内部类

概述

概念

> 所谓内部类(InnerClass)，顾名思义，就是将一个类定义在另一个类的内部。内部的类称之为内部类。

主要特点

1. 内部类可以很好的实现隐藏，可以使用protected、private修饰符。
2. 内部类可以直接访问外部类的所有成员，包括私有的成员。
3. 外部类不能直接访问内部类的成员，必须首先要建立内部类的对象才可访问。

用途

1. 可以间接地去实现多继承
2. 可以避免修改接口而实现同一个类中两种同名方法的调用。

成员内部类

特点

1. 成员内部类属于外部类的实例成员，成员内部类可以有`public`,`private`,`default`,`protected`权限修饰符。
2. 在成员内部类中访问外部类的成员方法和属性，要使用`外部类名.this.成员方法`和`外部类名.this.成员属性`的形式。
3. 创建成员内部类的实例使用`外部类名.内部类名实例名=外部类实例名.new 内部类构造方法(参数)`的形式。

限制

1. 成员内部类不能与外部类重名。
2. 不能在成员内部类中定义`static`属性、方法和类（`static final`形式的常量定义除外）。因为一个成员内部类实例必然与一个外部类实例关联，`static`成员完全可以移到其外部类中去。

示例代码

```
public class MemberInnerClass {
    public static void main(String[] args) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();
        inner.showInner();
    }
}
class Outer{
    String name = "张三";
    int num = 3;
    public void showOuter(){
        System.out.println(name);
        System.out.println(num);
    }
    class Inner{
        String name = "李四";
        int num = 4;
        public void showInner(){
            int num = 5;
            System.out.println(name);
            System.out.println(num);
            System.out.println(this.num);
            System.out.println(Outer.this.num);
        }
    }
}
```

静态内部类

特点

1. 使用static修饰的成员内部类叫静态内部类
2. 静态内部类跟外部类没有任何关系，只是在生成类名和类定义时有影响
3. 静态内部类可以看做是与外部类平级的类，使用方式与外部类平级的类完全相同。
4. 创建静态内部类的实例使用`外部类名.内部类名实例名=new外部类名.内部类名(参数)`。

限制

1. 静态内部类不能与外部类重名。
2. 静态内部类不能访问外部类的非静态的属性和方法。
3. 外部类不能访问内部类的非静态的属性和方法。

示例代码

```
public class StaticInnerClass {
    public static void main(String[] args) {
        Outer.Inner inner = new Outer.Inner();
        inner.showInner();
    }
}
class Outer{
    String name = "张三";
    int num = 3;
    public void showOuter(){
        System.out.println(name);
        System.out.println(num);
    }
    static class Inner{
        String name = "张三";
        int num = 3;
        public void showInner(){
            System.out.println(name);
            System.out.println(num);
        }
    }
}
```

匿名内部类

特点

1. 匿名内部类是没有名称的内部类，没办法引用它们。
2. 必须在创建时，作为`new`语句的一部分来声明并创建它们的实例。
3. 匿名内部类必须继承一个类（抽象的、非抽象的都可以）或者实现一个接口。如果父类（或者父接口）是抽象类，则匿名内部类必须实现其所有抽象方法。
4. 匿名内部类中可以定义代码块，用于实例的初始化，但是不能定义静态代码块。

语法

```
new interface/superclass() { //类体}
```

说明:这种形式的new语句声明一个新的匿名类，它对一个给定的类进行扩展，或者实现一个给定的接口，并同时创建该匿名类的一个新实例。

示例代码

```
public class AnonymousInnerClass {
    public static void main(String[] args) {
        Person person = new Person();
        person.feed(new Animal() {

            @Override
            void eat() {
                System.out.println("喂养动物");
            }
        });
    }
}
abstract class Animal{
    abstract void eat();
}
class Person{
    public void feed(Animal animal){
        animal.eat();
    }
}
```

局部内部类

特点

1. 定义在代码块、方法体内、作用域（使用花括号“{}”括起来的一段代码）内的类叫局部内部类。
2. 局部内部类访问外部类的属性和方法使用`外部类名.this.属性名`和`外部类名.this.方法名(参数)`的形式。
3. 对外部世界完全隐藏，只能在其作用域内生成对象。

限制

1. 局部类不能加访问修饰符，因为它们不是类成员。
2. 成员内部类不能与外部类重名。
3. 局部内部类访问作用域内的局部变量，该局部变量需要使用final修饰。

示例代码

```java
public class LocalInnerClass {
    public static void main(String[] args) {
        Outer outer = new Outer();
        outer.showOuter();
    }
}
class Outer{
    int num1 = 1;
    public void showOuter(){
        int num2 = 2;//jdk1.8之前要加final
        class Inner{
            int num3 = 3;
            public void showInner(){
                System.out.println(num1);
                System.out.println(num2);
                System.out.println(num3);
            }
        }
        Inner inner = new Inner();
        inner.showInner();
    }
}
```

间接实现多继承

```java
/**
 * 间接实现多继承
 * @author Kevin
 *
 */
public class InnerClassDemo1 {

    public static void main(String[] args) {
        C c = new C();
        c.showA();
        c.showB();
    }
}
class A{
    public void showA(){
        System.out.println("A");
    }
}
class B{
    public void showB(){
        System.out.println("B");
    }
}
class C{
    private class A1 extends A{}
    private class B1 extends B{}

    public void showA(){
        new A1().showA();
    }

    public void showB(){
        new B1().showB();
    }
}
```

多个接口中同名方法的调用

```java
/**
 * 实现多个接口中同名方法的调用
 * @author Kevin
 *
 */
public class InnerClassDemo2 {

    public static void main(String[] args) {
        C c = new C();
        c.showA();
        c.showB();
    }
}
interface A{
    void show();
}
interface B{
    void show();
}
class C {
    private class A1 implements A{

        @Override
        public void show() {
            System.out.println("接口A的方法");
        }

    }
    private class B1 implements B{

        @Override
        public void show() {
            System.out.println("接口B的方法");
        }

    }
    public void showA(){
        new A1().show();
    }
    public void showB(){
        new B1().show();
    }
}
```