2018/1/17 树的预备功能与打包

上一章里大概实现了二叉树的管理和遍历，这只是一个小实验，有很多不完全的地方，而且说明的有点仓促，在这里也打算作个小总结。

TreeNode：基本的节点，有id，text和parentid（用不着）

ManyTreeNode：多叉树节点，内部包含TreeNode来存储基本信息，有父节点和子节点集合来联系各个节点。

ManyNodeTree：多叉树，用于管理各个节点，可以插入节点，遍历节点（之前的代码里是输出id，其实应该输出text）。(还没做删除节点和改变节点内部信息，应该做起来也不是很难)

刚才我回顾了昨天的代码，感觉在ManyNodeTree类里有一些还没解释到位的几点：

类成员floor指的是树的tmp（当前结点）的深度，为了到时候能让代码的层次感更强。

函数iteratorTree里的stringBuilder比string更快，具体用法和string差不多，差别可以上网查

在这一章里，我打算把上一章的树嵌入Drone（第一章的编程代码）里面，并且做个全部代码的界面显示。为了加大效率，我打算一边打代码一边把思路写在这里，但是截图的话得看今天有没有时间了。

首先我不打算写TreeNode，而是把它挑选有用的信息，直接并入ManyTreeNode这个类中。所以先创建的是ManyTreeNode这个java class。

想到要有个Button显示所有产生的代码的，所以初始化了一个叫to\_code\_screen的button

以及加了一个用来测试的text，之后会变成另一个页面专门显示所有产生的代码

上一章中ManyNodeTree的beParent（）函数中的floor应该是减一的，由于floor没怎么用，所以没有注意，打成了加一。

以下是适应图形化界面的一棵树

ManyTreeNode:

**import** java.util.ArrayList;  
**import** java.util.List;  
  
*/\*\*  
 \* Created by 1535725170 on 2018/1/19.  
 \*/***public class** ManyTreeNode {  
  
 **private** String **text**;  
  
 **private** ManyTreeNode **parentNode**;  
  
 **private** List<ManyTreeNode> **childList**;  
  
 **private int floor**;  
  
 **public** ManyTreeNode(String text){  
 **this**.**text**=text;  
 **this**.**childList**= **new** ArrayList<ManyTreeNode>();  
 }  
  
 **public void** setText(String text){  
 **this**.**text**=text;  
 }  
  
 **public** String getText(){  
 **return text**;  
 }  
  
 **public void** setParentNode(ManyTreeNode parentNode){  
 **this**.**parentNode**=parentNode;  
 }  
  
 **public** ManyTreeNode getParentNode(){  
 **return parentNode**;  
 }  
  
 **public** List<ManyTreeNode> getChildList(){  
 **return childList**;  
 }  
  
 **public void** addChild(ManyTreeNode child){  
 **this**.**childList**.add(child);  
 child.setParentNode(**this**);  
 child.setFloor(**this**.getFloor()+1);  
 }  
  
 **public void** setFloor(**int** floor){  
 **this**.**floor**=floor;  
 }  
  
 **public int** getFloor(){  
 **return floor**;  
 }  
  
}

ManyNodeTree:

**public class** ManyNodeTree {  
 **private** ManyTreeNode **root**;  
  
 **private** ManyTreeNode **tmp**;  
  
 **private int floor**;  
  
 **public** ManyNodeTree(){  
 **root** = **new** ManyTreeNode(**"root"**);  
 **tmp** = **root**;  
 **floor**=0;  
 }  
  
 **public** ManyTreeNode getRoot(){  
 **return this**.**root**;  
 }  
  
 **public void** addChild(ManyTreeNode manyTreeNode){  
 **this**.**tmp**.addChild(manyTreeNode);  
 }  
  
 **public void** deleteChild(**int** n){  
 **this**.**tmp**.getChildList().remove(n);  
 }  
  
 **public void** beChild(**int** n){  
 **if**(**tmp**.getChildList()!=**null** && **tmp**.getChildList().size()>n){  
 ManyTreeNode newNode = **this**.**tmp**.getChildList().get(n);  
 **tmp** = newNode;  
 **floor** = **floor** + 1;  
 }  
 }  
  
 **public void** beParent(){  
 **if**(**tmp**.getText()!=**"root"**){  
 ManyTreeNode newNode = **tmp**.getParentNode();  
 **tmp** = newNode;  
 **floor** = **floor** - 1;  
 }  
 }  
  
 **public** String iteratorTree(ManyTreeNode manyTreeNode){  
 StringBuilder buffer = **new** StringBuilder();  
 buffer.append(**"\n"**);  
 **if**(manyTreeNode!=**null**){  
 **for**(ManyTreeNode index : manyTreeNode.getChildList()){  
 buffer.append(index.getText()+**","**);  
 **if**(index.getChildList()!=**null** && index.getChildList().size()>0){  
 buffer.append(iteratorTree(index));  
 }  
 }  
 }  
 buffer.append(**"\n"**);  
 **return** buffer.toString();  
 }  
  
}

打包的内容在于多叉树内部成员tmp的使用，因为有tmp，我们图形化界面的实现可以和内部数据结构分离，也就是说，原来一个节点的内部有一个按钮，按钮相当于这个节点的肢体，而我们有了tmp之后，多叉树相当于按钮们通过tmp的一个映射，可能现在说的有些抽象，等明后天实现了图形化编程之后，我们就可以看到做到分离图形和内部结构也不是很难，并且实际上比把这两个放在同一个类里面更容易。

其实今天我的工作量不止这些，只是有些图形化界面的编程没有成功，比如scollView和listView的结合（就是多出来的那几个文件，不用管它们的），作为入门的新手我直接看代码有点急于求成了，最后还是选择找了资源，明天看！