**对象的交互：**

当一个对象里有多个对象的时候，那些对象之间是如何交互的，对象和对象之间的联系是如何建立的，对象如何和其他对象交流。对象和对象之间的联系紧密程度叫做耦合。对象和对象的耦合程度越紧，表现在源代码上，就是它们的代码是互相依赖、互相牵制的。我们理想的模型，是对象和对象之间的耦合要尽可能的松，平行的对象要尽量减少直接联系，让更高层次的对象来提供通信服务。这些就是在我们这个非常简单的数字钟的例子中所体现的。

**封装**，就是把数据和对这些数据的操作放在一起，并且用这些操作把数据掩盖起来，是面向对象的基本概念之一，也是**最核心**的**概念**。

我们有一个非常直截了当的手段来保证在类的设计的时候**做到封装**：

1. 所有的成员变量必须是private的，这样就避免别人任意使用你的内部数据；
2. 所有public的函数，只是用来实现这个类的对象或类自己要提供的服务的，而不是用来直接访问数据的。除非对数据的访问就是这个类及对象的服务。简单地说，给每个成员变量提供一对用于读写的get/set函数也是不合适的设计。

**Private 修饰符：**

针对的是一个类里的，而对同一个类的不同对象来说，不同的对象之间私有的变量可以互相访问。

**包：**

当你的程序越来越大的时候，你就会需要有一个机制帮助你**管理**一个工程中众多的**类**了。**包**就是Java的**类库管理机制**，它借助文件系统的目录来管理类库，一个包就是一个目录，一个包内的所有的类必须放在一个目录下，那个目录的名字必须是包的名字。

作为初学者，你可以忽略不看包，反正一切靠Eclipse。但是作为一个Java程序员，你不能不懂包。要不然，在使用别人的类库和部署你的程序的时候，会遇到不少莫名其妙的麻烦。

类是描述，对象是实体。在类里所描述的成员变量，是位于这个类的每一个对象中的。

而如果某个成员有**static**关键字做修饰，它就**不再属于每一个对象，而是属于整个类的了。**

通过每个对象都可以访问到这些**类变量**和**类函数**，但是也可以通过类的名字来访问它们。**类函数**由于不属于任何对象，因此也没有办法建立与调用它们的对象的关系，就不能访问任何非static的成员变量和成员函数了。

**容器**是现代程序设计非常基础而重要的手段。

所谓容器，就是“放东西的东西”。数组可以看作是一种容器，但是数组的元素个数一旦确定就无法改变，这在实际使用中是很大的不足。一般意义上的容器，是指具有自动增长容量能力的存放数据的一种数据结构。在面向对象语言中，这种数据结构本身表达为一个对象。所以才有“放东西的东西”的说法。

Java具有丰富的容器，Java的容器具有丰富的功能和良好的性能。熟悉并能充分有效地利用好容器，是现代程序设计的基本能力。

我们首先学习的是顺序容器，即放进容器中的对象是按照指定的顺序（放的顺序）排列起来的，而且允许具有相同值的多个对象存在。

（在一些书中，将容器（英文为collection或container）翻译为“集合”，由于数学中的集合（Set）也是一种特定的容器类型，我们认为将collection翻译为集合是不恰当的。所以我们只会使用容器一词。）

当**数组的元素**的类型**是类**的时候，数组的每一个元素其实只是对象的管理者而不是对象本身。因此，**仅仅创建数组并没有创建其中的每一个对象！**

**对象数组中的每个元素都是对象的管理者而非对象本身（即：字符串数组中，每个单元存放的是该单元字符串的地址（指针））**

对理解**继承**来说，最重要的事情是，知道哪些东西被继承了，或者说，子类从父类那里得到了什么。答案是：所有的东西，所有的父类的成员，包括变量和方法，都成为了子类的成员，除了构造方法。构造方法是父类所独有的，因为它们的名字就是类的名字，所以父类的构造方法在子类中不存在。除此之外，子类继承得到了父类所有的成员。

但是得到不等于可以随便使用。每个成员有不同的访问属性，子类继承得到了父类所有的成员，但是不同的访问属性使得子类在使用这些成员时有所不同：有些父类的成员直接成为子类的对外的界面，有些则被深深地隐藏起来，即使子类自己也不能直接访问。下表列出了不同访问属性的父类成员在子类中的访问属性：

|  |  |  |
| --- | --- | --- |
| 父类成员访问属性 | 在父类中的含义 | 在子类中的含义 |
| public | 对所有人开放 | 对所有人开放 |
| protected | 只有包内其它类、自己和子类可以访问 | 只有包内其它类、自己和子类可以访问 |
| 缺省 | 只有包内其它类可以访问 | 如果子类与父类在同一个包内：只有包内其它类可以访问  否则：相当于private，不能访问 |
| private | 只有自己可以访问 | 不能访问 |

public的成员直接成为子类的public的成员，protected的成员也直接成为子类的protected的成员。Java的protected的意思是包内和子类可访问，所以它比缺省的访问属性要宽一些。而对于父类的缺省的未定义访问属性的成员来说，他们是在父类所在的包内可见，如果子类不属于父类的包，那么在子类里面，这些缺省属性的成员和private的成员是一样的：不可见。父类的private的成员在子类里仍然是存在的，只是子类中不能直接访问。我们不可以在子类中重新定义继承得到的成员的访问属性。如果我们试图重新定义一个在父类中已经存在的成员变量，那么我们是在定义一个与父类的成员变量完全无关的变量，在子类中我们可以访问这个定义在子类中的变量，在父类的方法中访问父类的那个。尽管它们同名但是互不影响。

在构造一个子类的对象时，父类的构造方法也是会被调用的，而且父类的构造方法在子类的构造方法之前被调用。在程序运行过程中，子类对象的一部分空间存放的是父类对象。因为子类从父类得到继承，在子类对象初始化过程中可能会使用到父类的成员。所以父类的空间正是要先被初始化的，然后子类的空间才得到初始化。在这个过程中，**如果父类的构造方法需要参数，如何传递参数就很重要了。**

类定义了类型,DVD类所创建的对象的类型就是DVD。类可以有子类,所以由那些类定义的类型可以有子类型。在DoME的例子中,DVD类型就是Item类型的子类型。

子类型类似于类的层次,类型也构成了类型层次。子类所定义的类型是其超类的类型的子类型。

当把一个对象赋值给一个变量时,对象的类型必须与变量的类型相匹配,如:

    Car myCar = new Car();

是一个有效的赋值,因为Car类型的对象被赋值给声明为保存Car类型对象的变量。但是由于引入 了继承,这里的类型规则就得叙述得更完整些:

****一个变量可以保存其所声明的类型或该类型的任何子类型。****

对象变量可以保存其声明的类型的对象,或该类型的任何子类型的对象。

Java中保存对象类型的变量是多态变量。“多态”**这个术语(字面意思是许多形态)是指一个变量可以保存不同类型(即其声明的类型或任何子类型)的对象**。

**接口**

Java把抽象的概念又更推进了一步。这就是接口（Interface）。接口其实就是完全抽象的类，因此接口和类的地位是一样的，此前所有针对类的所有法则同样适用于接口。

接口所有的方法都是没有方法体的，而且都是public abstract，即使你没有这样声明。而接口中的所有成员变量都是public static final的变量，并且必须有定义初始化，因为所有变量都必须在编译的时候有确定值。

接口和类的地位是一样的。因此我们可以看到其定义方式跟类也是基本上一样的。当然，其中的所有方法都没有方法体。而当类从接口得到继承的时候，Java用另外一个动词：实现（implements）。当然，实现接口的类必须覆盖接口中所有的方法，否则这个类就成为一个抽象类。

Java不允许多继承，但是允许一个类实现多个接口，也允许一个接口从多个接口得到继承，但是不允许接口从类继承。

GUI（图形用户界面）给应用程序提供界面,其中包括窗口、菜单、按钮和其他图形组件,这就是今天大多 数人所熟悉的“典型”应用程序界面。

图形用户界面所涉及的细节很多,我们的课程并不打算教授GUI，但是我们打算借助GUI来介绍两个**设计思想**：**控制反转**和**MVC设计模式**。

部件是创建GUI的独立部分,比如像按钮、菜单、菜单项、选择框、滑动条、文本框等。Java类库中有不少现成的部件。

**布局**是指如何在屏幕上放置组件。过去,大多数简单的GUI系统让程序员在二维坐标系上 指定每个组件的x和y坐标(以像素点为单位),这对于现代的GUI系统来说太简单了。因为现代的GUI系统还得考虑不同的屏幕分辨率、不同的字体、用户可改变的窗口尺寸,以及许多其他使得布局困难的因素。所以需要有一种能更通用地指定布局的方法,比如,要求“这个部件应该在那个部件的下面“或者”这个部件在窗口改变尺寸时能自动拉伸,但是其他部件保持尺寸不变”。这些可以通过**布局管理器**(layout manager)来实现。

**事件处理**是用来响应用户输入的技术。创建了部件并且放在屏幕上合适的位置以后,就得 要有办法来处理诸如用户点击按钮这样的事情。Java类库处理这类事情的模型是基于事件的。 如果用户激活了一个部件(比如,点击按钮或者选择菜单项),系统就会产生一个事件。应用 程序可以收到关于这个事件的通知(以程序的一个方法被调用的方式),然后就可以采取程序该做的动作了。

Swing使用一个非常灵活的模型来处理GUI的输入:采用**事件监听器的事件处理(event handling)模型**。

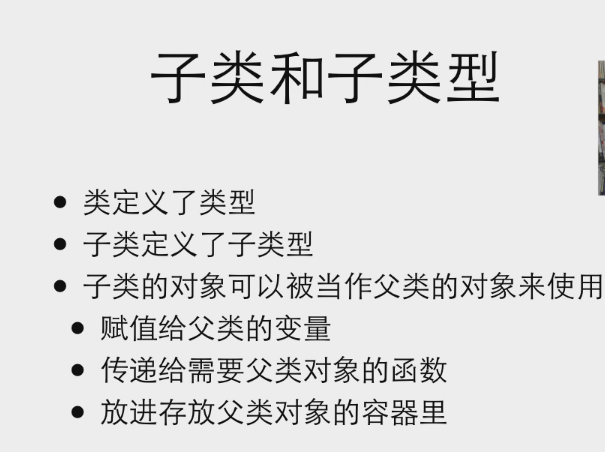
Swing框架本身以及大部分部件在发生一些情况时会触发相关的事件,而其他的对象也许会对这些事件感兴趣。不同类型的动作会导致不同类型的事件。当点击一个按钮或选中一个菜单项,部件就会触发动作事件;而当点击或移动鼠标时,会触发鼠标事件;当框架被关闭或最小化时,会触发窗口事件。另外还有许多种其他事件。

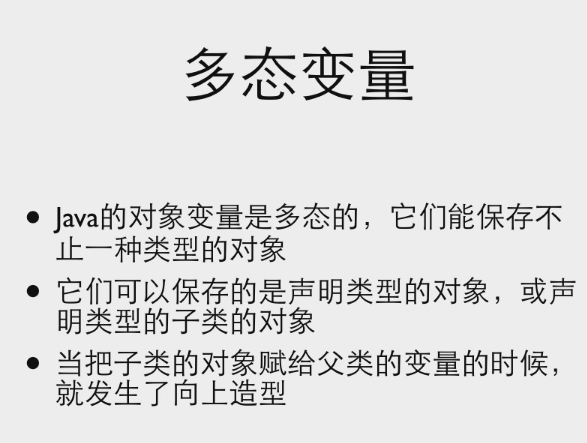
所有的对象都可以成为任何这些事件的监听器,而一旦成为监听器,就可以得到这些事件触发的通知。

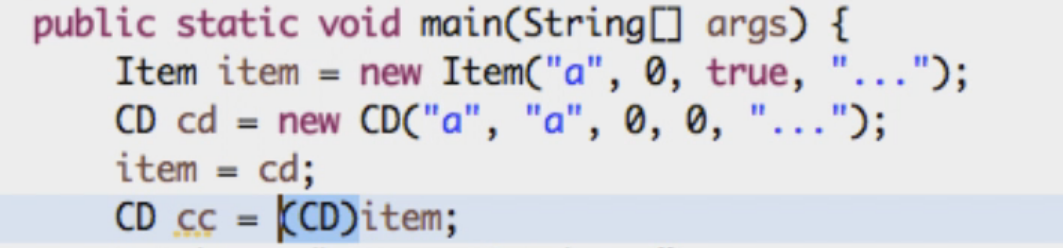
实现了众多监听器接口之一的对象就成为一个事件监听器。如果对象实现了恰当的接口, 就可以注册到它想监听的组件上。

**内部类**就是指一个类定义在另一个类的内部，从而成为外部类的一个成员。因此一个类中可以有成员变量、方法，还可以有内部类。实际上Java的内部类可以被称为成员类，内部类实际上是它所在类的成员。所以内部类也就具有和成员变量、成员方法相同的性质。比如，成员方法可以访问私有变量，那么成员类也可以访问私有变量了。也就是说，成员类中的成员方法都可以访问成员类所在类的私有变量。内部类最重要的特点就是能够访问外部类的所有成员。

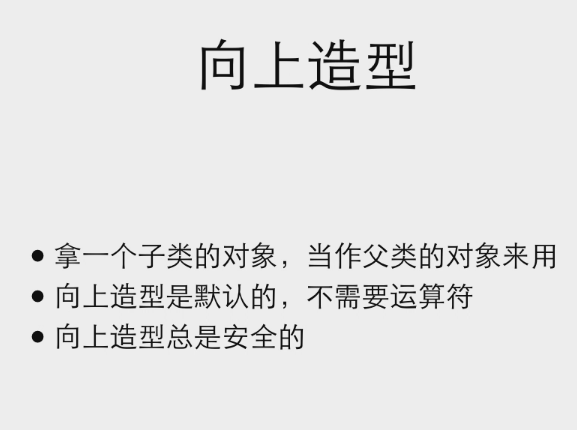
内部类（nei4 bu4 lei4），念的时候可得小心了 -\_-#。

****



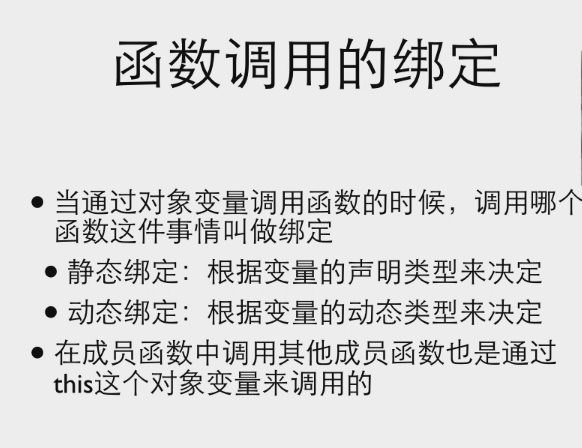




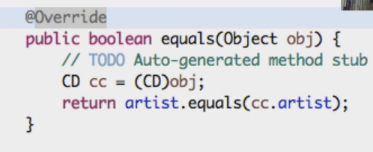


如果子类的方法覆盖了父类的方法，我们也说父类的那个方法在子类有了新的版本或者新的实现。覆盖的新版本具有与老版本相同的方法签名：相同的方法名称和参数表。因此，对于外界来说，子类并没有增加新的方法，仍然是在父类中定义过的那个方法。不同的是，这是一个新版本，所以通过子类的对象调用这个方法，执行的是子类自己的方法。

覆盖关系并不说明父类中的方法已经不存在了，而是当通过一个子类的对象调用这个方法时，子类中的方法取代了父类的方法，父类的这个方法被“覆盖”起来而看不见了。而当通过父类的对象调用这个方法时，实际上执行的仍然是父类中的这个方法。注意我们这里说的是对象而不是变量，因为一个类型为父类的变量有可能实际指向的是一个子类的对象。

当调用一个方法时，究竟应该调用哪个方法，这件事情叫做绑定。绑定表明了调用一个方法的时候，我们使用的是哪个方法。绑定有两种：**一种是早绑定，又称静态绑定**，这种绑定在编译的时候就确定了；**另一种是晚绑定，即动态绑定**。动态绑定在运行的时候根据变量当时实际所指的对象的类型动态决定调用的方法。Java缺省使用动态绑定。







**数据与表现分离**