即使没有 new 关键字，也能清晰表达出你的意图。Kotlin 本着最简化的设计原则，将诸如 new、行尾分号这种不必要的语法结构都取消了。

上述代码将实例化后的类赋值到了 p 这个变量上面，p 就可以称为 Person 类的一个实例，也可以称为一个对象。

下面我们开始在 main()函数中对 p 对象进行一些操作：

```
fun main() {
    val p = Person()
    p.name = "Jack"
    p.age = 19
    p.eat()
}
```

这里将 p 对象的姓名赋值为 Jack，年龄赋值为 19，然后调用它的 eat()函数，运行结果如图 2.18 所示。

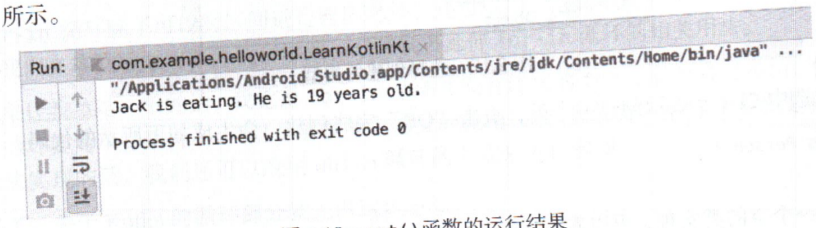

图 2.18　eat()函数的运行结果

这就是面向对象编程最基本的用法了，简单概括一下，就是要先将事物封装成具体的类，然后将事物所拥有的属性和能力分别定义成类中的字段和函数，接下来对类进行实例化，再根据具体的编程需求调用类中的字段和方法即可。

2.5.2　继承与构造函数

现在我们开始学习面向对象编程中另一个极其重要的特性——继承。继承也是基于现实场景总结出来的一个概念，其实非常好理解。比如现在我们要定义一个 Student 类，每个学生都有自己的学号和年级，因此我们可以在 Student 类中加入 sno 和 grade 字段。但同时学生也是人呀，学生也会有姓名和年龄，也需要吃饭，如果我们在 Student 类中重复定义 name、age 字段和 eat()函数的话就显得太过冗余了。这个时候就可以让 Student 类去继承 Person 类，这样 Student 就自动拥有了 Person 中的字段和函数，另外还可以定义自己独有的字段和函数。

这就是面向对象编程中继承的思想，很好理解吧？接下来我们尝试用 Kotlin 语言实现上述功能。右击 com.example.helloworld 包→New→Kotlin File/Class，在弹出的对话框中输入“Student”，并选择创建一个 Class，你可以通过上下按键快速切换创建类型。

点击“OK”完成创建，并在 Student 类中加入学号和年级这两个字段，代码如下所示：