# Assignment 8

*BSTs and Maps*

## Introduction

In this assignment you will implement a binary search tree (BST). Each node in the BST will contain a key and a value associated with the key. The elements in the tree will be ordered by the key stored at each node.

After you have completed your implemention the binary search tree, you will use that implementation to implement the Map ADT.

**NOTE:** The automated grading of your assignment will include some different and additional tests to those found in the `A7Tester.java` file. For all assignments, you are expected to write additional tests until you are convinced each method has full test coverage.

## Objectives

Upon finishing this assignment, you should be able to:

- Write an implementation of a reference-based binary search tree
- Write code that uses an implementation of the Binary Search Tree ADT
- Write code that implements the Map ADT
- Describe the differences between the four different tree traversal algorithms

## Submission and Grading

Attach `BinarySearchTree.java` and `BSTMap.java` to the BrightSpace assignment page. Remember to click **submit** afterward. You should receive a notification that your assignment was successfully submitted.

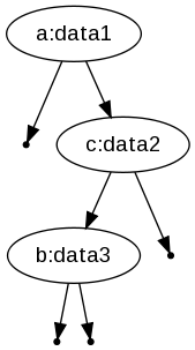
If you chose not to complete some of the methods required, you **must** provide a stub for the incomplete method(s) in order for our tester to compile. If you submit files that do not compile with our tester, you will receive a zero grade for the assignment. It is your responsibility to ensure you follow the specification and submit the correct files. Additionally, your code must not be written to specifically pass the test cases in the tester, instead, it must work on all valid inputs. We may change the input values during grading and we will inspect your code for hard-coded solutions.

Be sure you submit your assignment, not just save a draft. All late and incorrect submissions will be given a zero grade. A reminder that it is OK to talk about your assignment with your classmates, but not to share code electronically or visually (on a display screen or paper). Plagiarism detection software will be run on all submissions.

## Part 1 Instructions

The binary search tree for this assignment will use generics to allow users to specific the data type for both the key and associated value for all elements stored in the tree. For example, the following code produces the tree shown:

```
BinarySearchTree<String, String> t1;
t1 = new BinarySearchTree<String,String>();
t1.insert("a", "data1");
t1.insert("c", "data2");
t1.insert("b", "data3");
```



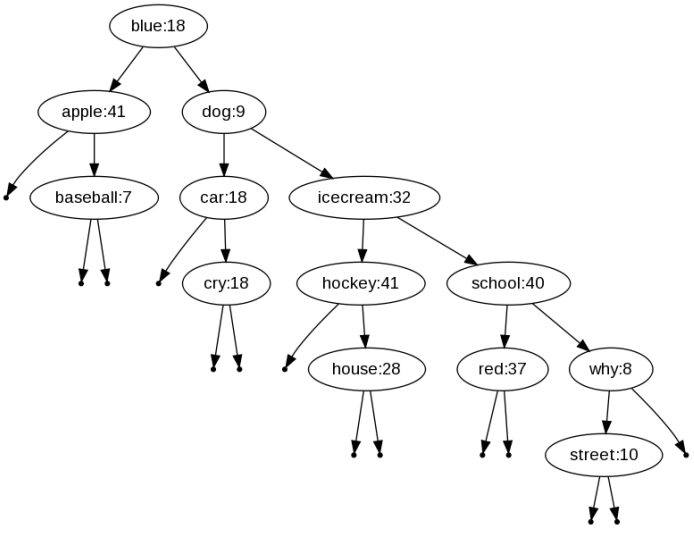

When testing your code, you may find it useful to look at the trees the created and used in `A8Tester.java`. A visual representation of each of the four test trees is provided on the following pages of this assignment description.

Since it is helpful to be able to visualize trees, you have been provided with a TreeView class which takes an instance of a binary search tree as the constructor's parameter. For example, if after the code shown above you use the following code:

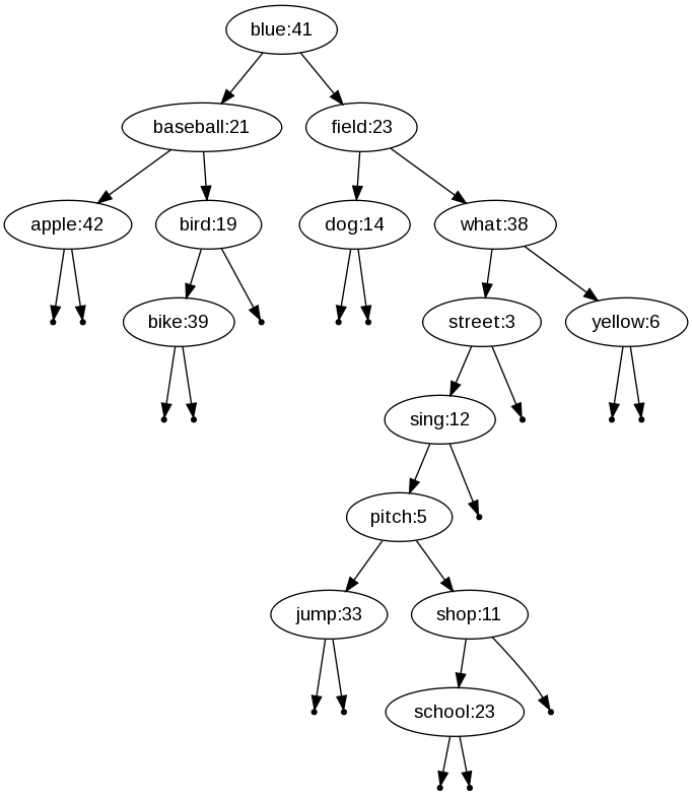
```
TreeView<String,String> v1 = new TreeView(t1);
v1.dotPrint();
```

The result will be the contents of a dot file printed to the standard output. You can copy and paste that output text into the following web page to get a picture of your tree: http://www.webgraphviz.com/.

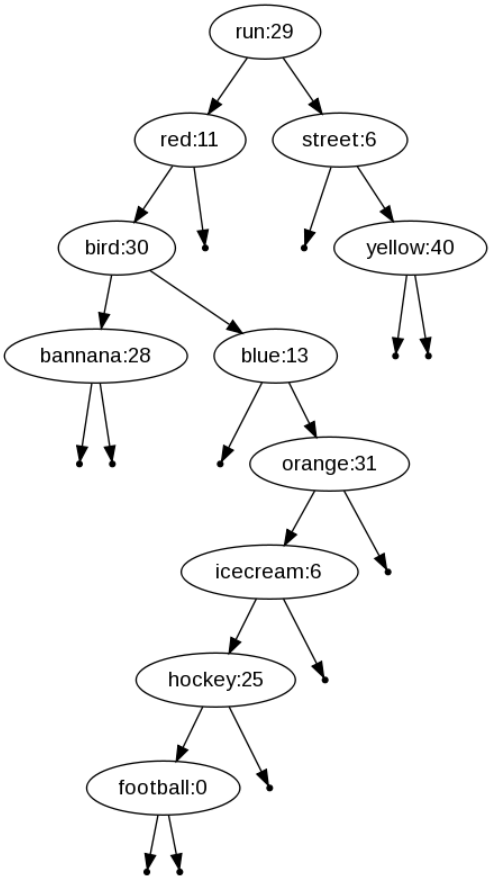
The following images are visual representations of the four trees from `A8Tester.java` that are generated using the TreeView class as described above.



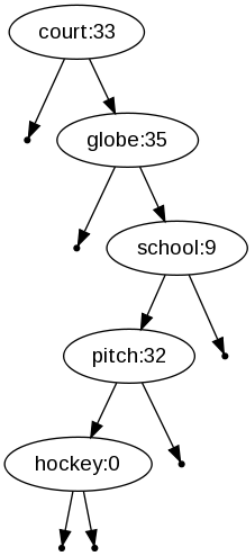

Tree 1




Tree 2




Tree 4




Tree 3

## Part 2 Instructions

Once you have finished implementing the Binary Search Tree, you will use this to implement the Map ADT in `BSTMap.java`. The Map interface is specified in `Map.java`, and provides two main operations:

- `get(key)` – return the value stored at key
- `put(key,value)` – insert the key/value pair into the map. If the key already exists, update the value stored at key to be the new value.

You have to write very little code to accomplish this – simply create an instance of your BinarySearchTree in the BSTMap constructor and then use methods you have already completed in the BinarySearchTree class to provide the services required by the Map. (In the instructor solution, all methods except `containsKey` are only one line, and containsKey contains seven lines.)