```java
/*
 * Written by JJ Shepherd
 */
public class LinkedBST <T extends Comparable<T>>
{
    private class Node
    {
        T data;
        Node leftChild;
        Node rightChild;
        public Node(T aData)
        {
            data = aData;
            leftChild = rightChild = null;
        }
    }
    private Node root;//head
    public LinkedBST()
    {
        root = null;
    }
    public void add(T aData)
    {
        if(root == null)
            root = new Node(aData);
        else
            add(root,aData);
    }
    private Node add(Node aNode, T aData)
    {
        if(aNode == null)
            aNode = new Node(aData);
        else if(aData.compareTo(aNode.data)<0)//GO LEFT
            aNode.leftChild = add(aNode.leftChild,aData);
        else if(aData.compareTo(aNode.data)>0)//GO RIGHT
            aNode.rightChild = add(aNode.rightChild,aData);
        return aNode;
    }
    public void printPreorder()
    {
        printPreorder(root);
    }
    private void printPreorder(Node aNode)
    {
        if(aNode == null)
            return;
        System.out.println(aNode.data);//PROCESS
        printPreorder(aNode.leftChild);//LEFT
        printPreorder(aNode.rightChild);//RIGHT
    }
    public void printInorder()
    {
        printInorder(root);
    }
    private void printInorder(Node aNode)
    {
        if(aNode == null)
```

```
58              return;
59          printInorder(aNode.leftChild);//LEFT
60          System.out.println(aNode.data);//PROCESS
61          printInorder(aNode.rightChild);//RIGHT
62      }
63      public boolean search(T aData)
64      {
65          return search(root,aData);
66      }
67      private boolean search(Node aNode, T aData)
68      {
69          if(aNode == null)
70              return false;
71          else if(aData.compareTo(aNode.data)<0)//GO LEFT
72              return search(aNode.leftChild,aData);
73          else if(aData.compareTo(aNode.data)>0)//GO RIGHT
74              return search(aNode.rightChild,aData);
75          else
76              return true;
77      }
78      public void remove(T aData)
79      {
80          root = remove(root,aData);
81      }
82      private Node remove(Node aNode, T aData)
83      {
84          //Find the node
85          if(aNode == null)
86              return null;
87          else if(aData.compareTo(aNode.data)<0)
88              aNode.leftChild = remove(aNode.leftChild,aData);
89          else if(aData.compareTo(aNode.data)>0)
90              aNode.rightChild = remove(aNode.rightChild,aData);
91          else//FOUND IT!
92          {
93              if(aNode.rightChild == null)
94                  return aNode.leftChild;
95              else if(aNode.leftChild == null)
96                  return aNode.rightChild;
97              Node temp = findMinInTree(aNode.rightChild);
98              aNode.data = temp.data;
99              aNode.rightChild = remove(aNode.rightChild,temp.data);
100         }
101         return aNode;
102     }
103     private Node findMinInTree(Node aNode)
104     {
105         if(aNode == null)
106             return null;
107         else if(aNode.leftChild == null)
108             return aNode;
109         else
110             return findMinInTree(aNode.leftChild);
111     }
112 }
113
```