1. **PHBInsert(String name, String phoneNumber) Method: It has a worst case running time complexity of O(n), where n is the total number of nodes in the tree since it must traverse all nodes inorder to returns True if the name and number are already added and False if the phone number is already exist..**

    - It takes input name and phone#
    - It inserts persons in both BSTs(BSTPhoneNo and BSTName). If it is inserted in both BSTs, it returns true otherwise it returns false
    - This is a snapshot of my inputs below, name and its associated phone number. I added Rosy, Alie, Isha(3 times), Amee, Jane, and Tom, and then lastly Izzy with the same number as Rosy. Since that number is already there it did not insert Izzy, and returned false.

```
DoubleTreePhBook dblTreePhBook = new DoubleTreePhBook();
//inserts the value in both the BSTName & BST Phone# Trees
dblTreePhBook.PhbBInsert("Rosy", "2134567890");
dblTreePhBook.PhbBInsert("Alie", "8606313009");
dblTreePhBook.PhbBInsert("Isha", "8608302732");
dblTreePhBook.PhbBInsert("Isha", "9123456789");
dblTreePhBook.PhbBInsert("Amee",  "1111113467");
dblTreePhBook.PhbBInsert("Jane", "0000000030");
dblTreePhBook.PhbBInsert("Isha", "4567892341");
dblTreePhBook.PhbBInsert("Tom", "9123456789");
boolean insertTesting = dblTreePhBook.PhbBInsert("Izzy", "2134567890");
System.out.println( "If the phone# is already there, it will return false since it can't add to Double Tree Book: " + insertTesting);
```

I also added three Isha's with the same name but with different numbers, and since the same name is allowed it inserts multiple Isha's with different numbers. As seen by the output below: I also tested both the BST Phone Number and Name Trees.

```
Prints the inserted persons using BST Phone Number Tree (using phone# as the key):
Name: Jane, Phone Number: 0000000030
Name: Amee, Phone Number: 1111113467
Name: Rosy, Phone Number: 2134567890
Name: Isha, Phone Number: 4567892341
Name: Alie, Phone Number: 8606313009
Name: Isha, Phone Number: 8608302732
Name: Isha, Phone Number: 9123456789
Prints the inserted person using BST Name Tree (using name as key):
Name: Alie, Phone Number: 8606313009
Name: Amee, Phone Number: 1111113467
Name: Isha, Phone Number: 8608302732
Name: Isha, Phone Number: 9123456789
Name: Isha, Phone Number: 4567892341
Name: Jane, Phone Number: 0000000030
Name: Rosy, Phone Number: 2134567890
```

2. **PHBDelete(String name, String phoneNumber) Method: It has a worst case running time complexity of O(n), where n is the total number of nodes in the tree since it must traverse all nodes.**

   - It takes input name and phone#
   - It deletes persons in both BSTs(BSTPhoneNo and BSTName). If it is deleted in both BSTs, it returns true otherwise it returns false.
   - I deleted Jane & Amee from the phone book, and tested using the following lines of code in the Test class.

```
System.out.println("Delete Testing for Jane with Phone# 0000000030 & Amee with Phone# 1111113467");
// this deletes the person Jane
dblTreePhBook.PhbBDelete("Jane", "0000000030");
// this deletes the person Amee
dblTreePhBook.PhbBDelete("Amee", "1111113467");
```

As seen in the output below, the new phonebook doesn't have Jane & Amee since I used the PhBDelete method to delete them.

```
Prints the phonebook without("Jane", "0000000030") (using phone# as the key & inorder tree traversal):
Name: Rosy, Phone Number: 2134567890
Name: Isha, Phone Number: 4567892341
Name: Alie, Phone Number: 8606313009
Name: Isha, Phone Number: 8608302732
Name: Isha, Phone Number: 9123456789
Prints the phonebook without("Jane", "0000000030") (using name as key  & inorder tree traversal):
Name: Alie, Phone Number: 8606313009
Name: Isha, Phone Number: 8608302732
Name: Isha, Phone Number: 9123456789
Name: Isha, Phone Number: 4567892341
Name: Rosy, Phone Number: 2134567890
```

Then I tested a person John and a phone number that was not in the phonebook and tried deleting it. Since, this isn't possible the method returned false as shown in the output below.

```
Delete Testing for John & Phone# 111111111
The method will return false, since John is not found in the phonebook: false
```

3. **PhBNameSearch((String name) Method-- has a worst case running time complexity of O(n), where n is the total number of nodes in the tree since it must traverse all nodes**

   - It takes input as name of the person

- It returns a linked list of phone numbers with this name.
- I searched for the name Isha (that was listed three times in the Phonebook) and tested using the code segment shown in the TestClass below

```
    System.out.println("Search for all phonenumbers with name Isha in the Phonebook ");
    String personName = "Isha";
    Link
    if (
        for (Person p : phoneNumberList) {
            System.out.print(p.getPhoneNumber() + " ");

        }
    } else {
        System.out.print("The " + personName + " doesnot exist in the Double Tree book ");
    }
}
```

```
Search for all phonenumbers with name Isha in the Phonebook
The phone numbers associated with Isha are: 8608302732 9123456789 4567892341
```

- In the below screenshot is the output that displays when, I searched for the name Isha(that is written 3 times in the phonebook with different numbers), it shows that my PhBNameSearch method works properly and is able to find all instances of the number

```
================================================================================
Search for all phonenumbers with name Isha in the Phonebook
The phone numbers associated with Isha are: 8608302732 9123456789 4567892341
================================================================================
```

4. **PhBPhoneSerach((String phoneNumber) Method-- has a worst case running time complexity of O(n), where n is the total number of nodes in the tree since it must traverse all nodes.**
    - It takes input as PhoneNo of the person and returns the name of the person associated with that PhoneNumber.
    - The code segment below shows the Test class testing implementation of this method

```
// Finds list of all phonenumbers assocaited with that same name
// in this case, it is searcing all instances of name "Isha" & returning a list
// of all the associated phone#'s
System.out.println("Search for all phonenumbers with name Isha in the Phonebook ");
String personName = "Isha";
LinkedList<Person> phoneNumberList = dblTreePhBook.PhbBNameSeach(personName);
if (phoneNumberList.size() > 0) {
    // using for each loop
    System.out.print("The phone numbers associated with " + personName + " are: ");
    for (Person p : phoneNumberList) {
        System.out.print(p.getPhoneNumber() + " ");

    }
} else {
    System.out.print("The " + personName + " doesnot exist in the Double Tree book ");
}
```

- I searched for the phone number 8608302732 which is associated with Isha and then 8606313009 which was associated with Alie and code outputs the person node Isha and Alie found(shown in the screenshot below meaning that the PhBPhoneSearch method successfully works

```
Search the phone book by phone number and returns the associated name and phone number
Person name and phone number with searched phone number 8608302732 -> Name: Isha, Phone Number: 8608302732
Person name and phone number with searched phone number 8606313009 -> Name: Alie, Phone Number: 8606313009
```

Explanation of Classes Created:

- **TreeNode Class**: It is a helper Treenode class of type Person and used in both BSTs(BSTName. BSTPhoneNo). It has 5 private fields - root, data , key, leftChild, rightChild and it's getter and setter methods in order to have a more proper way of information hiding and encapsulation.

- **BSTPhoneNo class** - It has a private rootPhoneNumber field of type TreeNode<Person> and PhoneNo is key. It has 6 methods .

    1. **public boolean insert(Person data)** (It takes a person object and returns true if added and False if the phone number already exists.
        A. The worst case running time complexity of the above method - **O(n)**

2. **public boolean delete(Person data)** (delete method that takes a name and phone number and returns True if deleted and False if there is no person with such a pair of values exists)
    A. The worst case running time complexity of the above method - **O(n)**

3. **private void deleteNode (TreeNode<Person> toDelete, TreeNode<Person> parent)**
   It's a helper method for above delete method
    A. The worst case running time complexity of the above method - **O(n)**

4. **private TreeNode<Person> searchTree(TreeNode<Person> root, String key)**
    A. The worst case running time complexity of the above method - **O(n)**
    B. Key is PhoneNumber

5. **public String search(String key)**
    A. The worst case running time complexity of the above method - **O(n)**
    B. Key is PhoneNumber

6. **public void inOrder(TreeNode<Person> root)(**This method prints InOrder (i.e it will print all persons in the phone book in the ascending order of the key i.e PhoneNo)
    A. The worst case running time complexity of the above method - **O(n).** This is because you traverse each node once and it must go through all the nodes.

- **BSTName class** - It has a private rootName field of type TreeNode<Person> and Name is key . It has 5 methods .

1. **public boolean insert(Person data)** (It takes a person object and returns true if added and False if the phone number already exists.
    A. The worst case running time complexity of the above method - **O(n)**
2. **public boolean delete(Person data)** (delete method that takes a name and phone number and returns True if deleted and False if there is no person with such a pair of values exists)

A. The worst case running time complexity of the above method - **O(n)**

3. **private void deleteNode**
   (TreeNode<Person> toDelete, TreeNode<Person> parent)
   It's a helper method for above delete method
   A. The worst case running time complexity of the above method - **O(n)**

4. **public LinkedList<Person> searchTree(String key)**
   A. The worst case running time complexity of the above method - **O(n)**
   B. Input key is person's name
   C. It returns a List Person's Phone Number associated with Name

5. **public void inOrder(TreeNode<Person> root)(**This method prints InOrder (i.e it will print all persons in the phone book in the ascending order of the key i.e. name)
   A. The worst case running time complexity of the above method - **O(n).** This is because you traverse each node once and it must go through all the nodes.

- **DoubleTreePhBook class - It implements PhoneBook<Person> interface.**

  A. It has 2 root variables
     - (rootName(key is name)
     - rootPhoneNo(key s phoneNo) of type TreeNode<Person>
  B. It has 2 reference variables of type BSTPhoneNo(bstPhoneNo) and BSTName(BSTName)
  C. It has 4 methods- PhbBInsert, PhbBDelete,PhbBNameSeach,PhbBPhoneSeach, sd discussed above.

- **Person Class** - It implements Comparable Interface of type Person. It has 2 private fields name and phone number and getter and setter methods
- **PhoneBook Interface** : A phonebook Interface of type Person created for abstraction and it has 4 methods:
  A. **a.boolean PhbBInsert**(String name, String phoneNumber);
  B. **boolean PhbBDelete**(String name, String phoneNumber);
  C. **LinkedList<Person> PhbBNameSeach**(String name);
  D. **String PhbBPhoneSeach(String phoneNumber)**;