👉 এই interface guarantee করে যে
প্রত্যেক region সব car type বানাতে পারবে

🏭 Step-6: Concrete Factories

USA Factory

```
public class USACarFactory implements CarFactory {

    public Car createSmallCar() {
        return new USASmallCar();
    }

    public Car createSedanCar() {
        return new USASedanCar();
    }

    public Car createLuxuryCar() {
        return new USALuxuryCar();
    }
}
```

Asia Factory

```
public class AsiaCarFactory implements CarFactory {

    public Car createSmallCar() {
        return new AsiaSmallCar();
    }

    public Car createSedanCar() {
        return new AsiaSedanCar();
    }

    public Car createLuxuryCar() {
        return new AsiaLuxuryCar();
    }
}
```

▶️ Step-7: Client / Demo Class

```
public class AbstractFactoryDemo {

    public static void main(String[] args) {

        CarFactory usaFactory = new USACarFactory();
        Car usaSmall = usaFactory.createSmallCar();
        Car usaLuxury = usaFactory.createLuxuryCar();

        usaSmall.showSpecifications();
```

```
        usaLuxury.showSpecifications();

        CarFactory asiaFactory = new AsiaCarFactory();
        Car asiaSedan = asiaFactory.createSedanCar();

        asiaSedan.showSpecifications();
    }
}
```

Sample Output

```
Region: USA
Car Type: Small
Engine: 1.0L
Color: Red
Fuel: Petrol
----------------------
Region: USA
Car Type: Luxury
Engine: 5.0L
Color: White
Fuel: Petrol
----------------------
Region: Asia
Car Type: Sedan
Engine: 1.5L
Color: White
Fuel: Hybrid
----------------------
```

UML-Style Diagram (Text)

```
              CarFactory
        ------------------------
        |      |        |
   createSmall createSedan createLuxury
        |
  --------------------------
  |                        |
USACarFactory         AsiaCarFactory
  |                        |
USASmallCar           AsiaSmallCar
USASedanCar           AsiaSedanCar
USALuxuryCar          AsiaLuxuryCar
```

Chain of responsibility with bank

🧩 Step-1: Abstract Handler (Base Class)

```
public abstract class Approver {

    protected Approver nextApprover;

    public void setNextApprover(Approver nextApprover) {
        this.nextApprover = nextApprover;
    }

    public abstract void approve(double amount);
}
```

🔑 সব approver এর common structure

🟢 Step-2: Cashier (Concrete Handler)

```
public class Cashier extends Approver {

    @Override
    public void approve(double amount) {

        if (amount < 10000) {
            System.out.println("Cashier approved withdrawal of Tk. " +
amount);
        } else {
            System.out.println("Cashier forwarded request of Tk. " + amount);
            nextApprover.approve(amount);
        }
    }
}
```

🟢 Step-3: Senior Officer (Concrete Handler)

```
public class SeniorOfficer extends Approver {

    @Override
    public void approve(double amount) {

        if (amount >= 10000 && amount <= 1000000) {
            System.out.println("Senior Officer approved withdrawal of Tk. " +
amount);
        } else {
            System.out.println("Senior Officer forwarded request of Tk. " +
amount);
            nextApprover.approve(amount);
        }
    }
}
```

Step-4: Manager (Concrete Handler)

```
public class Manager extends Approver {

    @Override
    public void approve(double amount) {

        if (amount > 1000000) {
            System.out.println("Manager approved withdrawal of Tk. " +
amount);
        }
    }
}
```

Step-5: Chain Setup (Client / Demo)

```
public class ChainOfResponsibilityDemo {

    private static Approver createChain() {

        Approver cashier = new Cashier();
        Approver seniorOfficer = new SeniorOfficer();
        Approver manager = new Manager();

        cashier.setNextApprover(seniorOfficer);
        seniorOfficer.setNextApprover(manager);

        return cashier;   // chain starts here
    }

    public static void main(String[] args) {

        Approver approverChain = createChain();

        approverChain.approve(5000);
        approverChain.approve(50000);
        approverChain.approve(2000000);
    }
}
```

Output (Expected)

```
Cashier approved withdrawal of Tk. 5000

Cashier forwarded request of Tk. 50000
Senior Officer approved withdrawal of Tk. 50000

Cashier forwarded request of Tk. 2000000
Senior Officer forwarded request of Tk. 2000000
Manager approved withdrawal of Tk. 2000000
```

Step-by-Step Flow Explanation

- Case-1: Tk. 5,000

```
Cashier → approves → STOP
```

- Case-2: Tk. 50,000

```
Cashier → forwards
Senior Officer → approves → STOP
```

- Case-3: Tk. 20,00,000

```
Cashier → forwards
Senior Officer → forwards
Manager → approves → STOP
```

UML Class Diagram (Text Form)