内部类

内部类顾名思义就是创建在内部的类，在Java中，可以将一个类定义在另一个类里面或者一个方法里面，这样的类称为内部类。广泛意义上的内部类一般来说包括这四种：成员内部类、局部内部类、匿名内部类和静态内部类

一、成员内部类

1️⃣ 成员内部类和成员变量一样，是定义在一个类里面，也就是类里面在定义一个类，**而且它和成员变量一样是属于一个对象，不是属于类的**，所以在使用的时候就需要创建一个对象

```
public class Person {
    public class Inner{
        public void test(){
            System.out.printf("我是成员内部类");
        }
    }
}
```

调用

```
public class Main {

    public static void main(String[] args)  {
        Person person = new Person(); //需要先创建类
        Person.Inner inner = person.new Inner(); //内部类的类型为外层类.内部类 使用对象来直接new一个内部类
        inner.test(); //调用内部类的成员方法
    }
}
```

⚠️ **成员内部类和成员变量一样也是可以使用访问权限的，如果改为`private`，就和成员变量一样无法在外部使用**

2️⃣ **在成员变量中，是可以直接访问导外层类的变量的**，这是因为内部类默认包含了一个指向外部类对象的引用。这个引用使得内部类能够访问外部类的成员，包括私有成员。这种设计允许内部类在需要时使用外部类的状态，提供了一种强大的机制来实现复杂的设计模式，如回调或生成器模式

```java
public class Person {
    public String name;

    public Person(String name) {
        this.name = name;
    }

    public class Inner{
        public void test(){
            System.out.printf("我是成员内部类,我叫"+name);
        }
    }
}
```

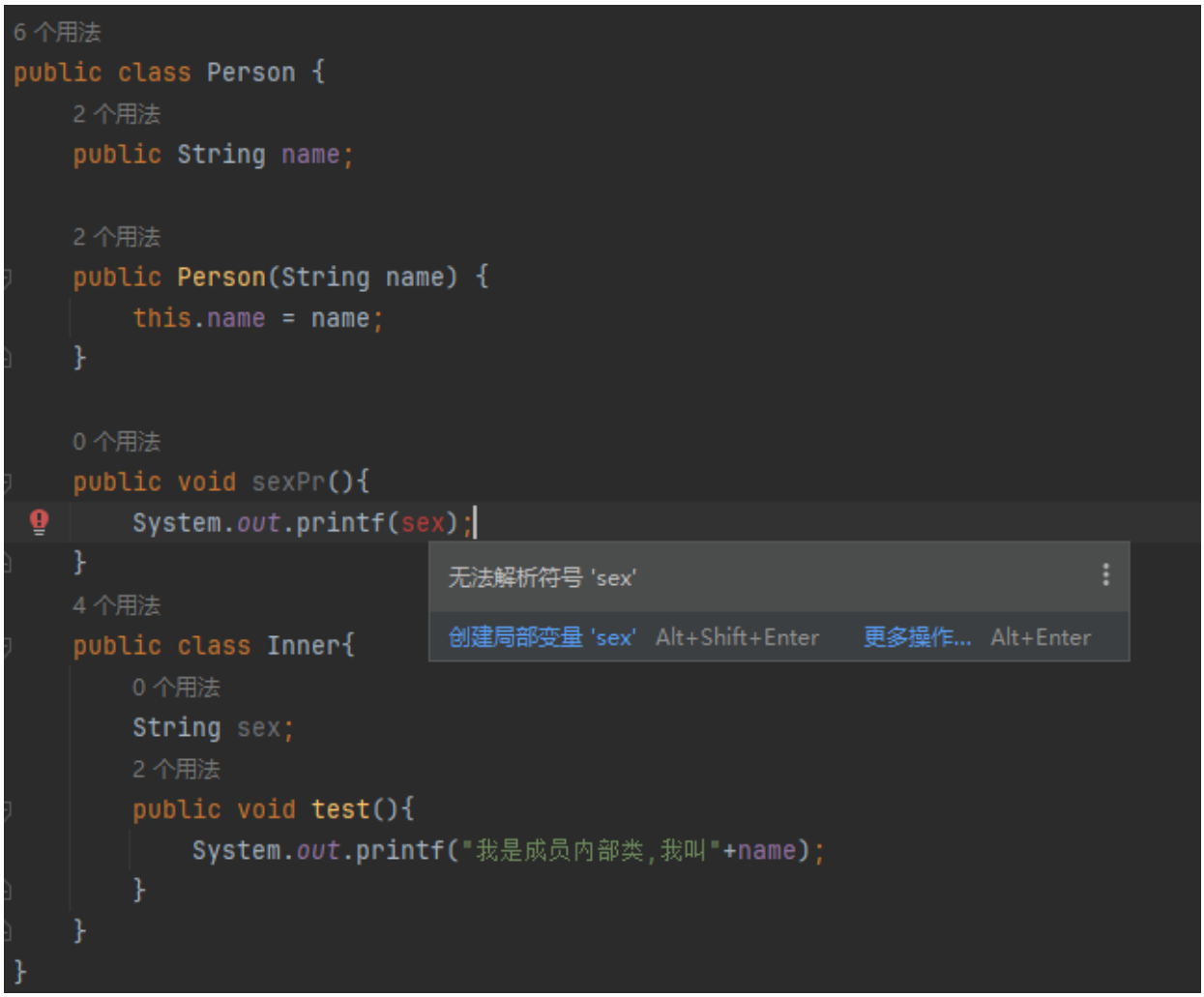
调用

```java
public class Main {

    public static void main(String[] args)  {
        Person person = new Person("outer"); //传入成员变量
        Person.Inner inner = person.new Inner();
        inner.test(); //输出
    }
}


输出:
我是成员内部类,我叫outer
```

3 每个类可以创建一个对象，每个对象中都有一个单独的类定义，又可以通过这个成员类创建对象，这样套娃

```java
public class Main {

    public static void main(String[] args)  {
        Person personOne = new Person("outerOne");
        Person.Inner innerOne = personOne.new Inner(); //使用personOne创建的对象
        innerOne.test();
        Person personTwo = new Person("outerTwo");
        Person.Inner innerTwo = personTwo.new Inner(); //使用personTwo创建的对象
        innerTwo.test();
    }
}

```

4 外部类是不可以访问到内部类的成员变量的，因为内部类的定义和使用都在外部类的内部。内部类的实例化通常依赖于外部类的实例，且内部类可能访问了外部类的私有成员，因此直接从外部类访问内部类的成员会破坏封装性，可能导致内部状态的不一致。



5 如果内部类中也定义了同名的变量，如果未指定this,默认是就近原则范围

```java
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }
    public class Inner{
        String name;

        public Inner(String name) {
            this.name = name;
        }

        public void test(String name){
            System.out.println("我是就近原则的name = "+name); //默认就近原则
            System.out.println("成员内部类的name = "+this.name); //this表示当前类的对象
            System.out.println("外部部类的name = "+Person.this.name); //这是外部类的对象参数
        }
    }
}
```

方法调用和`super`关键字也是一样

```
    public class Inner {

        String name;
        public void test(String name){
            this.toString();        //内部类自己的toString方法
            super.toString();    //内部类父类的toString方法
            Person.this.toString();   //外部类的toSrting方法
            Person.super.toString();  //外部类父类的toString方法
        }
    }
```

成员内部类在某些情况下使用起来非常麻烦，一般只会在类的内部自己使用

二、静态内部类

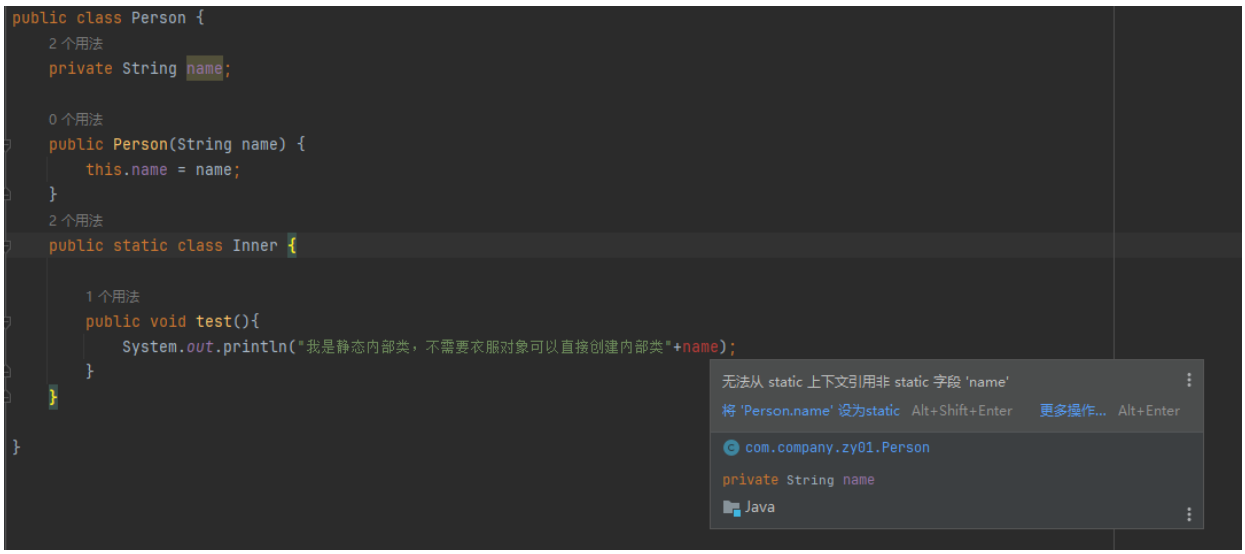
1️⃣ 静态内部类和静态方法一样是始于类的，不是属于对象的

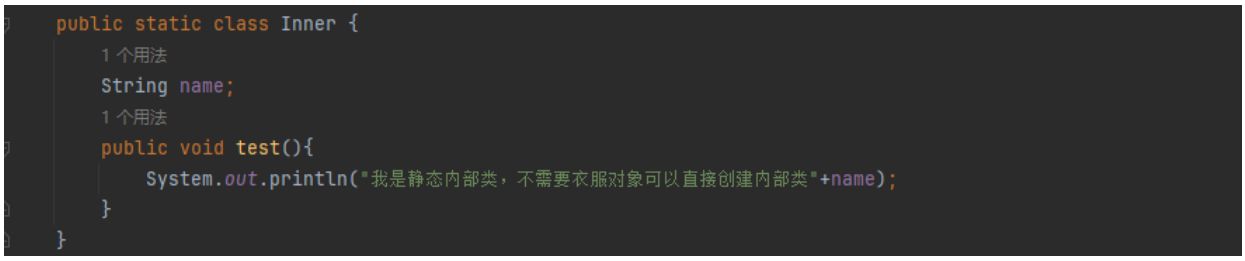
```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }
    public static class Inner {

        public void test(){
            System.out.println("我是静态内部类，不需要衣服对象可以直接创建内部类");
        }
    }

}
```

同样的不需要依附对象(也就是外部类)就可以直接调用

```
public class Main {

    public static void main(String[] args)  {
        Person.Inner inner = new Person.Inner();
        inner.test();
    }
}
```

2 ⚠ 静态内部类是静态的，是无法访问到外部类的内容的，但是可以访问内部类自己的内容





静态内部类和静态成员变量差不多，不依附任何对象，那么肯定就是不能访问外部类的变量了，我都不需要靠它了，肯定用不着

三、局部内部类

1 局部内部类就像局部变量一样，**可以在方法中直接定义**，也就是说局部内部类就是定义在方法中的类

```
1  public class Person {
2
3      //成员方法
4      public void test(){
5          //局部内部类
6          class Inner{
7
8          }
9      }
10 }
```

2 局部内部变量的作用域只在一个方法中

```
1  package com.company.zy01;
2
3  public class Person {
4
```

```
    //成员方法
    public void test(){
        //局部内部类
        class Inner{
            public void innerTest(){
                System.out.printf("我是局部方法类");
            }
        }
        Inner inner = new Inner(); //直接在方法内部new一个对象
        inner.innerTest();//调用
    }
}

```

调用

```
package com.company.zy01;


public class Main {

    public static void main(String[] args)  {
        Person p1 = new Person();
        p1.test();
    }
}

```

局部变量的使用频率很低，基本上不会使用到，可以了解了解

四、匿名内部类

匿名内部类是使用频率非常高的一种内部类，它是局部内部类的简化版。

1️⃣ 之前提到过抽象类 和 接口，他们可能会含有偶像抽象方法需要子类去实现，当时是说明不能使用new直接去创建一个抽象类和一个接口对象，但是现在可以使用匿名内部类

```
//创建一个匿名类
public abstract class Student {
    public abstract  void  studentTest();
}
```

在正常况下，要创建一个子类来继承这个抽象类，然后实现这个抽象方法，使用子类来创建对象才可以正常使用，但是现在可以直接使用匿名内部类

```
package com.company.zy01;
```

```
2
3
4  public class Main {
5
6      public static void main(String[] args)  {
7          Student s1 = new Student() { //使用匿名内部类直接实现抽象方法
8              @Override
9              public void studentTest() {
10                 System.out.printf("我是匿名内部类的实现");
11             }
12         };
13         s1.studentTest(); //调用抽象方法
14     }
15 }
16
```

2 匿名内中可以使用类中的属性(因为本质上就相当于对应类型的子类)也可以直接定义成员变量

```
1  package com.company.zy01;
2
3  public abstract class Student {
4      String name = "innerOne"; //定义成员变量
5      public abstract  void  studentTest();
6  }
7
```

```
1  package com.company.zy01;
2
3
4  public class Main {
5
6      public static void main(String[] args)  {
7          Student s1 = new Student() {
8              int a;
9              @Override
10             public void studentTest() {
11                 System.out.printf("我是匿名内部类的实现 = " + name);
12             }
13         };
14         s1.studentTest(); //调用抽象方法
15     }
16 }
17
```

3 接口也可以使用匿名类的形式，直接创建一个匿名的接口实现类

```
1  //创建接口
2  public interface Study {
3      public void study();
4  }
```

```java
public class Main {

    public static void main(String[] args)  {
        //匿名接口实现类
        Study study  = new Study() {
            @Override
            public void study() {
                System.out.println("我是学习方法");
            }
        };
        study.study();
    }
}
```

⚠ 不仅仅只是接口和抽象类可以创建匿名内部类，普通的也可以，但是没有什么意义

五、Lambda表达式

Lambda表达式是前面的匿名类的简写版本（我现在是这么理解的）

1 如果一个接口中有且只有一个待实现的抽象方法，就可以将匿名内部类简写为Lambda表达式

```java
public class Main {

    public static void main(String[] args)  {
        //匿名接口实现类Lambda表达式
        Study study  = () -> System.out.println("我是学习方法");
        study.study();
    }
}
```

2 Lambda简写的规则

- 标准格式为：`([参数类型 参数名称,]...) -> { 代码语句，包括返回值 }`
- 和匿名内部类不同，Lambda仅支持接口，不支持抽象类
- 接口内部必须有且仅有一个抽象方法（可以有多个方法，但是必须保证其他方法有默认实现，必须留一个抽象方法出来）

3 如果有传参和返回值

```java
public interface Study {
    public String study(String a); //修改返回值类型
}

```

```
public class Main {

    public static void main(String[] args)  {
        //匿名接口实现类Lambda表达式
        Study study  = (a) -> { //传参
            System.out.println("我是学习方法"); //{}内部实际上就是一个方法体
            return "今天学习了"+a;
        };
        System.out.printf(study.study("Java的内部类"));
    }
}
```

4️⃣ 如果方法体中只有一个返回语句，可以直接省区花括号和`return`

```
public class Main {

    public static void main(String[] args)  {
        //匿名接口实现类Lambda表达式
        Study study  = (a) -> {
            return "今天学习了"+a;
        };
        System.out.printf(study.study("Java的内部类"));
    }
}
```

简化后

```
public class Main {

    public static void main(String[] args)  {
        //匿名接口实现类Lambda表达式
        Study study  = (a) -> "今天学习了"+a;
        System.out.printf(study.study("Java的内部类"));
    }
}
```

而且如果只有一个传参还可以简化`()`

```
public class Main {

    public static void main(String[] args)  {
        //匿名接口实现类Lambda表达式
        Study study  = a -> "今天学习了"+a;
        System.out.printf(study.study("Java的内部类"));
    }
}
```

如果使用的是`idea`他会提示你转换

六、方法引用

方法引用就是将一个已经实现的方法，直接作为接口抽象方法实现，前提是方法定义一样才行

比如我现在有个接口定义了一个求和的方法

```
public interface Study {
    int sum(int a,int b);
}
```

调用的时候可以直接使用`lambda`表达式

```
public class Main {

    public static void main(String[] args)  {
        //匿名接口实现类Lambda表达式
        Study study  = (a,b) -> a+b;
    }
}
```

但是不够简洁，就可以使用方法引用，在Integer类中默认就提供了int值的求和方法，定义也是和我们定义一样

```
    public static int sum(int a, int b) {
        return a + b;
    }
```

所以就可以直接拿来用

```
public class Main {

    public static void main(String[] args)  {
        //匿名接口实现类Lambda表达式
        Study study  = Integer::sum;
        System.out.println(study.sum(10,20));
    }
}
```

方法引用其实就是相当于将其他实现好的方法，直接作为接口重点抽象方法的实现，任何方法都可以通过方法引用来实现

比如我在`Main`中添加了一个方法，如果我想用接口调用，就需要创建一个对象，根据对象来使用他的方法

```
public class Main {

     public static void main(String[] args)  {
         Main main = new Main();
         //匿名接口实现类Lambda表达式
```

```
        Study study  = main::szy;

    }
    public String szy(){
        return "Szy Is Pig";
    }
}


//需要修改接口方法的定义，如果接口方法定义和方法引用定义不一样会报错
package com.company.zy01;

public interface Study {
    String main();
}

```

不仅仅是这个构建方法也可以被引用

```

public class Main {

    public static void main(String[] args)  {
        Study study = String::new;

    }

}
```