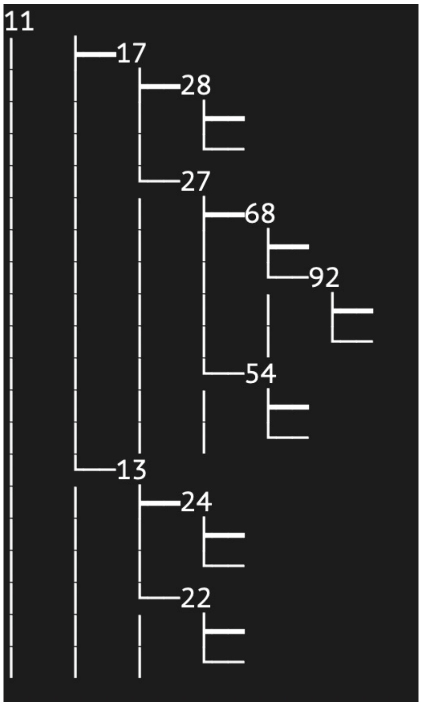
Lab 11

CSE 274

I. A Binary Tree

Import the `Application.java` file into Eclipse, run the application and make sure that you see the following output:



```
11
│   ├──17
│   │   ├──28
│   │   │   ├──
│   │   │   └──
│   │   └──27
│   │   │   ├──68
│   │   │   │   ├──
│   │   │   │   └──92
│   │   │   │   │   ├──
│   │   │   │   │   └──
│   │   │   └──54
│   │   │   │   ├──
│   │   │   │   └──
│   └──13
│   │   ├──24
│   │   │   ├──
│   │   │   └──
│   │   └──22
│   │   │   ├──
│   │   │   └──
```

II. The traversePreOrder method

To traverse a tree pre-order we call the `traversePreOrder` method of the `BinaryTree` class:

```
public void traversePreOrder() {
     RecursivetraversePreOrder(root);
     System.out.println();}
```

From above, the `traversePreOrder` method calls the `RecursivetraversePreOrder(root)` method to traverse the nodes of the tree **recursively** pre-order. Look into the body of this recursive method to see its logic. Use the following lines of code to test the `traversePreOrder` method:

```
BinaryTree myBinaryTree = new BinaryTree();
myBinaryTree.root=new Node(11);
myBinaryTree.root.up= new Node(17);
myBinaryTree.root.up.up= new Node(28);
myBinaryTree.root.up.down= new Node(27);
myBinaryTree.root.up.down.up= new Node(68);
myBinaryTree.root.up.down.up.down= new Node(92);
myBinaryTree.root.down= new Node(13);
myBinaryTree.root.down.up= new Node(24);
myBinaryTree.root.down.down= new Node(22);
myBinaryTree.root.up.down.down= new Node(54);
myBinaryTree.traversePreOrder();
```

The expected output is printed below:

```
11 13 22 24 17 27 54 68 92 28
```

III. THE traversePostOrder METHOD

To traverse a tree post-order we call the `traversePostOrder` method of the `BinaryTree` class:

```
public void traversePostOrder() {
     RecursivetraversePostOrder(root);
     System.out.println();}
```

From above, the `traversePostOrder` method calls the `RecursivetraversePostOrder(root)` method to traverse the nodes of the tree **recursively** post-order. Develop the following method for the `BinaryTree` class using a recursive logic:

```
public void RecursivetraversePostOrder(Node mynode)
```

Use the following lines of code to test the developed method:

```
BinaryTree myBinaryTree = new BinaryTree();
myBinaryTree.root=new Node(11);
myBinaryTree.root.up= new Node(17);
myBinaryTree.root.up.up= new Node(28);
myBinaryTree.root.up.down= new Node(27);
myBinaryTree.root.up.down.up= new Node(68);
myBinaryTree.root.up.down.up.down= new Node(92);
myBinaryTree.root.down= new Node(13);
myBinaryTree.root.down.up= new Node(24);
myBinaryTree.root.down.down= new Node(22);
myBinaryTree.root.up.down.down= new Node(54);
myBinaryTree.traversePostOrder();
```

The expected output is printed below:

```
22 24 13 54 92 68 27 28 17 11
```

IV. THE traverseInOrder METHOD

To traverse a tree in-order we call the `traverseInOrder` method of the `BinaryTree` class:

```
public void traverseInOrder() {
     RecursivetraverseInOrder(root);
     System.out.println();}
```

From above, the `traverseInOrder` method calls the `RecursivetraverseInOrder(root)` method to traverse the nodes of the tree **recursively** in-order. Develop the following method for the `BinaryTree` class using a recursive logic:

```
private void RecursivetraverseInOrder(Node mynode)
```

Use the following lines of code to test the developed method:

```
BinaryTree myBinaryTree = new BinaryTree();
myBinaryTree.root=new Node(11);
myBinaryTree.root.up= new Node(17);
myBinaryTree.root.up.up= new Node(28);
myBinaryTree.root.up.down= new Node(27);
myBinaryTree.root.up.down.up= new Node(68);
myBinaryTree.root.up.down.up.down= new Node(92);
myBinaryTree.root.down= new Node(13);
myBinaryTree.root.down.up= new Node(24);
myBinaryTree.root.down.down= new Node(22);
myBinaryTree.root.up.down.down= new Node(54);
myBinaryTree.traverseInOrder();
```

The expected output is printed below:

```
 22 13 24 11 54 27 92 68 17 28
```

V. The traverseLevelOrder Method

To traverse a tree level-order we call the `traverseLevelOrder` method of the `BinaryTree` class:

```
 public void traverseLevelOrder()
```

Develop the above method for the `BinaryTree` class using the following logic:

```
public void traverseLevelOrder()

      if (tree is empty)
            return

      Queue myQueue = new Queue()

      enQueue the root node of the tree in myQueue

      while (myQueue is not empty)

            Node mynode = myQueue.deQueue()

            System.out.print(" " + mynode.value)

            if (mynode has a down child)
                  enQueue the down child of mynode in myQueue

            if (mynode has an up child)
                  enQueue the up child of mynode in myQueue

System.out.println()
```

In developing the above code you can use the `Queue` class which is available in the `Application.java` file. The `Queue` class creates a queue that stores objects of `Node` class. Use the following lines of code to test the developed method:

```
BinaryTree myBinaryTree = new BinaryTree();
myBinaryTree.root=new Node(11);
myBinaryTree.root.up= new Node(17);
myBinaryTree.root.up.up= new Node(28);
myBinaryTree.root.up.down= new Node(27);
myBinaryTree.root.up.down.up= new Node(68);
myBinaryTree.root.up.down.up.down= new Node(92);
myBinaryTree.root.down= new Node(13);
myBinaryTree.root.down.up= new Node(24);
myBinaryTree.root.down.down= new Node(22);
myBinaryTree.root.up.down.down= new Node(54);
myBinaryTree.traverseLevelOrder();
```

The expected output is printed below:

```
 11 13 17 22 24 27 28 54 68 92
```

VI. Submitting the Assignment

When submitting your response to the assignment, keep the above lines of code in the body of the `main` method.