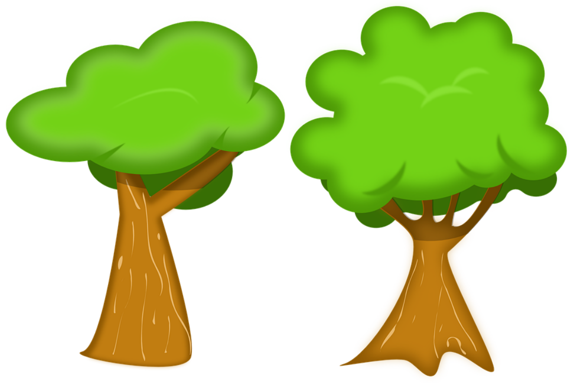
Trees Assignment Part #1

Re-submit Assignment

Due May 7, 2017 by 11:59pm **Points** 100 **Submitting** a file upload
File Types zip

Overview



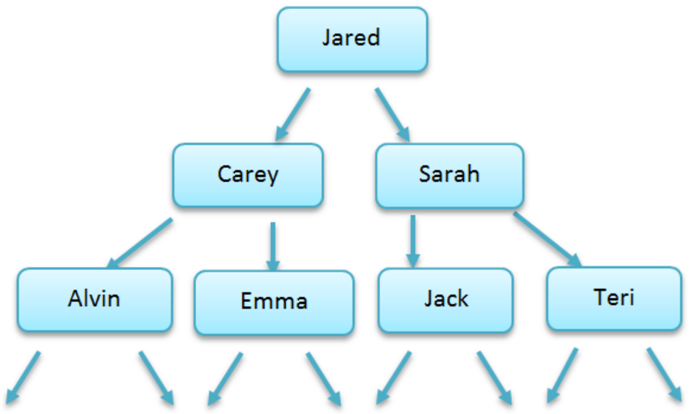
This assignment will focus on creating a variant of the Binary Search Tree class we wrote and tested in class. We will then create a Symbol Table API using our new tree class!

This is the first of two assignments that are related. The second assignment description can be found here: Trees Assignment Part #2.

Iterative Binary Search Tree

The prototypical binary search tree (BST) structure uses recursion for most of its methods, including: add(), contains(), remove() & tree traversal methods. There is a penalty in terms of memory and CPU usage when rapidly making recursive method calls as we invoke the methods above. Even more so with an unbalanced tree, where the methods can result in Stack Overflow errors and termination of your program.

We will be implementing a BST using iteration (loops) instead of recursion. To be clear, the structure described below uses no recursion whatsoever. Any students submitting solutions with recursive methods will be forced to resubmit your assignments.



Class Definition

Your BST class should use generics and your generic type must be comparable. For example:

```
public class BinarySearchTree<T extends Comparable<T>>
{
    //stuff goes here...
}
```

Binary Search Tree API

Complete your BST class by implementing the following methods according to their descriptions. Be careful to not how these methods differ from our examples in class.

Method	Description
public boolean addUpdate(T element)	Adds or updates an element in the binary search tree. Given bst.addUpdate(a), there are two scenarios to consider: Scenario #1: There is no element "b" in the tree for which a.compareTo(b) == 0 (i.e. the objects are considered equal). Then "a" should be added to the binary search tree. Scenario #2: There is an element "b", already in the tree, for which a.compareTo(b) == 0. Then "b" should replace "a" in the node containing "a". This method returns true when a new element is added and false if an update occurred. *Note: The reasoning behind this logic will become more clear as you progress in the assignment.* *Note: We are still not allowing duplicates at this point. Instead we are allowing updates to an object in the BST.*
public boolean contains(T element)	Returns true if the input element is in the tree, otherwise it returns false.
public T get(T element)	Given an input element "a", this returns any element "b" such that a.compareTo(b) == 0 (i.e. the objects are considered equal).
public int size()	This returns the number of elements in the tree.
public boolean isEmpty()	This returns true if there are no elements in the tree, but otherwise returns false.
public void clear()	This removes all elements in the tree.
public List<T> inOrder()	Returns a sorted list of all elements in the tree, using the In-Order tree traversal of the underlying structure.

BST Testing

Create a driver class that instantiates your binary search tree and tests the methods above. You will not need to write unit tests for this part of the assignment. But you should write informal tests to verify the following scenarios:

- Adding several elements to the BST
- Verify that elements are within the tree using contains()
- Verify that the number of elements in the tree are tracked correctly using size(), isEmpty() & clear()
- Retrieve elements contained in the tree using get()
- Retrieving all elements in the tree (in sorted order) using inOrder()

Creating a Symbol Table

Symbol tables are used widely in software development to store key/value pairs in a program. They are also known as dictionaries or maps and in Java are implemented as the HashMap and TreeMap classes. Create a new class called BSTSymbolTable that allows you to store key/value pairs.

Your BSTSymbolTable should have the the following class header:

```
public class BSTSymbolTable<K extends Comparable<K>, V>
{
    //stuff goes here...
}
```

Many data structures are built based on existing data structures that are used as underlying implementations. An Array List structure uses an array to store data internally. Stack & Queue classes can be implemented using arrays or linked lists. We will be using our BinarySearchTree class as an underlying structure for this symbol table. For example:

```
public class BSTSymbolTable<K extends Comparable<K>, V>
{
    private BinarySearchTree<Entry> data;

    //stuff goes here...
}
```

Entry Class

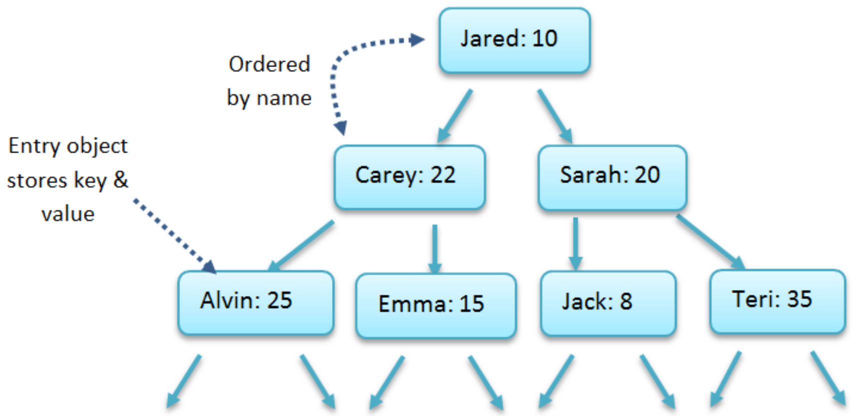
To hold key/value pairs in our map, we will use a private inner class called entry. Each entry object stores a key of generic type K and a value of generic type V. Entry objects are stored internally and are sorted by key. This allows us to keep our key/value pairs together while stored in the internal binary search tree. For example:

```
public class BSTSymbolTable<K extends Comparable<K>, V>
{
    private BinarySearchTree<Entry> data;

    //stuff goes here...

    private class Entry implements Comparable<Entry>
    {
        private K key;
        private V value;

        //stuff goes here...
    }
}
```



Symbol Table API

Complete your symbol table class by implementing the following methods according to their descriptions.

Method	Description
public boolean put(K key, V value)	Adds a key/value pair to the map if none is present, otherwise this replaces any existing entry in the map with the same key. This method returns true if a new key/value pair was added to the map, otherwise false.
public V get(K key)	Returns the value associated with the input key. If no key is found this method will return null. *Note: To look up a value in the map, you must first create a new Entry object with the key you are searching for.*
public boolean containsKey(K key)	Returns true if the input key is located in the map, otherwise

	false.
public boolean containsValue(V value)	Returns true if the input value is located in the map, otherwise false.
public List<K> keys()	Returns an ordered list of keys contained in the map.
public List<V> values()	Returns an unordered list of values contained in the map.
public int size()	This returns the number of key/value pairs in the table.
public boolean isEmpty()	This returns true if there are no key/value pairs in the table, but otherwise returns false.
public void clear()	This removes all key/value pairs in the table.

Symbol Table Testing

Create a driver class that instantiates your symbol table and tests the methods above. You will not need to write unit tests for this part of the assignment. But you should write informal tests to verify the following scenarios:

- Adding several key/value pairs to the table (names/ages, objects/colors, website/ratings, or any other examples you can think up)
- Verify that keys or values are within the table using containsKey() & containsValue()
- Verify that the number of key/value pairs in the tree are tracked correctly using size(), isEmpty() & clear()
- Verify that you can print out the keys in the table in sorted order
- Verify that you can print out the values in the table

Submission

Submit a project with your BST class, symbol table class and your test files.

Trees Assignment Part #1 Rubric

Criteria	Ratings		Pts
The addUpdate() method adds a new element to your tree or updates an existing element.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
The contains() and get() methods lets you query for elements or retrieve elements in the BST.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
The size(), isEmpty() and clear() methods correctly track or remove elements in the BST.	2.5 pts Full Marks	0.0 pts No Marks	2.5 pts
The inOrder() method returns an ordered list of unique elements from the BST.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Your informal tests on the BST test each of the listed scenarios.	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
Your symbol table correctly stores Entry objects.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
The put() and get() methods of your symbol table allow you to store and retrieve key/value pairs.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
The containsKey() and containsValue() methods allow you to query for keys or values in the symbol table.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
The keys() and values() methods correctly return lists of keys or values.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
The size(), isEmpty() and clear() methods allow you to track or remove elements in the symbol table.	2.5 pts Full Marks	0.0 pts No Marks	2.5 pts
Your informal tests on the symbol table test each of the listed scenarios.	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
Styles: naming conventions, brace placement, spacing, commented code, redundancy, packages and magic numbers.	20.0 pts Full Marks	0.0 pts No Marks	20.0 pts
Formal documentation: full Javadocs & comment block header.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
			Total Points: 100.0