设计模式课程设计报告

目录

- 课程设计背景与目的
- 常见设计模式介绍及分类
- 设计模式在软件开发中应用案例分析

目录

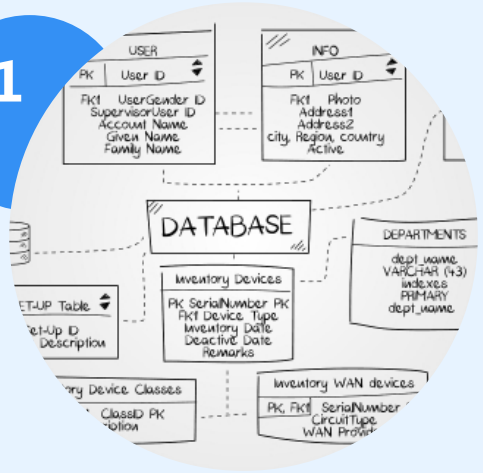
- 设计模式选择原则及实践方法论述
- 课程设计成果展示及评价
- 总结反思与未来展望

01

课程设计背景与目的

设计模式概念及重要性

01



设计模式定义

设计模式是在特定环境下，为解决某一类问题而提出的经过验证的、可复用的解决方案。

02

设计模式重要性

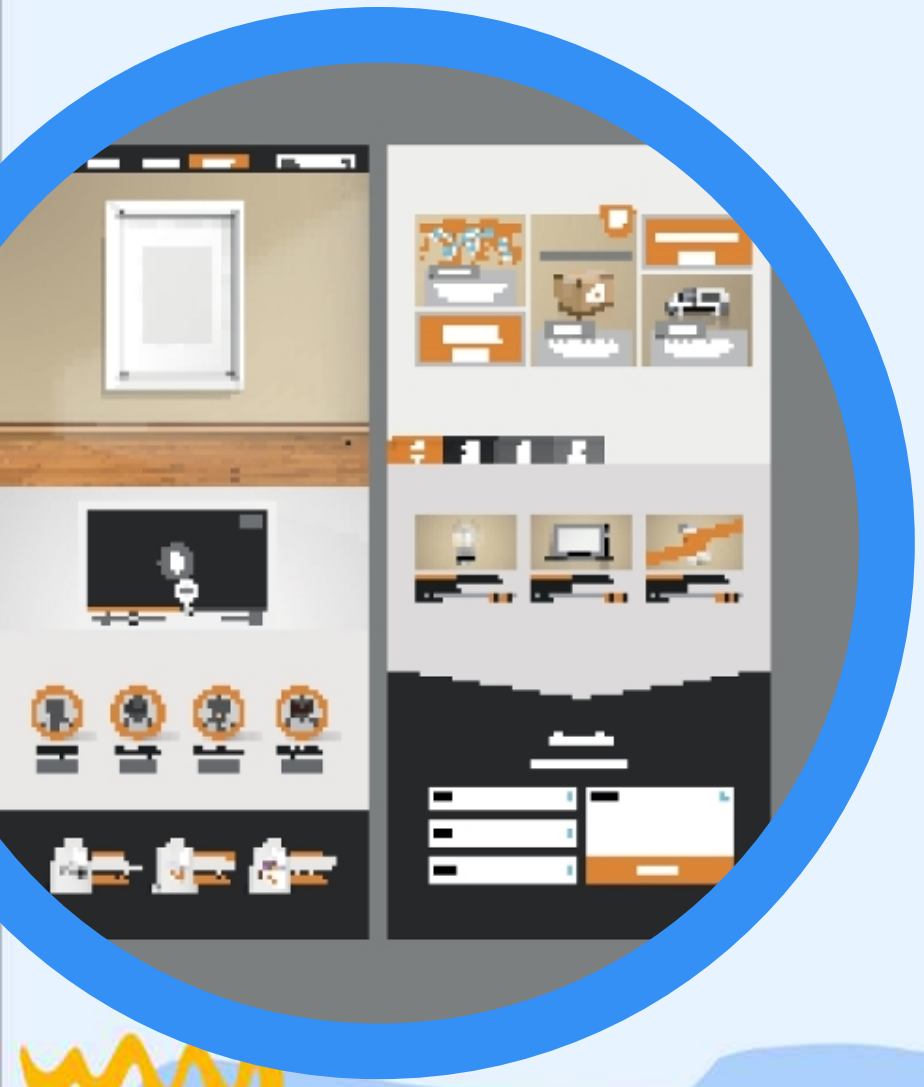
设计模式可以提高代码的可重用性、可读性和可维护性，降低软件开发的复杂度和成本。

03

常见设计模式

包括单例模式、工厂模式、观察者模式、策略模式等。

课程设计目标与要求



01

课程目标

通过课程设计，使学生掌握常见的设计模式，理解其原理和应用场景，提高面向对象编程能力。

02

课程要求

学生需要完成一系列设计模式的实现和应用任务，包括分析、设计、编码和测试等环节。

03

评价标准

课程设计成果的评价标准包括实现的正确性、代码质量、文档完整性和创新性等。

面向对象编程思想在设计模式中应用

01

面向对象编程基础

面向对象编程是一种基于类和对象的编程范式，通过封装、继承和多态等机制实现代码的重用和扩展。

02

设计模式与面向对象编程关系

设计模式是面向对象编程思想的重要体现，通过运用设计模式可以更好地实现面向对象编程的原则和优势。

03

设计模式中的面向对象原则

包括单一职责原则、开放封闭原则、里氏替换原则、依赖倒置原则、接口隔离原则等，这些原则在设计模式中得到了广泛的应用和体现。

02

常见设计模式介绍及分类

创建型设计模式

01

工厂方法模式（Fact...

定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法使一个类的实例化延迟到其子类。

02

抽象工厂模式（Abst...

提供一个接口，用于创建相关或依赖对象的家族，而不需要明确指定具体类。

03

单例模式（Single...

确保一个类仅有一个实例，并提供一个访问它的全局访问点。

04

建造者模式（Build...

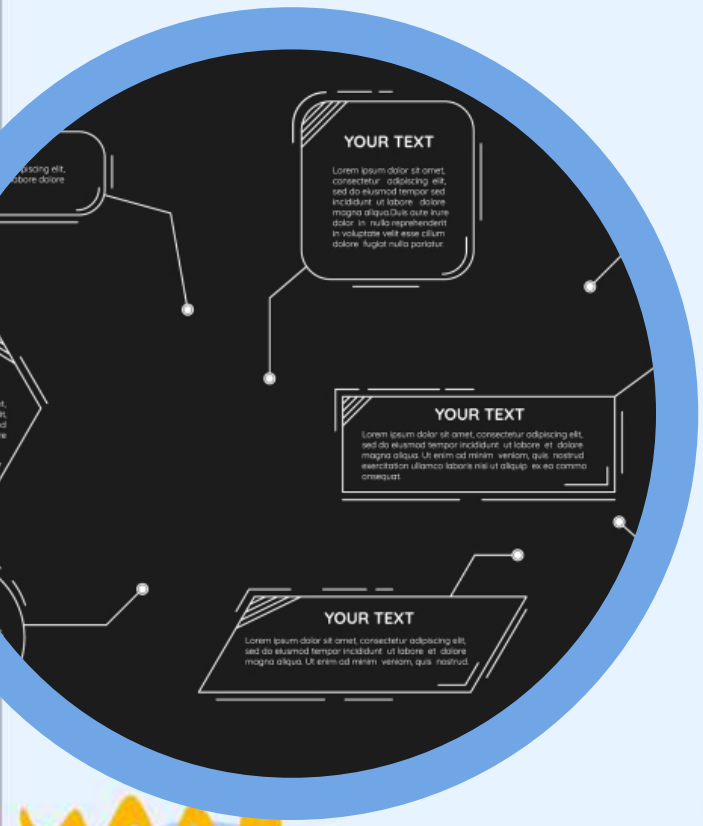
将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

05

原型模式（Protot...

用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。

结构型设计模式



适配器模式（Adapter Pattern）

将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

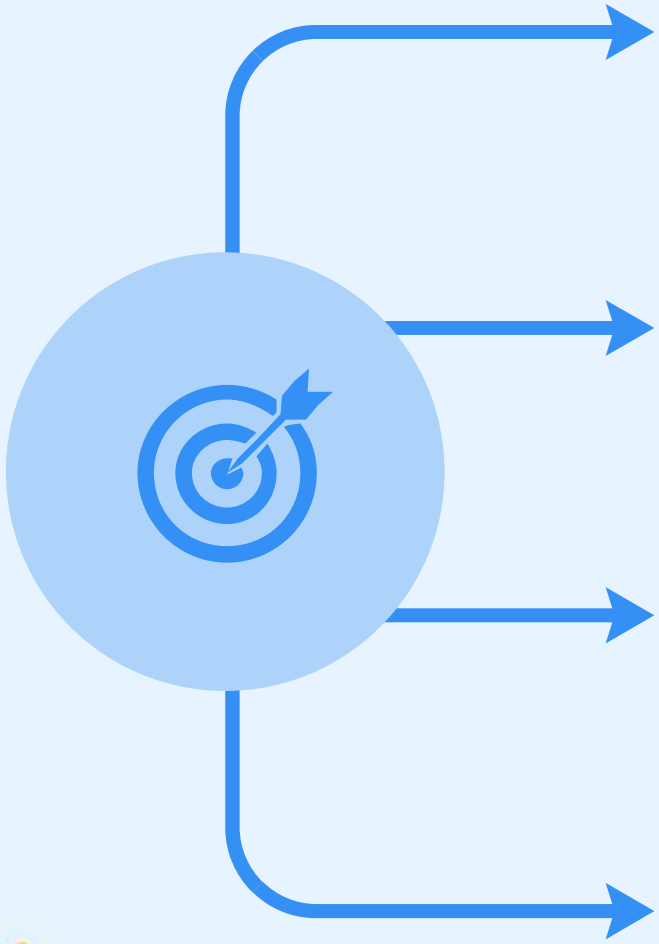
桥接模式（Bridge Pattern）

将抽象部分与实现部分分离，使它们都可以独立地变化。

组合模式（Composite Pattern）

将对象组合成树形结构以表示“部分-整体”的层次结构。组合模式使得用户对单个对象和复合对象的使用具有一致性。

结构型设计模式



装饰器模式（Decorator Patt...

动态地给一个对象添加一些额外的职责。就增加功能来说，装饰器模式相比生成子类更为灵活。

外观模式（Facade Pattern）

为子系统中的一组接口提供一个一致的界面，外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

享元模式（Flyweight Patte...

运用共享技术有效地支持大量细粒度的对象。

代理模式（Proxy Pattern）

为其他对象提供一种代理以控制对这个对象的访问。

行为型设计模式

01

模板方法模式（Temp...

定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

02

策略模式（Strate...

定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。本模式使得算法可独立于使用它的客户而变化。

03

观察者模式（Obser...

定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

04

状态模式（State...

允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它的类。

05

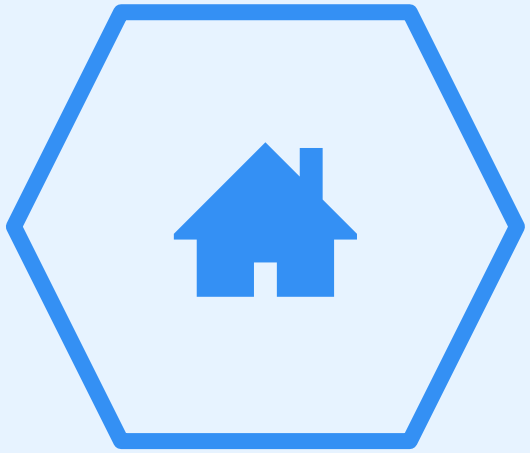
访问者模式（Visit...

表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。

03

设计模式在软件开发中应用案例分析

提高系统可维护性和可扩展性案例分析



01

工厂模式

通过工厂模式，将对象的创建与使用分离，降低系统耦合度，提高可维护性。例如，在UI组件库中，使用工厂模式可以根据不同的参数创建不同的组件实例，方便扩展和维护。

02

单例模式

确保系统中某个类只有一个实例，并提供全局访问点。这可以避免多个实例造成的资源浪费和状态不一致问题，提高系统的稳定性和可维护性。例如，数据库连接池、配置管理器等常用单例模式实现。

03

观察者模式

定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。这可以降低系统各部分之间的耦合度，提高系统的可扩展性和可维护性。例如，事件驱动编程中经常使用观察者模式。