软件工程复习概要

第一部分：软件工程基本知识体系和概念

一、软件工程基本概念

1. **软件工程**：（1）将工程化方法用于软件；（2）对（1）中的方法的研究。
2. **软件过程**：开发软件产品或者软件系统时，所要执行的步骤或者流程。**过程模型**就是指开发的步骤或者流程的通用模板，即为了改变软件开发的混乱状况，设计的模板。
3. **过程模式**包含以下几点：
   模式名称、驱动力、类型、启动条件、问题、解决方案、结果、相关模式、已知应用和实例。
4. **UML**：统一建模语言。
5. **敏捷性**：适当（或快速）响应变更。

二、软件设计概念

1．**软件体系结构**：建立计算机系统所需的数据结构和程序构件。软件设计的目标之一就是导出系统体系结构示意图，该示意图作为一个框架，将指导更详细的设计活动。

2. **设计模式**：描述了解决某个特定设计问题的设计结构，该设计问题具有特定条件，该条件会影响到模式的应用和使用方式。Eg:体系结构模式、构件级设计模式、用户界面设计模式、WebApp 设计模式、移动 App 设计模式。
3. **关注点分离**：关注点是一个特征或者行为被指定为软件需求模型的一部分。将关注点分割为更小的关注点即关注点分离（分而治之+更容易解决问题）。
4. **模块化**：模块化是关注点分离的最常见表现。
5. **信息屏蔽**：每个模块对其他所有模块都屏蔽自己的设计决策。信息屏蔽是个描述模块化独立性的概念，即模块因该被特殊说明并设计，使信息（算法和数据）都包含在模块内，其他模块无需对这些信息进行访问。
6. **重构**：简化构件的设计（代码），而无需改变其功能或者行为。
7. **设计类**：分析模型定义了一组分析类，每个分析类都描述问题域中的某些元素，这些元素关注用户可见的问题方面。五种不同的设计类：用户接口类、业务域类、过程类、持久类、系统类。

三、面向对象设计

1. **面向对象设计（Object-Oriented）**：面向对象设计是一种软件设计方法，它解决**“类与相互通信的对象之间的组织关系”**，它是独立于编程语言的，但是面向对象设计模式的最终实现仍然要使用面向对象编程语言来表达，如 Visual Basic、.NET、C++/CLI 等。
   其本质是**以建立模型体现出来的抽象思维过程和面向对象的方法**。模型是用来反映现实世界中事物特征的。任何一个模型都不可能反映客观事物的一切具体特征，只能对事物特征和变化规律的一种抽象，且在它所涉及的范围内更普遍、更集中、更深刻地描述客体的特征。通过建立模型而达到的抽象是人们对客体认识的深化。
2. 面向对象设计有**三个主要特征**：
   （1）封装性，即信息屏蔽；
   （2）继承性，继承是一种由已有类创建子类的机制，利用继承，可以先创建一个共有属性的一般类，根据这个类再创建具有特殊属性的子类，被继承的类成为父类，当然子类也可以成为父类来继续向下扩展；（在父类和子类之间存在着继承和扩展关系，子类

继承父类的属性和方法的同时，子类还可以扩展出新的属性和方法，并且还可以覆盖父类中方法的实现方式）

（3）多态性，同一个信息被不同的对象接收到时可能产生不同的行为。

3. 面向对象设计有**五个原则**：（或设计基于类的构件）

（1）单一职责原则（SRP）：一个类只有一种单一的功能；

（2）开放封闭原则（OCP）：软件对扩展开放，对修改封闭；

（3）Liskov 替换原则（LSP）：子类必须能够替换其父类，但是父类不一定能替换子类；

（4）依赖倒置原则（DIP）：即依赖于抽象：高层模块-抽象-底层模块；

（5）接口隔离原则（ISP）：使用多个小的专用接口，而不是一个大的总接口，即一个类对另外一个类的依赖应该建立在最小的接口上。

Remark：依赖与继承，依赖关系从左向右，继承关系自下向上（f 导出类 t 基类）。

4. 与面向对象的设计模式有三类：创建型模式、结构型模式和行为型模式。

四、软件体系概念

1. **软件体系结构的重要性**：
   a) 提供了一种表示，方便以后的交流与学习；
   b) 突出了早期的设计决策；
   c) 构建了一个相对小、易于理解的模型，该模型描述了系统如何构成、构件如何一起工作。
2. **体系结构模式**：描述了解决某个特定设计问题的设计结构，该设计问题具有特定条件，该条件会影响到模式的应用和使用方式。（即设计模式）
3. **体系结构风格**：不同于结构模式，为整个系统的所有构件建立一个结构。
4. **不同之处**：（1）模式涉及的范围要小一些，它更多集中在体系结构的某一方面而不是体系结构的整体；（2）模式在体系结构上施加规则，描述更面向具体问题。
5. **体系结构风格的简单分类**：（1）以数据为中心的体系结构；（2）数据流体系结构；（3）调用和返回体系结构；（4）面向对象体系结构；（5）层次体系结构。
6. **构件**：面向软件体系架构的可复用软件模块。将体系结构细化为构件：体系结构必须提供很多基础设施构件，使应用构件能够运作，但是这些基础设施构件于应用领域没有业务联系。（系统中模块化的、可复用的、可部署的部件，该部件封装实现并对外提供一组接口：构件）
7. **框架**：框架是可以重复使用的“微型体系结构”，可以作为应用其他设计模式的基础。
8. **设计模式与框架的区别**：（1）设计模式比框架更加抽象；（2）框架中含有设计模式，but 设计模式中没有框架；
   联系：共同搭配完成一个完整的设计。
9. **软件框架（架构）模式**：分层模式、客户端-服务器模式、主从设备模式、管道-过滤器模式、代理模式、点对点模式、事件总线模式、模型-视图-控制器模式、黑板模式、解释器模式。
10. **基于模式的软件设计**：基于模式的软件设计也是一种软件设计方法，区别于面向对象的设计方法或者传统的软件设计方法，给予了一种新的解决问题的思考方式。分为：理解全局（需求模型）、在抽象层上表示模式、在特定平台上进行关联。
11. **面向对象的设计方法和基于模式的设计方法的联系区别**：

面向对象设计在技术上很成熟，它的主要特点是交互性强、具有安全的存取模式、网络通信量低、响应速度快、利于处理大量数据。该结构的程序是针对性开发，变更不够灵活，维护和管理的难度较大，分布功能弱且兼容性差，因此缺少通用性，具有较大的局限性。

基于模式的设计方法主要特点是分布性强、维护方便、开发简单且共享性强、总体拥有成本低。但数据安全性问题、对服务器要求过高、数据传输速度慢、软件的个性化特点明显降低，这些缺点是有目共睹的，难以实现传统模式下的特殊功能要求。例如通过浏览器进行大量的数据输入或进行报表的应答、专用性打印输出都比较困难和不便。此外，实现复杂的应用构造有较大的困难。虽然可以用 ActiveX、Java 等技术开发较为复杂的应用，但是相对于发展已非常成熟面向对象的设计方法的一系列应用工具来说，这些技术的开发复杂，并没有完全成熟的技术工具供使用。

第二部分：十种常见的架构模式

一、分层模式

1．这种模式也称为**多层体系架构模式**。

- 每个模块必须属于某个层次
- 每个层次为下一个层次提供更高层次的服务，同时派任务给上一个层次的模块
- 层次间不可逆向调用，不可跨层调用

2．一般信息系统中最常见的是如下 4 层：

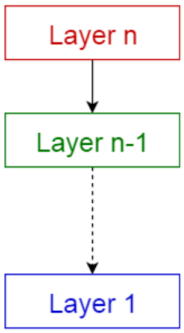
- 表示层(也称 UI 层)
- 应用层(也称为服务层)
- 业务逻辑层(也称为领域层)
- 数据访问层(也称为持久化层)

3．使用场景：一般的桌面应用程序、电子商务 Web 应用程序。

4．优缺点：

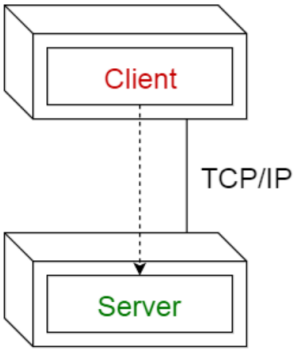
(1) 适应集成不同类型功能，eg：集成不同功能的代码；

(2) 不适应简单的业务模型，eg：系统本身不复杂，可预期的修改也很少，分层建模会大材小用，增加很多不必要的代码，过度设计。



二、客户端-服务器模式

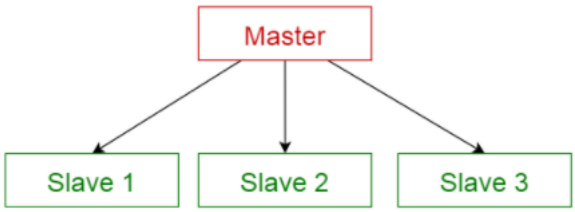
1. 这种模式由两部分组成：**一个服务器和多个客户端**。服务器组件将为多个客户端组件提供服务。客户端从服务器请求服务，服务器为这些客户端提供相关服务。此外，服务器持续侦听客户机请求。
2. 使用场景：电子邮件、文件共享、银行等在线应用程序。



三、主从设备模式

1. 这种模式由两方组成：**主设备和从设备**。主设备组件在相同的从设备组件中分配工作，并计算最终结果，这些结果是由从设备返回的结果。
2. 使用场景：

- 在数据库复制中，主数据库被认为是权威的来源，并且要与之同步
- 在计算机系统中与总线连接的外围设备(主和从驱动器)



四、管道-过滤模式

1. 此模式可用于构造**生成和处理数据流**的系统。

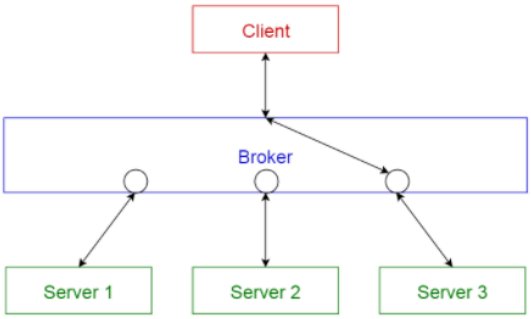
每个处理步骤都封装在一个过滤器组件内
要处理的数据是通过管道传递的
这些管道可以用于缓冲或用于同步

2. 使用场景：

- 编译器。连续的过滤器执行词法分析、解析、语义分析和代码生成
- 生物信息学的工作流

五、 代理模式

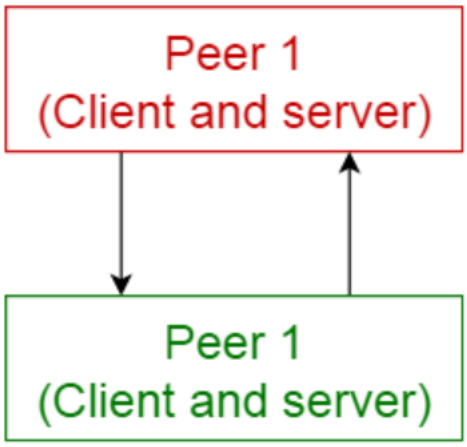
1. 此模式用于构造**具有解耦组件的分布式系统**。这些组件可以通过远程服务调用彼此交互。代理组件负责组件之间的通信协调。
2. 服务器将其功能(服务和特征)发布给代理。客户端从代理请求服务，然后代理将客户端重定向到其注册中心的适当服务。
3. 使用场景：消息代理软件，如 Apache ActiveMQ，Apache Kafka，RabbitMQ 和 JBoss Messaging。



六、 点对点模式

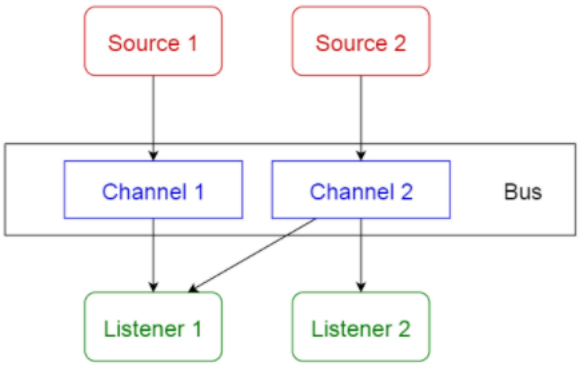
1. 在这种模式中，单个组件被称为对等点。对等点可以作为客户端，从其他对等点请求服务，作为服务器，为其他对等点提供服务。**对等点可以充当客户端或服务器或两者的角色，并且可以随时间动态地更改其角色。**
2. 使用场景：

- 像 Gnutella 和 G2 这样的文件共享网络生物信息学的工作流
- 多媒体协议，如 P2PTV 和 PDTP
- 像 Spotify 这样的专有多媒体应用程序



七、 事件总线模式

1. 这种模式主要是处理事件，包括 4 个主要组件：**事件源、事件监听器、通道和事件总线**。消息源将消息发布到事件总线上的特定通道上。侦听器订阅特定的通道。侦听器会被通知消息，这些消息被发布到它们之前订阅的一个通道上。
2. 使用场景：安卓开发、通知服务。



八、 模型-视图-控制器模式

1. 这种模式，也称为 MVC **模式**，把一个交互式应用程序划分为 3 个部分，

模型：包括核心功能和数据
视图：查看和修改数据的接口，即显示模型数据，发送用户动作到控制器
控制器：解释处理用户输入的信息/对数据施加操作

运作如下：模型是一些人名的数据，视图以表格的形式显示这些数据，用户进行操作(eg: 按钮点击)，视图将该操作发给控制器，控制器解释该操作，将更新后的模型数据发送给视图。这样的话，可以实现模型不依赖于控制器或视图，控制器提供模型数据至视图，解释用户的行为。控制器依赖于视图和模型。这样做是为了将信息的内部表示与信息的呈现方式分离开来，并接受用户的请求。它分离了组件，并允许有效的代码重用。
在一些情况下，控制器和视图可以合二为一。

2. 使用场景：

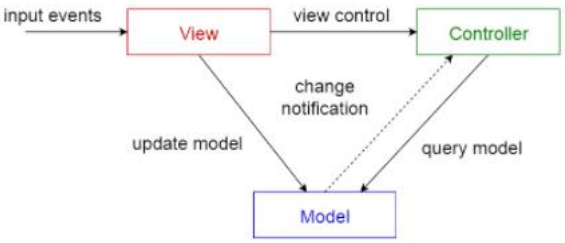
- 在主要编程语言中互联网应用程序的体系架构
- 像 Django 和 Rails 这样的 Web 框架

3. 优点：

- MVC **使得模型类可重复使用而不需修改**
  不必要的复杂性是软件开发的梦魇。它导致软件漏洞百出，维护费用昂贵。而到处引入依赖很容易就会造成代码过于复杂。**控制器存在的目的是消除模型与视图依赖关系。 从模型中移除视图依赖后，模型代码变得整洁起来，而且重复使用而无需修改。**
- MVC **使视图可重复使用而无需修改**
  当视图只显示一种类型的模型对象时，合并视图和控制器是比较合适的。然而，分离控制器和视图，可以通过控制器解除模型和视图的依赖关系，使得视图更加简洁且可以重复使用。
- 这样，实现新的功能和维护变得轻而易举。

4. 使用场景：

- 在主要编程语言中互联网应用程序的体系架构
- 像 Django 和 Rails 这样的 Web 框架



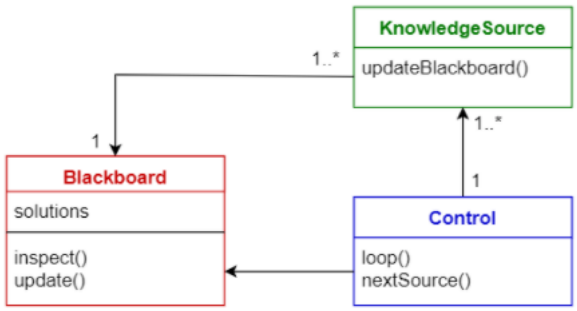
九、 黑板模式

1. 这种模式对于没有确定解决方案策略的问题是有用的。黑板模式由 3 个主要组成部分组成。

黑板：包含来自解决方案空间的对象的结构化全局内存
知识源：专门的模块和它们自己的表示
控制组件：选择、配置和执行模块

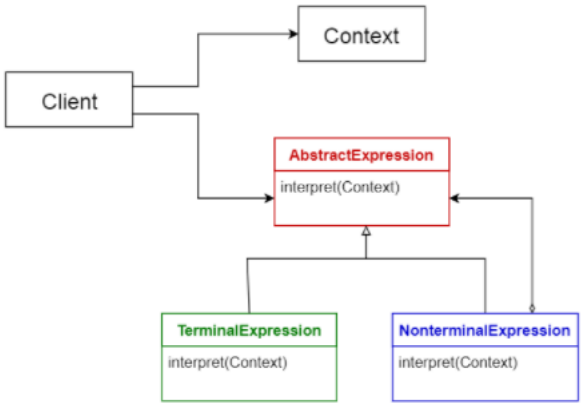
所有的组件都可以访问黑板。组件可以生成添加到黑板上的新数据对象。组件在黑板上查找特定类型的数据，并通过与现有知识源的模式匹配来查找这些数据。

2. 使用场景：语音识别、车辆识别和跟踪、蛋白质结构识别、声纳信号的解释。



十、 解释器模式

1. 这个模式用于**设计一个解释用专用语言编写的程序的组件**。它主要指定如何评估程序的行数，即以特定的语言编写的句子或表达式。其基本思想是为每种语言的符号都有一个分类。
2. 使用场景：数据库查询语言(eg：SQL)、用于描述通信协议的语言。



体系架构模式的比较

名称	优点	缺点
分层模式	一个较低的层可以被不同的层所使用。层使标准化更容易，因为我们可以清楚地定义级别。可以在层内进行更改，而不会影响其他层。	不是普遍适用的。在某些情况下，某些层可能会被跳过。
客户端-服务器模式	很好地建立一组服务，用户可以请求他们的服务。	请求通常在服务器上的单独线程中处理。由于不同的客户端具有不同的表示，进程间通信会导致额外开销。
主从设备模式	准确性——将服务的执行委托给不同的从设备，具有不同的实现。	从设备是孤立的：没有共享的状态。主-从通信中的延迟可能是一个问题，例如在实时系统中。这种模式只能应用于可以分解的问题。

名称	优点	缺点
管道-过滤器模式	展示并发处理。当输入和输出由流组成时，过滤器在接收数据时开始计算。轻松添加过滤器，系统可以轻松扩展。过滤器可重复使用。 可以通过重新组合一组给定的过滤器来构建不同的管道。	效率受到最慢的过滤过程的限制。从一个过滤器移动到另一个过滤器时的数据转换开销。
代理模式	允许动态更改、添加、删除和重新定位对象，这使开发人员的发布变得透明。	要求对服务描述进行标准化。
点对点模式	支持分散式计算。对任何给定节点的故障处理具有强大的健壮性。在资源和计算能力方面具有很高的可扩展性。	服务质量没有保证，因为节点是自愿合作的。安全是很难得到保证的。性能取决于节点的数量。
事件总线模式	新的发布者、订阅者和连接可以很容易地添加。对高度分布式的应用程序有效。	可伸缩性可能是一个问题，因为所有消息都是通过同一事件总线进行的。
模型-视图-控制器模式	可以轻松地拥有同一个模型的多个视图，这些视图可以在运行时连接和断开。	增加复杂性。可能导致许多不必要的用户操作更新。
黑板模式	很容易添加新的应用程序。扩展数据空间的结构很简单。	修改数据空间的结构非常困难，因为所有应用程序都受到了影响。可能需要同步和访问控制。
解释器模式	高度动态的行为是可行的。对终端用户编程性提供好处。提高灵活性，因为替换一个解释程序很容易。	由于解释语言通常比编译后的语言慢，因此性能可能是一个问题。

软件方法考核制试题

1、 简述什么是模型驱动体系结构（MDA），以及 MDA 开发方法的优点。

2、 在 MDA（模型驱动的体系结构）开发方法中，模型被分为平台无关的模型和特定实现平台上的平台相关模型。请简单论述这么做的好处。

3、 在软件建模过程中，人们往往先建立平台无关的模型来描述软件系统的业务逻辑，然后再建立特定实现平台上的平台相关模型。请简单论述这种建模方法的优点。

4、 在模型驱动的软件体系结构中，平台无关模型（**Platform Independent Models，PIM**）和平台相关模型（**Platform Specific Models，PSM**）分别描述软件的哪些方面？软件运行时依赖的网络拓扑结构信息应该在 **PIM** 还是 **PSM** 中描述？（**15** 分）

5、 简述模型-视图-控制器（Model-View-Control）体系结构模式中，模型，视图和控制器三者的功能以及它们之间的关系。

6、 简述软件体系结构的概念。在模型-视图-控制器模式中，模型修改之后通过发布-订阅模式来通知视图更改显示的好处是什么？模型为什么不直接调用视图的例程进行修改？

7、 四、简述软件体系结构的概念。在模型-试图-控制器模式（**Model View Controller，MVC**）中，视图主要担负什么样的责任？（7 分）

8、 简述什么叫做面向对象程序设计,其主要特征是什么？

9、 在面向对象程序设计中，类的封装和继承为什么存在矛盾？C++是如何处置这个矛盾的？

10、 简述面向对象语言中是如何实现封装的。

11、 （10 分）简述什么叫做设计模式？在系统设计中使用设计模式可以带来哪些好处？

12、

四.（10）统一建模语言 UML 的主要特点是什么？它有哪几种图示？ 简述其中的 2 种图示。

答案

1-4 **模型驱动体系构架（MDA）**是一种独立于特定平台和软件供应商的软件体系结构设计和开发方法，它适用于设计、部署、集成等软件开发的整个生命周期。MDA 建模是基于功能，而非基于特定语言、平台或实现技术，它可以简化系统集成、缩短开发周期和节省企业资源。模型通常以图和文字的形式来描述一个系统及其环境。**模型驱动的方法就是利用模型来引导系统的设计、开发和维护。而模型驱动架构即是用系统的模型来生成系统的体系结构。**

MDA 有三个视图：

（1） **计算无关视图(CIV)**，其作用就是将系统基本处理逻辑同平台相关的技术规范分离开来。CIV 视图关注于系统的环境和需求，而系统的具体结构和实现是隐藏的。（有时候没有该视图，该视图合并于 PIV）**（基本需求）**

（2） **平台无关视图(PIV)**，该视图关注于系统的操作而隐藏了平台相关的细节，该视图可能用一种通用的、平台无关的建模语言如 UML 来描述。**（逻辑层）**

（3） **平台相关视图（PSV）**，该视图关注特定平台的实现细节。**（实现层）**

计算无关模型（CIM）通常由业务分析人员创建，展示了系统的业务模型。 平台无关模型（PIM）是系统功能的模型，通常由系统架构师创建。 平台相关模型（PSM）对一个或多个平台的 PIM 模型的具体实现建模。

MDA 的真正价值在于 CIM 模型可以通过简单的映射转换成 PIM 模型。同样的，PIM 模型也可以映射成 PSM 模型，而 PSM 模型则可以最终转换成具体的实现代码。**PIM 和 PSM 之间是抽象与具体的关系，与具体实现技术无关，PIM 可以被多种实现技术复用，当技术平台发生变化时，PIM 不必做改动。**

优点：MDA 提供了一种优雅而可靠的开发框架，这种框架使得系统架构师**在没有考虑到有关系统实现的任何细节之前就可以事先定义好系统的模型**。

（1） 可移植性，根据已有的 PIM 创建新的 PSM 是相当容易的；

（2） 跨平台的互操作性，可以根据特殊规则将一个 PIM 模型转化为一个异构的 PSM 模型；

（3） 开发效率，MDA 是一种极其高效的设计和开发方法；

（4） 软件质量，使用一种单一的模型来生成和导出系统的大部分代码可以极大地降低人为错误的发生；

（5） MDA 还有其它更多的优势，如对新技术的快速包容、平台无关性、领域相关性、降低开发成本和缩短开发周期等等。

5-7 MVC 模式主要有三部分构成：模型、视图和控制器。其中，模型保存数据和核心程序，视图用来显示数据以及接收用户指令，控制器处理数据和用户指令。模型与视图无关，模型中的数据由控制器处理，然后在视图显示；视图于模型无关，视图接收的用户指令由控制器处理，处理的数据结果在作用在模型中。这样的设计下实现了 模型的简单化和可复用性、视图的简单化和可复用性。

订阅发布模式：事件触发者被抽象出来，称为消息发布者，即图中的 P。事件接受都被抽象出来，称为消息订阅者，即图中的 S。P 与 S 之间通过 S.P（即订阅器）连接。这样就实现了 P 与 S 的解耦。

软件体系结构是建立计算机系统所需要的数据结构和程序构件（或者视为具有一定能够形式化的结构化的元素，即构件的集合）。

8-10　模型是描述了具体问题或者具体实物的一方面特征，是一种有定向的抽象化提炼。面向对象设计是一种新的软件开发设计方法，解决了类与相互通信对象之间的组织关系，其本质就是建立模型体现出来的抽象思维过程和面向对象的方法。（面向对象设计是一种把面向对象的思想应用于软件开发过程中，指导开发活动的系统方法，是建立在“对象”概念基础上的方法学。基于对象概念，以对象为中心，以类和继承为构造机制。）面向对象设计有三个特征：封装性、继承性和多态性；有五个基本原则：单一功能性、对扩展开放，对修改封装、L-继承性：子可以继承父，父不一定可以继承子、接口隔离：尽量使用多而小的专用接口而不是汇总的大接口、依赖倒置原则：高层和基层都通过抽象联系。

封装性指的是隐藏模块化软件的信息(数据、程序、细节等)；继承性指的是子类继承父类属性并可以扩展。矛盾在于如果一个父类是封装的，子类继承父类应该依然是封装的，but如果是封装的就不可以扩展，产生矛盾。故封装性和继承性是一个平衡关系，而不是绝对关系，一定程度的继承和封装可以提该开发效率。

C++使用**虚拟继承**的方式部分解决了继承性与封装性的矛盾。

封装就是封装，把信息（数据、方法）封装到一个类中，然后设置访问控制符。允许其它类访问的，就设置为 public；允许子类访问的，就设置为 protected；只能自己访问的，就设置为 private。封装是实现抽象的手段之一。实现封装的方法有：属性的封装、类的封装和方法的封装等。

11、 设计模式是一种广为人知、可复用、经过分类编目、代码设计的经验的总结。使用设计模式是为了可重用代码、让代码更加易于理解、保证代码的可靠性。总而言之，是让设计者可以更加方便有效的使用代码、设计模型。

12、UML 是一种用来对真实世界物体进行建模的语言。具有良定义、易于表达、功能强大、使用普遍等特点。

UML 定义了五类 10 种模型图：

(1)、用例图：展示系统外部的各类执行者与系统提供的各种用例之间的关系

(2)、类图：展示系统中类的静态结构(类是指具有相同属性和行为的对象，类图用来描述系统中各种类之间的静态结构)

(3)、对象图：是类图的一种实例化图(对象图是对类图的一种实例化)

(4)、包图：是一种分组机制。在 UML1.1 版本中，包图不再看作一种独立的模型图)

(5)、状态图：描述一类对象具有的所有可能的状态及其转移关系(它展示对象所具有的所有可能的状态以及特定事件发生时状态的转移情况)

(6)、顺序图：展示对象之间的一种动态协作关系(一组对象组成，随时间推移对象之间交换消息的过程，突出时间关系)

(7)、合作图：从另一个角度展示对象之间的动态协作关系(对象间动态协作关系，突出消息收发关系)

(8)、活动图：展示系统中各种活动的执行流程(各种活动的执行顺序、执行流程)

(9)、构件图：展示程序代码的物理结构(描述程序代码的组织结构，各种构件之间的依赖关系)

(10)、配置图：展示软件在硬件环境中(特别是在分布式及网络环境中)的配置关系(系统中硬件和软件的物理配置情况和系统体系结构)