

CS61B 课程笔记

黄毅男

目录

1 Java 编程基础	1
1.1 Class in Java	1
1.2 Private 与 public/Static 与 non-static	1
1.3 JUnit Testing	2
1.4 接口继承和 Override、实现继承	2
1.5 函数作为输入参数	3
1.6 Casting	3

1 Java 编程基础

1.1 Class in Java

Java 的所有 code 都必须定义在类中。类中定义的变量和函数分 static 和 non-static。static 变量/函数是抽象的,可以由类本身直接引用。non-static 变量/函数是具体的,必须先定义一个对象,然后对象才能调用 non-static 变量/函数。简单来说,当想要用类来调用某个成员时,请用 static 定义该成员;当想用具体的对象来调用某个成员时,用 non-static 定义该成员。

特别的,java 中的 main 函数也必须以 static 的方式定义在类中。

1.2 Private 与 public/Static 与 non-static

Private 的变量只能在 class 内部调用,而 public 的变量可以在 class 外部调用。

static 指的是类本身的变量,通过类来调用;non-static 必须通过对象调用。当定义类中的内部类时,若内部类为 static,则这个内部类只能通过

大类来声明、调用，故没办法访问大类中的 non-static 变量；当内部类是 non-static 时，需要通过大类的对象声明、调用。内部的 non-static 类不能有 static 变量。举个例子：

(1)Outerclass 里的 Innerclass 为 static 的。这时用 Outerclass.Innerclass 表示这个类，Innerclass 的对象声明为 new Outerclass.Innerclass(); Innerclass 内的 static 变量 x 可以用 Outerclass.Innerclass.x 表示；non-static 变量 y 用 in=new Outerclass.Innerclass(), in.y 表示。

(2)Outerclass 里的 Innerclass 为 non-static 的。这时还是用 Outerclass.Innerclass 表示这个类，Innerclass 的实例声明为 out.new Innerclass, out 为 Outerclass 的实例。Innerclass 内不能有 static 变量 x；Innerclass 内的 non-static 变量 y 用 out.in.y 表示。

简单来说，static 变量的前一级是类；non-static 变量的前一级是对象。

1.3 JUnit Testing

对于类中的每个方法，都可以单独写一个 test 函数。这个 test 函数用 non-static 定义，然后在函数前加上 @org.junit.Test，可以直接运行 test 函数。同时不同的 test 的独立进行。

1.4 接口继承和 Override、实现继承

接口是一个抽象的东西，它约定了类的方法有哪些。若某个类继承了这个接口，这个类必须包含接口中声明的所有方法（并且必须是 public 的），这些方法的声明称为对接口中方法的重写 (Override)。在接口中这些方法没有被实现，接口继承后的类可以用任意实现这些方法。继承了接口的类还可以有自己的属性和其他方法。实例化时，可以用接口类型来指向一个继承了接口的子类的对象，但是这个对象只能调用接口中的方法（即 override 的方法），不能调用子类中的属性和其他没有被重写的方法。

在接口方法前加上 default 关键词，可以给方法一个实现。这样类在继承这个接口时会“实现继承”，即把这个方法的实现也继承过去。实现继承的方法不需要声明就可以直接调用。当然在类中也可以重写这个 default 的方法 (Override)。

实例化时，可以用接口类型来指向一个继承了接口的子类的对象，但是这个对象只能调用接口中的方法（即 override 的方法和 default 的方法），但不能调用子类中的属性和其他接口中没有的方法。虽然是以接口类型声明，

但实例的方法实现均以类中 `override` 后的实现为准。这是因为声明接口类型是一个静态类型，而创建一个类的对象时，这个类是动态类型。编译时系统以静态类型为准，比如输入参数是否与静态类型中方法的参数匹配等。运行时，一个引用类型只能调用静态类型的方法，除非动态类型将这个方法重写了。如果静态类型中没有这个方法，即使动态类型里有，也不能调用。如果有多个 `overload` 的方法可以处理一个输入，系统会选择最 `specific` 的那个。

一个 `override` 和 `overload` 的例子。假设接口中有函数 `default func`。如果类继承了接口，并且这个方法与 `func` 一样 (输入返回参数格式也必须一样)，这时候类就可以重写 (`override`) 这个方法的实现；但是如果类中的方法格式与 `func` 同名但是输入返回参数格式不同，这时候类其实是实现继承了一个 `default func` (尽管在类中没有声明)，同时重载了这个方法 (`overload`)。简单来说，`override` 是修改方法的实现，但是 `overload` 是修改整个方法 (但是方法的名字相同)。

1.5 函数作为输入参数

java 不能直接把函数作为输入参数，因为函数不是一个数据类型。一般的方法是，首先定义一个接口 (比如叫做 `func`)，所有的函数都继承这个接口中 `apply` 方法 (即调用这个函数本身)。这样在调用函数时，我们可以先实例化这个函数，得到一个 `func` 对象，然后定义一个以 `func` 对象为输入参数的函数，这样就可以达到“构造一个以函数为输入参数的函数”的目的了。

1.6 Casting

Casting 是 java 中变换静态类型的一种方法。我们知道，一个赋值操作 `a=b` 是否能编译，取决于 `a` 的类型 `A` 是否是 `b` 的类型 `B` 的上义词。若 `B` 是 `A` 的下义词，编译会失败。这时我们可以用 `a=(A) b` 的 casting 方法来临时将后边的表达式的静态类型变为 `A` “，骗过”编译器从而编译成功。不过需要注意的是，即使通过了编译器，运行也可能报错。`cast` 赋值后，对象的静态类型比动态类型小，这时候 run-time 会报错。比如 `(Small) new Big(...)`，本来表达式 `new Big(...)` 的静态类型和动态类型都是上义词 `Big`，但是其静态类型被 Casting 为 `Small`，这时候静态类型是动态类型的下义词，那么这个表达式虽然可以通过编译，但是在运行时会报错。总而言之，Casting 可

以任意提高静态类型，但是在降低静态类型时，务必不能低于动态类型的大小，否则会运行报错。

总结：

- 编译时：检查赋值语句，被赋值方的静态类型必须是赋值方静态类型的上义词；检查方法调用语句，静态类型中必须有这个方法。
- 运行时：运行赋值语句时，每个对象的静态类型必须是其动态类型的上义词；运行方法调用语句时，从静态类型中选取最合适的方法调用，但如果这个方法被其动态类型 override(只有下义词才能 override，这就是为什么动态类型必须是静态类型的下义词)，那么调用动态类型中的方法。