# 继承

（1）通过继承可以简化类的定义，我们已经在上面的例子中了解到了。

（2）Java 只支持单继承，不允许多重继承。在 Java 中，一个子类只能有一个父类，不允许一个类直接继承多个类，但一个类可以被多个类继承，如类 X 不可能既继承类 Y 又继承类 Z。

（3）可以有多层继承，即一个类可以继承某一个类的子类，如类 B 继承了类 A，类 C 又可以继承类 B，那么类 C 也间接继承了类 A。这种应用如下所示：

```
class A
{}
class B extends A
{}
class C extends B
{}
```

（4）子类继承父类所有的成员变量和成员方法，但不继承父类的构造方法。在子类的



构造方法中可使用语句 super（参数列表）调用父类的构造方法。如：我们为 Student 类增加一个构造方法，在这个构造方法中用 super 明确指定调用父类的某个构造方法。

### 4.1.3 覆盖父类的方法

在子类中可以根据需要对从父类中继承来的方法进行改造——方法的覆盖（也叫重写）。覆盖方法必须和被覆盖方法具有相同的方法名称、参数列表和返回值类型。

例如前面那个 Student 程序，它继承了 Person 类的 getInfo 方法，这个继承到的方法只能打印出学生的 name 和 age，不能打印出学生专有的信息，比如学校的名称等，这时就应该在类 Student 中重新编写一个 getInfo 方法，这就是方法的覆盖。程序修改后如下：

程序清单：Student1.java

```
class Person
{
   public String name;
   public int age;
   public void getInfo()
   {
    System.out.println(name);
        System.out.println(age);
   }
```

```
class Student extends Person
```



```
{
    String school=new String();
   public void study()
   {
       System.out.println("Studding");
   }
   public void getInfo()
   {
        super.getInfo();
        System.out.println(school);
```

```
    }

    public static void main(String[] args)
    {
        Person p=new Person();
        p.name="person";
        p.age=30;
         p.getInfo();

         Student s=new Student();
        s.name="student";
        s.age=16;
        s.school="清华大学";
        s.getInfo();
        s.study();
       }
}
```

运行结果：

```
person
30
student
16
清华大学
Studding
```

从以上运行结果可以看出，p.getInfo()这一句中所用到的方法是父类 Person 的，而 s.getInfo()这一句用的方法却是子类 Student 的。如果在子类中想调用父类中的那个被覆盖的方法，可以用 super.方法的格式，如程序中的 super.getInfo()。

# 4.2 抽象类与接口

## 4.2.1 抽象类

Java 中可以定义一些不含方法体的方法，它的方法体的实现交给该类的子类根据自己的情况去实现，这样的方法就是抽象方法，包含抽象方法的类就叫抽象类。一个抽象类中可以有一个或多个抽象方法。

抽象方法必须用 abstract 修饰符来定义，任何带有抽象方法的类都必须声明为抽象类。

1．抽象类定义规则

- 抽象类必须用 abstract 关键字来修饰；抽象方法也必须用 abstract 来修饰。
- 抽象类不能被实例化，也就是不能用 new 关键字去产生对象。
- 抽象方法只需声明，而不需实现。
- 含有抽象方法的类必须被声明为抽象类，抽象类的子类必须覆盖所有的抽象方法后才能被实例化，否则这个子类还是个抽象类。

2．抽象方法的写法

abstract 返回值类型 抽象方法（参数列表）；

3．抽象类和抽象方法的例子



```
abstract class A
{
    abstract int aa(int x,int y);
}
```

### 4.2.2 接口（interface）

如果一个抽象类中的所有方法都是抽象的，就可以将这个类用另外一种方式来定义，也就是接口定义。接口是抽象方法和常量值的定义的集合，从本质上讲，接口是一种特殊的抽象类，这种抽象类中只包含常量和方法的定义，而没有变量和方法的实现。

下面是一个接口定义例子：

```
public interface Runner
{
        int ID = 1;
        void run();
}
```

在 Java 中，设计接口的目的是为了让类不必受限于单一继承的关系，而可以灵活地同时继承一些共有的特性，从而达到多重继承的目的，并且避免了 C++中多重继承的复杂关系所产生的问题。多重继承的危险性在于一个类有可能继承了同一个方法的不同实现，对接口来讲决不会发生这种情况，因为接口没有任何实现。

一个类可以在继承一个父类的同时，实现一个或多个接口，extends 关键字必须位于 implements 关键字之前，如我们可以这样定义类 Student。

vv

下面是对接口的实现及特点的小结：

- 实现一个接口就是要实现该接口的所有方法（抽象类除外）。
- 接口中的方法都是抽象的。
- 多个无关的类可以实现同一个接口，一个类可以实现多个无关的接口。

```
public static void CallA(A a)
{
    B b= (B) a;
    b.func1();
    b.func2();
    b.func3();
}
```

3. instanceof 操作符

可以用 instanceof 判断是否一个类实现了某个接口，也可以用它来判断一个实例对象是否属于一个类。instanceof 的语法格式为：

对象 instanceof 类（或接口）

它的返回值是布尔型的，或真（true）、或假（false）。

还是用上面的代码来举例：

```
public static void CallA(A a)
{
    if(a instanceof B)
    {
        B b=(B)a;
        b.func1();
        b.func2();
        b.func3();
    }
    else
    {



        a.func1();
        a.func2();
     }
 }
```

这样改的目的是要判断一下传入的“人”，是不是属于“女人”这个类的。如果是，则强制类型转换，如果不是就不转换。

只要记住：一个“男人”肯定也是“人”，一个“人”却不一定是“男人”的道理，就非常容易理解父类和子类之间的转换关系了。

## 4.3.2 Object 类

```
public class Person
{
…
}
```

等价于：

```
public class Person extends Object
{
…
}
```

接口的用法

程序清单：Interface.java

```
interface PCI
{
    void start();
    void stop();
}
class NetworkCard implements PCI
{
    public void start()
    {
        System.out.println("Send ...");
    }
    public void stop()
    {
        System.out.println("Network Stop.");
    }
}
```

```
class SoundCard implements PCI
{
    public void start()
    {
        System.out.println("Du du...");
    }
    public void stop()
    {
        System.out.println("Sound Stop.");
    }
}
class MainBoard
{
    public void usePCICard(PCI p)
    {
        p.start();
        p.stop();
    }
}
,
class Assembler
{
    public static void main(String [] args)
    {
        MainBoard mb=new MainBoard();
        NetworkCard nc=new NetworkCard();
        mb.usePCICard(nc);
        SoundCard sc=new SoundCard();
        mb.usePCICard(sc);
    }
}
```

在上面的程序代码中，类 Assembler 就是计算机组装者，他买了一块主板 mb 和一块网卡 nc，一块声卡 sc，无论是网卡还是声卡，他们都使用的是主板的 usePCICard 方法。由于 NetworkCard 与 SoundCard 都是 PCI 接口的子类，所以，他们的对象能直接传递给 usePCICard 方法中的 PCI 接口的引用变量 p，在参数传递的过程中发生了隐式自动类型转换。

Try catch 工作过程

我们看到程序在出现异常后，系统能够正常地继续运行，而没有异常终止。在上面的程序代码中，我们对可能会出现错误的代码用 try…catch 语句进行了处理，当 try 代码块中的语句发生了异常，程序就会跳转到 catch 代码块中执行，执行完 catch 代码块中的程序代码后，系统会继续执行 catch 代码块后的其他代码，但不会执行 try 代码块中发生异常语句后的代码，如程序中的 System.out.println( "the result is" + result )；不会再被执行。可见 Java 的异常处理是结构化的，不会因为一个异常影响整个程序的执行。

```
catch(Exception e)
{
   System.out.println(e.getMessage());
}
```

catch 关键字后跟有一个用括号括起来的 Exception 类型的参数 e，这跟我们经常用到的如何定义一个函数接收的参数格式是一样的。括号中的 Exception 就是 try 代码块传递给 catch 代码块的变量类型，e 就是变量名，所以我们也可以将 e 改用成别的名称（如 ex 等），如下所示：

### 4.4.5 finally 关键字

在 try…catch 语句后，还可以有一个 finally 语句，finally 语句中的代码块不管异常是否



被捕获总是要被执行的。我们将上面的程序作如下修改，来看看 finally 语句的用法与作用。

### 4.4.6 异常的一些使用细节

1．一个方法被覆盖时，覆盖它的方法必须抛出相同的异常或异常的子类。

2．如果父类抛出多个异常，那么重写（覆盖）方法必须抛出那些异常的一个子集，也就是说，不能抛出新的异常。

### 4.6.2 类的访问控制

除了类中的成员有访问控制外，类本身也有访问控制，即在定义类的 class 关键字前加上访问控制符，但类本身只有两种访问控制，即 public 和默认，父类不能是 private 和 protected，否则子类无法继承。public 修饰的类能被所有的类访问，默认修饰（即 class 关键字前没有访问控制符）的类，只能被同一包中的所有类访问。

 多学两招：

只要在 class 之前，没有使用 public 修饰符，源文件的名称可以是一切合法的名称。带有 public 修饰符的类的类名必须与源文件名相同，读者可以想一想，一个 java 源文件中能否包含多个 public 的类呢？

### 4.6.3 Java 的命名习惯

养成良好的命名习惯，意义重大，如果大家的习惯都一样，我们就能够很容易使用别人提供的类，别人也很容易理解我们的类，对此，我送给读者一句话，“勿以善小而不为，勿以恶小而为之”！下面是 Java 中的一些命名习惯，假设 xxx，yyy，zzz 分别是一个英文单词的拼写。

- 包名中的字母一律小写，如：xxxyyyzzz。
- 类名、接口名应当使用名词，每个单词的首字母大写，如：XxxYyyZzz。
- 方法名，第一个单词小写，后面每个单词的首字母大写，如：xxxYyyZzz。
- 变量名，第一个单词小写，后面每个单词的首字母大写，如：xxxYyyZzz。
- 常量名中的每个字母一律大写，如：XXXYYYZZZ。

各种变量的声明习惯