实现多风格图形绘制

一、功能目标

实现一个支持**多风格图形家族**的绘制系统，要求：

1. 能创建不同风格（如 `经典风格`、`现代风格`）的图形家族，每个家族包含圆形、方形等图形
2. 客户端代码与具体图形类和风格实现完全解耦
3. 新增风格家族时无需修改已有代码

二、核心功能描述

1. **抽象产品类**
   - 图形基类（`Shape`）
     - 纯虚方法 `void draw() const`：定义绘制行为
     - 虚析构函数保证多态安全
   - 具体图形类型：
     - 圆形（`Circle`）、方形（`Rectangle`）
2. **抽象工厂接口（`GUIFactory`）**
   - 定义创建同一风格家族中所有图形的方法：

     ```
     virtual Circle* createCircle() const = 0;
     virtual Rectangle* createRectangle() const = 0;
     ```

   - 虚析构函数保证工厂多态安全
3. **具体工厂实现**
   - 经典风格工厂（`ClassicFactory`）
     - 创建经典风格的图形：
       - `ClassicCircle`：`draw()` 输出 `"Drawing Classic Circle"`
       - `ClassicRectangle`：`draw()` 输出 `"Drawing Classic Rectangle"`
   - 现代风格工厂（`ModernFactory`）
     - 创建现代风格的图形：
       - `ModernCircle`：`draw()` 输出 `"Drawing Modern Circle"`
       - `ModernRectangle`：`draw()` 输出 `"Drawing Modern Rectangle"`
4. **客户端逻辑**
   - 通过传入不同的工厂对象，生成同一风格家族的图形
   - 示例代码：

```
GUIFactory* factory = new ModernFactory();
Circle* circle = factory->createCircle();
Rectangle* rect = factory->createRectangle();
circle->draw();  // 输出 "Drawing Modern Circle"
// 必须手动释放资源
delete circle;
delete rect;
delete factory;
```

三、扩展练习任务

1. **任务 1：新增复古风格家族**
   - 实现 RetroFactory工厂类：
     - `RetroCircle`：`draw()` 输出 `"Drawing Retro Circle"`
     - `RetroRectangle`：`draw()` 输出 `"Drawing Retro Rectangle"`
2. **任务 3：跨平台渲染支持**
   - 扩展抽象工厂支持不同平台（如 Windows/Linux）的图形：
     - `WindowsCircle` 输出 `"Drawing Windows Circle"`
     - `LinuxRectangle` 输出 `"Drawing Linux Rectangle"`

四、技术约束

1. **代码结构要求**
   - 每个风格家族独立成类（如 `ClassicCircle` 与 `ModernCircle` 无耦合）
   - 客户端代码中禁止直接使用具体类名（如 `new ClassicCircle()`）
2. **内存管理要求**
   - 必须手动管理内存（使用 `new`/`delete`）
   - 所有对象需在销毁前显式释放

五、测试要求

基础功能测试

```
void testModernStyle() {
    GUIFactory* factory = new ModernFactory();
    Circle* circle = factory->createCircle();
    assert(circle->draw() == "Drawing Modern Circle");
    delete circle;
    delete factory;
}
```

六、交付要求

1. **代码提交**
   - 实现至少 2 个风格家族（经典/现代）
   - 每个家族需包含 `Circle` 和 `Rectangle`
2. **设计文档**
   - 类图（需体现抽象工厂与产品家族的关系）
   - 对比工厂方法模式与抽象工厂模式的区别

> **提示**：
>
> 1. 优先定义抽象接口（`GUIFactory`、`Shape`）再实现具体类