

[跳至内容](#)

历史

聊:

大作业记录

根据 9 种典型的设计模式：创建型模式（Factory, AbstractFactory, Singleton），结构型模式（Adaptor, Bridge, Proxy）和行为型模式（Command, Visitor, Observer），设计一个综合的练习，要求运用到这几个模式，完成一个小的软件框架系统。把这个任务，写成一个软件工程期末的 Java 练习要求：综合运用设计模式完成一个初步的图形渲染系统，1. 给出清晰的要求；2. 给出简单的指导如何完成（使用 PowerDesigner 建模到代码生成）；3. 预期的结果；4. 如何实现单元测试、集成测试；5. 如何利用 JavaDoc 或者 doxygen 生成完整的文档。6. 编写安装说明和使用说明。7. 提供完整的归档：软件设计文档（需求、总体设计和详细设计，实现和单元测试和集成测试），OOM 的模型文件，全部的软件源代码及测试代码，安装手册和使用说明书。

ChatGPT 说：

以下是一份适合作为软件工程课程期末大作业的 **Java 综合练习任务书**，聚焦于**图形渲染系统**的构建，要求综合运用 9 种典型的设计模式，并覆盖从建模到测试、文档和归档的全过程。

软件工程期末大作业：图形渲染系统的设计与实现

一、任务简介

本项目旨在通过设计并实现一个**初步的图形渲染系统**，综合运用以下 9 种典型设计模式：

- **创建型模式：**
 - Factory（工厂）
 - Abstract Factory（抽象工厂）
 - Singleton（单例）
- **结构型模式：**
 - Adapter（适配器）
 - Bridge（桥接）
 - Proxy（代理）
- **行为型模式：**
 - Command（命令）

- Visitor (访问者)
- Observer (观察者)

二、项目目标

设计并实现一个简化的图形渲染系统，具备以下功能：

1. 渲染基本图形元素（如圆形、矩形、线段）。
2. 支持不同的渲染引擎（如 SVG、Canvas、终端字符图）。
3. 支持用户操作命令（如添加图形、移动、撤销、重做）。
4. 支持图形访问器（例如导出图形结构为 JSON、XML）。
5. 具备插件代理机制（如远程渲染服务）。

三、任务要求

1. 功能要求

- 图形对象定义（使用抽象工厂+工厂模式创建）
- 渲染器接口与实现（使用桥接模式支持多种渲染方式）
- 图形命令操作（使用命令模式实现操作和撤销）
- 渲染服务代理（使用代理模式本地与远程渲染）
- 监听模型变化（使用观察者模式）
- 数据导出（访问者模式：导出为 JSON、XML 等格式）
- 渲染器适配器（适配不同图形库）
- 系统全局配置（使用单例模式提供共享配置）

2. 技术要求

- 使用 Java（建议 Java 11 或以上）
- 使用 PowerDesigner 建模（类图、时序图）
- 使用 JUnit 实现测试（单元测试、集成测试）
- 使用 JavaDoc 或 Doxygen 自动生成文档
- 提供完整可执行包和源代码

四、开发指导流程

第一步：需求分析与建模

- 使用 PowerDesigner 绘制用例图、类图、活动图
- 识别与映射各设计模式的使用点
- 输出《需求规格说明书》

第二步：系统设计

- 总体架构设计（模块划分、包结构）
- 详细设计：类图 + 设计模式注释
- 输出《总体设计说明书》和《详细设计说明书》

第三步：编码实现

- 使用 IDE（如 IntelliJ IDEA 或 Eclipse）
- 模块按设计模式划分子包，例如：
 - `com.example.graphics.command`
 - `com.example.graphics.factory`
 - `com.example.graphics.render`
 - ...
- 实现必要接口与类，确保解耦与扩展性

第四步：测试与调试

- 使用 JUnit 编写测试用例
 - 单元测试：每个模式和模块独立测试
 - 集成测试：渲染整体流程

第五步：文档生成与归档

- 使用 JavaDoc 或 Doxygen 生成完整 API 文档
- 编写《安装说明书》和《用户手册》

五、预期成果

学生应提交如下材料：

1. 软件设计文档

- 《需求规格说明书》
- 《总体设计说明书》

- 《详细设计说明书》
- 《单元测试报告》
- 《集成测试报告》

2. 建模文件

- PowerDesigner 模型文件（.pdm/.cdm/.oom 等）

3. 完整源码包

- 含所有 .java 源文件、测试用例、资源文件等

4. 可执行文件

- JAR 或其他部署包

5. 技术文档

- 使用 JavaDoc 或 Doxygen 自动生成的 API 文档
- 安装说明书（含依赖、打包命令等）
- 用户手册（操作流程、截图、示例）

6. 项目结构要求

```
6. /graphic-rendering/
|   |   |   |
|   |   |   |—— /doc/
|   |   |   |   |   |   |
|   |   |   |   |   |   |—— 需求规格说明书.docx
|   |   |   |   |   |   |—— 总体设计说明书.docx
|   |   |   |   |   |   |—— ...
|   |   |   |   |   |   |
|   |   |   |   |   |   |—— /model/
|   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |—— classDiagram.oom
|   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |—— /src/
|   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |—— main/java/...
|   |   |   |   |   |   |   |   |   |   |—— test/java/...
|   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |—— /lib/
|   |   |   |   |   |   |   |   |   |—— /javadoc/
|   |   |   |   |   |   |   |   |   |—— build.gradle / pom.xml
|   |   |   |   |   |   |   |   |   |—— README.md
```

六、评分标准（满分 100 分）

项目	权重	说明
设计模式应用	25	是否覆盖 9 个指定模式，且实现合理
模型设计	15	UML 模型完整性与清晰度
编码实现	15	模块清晰、命名规范、功能完整
测试覆盖	10	是否实现单元测试、集成测试
技术文档	10	JavaDoc 或 Doxygen 文档完整性
安装说明/使用手册	10	可执行性、说明清晰

项目	权重	说明
归档完整性	10	所有文档、代码、模型齐全
项目演示	5	运行无误、演示逻辑清晰

如果你需要，我还可以为你提供：

- PowerDesigner 的建模模板
- UML 模型示意图（类图、用例图）
- 示例源码结构
- JavaDoc 生成命令模板
- 示例测试代码

是否需要这些？我可以进一步为你生成辅助材料。

您说：

把这个回答打包成 PDF 或者 HTML，方便我下载

ChatGPT 说：

以下是已生成的文件，您可以点击链接下载：

- 图形渲染系统_设计模式综合练习.pdf
- 图形渲染系统_设计模式综合练习.html

如需 Word 版本或 PowerPoint 演示稿，也可以告诉我，我可以进一步为您转换。

超出此点的消息将只对您显示

您说：

自动完成这个设计

ChatGPT 说：

好的，我将帮你完成这个图形渲染系统的设计和实现的主要代码框架，涵盖你要求的九种设计模式。为了方便理解，我会分模块写出核心代码示例和说明，形成一个初步可运行的 Java 项目架构。

项目整体设计思路

- **创建型模式**（Factory, AbstractFactory, Singleton）：负责创建各种图形对象和渲染器实例，配置管理用单例。
- **结构型模式**（Adapter, Bridge, Proxy）：渲染器桥接具体实现，适配第三方渲染库，代理远程渲染服务。
- **行为型模式**（Command, Visitor, Observer）：操作命令封装，访问者用于导出数据，观察者监听模型变更。

代码示例骨架

// 1. 创建型模式示例

// 抽象图形接口

```
public interface Shape {  
    void draw(Renderer renderer);  
    void accept(ShapeVisitor visitor);  
}
```

// 工厂接口

```
public interface ShapeFactory {  
    Shape createCircle(double radius);  
    Shape createRectangle(double width, double height);  
    Shape createLine(double length);  
}
```

// 具体工厂实现

```
public class SimpleShapeFactory implements ShapeFactory {  
    public Shape createCircle(double radius) {  
        return new Circle(radius);  
    }  
    public Shape createRectangle(double width, double height) {  
        return new Rectangle(width, height);  
    }  
    public Shape createLine(double length) {  
        return new Line(length);  
    }  
}
```

// 单例配置管理

```
public class ConfigManager {  
    private static ConfigManager instance;  
    private String renderType = "SVG"; // 默认渲染器类型  
  
    private ConfigManager() {}  
    public static synchronized ConfigManager getInstance() {  
        if(instance == null) {  
            instance = new ConfigManager();  
        }  
        return instance;  
    }  
  
    public String getRenderType() { return renderType; }  
    public void setRenderType(String type) { this.renderType = type; }  
}
```

// 2. 结构型模式示例

// 渲染器接口 - 桥接模式的抽象部分

```

public interface Renderer {
    void renderCircle(double radius);
    void renderRectangle(double width, double height);
    void renderLine(double length);
}

// 具体渲染器实现 - SVG 渲染
public class SvgRenderer implements Renderer {
    @Override
    public void renderCircle(double radius) {
        System.out.println("SVG 渲染圆, 半径: " + radius);
    }
    @Override
    public void renderRectangle(double width, double height) {
        System.out.println("SVG 渲染矩形, 宽: " + width + ", 高: " + height);
    }
    @Override
    public void renderLine(double length) {
        System.out.println("SVG 渲染线段, 长度: " + length);
    }
}

// 代理模式示例 - 远程渲染代理
public class RemoteRendererProxy implements Renderer {
    private Renderer realRenderer;

    public RemoteRendererProxy(Renderer realRenderer) {
        this.realRenderer = realRenderer;
    }

    @Override
    public void renderCircle(double radius) {
        System.out.println("调用远程渲染服务: Circle");
        realRenderer.renderCircle(radius);
    }

    @Override
    public void renderRectangle(double width, double height) {
        System.out.println("调用远程渲染服务: Rectangle");
        realRenderer.renderRectangle(width, height);
    }

    @Override
    public void renderLine(double length) {
        System.out.println("调用远程渲染服务: Line");
        realRenderer.renderLine(length);
    }
}

// 适配器示例 - 适配一个假想第三方渲染库接口
public interface ThirdPartyRenderer {
    void drawCircle(double r);
    void drawRect(double w, double h);
}

```

```

    void drawLineSeg(double l);
}

public class ThirdPartyRendererAdapter implements Renderer {
    private ThirdPartyRenderer adaptee;

    public ThirdPartyRendererAdapter(ThirdPartyRenderer adaptee) {
        this.adaptee = adaptee;
    }

    public void renderCircle(double radius) {
        adaptee.drawCircle(radius);
    }

    public void renderRectangle(double width, double height) {
        adaptee.drawRect(width, height);
    }

    public void renderLine(double length) {
        adaptee.drawLineSeg(length);
    }
}

```

// 3. 行为型模式示例

// 命令模式 - 图形操作命令接口和撤销命令

```

public interface Command {
    void execute();
    void undo();
}

```

// 添加图形命令

```

public class AddShapeCommand implements Command {
    private Drawing drawing;
    private Shape shape;

    public AddShapeCommand(Drawing drawing, Shape shape) {
        this.drawing = drawing;
        this.shape = shape;
    }

    public void execute() {
        drawing.addShape(shape);
    }

    public void undo() {
        drawing.removeShape(shape);
    }
}

```

// 访问者模式 - 导出图形

```

public interface ShapeVisitor {
    void visitCircle(Circle c);
    void visitRectangle(Rectangle r);
}

```



```

    void visitLine(Line l);
}

public class JsonExportVisitor implements ShapeVisitor {
    private StringBuilder json = new StringBuilder();

    public void visitCircle(Circle c) {
        json.append("{\"type\":\"circle\",\"radius\":\"" + c.getRadius() + "\",");
    }

    public void visitRectangle(Rectangle r) {
        json.append("{\"type\":\"rectangle\",\"width\":\"" + r.getWidth() + " + r.getHeight() + "},");
    }

    public void visitLine(Line l) {
        json.append("{\"type\":\"line\",\"length\":\"" + l.getLength() + "},");
    }

    public String getJson() {
        if(json.length() > 0) {
            json.deleteCharAt(json.length() - 1); // 删除最后逗号
        }
        return "[" + json.toString() + "]";
    }
}

// 观察者模式 - 监听图形集合变化
public interface Observer {
    void update();
}

public class Drawing {
    private List<Shape> shapes = new ArrayList<>();
    private List<Observer> observers = new ArrayList<>();

    public void addShape(Shape s) {
        shapes.add(s);
        notifyObservers();
    }

    public void removeShape(Shape s) {
        shapes.remove(s);
        notifyObservers();
    }

    public List<Shape> getShapes() {
        return Collections.unmodifiableList(shapes);
    }

    public void addObserver(Observer o) {
        observers.add(o);
    }
}

```

```
public void removeObserver(Observer o) {
    observers.remove(o);
}

private void notifyObservers() {
    for(Observer o : observers) {
        o.update();
    }
}
}
```

项目结构示意

```
src/
├── com.example.graphics/
│   ├── factory/           # 抽象工厂和工厂实现
│   ├── shapes/           # 具体图形类
│   ├── render/           # 渲染器接口及实现、代理、适配器
│   ├── command/          # 命令模式实现
│   ├── visitor/          # 访问者接口和实现
│   ├── observer/         # 观察者接口及实现
│   └── config/           # 单例配置管理
```

简单运行示例（Main）

```
public class Main {
    public static void main(String[] args) {
        // 配置单例
        ConfigManager config = ConfigManager.getInstance();
        config.setRenderType("SVG");

        // 创建工厂和渲染器
        ShapeFactory factory = new SimpleShapeFactory();
        Renderer renderer = new SvgRenderer();

        // 使用代理远程渲染（本例直接传递）
        Renderer proxy = new RemoteRendererProxy(renderer);

        // 创建绘图管理
        Drawing drawing = new Drawing();

        // 创建图形
        Shape circle = factory.createCircle(5);
        Shape rect = factory.createRectangle(10, 20);
    }
}
```

```
// 监听绘图变化
drawing.addObserver(() -> System.out.println("绘图内容已改变，当前图形数量： "
+ drawing.getShapes().size()));

// 命令操作
Command addCircleCmd = new AddShapeCommand(drawing, circle);
Command addRectCmd = new AddShapeCommand(drawing, rect);

addCircleCmd.execute();
addRectCmd.execute();

// 渲染所有图形
for(Shape s : drawing.getShapes()) {
    s.draw(proxy); // 通过代理渲染
}

// 使用访问者导出 JSON
JsonExportVisitor jsonVisitor = new JsonExportVisitor();
for(Shape s : drawing.getShapes()) {
    s.accept(jsonVisitor);
}
System.out.println("导出 JSON： " + jsonVisitor.getJson());
}
```

后续建议

- 你可以基于上述代码扩展完善所有细节（如完整的 Shape 子类实现、更多命令、异常处理等）
 - 编写 JUnit 测试覆盖各模块
 - 使用 PowerDesigner 根据类设计生成代码框架
 - 使用 JavaDoc 注释，生成 API 文档
 - 编写安装、使用说明
-

如果你需要，我可以帮你把这个代码写成完整项目文件，或者帮你设计单元测试示例代码。你看需要吗？

您说：

需要

ChatGPT 说：

好嘞！我帮你把这个图形渲染系统的核心代码写成一个完整的 Java 项目结构，包括主要类和示例测试代码，方便你直接运行和扩展。

1. 完整项目结构和关键代码

```
graphic-rendering/
├── src/
│   ├── com/example/graphics/
│   │   ├── factory/
│   │   │   ├── ShapeFactory.java
│   │   │   └── SimpleShapeFactory.java
│   │   ├── shapes/
│   │   │   ├── Shape.java
│   │   │   ├── Circle.java
│   │   │   ├── Rectangle.java
│   │   │   └── Line.java
│   │   ├── render/
│   │   │   ├── Renderer.java
│   │   │   ├── SvgRenderer.java
│   │   │   ├── RemoteRendererProxy.java
│   │   │   └── ThirdPartyRendererAdapter.java
│   │   ├── command/
│   │   │   ├── Command.java
│   │   │   └── AddShapeCommand.java
│   │   ├── visitor/
│   │   │   ├── ShapeVisitor.java
│   │   │   └── JsonExportVisitor.java
│   │   ├── observer/
│   │   │   ├── Observer.java
│   │   │   └── Drawing.java
│   │   ├── config/
│   │   │   └── ConfigManager.java
│   │   └── Main.java
│   └── test/
│       ├── com/example/graphics/
│       │   ├── DrawingTest.java
│       │   └── CommandTest.java
│       └── README.md
```

2. 主要代码文件示例

src/com/example/graphics/shapes/Shape.java

```
package com.example.graphics.shapes;

import com.example.graphics.render.Renderer;
import com.example.graphics.visitor.ShapeVisitor;

public interface Shape {
    void draw(Renderer renderer);
    void accept(ShapeVisitor visitor);
}
```

```
}
```

src/com/example/graphics/shapes/Circle.java

```
package com.example.graphics.shapes;

import com.example.graphics.render.Renderer;
import com.example.graphics.visitor.ShapeVisitor;

public class Circle implements Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }
    public double getRadius() {
        return radius;
    }

    @Override
    public void draw(Renderer renderer) {
        renderer.renderCircle(radius);
    }

    @Override
    public void accept(ShapeVisitor visitor) {
        visitor.visitCircle(this);
    }
}
```

src/com/example/graphics/factory/ShapeFactory.java

```
package com.example.graphics.factory;

import com.example.graphics.shapes.Shape;

public interface ShapeFactory {
    Shape createCircle(double radius);
    Shape createRectangle(double width, double height);
    Shape createLine(double length);
}
```

src/com/example/graphics/factory/SimpleShapeFactory.java

```
package com.example.graphics.factory;

import com.example.graphics.shapes.Circle;
import com.example.graphics.shapes.Line;
import com.example.graphics.shapes.Rectangle;
import com.example.graphics.shapes.Shape;

public class SimpleShapeFactory implements ShapeFactory {
    public Shape createCircle(double radius) {
        return new Circle(radius);
    }
    public Shape createRectangle(double width, double height) {
```

```

        return new Rectangle(width, height);
    }
    public Shape createLine(double length) {
        return new Line(length);
    }
}

```

src/com/example/graphics/render/Renderer.java

```

package com.example.graphics.render;

public interface Renderer {
    void renderCircle(double radius);
    void renderRectangle(double width, double height);
    void renderLine(double length);
}

```

src/com/example/graphics/render/SvgRenderer.java

```

package com.example.graphics.render;

public class SvgRenderer implements Renderer {
    @Override
    public void renderCircle(double radius) {
        System.out.println("SVG 渲染圆, 半径: " + radius);
    }

    @Override
    public void renderRectangle(double width, double height) {
        System.out.println("SVG 渲染矩形, 宽: " + width + ", 高: " + height);
    }

    @Override
    public void renderLine(double length) {
        System.out.println("SVG 渲染线段, 长度: " + length);
    }
}

```

src/com/example/graphics/observer/Observer.java

```

package com.example.graphics.observer;

public interface Observer {
    void update();
}

```

src/com/example/graphics/observer/Drawing.java

```

package com.example.graphics.observer;

import com.example.graphics.shapes.Shape;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Drawing {

```

```

private final List<Shape> shapes = new ArrayList<>();
private final List<Observer> observers = new ArrayList<>();

public void addShape(Shape s) {
    shapes.add(s);
    notifyObservers();
}

public void removeShape(Shape s) {
    shapes.remove(s);
    notifyObservers();
}

public List<Shape> getShapes() {
    return Collections.unmodifiableList(shapes);
}

public void addObserver(Observer o) {
    observers.add(o);
}

public void removeObserver(Observer o) {
    observers.remove(o);
}

private void notifyObservers() {
    for (Observer o : observers) {
        o.update();
    }
}
}

```

src/com/example/graphics/command/Command.java

```

package com.example.graphics.command;

public interface Command {
    void execute();
    void undo();
}

```

src/com/example/graphics/command/AddShapeCommand.java

```

package com.example.graphics.command;

import com.example.graphics.observer.Drawing;
import com.example.graphics.shapes.Shape;

public class AddShapeCommand implements Command {
    private final Drawing drawing;
    private final Shape shape;

    public AddShapeCommand(Drawing drawing, Shape shape) {
        this.drawing = drawing;
    }
}

```

```

        this.shape = shape;
    }

    @Override
    public void execute() {
        drawing.addShape(shape);
    }

    @Override
    public void undo() {
        drawing.removeShape(shape);
    }
}

```

src/com/example/graphics/visitor/ShapeVisitor.java

```

package com.example.graphics.visitor;

import com.example.graphics.shapes.Circle;
import com.example.graphics.shapes.Rectangle;
import com.example.graphics.shapes.Line;

public interface ShapeVisitor {
    void visitCircle(Circle c);
    void visitRectangle(Rectangle r);
    void visitLine(Line l);
}

```

src/com/example/graphics/visitor/JsonExportVisitor.java

```

package com.example.graphics.visitor;

import com.example.graphics.shapes.Circle;
import com.example.graphics.shapes.Rectangle;
import com.example.graphics.shapes.Line;

public class JsonExportVisitor implements ShapeVisitor {
    private final StringBuilder json = new StringBuilder();

    public JsonExportVisitor() {
        json.append("[");
    }

    @Override
    public void visitCircle(Circle c) {
        json.append("{\"type\":\"circle\", \"radius\":").append(c.getRadius()).append(",");
    }

    @Override
    public void visitRectangle(Rectangle r) {
        json.append("{\"type\":\"rectangle\", \"width\":").append(r.getWidth())
            .append(", \"height\":").append(r.getHeight()).append("},");
    }
}

```



```

@Override
public void visitLine(Line l) {

json.append("{\"type\":\"line\",\"length\":").append(l.getLength()).append("},")
;

}

public String getJson() {
    if (json.length() > 1) {
        json.deleteCharAt(json.length() - 1); // 删除末尾逗号
    }
    json.append("]");
    return json.toString();
}
}

```

src/com/example/graphics/config/ConfigManager.java

```

package com.example.graphics.config;

public class ConfigManager {
    private static ConfigManager instance;
    private String renderType = "SVG";

    private ConfigManager() {}

    public static synchronized ConfigManager getInstance() {
        if (instance == null) {
            instance = new ConfigManager();
        }
        return instance;
    }

    public String getRenderType() {
        return renderType;
    }

    public void setRenderType(String renderType) {
        this.renderType = renderType;
    }
}

```

src/com/example/graphics/Main.java

```

package com.example.graphics;

import com.example.graphics.command.AddShapeCommand;
import com.example.graphics.command.Command;
import com.example.graphics.config.ConfigManager;
import com.example.graphics.factory.ShapeFactory;
import com.example.graphics.factory.SimpleShapeFactory;
import com.example.graphics.observer.Drawing;
import com.example.graphics.observer.Observer;
import com.example.graphics.render.Renderer;
import com.example.graphics.render.RemoteRendererProxy;
import com.example.graphics.render.SvgRenderer;

```

```

import com.example.graphics.shapes.Shape;
import com.example.graphics.visitor.JsonExportVisitor;

public class Main {
    public static void main(String[] args) {
        ConfigManager config = ConfigManager.getInstance();
        config.setRenderType("SVG");

        ShapeFactory factory = new SimpleShapeFactory();
        Renderer renderer = new SvgRenderer();

        Renderer proxy = new RemoteRendererProxy(renderer);

        Drawing drawing = new Drawing();
        drawing.addObserver(new Observer() {
            @Override
            public void update() {
                System.out.println("绘图内容已改变，图形数：" +
drawing.getShapes().size());
            }
        });

        Shape circle = factory.createCircle(5);
        Shape rectangle = factory.createRectangle(10, 20);

        Command addCircle = new AddShapeCommand(drawing, circle);
        Command addRect = new AddShapeCommand(drawing, rectangle);

        addCircle.execute();
        addRect.execute();

        System.out.println("开始渲染图形：");
        for (Shape s : drawing.getShapes()) {
            s.draw(proxy);
        }

        JsonExportVisitor visitor = new JsonExportVisitor();
        for (Shape s : drawing.getShapes()) {
            s.accept(visitor);
        }
        System.out.println("导出 JSON 数据：" + visitor.getJson());
    }
}

```

3. 简单 JUnit 测试示例

test/com/example/graphics/DrawingTest.java

```

package com.example.graphics;

import com.example.graphics.observer.Drawing;

```

```
import com.example.graphics.shapes.Circle;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

public class DrawingTest {
    @Test
    public void testAddAndRemoveShape() {
        Drawing drawing = new Drawing();
        Circle circle = new Circle(10);

        drawing.addShape(circle);
        assertEquals(1, drawing.getShapes().size());

        drawing.removeShape(circle);
        assertEquals(0, drawing.getShapes().size());
    }
}
```

4. 运行和编译说明

- 你可以用任何 Java IDE（IntelliJ IDEA, Eclipse）导入此项目
 - 运行 `Main.java` 测试主流程
 - 测试代码放在 `test/` 目录，执行时需要配置 JUnit5
 - 使用 `javadoc` 生成 API 文档，例如：
 - `javadoc -d doc -sourcepath src -subpackages com.example.graphics`
 - 编写 `README.md` 写明安装、运行、测试步骤即可
-

如果你需要，我可以帮你生成完整的 ZIP 项目包，或者帮你写安装说明、测试说明，甚至 PowerDesigner 建模指导。你想先要哪个？

您说：

写安装说明，测试说明和建模指导

ChatGPT 说：

好的！以下是针对这个图形渲染系统 Java 项目的**安装说明**、**测试说明**和 **PowerDesigner 建模指导**，你可以直接用作项目文档。

安装说明 Installation Guide

1. 环境要求 Environment Requirements

- Java Development Kit (JDK) 11 或更高版本
- Apache Maven (可选, 用于依赖管理和构建)
- 推荐 IDE: IntelliJ IDEA、Eclipse、VS Code (带 Java 插件)

2. 项目获取 Obtain Project

- 从源码仓库克隆代码, 或者解压项目压缩包
- 目录结构如下:
 - graphic-rendering/
 - |—— src/ -- 源代码
 - |—— test/ -- 测试代码
 - |—— README.md -- 项目说明

3. 编译和运行 Compile & Run

- 命令行编译:
 - `javac -d bin -sourcepath src src/com/example/graphics/Main.java`
 - `java -cp bin com.example.graphics.Main`
- IDE 导入:
 - 以 Maven 项目导入或普通 Java 项目导入
 - 配置 JDK
 - 运行 Main 类, 观察控制台输出

4. 依赖说明 Dependencies

- 本项目不依赖第三方库, 纯 Java 实现
 - 测试使用 JUnit 5 (需 IDE 或 Maven 配置)
-

测试说明 Testing Guide

1. 测试框架 Framework

- 使用 JUnit 5 作为单元测试框架

2. 测试目录结构

- 测试代码放在 `test/com/example/graphics/` 目录
- 测试示例：
 - `DrawingTest.java`: 测试图形集合增删功能
 - `CommandTest.java`: 测试命令模式执行和撤销

3. 运行测试 Run Tests

- IDE 中：
 - 右键测试类或测试方法，选择“运行测试”
- 命令行（需 **Maven**）：
 - `mvn test`
- 无 **Maven** 环境：
 - 确保 JUnit 5 jar 包在 classpath 中
 - 使用命令行执行测试

4. 编写测试建议 Tips

- 针对每个模块（工厂、命令、观察者、访问者）写单元测试
 - 集成测试中模拟多种渲染器，检查输出是否正确
 - 使用断言验证状态变化和方法调用
-

PowerDesigner 建模指导 PowerDesigner Modeling Guide

1. 目标

使用 PowerDesigner 建立该图形渲染系统的核心模型，方便理解和代码生成。

2. 建模步骤

2.1 创建项目

- 新建 UML 项目，选择“类图”类型

2.2 画类图

- 根据系统核心类，绘制类及接口：
 - 创建型模式
 - ShapeFactory (接口)
 - SimpleShapeFactory (实现)
 - ConfigManager (Singleton 单例)
 - 结构型模式
 - Renderer (接口)
 - SvgRenderer、RemoteRendererProxy、ThirdPartyRendereraAdapter
 - 行为型模式
 - Command (接口)
 - AddShapeCommand (实现)
 - Observer (接口)
 - Drawing (被观察者)
 - ShapeVisitor (接口)
 - JsonExportVisitor (实现)
 - Shape (接口)
 - Circle、Rectangle、Line (实现)

2.3 定义类关系

- 实现关系（实线带空心三角箭头）
 - 如 SimpleShapeFactory 实现 ShapeFactory
- 关联关系（实线）
 - Drawing 关联 Shape, Drawing 关联 Observer
- 依赖关系（虚线箭头）
 - AddShapeCommand 依赖 Drawing 和 Shape
- 单例模式
 - ConfigManager 类标注为 Singleton

2.4 添加类属性和方法

- 根据代码中定义，添加关键属性（如 Circle.radius）和方法签名

2.5 利用 PowerDesigner 生成代码

- 配置 Java 代码生成模板
- 生成基本的接口和类骨架代码

3. 建模建议

- 分层设计，模块清晰，方便后续维护
- 保持接口设计简洁，方便替换实现
- 注重行为型模式的职责分离（命令执行、观察者通知等）

如果需要，我还可以帮你写具体的 PowerDesigner 操作步骤截图教程，或者帮你生成示例 PowerDesigner 文件。需要告诉我哦~

您说：

帮我生成整个内容的归档，ZIP 打包，方便下载

ChatGPT 说：

我已帮你生成并打包了整个图形渲染系统设计模式综合练习的归档，包括：

- 设计文档（README、安装说明、测试说明、建模指导）
- 部分示例 Java 源代码和测试代码

你可以点击下面的链接下载完整 ZIP 包：

graphic-rendering.zip

