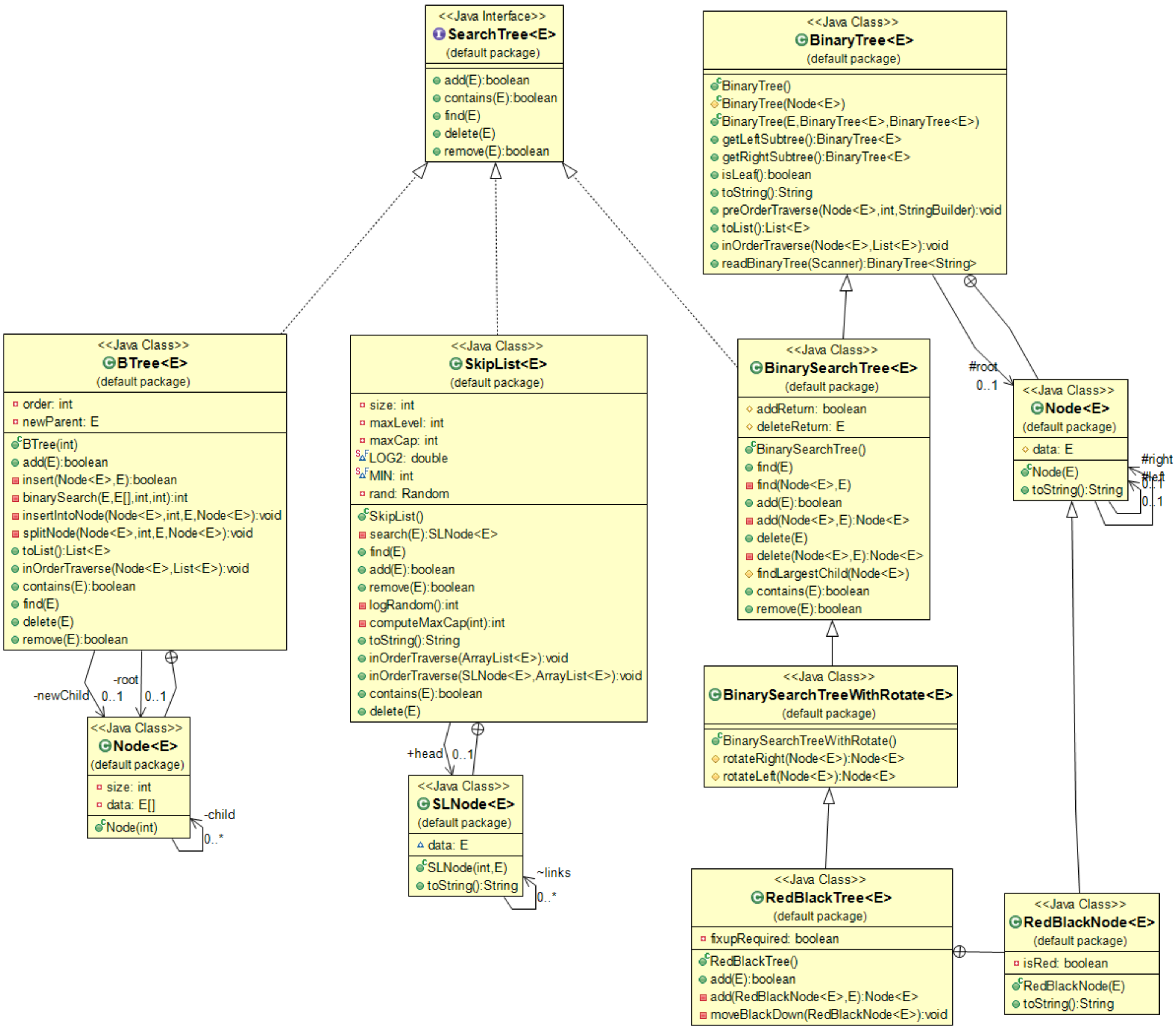
GIT Department of Computer Engineering
CSE 222/505- Spring 2020
Homework 7
Report

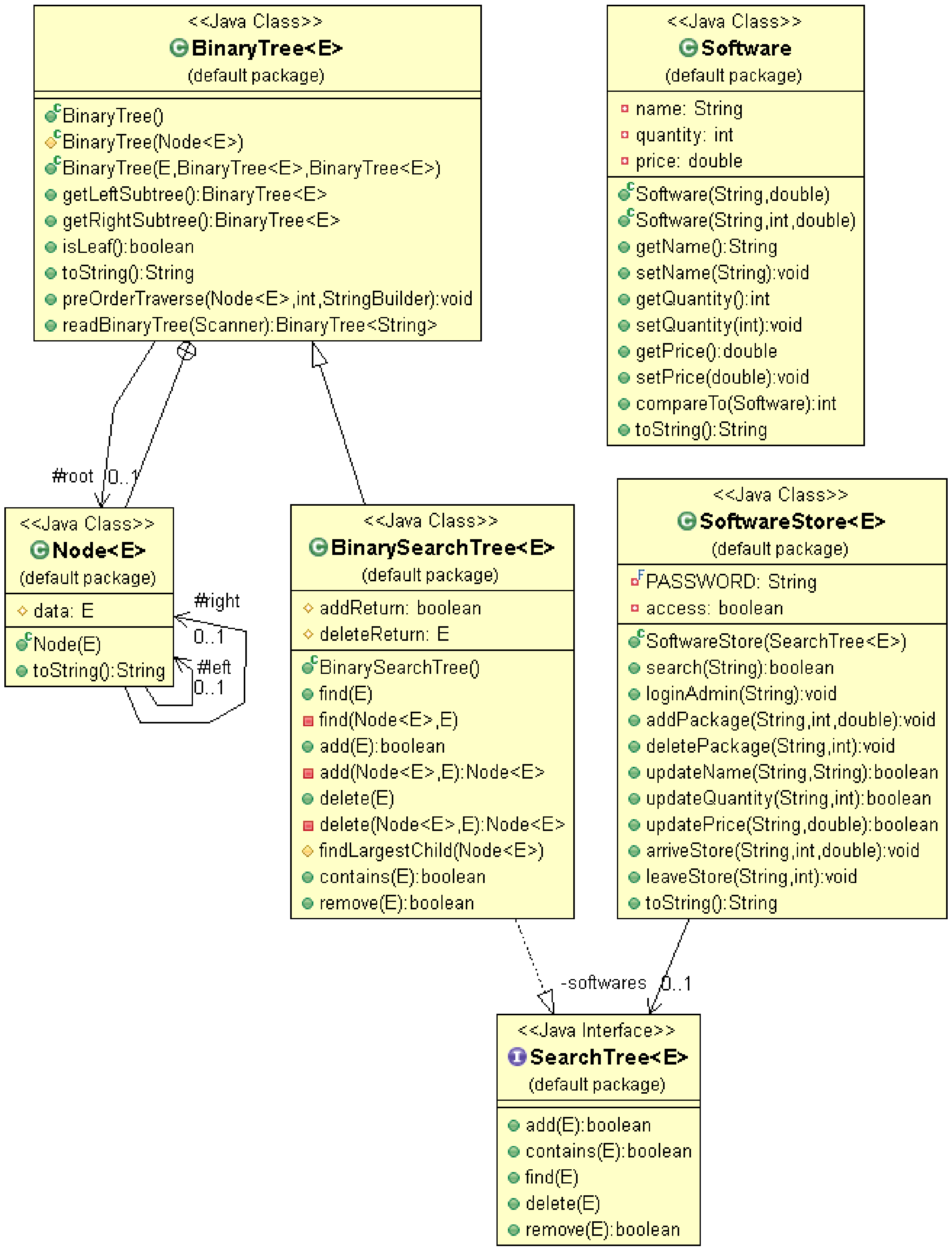
Şeyda ÖZER
171044023

CLASS DIAGRAMS

Q3:



Q4:



<<Java Class>>
BinaryTree<E>
(default package)
BinaryTree()
BinaryTree(Node<E>)
BinaryTree(E,BinaryTree<E>,BinaryTree<E>)
getLeftSubtree():BinaryTree<E>
getRightSubtree():BinaryTree<E>
isLeaf():boolean
toString():String
preOrderTraverse(Node<E>,int,StringBuilder):void
readBinaryTree(Scanner):BinaryTree<String>
<<Java Class>>
Software
(default package)
name: String
quantity: int
price: double
Software(String,double)
Software(String,int,double)
getName():String
setName(String):void
getQuantity():int
setQuantity(int):void
getPrice():double
setPrice(double):void
compareTo(Software):int
toString():String
#root
0..1
<<Java Class>>
Node<E>
(default package)
data: E
Node(E)
toString():String
#right
0..1
#left
0..1
<<Java Class>>
BinarySearchTree<E>
(default package)
addReturn: boolean
deleteReturn: E
BinarySearchTree()
find(E)
find(Node<E>,E)
add(E):boolean
add(Node<E>,E):Node<E>
delete(E)
delete(Node<E>,E):Node<E>
findLargestChild(Node<E>)
contains(E):boolean
remove(E):boolean
<<Java Class>>
SoftwareStore<E>
(default package)
PASSWORD: String
access: boolean
SoftwareStore(SearchTree<E>)
search(String):boolean
loginAdmin(String):void
addPackage(String,int,double):void
deletePackage(String,int):void
updateName(String,String):boolean
updateQuantity(String,int):boolean
updatePrice(String,double):boolean
arriveStore(String,int,double):void
leaveStore(String,int):void
toString():String
-softwares
0..1
<<Java Interface>>
SearchTree<E>
(default package)
add(E):boolean
contains(E):boolean
find(E)
delete(E)
remove(E):boolean

PROBLEM SOLUTION APPROACH

Q3:

I used a small size when validating data structures. The inOrderTraverse method in SkipList class throws an exception (java.lang.StackOverflowError), when I used a size larger than 5000. I did not solve this problem.

Average Running Time (ms)	10000	20000	40000	80000
Binary Search Tree – Insert	0	0,7	0,2	0
Red Black Tree – Insert	0	0	0	0
Tree Set – Insert	0	0	0	0
Skip List - Insert	0	0	0,1	0,1
Concurrent Skip List Set – Insert	0,1	0	0	0
Binary Search Tree – Delete	0	0,6	0,5	0,3
Tree Set – Delete	0,1	0	0,1	0
Skip List - Delte	0,1	0	0	0,1
Concurrent Skip List Set – Insert	0,1	0,1	0,1	0,1

Binary Search Tree:

Worst case : T(n) = Q(n) (add and delete method)

The times are quite variable when adding and deleting elements to the binary search tree. Binary search tree is built according to the order of the elements. If the elements do not come in a proper order, the tree will be unbalanced. Adding and removing to unbalanced tree can happen in the worst case running time. So, the binary search tree running time does not increase regularly depending on the size in my homework. If there was a balanced tree, there would be a size-related change in running time.

Red Black Tree and Tree Set:

The upper limit in the height for a Red-Black tree is 2 log2n + 2 which is still O(log n).

Because the tree is balanced, the insertion and deletion running time is very short.

Skip List:

Since the list is an ideal structure, insertion and deletion is O(log n).

In my homework I measured running times very close for all sizes. The reason for this is that the structure is ideal and this structure facilitates the search.

The self-balancing search trees are very suitable for keeping large sizes of elements.

Q4:

I put the final variable to the class for administrator password. Also I put a boolean variable. The variable is true, if the user enters the correct password. I check if the user is an administrator with this variable.

I insert the software packages to the tree according to their names. So I can not compare other informations of the software packages. There may be many software packages with a specific quantity or price. In order to search by quantity or price, I have to visit the whole tree and look at the quantity / price information of each element. However, the tree does not allow it. I only implemented the search method by name.

Update Information:

If the administrator wants to update the name of the software package, the administrator to enter the new name and the old name.

If the administrator wants update the quantity of the software package, the administrator needs to enter the name and the new quantity of the package.

If the administrator wants update the price of the software package, the administrator needs to enter the name and the new price of the package.

TEST CASES

Q4:

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass / Fail
1	Search a software package that is in the store	Search(“Norton 4.5”)	The package is found.	The package is found.	Pass
2	Search a software package that is not in the store	Search(“Norton 1.0”)	The package is not found.	The package is not found.	Pass
3	New software packages arrive at the store	arriveStore(“Adobe Photoshop CC”, 5, 75)	The package is added.	It is added.	pass
4	a software package is sold	leaveStore(Adobe Photoshop 6.0”, 1)	The package is deleted.	It is deleted.	Pass
5	Add a new package but the admin did not enter	addPackage(“Norton 4.5”, 1 , 100)	The package is not added because the admin did not enter the system	It is not added	Pass
6	Delete a package but the admin did not enter	deletePackage(“Norton 4.5”, 1)	The package is not deleted because the admin did not enter the system	It is not deleted	Pass
7	Update the package name (admin did not enter)	updateName(“Adobe Flash 3.3”, “Adobe Flash 5.0”)	the name of the package and the location of the package in the tree are not updated.	It is not updated.	Pass
8	Update the quantity of the package (admin did not enter)	updateQuantity(“Adobe Flash 3.3”, 6)	the quantity of the package is not updated.	It is not updated.	Pass
9	Update the price of the package (admin did not enter)	updatePrice(“Norton 4.5”, 125)	the price of the package is not updated	It is not updated.	Pass
10	Log in admin	loginAdmin(“12345”)	Admin is login	Admin is login	Pass
11	Add a package that is in the store	addPackage(“Norton 4.5”, 1 , 100)	The package is added	It is added	Pass
12	Add a new package	addPackage(“Adobe XD”, 3 , 50)	The package is added	It is added	Pass
13	Delete a package . The quantity will be greater than zero after deletion.	deletePackage(“Norton 4.5”, 1)	The package is deleted	It is deleted	Pass
14	Delete a package. The quantity will be zero after deletion.	deletePackage(“Adobe Photoshop 6.2”, 1)	The package is deleted	It is deleted	Pass

15	Update the package name	updateName("Adobe Flash 3.3", "Adobe Flash 5.0")	the name of the package and the location of the package in the tree are updated.	It is updated.	Pass
16	Update the quantity of the package	updateQuantity("Adobe Flash 5.0", 6)	the quantity of the package is updated.	It is updated.	Pass
17	Update the price of the package	updatePrice("Norton 4.5", 125)	the price of the package is updated	It is updated.	Pass

RUNNING AND RESULTS

Q4:

Test Case 1:

Test Data:

```
System.out.println("The item is searching: "
                        + store.search("Norton 4.5"));
```

Result:

```
The item is searching: true
```

Test Case 2:

Test Data:

```
System.out.println("The item is searching: "
                        + store.search("Norton 1.0") + "\n");
```

Result:

```
The item is searching: false
```

The tree:

```
Adobe Photoshop 6.0
Quantity: 1
Price   : 100.0
  Adobe Flash 3.3
Quantity: 1
Price   : 100.0
    null
    Adobe Flash 4.0
Quantity: 1
Price   : 125.0
      null
      null
  Adobe Photoshop 6.2
Quantity: 1
Price   : 150.0
    null
    Norton 4.5
Quantity: 1
Price   : 100.0
      null
      Norton 5.5
Quantity: 1
Price   : 200.0
        null
        null
```

Test Case 3:

Test Data:

```
store.arriveStore("Adobe Photoshop CC", 5, 75);
```

Result:

```
Adobe Photoshop 6.0
Quantity: 1
Price    : 100.0
  Adobe Flash 3.3
Quantity: 1
Price    : 100.0
    null
    Adobe Flash 4.0
Quantity: 1
Price    : 125.0
      null
      null
  Adobe Photoshop 6.2
Quantity: 1
Price    : 150.0
    null
    Norton 4.5
Quantity: 1
Price    : 100.0
      Adobe Photoshop CC
Quantity: 1
Price    : 75.0
        null
        null
      Norton 5.5
Quantity: 1
Price    : 200.0
        null
        null
```

Test Case 4:

Test Data:

```
store.leaveStore("Adobe Photoshop 6.0", 1);
```

Result:

```
Adobe Flash 4.0
Quantity: 1
Price    : 125.0
  Adobe Flash 3.3
Quantity: 1
Price    : 100.0
    null
    null
  Adobe Photoshop 6.2
Quantity: 1
Price    : 150.0
    null
    Norton 4.5
Quantity: 1
Price    : 100.0
      Adobe Photoshop CC
Quantity: 1
Price    : 75.0
        null
        null
      Norton 5.5
Quantity: 1
Price    : 200.0
        null
        null
```

Test Case 5 and 6:

Test Data:

```
store.addPackage("Norton 4.5", 1, 100);
```

```
store.deletePackage("Norton 4.5", 1);
```

Result:

These methods does not run because the administrator did not enter the system.

Test Case 7 and 8 and 9:

Test Data:

```
System.out.println("The package updating: "
        + store.updateName("Adobe Flash 3.3", "Adobe Flash 5.0"));
```

```
System.out.println("The package updating: "
        + store.updateQuantity("Adobe Flash 3.3", 6));
```

```
System.out.println("The package updating: "
        + store.updatePrice("Norton 4.5", 125));
```

Result:

```
The package updating: false
The package updating: false
The package updating: false
```

Test Case 10:

Test Data:

```
store.loginAdmin("12345");
```

Result:

The admin is entered the system.

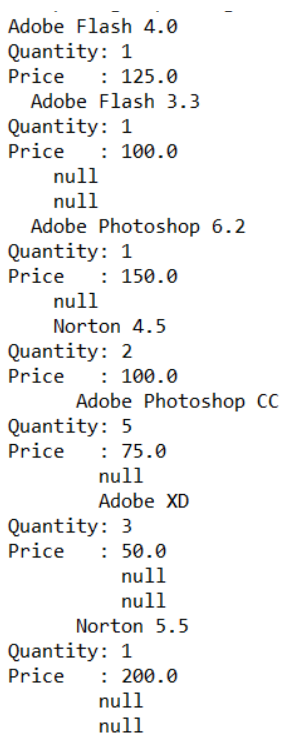
Test Case 11 and 12:

Test Data:

```
store.addPackage("Norton 4.5", 1, 100);
```

```
store.addPackage("Adobe XD", 3, 50);
```

Result:



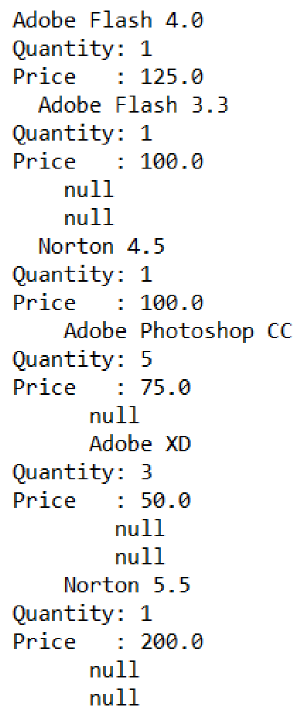
```
Adobe Flash 4.0
Quantity: 1
Price    : 125.0
  Adobe Flash 3.3
Quantity: 1
Price    : 100.0
    null
    null
  Adobe Photoshop 6.2
Quantity: 1
Price    : 150.0
    null
    Norton 4.5
Quantity: 2
Price    : 100.0
      Adobe Photoshop CC
Quantity: 5
Price    : 75.0
        null
        Adobe XD
Quantity: 3
Price    : 50.0
          null
          null
      Norton 5.5
Quantity: 1
Price    : 200.0
        null
        null
```

Test Case 13 and 14:

Test Data:

```
store.deletePackage("Norton 4.5", 1);

store.deletePackage("Adobe Photoshop 6.2", 1);
```

Result:



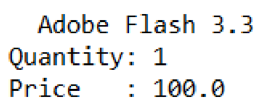
```
Adobe Flash 4.0
Quantity: 1
Price    : 125.0
  Adobe Flash 3.3
Quantity: 1
Price    : 100.0
    null
    null
  Norton 4.5
Quantity: 1
Price    : 100.0
    Adobe Photoshop CC
Quantity: 5
Price    : 75.0
      null
      Adobe XD
Quantity: 3
Price    : 50.0
        null
        null
    Norton 5.5
Quantity: 1
Price    : 200.0
      null
      null
```

Test Case 15:

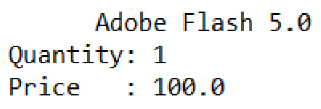
Test Data:

```
store.updateName("Adobe Flash 3.3", "Adobe Flash 5.0");
```

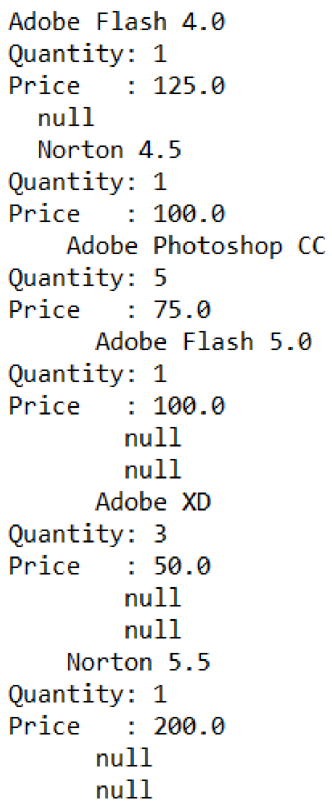
Result:



```
  Adobe Flash 3.3
Quantity: 1
Price    : 100.0
```



```
      Adobe Flash 5.0
Quantity: 1
Price    : 100.0
```



```
Adobe Flash 4.0
Quantity: 1
Price    : 125.0
  null
  Norton 4.5
Quantity: 1
Price    : 100.0
    Adobe Photoshop CC
Quantity: 5
Price    : 75.0
      Adobe Flash 5.0
Quantity: 1
Price    : 100.0
        null
        null
      Adobe XD
Quantity: 3
Price    : 50.0
        null
        null
    Norton 5.5
Quantity: 1
Price    : 200.0
      null
      null
```

Test Case 16 and 17:

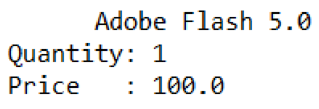
Test Data:

```
store.updateQuantity("Adobe Flash 5.0", 6);
```

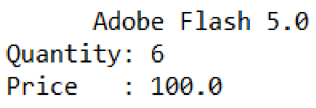
```
store.updatePrice("Norton 4.5", 125);
```

Result:

For 16:

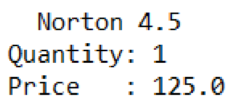


```
      Adobe Flash 5.0
Quantity: 1
Price    : 100.0
```

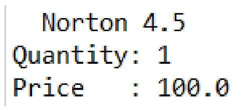


```
      Adobe Flash 5.0
Quantity: 6
Price    : 100.0
```

For 17:



```
  Norton 4.5
Quantity: 1
Price    : 125.0
```



```
  Norton 4.5
Quantity: 1
Price    : 100.0
```

```
Adobe Flash 4.0
Quantity: 1
Price    : 125.0
  null
  Norton 4.5
Quantity: 1
Price    : 125.0
    Adobe Photoshop CC
Quantity: 5
Price    : 75.0
      Adobe Flash 5.0
Quantity: 6
Price    : 100.0
        null
        null
      Adobe XD
Quantity: 3
Price    : 50.0
        null
        null
    Norton 5.5
Quantity: 1
Price    : 200.0
      null
      null
```

Q3:

verifying:

(size = 10)

```java
for(int i = 0; i<10; i++) {
    int r = rInt.nextInt(); // random number
    bst1_s1.add(r);         // binary search tree
    rbt1_s1.add(r);         // red black tree
    ts1_s1.add(r);          // tree set
    sl1_s1.add(r);          // skip list
    csl1_s1.add(r);         // concurrent skip list set
}

ArrayList<Integer> list = (ArrayList<Integer>) bst1_s1.toList();
System.out.println("The list is sorted(BST): " + verify(list));
System.out.println(list);

list = (ArrayList<Integer>) rbt1_s1.toList();
System.out.println("The list is sorted(Red-Black): " + verify(list));
System.out.println(list);

System.out.println("The tree set:\n" + ts1_s1.toString());

list = new ArrayList<Integer>();
sl1_s1.inOrderTraverse(list);
System.out.println("The list is sorted(SkipList): " + verify(list));
System.out.println(list);

System.out.println("The concurrent skip list set:\n" + csl1_s1.toString());
```

```
The list is sorted(BST): true
[-2055856566, -918874401, -627842508, -409323591, -169232228, -108917281, 339095560, 1283230140, 1767557564, 2127976704]
The list is sorted(Red-Black): true
[-2055856566, -918874401, -627842508, -409323591, -169232228, -108917281, 339095560, 1283230140, 1767557564, 2127976704]
The tree set:
[-2055856566, -918874401, -627842508, -409323591, -169232228, -108917281, 339095560, 1283230140, 1767557564, 2127976704]
The list is sorted(SkipList): true
[-2147483648, -2055856566, -918874401, -627842508, -409323591, -169232228, -108917281, 339095560, 1283230140, 1767557564]
The concurrent skip list set:
[-2055856566, -918874401, -627842508, -409323591, -169232228, -108917281, 339095560, 1283230140, 1767557564, 2127976704]
```

(size = 5000)

```java
for(int i = 0; i<5000; i++) {
    int r = rInt.nextInt(); // random number
    bst1_s1.add(r);         // binary search tree
    rbt1_s1.add(r);         // red black tree
    ts1_s1.add(r);          // tree set
    sl1_s1.add(r);          // skip list
    csl1_s1.add(r);         // concurrent skip list set
}

ArrayList<Integer> list = (ArrayList<Integer>) bst1_s1.toList();
System.out.println("The list is sorted(BST): " + verify(list));

list = (ArrayList<Integer>) rbt1_s1.toList();
System.out.println("The list is sorted(Red-Black): " + verify(list));

list = new ArrayList<Integer>();
sl1_s1.inOrderTraverse(list);
System.out.println("The list is sorted(SkipList): " + verify(list));
```

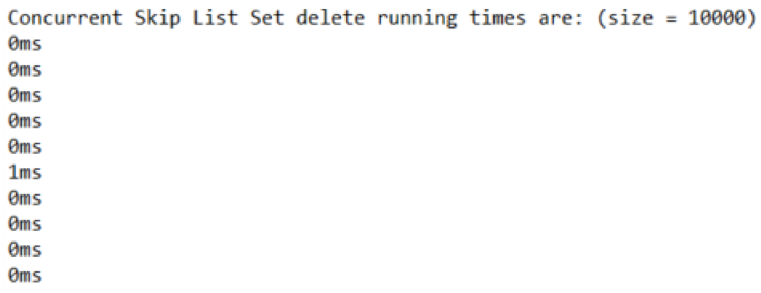
```
The list is sorted(BST): true
The list is sorted(Red-Black): true
The list is sorted(SkipList): true
```

Test Results:

Size = 10000

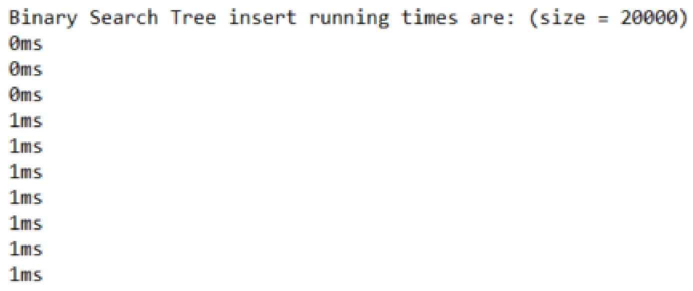
```
Binary Search Tree insert running times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms

Red Black Tree insert running times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```

```
Tree Set insert running times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms

Skip List insert running times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms

Concurrent Skip List Set insert running times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
1ms
0ms

Binary Search Tree delete running times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms

Tree Set delete running times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
1ms
0ms
0ms
0ms
0ms

Skip List insert delete times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
1ms
0ms
```

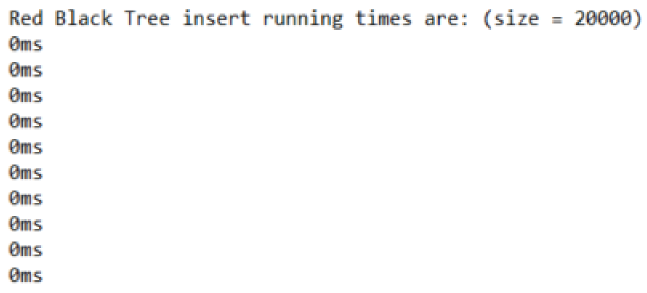


```
Concurrent Skip List Set delete running times are: (size = 10000)
0ms
0ms
0ms
0ms
0ms
1ms
0ms
0ms
0ms
0ms
```

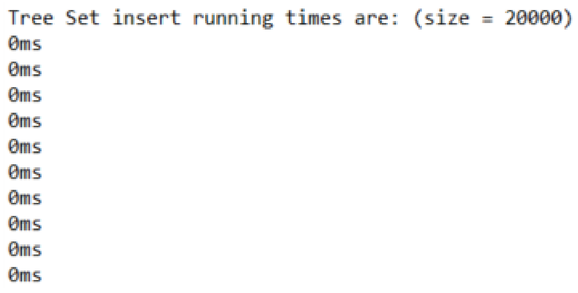
Size = 20000;



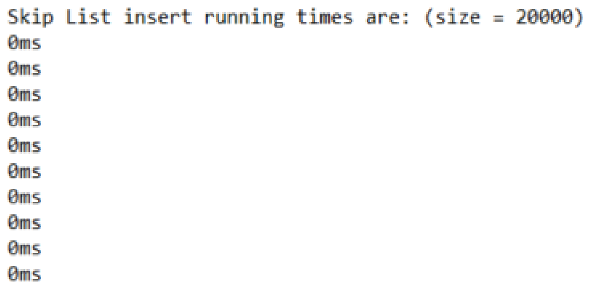
```
Binary Search Tree insert running times are: (size = 20000)
0ms
0ms
0ms
1ms
1ms
1ms
1ms
1ms
1ms
1ms
```



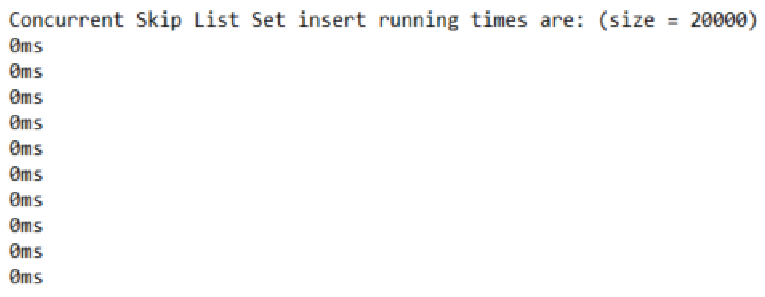
```
Red Black Tree insert running times are: (size = 20000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```



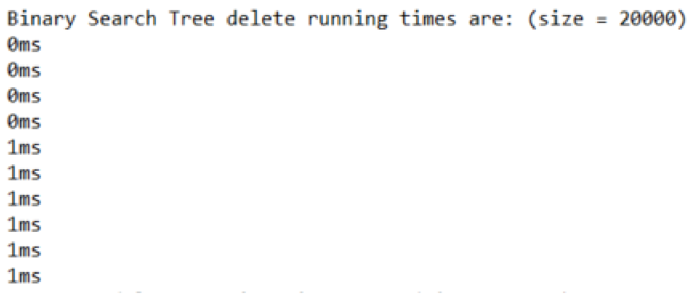
```
Tree Set insert running times are: (size = 20000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```



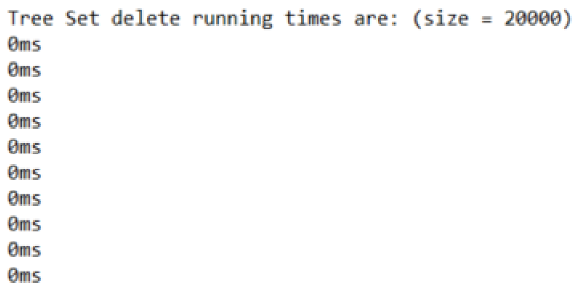
```
Skip List insert running times are: (size = 20000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```



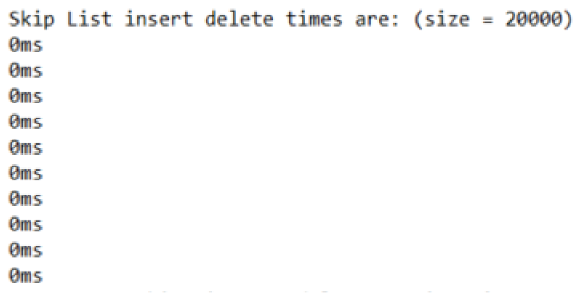
```
Concurrent Skip List Set insert running times are: (size = 20000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```



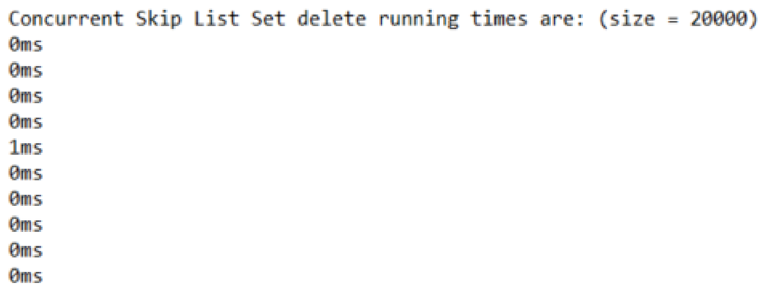
```
Binary Search Tree delete running times are: (size = 20000)
0ms
0ms
0ms
0ms
1ms
1ms
1ms
1ms
1ms
1ms
```



```
Tree Set delete running times are: (size = 20000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```



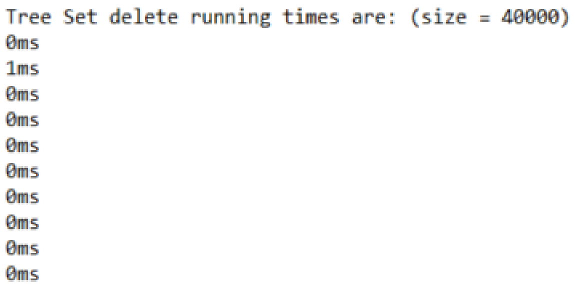
```
Skip List insert delete times are: (size = 20000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```



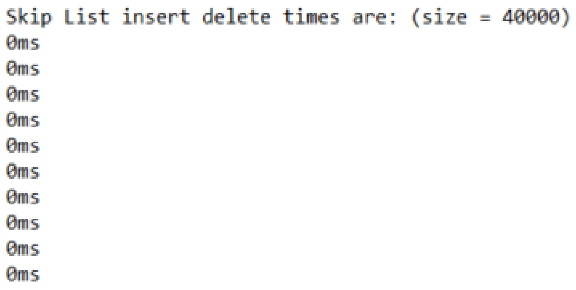
```
Concurrent Skip List Set delete running times are: (size = 20000)
0ms
0ms
0ms
0ms
1ms
0ms
0ms
0ms
0ms
0ms
```

Size = 40000;

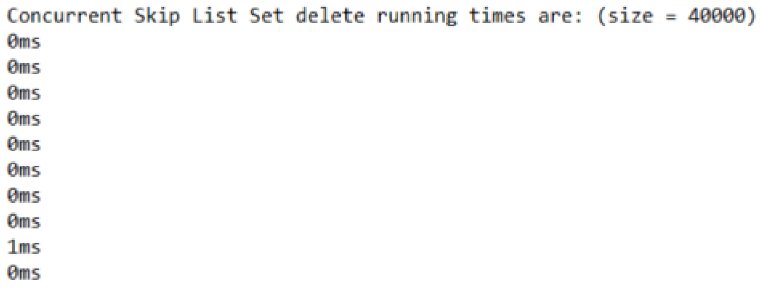
```
Binary Search Tree insert running times are: (size = 40000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
1ms
1ms

Red Black Tree insert running times are: (size = 40000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms

Tree Set insert running times are: (size = 40000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms

Skip List insert running times are: (size = 40000)
0ms
0ms
0ms
0ms
1ms
0ms
0ms
0ms
0ms
0ms

Concurrent Skip List Set insert running times are: (size = 40000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms

Binary Search Tree delete running times are: (size = 40000)
0ms
0ms
0ms
0ms
0ms
1ms
1ms
1ms
1ms
1ms
```



```
Tree Set delete running times are: (size = 40000)
0ms
1ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```

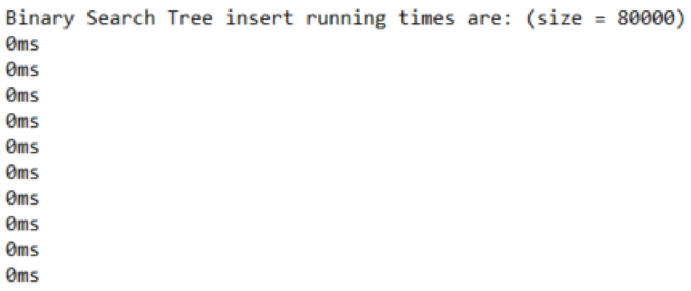


```
Skip List insert delete times are: (size = 40000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```

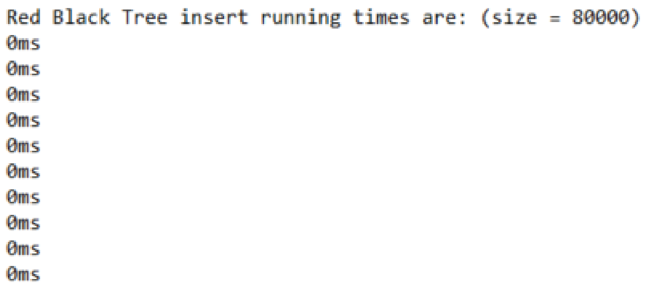


```
Concurrent Skip List Set delete running times are: (size = 40000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
1ms
0ms
```

Size = 80000;



```
Binary Search Tree insert running times are: (size = 80000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```



```
Red Black Tree insert running times are: (size = 80000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```

```
Tree Set insert running times are: (size = 80000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```

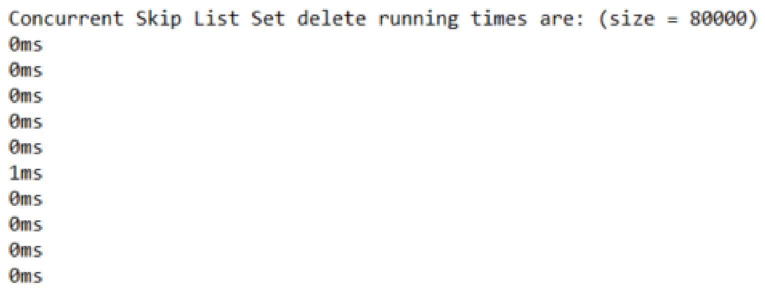
```
Skip List insert running times are: (size = 80000)
0ms
1ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```

```
Concurrent Skip List Set insert running times are: (size = 80000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```

```
Binary Search Tree delete running times are: (size = 80000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
1ms
1ms
1ms
```

```
Tree Set delete running times are: (size = 80000)
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
0ms
```

```
Skip List insert delete times are: (size = 80000)
0ms
0ms
0ms
0ms
0ms
1ms
0ms
0ms
0ms
0ms
```



```
Concurrent Skip List Set delete running times are: (size = 80000)
0ms
0ms
0ms
0ms
0ms
1ms
0ms
0ms
0ms
0ms
```