CS61B 课程笔记

黄毅男

目录

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.Interclass 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，并且这个方法的格式与 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。

1.7 泛型

类的定义中可以带上泛型：public class xxx<any>，这里 any 就是一个泛型，代指某种引用类型。实际实例化时，需要将 any 用某个引用类型填入。

方法的定义中也可以带上泛型，形式为 <any> any xxx()。实际使用时，不需要填入 <any>，系统会自动根据对应关系确定 any。

1.8 异常处理

Java 中异常也是类，分为 checked 和 unchecked。checked 的异常，比如 IO 异常，必须被用 try& catch 方法 catch，或者在可能出现 checked 异常的类后加上 throws 声明。checked 的异常如果不处理，则程序无法编译。unchecked 的异常，比如运行异常，不会影响程序编译，但是运行到异常处程序会停止。

1.9 Package

定义一个 package 文件夹，里面的文件都在首行加上文件夹的 path。这样是为了避免类的重名。引用类时，需要 import 这个 package。不加 package 声明的文件都属于一个 default package 的 package。

1.10 Access Control

一个 field 变量或者 method(以下统称变量)，有几种 access 等级：public、protected、private 以及默认的 package private。public 的变量是全局公开，可以被任意访问；protected 的变量可以在类内部、同一个 package 里、其子类访问，除此之外的访问是不允许的。private 只能在 class 内访问。默认的 package private 可以被类内部或者在同一 package 下访问。

注意对于接口，其默认的是 public 而不是 package private。

对于类本身，也有 access 的问题。但是 private 和 protected 的 modifier 是无效的，public 和默认的 package private 是可以的。

2 数据结构与算法

2.1 并查集

并查集指的是一种特殊的数据结构：只记录两个数据节点是否相交 (相邻)。具有的方法有：判断两个节点是否相交、将两个节点连接。

下面介绍几种实现并查集的方法，并分析其复杂度。

2.1.1 集合实现并查集

用自带的集合 (set) 数据结构来实现并查集。用并集操作实现节点连接功能，复杂度为 O(N)；用循环集合查找的方法来判断两个节点是否相交，复杂度为 O(N)。

2.1.2 数组实现并查集

首先将节点的数据映射到整数。构造一个数组，数组的 index 代表 code 为 index 的节点，而数组的 value 代表节点与之相交的节点。

判断节点是否相交，只需要判断两个节点对应的数组位置的值是否相同，复杂度为 O(1)；连接节点，需要将一方所有的相交集合在数组中的值改变，复杂为 O(N)。

2.1.3 树实现并查集

考虑相交的节点都在一个树中。用一个数组表示这多个树 (多个相交集合)。index 代表节点，value 代表树中节点的 parent 的 index。

判断是否连接时，需要向上爬树找到 root index，然后判断两个节点的 root index 是否相同，复杂度为 O(logN)，前提是树是比较均匀；连接两个节点时，总是用树的深度小的一方连向树的深度大的一方的 root 处 (树的深度信息可以记录在数组 root index 的位置)。所以复杂度由爬树时的深度决定，为 O(logN)。

这种实现方式是复杂度最小的，也是最常用的并查集实现方法。

2.2 树

2.2.1 树的遍历

树可以看成图的特例，分别深度优先遍历和广度优先遍历。深度优先遍历分为前序、中序、后序三种遍历方法，而广度优先遍历可以通过队列实现。

由于树具有递归结构，深度优先遍历可以通过递归简单的实现。广度优先遍历实现如下：最开始先让根节点入队，之后一直持续下述步骤：将队列头的节点的 children 全部入队，打印 (或者其他操作) 这个队列头节点，然后队列头出队。最后就实现了树的广度优先遍历。

2.2.2 二叉搜索树

二叉树指的就是子节点不超过两个的树结构。而二叉搜索树是指左节点的值小于根，右节点的值大于根的二叉树。显然，如果该二叉搜索树是比较平衡的，那么对于有 N 个总节点的二叉搜索树，树的深度为 log(N+1)，其插入、搜索时间复杂度为 log(N)。

二叉搜索树需要通过旋转等方法保持平衡。

2.2.3 B 树：2-3 树、2-3-4 树

先介绍 2-3 树。2-3 树是二叉搜索树的推广，考虑一个节点可以存放两个数据点，树有三个子节点，三个子节点的数据范围分别小于根的第一个数据点、大于第一个数据点但小于第二个数据点、大于第三个数据点。当一个节点存了超过两个数据时，将中间偏左 (这个情况下就是第二个) 的数据往父节点移动，然后重排子树。

这个意义下，二叉树其实就是 2-3-4-...-n 树在 n=2 时的特例。对于一个一般的 2-3-...-n 树，每个节点可以有至多 n-1 个数据，并且有 n 个子节点。

我们一般称这种树为 B-Tree。

2.2.4 红黑树

红黑树指的是一种将 B 树等价于一个二叉搜索树的方法。即对储存了超过一个以上数据的节点，将其数据单独分开到不同的节点并用红色线连接。红线连接的其实是在 B 树里同一个节点，而黑线连接的才是 B 树中不

同节点。用这种方法把 B 树变成的二叉搜索树，即红黑树，可以保证其深度不超过 B 树的两倍，这样搜索和插入的速度就会提升。

2.3 哈希表

哈希表是指一个映射，是将任意对象映射到一个整数上的函数。如果适当选择这个函数，可以把输出的整数 (哈希码) 作为存储对象的 index，这样在插入、读取时的时间复杂度就为 O(N)。注意两个相同的对象的哈希码必须一样。

如果要将对象映射到某个整数范围之内，可以用自带的 hashcode 方法取余数实现。注意所有对象都自带 hashCode() 和 equals() 方法，hashCode() 根据对象的地址进行映射，equals 判断对象的地址是否相同。如果重写了 equals() 方法，请务必重写 hasCode() 方法 (反之亦然)，否则两个 equals 的对象具有不同的 code，会查找出错。

2.4 堆和优先队列

优先队列：只能读取、删除最小的元素。优先队列可以用堆实现。

堆这里指最小堆，是一种特殊的二叉树结构。与二叉搜索树不同，其结构为：子节点的数值大于或者等于根节点的数值。堆必须是一个完全二叉树。堆的插入：先在完全二叉树的顺序尾端位置插入子节点，然后持续判断这个子节点需不需要“上浮”，即跟父节点交换位置，直到满足最小堆条件。堆读取最小元素：直接读取根节点即可。堆移除最小元素：移除根节点后，用完全二叉树顺序尾端位置的节点顶替根节点，然后持续判断是否需要“下浮”，直到满足最小堆条件。

对于完全二叉树，可以采用数组来存储，这也是完全二叉树除了搜索速度外的另一优势。

2.5 图

图是由节点 (称为顶点) 和边组成的数据结构。图需要解决的问题有：两个顶点是否连接，以什么样的路径连接？

图和树有些类似，但是不同的是，图可以形成闭环路径，所以树的一些遍历方法直接用在图中会导致死循环。为了防止重复访问已访问的顶点导致死循环，我们给图一个数组 marked[]，已访问的顶点设为 True，不再访问

True 的节点。另外为了记录路径，再赋予图一个 edgeTo[] 数组，edgeTo[i] = j 代表从 j 到 i 这样一个行动路径。

2.5.1 广度优先遍历

图的广度优先遍历与树的优先遍历类似：有一个队列，队列头的节点 v 出队，然后将出队节点的相邻且未被 marked 的节点 n 入队，并 mark 这些相邻节点 n，最后设 edgeTo[n]=v。循环直到队列为空。

2.5.2 深度优先遍历：递归实现

与树的深度优先遍历类似，但是注意要 mark 已经访问的节点。对于需要深度遍历的顶点，先 mark 这个节点，然后依次深度遍历与它相邻且未被 marked 的节点。

2.5.3 深度优先遍历：循环实现

读者可能可以想到，利用队列进出两头的特性，即先入队的才先出队的特点，可以实现广度优先遍历 (先把相邻节点全部入队，这样总能保证在访问下一层之前把这一层的节点全部访问完)。相反，为了实现深度优先遍历，就必须利用到栈的先进先出的特性：这样保证了在访问同一层节点之前，可以把后进栈的深层节点访问完。

具体实现如下：最开始出发节点入栈。然后循环：将栈顶节点弹栈，访问弹栈节点并 mark，然后将该节点的所有相邻且没有 mark 的节点入栈，持续这个过程直到栈空为止。

2.5.4 图的实现

上面讲了这么多，那么图应该用什么样的数据结构实现呢？邻接矩阵 (M[i,j] 表示 i 是否连接到 j) 是一种常用的方法。但是对于一个稀疏的图，邻接矩阵是个稀疏矩阵，浪费了大量的存储空间和搜索时间。更常用的方法还是邻接列表，即 list[i] 指向一串链表，表示与 i 顶点相邻的所有顶点的指标。

判断两个节点 (比如 i 是否连接到 j) 是否相邻：需要在 list[i] 中查找是否有 j，时间复杂度取决于 i 的度 (有多少连接节点)。连接两个节点 i 到 j，将 j 放到 list[i] 的链表头，复杂度为 O(1)；打印整个图的连接关系，复杂度为 O(V+E)，V 为总顶点数，E 为总边数。

2.5.5 拓扑排序

拓扑排序：将有向图的前后顺序看成是一种先后条件，拓扑排序将有向图整理成一个线性结构并输出。

方法：用深度优先搜索，记录搜索顺序，遍历整个图。最后将搜索顺序倒置即得到拓扑排序。