Course 2 Java intermediate

Module 1 类与对象

- 对象是实体，需要被创建，可以为我们做事情
  - 对象（这只猫）
    - 表达东西或事件
    - 运行时相应消息（提供服务）
  - 数据：属性或状态
  - 操作：函数
  - 对象内部的数据是被外部的操作紧密包围的 -》封装
    - 由操作去保护内部的数据，而数据是不对外公开的
- 类是规范，根据类的定义来创建对象
  - 类（猫）
    - 定义所有猫的属性
    - 就是Java中的类型
    - 可以用来定义变量

定义类

类的组成

表示对象有什么的成员变量

表示对象能做什么的成员函数

一旦定义了类，我们就可以创建这个类的多个对象，这些对象都会做那个类所定义的动作（函数），但是各自具有不同的数据。

This

是成员函数的一个固有的特殊的本地变量，它表达了调用这个函数的那个对象

成员变量定义初始化

- 成员变量在定义的地方就可以给出初始值
- 没有给出初始值的成员变量会自动获得0值
    - 对象变量的0值表示没有管理任何对象，也可以主动给null值
- 定义初始化可以调用函数，甚至可以使用已经定义的成员变量

构造函数

- 如果有一个成员函数的名字和类的名字完全相同，则在创建这个类的每一个对象的时候会自动调用这个函数 -》构造函数
- 这个函数不能有返回类型

函数重载

- 一个类可以有多个构造函数，只要他们的参数表不同
- 创建对象的时候给出不同的参数值，就会自动调用不同的构造函数
- 通过this()还可以调用其他构造函数
- 一个类里的同名但参数表不同的函数构成了重载关系

Module 2 对象交互

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。

访问属性

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。

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；
    a. private只能用于成员变量和成员函数，不能用在函数里面（本地变量不能是private）

b. private只能在这个类内部使用（只有自己能访问）

   i. 类内部指类的成员函数和定义初始化

   ii. 这个限制是对类的而不是对对象的

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。

   a. public是任何人都可以访问的

      i. 任何人指的是在任何类的函数或定义初始化中可以使用

      ii. 使用指的是调用、访问或定义变量

如果函数或变量前没有定义private或者public，那么意味着friendly，即和它位于同一个包package的其他类都可以访问它

如果一个类是public的，任何人都可以用这个类的定义去定义变量

并且有一个要求，这个类必须处于源代码文件里，而这个源代码的文件名必须和这个类的名字相同

编译单元

一个.java文件，一个源代码文件是一个编译单元

编译单元是编译的时候一次对这一个编译单元做编译的动作

一个编译单元可以有很多个类，但是只有一个可以是public

包 - package

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。

import别的包的时候一般不全import进来（import display.*），为了避免有重名而引起的冲突，一般import具体的东西

像java.util.Scanner，包的名字中间的那个点代表的是文件系统里文件夹的层次

类变量

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。

Static

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。

通过一个对象去修改static成员的时候，别的对象里那个static成员也会被修改

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。

在一个static函数里不能访问non-static变量或函数，只能访问static函数和成员变量

Module 3 对象容器

顺序容器

容器 - Collection、Container

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。

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。

容器类

- `ArrayList<String> notes = new ArrayList<String>();`
- 容器类有两个类型
  - 容器的类型 → ArrayList
  - 元素的类型 → String

对象数组

int数组里每一个放的是整数

string数组里的每一个不是字符串本身，而是指向一个别的字符串的管理者

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！

其中每一个对象一开始都是null

for-each循环

对于int数组来说，k++不会起到任何作业，因为在循环的每一轮，拿出来的每一个k都只是对这个元素的一个复制品

```
int[] ia = new int[10];
for (int= i=0; i<ia.length; i++) {
  ia[i]=i;
}
for (int k:ia) {
  k++
}
```

而对于对象数组来说，不一样，第二遍会输出0

因为数组a里，每一个单元都指向了外面的v，所以v=a[0]对对象变量来说代表了v这个变量指向了a[0]所管理的那个对象。而v.set(0)是让v所指的那个对象里面的值变成0，所以会改变数组里的值

```
class Value {
  private int i;
  public void set(int i) {this.i=i;}
  public int get() {return i;}
}

public class NoteBook {
  public static void main(String[] args) {
    Value[] a = new Value[10];
    for (int i = 0; i<a.length; i++) {
      a[i] = new Value();
      a[i].set(i);
    }
    for ( Value v:a) {
      System.out.println(v.get());
      v.set(0);
    }
    for ( Value v:a) {
      System.out.println(v.get());
    }
```

对容器类来说，for-each循环也是可以用的

集合容器 set

集合就是数学中的集合的概念：所有的元素都具有唯一的值，元素在其中没有顺序。

```
HashSet<String> s = new HashSet<String>();
```

散列表 Hash table

传统意义上的Hash表，是能以int做值，将数据存放起来的数据结构。Java的Hash表可以以任何实现了hash()函数的类的对象做值来存放对象。

```
HashMap<Integer, String> coinnames = new HashMap<Integer, String>();
```

> 这是一个面向对象的世界，在这些容器里面，所有的类型都必须是对象（Integer，String），而不能是基本元素（int，double，float）

key是唯一的，如果多次放入同一个key和不同的value，key等于最后放入的那个value

也可以用for loop

```
for (Integer k : coinnames.keySet() ) { // k is every key
  String s = coinnames.get(k); // get the value of key k
  System.out.println(s);
}
```

1. java的整型分为4种：byte、short、int、long。其中byte占1个字节、short占2个字节、int占4个字节、long占8个字节
2. 当知道循环具体次数的时候使用for loop，当希望loop在具体条件下停下的时候使用while loop
3. 从初始条件开始，代入到循环体里运行，完成一次运行后，根据迭代条件在迭代中使用的对象取下一个可迭代的值，检查是否达到判断条件，如果没有，把取出的值代入循环体里运行，如果达到判断条件，则循环终止
4. 在内存里数组是连续的，链表是离散的
5. 二维数组即当一维数组的元素，是一个个一维数组时构成
6. 在java里用s1.equals(s2)去判断两个字符串是否相等，因为字符串变量不是字符串的所有者，而是字符串的管理者

Module 4 继承

面向对象程序设计语言有三大特性：封装、继承和多态性。继承是面向对象语言的重要特征之一，没有继承的语言只能被称作“使用对象的语言”。继承是非常简单而强大的设计思想，它提供了我们代码重用和程序组织的有力工具。

类是规则，用来制造对象的规则。我们不断地定义类，用定义的类制造一些对象。类定义了对象的属性和行为，就像图纸决定了房子要盖成什么样子。

一张图纸可以盖很多房子，它们都是相同的房子，但是坐落在不同的地方，会有不同的人住在里面。假如现在我们想盖一座新房子，和以前盖的房子很相似，但是稍微有点不同。任何一个建筑师都会拿以前盖的房子的图纸来，稍加修改，成为一张新图纸，然后盖这座新房子。所以一旦我们有了一张设计良好的图纸，我们就可以基于这张图纸设计出很多相似但不完全相同的房子的图纸来。

基于已有的设计创造新的设计，就是面向对象程序设计中的继承。在继承中，新的类不是凭空产生的，而是基于一个已经存在的类而定义出来的。通过继承，新的类自动获得了基础类中所有的成员，包括成员变量和方法，包括各种访问属性的成员，无论是public还是private。当然，在这之后，程序员还可以加入自己的新的成员，包括变量和方法。显然，通过继承来定义新的类，远比从头开始写一个新的类要简单快捷和方便。继承是支持代码重用的重要手段之一。

类这个词有分类的意思，具有相似特性的东西可以归为一类。比如所有的鸟都有一些共同的特性：有翅膀、下蛋等等。鸟的一个子类，比如鸡，具有鸟的所有的特性，同时又有它自己的特性，比如飞不太高等等；而另外一种鸟类，比如鸵鸟，同样也具有鸟类的全部特性，但是又有它自己的明显不同于鸡的特性。

如果我们用程序设计的语言来描述这个鸡和鸵鸟的关系问题，首先有一个类叫做“鸟”，它具有一些成员变量和方法，从而阐述了鸟所应该具有的特征和行为。然后一个“鸡”类可以从这个“鸟”类派生出来，它同样也具有“鸟”类所有的成员变量和方法，然后再加上自己特有的成员变量和方法。无论是从“鸟”那里继承来的变量和方法，还是它自己加上的，都是它的变量和方法。

继承

我们把用来做基础派生其它类的那个类叫做父类、超类或者基类，而派生出来的新类叫做子类。Java用关键字extends表示这种继承/派生关系：

```
class ThisClass extends SuperClass {
    //…
```

```
}
```

继承表达了一种is-a关系，就是说，子类的对象可以被看作是父类的对象。比如鸡是从鸟派生出来的，因此任何一只都可以被称作是一只鸟。但是反过来不行，有些鸟是鸡，但并不是所有的鸟都是鸡。如果你设计的继承关系，导致当你试图把一个子类的对象看作是父类的对象时显然很不合逻辑，比如你让鸡类从水果类得到继承，然后你试图说：这只本鸡是一种水果，所以这本鸡煲就像水果色拉。这显然不合逻辑，如果出现这样的问题，那就说明你的类的关系的设计是不正确的。Java的继承只允许单继承，即一个类只能有一个父类。

子类父类关系

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。

如果子类中有父类中有过的 完全相同名字的成员变量，在子类里，父类那个就被隐藏了；在子类里面当我们说那个变量的时候，我们说的子类自己的变量，父类那个看不见；当回到父类那边对父类进行操作时，它用的是父类自己的变量

多态变量

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。

子类和子类型

- 类定义了类型
- 子类定义了子类型
- 子类的对象可以被当作父类的对象来使用
    - 赋值给父类的变量
    - 传递给需要父类对象的函数
    - 放进存放父类对象的容器里

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**。

- 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。
- 它们可以保存的是声明类型的对象，或声明类型的子类的对象
- 当把子类的对象赋给父类的变量的时候，就发生了向上造型

多态变量有两个类型

1. 声明类型（静态类型、字面上能看出来的类型）
2. 动态类型，当程序运行到这里的时候，它里面管理的是什么类型的变量，那就是动态类型
3. 有可能是一致的，有可能是不一致的

向上造型

把一个类型的对象赋给另外一个类型的变量，这个过程叫造型

造型cast

- 子类的对象可以赋值给父类的变量
  - 注意！java中不存在对象对对象的赋值！！
- 父类的对象不能赋值给子类的变量！

```
Vehicle v;
Car c = new Var();
v = c; // 可以
c = v; // 编译错误
```

- 可以用造型：

```
c = (Car) v;
// 只有当v这个变量实际管理的是Car的时候才行
```

造型

- 用括号围起类型放在值的前面
- 对象本身并没有发生任何变化

- 所以不是“类型转换”

```
// 类型转换
int i = (int)10.2;
```

类型转化后，10.2会被转换成对应的int的值：10，而不再是10.2了

而造型是把你当作另外一个类型来看待，而不是把你改造成另外一个类型

- 运行时有机制来检查这样的转化是否合理
  - ClassCastException

向上造型

- 拿一个子类的对象，当作父类的对象来用
- 向上造型是默认的，不需要运算符
- 向上造型总是安全的

多态

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。

函数调用的绑定

- 当通过对象变量调用函数的时候，调用哪个函数这件事情叫做绑定
  - 静态绑定：根据变量的声明类型来决定

    - 动态绑定：根据变量的动态类型来决定（默认所有绑定都是动态绑定）
- 在成员函数中调用其他成员函数也是通过this这个对象变量来调用的

覆盖override

- 子类和父类中存在名称和参数表完全相同的函数，这一对函数构成覆盖关系
- 通过父类的变量调用存在覆盖关系的函数时，会调用变量当时所管理的对象所属的类的函数

Module 5 设计原则

消除代码复制的两个基本手段，就是函数和父类。

封装

要评判某些设计比其他的设计优秀，就得定义一些在类的设计中重要的术语，以用来讨论设计的优劣。对于类的设计来说，有两个核心术语：耦合和聚合。 耦合这个词指的是类和类之间的联系。之前的章节中提到过，程序设计的目标是一系列通 过定义明确的接口通信来协同工作的类。耦合度反映了这些类联系的紧密度。我们努力要获得 低的耦合度，或者叫作松耦合（loose coupling）。

耦合度决定修改应用程序的容易程度。在一个紧耦合的结构中，对一个类的修改也会导致对其他一些类的修改。这是要努力避免的，否则，一点小小的改变就可能使整个应用程序发生 改变。另外，要想找到所有需要修改的地方，并一一加以修改，却是一件既困难又费时的事情。 另一方面，在一个松耦合的系统中，常常可以修改一个类，但同时不会修改其他类，而且 整个程序还可以正常运作。

本周会讨论紧耦合和松耦合的例子。 聚合与程序中一个单独的单元所承担的任务的数量和种类相对应有关，它是针对类或方法 这样大小的程序单元而言的理想情况下，一个代码单元应该负责一个聚合的任务（也就是说，一个任务可以被看作是 一个逻辑单元）。一个方法应该实现一个逻辑操作，而一个类应该代表一定类型的实体。聚合 理论背后的要点是重用：如果一个方法或类是只负责一件定义明确的事情，那么就很有可能在 另外不同的上下文环境中使用。遵循这个理论的一个额外的好处是，当程序某部分的代码需要改变时，在某个代码单元中很可能会找到所有需要改变的相关代码段。

增加代码可扩展性

- 可以运行的代码！=良好的代码

- 对代码做维护的时候最能看出代码的质量

用封装来降低耦合

- room类和game类都有大量的代码和出口相关
- 尤其是game类中大量使用了room类的成员变量
- 类和类之间的关系称作耦合
- 耦合越低越好，保持距离是形成良好代码的关键

可扩展性

优秀的软件设计者的一个素质就是有预见性。什么是可能会改变的？什么是可以假设在软件的生命期内不会改变的？ 在游戏的很多类中硬编码进去的一个假设是，这个游戏会是一个基于字符界面的游戏，通 过终端进行输入输出，会永远是这样子吗？ 以后如果给这个游戏加上图形用户界面，加上菜单、按钮和图像，也是很有意思的一种扩 展。如果这样的话，就不必再在终端上打印输出任何文字信息。还是可能保留着命令字，还是会在玩家输入帮助命令的时候显示命令字帮助文本，但是可能是显示在窗口的文字域中，而不是使用System.out.println？

可扩展性的意思就是代码的某些部分不需要经过修改就能适应将来可能的变化。

用容器来实现灵活性

- room的方向是用成员变量来表示的，增加或减少方向就要改变代码
- 如果用hash table来表示方向，那么方向就不是hard coding了

框架加数据

从程序中识别出框架和数据，以代码实现框架，将部分功能以数据的方式加载，这样能在很大程度上实现可扩展性。

Module 6 抽象与接口

抽象

Shape类表达的是一种概念，一种共同属性的抽象集合，我们并不希望任何Shape类的对象会被创建出来。那么，我们就应该把这个Shape类定义为抽象的。

我们用abstract关键字来定义抽象类。抽象类的作用仅仅是表达接口，而不是具体的实现细节。抽象类中可以存在抽象方法。抽象方法也是使用abstract关键字来修饰。抽象的方法是不完全的，它只是一个方法签名而完全没有方法体。

如果一个类有了一个抽象的方法，这个类就必须声明为抽象类。如果父类是抽象类，那么子类必须覆盖所有在父类中的抽象方法，否则子类也成为一个抽象类。一个抽象类可以没有任何抽象方法，所有的方法都有方法体，但是整个类是抽象的。设计这样的抽象类主要是为了防止制造它的对象出来。

抽象函数/抽象类

- 抽象函数 — 表达概念而无法实现具体代码的函数
- 抽象类 — 表达概念而无法构造出实体的类

- 带有abstract修饰符的函数
- 有抽象函数的类一定是抽象类
- 抽象类不能制造对象
- 但是可以定义变量
    - 任何继承了抽象类的非抽象类的对象可以付给这个变量

实现抽象函数

- 继承自抽象类的子类必须覆盖父类中的抽象函数
- 否则自己成为抽象类

两种抽象

- 与具体相对
    - 表示一种概念而非实体
- 与细节相对
    - 表示在一定程度上忽略细节而着眼大局

数据与表现分离

- 程序的业务逻辑与表现无关

    - 表现可以是图形的也可以是文本的
    - 表现可以是当地的也可以是远程的

责任驱动的设计

- 将程序要实现的功能分配到合适的类/对象中去是设计中非常重要的一环

网格化

- 图形界面本身有更高的解析度
- 但是将画面网格化以后，数据就更容易处理了

接口

- 接口是纯抽象类
    - 所有的成员函数都是抽象函数
    - 所有的成员变量都是public static final
- 接口规定类长什么样，但不管里面有什么

```
public interface Cell {}
```

实现接口

- 类用extends，接口用implements

  ```
  public class Fox extends Animal implements Cell {};
  ```

- 类可以实现很多接口
- 接口可以继承接口，但不能继承类
- 接口不能实现接口

面对接口的编程方式

- 设计程序时先定义接口，再实现类
- 任何需要在函数间传入传出的一定是接口而不是具体的类
- 是java成功的关键之一，因为极适合多人同时写一个大程序
- 也是java被批评的要点之一，因为代码量膨胀起来很快

Module 7 控制反转与MVC模式

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。

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。

布局管理器

Swing：所有东西都叫做部件，frame是容器，容器本身也是一种部件，容器可以被加到另外一个容器里去

容器去管理部件用的手段叫布局管理器。

对frame来说默认的管理器是BorderLayout，有north、south、east、west、center

如果不指定加到哪儿，就会默认加到center；在frame里同一个位置同一时间只能有一个东西

控制反转

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

注入反转

- 由按钮公布一个守听者接口和一对注册/注销函数
- 你的代码实现那个接口，将守听者对象注册在按钮上
- 一旦按钮被按下，就会反过来调用你的守听者对象的某个函数

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

- 定义在别的类内部、函数内部的类
- 内部类能直接访问外部的全部资源
    - 包括任何私有的成员
    - 外部是函数时，只能访问那个函数里final的变量

匿名类

- 在new对象的时候给出的类的定义形成了匿名类
- 匿名类可以继承某类，也可以实现某接口
- Swing的消息机制广泛使用匿名类

MVC模式

JTable

用JTable类可以以表格的形式显示和编辑数据。JTable类的对象并不存储数据，它只是数据的表现

数据放在TableModel里

MVC设计模式

- 数据、表现和控制三者分离，各负其责
  - M = model
    - 保存和维护数据，提供接口让外部修改数据，通知表现需要刷新
  - V = view
    - 从模型获得数据，根据数据画出表现
  - C = control
    - 从用户得到输入，根据输入调整数据

control和view之间没有联系，是通过调整内部数据来改变

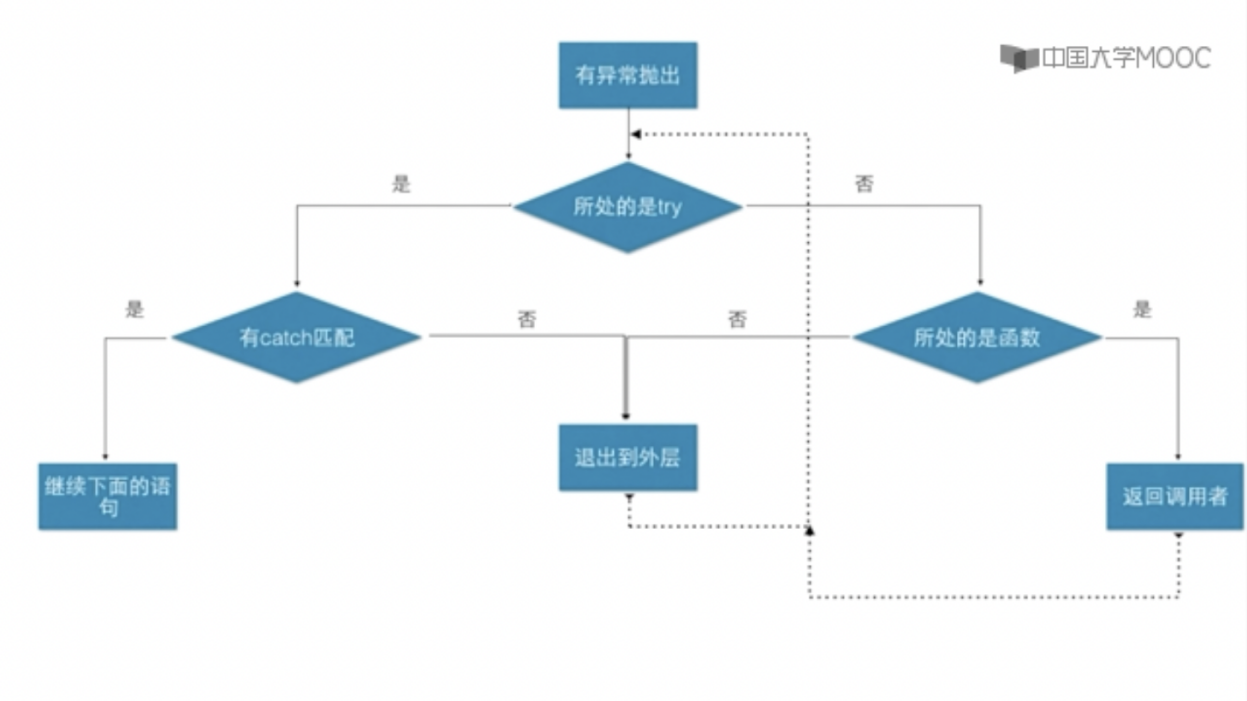
Module 8 异常处理与输入输出

异常

用try 和 catch捕捉和处理异常

捕捉异常

```
try (
  //可能产生异常的代码
} catch(Type1 id1) {
  //处理Type1异常的代码
} catch(Type2 id2) {
  //处理Type2异常的代码
} catch(Type3 id3) {
  //处理Type3异常的代码
}
```



捕捉到了做什么？

- 拿到异常对象之后
  - String getMessage（）；
  - String toString();
  - void printStackTrace();
- 但是肯定是回不去了，而具体的处理逻辑则取决于你的业务逻辑需要
- 如果在这个层面上需要处理，但是不能做最终的决定
  - 可以再度抛出

```
catch(Exception e) {
  System.err.println("An exception was thrown");
  throw e;
}
```

运行时刻异常

- 像ArrayIndexOutOfBoundsException这样的异常是不需要声明的
- 但是如果没有适当的机制来捕捉，就会最终导致程序终止

流 stream

流是一维单向的，是输入输出的方式

流的基础类

- InputStream
- OutputStream

文件流

- FileInputStream
- FileOutputStream
- 对文件作读写操作
- 实际工程中已经较少使用
    - 更常用的是以在内存数据或通信数据上建立的流，如数据库的二进制数据读写或网络端口通信
    - 具体的文件读写往往有更专业点类，比如配置文件和日志文件

文本流

reader/writer

- 二进制数据采用InputStream和OutputStream
- 文本数据采用Reader/Writer

在流上建立文本处理

```
PrintWriter pw = new PrintWriter(
  new BufferedWriter(
    new OutputStreamWriter(
      new FileOutputStream("abc.txt")));
```

LineNumberReader

可以得到行号

FileReader

汉字编码

UTF-8是一种变形的unicode编码形式

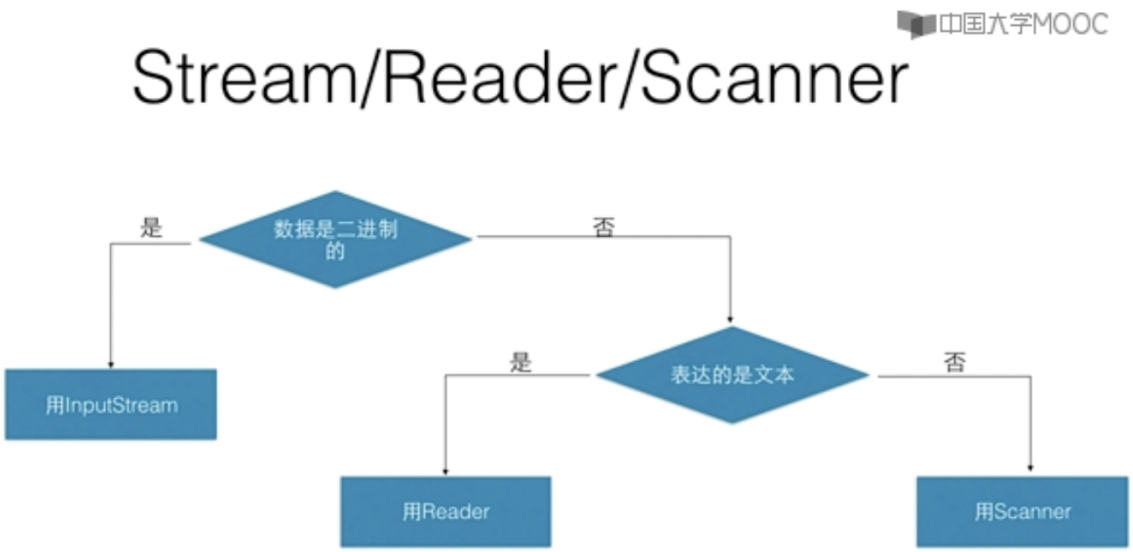
可以指定方式

```
new InputStreamReader(
    new FileInputStream("utf8.txt"),”utf8”));
```

格式化输入输出

Scanner

在INputStream或Reader上建立一个Scanner对象可以从流中的文本中解析出以文本表达的各种基本类型



流的应用

阻塞/非阻塞

- read（）函数是阻塞的，在读到所需的内容之前会停下来等
  - 使用read（）的更“高级”等函数，如nextInt（）、readLine（）都是这样
  - 所以常用单独的线程来做socket读的等待，或使用nio的channel选择机制
- 对于socket，可以设置SO时间

- setSoTimeout（int timeOut）