

- 1 内部类的语法规则
- 2 局部内部类
- 3 匿名内部类
- 4 静态内部类

内部类是定义在另一个类中的类。使用内部类有两个原因：

1. 内部类可以对同一个包中的其他类隐藏。
2. 内部类方法可以访问定义这个类的作用域中的数据，包括原本私有的数据。

1 内部类的语法规则

下面展示一个内部类的实例：

```
1 public class TalkingClock
2 {
3     private int interval; // 发出通知的时间间隔
4     private boolean beep; // 开关铃声的标志
5
6     public TalkingClock(int interval, boolean beep)
7     {
8         this.interval = interval;
9         this.beep = beep;
10    }
11
12    public void start()
13    {
14        TimePrinter listener = new TimePrinter();
15        Timer timer = new Timer(interval, listener);
16        timer.start();
17    }
18
19    public class TimePrinter implements ActionListener // 内部类
20    {
21        public void actionPerformed(ActionEvent event)
22        {
23            System.out.println("At the tone, the time is " +
24                               Instant.ofEpochMilli(event.getWhen()));
25            if (beep)
26                Toolkit.getDefaultToolkit().beep();
27        }
28    }
29 }
```

这里的 `TimePrinter` 类位于 `TalkingClock` 类内部，这并不意味着每个 `TalkingClock` 对象都有一个 `TimePrinter` 实例字段，`TimePrinter` 对象是由 `TalkingClock` 类中的 `start` 方法构造的。

`TimePrinter` 类中的 `actionPerformed()` 方法使用了变量 `beep`，这个 `beep` 变量实际上是创建 `TimePrinter` 对象的 `TalkingClock` 对象内部的字段。内部类方法可以访问自身的数据字段，也可以访问创建它的外围类对象的数据字段。为此，内部类对象有一个隐式引用，指向创建它的外围类对象。这个引用在内部类的定义中是不可见的。

外围类的引用在构造器中设置。编译器会修改所有的内部类构造器，添加一个对应外围类引用的参数。例如，上面的 `TimePrinter` 类没有定义构造器，编译器会自动生成一个无参数构造器，并添加一个外围类 `TalkingClock` 类型的参数，代码如下：

```
public TimePrinter(TalkingClock clock)
{
    TalkingClock.this = clock;
}
```

内部类中对外围类引用的调用是隐式的，可以通过 `外围类名.this` 显式调用。例如上面的 `TalkingClock.this` 就是对外围类引用的显式调用。为了明显区分内部类自身的字段和外围类的字段，可以在内部类方法中显式使用外围类引用。例如，`TimePrinter` 内部类的 `actionPerformed()` 方法可以修改如下：

```
public void actionPerformed(ActionEvent event)
{
    System.out.println("At the tone, the time is " +
        Instant.ofEpochMilli(event.getWhen()));
    if (TalkingClock.this.beep)
        Toolkit.getDefaultToolkit().beep();
}
```

反过来，可以采用以下语法更加明确地调用内部类的构造器：`外围类对象.new 内部类名(参数)`。例如：

```
public void start()
{
    TimePrinter listener = this.new TimePrinter();
    Timer timer = new Timer(interval, listener);
    timer.start();
}
```

在外围类的作用域之外，可以这样引用内部类：`外围类名.内部类名`。例如：

```
TalkingClock jabberer = new TalkingClock(1000, true);
TalkingClock.TimePrinter listener = jabberer.new TimePrinter();
```

内部类中声明的所有静态字段都必须是 `final`，并初始化为一个编译时常量。内部类中不能有静态方法。

2 局部内部类

在 `TalkingClock` 类中，只在 `start()` 方法中使用了 `TimePrinter` 内部类。当遇到这种情况时，可以在方法中局部地定义这个类。例如：

```
1 public void start()
2 {
3     class TimePrinter implements ActionListener
4     {
5         public void actionPerformed(ActionEvent event)
6         {
7             System.out.println("At the tone, the time is " +
8                                 Instant.ofEpochMilli(event.getWhen()));
9             if (beep)
10                 Toolkit.getDefaultToolkit().beep();
11         }
12     }
13
14     TimePrinter listener = new TimePrinter();
15     Timer timer = new Timer(interval, listener);
16     timer.start();
17 }
```

声明局部类时不能有访问说明符（`public` 或 `private`）。局部类的作用域被限定在声明这个局部类的块中，它对外部世界完全隐藏。除了 `start()` 方法之外，没有任何方法知道 `TimePrinter` 类的存在。

局部内部类不仅能够访问外部类的字段，还可以访问局部变量。不过，这些局部变量必须是事实最终变量，一旦赋值就不会改变。例如：

```

1 public class TalkingClock
2 {
3     public void start(int interval, boolean beep)
4     {
5         class TimePrinter implements ActionListener
6         {
7             public void actionPerformed(ActionEvent event)
8             {
9                 System.out.println("At the tone, the time is " +
10                     Instant.ofEpochMilli(event.getWhen()));
11                 if (beep)
12                     Toolkit.getDefaultToolkit().beep();
13             }
14         }
15
16         TimePrinter listener = new TimePrinter();
17         Timer timer = new Timer(interval, listener);
18         timer.start();
19     }
20 }

```

`actionPerformed()` 方法中使用的 `beep` 变量是 `start()` 方法的参数变量。编译器检测对局部变量的访问，在局部内部类中为每一个变量建立相应的实例字段，并将局部变量复制到构造器，从而能初始化这些实例字段。在这个例子中，编译器会在 `TimePrinter` 类中自动添加 `beep` 变量对应的字段，在构造 `TimePrinter` 对象时用 `beep` 参数变量初始化这个字段。

3 匿名内部类

使用局部内部类时，如果只是创建这个类的一个对象而不需要其他操作，甚至可以省略内部类的名字，这样的类称为**匿名内部类**。匿名内部类的语法如下：

```

new SuperType(构造器参数)
{
    // 内部类方法和字段
}

```

`SuperType` 可以是接口，也可以是一个类。当它是接口时，内部类就要实现这个接口；当它是一个类时，内部类就要继承这个类。这个语法的含义是：定义一个匿名内部类，这个内部类扩展了 `SuperType`，并创建这个类的对象。

由于构造器的名字必须与类名相同，而匿名内部类没有类名，所以匿名内部类不能有构造器。上述语法中的构造器参数要传递给超类构造器。如果 `SuperType` 是一个接口，就不需要构造器参数。尽管匿名内部类不能有构造器，但可以提供一个对象初始化块。

`start()` 方法中只创建 `TimePrinter` 类的一个对象，可以使用匿名内部类，代码如下：

```
1 public void start(int interval, boolean beep)
2 {
3     var listener = new ActionListener()
4     {
5         public void actionPerformed(ActionEvent event)
6         {
7             System.out.println("At the tone, the time is "
8                 + Instant.ofEpochMilli(event.getWhen()));
9             if (beep) Toolkit.getDefaultToolkit().beep();
10        }
11    };
12    Timer timer = new Timer(interval, listener);
13    timer.start();
14 }
```

不能在静态方法中直接调用 `getClass()` 获得类信息，因为调用 `getClass()` 时调用的是 `this.getClass()`，而静态方法没有 `this`。这时应该使用如下表达式：

```
new Object(){}.getClass().getEnclosingClass()
```

`new Object(){} 会建立 Object 的匿名子类的一个匿名对象，getEnclosingClass() 得到匿名内部类的外围类，也就是包含这个静态方法的类。`

4 静态内部类

有时使用内部类只是为了把一个类隐藏在另一个类的内部，并不需要内部类有外围类对象的引用。为此，可以将内部类声明为 `static`，这样就不会生成那个引用。

下面的例子展示了静态内部类的使用。`minmax()` 方法同时计算数组中的最大值和最小值，它需要返回两个数，为此将这两个数包装在 `Pair` 类中。为了防止 `Pair` 类和其他类产生名字冲突，将它定义为另一个类的公共内部类。这时，`Pair` 类不需要任何其他对象的引用，可以将它声明为 `static`。代码如下：

```

1  public class StaticInnerClassTest
2  {
3      public static void main(String[] args)
4      {
5          double[] values = new double[20];
6          for (int i = 0; i < values.length; i++)
7          {
8              values[i] = 100 * Math.random();
9          }
10         ArrayAlg.Pair p = ArrayAlg.minmax(values);
11         System.out.println("min = " + p.getMin());
12         System.out.println("max = " + p.getMax());
13     }
14 }
15
16 class ArrayAlg
17 {
18     public static class Pair
19     {
20         private double min;
21         private double max;
22
23         public Pair(double f, double s)
24         {
25             min = f;
26             max = s;
27         }
28
29         public double getMin()
30         {
31             return min;
32         }
33
34         public double getMax()
35         {
36             return max;
37         }
38     }
39
40     public static Pair minmax(double[] values)
41     {
42         double min = Double.POSITIVE_INFINITY;
43         double max = Double.NEGATIVE_INFINITY;
44         for (double v : values)
45         {
46             if (min > v) min = v;
47             if (max < v) max = v;
48         }
49         return new Pair(min, max);
50     }
51 }

```

在这里，`Pair` 类必须声明为 `static`。因为 `Pair` 类对象是在静态方法 `minmax()` 中构造的，没有外围类对象，无法对外围类对象的引用进行初始化。

只要内部类不需要访问外围类对象，就应该使用静态内部类。

与常规内部类不同，静态内部类可以有静态字段和方法。

在接口中声明的内部类自动是 `static` 和 `public`。