

**The Video URL:**

[**https://drive.google.com/file/d/15JpX9lVcCo\_Gg2ilGEDgJS2BRouJ30lt/view?usp=sharing**](https://drive.google.com/file/d/15JpX9lVcCo_Gg2ilGEDgJS2BRouJ30lt/view?usp=sharing)

**Shape Factory:**

interface ShapeFactory {

  Square createSquare();

  Circle createCircle();

  Rectangle createRectangle();

}

**Windows Shape Factory:**

class WindowsShapeFactory implements ShapeFactory {

  public Square createSquare() {

    return new WindowsSquare();

  }

  public Circle createCircle() {

    return new WindowsCircle();

  }

  public Rectangle createRectangle() {

    return new WindowsRectangle();

  }

}

**Linux Shape Factory:**

class LinuxShapeFactory implements ShapeFactory {

  public Square createSquare() {

    return new LinuxSquare();

  }

  public Rectangle createRectangle() {

    return new LinuxRectangle();

  }

  public Circle createCircle() {

    return new LinuxCircle();

  }

}

**Shape:**

public interface Shape {

  void draw();

  String getDescription();

}

**Square:**

class Square implements Shape {

  private Command command;

  public Square() {

    this.command = new BorderLightCommand(this);

    command.execute();

  }

  @Override

  public void draw() {

    System.out.println("Drawing a Square");

  }

  @Override

  public String getDescription() {

    return "This is a Square";

  }

}

**Circle:**

class Circle implements Shape {

  private Command command;

  public Circle() {

    this.command = new MotionCommand(this);

    command.execute();

  }

  @Override

  public void draw() {

    System.out.println("Drawing a Circle");

  }

  @Override

  public String getDescription() {

    return "This is a Circle";

  }

}

**Rectangle:**

class Rectangle implements Shape {

  @Override

  public void draw() {

    System.out.println("Drawing a Rectangle");

  }

  @Override

  public String getDescription() {

    return "This is a Rectangle";

  }

}

**Windows Circle:**

class WindowsCircle extends Circle {

  private Command command;

  public WindowsCircle() {

    super();

  }

  @Override

  public void draw() {

    System.out.println("Drawing a Windows-specific Circle");

  }

  @Override

  public String getDescription() {

    return "This is a Circle for Windows OS";

  }

}

**Windows Rectangle:**

class WindowsRectangle extends Rectangle {

  @Override

  public void draw() {

    System.out.println("Drawing a Windows-specific Rectangle");

  }

  @Override

  public String getDescription() {

    return "This is a Rectangle for Windows OS";

  }

}

**Windows Square:**

class WindowsSquare extends Square {

  private Command command;

  public WindowsSquare() {

    super();

  }

  @Override

  public void draw() {

    System.out.println("Drawing a Windows-specific Square");

  }

  @Override

  public String getDescription() {

    return "This is a Square for Windows OS";

  }

}

**Linux Circle:**

class LinuxCircle extends Circle {

  private Command command;

  public LinuxCircle() {

    super();

  }

  @Override

  public void draw() {

    System.out.println("Drawing a Linux-specific Circle");

  }

  @Override

  public String getDescription() {

    return "This is a Circle for Linux OS";

  }

}

**Linux Rectangle:**

class LinuxRectangle extends Rectangle {

  private Command command;

  public LinuxRectangle() {

  }

  @Override

  public void draw() {

    System.out.println("Drawing a Linux-specific Rectangle");

  }

  @Override

  public String getDescription() {

    return "This is a Rectangle for Linux OS";

  }

}

**Linux Square:**

public class LinuxSquare extends Square {

  private Command command;

  public LinuxSquare() {

    super();

  }

  @Override

  public void draw() {

    System.out.println("Drawing a Linux-specific Square");

  }

  @Override

  public String getDescription() {

    return "This is a Square for Linux OS";

  }

}

**Shape Decorator:**

public abstract class ShapeDecorator implements Shape {

  protected Shape decoratedShape;

  public ShapeDecorator(Shape shape) {

    this.decoratedShape = shape;

  }

  @Override

  public void draw() {

    decoratedShape.draw();

  }

  @Override

  public String getDescription() {

    return decoratedShape.getDescription();

  }

}

**Concrete class:**

1. **Border Color Decorator**

public class BorderColorDecorator extends ShapeDecorator {

  public BorderColorDecorator(Shape shape) {

    super(shape);

  }

  @Override

  public void draw() {

    decoratedShape.draw();

    System.out.println("Adding border color");

  }

  @Override

  public String getDescription() {

    return decoratedShape.getDescription() + " with border color";

  }

}

1. **Color Fill Decorator:**

public class ColorFillDecorator extends ShapeDecorator {

  public ColorFillDecorator(Shape shape) {

    super(shape);

  }

  @Override

  public void draw() {

    decoratedShape.draw();

    System.out.println("Filling with color");

  }

  @Override

  public String getDescription() {

    return decoratedShape.getDescription() + " with color fill";

  }

}

1. **Double Border Decorator:**
2. public class DoubleBorderDecorator extends ShapeDecorator {
3. public DoubleBorderDecorator(Shape shape) {
4. super(shape);
5. }
6. @Override
7. public void draw() {
8. decoratedShape.draw();
9. System.out.println("Adding double border size");
10. }
11. @Override
12. public String getDescription() {
13. return decoratedShape.getDescription() + " with double border size";
14. }
15. }

**Interface Command:**

public interface Command {

  void execute();

}

**Border Light Command:**

public class BorderLightCommand implements Command {

  private Shape shape;

  public BorderLightCommand(Shape shape) {

    this.shape = shape;

  }

  @Override

  public void execute() {

    System.out.println("Executing border light command for Square");

  }

}

**Motion Command:**

public class MotionCommand implements Command {

  private Shape shape;

  public MotionCommand(Shape shape) {

    this.shape = shape;

  }

  @Override

  public void execute() {

    System.out.println("Executing motion command for Circle");

  }

}

**SingleTon:**

class Singleton {

  private static Singleton inst;

  private Square square;

  private Rectangle rect;

  private Circle circle;

  private Singleton() {

  }

  public static synchronized Singleton getInstance() {

    if (inst == null) {

      inst = new Singleton();

    }

    return inst;

  }

  public Square getSquare(ShapeFactory shapeFactory) {

    if (square == null) {

      square = shapeFactory.createSquare();

    }

    return square;

  }

  public Rectangle getRectangle(ShapeFactory shapeFactory) {

    if (rect == null) {

      rect = shapeFactory.createRectangle();

    }

    return rect;

  }

  public Circle getCircle(ShapeFactory shapeFactory) {

    if (circle == null) {

      circle = shapeFactory.createCircle();

    }

    return circle;

  }

}

**TestDriver:**

public class TestDriver {

    public static void main(String[] args) {

        Singleton singleton = Singleton.getInstance();

        // Create a Windows factory

        ShapeFactory windowsFactory = new WindowsShapeFactory();

        // Create and configure a Windows Square with decorators

        Shape windowsSquare = singleton.getSquare(windowsFactory);

        windowsSquare = new ColorFillDecorator(windowsSquare);

        windowsSquare = new DoubleBorderDecorator(windowsSquare);

        windowsSquare.draw();

        System.out.println(windowsSquare.getDescription());

        System.out.println("==============================================================");

        // Create a Linux factory

        ShapeFactory linuxFactory = new LinuxShapeFactory();

        // Create and configure a Linux Circle with decorators

        Shape linuxCircle = singleton.getCircle(linuxFactory);

        linuxCircle = new BorderColorDecorator(linuxCircle);

        linuxCircle = new DoubleBorderDecorator(linuxCircle);

        linuxCircle.draw();

        System.out.println(linuxCircle.getDescription());

    }

}