Singleton

```
public class Singleton {
    private static volatile Singleton instance;

    private Singleton() {
        // Prevent reflection attack
        if (instance != null) {
            throw new IllegalStateException("Singleton already initialized");
        }
    }

    public static Singleton getInstance() {
        if (instance == null) {
            synchronized (Singleton.class) {
                if (instance == null) {
                    instance = new Singleton();
                }
            }
        }
        return instance;
    }

    // Prevent cloning attack
    @Override
    protected Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException("Cloning a Singleton is not allowed");
    }

    // Prevent deserialization attack
    protected Object readResolve() {
        return instance;
    }
}
```

Factory Design Pattern

```
    public static void main(String[] args) {
        Shape shape = ShapeFactory.getShape("Rectangle");
        shape.draw();
    }



interface Shape{
    public void draw();
}

class Circle implements Shape{

    @Override
    public void draw() {
        System.out.println("Circle is drawing");
    }
}

class Rectangle implements Shape{

    @Override
    public void draw() {
        System.out.println("Rectangle is drawing");
    }
}

class ShapeFactory{
    public static Shape getShape(String shape){
        if(shape.equalsIgnoreCase("Circle")){
            return new Circle();
        }
        else{
            return new Rectangle();
        }
    }
}
```

Builder Desgin Pattern

```
public class Main {
    public static void main(String[] args) {
        Pizza myPizza = new Pizza.Builder()
                .setBase("Thin Crust")
                .setSauce("Tomato Basil")
                .setToppings("Cheese, Olives, Jalapeno")
                .build();

        myPizza.display();
    }
}


// Product class
class Pizza {
    private String base;
    private String sauce;
    private String toppings;

    // Private constructor: only builder can create Pizza
    //Empty chod dena, ya banana mat
    private Pizza(Builder builder) {
        this.base = builder.base;
        this.sauce = builder.sauce;
        this.toppings = builder.toppings;
    }

    public void display() {
        System.out.println("Base: " + base);
        System.out.println("Sauce: " + sauce);
        System.out.println("Toppings: " + toppings);
    }

    // Builder Class
    public static class Builder {
        private String base;
        private String sauce;
        private String toppings;

        public Builder setBase(String base) {
            this.base = base;
            return this;
        }

        public Builder setSauce(String sauce) {
            this.sauce = sauce;
            return this;
        }

        public Builder setToppings(String toppings) {
            this.toppings = toppings;
            return this;
        }

        public Pizza build() {
            return new Pizza(this);
        }
    }
}
```