**一、简答题（**10**分）**

1.1（3 分）什么是设计模式？设计模式的目标是什么？

设计模式是软件设计过程当中某一类常见问题的一般性问题的解决方案。

1.2（3 分）设计模式具有哪三大特点？

封装、继承、多态、（抽象）

1.3（4 分）GOF 设计模式常用的有几种？GOF 设计模式按照模式的目的可

分为哪三类？

行为型：单例模式、工厂方法模式、抽象工厂模式、建造者模式、原型模式

结构型：适配器模式、装饰模式、桥接模式、组合模式、享元模式、代理模式、外观模式

创建型：观察者模式、模板方法模式、命令模式、状态模式、职责链模式、解释器模式、中介者模式、访问者模式、策略模式、备忘录模式、迭代器模式

1.4 设计模式遵循的原则

单一职责原则、开放封闭原则、依赖倒转原则、里氏代换原则、合成聚合复用原则、迪米特原则

1.5 面向对象思想

可维护，可复用，可扩展，灵活性好

**二、填空题（**20**分，每空**2**分）**

请从外观、访问者、组合、工厂方法、模板方法、观察者、

原型、职责链、单例、抽象工厂、命令11 种模式中选择10 种填入下列的

空缺中。

2.1 （ 工厂方法）模式中，父类负责定义创建对象的公共接口，子类则负责生

成具体的对象。

2.2 （抽象工厂）模式提供一系列相关或相互依赖对象的接口而无需指定它

们具体的类。

2.3 （原型）模式用原型实例指定创建对象的种类，并且通过拷贝这些原

型实例创建新的对象。

2.4 （单例）模式确保某一个类仅有一个实例，并自行实例化并向整个系

统提供这个实例。

2.5 （组合）模式将对象组合成树形结构以表示“部分-整体”的层次结

构。使得用户对单个对象和组合对象的使用具有一致性。

2.6 （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，为子系统中的一组接口提供一个一致的界面。

2.8 （职责链）模式使多个对象都有机会处理请求，从而避免请求的发送者

和接收者之间的耦合关系。

2.9 （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。

2.10 （模板方法）模式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。

第 3 页（共 12 页）

**三、选择题（**20**分，每题**2**分）**

3.1 设计模式具有的优点。A

A. 适应需求变化B. 程序易于理解

C. 减少开发过程中的代码开发工作量D. 简化软件系统的设计

3.2 下图描述了一种设计模式，该设计模式不可以。B

(职责链模式)



A. 动态决定由一组对象中某个对象处理该请求

B. 动态指定处理一个请求的对象集合, 并高效率地处理一个请求

C. 使多个对象都有机会处理请求，避免请求的发送者和接收者间的耦

合关系

D. 将对象连成一条链，并沿着该链传递请求

3.3 若系统中的某子模块需要为其它模块\_\_\_\_\_B\_\_\_\_\_提供访问不同数据库系统

（Oracle、SQL Server、DB2 UDB 等）的功能，这些数据库系统提供的访

问接口有一定的差异，但访问过程却都是相同的，例如，先连接数据库，

再打开数据库，最后对数据进行查询， 设计模式可抽象出相同的数

据库访问过程；

A. 外观B. 装饰C. 单件D. 模板方法

3.4 系统中的文本显示类（TextView）和图片显示类（PictureView）都继

承了组件类（Component），分别显示文本和图片内容，现需要构造带有滚

动条、或者带有黑色边框、或者既有滚动条又有黑色边框的文本显示控件

和图片显示控件，但希望最多只增加三个类，B

设计模式可以实现该目的。

A. 外观 B. 装饰C. 单件D. 模板方法

3.5 下面的类图表示的是哪个模式？A



A、抽象工厂模式(Abstract Factory)

B、工厂方法模式(Factory Method)

C、命令模式(Command)

D、责任链模式(Chain of Responsibility)

3.6 下面的类图表示的是哪个设计模式？D

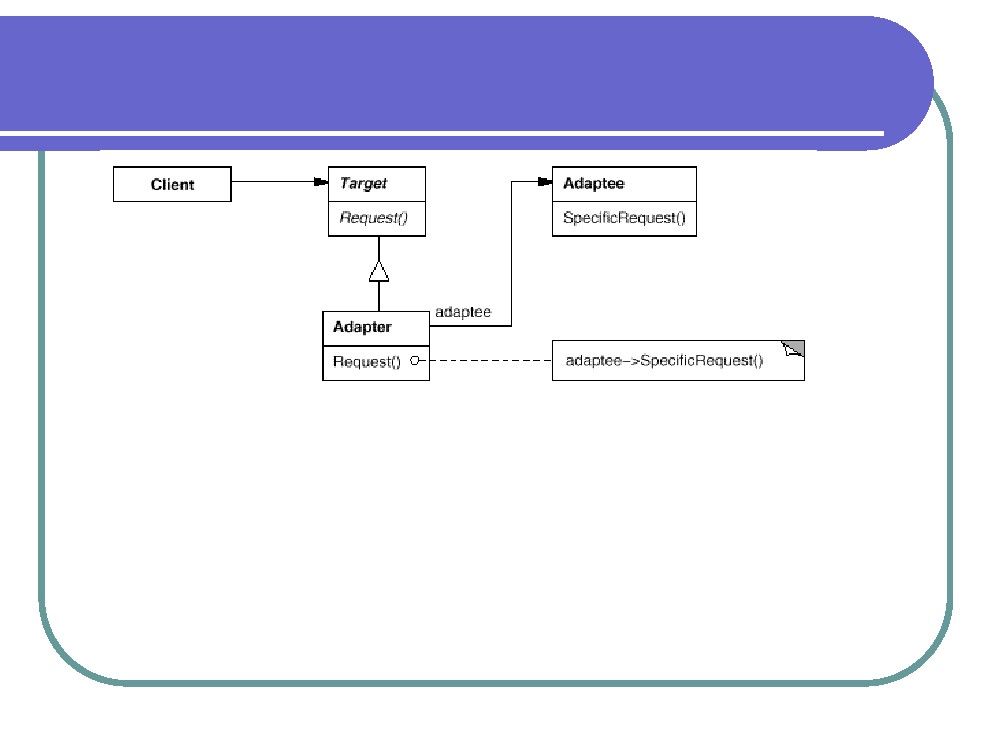


A、代理模式（Proxy） B、装饰模式（Decorator）

C、桥接模式（Bridge） D、观察者模式（Observer）

3.7 下面的类图表示的是哪个模式？

第 5 页（共 12 页）



A、代理模式（Proxy） B、外观模式（Facade）

C、适配器模式（Adapter） D、桥接模式（Bridge）

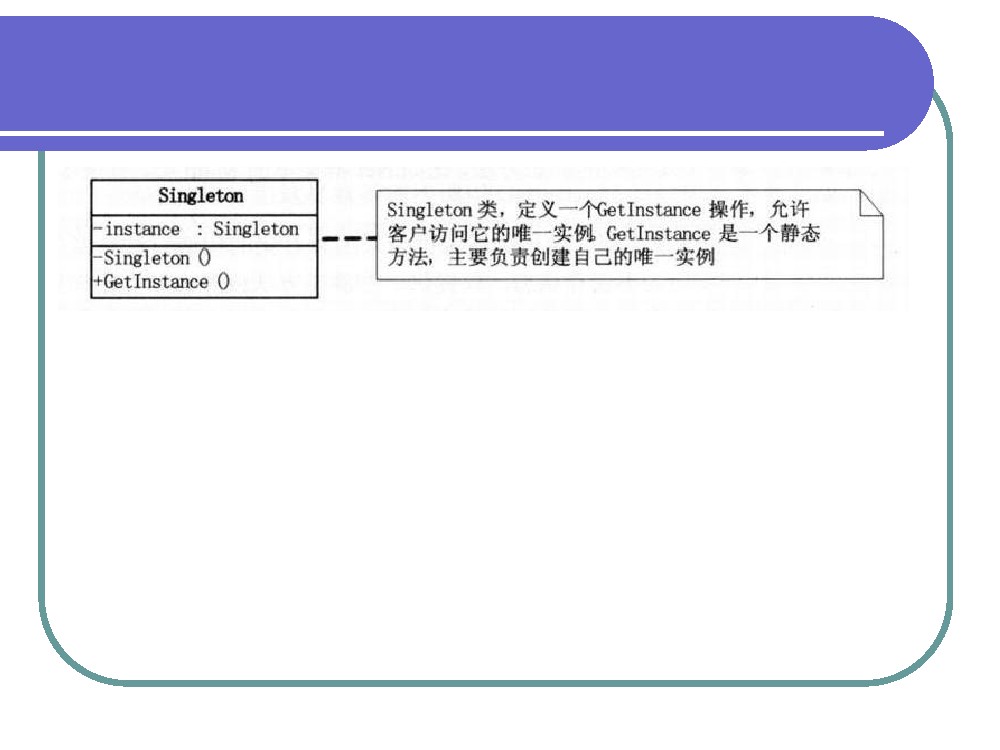
3.8 下面的类图表明了哪个模式？

A、职责链模式(Chain of Responsibility)

B、命令模式(Command)

C、备忘录模式（Memento）

D、工厂方法模式（Factory Method）



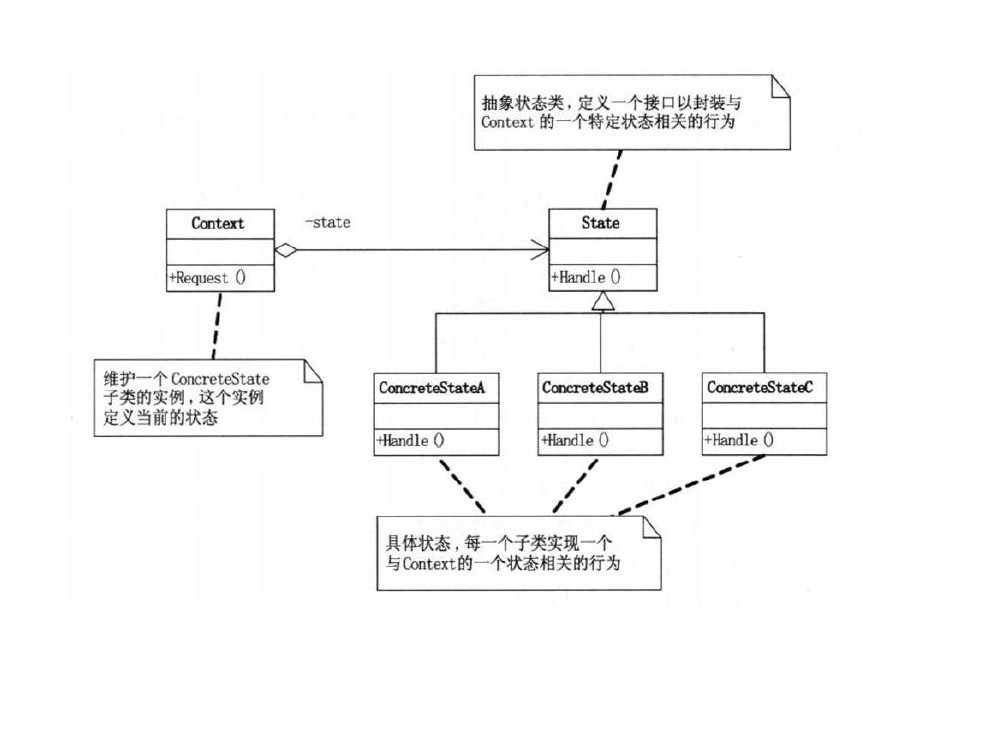
3.9 下面的类图表示的是哪个设计模式？

A、模板方法模式(Template Method) B、命令模式(Command)

C、单例模式(Singleton) D、状态模式(State)

第 6 页（共 12 页）

3.10 下面的类图表明了哪个模式？



A、模板方法模式(Template Method) B、命令模式(Command)

C、单例模式(Singleton) D、状态模式(State)

**四、读程题（**30**分，每题**15**分）**

4.1 在下面的C++代码中，类SalesTicket能够完成打印票据正文的功能，

类HeadDecorator 与FootDecorator 分别完成打印票据的台头和脚注的功

能。已知该程序运行后的输出结果如下所示。

这是票据的台头！

这是票据正文！

这是票据的脚注！

------------------------

这是票据的台头！

这是票据的脚注！

[C++**程序代码**]

#include <iostream>

using namespace std;

class SalesTicket {

public:

virtual void printTicket() { cout << "这是票据正文!" << endl;}

};

class Decorator : public SalesTicket{

SalesTicket \*ticket;

public:

Decorator(SalesTicket \*t){ ticket = t; }

void printTicket(){

if(ticket != NULL)

ticket->printTicket();

第 7 页（共 12 页）

}

};

class HeadDecorator : public Decorator{

public:

HeadDecorator(SalesTicket \*t): Decorator(t) { }

void printTicket() {

cout << "这是票据的台头！" << endl;

Decorator::printTicket();

}

};

class FootDecorator :public Decorator{

public:

FootDecorator(SalesTicket \*t): Decorator(t){ }

void printTicket() {

Decorator::printTicket();

cout << "这是票\_\_\_\_\_\_\_j\_\_据的脚注！" << endl;

}

};

void main(void) {

SalesTicket t;

FootDecorator f(&t);

HeadDecorator h(&f);

h.printTicket();

cout << "------------------------"<< endl;

FootDecorator a(NULL);

HeadDecorator b(&a);

b.printTicket();

}

请根据题目描述、程序和输出，阐述采用该设计模式进行设计的优势？

第 8 页（共 12 页）

4.2 某游戏公司现欲开发一款面向儿童的模拟游戏，该游戏主要模拟现实

世界中各种鸭子的发声特征、飞行特征和外观特征。游戏需要模拟的鸭子

种类及其特征如下表所示：

**鸭子种类 发声特征 飞行特征 外观特征**

灰 鸭

（MallardDuck）

发出“嘎嘎”声

（Quack）

用翅膀飞行

（FlyWithWings）

灰色羽毛

红 头 鸭

（RedHeadDuck）

发出“嘎嘎”声

（Quack）

用翅膀飞行

（FlyWithWings）

灰色羽毛、

头部红色

棉 花 鸭

（CottonDuck）

不 发声

（QuackNoWay）

不能飞行

（FlyNoWay）

白色

橡 皮 鸭

（RubberDuck）

发出橡皮与空气摩

擦的声（Squeak）

不能飞行

（FlyNoWay）

黑白橡皮

颜色

为支持将来能够模拟更多种类鸭子的特征，采用某设计模式设计的类

图如下所示：

第 9 页（共 12 页）

其中，Duck 为抽象类，描述了抽象的鸭子，而类RubberDuck、

MallardDuck、CottonDuck 和RedHeadDuck 分别描述具体的鸭子种类，方

法fly()、quack()和display()分别表示不同种类的鸭子都具有飞行特征、

发声特征和外观特征；类FlyBehavior 与QuackBehavior为抽象类，分别

用于表示抽象的飞行行为与发声行为；类FlyNoWay与FlyWithWings分别

描述不能飞行的行为和用翅膀飞行的行为；类Quack、Squeak与QuackNoWay

分别描述发出“嘎嘎”声的行为、发出橡皮与空气摩擦声的行为与不发声

的行为。

[C++**代码**]

#include<iostream>

using namespace std ;

class FlyBehavior {

public : virtual void fly() = 0;

};

class QuackBehavior {

public: virtual void quack() = 0;

};

class FlyWithWings:public FlyBehavior{

public: void fly(){ cout << "使用翅膀飞行！" << endl; }

};

class FlyNoWay:public FlyBehavior{

public: void fly(){ cout << "不能飞行！" << endl; }

};

class Quack:public QuackBehavior{

public: void quack(){ cout << "发出\'嘎嘎\'声！" << endl;

}

};

class Squeak:public QuackBehavior{

public: void quack(){ cout << "发出空气与橡皮摩擦声！" << endl;

}

};

class QuackNoWay:public QuackBehavior{

public: void quack (){ cout << "不能发声！" << endl; }

};

class Duck {

protected:

FlyBehavior \* flyBehavior;

QuackBehavior \* quackBehavior;

public:

void fly(){ flyBehavior->fly(); }

第 10 页（共 12 页）

void quack() { quackBehavior->quack(); };

virtual void display()=0;

};

class RubberDuck: public Duck {

public:

RubberDuck(){

flyBehavior = new FlyNoWay();

quackBehavior = new Squeak();

}

~RubberDuck(){

if(!flyBehavior) delete flyBehavior;

if(!quackBehavior) delete quackBehavior;

}

void display(){ /\*此处省略显示橡皮鸭的代码 \*/ }

};

//其它代码省略

请根据上述类图和代码，阐述采用该设计模式进行设计的优势？

第 11 页（共 12 页）

**五、设计题（**20**分，每题**10**分）**

**请根据下面的叙述，选择正确的设计模式，并给出该设计模式的类图。**

5.1 为实现应用的界面与应用数据的分离，通常定义应用数据的类和应用

界面类，它们都可以各自独立地被复用，当然它们也可一起工作。一个表

格对象和一个柱状图对象可使用不同的表示形式描述同一个应用数据对象

的信息。表格对象和柱状图对象互相并不知道对方的存在，这样使你可以

根据需要单独复用表格或柱状图。但是它们表现的似乎互相知道彼此的存

在。当用户改变表格中的信息时,柱状图能立即反映这一变化, 反过来也是

如此。如下图所示：

请根据叙述和上图的描述，选择合适的设计模式，并给出该设计模式的类

图。

第 12 页（共 12 页）

5.2 一般的企业采购审批都是分级的，采购量的不同就需要不同层次的主

管人员来审批，如主任可以审批1 万元以下的采购单，副董事长可以申批

2.5 万元以下的采购单，董事长可以审批10万元以下的采购单，10 万元以

上的采购单就需要开会决定。请根据该描述，选择合适的设计模式并给出

其类图。