Sunday

22/Dec/2012

King Fahad University Of Petroleum and Memorials

Information and Computer Science Department (ICS)

ICS 202 – Project

ICS 202 Final Project

ID: 201040340 **Student Name:** Saleh Al-Ghusson

ID: 201049260 **Student Name:** Faisal Alsalman

1) Introduction:

This project is about managing a windows directory by inserting, deleting and much more methods.

In our project we constructed 20 classes:.

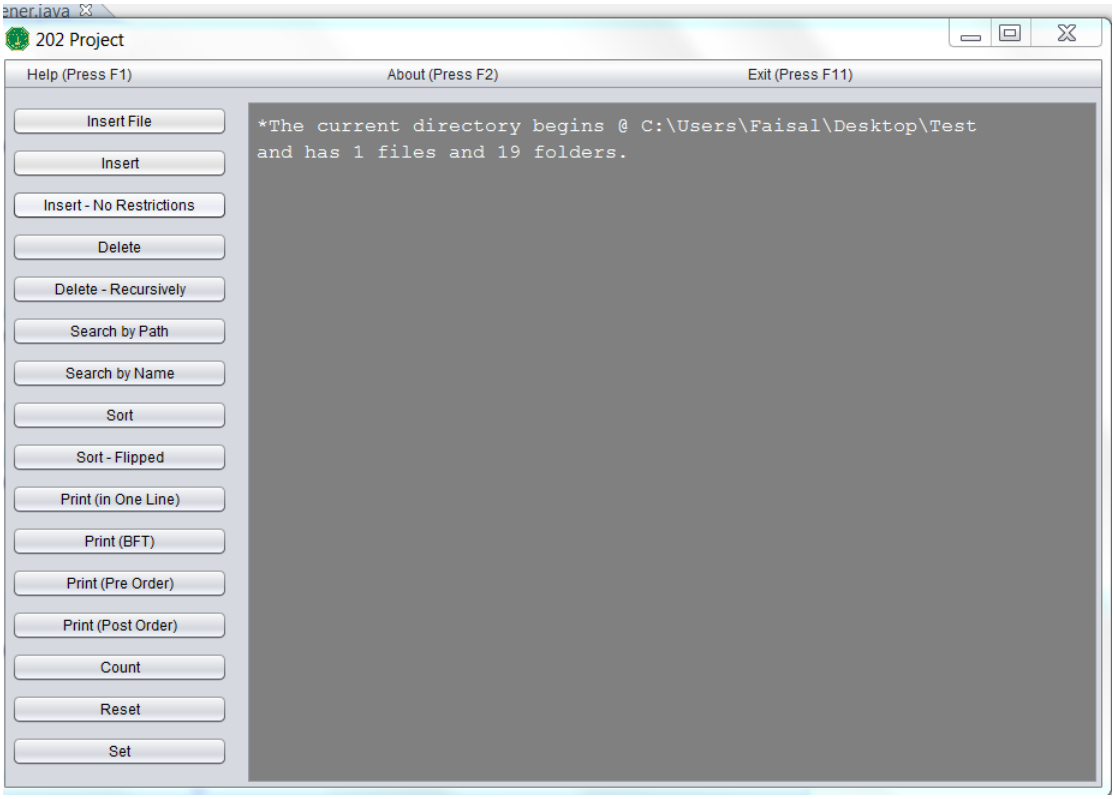
- Node.
- NodeFile.
- NodeFolder.
- Tree
- Window.
- One.
- Two.
- Three
- Listener.
- Stack.
- Queue.
- Test.

Also defined two Exceptions:

- GreaterThanMaxException.
- DirectoryDoesNotExist. In this report the source code will be listed and a brief description will be provided.

2) Screen shots and their implementations:

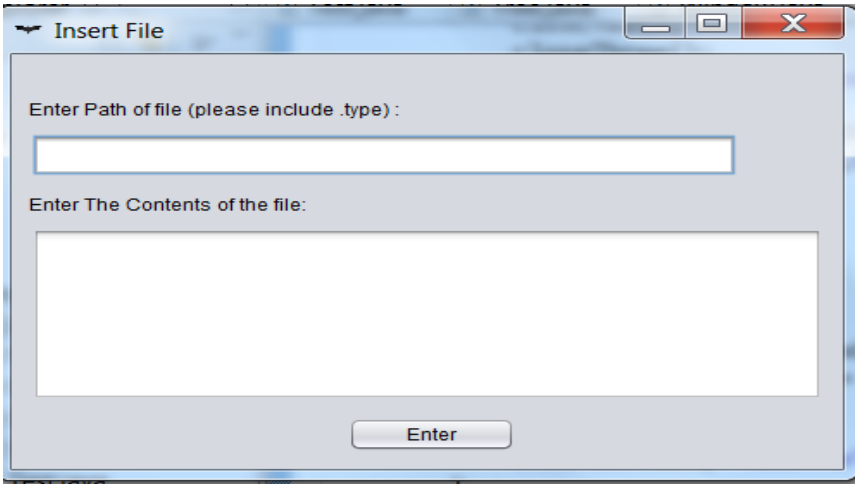
Firstly when the program runs a Java frame will appear showing the options that a user can choose from.



When the user chooses:

1- Insert File:- (File Only)

A frame will appear asking the user to write the path to the file he wants to create. The contents of the file, for example as shown:-



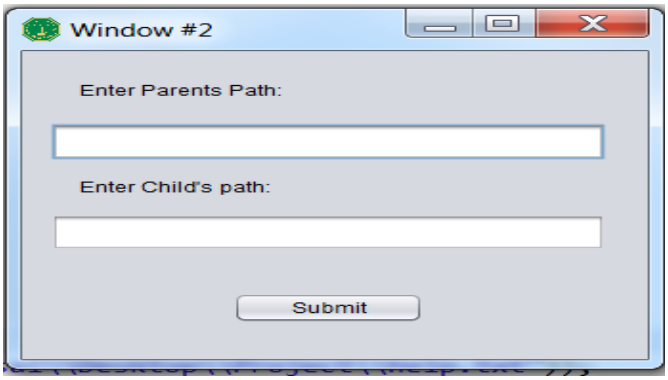
The Insert File implementation is in the tree class:-

```
protected int insertFile(String h, String hj)throws GreaterThanMaxException,
IOException, FileNotFoundException, DirectoryDoesNotExist{
            int x = 0;
            if(!h.contains("\\")){           //path is wrong (to avoid
exceptions)
                  return 0;
            }
            else if(searchPath(h.substring(0,h.lastIndexOf("\\"))) == null){
      //direct parent does not exist
                  return 0;
            }
            else if(searchPath(h) != null){       //direct parent does not
exist
                  x = 1;
            }
            File f = new File(h);
            if(!f.exists())
                  f.createNewFile();
            PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(h, true)));
            out.println(hj);
```

```
            out.close();
            callScan();
            if(x == 0)
                  return 2;
            else
                  return x;
      }
```

2- Insert: (Folder only)

A new frame will appear asking the user to write the path of the parent folder and the path of the folder he wants to add as shown:-



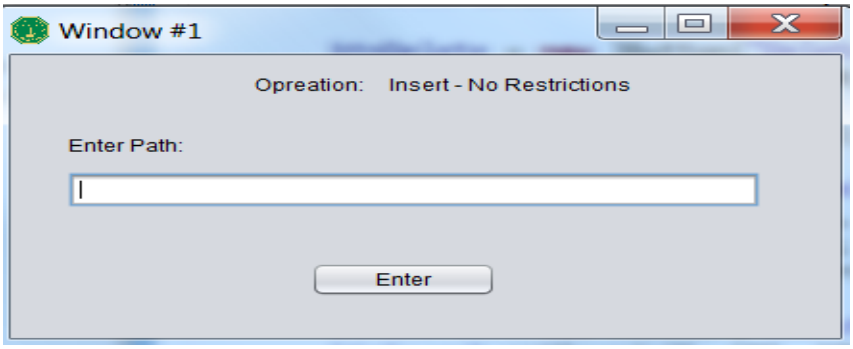
The insert implementation is in the tree class:-

```
protected boolean insert(String p, String c)throws GreaterThanMaxException,
DirectoryDoesNotExist{           //restricted
            if(searchPath(p) == null){                  //parent does not
exist
                  return false;
            }
            else if(!c.contains(p)){         //child unrelated to parent
                  return false;
            }
            else if(searchPath(c) != null && new File(c).isDirectory()){
      //child exists
                  return false;
            }
            boolean n = new File(c).mkdir();
            callScan();
            return n;}}
```

The method will take two variables, the parent path and the child path. Then will return a boolean true if the insertion was a success, false otherwise.

3- Insert-No Restrictions:- (Folder only)

A new frame will appear asking the user to insert the path exactly to where he/she wants to create the new folder as shown:-



The insertDirect method is in the tree class:-

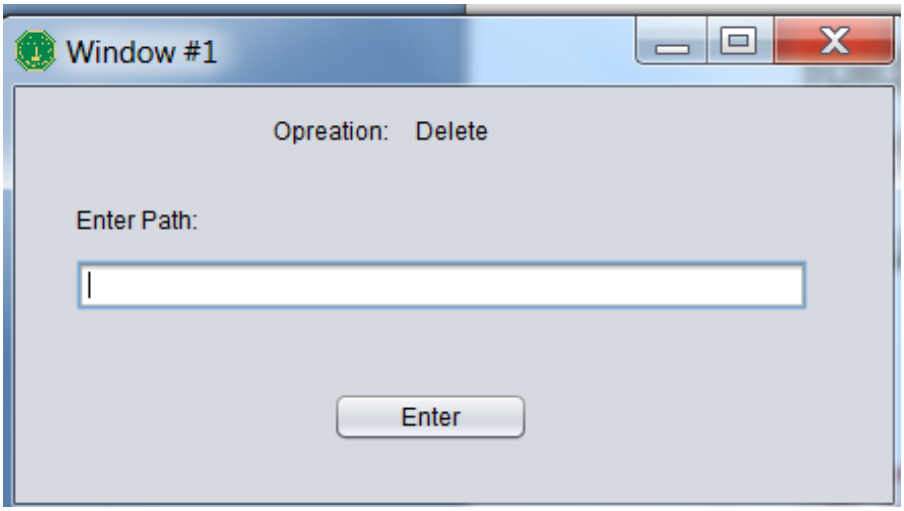
```
     protected boolean insertDirect(String c)throws GreaterThanMaxException,
DirectoryDoesNotExist{           //no restrictions
          if(c.length() < 1 || c.charAt(1) != ':')
               return false;
          if(!c.contains(root.path))
               return false;
          boolean n = new File(c).mkdirs();
          callScan();
          return n; }
```

This method will take a complete path to create a new folder. Unlike the previous method, a directory will be created if the

direct parent does not exist, this method will automatically create all the needed directories to insert the given directory.

4- Delete:- (File and Directory)

A frame will appear asking the user to insert the folder he wants to be deleted by writing its path as shown:-



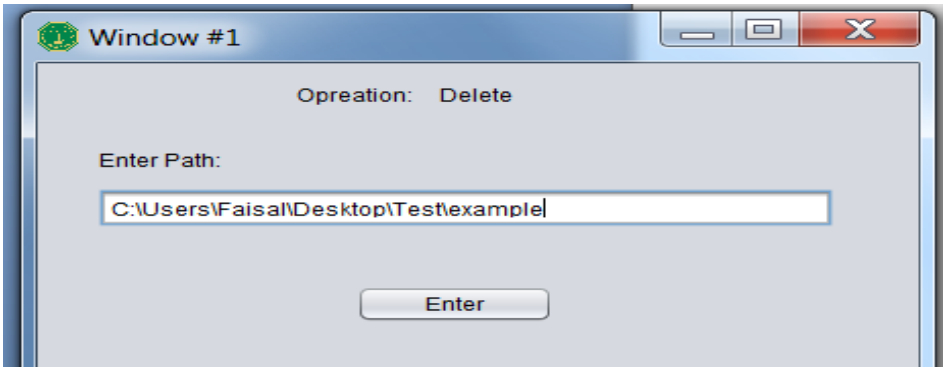
The Delete method is in the tree class:-

```
     protected boolean delete(String h)throws GreaterThanMaxException,
DirectoryDoesNotExist{
          if(searchName(h) == null && searchPath(h) == null)
               return false;
          boolean n = new File(h).delete();
          callScan();
          return n;
     }
```

This method will only delete files and empty directories

5- Delete -Recursively:-

A frame will appear asking the user to insert the path of folder to be deleted, and the children of the folder will be deleted too.



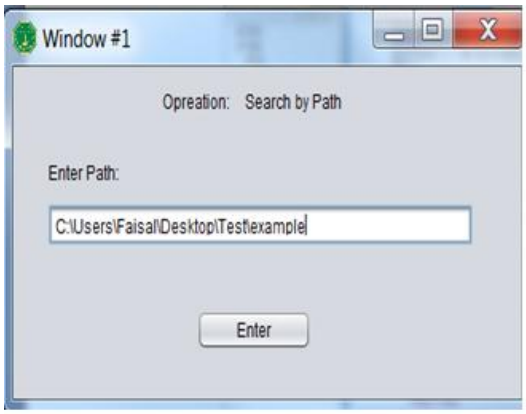
The deleteRec method is in the tree class:-

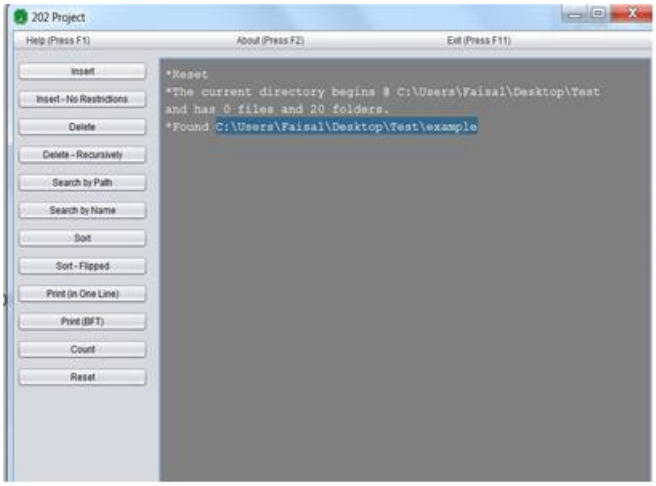
```
     protected boolean deleteRec(String s)throws GreaterThanMaxException,
DirectoryDoesNotExist{
          if(s.length() < 1 || s.charAt(1) != ':')
               return false;
          if(!s.contains(root.path))
               return false;

          File f = new File(s);
          if (f.isDirectory()) {
               for (File c : f.listFiles())
                    deleteRec(c.getPath());
          }
          if (!f.delete()){
               callScan();
               return false;
          }
          callScan();
          return true;
     }
```

Unlike the previous method this one will delete directories that are not empty, in addition to all the features of the previous method.

6- Search by path:-

A frame will appear asking the user to write the path of a file to search and then it will print the file path as shown:-





The method searchPath is in the tree class :-

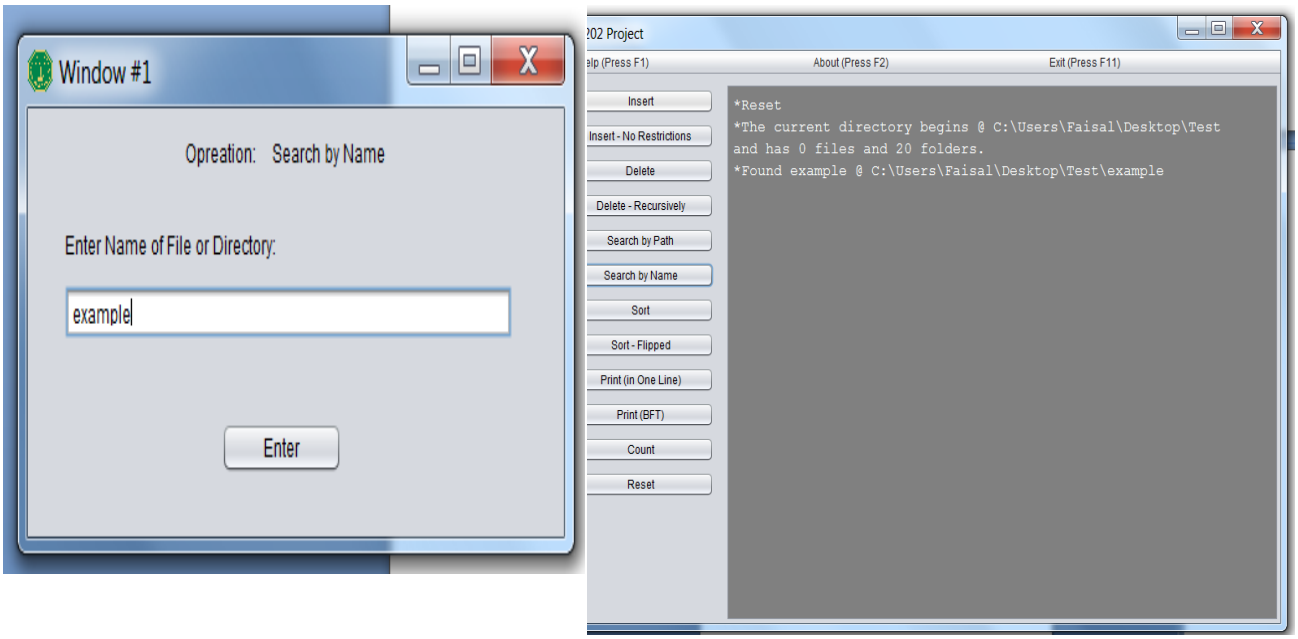
```
protected String searchPath(String h){          //works w & w/o .type
      String search = h;
      boolean found = false;
      Queue<Node> q = new Queue<Node>();
      Node r;
      NodeFolder p;
      q.enqueue(root);

      while(!q.isEmpty() && !found){
            r = q.dequeue();
            File f = new File(r.path);
            if(f.exists()){
                  if(f.isDirectory()){
```

```
                        p = (NodeFolder) r;
                        for(Node nd : p.children)
                              q.enqueue(nd);
                  }
                  if(r.name.contains("."))
                        if(r.path.substring(0,
r.path.lastIndexOf('.')).equals(search)){
                              found = true;
                              search = r.path;
                        }
                  if(r.path.equals(search)){
                        found = true;
                        search = r.path;
                  }
            }
      }
      if(found)
            return search;
      return null;
}
```

The method will take the path and then store it as a string. It will create a boolean "found" as false and will create queue to insert the elements in it. The it will check and compare if the inserted path does exist the it will return "found" as true.

7- Search by name:-

A frame will appear asking the user to write the name of a folder as shown:-



The method searchName is in the tree class:-

```
protected String searchName(String h){
      String search = h;          //name w & w/o .type
      boolean found = false;
      Queue<Node> q = new Queue<Node>();
      Node r;
      NodeFolder p;
      q.enqueue(root);

      while(! q.isEmpty()){
            r = q.dequeue();
            File f = new File(r.path);
            if(f.exists()){
                  if(f.isDirectory()){
                        p = (NodeFolder) r;
                        for(Node nd : p.children)
                              q.enqueue(nd);

                  }
                  if(r.name.contains("."))
                        if(r.name.substring(0,
r.name.lastIndexOf('.')).equals(search)){
```

```
                        found = true;
                        search = r.path;
                    }

                if(r.name.equals(search)){
                    found = true;
                    search = r.path;
                }
            }
        }
        if(found)
            return search;
        return null;
    }
```

Unlike the previous method this only needs the name of the file, and will return it's path. In case multiple files with the same name exist, the first one encountered will be returned.

8- Sort:-

Will sort the tree node by node in alphabetical order.

The method Sort is in the tree class:-

```
    protected void sort(){
        Queue<Node> q = new Queue<Node>();
        Node r;
        NodeFolder p;
        q.enqueue(root);

        while(!q.isEmpty()){
            r = q.dequeue();
            if(new File(r.path).exists() && new
File(r.path).isDirectory()){
                int x = 0;
                if(new File(r.path).list() != null)
                    x = new File(r.path).list().length;
                p = (NodeFolder) r;

                String[] array = new String[x];
                Node[] tmp = p.children.clone();

                for(int i = 0; i<x; i++){
                    array[i] = p.children[i].name;
                }
```

```
                    Arrays.sort(array);
                    for(int i = 0; i<x; i++){
                        for(Node nd : tmp)
                            if(nd != null)
                                if(nd.name .equals(array[i])){
                                    p.children[i] = nd;
                                }
                    }
                    for(Node n: p.children)
                        q.enqueue(n);
                }
            }
        }
```

9- Sort- flipped:-

Will sort the tree node by node in reverse alphabetical order.

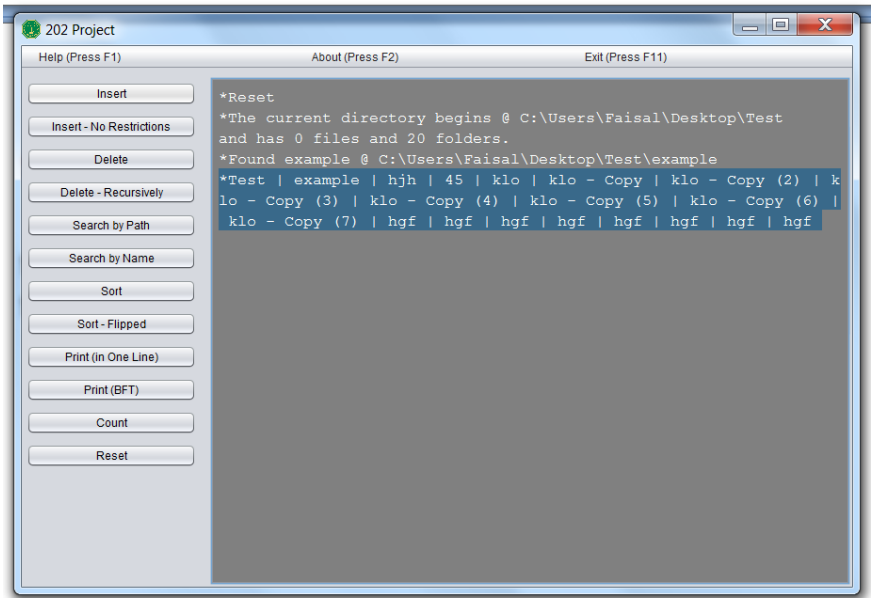
The method SortFlip is in the tree class:-

```
protected void sortFlip(){
        Queue<Node> q = new Queue<Node>();
        Node r;
        NodeFolder p;
        q.enqueue(root);

        while(!q.isEmpty()){
            r = q.dequeue();
            if(new File(r.path).exists() && new
File(r.path).isDirectory()){          //if folder
                    int x = 0;
                    if(new File(r.path).list() != null)
                        x = new File(r.path).list().length;
                    p = (NodeFolder) r;

                    String[] array = new String[x];
                    Node[] tmp = p.children.clone();

                    for(int i = 0; i<x; i++){
                        array[i] = p.children[i].name;
                    }
                    Arrays.sort(array);

                    for(int i = 0; i < array.length/2; i++){          //flip
algorithm
                        String temp = array[i];
```

```
                    array[i] = array[array.length - i - 1];
                    array[array.length - i - 1] = temp;
                }
                for(int i = 0; i<x; i++){
                    for(Node nd : tmp)
                        if(nd != null)
                            if(nd.name .equals(array[i])){
                                p.children[i] = nd;
                            }
                }
                for(Node n: p.children)
                    q.enqueue(n);
            }
        }
    }
```

10- Print (in one line) :

When choosing the print (in one line) the program will show the files name in one line in the output screen as shown:-



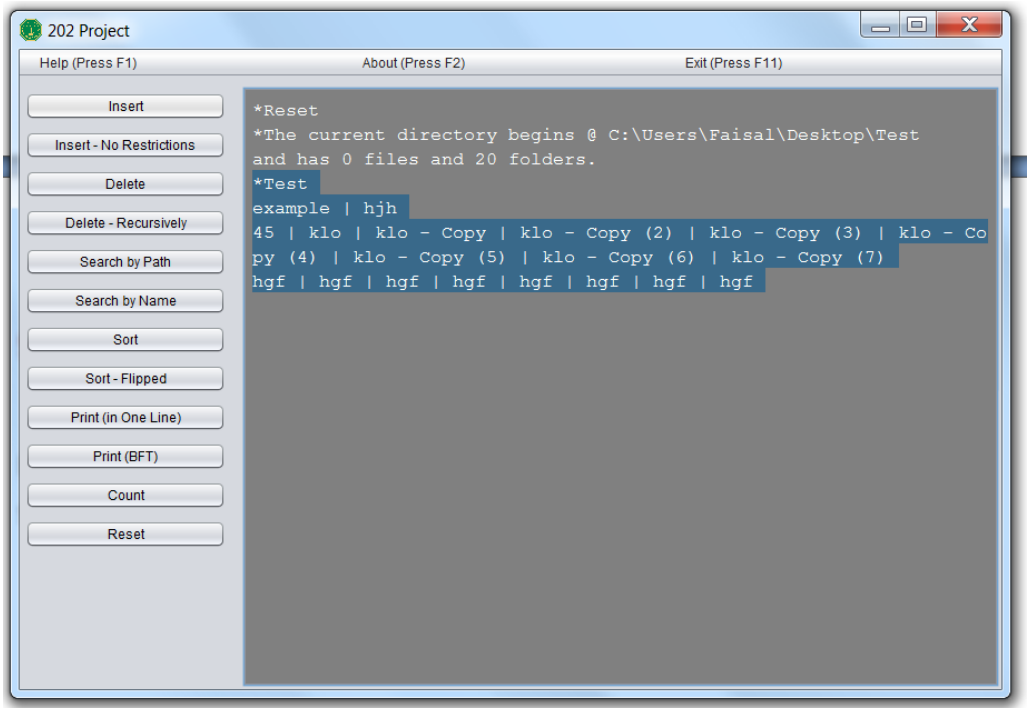
The method Print is in the tree class:-

```
protected String print(){
      return print(root);
}
protected String print(NodeFolder r){       //in one line
      Node f;
      Queue<Node> q = new Queue<Node>();
      String x = new String(">");

      q.enqueue(r);

      while(! q.isEmpty()){
            f = q.dequeue();

            if(new File(f.path).isFile()){
                  x+=f.name + " | ";
            }
            else if(new File(f.path).isDirectory()){
                  r = (NodeFolder) f;
                  x+=r.name + " | ";
```

```
                    for(int i = 0; i<new File(f.path).list().length; i++){
                        q.enqueue(r.children[i]);
                    }
                }
            }
            return x.substring(0, x.lastIndexOf('|'))+"#";
        }
```

11- PrintBFT:

Will print the tree in Breadth First Traversal with each level in a new line.



12- Print(Pre-Order):

This method is about printing the names of the file in pre-order traversal.

The method Printpre is in the tree class:-

```
protected String printPre(){
	return printPre(root);
}
protected String printPre(NodeFolder r){		//level by level
	Node f;
	Stack<Node> s = new Stack<Node>();
	String x = new String(">");
	s.push(r);

	while(! s.isEmpty()){
		f = s.pop();
		if(new File(f.path).isFile()){
			x+=f.name + " | ";
		}
		else if(new File(f.path).isDirectory()){
			r = (NodeFolder) f;

			x+=r.name + " | ";
			for(int i = 0; i<new File(f.path).list().length; i++){
				s.push(r.children[i]);
			}
		}
	}
	return x.substring(0, x.lastIndexOf(" |")) + " #";
}
```

13- Print(Post-Order) :-

The method will print it in a Post -order traversal.

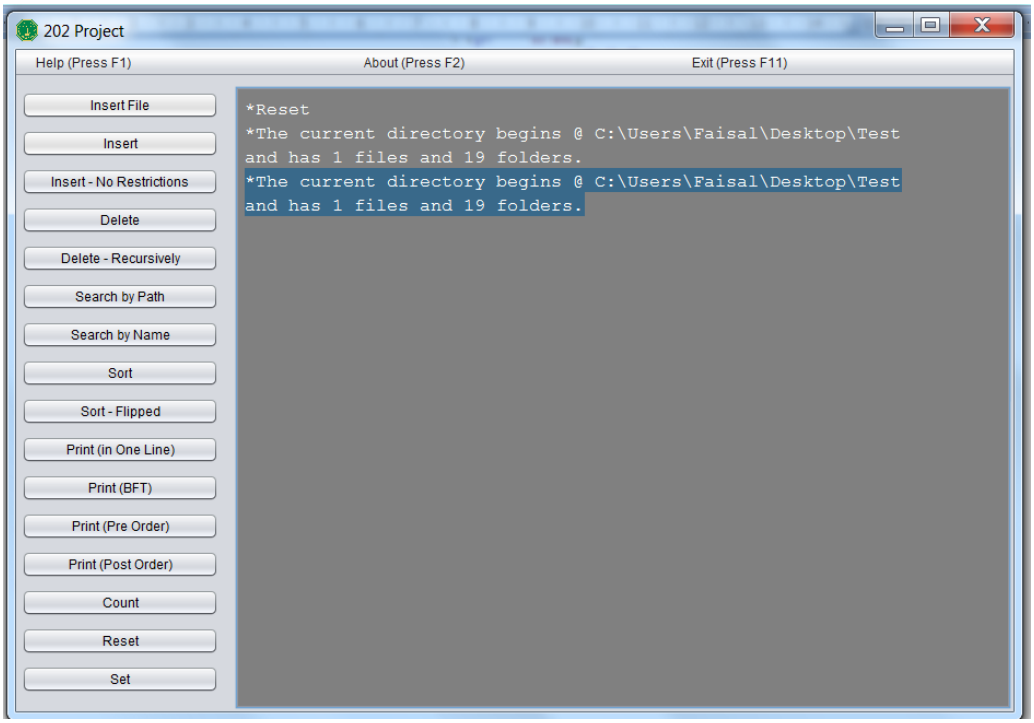
The method printPre is in the tree class:-

```
protected String printPost(){
	return printPost(root);
}
protected String printPost(NodeFolder r){		//level by level
	Node f;
	Stack<Node> s = new Stack<Node>();
	String x = new String(">");
	s.push(r);
	boolean k = false;
	while(! s.isEmpty()){
		f = s.pop();
		k = false;
		if(new File(f.path).isFile()){
			if(!f.vis){
				f.vis = true;
				x+=f.name + " | ";
			}
		}
		else if(new File(f.path).isDirectory()){
			r = (NodeFolder) f;

			if(new File(r.path).listFiles().length > 0 && !r.vis){
				k = true;
			}
			if(k){
				r.vis = true;
				s.push(r);
			}
			else
				if(!r.pr){
					r.pr = true;
					x+=r.name + " | ";
				}
			for(int i = 0; i<new File(f.path).list().length; i++){
				s.push(r.children[i]);
			}

		}
	}
```

```
        return x.substring(0, x.lastIndexOf(" |")) + " #";
    }
```

14- Count:-

This method will count the numbers of files and folders in the root and thin print it in the output screen as shown



The method is in the tree class :-

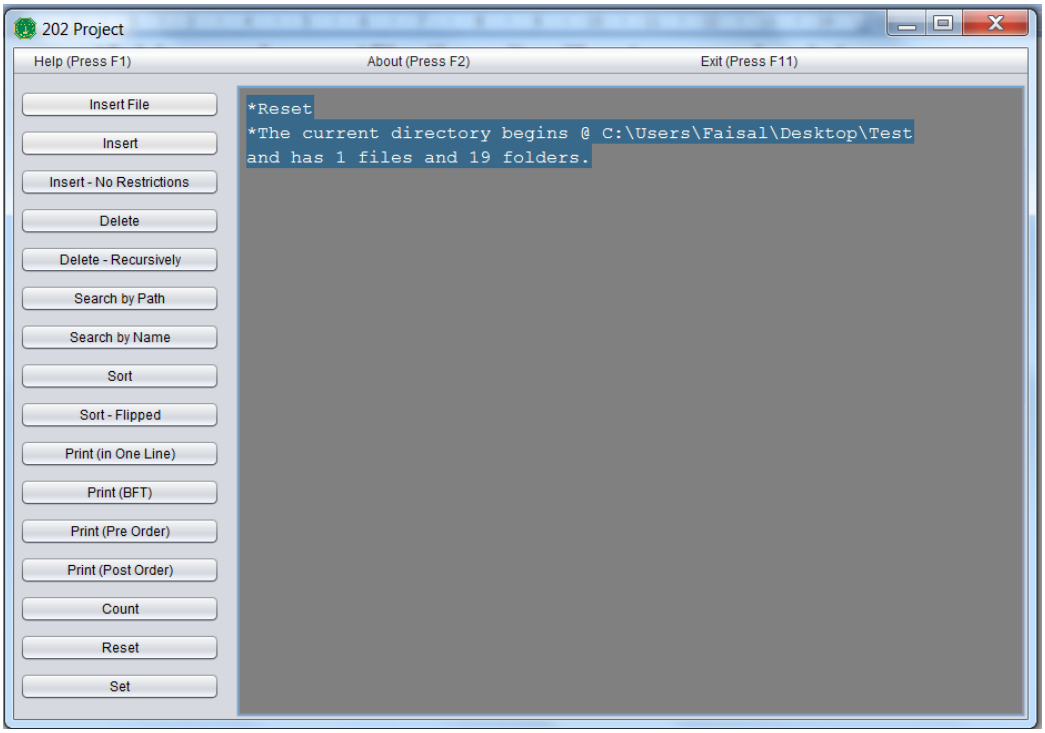
```
    protected String count(){                    //includes folder
        root.files = countFiles(root);
        root.folders = countFolders(root);
        return root.files + " files and " + root.folders +" folders." ;
    }
    private int countFiles(NodeFolder r){
        Queue<Node> q = new Queue<Node>();
        Node t;
        int d=0;
        File f = new File(r.path);

        q.enqueue(r);
        while(!q.isEmpty()){
            t = q.dequeue();
            f = new File(t.path);
```

```
                if(f.isFile()){
                      d++;
                }
                else if(f.isDirectory()){
                      String[] a = f.list();
                      int u = a.length;
                      r = (NodeFolder) t;

                      for(int i = 0; i<u; i++){
//                          if(r.children[i] != nnnn)
                            q.enqueue(r.children[i]);
                      }
                }
          }
          return d;
     }
     private int countFolders(NodeFolder r){
          Queue<Node> q = new Queue<Node>();
          Node t;
          int d=0;
          File f = new File(r.path);

          q.enqueue(r);
          while(!q.isEmpty()){
                t = q.dequeue();
                f = new File(t.path);

                if(f.isFile()){
                      continue;
                }
                else if(f.isDirectory()){

                      d++;
                      String[] a = f.list();
                      int u = a.length;
                      r = (NodeFolder) t;

                      for(int i = 0; i<u; i++){
//                          if(r.children[i] != nnnn)
                            q.enqueue(r.children[i]);
                      }
                }
          }
          return d;
     }
```

When count is called it will call two method countfolder and countfile then it will return and print the number of each.

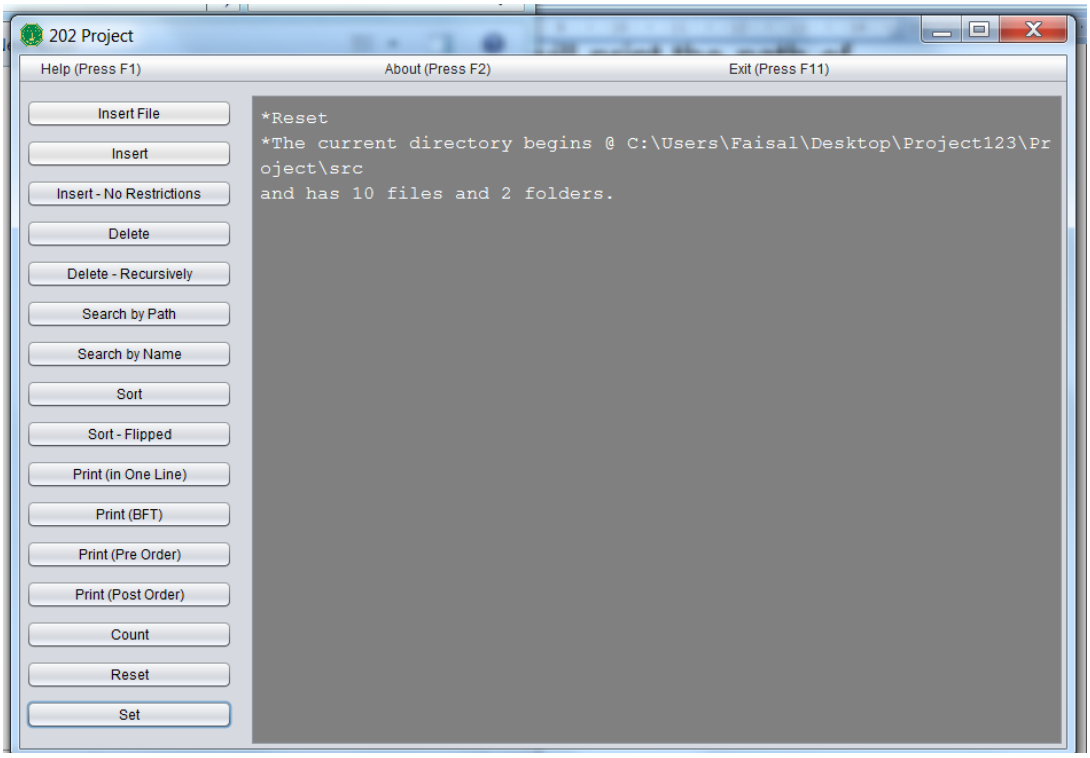
15- Reset:-

When the user select reset the output will delete everything in it the it will print the number of the file and folders and also it will print the path of the folder the user work in as shown:-



16- Set:-

When the user presses set a frame will appear asking the user to write the path of the new file wanted to be work in and then it will print the new file name in the output screen.


202 Project
Help (Press F1)
About (Press F2)
Exit (Press F11)
Insert File
Insert
Insert - No Restrictions
Delete
Delete - Recursively
Search by Path
Search by Name
Sort
Sort - Flipped
Print (in One Line)
Print (BFT)
Print (Pre Order)
Print (Post Order)
Count
Reset
Set
*Reset
*The current directory begins @ C:\Users\Faisal\Desktop\Project123\Project\src
and has 10 files and 2 folders.

3) Class description and code:

1- Node class:

The ancestor to NodeFile and NodeFolder.

Code:

```
public class Node{
      String name;
      String path;                //includes .type
      boolean vis = false;

      public Node(String x, String y){
            name = x;
            path = y;
      }
      public Node(String x){
            path = x;
            name = x.substring(x.lastIndexOf("\\")+1);
      }
   }
```

2- NodeFile class:

Code:

```
class NodeFile extends Node{

      public NodeFile(String x, String y){
            super(x,y);
      }
   }
```

3- NodeFolder class:

Code:

```
class NodeFolder extends Node{
      Node[] children;
      int files = -1;        //initlize
      int folders = -1;
      boolean pr = false;

      public NodeFolder(String name, String path, int m){
            super(name,path);
            children = new Node[m];
      }
      public NodeFolder(String path, int m){
            super(path);
            children = new Node[m];
      }
   }
```

4- Tree:

This class contains the main implantation of the tree. Also handles all the exceptions from all the program.

Code:

```
/*
 *
 * This is The Main tree class it has all the main methods and also handles
all the exceptions
 */

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;
import javax.swing.JOptionPane;

@SuppressWarnings("serial")
class GreaterThanMaxException extends Exception{
	public GreaterThanMaxException() { super(); }
	public GreaterThanMaxException(String message) { super(message); }
	public GreaterThanMaxException(String message, Throwable cause) {
super(message, cause); }
	public GreaterThanMaxException(Throwable cause) { super(cause); }
}
@SuppressWarnings("serial")
class DirectoryDoesNotExist extends Exception{		//used For Root Only
	public DirectoryDoesNotExist() { super(); }
	public DirectoryDoesNotExist(String message) { super(message); }
	public DirectoryDoesNotExist(String message, Throwable cause) {
super(message, cause); }
	public DirectoryDoesNotExist(Throwable cause) { super(cause); }
}

public class Tree {

	protected final static int MAX = Test.MAX;
	NodeFolder root, tmp;
	Window window;
	String hash;
```

```
	public Tree(String x) throws IOException{

		prn("Constructor -Tree");
		String y = x.substring(x.lastIndexOf("\\")+1);
		File f = new File(x);
		root = new NodeFolder(y, x, MAX);
		if(!f.exists())
			throw new FileNotFoundException("Folder does not exist");
		callScan();
		write();
		window = new Window(this);
	}
	protected boolean change(){
		try {
			write();
			return check();
		}catch (IOException e) {
			error(e);
		}
		return false;
	}
	private void write() throws IOException{
		String tmp = print();
		tmp = ""+tmp.hashCode();
		PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(Test.content, false)));
		out.println(tmp);
		out.close();
	}
	private boolean check() throws IOException{
		String n = null, m = null;
		BufferedReader br = new BufferedReader(new
FileReader(Test.content));
	   StringBuilder sb = new StringBuilder();
	   String line = br.readLine();

	   while (line != null) {
	       sb.append(line);
	       sb.append("\n");
	       line = br.readLine();
	   }
	   n = sb.toString();
	   br.close();
	   callScan();
	   m = ""+print().hashCode();
	   m = m.trim();
	   n = n.trim();
		return n.equals(m);
```

```
	}
	protected static void prn(String s){		//to minimise typing
		System.out.println(s);
	}
	protected void callScan(){
		try {
			scan();
		}catch(GreaterThanMaxException | DirectoryDoesNotExist |
IOException e) {
			error(e);
		}
	}
	private void scan()throws GreaterThanMaxException,
DirectoryDoesNotExist, IOException{
		File f = new File(root.path);
		String[] x = f.list();			//names of children
		int j = 0;
		Queue<Node> q = new Queue<Node>();
		File tmp;
		Node r;
		NodeFolder p;

		if(!f.exists())
			throw new DirectoryDoesNotExist();
		q.enqueue(root);

		while(! q.isEmpty()){

			r = q.dequeue();
			f = new File(r.path);
			x = f.list();

			if(x != null)
				if(x.length >= MAX)
					throw new GreaterThanMaxException("Directory has
greater than max");

			if(f.isDirectory()){
				p = (NodeFolder) r;
				for(j=0; j<x.length; j++){
					tmp = new File(p.path+"\\"+x[j]);
					if(tmp.isFile()){
						p.children[j] = new NodeFile(x[j],
p.path+"\\"+x[j]);
					}
					else if(tmp.isDirectory()){
						p.children[j] = new NodeFolder(x[j],
p.path+"\\"+x[j], MAX);
					}
```

```
                        q.enqueue(p.children[j]);
                    }
                }
            }
            write();
            count();
        }
        protected boolean setPath(String h) throws GreaterThanMaxException,
DirectoryDoesNotExist{
            File f = new File(h);
            if(!f.exists() || ! f.isDirectory())
                return false;
            root = new NodeFolder(h, MAX);
            callScan();
            return true;
        }

        protected String print(){
            return print(root);
        }
        protected String print(NodeFolder r){       //in one line
            Node f;
            Queue<Node> q = new Queue<Node>();
            String x = new String(">");

            q.enqueue(r);

            while(! q.isEmpty()){
                f = q.dequeue();

                if(new File(f.path).isFile()){
                    x+=f.name + " | ";
                }
                else if(new File(f.path).isDirectory()){
                    r = (NodeFolder) f;
                    x+=r.name + " | ";

                    for(int i = 0; i<new File(f.path).list().length; i++){
                        q.enqueue(r.children[i]);
                    }
                }
            }
            return x.substring(0, x.lastIndexOf('|'))+"#";
        }
        protected String printBFT(){
            return printBFT(root);
        }
        protected String printBFT(NodeFolder r){          //level by level
            Node f;
```

```
            Queue<Node> q = new Queue<Node>();
            String x = new String(">");
            q.enqueue(r);

            int scap = 0;
            int tmp = 0;
            String orig = root.path;
            char[] ch = new char[1000];
            ch = orig.toCharArray();

            for(char v: ch)                  //algorithm to decide when to
create a new line
                  if(v == '\\')
                        scap++;
            tmp = 0;

            while(! q.isEmpty()){
                  f = q.dequeue();

                  orig = f.path;
                  ch = orig.toCharArray();
                  tmp = scap;
                  scap = 0;
                  for(char v: ch){
                        if(v == '\\')
                              scap++;
                  }
                  if(scap > tmp){
                        x = x.substring(0, x.lastIndexOf(" |")) + " #\n>";
      //remove last '|' before line break
                  }

                  if(new File(f.path).isFile()){
                        x+=f.name + " | ";
                  }
                  else if(new File(f.path).isDirectory()){
                        r = (NodeFolder) f;

                        x+=r.name + " | ";
                        for(int i = 0; i<new File(f.path).list().length; i++){
                              q.enqueue(r.children[i]);
                        }
                  }
            }
            return x.substring(0, x.lastIndexOf(" |")) + " #";
      }
      protected String printPre(){
            return printPre(root);
      }
```

```
protected String printPre(NodeFolder r){          //level by level
	Node f;
	Stack<Node> s = new Stack<Node>();
	String x = new String(">");
	s.push(r);

	while(! s.isEmpty()){
		f = s.pop();
		if(new File(f.path).isFile()){
			x+=f.name + " | ";
		}
		else if(new File(f.path).isDirectory()){
			r = (NodeFolder) f;

			x+=r.name + " | ";
			for(int i = 0; i<new File(f.path).list().length; i++){
				s.push(r.children[i]);
			}
		}
	}
	return x.substring(0, x.lastIndexOf(" |")) + " #";
}
protected String printPost(){
	return printPost(root);
}
protected String printPost(NodeFolder r){         //level by level
	Node f;
	Stack<Node> s = new Stack<Node>();
	String x = new String(">");
	s.push(r);
	boolean k = false;
	while(! s.isEmpty()){
		f = s.pop();
		k = false;
		if(new File(f.path).isFile()){
			if(!f.vis){
				f.vis = true;
				x+=f.name + " | ";
			}
		}
		else if(new File(f.path).isDirectory()){
			r = (NodeFolder) f;

			if(new File(r.path).listFiles().length > 0 && !r.vis){
				k = true;
			}
			if(k){
				r.vis = true;
				s.push(r);
```

```
                    }
                    else
                        if(!r.pr){
                            r.pr = true;
                            x+=r.name + " | ";
                        }
                    for(int i = 0; i<new File(f.path).list().length; i++){
                        s.push(r.children[i]);
                    }

                }
            }
            return x.substring(0, x.lastIndexOf(" |")) + " #";
        }
        protected String count(){                    //includes folder
            root.files = countFiles(root);
            root.folders = countFolders(root);
            return root.files + " files and " + root.folders +" folders." ;
        }
        private int countFiles(NodeFolder r){
            Queue<Node> q = new Queue<Node>();
            Node t;
            int d=0;
            File f = new File(r.path);

            q.enqueue(r);
            while(!q.isEmpty()){
                t = q.dequeue();
                f = new File(t.path);

                if(f.isFile()){
                    d++;
                }
                else if(f.isDirectory()){
                    String[] a = f.list();
                    int u = a.length;
                    r = (NodeFolder) t;

                    for(int i = 0; i<u; i++){
//                      if(r.children[i] != nnnn)
                        q.enqueue(r.children[i]);
                    }
                }
            }
            return d;
        }
        private int countFolders(NodeFolder r){
            Queue<Node> q = new Queue<Node>();
            Node t;
```

```
        int d=0;
        File f = new File(r.path);

        q.enqueue(r);
        while(!q.isEmpty()){
              t = q.dequeue();
              f = new File(t.path);

              if(f.isFile()){
                    continue;
              }
              else if(f.isDirectory()){

                    d++;
                    String[] a = f.list();
                    int u = a.length;
                    r = (NodeFolder) t;

                    for(int i = 0; i<u; i++){
                          q.enqueue(r.children[i]);
                    }
              }
        }
        return d;
  }
  protected String getParentPath(Node p){                  //never used
        String x =  p.path.substring(0, p.path.lastIndexOf("\\"));
        if(x.contains(root.path)){
              return p.path.substring(0, p.path.lastIndexOf("\\"));
        }
        return null;
  }
  protected String searchPath(String h){             //works w & w/o .type
        String search = h;
        boolean found = false;
        Queue<Node> q = new Queue<Node>();
        Node r;
        NodeFolder p;
        q.enqueue(root);

        while(!q.isEmpty() && !found){
              r = q.dequeue();
              File f = new File(r.path);
              if(f.exists()){
                    if(f.isDirectory()){
                          p = (NodeFolder) r;
                          for(Node nd : p.children)
                                q.enqueue(nd);
                    }
```

```
                        if(r.name.contains("."))
                              if(r.path.substring(0,
r.path.lastIndexOf('.')).equals(search)){
                                    found = true;
                                    search = r.path;
                              }
                        if(r.path.equals(search)){
                              found = true;
                              search = r.path;
                        }
                  }
            }
            if(found)
                  return search;
            return null;
      }
      protected String searchName(String h){
            String search = h;          //name w & w/o .type
            boolean found = false;
            Queue<Node> q = new Queue<Node>();
            Node r;
            NodeFolder p;
            q.enqueue(root);

            while(! q.isEmpty()){
                  r = q.dequeue();
                  File f = new File(r.path);
                  if(f.exists()){
                        if(f.isDirectory()){
                              p = (NodeFolder) r;
                              for(Node nd : p.children)
                                    q.enqueue(nd);

                        }
                        if(r.name.contains("."))
                              if(r.name.substring(0,
r.name.lastIndexOf('.')).equals(search)){
                                    found = true;
                                    search = r.path;
                              }

                        if(r.name.equals(search)){
                              found = true;
                              search = r.path;
                        }
                  }
            }
            if(found)
                  return search;
```

```
            return null;
      }
      protected boolean insert(String p, String c)throws
GreaterThanMaxException, DirectoryDoesNotExist{        //restricted
            if(searchPath(p) == null){                  //parent does not
exist
                  return false;
            }
            else if(!c.contains(p)){          //child unrelated to parent
                  return false;
            }
            else if(searchPath(c) != null && new File(c).isDirectory()){
      //child exists
                  return false;
            }
            boolean n = new File(c).mkdir();
            callScan();
            return n;
      }
      protected boolean insertDirect(String c)throws GreaterThanMaxException,
DirectoryDoesNotExist{            //no restrictions
            if(c.length() < 1 || c.charAt(1) != ':')
                  return false;
            if(!c.contains(root.path))
                  return false;
            boolean n = new File(c).mkdirs();
            callScan();
            return n;
      }
      /*
       * if the file exists it will append it
       */
      protected int insertFile(String h, String hj)throws
GreaterThanMaxException, IOException, FileNotFoundException,
DirectoryDoesNotExist{
            int x = 0;
            if(!h.contains("\\")){            //path is wrong (to avoid
exceptions)
                  return 0;
            }
            else if(searchPath(h.substring(0,h.lastIndexOf("\\"))) == null){
      //direct parent does not exist
                  return 0;
            }
            else if(searchPath(h) != null){        //direct parent does not
exist
                  x = 1;
            }
            File f = new File(h);
```

```
            if(!f.exists())
                  f.createNewFile();
            PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(h, true)));
            out.println(hj);
            out.close();
            callScan();
            if(x == 0)
                  return 2;
            else
                  return x;
      }
      protected boolean delete(String h)throws GreaterThanMaxException,
DirectoryDoesNotExist{
            if(searchName(h) == null && searchPath(h) == null)
                  return false;
            boolean n = new File(h).delete();
            callScan();
            return n;
      }
      protected boolean deleteRec(String s)throws GreaterThanMaxException,
DirectoryDoesNotExist{
            if(s.length() < 1 || s.charAt(1) != ':')
                  return false;
            if(!s.contains(root.path))
                  return false;

            File f = new File(s);
            if (f.isDirectory()) {
                  for (File c : f.listFiles())
                        deleteRec(c.getPath());
            }
            if (!f.delete()){
                  callScan();
                  return false;
            }
            callScan();
            return true;
      }
      protected void sort(){
            Queue<Node> q = new Queue<Node>();
            Node r;
            NodeFolder p;
            q.enqueue(root);

            while(!q.isEmpty()){
                  r = q.dequeue();
                  if(new File(r.path).exists() && new
File(r.path).isDirectory()){
```

```
                    int x = 0;
                    if(new File(r.path).list() != null)
                          x = new File(r.path).list().length;
                    p = (NodeFolder) r;

                    String[] array = new String[x];
                    Node[] tmp = p.children.clone();

                    for(int i = 0; i<x; i++){
                          array[i] = p.children[i].name;
                    }
                    Arrays.sort(array);
                    for(int i = 0; i<x; i++){
                          for(Node nd : tmp)
                                if(nd != null)
                                      if(nd.name .equals(array[i])){
                                            p.children[i] = nd;
                                      }
                    }
                    for(Node n: p.children)
                          q.enqueue(n);
              }
        }
    }
    protected void sortFlip(){
          Queue<Node> q = new Queue<Node>();
          Node r;
          NodeFolder p;
          q.enqueue(root);

          while(!q.isEmpty()){
                r = q.dequeue();
                if(new File(r.path).exists() && new
File(r.path).isDirectory()){          //if folder
                    int x = 0;
                    if(new File(r.path).list() != null)
                          x = new File(r.path).list().length;
                    p = (NodeFolder) r;

                    String[] array = new String[x];
                    Node[] tmp = p.children.clone();

                    for(int i = 0; i<x; i++){
                          array[i] = p.children[i].name;
                    }
                    Arrays.sort(array);

                    for(int i = 0; i < array.length/2; i++){          //flip
algorithm
```

```
                        String temp = array[i];
                        array[i] = array[array.length - i - 1];
                        array[array.length - i - 1] = temp;
                    }
                    for(int i = 0; i<x; i++){
                        for(Node nd : tmp)
                            if(nd != null)
                                if(nd.name .equals(array[i])){
                                    p.children[i] = nd;
                                }
                    }
                    for(Node n: p.children)
                        q.enqueue(n);
                }
            }
        }
        @SuppressWarnings("static-access")
        protected static void error(Exception e){        //Exception Handling
            prn("Test. error");
            if(e instanceof GreaterThanMaxException)
                new JOptionPane().showMessageDialog(null, "You have
Exceeded the limit of " + Tree.MAX + " Objects in a directory\n Program will
exit", "Error", JOptionPane.INFORMATION_MESSAGE);

            else if(e instanceof DirectoryDoesNotExist)
                new JOptionPane().showMessageDialog(null, "The Main
Directory Does Not Exist\n Program will exit", "Error",
JOptionPane.INFORMATION_MESSAGE);
            else{
                new JOptionPane().showMessageDialog(null, "UnKnown error\n"
+ e.getMessage(), "Error", JOptionPane.INFORMATION_MESSAGE);
            }
            e.printStackTrace();
            System.exit(0);
        }

}
```

5- Window:-

Main GUI.

Code:

```
/*
 *
 * This is the main GUI
 */

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import javax.swing.JTextArea;
import javax.swing.UIManager;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JScrollPane;
import javax.swing.UnsupportedLookAndFeelException;

import java.awt.AWTEvent;
import java.awt.Font;
import java.awt.Color;
import java.awt.event.AWTEventListener;          //for keybindings
import java.awt.event.KeyEvent;

@SuppressWarnings("serial")
public class Window extends JFrame implements AWTEventListener{

	private JPanel contentPane;
	private JTextArea textField;
	private JMenuItem exit,	help, about;
	private JMenuBar bar;
	private JScrollPane pn;
	private JButton btnInsert, btnInsert2,	btnDelete, btndeleteRec,
	btnSearch,
				button,		btnSort,	btnSort2,	btnPrint,
	btnPrintBFT,
				btnCount,	btnReset,	set,		btnInsertFile,
btnPrintPre,
				btnPrintPost;

	private Tree t = null;
	protected static Listener l;

	public Window(Tree x) {
```

```
        setLook();            //set look and feel
        t = x;
        Tree.prn("Constructor -Window");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 930, 650);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        l = new Listener(t);

        this.getToolkit().addAWTEventListener(this,
AWTEvent.KEY_EVENT_MASK);        //key binding

        btnInsertFile = new JButton("Insert File");      //Window Three
        btnInsertFile.setBounds(5, 15, 180, 25);
        btnInsertFile.setBackground(Color.lightGray);
        contentPane.add(btnInsertFile);
        btnInsertFile.addActionListener(l);

        btnInsert = new JButton("Insert");               //Window One
        btnInsert.setBounds(5, 50, 180, 25);
        btnInsert.setBackground(Color.lightGray);
        contentPane.add(btnInsert);
        btnInsert.addActionListener(l);

        btnInsert2 = new JButton("Insert - No Restrictions");
        btnInsert2.setBounds(5, 85, 180, 25);
        contentPane.add(btnInsert2);
        btnInsert2.addActionListener(l);

        btnDelete = new JButton("Delete");
        btnDelete.setBounds(5, 120, 180, 25);
        contentPane.add(btnDelete);
        btnDelete.addActionListener(l);

        btndeleteRec = new JButton("Delete - Recursively");
        btndeleteRec.setBounds(5, 155, 180, 25);
        contentPane.add(btndeleteRec);
        btndeleteRec.addActionListener(l);

        btnSearch = new JButton("Search by Path");
        btnSearch.setBounds(5, 190, 180, 25);
        contentPane.add(btnSearch);
        btnSearch.addActionListener(l);

        button = new JButton("Search by Name");
```

```
button.setBounds(5, 225, 180, 25);
contentPane.add(button);
button.addActionListener(l);

btnSort = new JButton("Sort");
btnSort.setBounds(5, 260, 180, 25);
contentPane.add(btnSort);
btnSort.addActionListener(l);

btnSort2 = new JButton("Sort - Flipped");
btnSort2.setBounds(5, 295, 180, 25);
contentPane.add(btnSort2);
btnSort2.addActionListener(l);

btnPrint = new JButton("Print (in One Line)");
btnPrint.setBounds(5, 330, 180, 25);
contentPane.add(btnPrint);
btnPrint.addActionListener(l);

btnPrintPre = new JButton("Print (Pre Order)");
btnPrintPre.setBounds(5, 365, 180, 25);
contentPane.add(btnPrintPre);
btnPrintPre.addActionListener(l);

btnPrintPost = new JButton("Print (Post Order)");
btnPrintPost.setBounds(5, 400, 180, 25);
contentPane.add(btnPrintPost);
btnPrintPost.addActionListener(l);

btnPrintBFT = new JButton("Print (BFT)");
btnPrintBFT.setBounds(5, 435, 180, 25);
contentPane.add(btnPrintBFT);
btnPrintBFT.addActionListener(l);

btnCount = new JButton("Count");
btnCount.setBounds(5, 470, 180, 25);
contentPane.add(btnCount);
btnCount.addActionListener(l);

btnReset = new JButton("Reset");
btnReset.setBounds(5, 505, 180, 25);
contentPane.add(btnReset);
btnReset.addActionListener(l);

set = new JButton("Set");
set.setBounds(5, 540, 180, 25);
contentPane.add(set);
set.addActionListener(l);
```

```
            this.setBackground(Color.lightGray);

            textField = new JTextArea("*The current directory begins @ " +
t.root.path + "\nand has " + t.count()+"\n");
            textField.setFont(new Font("Monospaced", Font.PLAIN, 16));
            textField.setBackground(Color.gray);
            textField.setForeground(Color.WHITE);
            textField.setColumns(10);
            textField.setLineWrap(true);
            textField.setEditable(false);

            pn = new JScrollPane(textField);
            pn.setBounds(200, 10, 710, 570);

      pn.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER)
;

      pn.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED)
;
            contentPane.add(pn);

            bar = new JMenuBar();
            help = new JMenuItem("Help (Press F1)");
            help.addActionListener(l);
            bar.add(help);
            about = new JMenuItem("About (Press F2)");
            about.addActionListener(l);
            bar.add(about);
            exit = new JMenuItem("Exit (Press F11)");
            exit.addActionListener(l);bar.add(exit);

            setJMenuBar(bar);
            setTitle("202 Project");
            this.setIconImage(Test.image.getImage());

            setVisible(true);
      }
      protected void set(String i){
            textField.setText(i);
      }
      protected void append(String i){
            textField.append("*"+i+"\n");
      }
      protected void appendPlain(String i){
            textField.append(i+"\n");
      }
      private void setLook(){
            try {
```

```
      UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
            }catch (UnsupportedLookAndFeelException e) {
             setLook2();
         }
         catch (ClassNotFoundException e) {
            setLook2();
         }catch(Exception ex){
                  Tree.error(ex);
         }
      }
      private void setLook2(){
            try {
                  UIManager.setLookAndFeel(
UIManager.getSystemLookAndFeelClassName());
         }catch(Exception ex){
                  Tree.error(ex);
         }
      }
      protected void clickReset(){
            btnReset.doClick();
      }
      @Override
      public void eventDispatched(AWTEvent event) {
            if(event instanceof KeyEvent){
                  KeyEvent key = (KeyEvent)event;
                  int k = key.getKeyCode();
                  if(key.getID()==KeyEvent.KEY_PRESSED){
                        if(k == 112){
                              help.doClick();
                        }
                        if(k == 113){
                              about.doClick();
                        }
                        if(k == 122){
                              exit.doClick();
                        }
//                      key.consume();
             }
           }
         }
}
```

6- One:-

A frame with one field.

Code:

```
/*
 *
 * This frame will be called by the buttons that require one input such as
the Search buttons
 */

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

@SuppressWarnings("serial")
public class One extends JFrame{

	private JPanel contentPane;
	private JTextField textField;
	private JLabel lblTmp,	lblEnterPath,	lblOpreation;
	private JButton btnEnter;

	public One() {
		Tree.prn("Constructor -One");
		setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		setBounds(100, 100, 450, 250);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(null);

		lblEnterPath = new JLabel("Enter Path: ");
		lblEnterPath.setBounds(30, 55, 200, 20);
		contentPane.add(lblEnterPath);

		textField = new JTextField();
		textField.setBounds(30, 85, 360, 25);
		contentPane.add(textField);

		textField.setColumns(10);

		btnEnter = new JButton("Enter");
		btnEnter.setBounds(156, 150, 97, 25);
```

```
            contentPane.add(btnEnter);
            btnEnter.setActionCommand("E1");
            btnEnter.addActionListener(Window.l);

            lblOpreation = new JLabel("Opreation:");
            lblOpreation.setBounds(105, 15, 80, 20);
            contentPane.add(lblOpreation);

            lblTmp = new JLabel("tmp");
            lblTmp.setBounds(170, 15, 140, 20);
            contentPane.add(lblTmp);

            setTitle("tmp");
            this.setIconImage(Test.image2.getImage());

            setVisible(false);
      }
      protected void clear(){
            textField.setText("");
      }
      protected String get(){
            return textField.getText();
      }
      protected void set(String s){
            lblTmp.setText(s);
      }
      protected void set2(String s){
            lblEnterPath.setText(s);
      }
}
```

7- Two:-

A frame with two fields.

Code:

```
/*
 *
 * This frame will only be called by the insert button, it has no other
functions :(
 */

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;

@SuppressWarnings("serial")
public class Two extends JFrame{

	private JPanel contentPane;
	private JTextField textField,	textField_1;
	private JButton btnSubmit;
	private JLabel lblInsertChildsPath,	lblInsertParentsPath;

	public Two() {
		Tree.prn("Constructor -Two");
		setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		setBounds(100, 100, 185+120+40, 300);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(null);

		lblInsertParentsPath = new JLabel("Enter Parents Path: ");
		lblInsertParentsPath.setBounds(30, 20, 120, 25);
		contentPane.add(lblInsertParentsPath);

		lblInsertChildsPath = new JLabel("Enter Child's path: ");
		lblInsertChildsPath.setBounds(30, 110, 120, 25);
		contentPane.add(lblInsertChildsPath);

		textField = new JTextField("");
		textField.setBounds(15, 60, 290, 30);
```

```
            contentPane.add(textField);
            textField.setColumns(10);

            textField_1 = new JTextField("");
            textField_1.setBounds(15, 150, 290, 30);
            contentPane.add(textField_1);
            textField_1.setColumns(10);

            btnSubmit = new JButton("Submit");
            btnSubmit.setBounds(110, 200, 100, 25);
            contentPane.add(btnSubmit);
            btnSubmit.setActionCommand("E2");
            btnSubmit.addActionListener(Window.l);

            setTitle("tmp");
            this.setIconImage(Test.image3.getImage());

            setVisible(false);
      }
      protected void clear(){
            textField.setText("");
            textField_1.setText("");
      }
      protected String get1(){
            return textField.getText();
      }
      protected String get2(){
            return textField_1.getText();
      }
}
```

8- Three:-

A frame with one field and one textarea.

Code:

```
/*
 *
 * This frame will only be called by the insert file button, it has no other
functions :(
 */

import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JTextArea;

@SuppressWarnings("serial")
public class Three extends JFrame {

	private JPanel contentPane;
	private JTextField textField;
	private JLabel lblNewLabel1,	lblNewLabel_1;
	private JTextArea textArea;
	private JButton btnCreate;

	public Three() {
		Tree.prn("Constructor -Three");
		setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		setBounds(100, 100, 500, 350);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(null);

		lblNewLabel1 = new JLabel("Enter Path of file (please include
.type) :");
		lblNewLabel1.setBounds(10, 25, 370, 30);
		contentPane.add(lblNewLabel1);

		textField = new JTextField();
		textField.setBounds(12, 59, 408, 30);
		contentPane.add(textField);
		textField.setForeground(Color.gray);
```

```
            textField.setColumns(10);

            lblNewLabel_1 = new JLabel("Enter The Contents of the file:");
            lblNewLabel_1.setBounds(10, 95, 170, 30);
            contentPane.add(lblNewLabel_1);

            textArea = new JTextArea();
            textArea.setBounds(12, 128, 458, 126);
            textArea.setForeground(Color.gray);
            contentPane.add(textArea);

            btnCreate = new JButton("Enter");
            btnCreate.setBounds(195, 267, 97, 25);
            contentPane.add(btnCreate);
            btnCreate.setActionCommand("E3");
            btnCreate.addActionListener(Window.l);

            setTitle("tmp");
            this.setIconImage(Test.image3.getImage());

            setVisible(false);
      }
      protected void clear(){
            textField.setText("");
            textArea.setText("");
      }
      protected String get1(){
            return textField.getText();
      }
      protected String get2(){
            return textArea.getText();
      }
}
```

9) Listener:-

The main Listener in the package.

Code:

```
/*
 *
 * This class handles the events from all 3 GUIs, it also handles certain key
  strokes
 */

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JOptionPane;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

public class Listener implements ActionListener{
  String u1 = null, u2 = null, tmp = null,          tmp2 = null,    helpmsg = "This
  is Help",   aboutmsg = "This is About";
  Tree t = null;
  One o;
  Two w;
  Three th;

  private boolean n = false;              //submit
  private boolean b1 = false;             //Operations
  private boolean b2 = false;
  private boolean b3 = false;
  private boolean v = false;              //valid
  private boolean m = true;               //enter
  private boolean has = false;      //creating frames

  public Listener(Tree x) {
   t = x;

     try{
        BufferedReader br = new BufferedReader(new FileReader(Test.help));
         StringBuilder sb = new StringBuilder();
         String line = br.readLine();

         while (line != null) {
             sb.append(line);
             sb.append("\n");
             line = br.readLine();
         }
```

```
            helpmsg = sb.toString();
            br.close();

            br = new BufferedReader(new FileReader(Test.about));
            sb = new StringBuilder();
            line = br.readLine();

            while (line != null) {
                sb.append(line);
                sb.append("\n");
                line = br.readLine();
            }
            aboutmsg = sb.toString();
            br.close();
     }catch(Exception r){
           Tree.prn("Catch Buffer " + helpmsg);
           Tree.error(r);
     }
    }
    private void create(){
     if(!has){
           th = new Three();
           o = new One();
           w = new Two();
           has = true;
     }
    }
    private void closeOne(){
     o.clear(); o.setVisible(false);  o.dispose();
    }
    private void closeTwo(){
     w.clear(); w.setVisible(false);  w.dispose();
    }
    private void closeThree(){
     th.clear();      th.setVisible(false); th.dispose();
    }

    @SuppressWarnings("static-access")
    @Override
    public void actionPerformed(ActionEvent ae) {

     if(!t.change()){
//         Tree.prn("Change");
           b1 = false;b2 = false;b3 = false; v = false;
           closeOne();
           closeTwo();
           closeThree();
           t.callScan();
```

```
		new JOptionPane().showMessageDialog(null, "Directory was changed outside of
this program's environment, program will reset", "Error",
JOptionPane.INFORMATION_MESSAGE);
		t.window.set("*Reset\n" + "*The current directory begins @ " + t.root.path
+ "\nand has " + t.count()+"\n");
		return;
	}
	String x = ae.getActionCommand();

	create();
	if(new File(t.root.path) == null){
		new JOptionPane().showMessageDialog(null, "You have Exceeded the limit of "
+ Tree.MAX + " Objects in a directory\n Program will exit", "Error",
JOptionPane.INFORMATION_MESSAGE);
		Tree.error(new NullPointerException());
	}

	if(x.equals("Insert")){
		closeOne();
		closeThree();
		w.setTitle("Insert");
		w.setVisible(true);
	}
	else if(x.equals("E2")){
		try{
			u1 = w.get1();
			u2 = w.get2();
			if(u1 != null || u2 != null){
				n = t.insert(u1, u2);
				if(n)
					t.window.append("Insertion Successful @" + u2);
				else
					t.window.append("Insertion Failed @ " + u2);
			}
			else
			w.clear(); w.setVisible(false);  w.dispose();

		}catch(Exception r){
			Tree.error(r);
		}
	}
	/*
	 * truth table:
	 * b3b2b1
	 * 000		--	Delete: boolean
	 * 001		--	Path: boolean
	 * 010		--	Name: String
	 * 011		--	Delete - Recursively: boolean
	 * 100		--	Set: boolean
```

```
 * 111     --  Insert - no rest: boolean
 */
else if(x.equals("Delete")){
      b1 = false;      b2 = false;
      b3 = false; v = true;
      o.setTitle("Delete");
      o.set("Delete");
      o.set2("Enter Path: ");
      closeTwo();
      closeThree();
      o.setVisible(true);
}
else if(x.equals("Search by Path")){
      b1 = true; b2 = false;
      b3 = false; v = true;
      o.setTitle("Search by Path");
      o.set("Search by Path");
      o.set2("Enter Path: ");
      closeTwo();
      closeThree();
      o.setVisible(true);
}
else if(x.equals("Search by Name")){
      b1 = false;      b2 = true;
      b3 = false; v = true;
      o.setTitle("Search by Name");
      o.set("Search by Name");
      o.set2("Enter Name of File or Directory: ");
      closeTwo();
      closeThree();
      o.setVisible(true);
}
else if(x.equals("Delete - Recursively")){
      b1 = true; b2 = true;
      b3 = false; v = true;
      o.setTitle("Delete - Recursively");
      o.set("Delete - Recursively");
      o.set2("Enter Path: ");
      closeTwo();
      closeThree();
      o.setVisible(true);
}
else if(x.equals("Set")){
      b1 = false;      b2 = false;
      b3 = true; v = true;
      o.setTitle("Set a New Path");
      o.set("Set New Path: ");
      o.set2("Enter Path: ");
      closeTwo();
```

```
		closeThree();
		o.setVisible(true);
	}
	else if(x.equals("Set")){
		b1 = false;		b2 = false;
		b3 = true; v = true;
		o.setTitle("Set a New Path");
		o.set("Set New Path: ");
		o.set2("Enter Path: ");
		closeTwo();
		closeThree();
		o.setVisible(true);
	}
	else if(x.equals("Insert - No Restrictions")){
		b1 = true; b2 = true;
		b3 = true; v = true;
		o.setTitle("Insert - No Restrictions");
		o.set("Insert - No Restrictions");
		o.set2("Enter Path: ");
		closeTwo();
		closeThree();
		o.setVisible(true);
	}
	else if(x.equals("E1")){
		try{
			tmp = o.get();
			if(!b3 && !b2 && !b1 && v){		//delete
				m = t.delete(tmp);
				if(m)
					t.window.append("Delete Succeful @ " + tmp);
				else
					t.window.append("Delete failed @ " + tmp);
			}
			else if(!b3 && !b2 && b1 && v){  //Search - path
				String mt = t.searchPath(tmp);
				if(mt != null)
					t.window.append("Found " + mt);
				else
					t.window.append("Have Not Found " + tmp);
			}
			else if(!b3 && b2 && !b1 && v){  //Search - name
				tmp2 = t.searchName(tmp);
				if(tmp2 == null)
					t.window.append("Have Not Found " + tmp);
				else
					t.window.append("Found " + tmp + " @ " + tmp2);
			}
			else if(b2 && b1 && v && !b3){		//delete -r
				m = t.deleteRec(tmp);
```

```
                    if(m)
                          t.window.append("Recursive Deletion Succeful @ " + tmp);
                    else
                          t.window.append("Recursive Deletion failed @ " + tmp);
              }
              else if(b3 && !b2 && v && !b1){       //set
                    m = t.setPath(tmp);
                    if(m)
                          t.window.clickReset();
                    else
                          t.window.append("Recursive Deletion failed @ " + tmp);
              }
              else if(b2 && b1 && v && b3){         //insert -no rest
                    m = t.insertDirect(tmp);
                    if(m)
                          t.window.append("Insert - No Restrictions Succeful @ " +
tmp);
                    else
                          t.window.append("Insert - No Restrictions Failed @ " +
tmp);
              }

              o.clear(); o.setVisible(false);  o.dispose();
              w.clear(); w.setVisible(false);  w.dispose();
        }catch(Exception r){
              Tree.error(r);
        }
  }
  else if(x.equals("Sort")){
        b1 = false;b2 = false;      b3 = false; v = false;
        t.sort();
        closeOne();
        closeTwo();
        closeThree();
        t.window.append("The Tree Has been sorted in alphatecal order");
  }
  else if(x.equals("Sort - Flipped")){
        b1 = false;b2 = false;b3 = false; v = false;
        t.sortFlip();
        closeOne();
        closeTwo();
        closeThree();
        t.window.append("The Tree Has been sorted in reverse alphatecal order");
  }
  else if(x.equals("Print (in One Line)")){
        b1 = false;b2 = false;b3 = false; v = false;
        closeOne();
        closeTwo();
        closeThree();
```

```
		t.window.append("Print (In One Line)");
		t.window.appendPlain(t.print());
	}
	else if(x.equals("Print (BFT)")){
		b1 = false;b2 = false;b3 = false; v = false;
		closeOne();
		closeTwo();
		closeThree();
		t.window.append("Print (Breadth First Traversal):");
		t.window.appendPlain(t.printBFT());
	}
	else if(x.equals("Print (Pre Order)")){
		b1 = false;b2 = false;b3 = false; v = false;
		closeOne();
		closeTwo();
		closeThree();
		t.window.append("Print (PreOrder):");
		t.window.appendPlain(t.printPre());
	}
	else if(x.equals("Print (Post Order)")){
		b1 = false;b2 = false;b3 = false; v = false;
		closeOne();
		closeTwo();
		closeThree();
		t.window.append("Print (Post Order):");
		t.window.appendPlain(t.printPost());
	}
	else if(x.equals("Count")){
		b1 = false;b2 = false;b3 = false; v = false;
		closeOne();
		closeTwo();
		closeThree();
		t.window.append("The current directory begins @ " + t.root.path + "\nand
has " + t.count());
	}
	else if(x.equals("Reset")){
		b1 = false;b2 = false;b3 = false; v = false;
		closeOne();
		closeTwo();
		closeThree();

		t.callScan();
		t.window.set("*Reset\n" + "*The current directory begins @ " + t.root.path
+ "\nand has " + t.count()+"\n");
	}
	else if(x.equals("Insert File")){
		b1 = false;b2 = false;b3 = false; v = false;
		closeOne();
		closeTwo();
```

```
            th.setVisible(true);
            th.setTitle("Insert File");
        }
        else if(x.equals("E3")){
            b1 = false;b2 = false;b3 = false; v = false;
            String hj = th.get1();
            String jk = th.get2();
            int k = 0;
            try{
                  k = t.insertFile(hj, jk);
                  t.callScan();
            }catch (Exception e) {
                  Tree.error(e);
            }
            if(k == 2)
                  t.window.append("File Insertion Succeful @ " + hj);
            else if(k == 1)
                  t.window.append("File Appended Succefully @ " + hj);
            else
                  t.window.append("File Insertion Failed @ " + hj);
            closeOne();
            closeTwo();
            closeThree();
        }
        else if(x.equals("Help (Press F1)")){
            JOptionPane.showMessageDialog(null, helpmsg, "Help",
      JOptionPane.INFORMATION_MESSAGE);

        }
        else if(x.equals("About (Press F2)")){
            JOptionPane.showMessageDialog(null, aboutmsg, "Help",
      JOptionPane.INFORMATION_MESSAGE);
        }
        else if(x.equals("Exit (Press F11)")){
            Tree.prn("Exit");
            System.exit(0);
        }
      }
}
```

10- Stack:-

Basic implementation only change is in the push method.

Code:

```
public class Stack<E>{
    private java.util.ArrayList<E> pool = new java.util.ArrayList<E>();
    public Stack() {
    }
    public Stack(int n) {
        pool.ensureCapacity(n);
    }
    public void clear() {
        pool.clear();
    }
    public boolean isEmpty() {
        return pool.isEmpty();
    }
    public E topEl() {
        if (isEmpty())
            throw new java.util.EmptyStackException();
        return pool.get(pool.size()-1);
    }
    public E pop() {
        if (isEmpty())
            throw new java.util.EmptyStackException();
        return pool.remove(pool.size()-1);
    }
    public void push(E el) {
      if(el == null)
           return;
        pool.add(el);
    }
    public String toString() {
        return pool.toString();
    }
}
```

11- Queue:-

Basic implementation only change is in the enqueue method.

Code:

```
public class Queue<T>{
    private java.util.LinkedList<T> list = new java.util.LinkedList<T>();

    public Queue(){
    }
    public void clear(){
        list.clear();
    }
    public boolean isEmpty(){
        return list.isEmpty();
    }
    public T firstEl(){
        return list.getFirst();
    }
    public T dequeue(){
        return list.removeFirst();
    }
    public void enqueue(T el){
      if(el == null)              //modified here for easiness
            return;
      list.add(el);
    }
    public String toString(){
        return list.toString();
    }
   }
```

12- Test:-

A tester class, has the paths to icons and .txt files that the program will use.

Code:

```
/*
 *
 * Tester class
 */

import javax.swing.ImageIcon;

public class Test {

	protected final static String about =
"D:\\Academic\\Project\\about.txt",
                                     help =
"D:\\Academic\\Project\\help.txt",
                                     path =
"C:\\Users\\sal7\\Desktop\\test",
                                     content =
"D:\\Academic\\Project\\content.txt";
	protected final static ImageIcon image = new
ImageIcon("D:\\Academic\\Project\\icon.png"),
                                          image2 = new
ImageIcon("D:\\Academic\\Project\\icon2.jpg"),
                                          image3 = new
ImageIcon("D:\\Academic\\Project\\icon3.png");
	protected final static int MAX = 50;

	@SuppressWarnings("unused")
	public static void main(String[] args){
		try{
			Tree y = new Tree(path);
		}
		catch(Exception m){
			Tree.prn("Catch " + m.getMessage());
			Tree.error(m);
		}
	}
}
```

4) Exception Classes code:

1- GreaterThanMaxException:-

A tester class, has the paths to icons and .txt files that the program will use.

Code:

```
@SuppressWarnings("serial")
class GreaterThanMaxException extends Exception{
      public GreaterThanMaxException() { super(); }
      public GreaterThanMaxException(String message) { super(message); }
      public GreaterThanMaxException(String message, Throwable cause) {
super(message, cause); }
      public GreaterThanMaxException(Throwable cause) { super(cause); }
   }
```

2- DirectoryDoesNotExist:-

A tester class, has the paths to icons and .txt files that the program will use.

Code:

```
@SuppressWarnings("serial")
class DirectoryDoesNotExist extends Exception{         //used For Root Only
      public DirectoryDoesNotExist() { super(); }
      public DirectoryDoesNotExist(String message) { super(message); }
      public DirectoryDoesNotExist(String message, Throwable cause) {
super(message, cause); }
      public DirectoryDoesNotExist(Throwable cause) { super(cause); }
       }
```