TRINITY COLLEGE
DEPARTMENT OF COMPUTER SCIENCE
CPSC 215: Data Structures and Algorithms
Instructor: Dr. Chandranil Chakraborttii
Spring 2025

Laboratory 8

Academic Honesty Policy

In working on programming assignments, you may discuss broad issues of interpretation and understanding and general approaches to a solution. However, conversion to a specific solution or to program code must be your own work. Programming assignments are expected to be the work of the individual student, designed, and coded by him or her alone. Violations are easy to identify and will be dealt with promptly according to the Academic Integrity and Intellectual Dishonesty outlined in the Student Handbook.

- Copying another person's programs or encouraging or assisting another person to commit plagiarism is cheating. In particular, the following activities are strictly prohibited:
- Giving and receiving help in the actual development of code or writing of an assignment.
- Looking at another person's code or showing your code to another person.
- Sharing a copy of all or part of your code regardless of whether that copy is on paper or in a computer file.
- Turning in the work of any other person(s) (former students, friends, textbook authors, people on the Internet, etc.) and representing it as your own work.
- Fabricating compilation or execution results.
- Use of AI ChatBots for any help regarding the assignment is **strictly prohibited.**

The penalty for cheating is a failing grade (F) for the course, and the student is asked to appear at an Academic Dishonesty Hearing. In addition, the College also places a record of the incident in the student's permanent record. It is your responsibility to protect your work from unauthorized access. Do not discard copies of your programs in public places. Do not leave computers unattended and copies of output lying around.

Objective

- To get familiar with Dictionary ADT and Binary Search Trees
- To implement a Binary Search Tree (BST) data structure in Java. Specifically, the task is to complete the methods in the BST.java class to support various operations on a binary search tree, such as insertion, removal, searching, and tree traversal.

Lab 8: Binary Search Trees and Dictionary ADT

In this exercise we will implement a Binary Search Tree (BST) along with various methods to manipulate and interact with the tree.

Input Files (Starter Code)

You will need the following classes to get started:

- BinNode.java: [ADT for binary tree nodes] (***No need to change)***
- BSTNode.java: [Definition of a Binary Node] ***(No need to change)***
- Dictionary.java: [Dictionary abstract class] ***(No need to change)***
- **BST.java:** [Binary Search Tree implementation for Dictionary ADT] ***(Need to change)***
- BSTTest.java: [Driver application] ***(No need to change)***

Description

- **BinNode Interface:**
    - This interface defines the abstract data type (ADT) for binary tree nodes.
    - It contains methods to get and set the element value, access the left and right children, and check if a node is a leaf.

- **BSTNode Class:**
    - This class implements the BinNode interface and represents a node in a binary search tree.
    - It contains fields for the key, element, and references to the left and right children.
    - Constructors are provided to create instances of BSTNode with different configurations.

- **Dictionary Interface:**
    - This interface defines the abstract data type (ADT) for a dictionary, which is a collection of key-value pairs.
    - It specifies methods for initializing the dictionary, inserting and removing records, finding records by key, and retrieving the size of the dictionary.
    - Additionally, methods are provided for displaying the elements of the tree in various traversal orders: inorder, preorder, and postorder.

- **BST Class:**
    - This class implements the Dictionary interface using a binary search tree.
    - It contains a private field root representing the root node of the tree and an integer nodecount to keep track of the number of nodes in the tree.
    - Methods are provided to insert, remove, find, and clear records from the tree.
    - Helper methods are implemented to support these operations, including recursive methods for insertion, removal, and traversal.

    - Methods are also provided to display the elements of the tree in inorder, preorder, and postorder traversal orders.

- **BSTTest Class:**
    - This class contains a main method to test the functionality of the BST class.
    - It creates instances of BST and performs various operations such as insertion, removal, searching, and traversal.
    - Test cases are designed to validate the correctness of the implemented methods and to ensure that the binary search tree behaves as expected.

Tasks:

Step 1: Review the completed classes (BinNode.java, BSTNode.java and Dictionary.java)

Step 2: Implement the remaining methods in BST.java

The Dictionary.java contains the following method signatures which are implemented in BST.java. The following methods are already completed.

- **insert(Key k, E e):** Insert a new element with the given key and value into the tree.
- **getMin():** Return the minimum key stored in the tree.
- **find(Key k):** Find and return the element with the given key in the tree, otherwise null.
- **remove(Key k):** Remove the element with the given key from the tree, if it exists.

Complete the following methods:

- **clear():** Remove all elements from the tree.
- **removeAny():** Remove and return an arbitrary element from the tree.
- **size():** Return the number of elements in the tree.
- **isEmpty():** Check if the tree is empty.
- **height():** Return the height of the tree.
- **getMax():** Return the maximum key stored in the tree.
- **display_inorder():** Display the elements of the tree in inorder traversal order.
- **display_preorder():** Display the elements of the tree in preorder traversal order.

- **display_post_order():** Display the elements of the tree in postorder traversal order.
- **containsValue(E e):** Check if the tree contains an element with the given value.
- **successor(Key k):** Return the successor key of the given key in the tree.
- **predecessor(Key k):** Return the predecessor key of the given key in the tree.

***Note**: You will need to extend the Comparable class to compare the data. You may create additional helper methods in BST class to support the implemented methods.*

Step 3: Testing for correctness

- Use class BSTTest to test the correctness of your BSTNode class.
- The class Tester is already completed, so you should not have to make any more modifications.

Step 4: Documentation and Reflection

- Ensure that all methods work correctly and handle edge cases effectively.
- Verify that the binary search tree maintains the properties of a binary search tree, such as maintaining the order of keys.
- Be sure the code is runnable, testable, and easily readable.
- Include **JavaDoc** to explain the purpose and functionality of each class, method, and exception as per standards.

Step 5: JUnit Testing (Extra Credit – 10 points)

- Add JUnit test class with methods to test any five of the methods you completed in Step 2

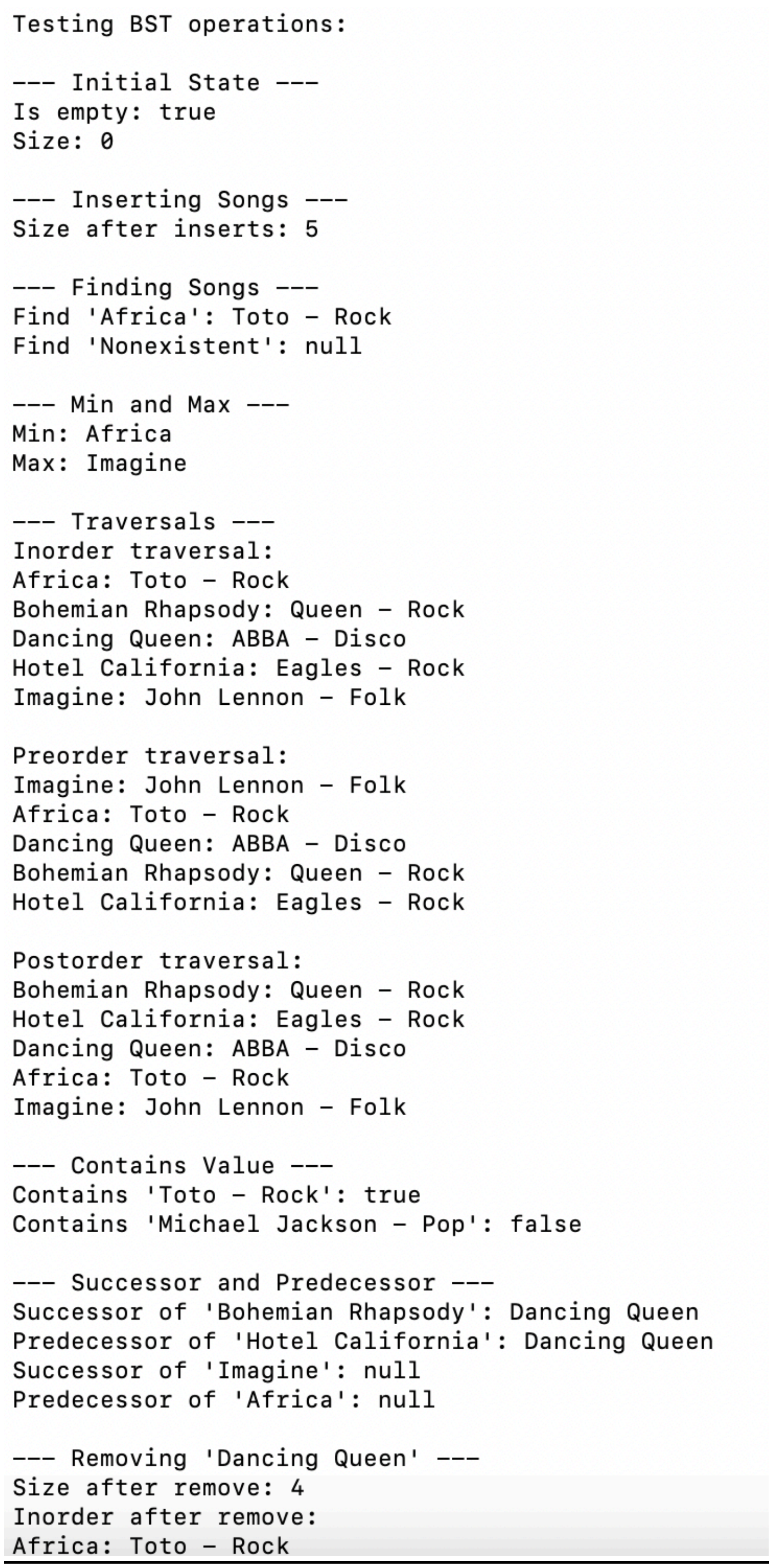
———————————————————————————————————————

Submission Guidelines:

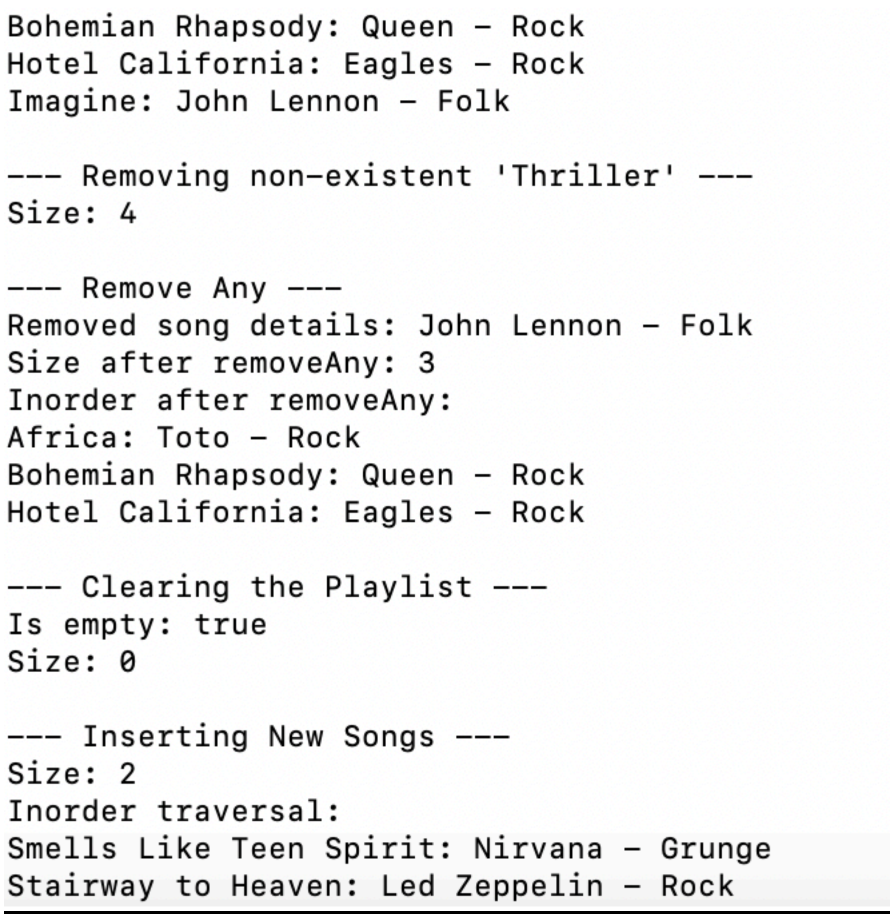
- Make sure you completed all sections with "To do" statements and added JavaDoc for the implemented methods/classes.
- Upon the completion of your lab, upload your submission to the Moodle course website as a single Zip file.
- See your TA and get your Assignment graded before leaving the lab. Otherwise, inform the instructor.

———————————————————————————————————————

Sample Output (next page)



```
Testing BST operations:

--- Initial State ---
Is empty: true
Size: 0

--- Inserting Songs ---
Size after inserts: 5

--- Finding Songs ---
Find 'Africa': Toto - Rock
Find 'Nonexistent': null

--- Min and Max ---
Min: Africa
Max: Imagine

--- Traversals ---
Inorder traversal:
Africa: Toto - Rock
Bohemian Rhapsody: Queen - Rock
Dancing Queen: ABBA - Disco
Hotel California: Eagles - Rock
Imagine: John Lennon - Folk

Preorder traversal:
Imagine: John Lennon - Folk
Africa: Toto - Rock
Dancing Queen: ABBA - Disco
Bohemian Rhapsody: Queen - Rock
Hotel California: Eagles - Rock

Postorder traversal:
Bohemian Rhapsody: Queen - Rock
Hotel California: Eagles - Rock
Dancing Queen: ABBA - Disco
Africa: Toto - Rock
Imagine: John Lennon - Folk

--- Contains Value ---
Contains 'Toto - Rock': true
Contains 'Michael Jackson - Pop': false

--- Successor and Predecessor ---
Successor of 'Bohemian Rhapsody': Dancing Queen
Predecessor of 'Hotel California': Dancing Queen
Successor of 'Imagine': null
Predecessor of 'Africa': null

--- Removing 'Dancing Queen' ---
Size after remove: 4
Inorder after remove:
Africa: Toto - Rock
```



```
Bohemian Rhapsody: Queen - Rock
Hotel California: Eagles - Rock
Imagine: John Lennon - Folk

--- Removing non-existent 'Thriller' ---
Size: 4

--- Remove Any ---
Removed song details: John Lennon - Folk
Size after removeAny: 3
Inorder after removeAny:
Africa: Toto - Rock
Bohemian Rhapsody: Queen - Rock
Hotel California: Eagles - Rock

--- Clearing the Playlist ---
Is empty: true
Size: 0

--- Inserting New Songs ---
Size: 2
Inorder traversal:
Smells Like Teen Spirit: Nirvana - Grunge
Stairway to Heaven: Led Zeppelin - Rock
```