```
package ds;

public class BasicBinaryTree<X extends Comparable<X>>
{

    private Node root;
    private int size;

    public BasicBinaryTree(){
        this.root = null;
    }

    public int size(){
        return size;
    }

    public void add(X item){
        Node node = new Node(item);

        if(root == null){
            this.root = node;
            System.out.println("Set root: " + node.getItem());
            this.size++;
        } else {
            insert(this.root, node);
        }
    }

    public void insert(Node parent, Node child){

        if(child.getItem().compareTo(parent.getItem()) < 0){

            if(parent.getLeft() == null){
                parent.setLeft(child);
                child.setParent(parent);
                this.size++;
            } else {
                insert(parent.getLeft(), child);
```

```
            }
        } else if (child.getItem().compareTo(parent.getItem())
> 0){

            if(parent.getRight() == null){
                parent.setRight(child);
                child.setParent(parent);
                this.size++;
            } else {
                insert(parent.getRight(), child);
            }
        }
    }

    public boolean contains(X item){
        Node currentNode = getNode(item);

        if(currentNode != null)
            return true;
        else
            return false;
    }

    public boolean delete(X item){
    boolean deleted = false;

    if(this.root == null )
        return deleted;

    Node currentNode = getNode(item);

    if(currentNode != null){

        if(currentNode.getLeft() == null &&
currentNode.getRight() == null){
            unlink(currentNode, null);
            deleted = true;
        } else if(currentNode.getLeft() != null &&
```

```
currentNode.getRight() == null){
            unlink(currentNode, currentNode.getLeft());
            deleted = true;
        } else if(currentNode.getLeft() == null &&
currentNode.getRight() != null){
            unlink(currentNode, currentNode.getRight());
            deleted = true;
        } else {

            Node child = currentNode.getLeft();

            while(child.getRight() != null && child.getLeft()!=
null){
                child = child.getRight();
            }

            child.getParent().setRight(null);

            child.setLeft(currentNode.getLeft());
            child.setRight(currentNode.getRight());

            unlink(currentNode, child);

            deleted = true;
        }

    }

    if(deleted == true)
        this.size--;

    return deleted;
    }

    //helper method to just swap the nodes
    public void unlink(Node currentNode, Node newNode){

        if(currentNode == this.root){
```

```
            newNode.setRight(currentNode.getRight());
            newNode.setLeft(currentNode.getLeft());
            this.root = newNode;
        } else if(currentNode.getParent().getRight() ==
currentNode){
            currentNode.getParent().setRight(newNode);
        } else if(currentNode.getParent().getLeft() ==
currentNode){
            currentNode.getParent().setRight(newNode);
        }
    }

    public Node getNode(X item){
        Node currentNode = this.root;

        while(currentNode != null){
            int val = currentNode.getItem().compareTo(item);

            if(val == 0){
                return  currentNode;
            }
            else if(val < 0)
                currentNode = currentNode.getLeft();
            else if (val > 0)
                currentNode = currentNode.getRight();
        }

        return null;
    }

    private class Node {

        private Node left;
        private Node right;
        private Node parent;
        private X item;

        public Node(X item){
```

```
    this.item = item;
    this.left = null;
    this.right = null;
    this.parent = null;
}

public Node getLeft() {
    return left;
}

public void setLeft(Node left) {
    this.left = left;
}

public Node getRight() {
    return right;
}

public void setRight(Node right) {
    this.right = right;
}

public Node getParent() {
    return parent;
}

public void setParent(Node parent) {
    this.parent = parent;
}

public X getItem() {
    return item;
}

public void setItem(X item) {
    this.item = item;
}
```

```
    }
}
```

To test the binary tree datastructure :

```
public class ContactManagerApp {
    BasicBinaryTree<Contact> contacts = new
BasicBinaryTree<Contact>();

    public static void main(String[] args) {
        ContactManagerApp app = new
ContactManagerApp();
        app.loadContacts();

        //See if we have any of the following contacts
        app.searchContacts();

        //delete some contacts
        app.cleanupContacts();
    }

    public void loadContacts() {
        contacts.add(new Contact("Abe", "Lincoln",
"123-555-5555"));
        contacts.add(new Contact("Sheree", "Whisman",
"123-123-4567"));
        contacts.add(new Contact("Doreatha", "Crumbley",
"123-123-4568"));
        contacts.add(new Contact("Mitchel", "Wear",
"123-123-4567"));
        contacts.add(new Contact("Lisabeth", "Espiritu",
"123-123-4569"));
        contacts.add(new Contact("Cora", "Bonhomme",
"123-123-4567"));
        contacts.add(new Contact("Brigette", "Aikins",
"123-123-3567"));
        contacts.add(new Contact("Julieann", "Kellett",
"123-123-4367"));
```

```
        contacts.add(new Contact("Floy", "Collin",
"123-123-4537"));
        contacts.add(new Contact("Abel", "Rocco",
"123-123-4563"));
        contacts.add(new Contact("Karen", "Presler",
"123-223-4567"));
        contacts.add(new Contact("Maryjane",
"Archambault", "123-123-1234"));
        contacts.add(new Contact("Afton", "Tadlock",
"123-123-2223"));
        contacts.add(new Contact("Ines", "Ludlow",
"123-123-2224"));
        contacts.add(new Contact("Cristen", "Skillern",
"123-123-3334"));
        contacts.add(new Contact("Porsche", "Gadsden",
"123-123-4444"));
        contacts.add(new Contact("Sharee", "Glazer",
"123-123-3335"));
        contacts.add(new Contact("Ashly", "Absher",
"123-123-4443"));
        contacts.add(new Contact("Rebekah", "Eide",
"123-123-3336"));
        contacts.add(new Contact("Dian", "Goheen",
"123-123-4555"));
        contacts.add(new Contact("Velda", "Mitchem",
"123-123-5554"));
        contacts.add(new Contact("Fay", "Moro",
"123-123-4556"));
        contacts.add(new Contact("Claudette", "Damewood",
"123-123-6664"));
        contacts.add(new Contact("Pei", "Rogan",
"123-123-6545"));
        contacts.add(new Contact("Sandie", "Mcmillion",
"123-123-5557"));
        contacts.add(new Contact("Gemma", "Uy",
"123-123-7776"));
        contacts.add(new Contact("Annetta", "Coale",
"123-123-8899"));
```

```
        contacts.add(new Contact("Ona", "Hynes",
"123-123-9876"));
        contacts.add(new Contact("Daryl", "Clukey",
"123-123-4721"));
        contacts.add(new Contact("Meghann", "Wischmeier",
"123-123-4321"));
        System.out.println("Loaded " + contacts.size() + "
contacts");
    }

    public void searchContacts() {
        // We don't need to populate phone since we only
compare on last name
        System.out.println(contacts.contains(new
Contact("Fay", "Moro", null))); // should be true
        System.out.println(contacts.contains(new
Contact("Bob", "Hope", null))); // should be false
    }

    public void cleanupContacts() {
        // we just need to remove by last name since that's all
we check in the comparable
        contacts.delete(new Contact("Pei", "Rogan", null));
//delete a leaf node
        contacts.delete(new Contact("Porsche", "Gadsden",
null)); //delete a node with a right node only
        contacts.delete(new Contact("Annetta", "Coale",
null)); //delete a node with a left node only
        contacts.delete(new Contact("Cora", "Bonhomme",
null)); //delete a node with both children
        contacts.delete(new Contact("Abe", "Lincoln", null));
//delete the root node
        System.out.println(contacts.size() + " contacts
remaining");
    }

    class Contact implements Comparable<Contact> {
        private String firstName;
```

```
    private String lastName;
    private String phone;

    public Contact(String firstName, String lastName,
String phone) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.phone = phone;
    }

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public int compareTo(Contact other) {
        // For simplicity, we're only going to sort on the
contact's last name
        return this.lastName.compareTo(other.lastName);
    }

    @Override
    public String toString() {
```

```
            return "Contact [firstName=" + firstName + ",
lastName=" + lastName + ", phone=" + phone + "]";
        }
    }
}
```