《高级程序设计》项目二：简易绘图板设计与实现报告

一、项目概述

1.1 项目背景

本项目旨在开发一款基于 Windows 平台的简易矢量绘图应用程序。通过该项目的开发，深入理解 MFC 应用程序框架，掌握 GDI/GDI+绘图技术，并实践面向对象设计模式（工厂模式、命令模式、策略模式）在实际软件工程中的应用。项目重点解决图形对象的动态创建、交互绘制以及撤销/重做（Undo/Redo）功能的实现难点。

1.2 项目目标

构建基于 MFC Doc/View 架构的单文档应用程序 。

实现基本的图形绘制功能（直线、矩形、椭圆、自由曲线）。

实现完善的交互体验，包括双缓冲防闪烁、属性设置（颜色、线宽）及多语言界面。

实现基于命令模式的无限次撤销与重做功能 。

保证代码的高质量与健壮性，包括异常处理、RAII 资源管理及单元测试覆盖。

二、需求分析

2.1 功能性需求

根据项目任务书，系统需满足以下核心功能：

F-01 画布初始化：提供绘图区域，采用双缓冲技术（Double Buffering）消除绘制闪烁。

F-02 基本图形绘制：

支持直线、矩形、椭圆的标准几何图形绘制。

支持自由曲线（Freehand）绘制。

支持“鼠标按下开始-移动预览-松开完成”的交互逻辑。

F-03 属性设置：

支持颜色选择（调用 Windows 颜色对话框）。

支持线宽调节（1px, 3px, 5px, 8px）。

F-04 撤销/重做：支持操作的历史记录回溯，可撤销或重做任意绘图步骤。

F-07 错误处理：文件读写异常捕获，GDI 资源自动释放。

F-10 多语言支持：支持中文、英文、日文三种界面语言的实时切换。

2.2 非功能性需求

NF-02 性能：绘制延迟低，文件 I/O 响应迅速 。

NF-03 可维护性：采用 MVC 分层架构，代码包含完整的 Doxygen 注释 。

NF-04 安全性：使用 RAII 包装 GDI 对象，防止资源泄漏；禁止硬编码路径。

NF-09 测试：核心逻辑单元测试覆盖率 > 70%。

三、系统设计

3.1 总体架构

本项目采用经典的 MFC Document/View (文档/视图) 架构，严格遵循 MVC 设计原则：

Model (CMyDrawBoardDoc)：负责数据存储。维护图形对象列表 (std::vector<std::shared_ptr<IShape>>)，负责数据的序列化（保存/读取）。

View (CMyDrawBoardView)：负责界面渲染与用户交互。处理鼠标消息，调用 GDI 绘图，并负责双缓冲的具体实现。

Controller (CommandManager)：负责业务逻辑调度。管理撤销栈和重做栈，解耦用户的操作请求与具体执行。

3.2 类设计与接口

系统核心抽象层为 IShape 接口，定义了所有图形对象的通用行为：

Draw(CDC* pDC): 绘制策略。

SetStartPoint/SetEndPoint: 坐标设置。

Save/Load: 序列化接口。

具体图形类 CLineShape, CRectShape, CEllipseShape, CFreehandShape 均继承自 IShape 并实现上述接口。

3.3 设计模式应用

工厂模式 (Factory Pattern)：

应用：ShapeFactory 类。

作用：根据 ShapeType 枚举动态创建具体的 IShape 对象，实现了对象创建与使用的解耦 。

命令模式 (Command Pattern)：

应用：IDrawCommand 接口及其子类 CAddShapeCommand，以及 CommandManager。

作用:将“绘图”这一动作封装为对象,使得操作可以被存储、撤销和重做 。

策略模式 (Strategy Pattern)：

应用：IShape::Draw()。

作用：不同的图形拥有不同的绘制算法，View 层只需调用统一接口，无需关心具体实现细节。

四、关键技术实现

4.1 双缓冲绘图

为解决屏幕闪烁问题，在 OnDraw 函数中实现了内存绘图：

创建兼容内存 DC (CreateCompatibleDC)和位图 (CreateCompatibleBitmap)。

在内存 DC 上完成背景填充和所有图形的绘制。

使用 BitBlt 函数一次性将内存画面拷贝至屏幕 DC。

4.2 RAII 资源管理

设计了模板类 CGdiObjectWrapper<T>，在构造函数中接管 GDI 对象指针，在析构函数中自动调用 DeleteObject。确保了在发生异常或函数返回时,GDI 资源（如 CPen, CBrush）一定会被释放，满足 F-07 需求。

4.3 序列化与异常安全

重写 CDocument::Serialize 函数，采用二进制流格式存储数据。引入

try-catch 块捕获 CArchiveException，确保在读取损坏文件时程序不会崩溃，而是弹出友好的错误提示框。

4.4 多语言动态切换

通过 UpdateMenuText 函数实现语言切换。根据当前选中的语言枚举值（Chinese/English/Japanese），动态调用 CMenu::ModifyMenu 修改菜单项的文本字符串，并刷新菜单栏，满足 F-10 需求。

五、测试与质量保证

5.1 单元测试

引入 GoogleTest 框架建立独立的测试项目 MyDrawBoardTests。针对核心逻辑编写了测试用例：

ShapeFactoryTest：验证工厂类能否正确创建不同类型的图形对象。

SerializerTest：使用 CMemFile 模拟内存文件，验证 IShape::Save 和 Load 的数据一致性。

CommandPatternTest：模拟绘图操作入栈、撤销、重做流程，验证画布状态的正确性。测试结果显示全通过，核心模块代码覆盖率超过 70%。

5.2 功能测试

经过手动测试，所有图形绘制流畅，撤销重做逻辑正确，文件保存读取无误，多语言切换即时生效。

六、总结与展望

6.1 项目成果

本项目成功实现了一个功能完备的简易绘图板，不仅满足了所有功能性需求（F-01 至 F-10），还在架构设计上达到了高内聚低耦合的要求。通过引入设计模式和单元测试，代码具有良好的可维护性和扩展性。

6.2 AI 辅助编程反思

在项目开发中，合理使用了 AI 工具辅助生成了工厂模式骨架和序列化逻辑代码。通过人工代码审查（Code Review），修正了部分 GDI 资源释放的潜在风险，并补全了多语言支持的细节。这不仅提高了开发效率，也加深了对代码质量控制的理解。但 AI 生成的变量与函数往往不是特别直观，乱改动又不便于 AI 进行 Debug，应找到两者之间的平衡。