GTU Department of Computer Engineering

CSE 222/505 - Spring 2023

Homework 5 Report

BERRU LAFCI

1901042681

Hello teacher,

I just did A and B part of homework. I'm going to an Erasmus internship this summer and I went to a visa appointment this week. I was trying to organize my documents and collect signatures so it was a very stressful week for me. That's why I was able to do this much. In addition, since the midterms have just ended, I had to handle my visa procedures quickly until 3rd May which was my appointment day.

Best regards

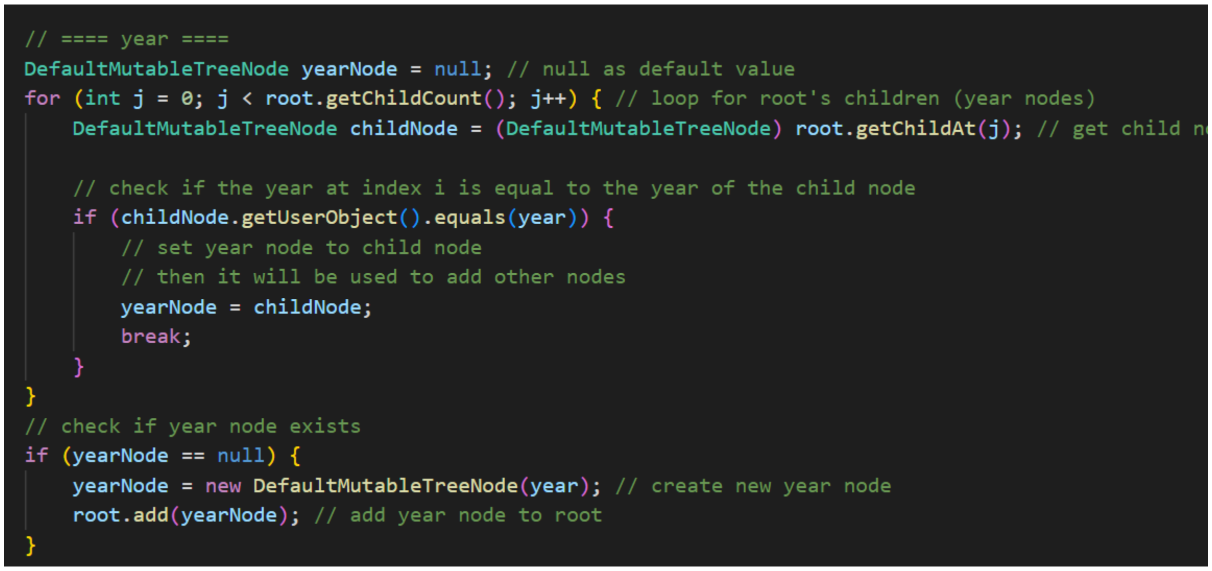
==

PART A:

I created the tree with createTree() function. I tokenized the input with semicolon. Then, I put them into ArrayList. **As an example, I did year, course, lecture and problem with same logic:**

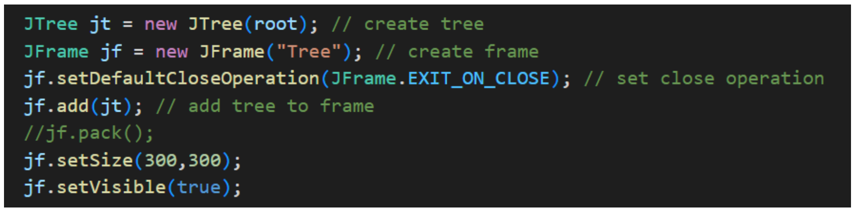
If there is yearNodes, it will traverse them. If there is a same yearNode, it will be marked to use in other child nodes for linking.

If there is no yearNode or no same yearNode, it is going to create a new one and link it with its parent node (root).



```java
// ==== year ====
DefaultMutableTreeNode yearNode = null; // null as default value
for (int j = 0; j < root.getChildCount(); j++) { // loop for root's children (year nodes)
    DefaultMutableTreeNode childNode = (DefaultMutableTreeNode) root.getChildAt(j); // get child n

    // check if the year at index i is equal to the year of the child node
    if (childNode.getUserObject().equals(year)) {
        // set year node to child node
        // then it will be used to add other nodes
        yearNode = childNode;
        break;
    }
}
// check if year node exists
if (yearNode == null) {
    yearNode = new DefaultMutableTreeNode(year); // create new year node
    root.add(yearNode); // add year node to root
}
```

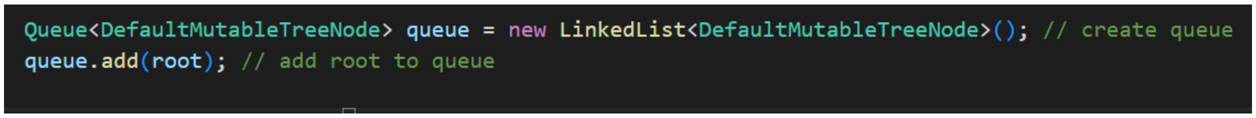
And here I created tree and the frame:



```
JTree jt = new JTree(root); // create tree
JFrame jf = new JFrame("Tree"); // create frame
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // set close operation
jf.add(jt); // add tree to frame
//jf.pack();
jf.setSize(300,300);
jf.setVisible(true);
```

PART B:

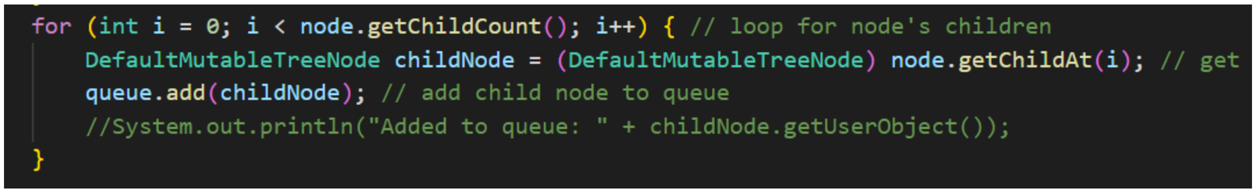
The method creates a queue of DefaultMutableTreeNode objects using a LinkedList. It then adds the root node to the queue.



```
Queue<DefaultMutableTreeNode> queue = new LinkedList<DefaultMutableTreeNode>(); // create queue
queue.add(root); // add root to queue
```

Then enters a loop that continues until the queue is empty. In each iteration of the loop, it removes the first element from the queue and checks whether its user object value is equal to the value being searched for. If it is, the method prints a message indicating that the value has been found and returns from the method.

If the value is not found in the current node, the method loops through the node's children and adds each child to the end of the queue. This ensures that the algorithm processes all nodes at a given depth (level) before moving on to the next level of the tree.

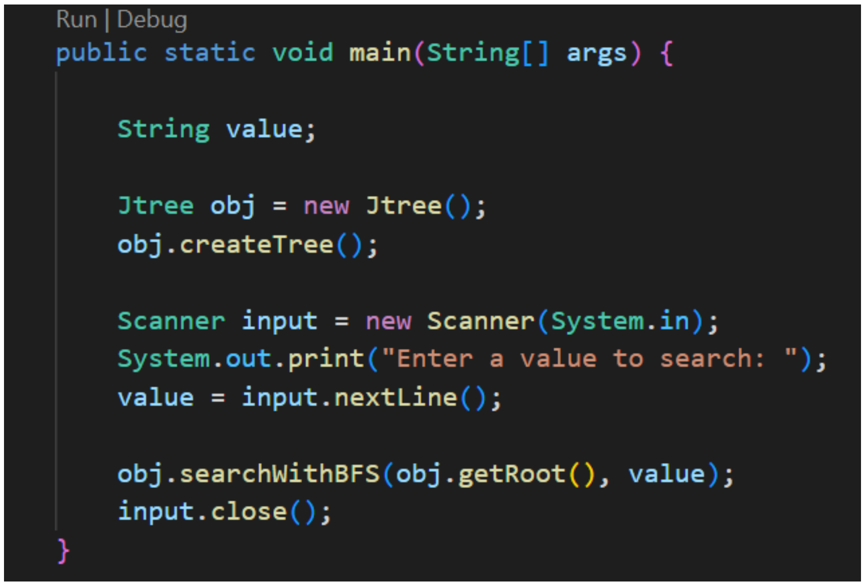


```
for (int i = 0; i < node.getChildCount(); i++) { // loop for node's children
    DefaultMutableTreeNode childNode = (DefaultMutableTreeNode) node.getChildAt(i); // get
    queue.add(childNode); // add child node to queue
    //System.out.println("Added to queue: " + childNode.getUserObject());
}
```

If the loop completes without finding the value, the method prints a message indicating that the value was not found in the tree.

In main:

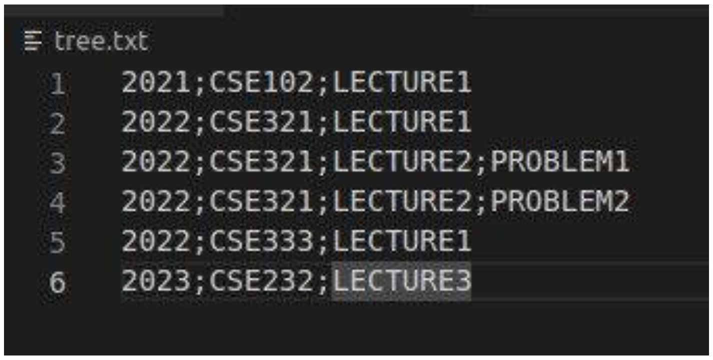
I first created the tree. Then read the value from user and search for the value with BFS.



```
public static void main(String[] args) {

    String value;

    Jtree obj = new Jtree();
    obj.createTree();

    Scanner input = new Scanner(System.in);
    System.out.print("Enter a value to search: ");
    value = input.nextLine();

    obj.searchWithBFS(obj.getRoot(), value);
    input.close();
}
```

TEST CASES:

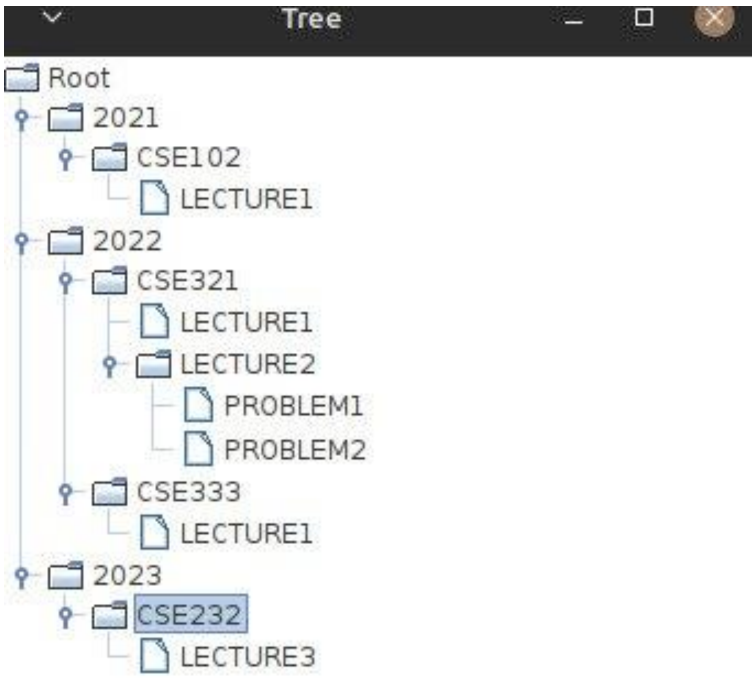
Txt:



tree.txt

```
2021;CSE102;LECTURE1
2022;CSE321;LECTURE1
2022;CSE321;LECTURE2;PROBLEM1
2022;CSE321;LECTURE2;PROBLEM2
2022;CSE333;LECTURE1
2023;CSE232;LECTURE3
```

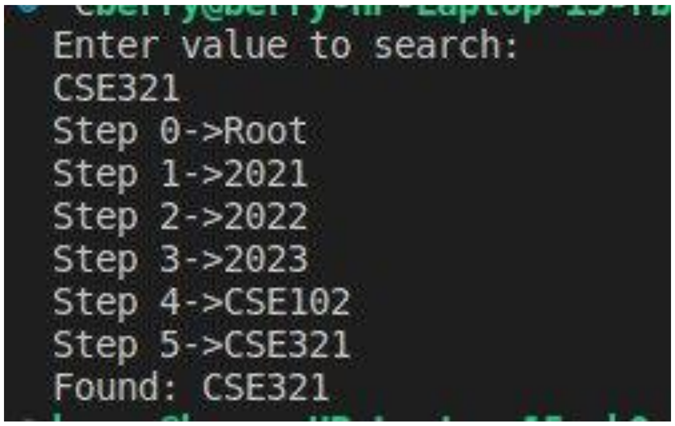
JTree:



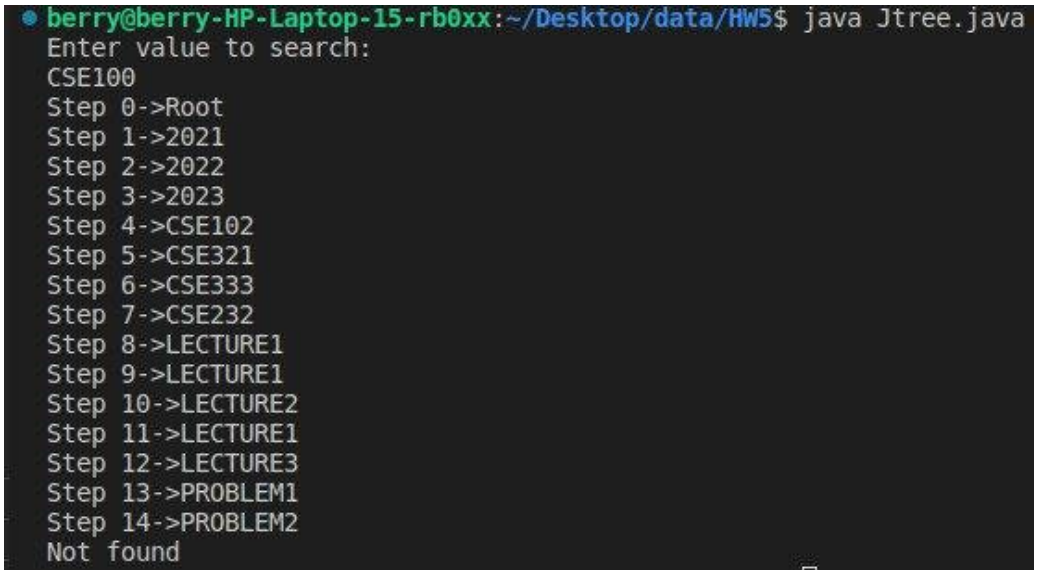
Tree

```
Root
  2021
    CSE102
      LECTURE1
  2022
    CSE321
      LECTURE1
      LECTURE2
        PROBLEM1
        PROBLEM2
    CSE333
      LECTURE1
  2023
    CSE232
      LECTURE3
```

With BFS:

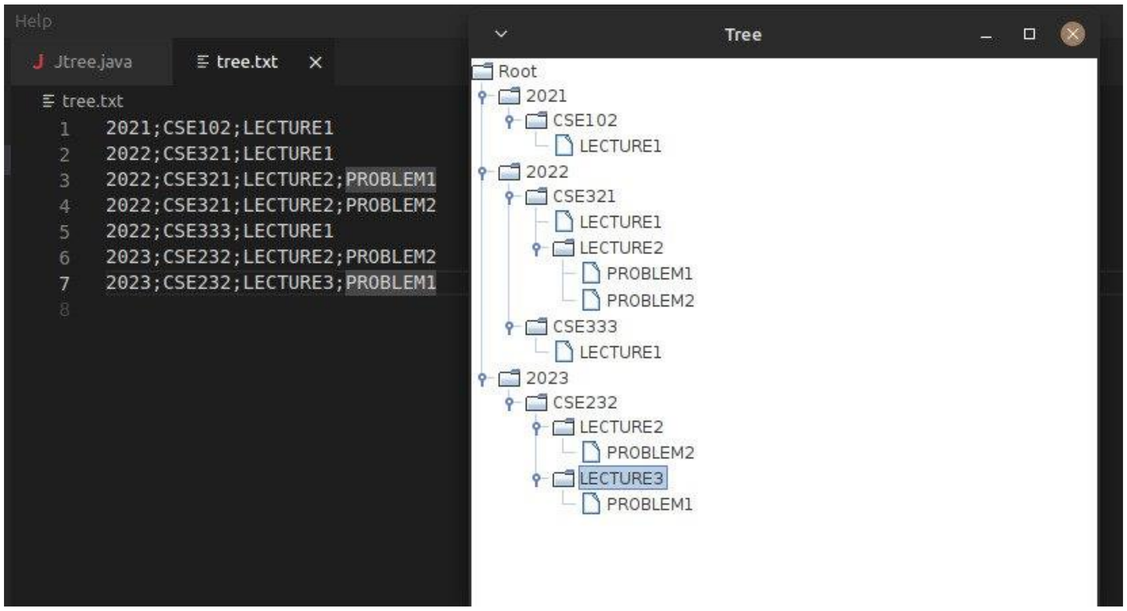


```
Enter value to search:
CSE321
Step 0->Root
Step 1->2021
Step 2->2022
Step 3->2023
Step 4->CSE102
Step 5->CSE321
Found: CSE321
```



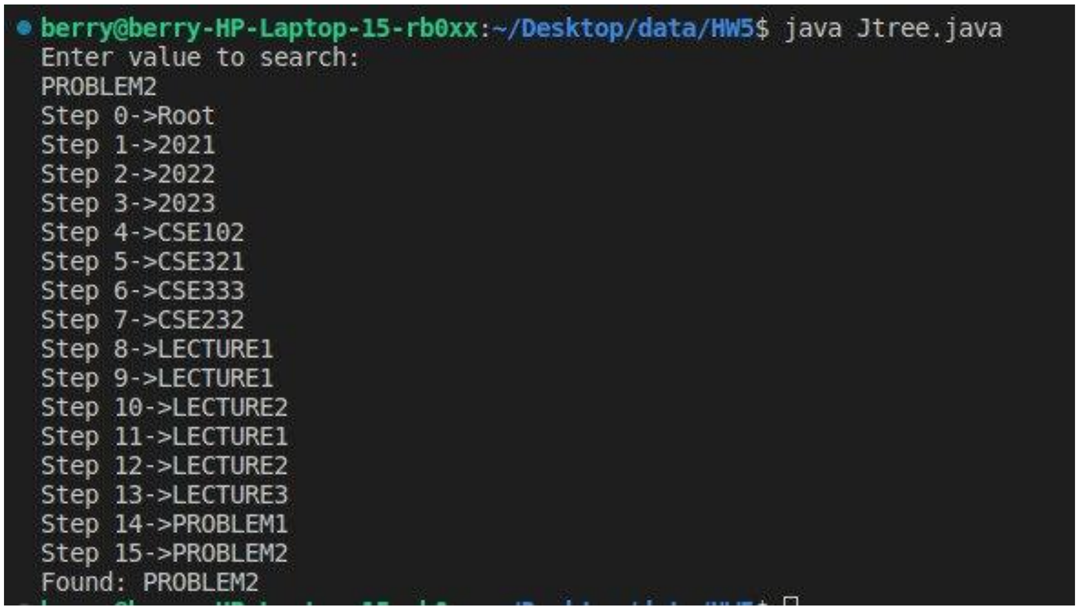
```
berry@berry-HP-Laptop-15-rb0xx:~/Desktop/data/HW5$ java Jtree.java
Enter value to search:
CSE100
Step 0->Root
Step 1->2021
Step 2->2022
Step 3->2023
Step 4->CSE102
Step 5->CSE321
Step 6->CSE333
Step 7->CSE232
Step 8->LECTURE1
Step 9->LECTURE1
Step 10->LECTURE2
Step 11->LECTURE1
Step 12->LECTURE3
Step 13->PROBLEM1
Step 14->PROBLEM2
Not found
```

Another example:



```
2021;CSE102;LECTURE1
2022;CSE321;LECTURE1
2022;CSE321;LECTURE2;PROBLEM1
2022;CSE321;LECTURE2;PROBLEM2
2022;CSE333;LECTURE1
2023;CSE232;LECTURE2;PROBLEM2
2023;CSE232;LECTURE3;PROBLEM1
```

```
Root
├ 2021
│ └ CSE102
│   └ LECTURE1
├ 2022
│ ├ CSE321
│ │ ├ LECTURE1
│ │ └ LECTURE2
│ │   ├ PROBLEM1
│ │   └ PROBLEM2
│ └ CSE333
│   └ LECTURE1
└ 2023
  └ CSE232
    ├ LECTURE2
    │ └ PROBLEM2
    └ LECTURE3
      └ PROBLEM1
```



```
berry@berry-HP-Laptop-15-rb0xx:~/Desktop/data/HW5$ java Jtree.java
Enter value to search:
PROBLEM2
Step 0->Root
Step 1->2021
Step 2->2022
Step 3->2023
Step 4->CSE102
Step 5->CSE321
Step 6->CSE333
Step 7->CSE232
Step 8->LECTURE1
Step 9->LECTURE1
Step 10->LECTURE2
Step 11->LECTURE1
Step 12->LECTURE2
Step 13->LECTURE3
Step 14->PROBLEM1
Step 15->PROBLEM2
Found: PROBLEM2
```