掌握了数据类的使用技巧之后，接下来我们再来看另外一个 Kotlin 中特有的功能——单例类。

想必你一定听说过单例模式吧，这是最常用、最基础的设计模式之一，它可以用于避免创建重复的对象。比如我们希望某个类在全局最多只能拥有一个实例，这时就可以使用单例模式。当然单例模式也有很多种写法，这里就演示一种最常见的 Java 写法吧：

```
public class Singleton {
    private static Singleton instance;

    private Singleton() {}

    public synchronized static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    public void singletonTest() {
        System.out.println("singletonTest is called.");
    }
}
```

这段代码其实很好理解，首先为了禁止外部创建 Singleton 的实例，我们需要用 private 关键字将 Singleton 的构造函数私有化，然后给外部提供了一个 getInstance()静态方法用于获取 Singleton 的实例。在 getInstance()方法中，我们判断如果当前缓存的 Singleton 实例为 null，就创建一个新的实例，否则直接返回缓存的实例即可，这就是单例模式的工作机制。

而如果我们想调用单例类中的方法，也很简单，比如想调用上述的 singletonTest()方法，就可以这样写：

```
Singleton singleton = Singleton.getInstance();
singleton.singletonTest();
```

虽然 Java 中的单例实现并不复杂，但是 Kotlin 明显做得更好，它同样是将一些固定的、重复的逻辑实现隐藏了起来，只暴露给我们最简单方便的用法。

在 Kotlin 中创建一个单例类的方式极其简单，只需要将 class 关键字改成 object 关键字即可。现在我们尝试创建一个 Kotlin 版的 Singleton 单例类，右击 com.example.helloworld 包→New→Kotlin File/Class，在弹出的对话框中输入“Singleton”，创建类型选择“Object”，点击“OK”完成创建，初始代码如下所示：

```
object Singleton {
}
```

现在 Singleton 就已经是一个单例类了，我们可以直接在这个类中编写需要的函数，比如加入一个 singletonTest()函数：