第十四章 内部类

1. 内部类的概述

1. 内部类：在类中又写了一个类
2. 内部的作用：为外部类提供服务

代码块

```
package com.powernode.innerclass10;
class OuterClass{
    private int x;
    private int y;

    public OuterClass(int x, int y) {
        this.x = x;
        this.y = y;
    }
    //做业务的方法
    //增
    public void insert(){
        System.out.println("OuterClass.insert");
    }
    //删
    public void delete(){
        System.out.println("OuterClass.delete");
    }
    //改
    public void update(){
        System.out.println("OuterClass.update");
    }
    //查
    public void query(){
        System.out.println("OuterClass.query");
    }
    //....
    //做运算的方法
    public void add(){
        System.out.println(x + y);
    }
    public void div(){
        System.out.println(x / y);
    }
```

```
35
36  }
37  public class Test {
38  }
```

- 以上类的设计，不符合高内聚，低耦合的原则

代码块

```
1   package com.powernode.innerclass11;
2
3   class OuterClass {
4       private int x;
5       private int y;
6
7       public OuterClass(int x, int y) {
8           this.x = x;
9           this.y = y;
10      }
11
12      /**
13       * 1.当前类不符合高内聚，低耦合的原则
14       * 2.因为当前类，有业务的处理方法和运算方法，这样类处理的功能就不单一了
15       *       1.单独写一个类
16       *       2.写一个内部类
17       *           1.参数不用传递
18       *           2.内部类开发的过程中，一般代表不多
19       */
20      //做业务的方法
21      //增
22      public void insert() {
23          System.out.println("OuterClass.insert");
24      }
25
26      //删
27      public void delete() {
28          System.out.println("OuterClass.delete");
29      }
30
31      //改
32      public void update() {
33          System.out.println("OuterClass.update");
34      }
35
36      //查
37      public void query() {
```

```
        System.out.println("OuterClass.query");
    }

    //....
    class IMath {
        //做运算的方法
        public void add() {
            System.out.println(x + y);
        }

        public void div() {
            System.out.println(x / y);
        }
    }

}

public class Test {
}
```

2. 内部类的分类

2.1 实例内部类

2.1.1 访问实例内部类的实例方法

代码块

```
package com.powernode.innerclass12;
//外部类
class OuterClass{
    private int x;
    private int y;

    public OuterClass(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public void add(){
        System.out.println("OuterClass.add");
    }
    //内部类
    class InnerClass{
```

```
16          public InnerClass() {
17              System.out.println("InnerClass.InnerClass");
18          }
19          public int max(){
20              return x > y ? x : y;
21          }
22      }
23      //一个外部类可以包含多个内部类
24      class InnerClass02{}
25  }
26  public class Test {
27      public static void main(String[] args) {
28          //1.创建外部类对象
29          OuterClass outerClass = new OuterClass(2,3);
30          //2.语法：外部类.内部类  内部类对象  =  外部类对象.new 内部类构造器()
31          OuterClass.InnerClass innerClass = outerClass.new InnerClass();
32          System.out.println(innerClass.max());
33          /**
34           * 1.内部类会生成单独的.class文件
35           * 2.内部类.class文件命名规则：外部类$内部类.class
36           * 3.类命名的时候尽量不使用$
37           */
38      }
39  }
```

2.1.2 访问实例内部类的实例变量

代码块

```
1   package com.powernode.innerclass13;
2   class OuterClass{
3       private int x  = 10;
4       private int y;
5       class InnerClass{
6           public int x = 30;
7           public void method(){
8               System.out.println("x = " + x);//30
9               System.out.println("this.x = " + this.x);//30
10              System.out.println("InnerClass.this.x = " + InnerClass.this.x);//30
11              System.out.println("OuterClass.this.x = " + OuterClass.this.x);//10
12          }
13      }
14  }
15  public class Test {
16      public static void main(String[] args) {
17          OuterClass outerClass = new OuterClass();
```

```
        OuterClass.InnerClass innerClass = outerClass.new InnerClass();
        System.out.println(innerClass.x);//30
        innerClass.method();
    }

}
```

2.1.3 访问实例内部类的静态成员

代码块

```
package com.powernode.innerclass14;
class OuterClass{
    private int x = 10;
    private int y = 20;
    class InnerClass{
        public static int x = 30;
        public static void method(){
            System.out.println("InnerClass.method");
        }
    }
}
public class Test {
    public static void main(String[] args) {
        //1.访问实例内部类的静态属性
        System.out.println(OuterClass.InnerClass.x);
        //2.访问实例内部类的静态方法
        OuterClass.InnerClass.method();
    }
}
```

2.2 静态内部类

代码块

```
package com.powernode.innerclass15;
class OuterClass{
    private int x;
    private int y;
    private static  int z = 60;

    public OuterClass(int x, int y) {
        this.x = x;
```

```
9           this.y = y;
10      }
11
12      public int getX() {
13          return x;
14      }
15
16      public static int getZ() {
17          return z;
18      }
19      /**
20       * 1.静态内部类,在自己的作用域内，就当成一个普通的类
21       *    1.实例成员
22       *    2.静态成员
23       */
24      static class InnerClass{
25          public int x = 20;
26          public static int n = 30;
27
28          public void method(){
29              //System.out.println(y);1.静态内部类中，实例方法不可以访问外部类实例变量
30              System.out.println(z);//2.静态内部类，实例方法可以访问外部的静态变量
31              System.out.println(getZ());//3.静态内部类，实例方法可以访问外部的静态方
    法
32              //System.out.println(getX());//4.静态内部类，实例方法不可以访问外部的实
    例方法
33              //5.静态内部类中，不持有外部类引用
34              //System.out.println(OuterClass.this);
35
36          }
37          public static int max(int x,int y){
38              return x > y ? x : y;
39          }
40      }
41  }
42  public class Test {
43      public static void main(String[] args) {
44          //1.访问静态内部类的静态成员
45          System.out.println(OuterClass.InnerClass.n);
46          System.out.println(OuterClass.InnerClass.max(2, 3));
47          //2.访问静态内部类的实例成员
48          OuterClass.InnerClass innerClass = new OuterClass.InnerClass();
49      }
50  }
```

2.3 局部内部类

- 局部内部类，就是在方法中声明了一个类

代码块

```
package com.powernode.innerclass16;
class OuterClass{
    public void method( int x,int y){
        /**
         * - 局部内部类就是一个局部的成员（类似于局部变量）
         *   1.不可以使用public ,private和protected修饰
         *   2.可以使用final修饰
         *   3.不可以static修饰
         */
        class InnerClass{
            public int x = 10;
            public static int y = 20;

            public void method(){
                System.out.println("x = " + x);
                System.out.println("y = " + y);
            }

            public static int max(int x, int y){
                return x > y ? x : y;
            }
        }
        //局部内部类，在这类创建对象
        InnerClass innerClass = new InnerClass();
        System.out.println(innerClass.x);
        innerClass.method();

        System.out.println(InnerClass.y);
        System.out.println(InnerClass.max(2, 3));
    }
}
public class Test {
    public static void main(String[] args) {
        new OuterClass().method(2,3);
    }
}
```

2.4 匿名内部类（重点）

2.4.1 不使用匿名内部类的代码冗余

```java
package com.powernode.innerclass17;
interface Flyer{
    void fly();
}
class Bird implements Flyer{

    @Override
    public void fly() {
        System.out.println("Bird.fly");
    }
}
public class Test {
    public static void main(String[] args) {
        Flyer flyer = new Bird();
        flyer.fly();
    }
}
```

2.4.2 使用匿名内部类来解决代码冗余

代码块

```java
package com.powernode.innerclass18;
interface Flyer{
    void fly();
 /*   void land();*/
}

public class Test {
    public static void main(String[] args) {

        /*Flyer flyer = new Flyer() {
            @Override
            public void fly() {
                System.out.println("Test.fly");
            }
        };
        flyer.fly();*/
        /**
         * 1.接口只有一个抽象方法
         * 2.实现类只创建一个对象，且调用一次重写的方法
         * 3.类似这种情况，我们通常使用匿名内部类来解决
         */
        new Flyer() {
            @Override
            public void fly() {
```

```
25                  System.out.println("Test.fly");
26              }
27          }.fly();
28      }
29  }
```

名称	修改日期	类型	大小
Flyer.class	2025/8/4 10:28	CLASS 文件	1 KB
Test$1.class	2025/8/4 10:28	CLASS 文件	1 KB
Test.class	2025/8/4 10:28	CLASS 文件	1 KB