```
class BSTNode
{
   BSTNode left, right;
   int data;

   public BSTNode()
   {
      left = null;
      right = null;
      data = 0;
   }
   public BSTNode(int n).      {
      left = null;
     right = null;
      data = n;
   }
   public void setLeft(BSTNode n)
   {
      left = n;
   }
   public void setRight(BSTNode n)
   {
      right = n;
   }
   public BSTNode getLeft()
```

```
    {
        return left;
    }
    public BSTNode getRight()
    {
        return right;
    }
    public void setData(int d)
    {
        data = d;
    }
    public int getData()
    {
        return data;
    }
  }

  class BST
  {
    private BSTNode root;

    public BST()
{
        root = null;
    }
```

```
    public boolean isEmpty()
    {
        return root == null;
    }
    public void insert(int data)
{
      root = insert(root, data);
    }
    private BSTNode insert(BSTNode node, int data)
    {
        if (node == null)
            node = new BSTNode(data);
        else
        {
            if (data <= node.getData())
                node.left = insert(node.left, data);
            else
                node.right = insert(node.right, data);
        }
        return node;
    }
    public void delete(int k)
    {
        if (isEmpty())
            System.out.println("Tree Empty");
```

```
     else if (search(k) == false)
         System.out.println("Sorry "+ k +" is not present");
    else
    {
        root = delete(root, k);
        System.out.println(k+ " deleted from the tree");
    }
}
private BSTNode delete(BSTNode root, int k)
{
    BSTNode p, p2, n;
    if (root.getData() == k)
    {
        BSTNode lt, rt;
        lt = root.getLeft();
        rt = root.getRight();
        if (lt == null && rt == null)
            return null;
        else if (lt == null)
        {
            p = rt;
            return p;
        }
        else if (rt == null)
        {
```

```
            p = lt;
            return p;
        }
        else
        {
            p2 = rt;
            p = rt;
            while (p.getLeft() != null)
                p = p.getLeft();
            p.setLeft(lt);
            return p2;
        }
    }
    if (k < root.getData())
    {
        n = delete(root.getLeft(), k);
        root.setLeft(n);
    }
    else
    {
        n = delete(root.getRight(), k);
        root.setRight(n);
    }
    return root;
}
```

```
public int countNodes()
{
   return countNodes(root);
}
private int countNodes(BSTNode r)
{
   if (r == null)
      return 0;
   else
   {
      int l = 1;
      l += countNodes(r.getLeft());
      l += countNodes(r.getRight());
      return l;
   }
}
public boolean search(int val)
{
   return search(root, val);
}
private boolean search(BSTNode r, int val)
{
   boolean found = false;
   while ((r != null) && !found)
   {
```

```
        int rval = r.getData();
        if (val < rval)
            r = r.getLeft();
        else if (val > rval)
            r = r.getRight();
        else
        {
            found = true;
            break;
        }
        found = search(r, val);
    }
    return found;
}
public void inorder()
{
    inorder(root);
}
private void inorder(BSTNode r)
{
    if (r != null)
    {
        inorder(r.getLeft());
        System.out.print(r.getData() +" ");
        inorder(r.getRight());
```

```
    }
}
public void preorder()
{
    preorder(root);
}
private void preorder(BSTNode r)
{
    if (r != null)
    {
        System.out.print(r.getData() +" ");
        preorder(r.getLeft());
        preorder(r.getRight());
    }
}
public void postorder()
{
    postorder(root);
}
private void postorder(BSTNode r)
{
    if (r != null)
    {
        postorder(r.getLeft());
        postorder(r.getRight());
```

```
            System.out.print(r.getData() +" ");
        }
    }
}
```