**数据结构课程设计**

**题 目: 家谱管理**

**班 级：**

**学 生 姓 名：**

**学 生 学 号：**

**2022 年 3月 23日**

目录

[1. 开发环境 3](#_Toc477208932)

[2. 程序的功能 3](#_Toc477208933)

[3. 代码清单 3](#_Toc477208934)

[4. 运行界面 14](#_Toc477208935)

## 开发环境

## 功能描述（内容仅供参考）

用Java实现了一个普通的二叉树，有左右孩子和父节点，并添加查找和前序遍历的功能，最后通过一个测试类验证这些功能。

1. 创建家谱：Tree类代表家谱，有一个父节点，接下来所有的操作都依赖于这个父节点

2. 添加家谱成员：根据成员名称（String字符串）查找到某个节点后返回其引用，然后用这个引用，手动创建左右孩子并添加

3. 删除成员：输入该成员的名字，找到其引用即可删除该成员的子孙。（由于Java按值传递，所以直接删除节点本身有点麻烦）

4. 遍历成员：前序遍历和后序遍历两种方式，打印整棵树

5. 查找成员：输入成员的编号后可以查询该成员，返回这个节点的引用。

## 代码清单

树节点类：

|  |
| --- |
| TreeNode.java |
| package com.example.sjjg4;  public class TreeNode {  private String content;  private TreeNode leftChild,RightChild,parent;   public String getContent() {  return content;  }   public void setContent(String content) {  this.content = content;  }   public TreeNode getLeftChild() {  return leftChild;  }   public void setLeftChild(TreeNode leftChild) {  this.leftChild = leftChild;  }   public TreeNode getRightChild() {  return RightChild;  }   public void setRightChild(TreeNode rightChild) {  RightChild = rightChild;  }   public TreeNode getParent() {  return parent;  }   public void setParent(TreeNode parent) {  this.parent = parent;  } } |

树类（封装节点，提供查询删除遍历功能）

|  |
| --- |
| Tree.java |
| package com.example.sjjg4;  public class Tree {  private TreeNode root;   public Tree(TreeNode node)  {  root=node;  }   public Tree(){}   public Tree(String content)  {  root=new TreeNode();  root.setContent(content);  }   private TreeNode find(TreeNode root,String s)  {  if(root==null)  {  return null;  }  if(root.getContent().equals(s))  {  return root;*//找到了* }  TreeNode left=find(root.getLeftChild(),s);  TreeNode right=find(root.getRightChild(),s);   if(left!=null){return left;}  else if(right!=null){return right;}  else{return null;}  }   public TreeNode findNodeByContent(String s)  {  return find(this.root,s);  }   private void \_qxbl(TreeNode root)  {  if(root!=null)  {  System.*out*.println(root.getContent());  \_qxbl(root.getLeftChild());  \_qxbl(root.getRightChild());  }  }   private void \_hxbl(TreeNode root)  {  if(root!=null)  {  \_hxbl(root.getLeftChild());  \_hxbl(root.getRightChild());  System.*out*.println(root.getContent());  }  }    public void qxbl()  {  \_qxbl(this.root);  }   public void hxbl()  {  \_hxbl(this.root);  }   private void \_destroy(TreeNode root)  {  if(root!=null)  {  \_destroy(root.getLeftChild());  \_destroy(root.getRightChild());  root.setLeftChild(null);  root.setRightChild(null);  root=null;  }  }   public void removeFollowingNodes(TreeNode node)  {  \_destroy(node);  } } |

测试类

|  |
| --- |
| TreeTest.java |
| package com.example.sjjg4;  import org.junit.jupiter.api.Test;  public class TreeTest {  private Tree tree=new Tree("张三");    @Test  public void test()  {  var node=tree.findNodeByContent("张三");  if(node!=null)  {  *//插入两个子节点* TreeNode left=new TreeNode();  left.setContent("李四");  TreeNode right=new TreeNode();  right.setContent("王五");  *//插入两个节点* node.setLeftChild(left);  left.setParent(node);  node.setRightChild(right);  right.setParent(node);   }  *//先多插入几个节点，便于测试* var node0=tree.findNodeByContent("李四");  if(node0!=null)  {  *//李四下面插入两个节点* var left=new TreeNode();  left.setContent("东方不败");  node0.setLeftChild(left);  left.setParent(node0);   var right=new TreeNode();  right.setContent("岳不群");  node0.setRightChild(right);  right.setParent(node0);  }  var node2=tree.findNodeByContent("王五");  if(node2!=null)  {  var right=new TreeNode();  right.setContent("令狐冲");  node2.setRightChild(right);  right.setParent(node2);  System.*out*.println("前序遍历：==================================");  tree.qxbl();  System.*out*.println("后序遍历：==================================");  tree.hxbl();  }   *//删除一个分支看看* var dep=tree.findNodeByContent("李四");  tree.removeFollowingNodes(dep);  var x=tree.findNodeByContent("东方不败");  if(x==null){  System.*out*.println("没了！");   *//由于Java只能传值，因此dep本身不会被删除，但是其子节点都没了  //现在已经找不到东方不败和岳不群了* }   System.*out*.println("前序遍历：==================================");  tree.qxbl();  } } |

## 运行界面（通过图例展示操作流程）

为了方便测试，我们把输入内容写入测试代码，不需要手动输入

运行测试类，得到如下输出

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 应用程序, Word

描述已自动生成

完整的输出如下：

|  |
| --- |
| Console |
| "C:\Program Files\Java\jdk-16.0.1\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.1.3\lib\idea\_rt.jar=65185:C:\Program Files\JetBrains\IntelliJ IDEA 2021.1.3\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\JetBrains\IntelliJ IDEA 2021.1.3\lib\idea\_rt.jar;C:\Users\Administrator\.m2\repository\org\junit\platform\junit-platform-launcher\1.7.0\junit-platform-launcher-1.7.0.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2021.1.3\plugins\junit\lib\junit5-rt.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2021.1.3\plugins\junit\lib\junit-rt.jar;C:\Users\Administrator\IdeaProjects\实验4\out\production\实验4;C:\Users\Administrator\.m2\repository\org\junit\jupiter\junit-jupiter\5.7.0\junit-jupiter-5.7.0.jar;C:\Users\Administrator\.m2\repository\org\junit\jupiter\junit-jupiter-api\5.7.0\junit-jupiter-api-5.7.0.jar;C:\Users\Administrator\.m2\repository\org\apiguardian\apiguardian-api\1.1.0\apiguardian-api-1.1.0.jar;C:\Users\Administrator\.m2\repository\org\opentest4j\opentest4j\1.2.0\opentest4j-1.2.0.jar;C:\Users\Administrator\.m2\repository\org\junit\platform\junit-platform-commons\1.7.0\junit-platform-commons-1.7.0.jar;C:\Users\Administrator\.m2\repository\org\junit\jupiter\junit-jupiter-params\5.7.0\junit-jupiter-params-5.7.0.jar;C:\Users\Administrator\.m2\repository\org\junit\jupiter\junit-jupiter-engine\5.7.0\junit-jupiter-engine-5.7.0.jar;C:\Users\Administrator\.m2\repository\org\junit\platform\junit-platform-engine\1.7.0\junit-platform-engine-1.7.0.jar" com.intellij.rt.junit.JUnitStarter -ideVersion5 -junit5 com.example.sjjg4.TreeTest,test  前序遍历：==================================  张三  李四  东方不败  岳不群  王五  令狐冲  后序遍历：==================================  东方不败  岳不群  李四  令狐冲  王五  张三  没了！  前序遍历：==================================  张三  李四  王五  令狐冲  进程已结束，退出代码为 0 |