目录

第11章 Class Design

11.1 类的设计要求

1. 要易于理解，易于维护，易于重用
2. 要做些什么
    1. 我们需要多少种类?
    2. 何时定义成一个类?
    3. 类中有什么接口和数据?
    4. 我们需要构造继承来促进接口和代码重用吗?
    5. 哪个函数应该是**virtual**的，以支持运行时的动态绑定?
3. 考量内容
    1. **Responsibility-driven design**：职责驱动的设计
    2. **Coupling**：耦合
    3. **Cohesion**：内聚
    4. **Refactoring**：重构

11.2 代码质量

11.2.1 Coupling 耦合

Coupling耦合：指的是一个程序中，不同单元之间的联系

1. 如果两个类紧密地依赖于彼此的许多细节，我们就说它们是紧密耦合**tightly coupled**的
2. 我们的目标是：尽可能松耦合**Loose coupling**
3. 直观表述：如果X改变了，在Y中我们需要修改多少代码

Loose coupling松耦合：

1. 理解类X，我们不需要理解类Y
2. 改变类X，我们不需要改变类Y
3. 因此会提高可维护性**maintainability**

实现松耦合的方法：

1. 回调函数**call-back**：通过接口实现调用
2. 消息机制**message mech**

11.2.2 Cohesion 内聚

Cohesion内聚：指的是一个单元，需要负责的任务的数量和多样性

1. 如果**每个单元负责一项逻辑任务**，我们就说它具有高内聚
2. 内聚适用于类和方法
3. 我们的目标是高内聚**hign cohension**

hign cohension高内聚的优点：

1. 更容易理解类或方法的作用
2. 更容易使用描述性的名字
3. 更容易重用类或方法

方法的内聚性：

1. 一个**方法**应该负责且仅负责**一个定义良好的任务**

类的内聚性：

1. **类**应该表示一个定义良好的**单一实体**

11.2.3 Code duplication 代码重复

1. 是糟糕设计的标志
2. 使得维护困难
3. 可能导致维修过程中出现错误

11.2.4 Responsibility-driven design 职责驱动的设计

1. 问题：我们应该在哪里添加一个新方法(哪个类)
2. 每个类都应该负责操作自己的数据
3. 拥有数据的类应该负责处理数据
4. RDD可以实现低耦合

11.2.5 Localizing change 局部化修改

1. 减少耦合和责任驱动设计的一个目标是将更改本地化
2. 当需要**更改时，受影响的类越少越好**

11.2.6 Thinking ahead 提前思考

1. 在设计类时，我们试图**思考将来可能会做出哪些更改**
2. 我们的目标是让这些更改变得容易

11.2.7 Refactoring 重构

Refactoring重构：

1. 维护类时，通常会添加代码
2. 类和方法变得越来越长
3. 应该不时地重构类和方法，以保持内聚和低耦合

Refactoring and testing重构和测试：

1. 在重构代码时，要将重构与其他更改分开

2. 首先只进行重构，不改变功能
3. 在重构之前和之后进行测试，以确保没有任何问题

11.3 设计的问题

通性问题：

1. 一个类需要多长
2. 一个方法需要多长

设计的准则

1. 如果一个方法执行多个逻辑任务，那么它就太长了
2. 如果一个类表示多个逻辑实体，那么它就太复杂了
3. 注意：这些是指导原则—它们仍然给设计师留下了很大的空间。