蓝桥 LAN QIAO

# Java程序设计基础

## 抽象类和接口

# 抽象类和接口主要内容

- 9.1 抽象类（重点）
- 9.2 接口（重点）
- 9.3 内部类

# 教学目标

- 能理解抽象类的概念及作用
- 能理解接口的概念及作用
- 能准确区分接口继承与类继承的不同之处
- 能准确区分接口和抽象类的不同及联系
- 能使用接口和抽象类实现面向基类的编程思想
- 能理解内部类的概念及作用
- 能正确使用成员内部类，静态内部类，局部内部类、匿名内部类等四种类型的内部类
- 能使用接口和抽象类提高程序的复用度

# 抽象类简介

- 除了前面介绍的类以外，还存在一种特殊的类——抽象类。如果在类的定义中存在着一些抽象的方法，那么这种类就称为抽象类。语法上，抽象类是不能用于实例化对象的。
- 抽象类往往用来表示抽象概念。
- 举个例子，中国人（Chinese 类）和美国人（American 类）都有“吃饭”这个行为，因此可以先定义一个 Person 类，然后让 Chinese 和 American 都继承这个类。但如何在父类 Person 中定义“吃饭”这个方法呢？一般而言，中国人是用筷子吃饭，并且吃的是中餐；而美国人是用刀叉吃饭，吃的是西餐，显然二者对于“吃饭”这一行为的具体实现是不同的。因此，无法在父类 Person 中具体的定义“吃饭”这一方法。此时，就可以将 Person 定义成一个抽象类，并将“吃饭”这个行为定义成抽象方法（只有方法声明，但没有方法体的方法），然后再在子类 Chinese 和 American 中分别对“吃饭”进行具体的实现。

# 抽象类的语法形式

- 在面向对象分析和设计的过程中，经过封装和继承的分析之后，可以先创建一个抽象的父类，该父类定义了其所有子类共享的一般形式（如 Person 类），具体细节再由子类来完成（如 Chinese 类和 American 类）。 Java 中定义抽象类的语法形式如下。

  abstract class 类名{ }

- Java 也提供了一种特殊的方法，这个方法不是一个完整的方法，只含有方法的声明，没有方法体，这样的方法叫做抽象方法，其语法形式如下。

  访问修饰符 abstract 返回值 方法名( );

# 抽象类的三个语法特征



- 抽象类不能被直接实例化
- 抽象类的子类必须实现抽象方法（除非这个子类也是抽象类）
- 抽象类里可以有普通方法，也可以有抽象方法，但是有抽象方法的类必须是抽象类。

需要注意的是，抽象类里面也可以没有抽象方法，只是把原来的类前面加上 abstract 关键字，使其变为抽象类。

# 接口简介

- 接口是一系列抽象方法的集合，与抽象类不同，不可以声明普通方法。
- 虽然有人常说，接口是一种特殊的抽象类，但是在面向对象编程的设计思想层面，两者还是有显著区别的。抽象类更侧重于对相似的类进行抽象，形成抽象的父类以供子类继承使用；而接口往往在程序设计的时候，定义模块与模块之间应满足的规约或者定义一种标准，使各模块之间能协调工作。

# 接口的定义语法

- Java 接口定义的语法形式如下。

  [修饰符] interface 接口名 [extends] [接口列表]{ 接口体 }

- interface 前的修饰符是可选的，如果使用修饰符，则只能用 public 修饰符，表示此接口是公有的，在任何地方都可以引用它，这一点和类是相同的。 接口是和类同一层次的，所以接口名的命名规则参考类名命名规则即可。

- extends 关键词和类语法中的 extends 类似，用来定义直接的父接口。和类不同，一个接口可以继承多个父接口，当 extends 后面有多个父接口时，它们之间用逗号隔开。

# 接口的定义语法

- 接口体就是用大括号括起来的那部分，在接口体里定义接口的成员，包括常量和抽象方法。
- 类实现接口的语法形式如下：

  [修饰符] class 类名 implements 接口列表{ 类体 }

- 类实现接口用 implements 关键字，Java 中的类只能是单继承的，但一个 Java 类可以实现多个接口，这也是 Java 解决多继承的方法。

# 接口中不允许有实体方法

- 如果 接口中增加实体方法，编译时就会报错，提示接口中不能有实体方法

```
public interface EmailInterface {
    ...
    //显示邮件全部信息
    public void showEmail(){
    }
}
```

```
shiyanlou:Interface/ $ javac TestInterface2.java
./EmailInterface.java:15: 错误: 接口抽象方法不能带有主体
    public void showEmail(){
                          ^
1 个错误
```

# 接口中的成员变量

- 接口中可以有成员变量，修饰符默认为 public static final（即便不写修饰符也默认是这样），因为是常量所以必须在声明时对这些成员变量赋初值。
- 可以这样说，接口中的成员变量实际就是常量。接口中的抽象方法默认且必须是 public 的。

# 接口继承与合并

- 如果让一个类（Email）实现了多个接口（EmailInterface 和 PortInterface 接口），但是再在 EmailWriter 类的 writeEmail() 方法中传入对象时，形参就必须是这个类（Email），而不能是该类实现的某个接口。
- 根据多态的知识，多态对象只能调用定义该对象的类和其父接口中的方法，因此如果给 writeEmail() 方法设置的形参只是某一个接口（如 EmailInterface 接口）类型，那么该形参将无法调用其他接口（PortInterface 接口）中的方法。但如果将方法的参数类型设置为一个类而不是一个接口，这样做就不是我们推荐的面向接口编程了。

- 接口继承的方式解决上述问题，即用一个接口继承多个接口，从而实现接口合并的效果。

# 内部类

- 内部类是定义在类中的类，根据定义的类位置及修饰符的不同，分为了成员内部类，静态内部类，局部内部类、匿名内部类等四种类型。

- 成员内部类

成员内部类 InnerClass 可以直接访问外部类 OuterClass 中的属性和方法，对于成员内部类的使用，需要先生成外部类对象，然后再以 外部类对象.new 内部类() 的形式生成内部类对象。

- 静态内部类

静态内部类 InnerClass 只能访问外部类 OuterClass 的静态属性和静态方法，可以通过类名直接调用，如 类名.方法()**。与之类似，静态内部类中的方法也可以通过 **静态内部类名.方法() 的形式调用。如果测试类和静态内部类不在同一个.java 文件中，那么可以通过 外部类名.静态内部类.方法() 的形式调用。

# 内部类

- 局部内部类

JDK8.0 以后，局部内部类 InnerClass 在访问外部方法 method() 定义的变量时，可以省略给变量加 final 修饰。局部内部类只能在定义它的方法内部使用，局部内部类的定义并不会影响外部类方法的使用。

- 匿名内部类

我们在后续会学习多线程的知识，多线程对象是 Thread 类型的，启动线程的方法是 start() 方法，并且线程的执行逻辑可以以 Runnable 参数的形式放在 Thread 的构造方法中，即 new Thread(Runnable 对象).start() 就可以启动一个线程。但 Runnable 是一个接口，其中包含了一个 run() 抽象方法，如果用以前的做法，我们就需要先定义一个 Runnable 的实现类，然后再实现类中重写 run() 方法，最后再创建一个 Runnable 实现类的对象，并把这个对象传入到 Thread() 构造方法中。但显然这样做过于复杂，此时就可以通过匿名内部类的形式简化代码。

# 内部类

```
public class OuterClass4 {
    public static void main(String[] args) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                //线程执行逻辑...
            }
        }).start();
    }
}
```

# 笔试题

- 下列关于抽象类和接口描述正确的是（ ）。

  A 抽象类里必须含有抽象方法

  B 接口中不可以有普通方法

  C 抽象类可以继承多个类，实现多继承

  D 接口中可以定义局部变量

# 笔试题

- 在Java中，已经定义两个接口B和C，要定义一个实现这两个接口的类，以下那个语句正确（）

  A interface A extends B,C

  B interface A implements B,C

  C class A implements B,C

  D class A implements B,implements C

# 笔试题

- 下列选项正确实现ITest接口的是（）

```
interface ITest{
    String method(int i);
}
```

A class ITestImpl  implements ITest { String method (int i){}}

B class ITestImpl  implement { String method (int i){} }

C class ITestImpl  extends ITest {String method(int i){} }

D class ITestImpl  implements ITest { public  String method(int i){} }

# 笔试题

- 以下程序的输出结果为?

```
public class Base {
    public Base(String s) {
        System.out.print("B");
    }
}
public class Derived extends Base {
    public Derived(String s) {
        System.out.print("D");
    }
    public static void main(String[] args) {
        new Derived("C");
    }
}
```