1. 简要说明什么是软件体系结构，软件体系结构模型，为什么要建立软件体系结构模型？

答：软件体系结构指一个软件系统在高层次上的结构化组织方式，包括系统的组成部分和各个部分之间的关系，以及它们与环境之间的交互，同时它描述了各个组件之间的联系和功能分配，为系统的设计、维护提供指导和支持。

软件体系结构模型是描述软件体系结构的抽象表达，通常以图形化的方式呈现，用于表达软件系统整体结构和各个部分之间的关系，包含组件、连接、约束等等。

使用软件体系结构模型可以使开发人员能够更好理解整个系统开发的结构和组成，使软件开发更加有序、高效，有益于软件的质量和可维护性。

1. 简要说明什么是构件和软件重用？



答：构件（component）：指的是一个系统中的模块或部件，具有独立的功能和接口，并且可以通过接口与其他构件进行交互和组合。它是软件系统的基本构成单位，它可以独立地进行开发、测试、部署和维护。

软件重用：将软件开发过程中完成并通过测试的模块、组件库等重复利用的过程。

1. 4+1视图模型中各视图的作用。

答：逻辑视图：描述软件功能和系统架构的

开发视图：描述软件开发过程

过程视图：描述软件运行过程（交互、执行）

物理视图：描述软件的物理结构和部署方式（硬件环境）

场景视图：描述软件的使用场景和使用流程（用户需求）

1. UML模型图在软件体系结构建模中的作用。图的画法

答：用例图：描述用户的需求和功能

类图：用于描述软件的静态结构

活动图：描述软件系统结构的操作流程和业务流程

状态图：描述软件系统中的状态和状态转换

时序图：用于描述软件系统中的对象交互和信息传递的顺序

组件图：用于描述组件和依赖之间的关系

部署图：用于描述软件系统的物理部署结构

1. 简单工厂模式和工厂方法模式的优缺点。

答：

|  |  |  |
| --- | --- | --- |
| 模式名称 | 优点 | 缺点 |
| 简单工厂模式 | 隐藏具体实现、降低客户端和实现类的耦合度 | 不易扩展、代码复杂度高 |
| 工厂方法模式 | 将创建对象交给子类进行，具有可扩展性、降低耦合度、代码可读性高 | 增加代码复杂度 |

1. WSDL文档开发客户端应用程序的流程。

答： WSDL （Web Server Description Language）

1. 获取WDSL文件： web服务网址+“?WDSL”
2. 生成客户端代码
3. 编写客户端代码
4. 部署客户端应用程序
5. 测试
6. 二层C/S、三层C/S、B/S，以及C/S与B/S混合架构风格示意图及其含义。

答：B/S（browser/server）: 浏览器-服务端架构，由浏览器向服务器发送请求，服务器向浏览器页面进行响应。

C/S（client/server）: 客户端-服务端，由客户端应用向服务器发送网络请求，有服务端向客户端应用响应。客户端通常负责处理用户界面和输入数据，而服务器则负责处理应用程序逻辑和数据存储。

三层C/S，它将应用程序分成三个层次：表示层（Presentation Layer）、应用层（Application Layer）和数据层（Data Layer）。表示层负责处理用户界面，应用层负责处理应用程序逻辑，数据层负责处理数据存储和检索。

B/S、C/S混合架构：指将B/S架构和C/S架构进行结合，通过服务器端的中间件来协调前端浏览器和后端应用程序之间的交互。在这种架构中，前端用户界面仍然是基于浏览器的Web应用程序，但是通过中间件与后端服务器进行交互，后端服务器可以提供更多的应用逻辑和数据处理功能。

1. 设计模式以及对应的编程

// 产品接口

public interface Product {

void use();

}

// 具体产品1

public class ConcreteProduct1 implements Product {

public void use() {

System.out.println("I am ConcreteProduct1.");

}

}

// 具体产品2

public class ConcreteProduct2 implements Product {

public void use() {

System.out.println("I am ConcreteProduct2.");

}

}

// 工厂类

public class Factory {

public static Product createProduct(String type) {

if ("Product1".equals(type)) {

return new ConcreteProduct1();

} else if ("Product2".equals(type)) {

return new ConcreteProduct2();

} else {

return null;

}

}

}

// 客户端

public class Client {

public static void main(String[] args) {

Product product1 = Factory.createProduct("Product1");

product1.use();

Product product2 = Factory.createProduct("Product2");

product2.use();

}

}

1. 某客户要求开发一个无人自动售货机系统，主要需求调研记录如下：送货员在每周固定时间查看售货机中的商品数量和钱币数量情况，并根据计划适当添加商品、钱币或取走适当的钱币。同时,送货员也可以根据需要增加商品的种类,或修改商品的价格。售货时，顾客把钱币投入机器的投币口中，机器检查钱币的真伪，如果发现假币机器拒收，并将其从退币口退出。当机器接收了有效的钱币之后，将之送入钱币储藏器。顾客支付的货币根据钱币的面值进行累加。自动售货机装有货物分配器，每个分配器中包含0个或多个价格相同的货物。顾客通过选择货物分配器来选择货物。如果有货，且顾客支付的货币值不小于该货物的价格，货物将被分配到货物传送孔送给顾客，并将找零返回到退币口。如果无货，则和顾客支付的货币值相等的钱币将被送到退币口。如果顾客支付的货币值小于所选货物的价格，机器将等待顾客投进更多的货币。如果顾客决定不买所选择的货物，他投放进的钱币将从退币口中退出。

请根据以上描述，进行分析设计，完成UML多视图体系架构建模任务：

为了完成这个无人自动售货机系统的设计，需要进行多视图体系架构建模任务。下面将分别从逻辑视图、进程视图、物理视图

1. 逻辑视图 逻辑视图描述系统的静态结构和行为。在这个自动售货机系统中，存在以下类：

* VendingMachine：自动售货机类，负责售货、找零等操作。
* Coin：钱币类，包括面值、数量、真伪等属性。
* Product：商品类，包括名称、价格等属性。
* Dispenser：分配器类，包括存储的商品、库存等属性。
* Refund：退款类，记录退款的钱币和数量。

类之间的关系如下：

* VendingMachine和Coin之间是一对多的关系，即一个自动售货机可以接收多种面值的钱币。
* VendingMachine和Product之间是多对多的关系，即一个自动售货机可以有多种商品，一个商品可以被多个自动售货机售卖。
* VendingMachine和Dispenser之间是一对多的关系，即一个自动售货机可以有多个分配器，一个分配器只属于一个自动售货机。
* VendingMachine和Refund之间是一对多的关系，即一个自动售货机可以有多个退款记录，一个退款记录只属于一个自动售货机。

进程视图 进程视图描述系统的动态行为。在这个自动售货机系统中，存在以下进程：

* VendingMachineProcess：自动售货机进程，负责处理售货、找零等业务逻辑。
* RefundProcess：退款进程，负责退款操作。

自动售货机进程接收来自客户端的请求，包括选择商品、投入钱币、查询库存等。如果客户投入的钱币足够支付选择的商品，自动售货机进程将发起退款进程，完成找零和出货的操作。如果客户选择的商品无货或投入的钱币不足，自动售货机进程将等待客户继续投入钱币或重新选择商品。

1. 物理视图 物理视图描述系统的物理组成和部署。在这个自动售货机系统中，存在以下物理组件：

Client

* VendingMachineDevice：自动售货机设备，包括货物分配器、钱币投入口、退币口、货物传送孔等组件。
* VendingMachineServer：自动售货机服务器，负责接收客户端请求并调用自动售货机进程处理请求。

