Osama Salah Alsubaie

201832180

ICS 202 DATA STRUCTERS LAB PROJECT

Source code

Code for GeneralTreeNode class:

```
import java.util.ArrayList;

public class GeneralTreeNode<T extends Comparable<? super T>> implements
Comparable<GeneralTreeNode> {

    public T info;
    private int MAXIMUM = 10;
    public GeneralTreeNode<T> parent;
    public ArrayList<GeneralTreeNode> children;

    public GeneralTreeNode(T info) {
        this.info = info;
        this.parent = null;
        children = new ArrayList<>(getMAXIMUM());
    }

    public void addChild(GeneralTreeNode child) {
        if (this.children.size() == 1 - getMAXIMUM()) {
            System.out.println("Parent node have a maximum number of children !");
        } else {
            this.children.add(child);
            child.parent = this;
        }
    }

    public int getMAXIMUM() {
        return MAXIMUM;
    }

    public void setMAXIMUM(int MAXIMUM) {
        this.MAXIMUM = MAXIMUM;
    }

    @Override
    public String toString() {
        return this.info.toString();
    }

    @Override
    public int compareTo(GeneralTreeNode node) {
        return ((String) this.info).compareTo((String) node.info);
    }
}
```

Code for GeneralTree class:

```
import java.util.*;

public class GeneralTree<T extends Comparable<? super T>> {

   GeneralTreeNode<T> root = null;

   public GeneralTree() {
   }

   public GeneralTree(GeneralTreeNode rootNode) {
      if (root == null)
         root = new GeneralTreeNode(rootNode);
      else
         System.out.println("Root is not empty, use insert method");
   }

   public void insert(GeneralTreeNode parentNode, GeneralTreeNode newNode) {
      if (isEmpty()) {
         root = parentNode;
         root.addChild(newNode);
      } else if (root.children.isEmpty() && root.info.equals(parentNode.info)) {
         root.addChild(newNode);
      } else {
         find(root, parentNode).addChild(newNode);
      }
   }

   public void delete(GeneralTreeNode existingNode) {
      if (isEmpty()) {
         System.out.println("The tree is empty !");
      } else if (find(root, existingNode) != null) {
         if (root.info.equals(existingNode.info)) {
            root.children.clear();
            root = null;
         } else {
            GeneralTreeNode tmp = find(root, new GeneralTreeNode(existingNode.toString()));
            tmp.parent.children.remove(tmp);
         }
      } else System.out.println("The node you want to delete does not exist !");

   }
```

```
public String search(GeneralTreeNode newNode) {
   if (isEmpty()) {
      System.out.println("The tree is empty !");
   } else if (find(root, newNode) != null) {
      if (root.info.equals(newNode.info))
         return newNode.toString();
      else {
         ArrayList arraylistPath = new ArrayList();
         GeneralTreeNode tmp = find(root, newNode);
         while (tmp != null) {
            arraylistPath.add(tmp);
            tmp = tmp.parent;
         }
         Collections.reverse(arraylistPath);
         return arraylistPath.toString().replace("[", "").replace("]", "");
      }
   }
   return "The node you want to search for does not exist !";
}


public void sortByLevel() {
   sortByLevel(root);
}

private void sortByLevel(GeneralTreeNode root) {
   if (isEmpty())
      return;
   Collections.sort(root.children);
   for (Object child : root.children) {
      if (!((GeneralTreeNode) child).children.isEmpty()) {
         sortByLevel((GeneralTreeNode) child);
      }
   }
}

public GeneralTreeNode find(GeneralTreeNode tmp, GeneralTreeNode findNode) {
   if (tmp.info.equals(findNode.info))
      return tmp;
   else {
      for (Object child : tmp.children) {
         GeneralTreeNode generalTreeNode = find((GeneralTreeNode) child, findNode);
         if (generalTreeNode != null)
            return generalTreeNode;
      }
   }
   return null;
}
```

```
public ArrayList<T> preOrder() {
   ArrayList treeList = new ArrayList();
   return preOrder(root, treeList);
}

private ArrayList<T> preOrder(GeneralTreeNode root, ArrayList list) {
   if (isEmpty())
      return list;
   list.add(root.info);
   for (Object child : root.children) {
      preOrder((GeneralTreeNode) child, list);
   }
   return list;
}

public ArrayList<T> postOrder() {
   ArrayList treeList = new ArrayList();
   return postOrder(root, treeList);
}

private ArrayList<T> postOrder(GeneralTreeNode root, ArrayList list) {
   if (isEmpty())
      return list;
   for (Object child : root.children) {
      postOrder((GeneralTreeNode) child, list);
   }
   list.add(root.info);
   return list;
}

public void bfs(GeneralTreeNode root) {
   if (root == null)
      return;

   Queue<GeneralTreeNode> queue = new LinkedList<>();
   queue.add(root);
   while (!queue.isEmpty()) {
      int size = queue.size();

      while (size > 0) {

         GeneralTreeNode generalTreeNode = queue.peek();
         queue.remove();
         System.out.print(generalTreeNode.info + " ");


         for (int i = 0; i < generalTreeNode.children.size(); i++)
            queue.add((GeneralTreeNode) generalTreeNode.children.get(i));
         size--;
      }
      System.out.println();
   }
}
```

```
    public boolean isEmpty() {
        return root == null;
    }

    public void numberOfFilesAndFolders(GeneralTreeNode tmp) {

        int folders = 0;
        int files = 0;

        if (!root.children.isEmpty()) {
            folders++;
            for (Object child : tmp.children) {
                if (!((GeneralTreeNode) child).children.isEmpty())
                    folders++;
                else
                    files++;
                for (Object childOfChild : ((GeneralTreeNode) child).children) {
                    files++;
                }
            }
        } else
            files++;
        System.out.println("The number of folders : " + folders + "\nThe number of files :" + files);
    }

}
```

Code for TestClass class:

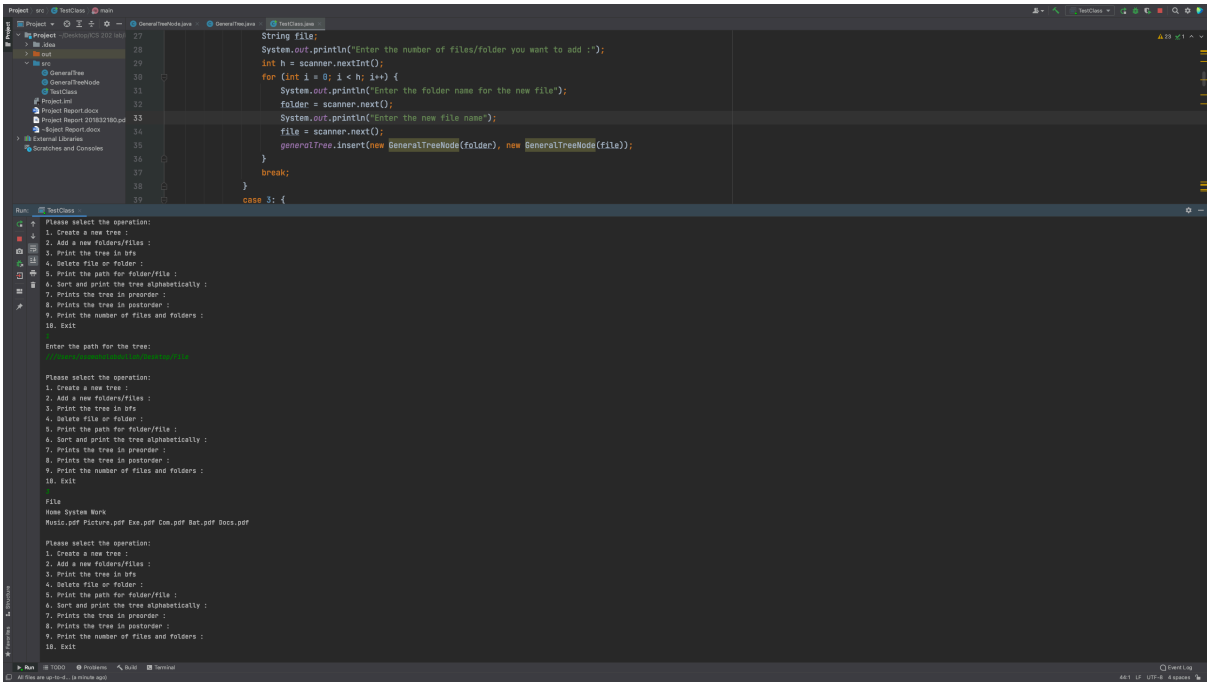
```
import java.io.File;
import java.util.Scanner;

public class TestClass {

    private static GeneralTree generalTree = new GeneralTree();
    private static int num;

    public static void main(String[] args) throws Exception {


        do {
            Scanner scanner = new Scanner(System.in);
            num = getMenuChoice();

            switch (num) {
                case 1: {

                    System.out.println("Enter the path for the tree:");
                    insert(new File(scanner.nextLine()));
                    break;
                }
                case 2: {
                    String folder;
                    String file;
                    System.out.println("Enter the number of files/folder you want to add :");
                    int h = scanner.nextInt();
                    for (int i = 0; i < h; i++) {
                        System.out.println("Enter the folder name for the new file");
                        folder = scanner.next();
                        System.out.println("Enter the new file name");
                        file = scanner.next();
                        generalTree.insert(new GeneralTreeNode(folder), new GeneralTreeNode(file));
                    }
                    break;
                }
                case 3: {
                    if (!generalTree.isEmpty())
                        generalTree.bfs(generalTree.root);
                    else
                        System.out.println("The tree is empty !");
                    break;
                }


                case 4: {
                    if (!generalTree.isEmpty()) {
                        System.out.println("Enter the file/folder name to delete :");
                        generalTree.delete(new GeneralTreeNode(scanner.next()));
                    } else
                        System.out.println("The tree is empty !");
                    break;
                }
```

```
            case 5: {
                if (!generalTree.isEmpty()) {
                    System.out.println("Enter the file/folder name to print it's path :");
                    System.out.println(generalTree.search(new GeneralTreeNode(scanner.next())));
                } else
                    System.out.println("The tree is empty !");
                break;
            }
            case 6: {
                if (!generalTree.isEmpty()) {
                    generalTree.sortByLevel();
                    generalTree.bfs(generalTree.root);
                } else
                    System.out.println("The tree is empty !");
                break;
            }
            case 7: {
                if (!generalTree.isEmpty())
                    System.out.println(generalTree.preOrder());
                else
                    System.out.println("The tree is empty !");
                break;
            }
            case 8: {
                if (!generalTree.isEmpty())
                    System.out.println(generalTree.postOrder());
                else
                    System.out.println("The tree is empty !");
                break;
            }
            case 9: {
                if (!generalTree.isEmpty())
                    generalTree.numberOfFilesAndFolders(generalTree.root);
                else
                    System.out.println("The number of folders : " + 0 + "\nThe number of files :" + 0);
                break;
            }
            case 10: {
                System.out.println("Terminating ...");
                break;
            }
        }
        System.out.println();
    } while (num != 10);


}
```

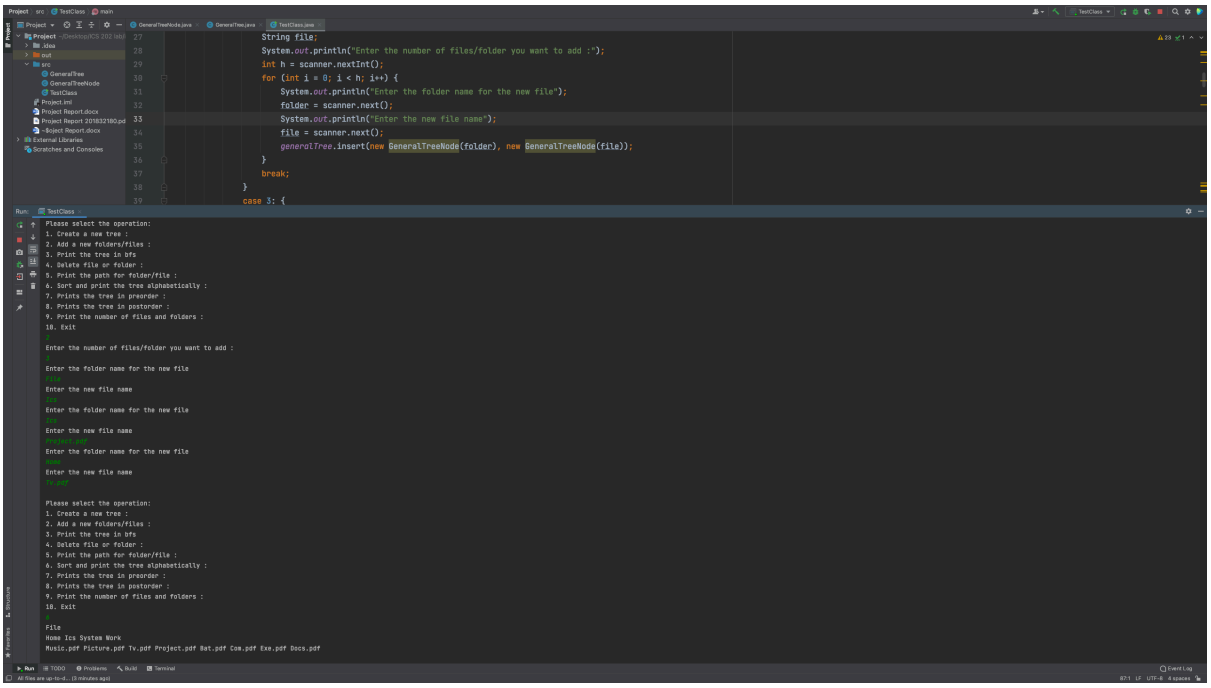
```
    public static int getMenuChoice() {
        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("Please select the operation: ");
            System.out.println("1. Create a new tree :");
            System.out.println("2. Add a new folders/files :");
            System.out.println("3. Print the tree in bfs ");
            System.out.println("4. Delete file or folder :");
            System.out.println("5. Print the path for folder/file :");
            System.out.println("6. Sort and print the tree alphabetically :");
            System.out.println("7. Prints the tree in preorder :");
            System.out.println("8. Prints the tree in postorder :");
            System.out.println("9. Print the number of files and folders :");
            System.out.println("10. Exit");

            choice = scanner.nextInt();
            if (choice < 1 || choice > 10)
                System.out.println("Error: Wrong operation!");
        } while (choice < 1 || choice > 10);
        return choice;
    }

public static void insert(File folder) {
        for (File fileEntry : folder.listFiles()) {
            if (fileEntry.isDirectory() && !fileEntry.getName().equals(".DS_Store")) {
                generalTree.insert(new GeneralTreeNode(fileEntry.getParentFile().getName()),
                        new GeneralTreeNode(fileEntry.getName()));
                insert(fileEntry);
            } else {
                if (!fileEntry.getName().equals(".DS_Store")) {
                    generalTree.insert(new GeneralTreeNode(fileEntry.getParentFile().getName()),
                            new GeneralTreeNode(fileEntry.getName()));
                }
            }
        }
    }

}
```

Output screen shots

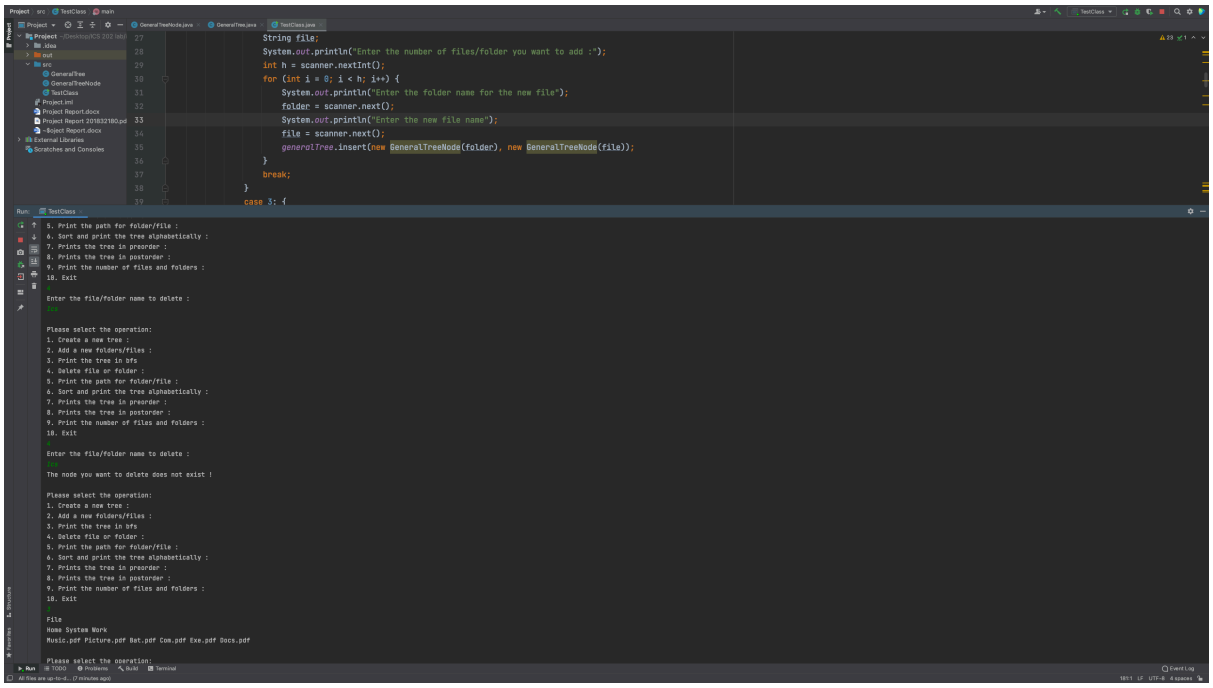
Generate a new tree and print the tree:



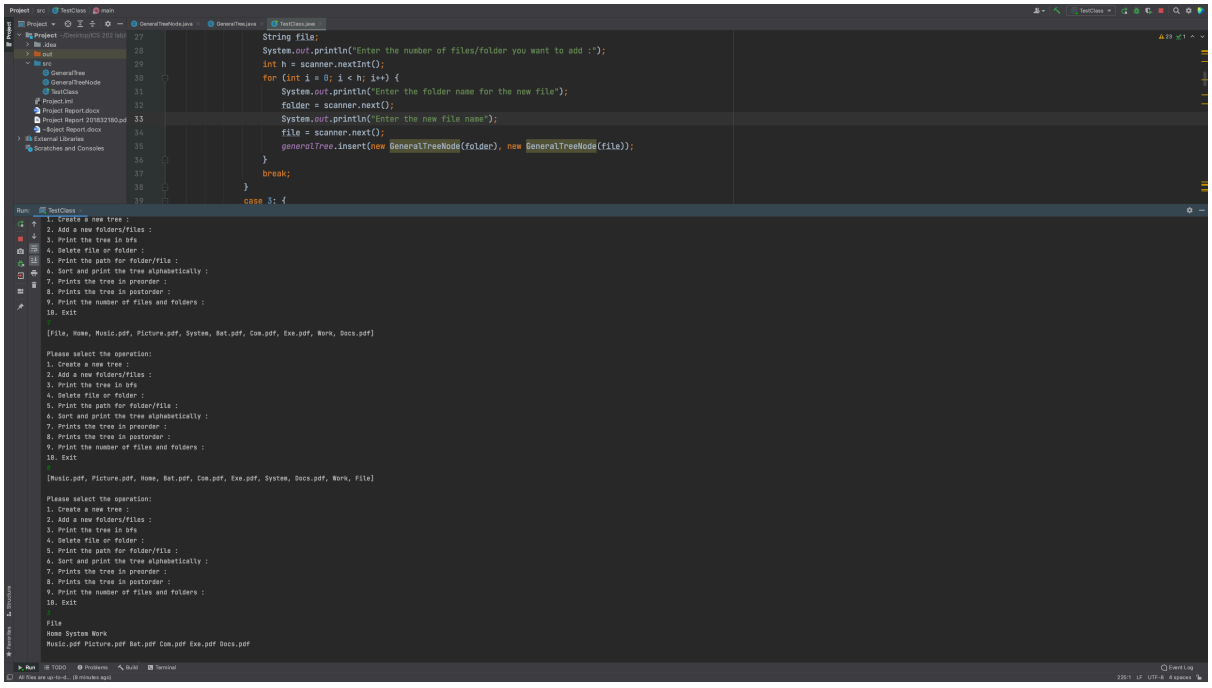
Insert a new folder “Ics” which has a child “Project.pdf”, also insert file ”Tv.pdf” to Home folder. Also, sort and print the tree after that.



Delete folder "Test", then print the tree:



Pre order && Post order && bfs for the tree:



Print the number of files and folders:

```
27                  String file;
28                  System.out.println("Enter the number of files/folder you want to add :");
29                  int h = scanner.nextInt();
30                  for (int i = 0; i < h; i++) {
31                      System.out.println("Enter the folder name for the new file");
32                      folder = scanner.next();
33                      System.out.println("Enter the new file name");
34                      file = scanner.next();
35                      generalTree.insert(new GeneralTreeNode(folder), new GeneralTreeNode(file));
36                  }
37                  break;
38              }
39              case 3: {
```

```
Please select the operation:
1. Create a new tree :
2. Add a new folders/files :
3. Print the tree in bfs
4. Delete file or folder :
5. Print the path for folder/file :
6. Sort and print the tree alphabetically :
7. Prints the tree in preorder :
8. Prints the tree in postorder :
9. Print the number of files and folders :
10. Exit

File
Home System Work
Music.pdf Picture.pdf Bat.pdf Com.pdf Exe.pdf Docs.pdf

Please select the operation:
1. Create a new tree :
2. Add a new folders/files :
3. Print the tree in bfs
4. Delete file or folder :
5. Print the path for folder/file :
6. Sort and print the tree alphabetically :
7. Prints the tree in preorder :
8. Prints the tree in postorder :
9. Print the number of files and folders :
10. Exit

The number of folders : 4
The number of files :6

Please select the operation:
1. Create a new tree :
2. Add a new folders/files :
3. Print the tree in bfs
4. Delete file or folder :
5. Print the path for folder/file :
6. Sort and print the tree alphabetically :
7. Prints the tree in preorder :
8. Prints the tree in postorder :
9. Print the number of files and folders :
10. Exit
```

Print the path for selected node:

```
27                  String file;
28                  System.out.println("Enter the number of files/folder you want to add :");
29                  int h = scanner.nextInt();
30                  for (int i = 0; i < h; i++) {
31                      System.out.println("Enter the folder name for the new file");
32                      folder = scanner.next();
33                      System.out.println("Enter the new file name");
34                      file = scanner.next();
35                      generalTree.insert(new GeneralTreeNode(folder), new GeneralTreeNode(file));
36                  }
37                  break;
38              }
39              case 3: {
```

```
Please select the operation:
1. Create a new tree :
2. Add a new folders/files :
3. Print the tree in bfs
4. Delete file or folder :
5. Print the path for folder/file :
6. Sort and print the tree alphabetically :
7. Prints the tree in preorder :
8. Prints the tree in postorder :
9. Print the number of files and folders :
10. Exit

The number of folders : 4
The number of files :6

Please select the operation:
1. Create a new tree :
2. Add a new folders/files :
3. Print the tree in bfs
4. Delete file or folder :
5. Print the path for folder/file :
6. Sort and print the tree alphabetically :
7. Prints the tree in preorder :
8. Prints the tree in postorder :
9. Print the number of files and folders :
10. Exit

Enter the file/folder name to print it's path :
Bat.pdf
File, System, Bat.pdf

Please select the operation:
1. Create a new tree :
2. Add a new folders/files :
3. Print the tree in bfs
4. Delete file or folder :
5. Print the path for folder/file :
6. Sort and print the tree alphabetically :
7. Prints the tree in preorder :
8. Prints the tree in postorder :
9. Print the number of files and folders :
10. Exit
```