抽象类：

随着继承的不断发生，子类相较于父类来说越来越具体。父类在设计时的需求是要具有通用性，而通用性越强，这使就是得越抽象，有时候一个过于抽象的类并不适宜实例化。

抽象方法也是一个道理。

abstract 修饰 { 类 / 方法

- 抽象方法只需声明，没有实现：abstract int ——(int a);
- 含有抽象方法的类必须被声明为抽象类
- 抽象类不能实例化
- 抽象类的子类没有重写全部的抽象方法时，其仍为子类
- abstract 不能修饰属性、私有方法、构造器、静态方法、final 方法

接口（一个特殊的抽象类）：

解决 Java 不能多重继承的问题

特点：

> interface 来定义
> 所有成员变量默认为 public static final
> 接口中的所有方法默认是 public abstract
> 没有普通的 构造器
> 接口没有构造器
> 接口采用多层继承机制

例：

默认为 public

```
public interface Runner{
    int id = 1;
```

```
    void start();
    public void run();
    void stop();
}
```

子类继承接口：

```
public class ___ implements 接口名 {

}
```

接口继承接口：

```
public interface ___ extend ___ {

}
```

接口的主要作用就是被其他的实现。

一个类可以同时 extend 和 implement，先继承后实现

有这样一种情况，一个抽象类需要新增抽象方法，但它已经有实例化的子类了，这时我们可以用抽象类实现一个接口来实现。

一个子类，它带有很重的父类色彩，例如人是灵长类的子类，人的一些行为特征都是灵长类共通的特征，但人也有些特征是灵长类不共通的，这些特征返倒与爬行类相似，那这些特征如何在人类中实现呢？我们当然可以直接在人类中实现，但有没有方法可以重用爬行类的代码呢？有，那就是接口，我们将这些的方法从爬行类中剥离出作为接口，交由人来与爬行类继承。

可以将相关性不大的类的一些方法剥离出作为接口 接口是一些方法的集合

相似

接口的多态性：

内部类：

```
public class Test3{
    int i;
    public int z;
    private int k;
    class A{
        int i;
        public void setTest3Fileds(){
            Test3.this.i = 1;
            Test3.this.z = 2;
            Test3.this.k = 3;
        }
        public void set(){
            this.i = 10;
        }
    }
    public void setInfo(){
        new A().setTest3Fileds(); //调用内部类的方法
                                  //要先new内部类对象
    }
}
```

内部类特性：

可以声明为final

和外部类不同，内部类可以声明为private和protected

可以声明为static [illegible]

可以声明为static，这时就不能再使用外层的非static成员
可以声明为abstract，可以被其他内部类继承，
非static的内部类的成员不能声明为static

内部类主要是解决java不能多重继承的问题

A要同时继承B、C的方法：

```
class A{
    private class InnerB extend B{

    }
    private class InnerC extend C{

    }
}
```