# 面向对象的概念简介

面向对象的语言的特点：抽象（abstract）、封装（encapsulation）、继承（inheritance）、多态（polymorphism）、组合（compose）；

对象是包含了数据与行为的实体，过程式编程，属性与行为分开，面向对象中属性与行为在一个对象中； 限制访问具体属性与方法的行为叫数据隐藏；将属性与方法合并到一个对象中叫做封装，类是对象的蓝图，必须先定义类，在定义对象，定义的属性存放对象的状态， 数据隐藏是封装的主要目的与实现方式，在Java中主要通过接口与实现分离来实现。

接口定义了对象通信的手段，public方法都属于接口；

继承主要是为了代码复用，继承就是通过识别与抽象公共属性与行为来创建新类，这种类就是父类或者基类，继承主要是is-a的关系。

子类对父类定义的行为有不同的实现，重载（override）就是覆盖父类的实现，这就是多态。

使用其他对象来构建或结合成新的对象，这就是组合，组合是has-a的关系。

复用其他类来创建新类的手段就是继承与组合。

# 如何以面向对象的方式进行思考

面向对象设计的基本单位是类，结果就是一个好的对象模型，不要总执迷于一开始就做出完美的设计

解决问题时不要尝试遵从任何标准与约定，要有创造性。

首要的事情是定位与解决业务问题，先进行概念分析与设计。

* 清楚接口与实现的区别：设计类时，要思考向用户暴露什么？隐藏什么很重要；接口是不可变的，实现是可变的，因为接口涉及到与外部的协作；接口的设计要遵循最小原则；
* 使用抽象思维设计接口：高层次的抽象接口比高度具体的接口更有用，面向对象的目标就是设计抽象的可高度重用的类；
* 尽可能提供最小化的用户接口：可以隐藏全部，随着用户的需求增加，主键开放方法为公开，从用户的角度定义类，首先确定有哪些用户以及用户的需求都是什么？然后根据需求定义接口的行为，最后收集环境约束。

# 高级的面向对象的概念

对于构建一个面向对象系统需要的概念：

1. 构造函数：创建对象先调用构造函数，new关键字分配内存，然后调用构造函数初始化内存，在类中至少包含一个构造函数，如果有属性，最好在构造函数中初始化为一个稳定有效的状态。
2. 错误处理：发生错误时忽略问题不是好方案，检查问题并关闭程序;
3. 作用域的重要性：局部变量、属性与静态属性；
4. 操作符重载；
5. 多重继承：接口是行为继承、抽象类是实现继承；
6. 对象操作：比较/拷贝等需要注意；

# 类的剖析

* 类名：识别与描述用途、交互方式；
* 注释：
* 属性：属性私有，接口设计尽量最小化原则；
* 构造函数：在构造函数中初始化属性是一个优秀的实践、检查属性值是否是null也是优秀的实践；
* 访问器：set/get，因为需要读改属性信息，但是属性是私有的；
* 公共接口方法：倾向于抽象化；
* 私有实现方法：

# 类设计指导

* 对现实世界系统建模：面向对象编程的主要目标是按照人们实际所想的相似方式对现实世界系统建模，通过类来代表模型，现实世界中的对象如何交互，类的对象就如何交互；
* 识别公共接口：设计良好的接口描述了客户想要的服务，最小化公共接口可以是类简单，不容易出错，如果后面发现需要增加行为，可以使用继承或者组合扩展，接口隐藏了细节实现，实现可以随便改，封装的最高指导原则是：所有的字段都是私有的；
* 设计健壮的构造函数（析构函数)：构造函数应该把对象设置微安全的初始状态，通常也要考虑到默认情况下的如何构造；
* 在类中设计错误处理：
* 设计时考虑重用：
* 设计时考虑拓展性：主要是添加新的功能比较方便继承与组合，类必须是内聚的，不能包含额外的功能，妨碍拓展，因为你子类可能不需要的功能，类、方法、属性的名字要遵循约定，并且名字描述用途，抽象不可移植的代码，就是把if/switch这种结构的分之代码放到各自的类中，设计一个中间类来抽象，客户只需要与中间类打交道；提供复制与比较对象的操作，保持尽可能小的作用域，类的职责与自身高度相关（单一职责）；
* 设计时考虑可维护性：减少代码依赖，使用迭代的方式逐步开发-设计-重构，最终达到合理的系统设计，测试时可以mock代码mock数据，而不需要连接实际的环境；
* 使对象持久化，存储到文件/数据库/ODB中。

# 使用对象进行设计

面向对象(OO)的设计是健壮与灵活的软件开发方式。

1. 设计指导，一个稳固的面向对象的设计过程包含一下步骤：

* 进行正确的分析：项目需求、软件价值、软件内容、可行性等分析；
* 编写工作陈述文档来描述系统：整个系统的需求与知识，就是把分析阶段的内容文档化，形成正式的文案；
* 通过规格说明收集需求：就是编写需求文档，相比工作文档，更细致，能对设计进行参考；
* 开发用户接口的原型；用户与开发人员理解系统的最好的方式就是原型，原型可以是任何动心，可以是粗略的界面与流程，优秀的原型可以帮助识别类；
* 识别类；需求文档确定后，开始识别类，识别类的简单方式是名词高亮，设计是迭代的过程，后续随着系统的理解深入与设计的优化，类是增加或者减少的；
* 确定每个类的职责；识别类的属性与操作；
* 确定类之间如何交互；确定交互的信息；
* 创建一个高层次的模型来描述系统的构建；通常就是UML。

1. 对象包装，面向对象编程与面向过程编程是一致的；

* 结构化代码：结构化编程的基本结构是序列、条件判断与迭代，面向对象编程的基本结构是抽象、封装、继承、多态与组合；过程化编程中代码被包装到函数中，面向对象则是包装到对象的方法中；
* 包装结构化代码：就是把行为提炼成对象的方法；
* 包装不可移植的代码：就是多态，定义接口，多态来自定义实现；
* 包装已有的类：包装是为了能够在复用以前类的基础上，改变或者增加接口定义（适配器模式）或者实现自己的逻辑（装饰器模式），注意主要是为了复用以前的代码。如果直接修改实现，则接口定义就可能不对了，如果修改了接口定义，实现就可能不对了，所以要包装了。

# 精通组合与继承

组合与继承都是一种重用机制，继承是继承其他类的属性与行为，组合是使用其他对象构建新的对象。

组合代表has-a的关系，继承代表is-a的关系，

继承就是从通用到具体的过程，组合与继承都容易引起过度复杂的设计，要注意设计的粒度。

封装等同于数据与行为，就是打包程序的过程，将程序分为接口与实现的过程，接口是行为，实现是数据；

继承削弱了父类与子类之间的封装，父类修改，影响子类，子类覆盖导致父类定义的行为不准；如果修改父类实现破坏了子类的预期行为，则他们不是is-a的关系，不需要继承。

多态是对继承带来的最优雅的使用。

# 使用接口与抽象类设计

接口与抽象类是代码重用的重要的机制，提供了契约的功能，框架提供标准化功能组件，实现通用功能的重用机制，契约是指要求开发人员遵循API规格要求的一种机制，通常API就是框架，契约必须得到遵守，因为它是标准，实现契约的2种方式是抽象类与接口。

* 抽象类：包含一个或者多个抽象方法，这些抽象方法就是契约；
* 接口：接口脱离与继承体系之外，属于实现，并且可以多实现，但是只能单继承，严格的继承关系中，类与抽象类有近似的关系，但是接口可以应用到不相关的类，这是使用抽象类与接口的关键区别，接口指定了没有明显联系类的共同的行为。

# 第九章 创建对象与面向对象设计

使用组合的原因是通过组合可以降低构建系统的复杂性，这是解决问题的通用的方式，另外，也保证了组件的可替换性；

稳定复杂系统的特点：

* + 稳定的复杂的系统通常具有一定的层级，每个系统由更简单的子系统构成，而子系统又由更简单的子系统构成，在面向对象的设计中，组合满足这个规则，通过简单的对象构造复杂的对象；
  + 稳定的复杂的系统是可分解的，可以识别出组成部分，并且组成部分的交互要少于组成部分的内部交互；
  + 可工作的复杂系统是由可工作的简单系统演化而来，总是基于已经验证过的系统来构建新的系统；

组合是对抗软件复杂度的重要的策略之一。

组合的2种方式：

* 聚合：视角就是看到组合后的整体，最简单的has-a关系，内部包含，而外部不知，是整个系统的必须的根本属性对象；
* 联合：明显看出来组成系统的部分有哪些，并且是如何通信的，也就是联合组成的系统更为分散一些；

# 创建对象模型

UML是对象模型，UML是一种可视化的建模语言，可以设计、构造与文档化软件系统，类图由3部分组成：类名、属性与方法；

* 属性模式: [+\-]name:Type;
* 方法名模式: [+|-]method name[(parameter,...)]:return type;

访问符号+｜-表示属性的访问修饰符，可以没有，则是默认的访问修饰符；继承使用带箭头的连线表示，接口实现使用带箭头的虚线表示；

一个类由其他类组成时是聚合关系，使用带菱形的线表示，一个类需要其他类的服务时是联合关系使用一个线表示，基数代表对象之间对应的个数，也可以标注。

# 对象与可移植数据：XML和JSON

XML与JSON提供了一种在相互独立的应用程序之间共享数据的机制；XML全称是扩展标记语言，与HTML都是继承与SGML（通用标记语言），HTML用于展示数据，但是对数据验证没有要求，XML要严格一些，但是只用于表示数据；为了表示数据，XML需要自定义标签，为了验证标签的正确性与正确的使用方式，产生了一种叫做文档类型定义DTD的文件，用于验证XML的正确性，JSON全称是JavaScript对象标记，是轻量级的文本数据交换格式，与语言无关，并且JSON是自描述的，与语言无关，JSON使用与JS创建对象的语法来创建数据，但是是语言无关的。

# 持久化对象：序列化、封送与关系型数据库