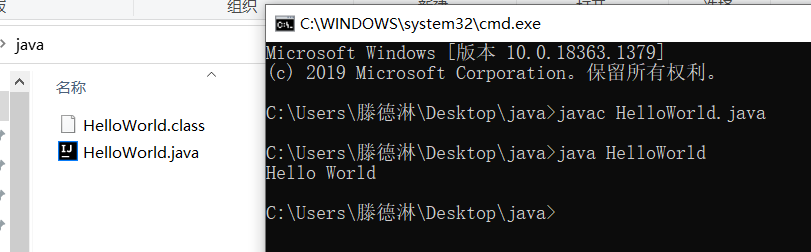
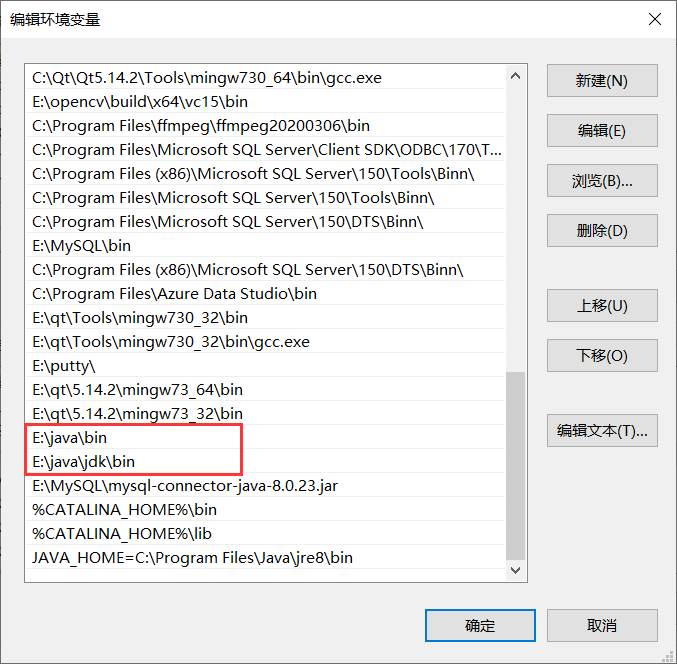
**【实习一】**

1. **安装使用JDK，学习使用Java命令行调试**
2. 下载jdk
3. dos编译和path配置





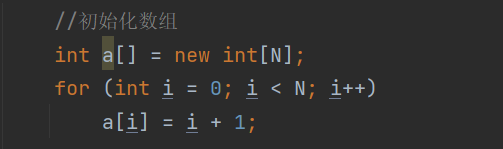


通过命令行调试程序，了解了程序运行的过程，需要先编译，再链接；同时知道jdk包的使用方式；如何配置java运行环境；

1. **求解约瑟夫环**
2. 数组方法

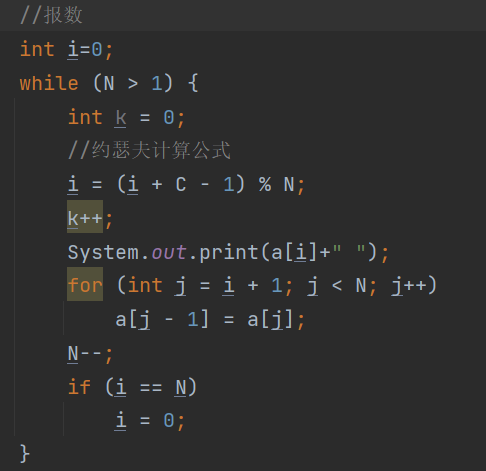
分为三步：

第一步：建立循环链表



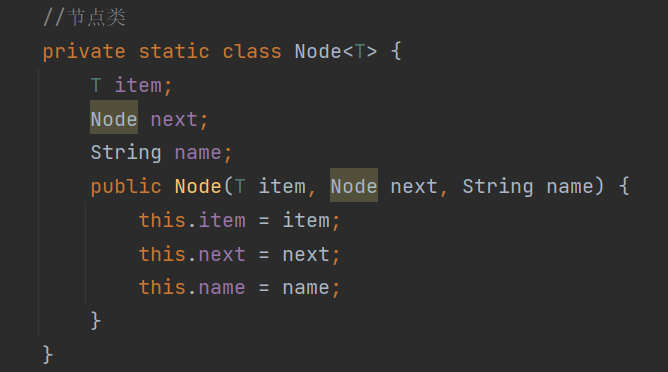
第二步：模拟报数，创建变量count记录；

第三步：执行报数，使用数组进行模拟约瑟夫问题，采取的方式是用约瑟夫计算公式计算要删掉的下标是多少，则删掉该下标的数据，通过往前循环的方式，覆盖值；同时，当值等于输入的总人数时，需要重新将下标赋值为0，这样就实现了循环的报数；



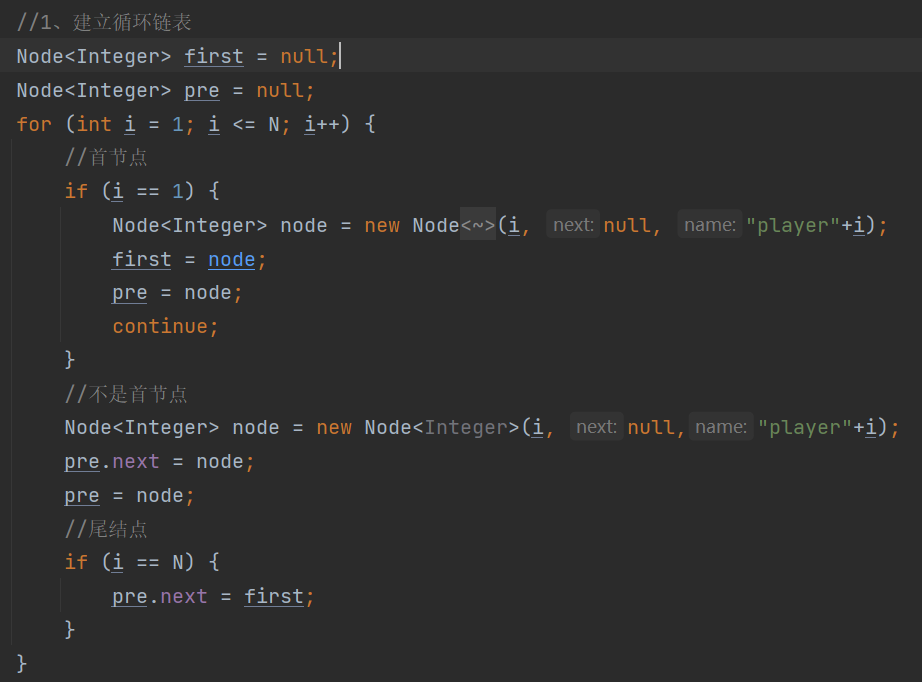
1. 链表方法

创建节点Node类：成员变量有序号、姓名、next；



1. 建立循环链表

创建first节点和pre节点用来初始化链表；分为三种情况：首节点、中间节点、尾结点，构建方法和单链表构建方法类似，只不过当构建尾结点的时候，需要将next指向first；

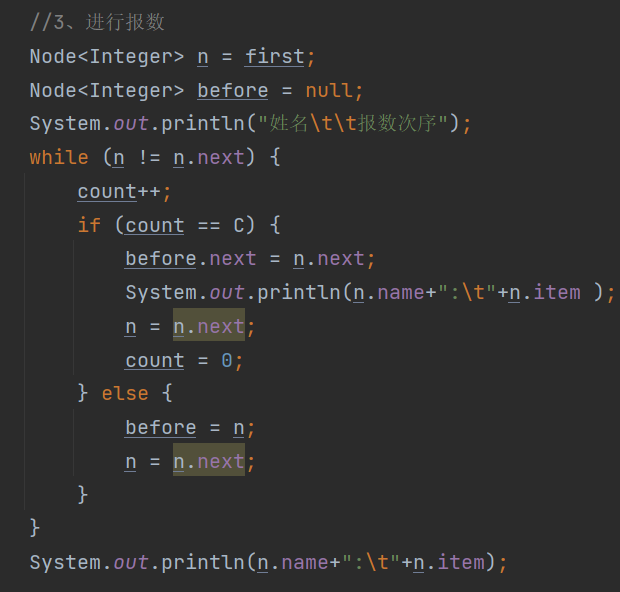


1. 模拟报数

设置count变量模拟；

1. 进行报数

当本节点的值等于要报的数的时候，就删掉该节点；通过before.next指向下一个节点完成；当只剩下一个节点的时候就退出报数；



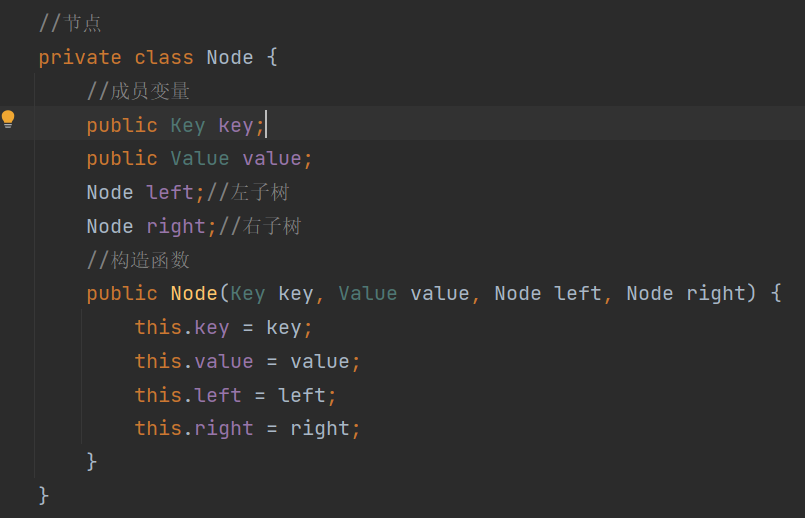
通过约瑟夫环的学习，我回顾了循环链表的编写方法，对循环链表的使用更加熟练；

1. 二分搜索树

二分搜索树的特点就是一个节点的左子树的全部数据都比自己小，右子树比自己都大。所以在构建的时候就需要根据这种特定的方法进行构建；

1. 节点结构

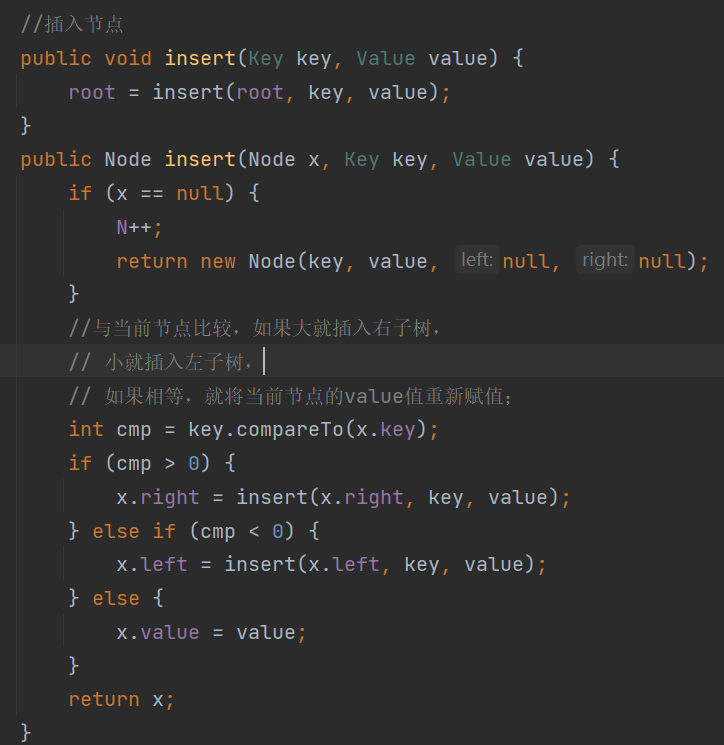
成员变量有key节点序号、value节点的值、left左子树节点、right右子树节点；



1. Insert()函数

分为三种情况，插入的序号比当前节点的值小，则向左子树插入；若大，则向右子树插入；如果相等，则将插入的value值赋值给当前节点的value值；

向左右子树插入采用递归的方式实现，同时每插入一个节点，需要将N++;



1. Visit()函数

输入key值，返回value值；分为两种情况，没有找到key值，则返回null；如果找到了key值则返回当前节点的value值；

同样是使用递归分三种情况向下查找；



1. Delete()函数

输入key值，则删除该节点；

删除节点，首先需要找到该节点，如果要删除，则要考虑删除之后如何将树节点连接才能满足二叉搜索树的特点；应该要将删除节点的右子树中的最小值节点作为子树的根节点，也就是右子树的左子树的最下面的那个点；

整个过程应该是先通过递归向下找到要删除的节点，然后分为三种情况，该节点没有left，则将该节点的right返回作为连接的节点；该节点没有right，则返回该节点的left作为连接的节点；如果左右节点都有，那么需要先找到右节点中最小的节点，在将最小的节点与其父节点断开，然后再将最小节点的left和right分别等于需要删除的节点的left和right，最后再将最小的节点赋值给需要删除的节点，则完成了删除一个节点操作；

1. *//删除节点*
2. public void delete(Key key) {
3. delete(root, key);
4. }
5. public Node delete(Node x, Key key) {
6. *//没找到*
7. if (x == null) {
8. return null;
9. }
10. *//向下递归找*
11. int cmp = key.compareTo(x.key);
12. if (cmp > 0) {
13. x.right = delete(x.right, key);
14. } else if (cmp < 0) {
15. x.left = delete(x.left, key);
16. } else {*//找到了之后，进行删除操作*
17. N--;
18. *//删除节点的右子树为空*
19. if (x.right == null) {
20. return x.left;
21. }
22. *//删除节点的左子树为空*
23. if (x.left == null) {
24. return x.right;
25. }
26. *//找到最小节点*
27. Node minNode = x.right;
28. while (minNode.left != null) {
29. minNode = minNode.left;
30. }
31. *//断开有子树中最小节点与上一个节点的连接*
32. Node n = x.right;
33. while (n.left != null) {
34. if (n.left.left == null) {
35. n.left = null;
36. } else {
37. n = n.left;
38. }
39. }
40. *//将右子树中的最小节点作为当前节点，表示删除了当前节点*
41. minNode.left = x.left;
42. minNode.right = x.right;
43. x = minNode;
44. }
45. return x;
46. }
47. 测试案例

测试代码：

1. public static void main(String[] args) {
2. searchTree<Integer,String> s = new searchTree<>();
3. System.out.println("1、测试insert():");
4. s.insert(1,"张三");
5. s.insert(2,"李四");
6. s.insert(3,"王五");
7. s.insert(4,"赵六");
8. s.insert(5,"刘七");
9. System.out.println("插入了"+s.size()+"个数据！");
10. System.out.println("2、测试visit():");
11. System.out.println("key为2的名字叫做："+s.visit(2));
12. System.out.println("3、测试delete():");
13. System.out.println("删除key为2的数据");
14. s.delete(2);
15. System.out.println("删除之后树的所有数据为：");
16. for (int i=1;i<=s.size();i++){
17. System.out.println(s.visit(i));
18. }
19. }

输出结果：

