表 2.2 更直观地对比了 Java 和 Kotlin 中函数可见性修饰符之间的区别。

表 2.2　Java 和 Kotlin 函数可见性修饰符对照表

修 饰 符	Java	Kotlin
public	所有类可见	所有类可见（默认）
private	当前类可见	当前类可见
protected	当前类、子类、同一包路径下的类可见	当前类、子类可见
default	同一包路径下的类可见（默认）	无
internal	无	同一模块中的类可见

2.5.4　数据类与单例类

在面向对象编程这一节，我们已经学习了很多的知识，那么在本节的最后我们再来了解几个 Kotlin 中特有的知识点，从而圆满完成本节的学习任务。

在一个规范的系统架构中，数据类通常占据着非常重要的角色，它们用于将服务器端或数据库中的数据映射到内存中，为编程逻辑提供数据模型的支持。或许你听说过 MVC、MVP、MVVM 之类的架构模式，不管是哪一种架构模式，其中的 M 指的就是数据类。

数据类通常需要重写 equals()、hashCode()、toString()这几个方法。其中，equals()方法用于判断两个数据类是否相等。hashCode()方法作为 equals()的配套方法，也需要一起重写，否则会导致 HashMap、HashSet 等 hash 相关的系统类无法正常工作。toString()方法用于提供更清晰的输入日志，否则一个数据类默认打印出来的就是一行内存地址。

这里我们新构建一个手机数据类，字段就简单一点，只有品牌和价格这两个字段。如果使用 Java 来实现这样一个数据类，代码就需要这样写：

```
public class Cellphone {
    String brand;
    double price;

    public Cellphone(String brand, double price) {
        this.brand = brand;
        this.price = price;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Cellphone) {
            Cellphone other = (Cellphone) obj;
            return other.brand.equals(brand) && other.price == price;
        }
        return false;
    }
```