#### 

#### 

Athlone Institute of Technology

Athlone

REPORT

INVENTORY MANAGEMENT

WITH BINARY SEARCH TREE & HASH TABLE

SOFTWARE DESIGN 4.2

A00258745

Meet Mehta

**ABSTRACT**

Inventory Management system allows user to manage all the inventory items in the stock. Every store has some inventories which it needs to manage in order to keep its business running. This system makes it easy for such stores to keep track of their inventories.

System is divided into two sections: BST & HashTable

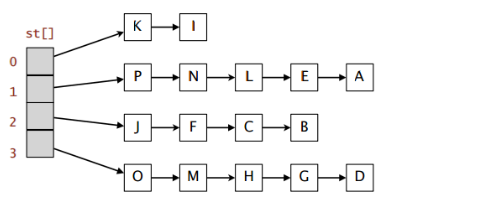
**A drawing of a cartoon character

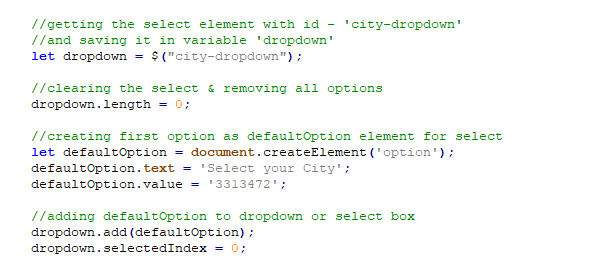
Description automatically generated**For BST :

1. **Add Item** : Add items in the inventory.
2. **Remove Item**: Remove item from the inventory.
3. **Display Items**: List of all items in the inventory.
4. **Search Item**: Search for any Item in the inventory.
5. **Save Items**: Save all the Items into the File.
6. **Read Items**: Read all the items from the File.
7. **Iterator** : find the occurrence of item in list

For HashTable :

1. **Add Item** : Add items in the inventory.
2. **Search Item**: Search for any Item in the inventory.
3. **Delete Item :** Delete item from the inventory.
4. **Iterator :** Iterate through the arrays of item.

****



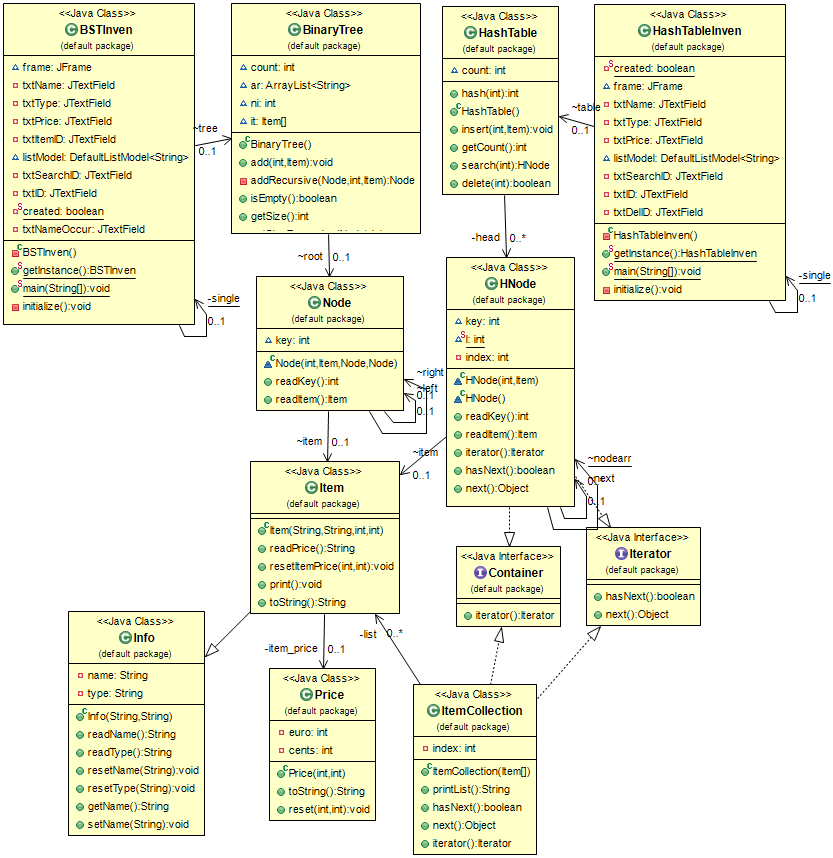
**INTRODUCTION**

Inventory Management System aims to ease the management of Inventories. First, user enters what he wants to manage, it may be books, groceries or even people. This modifies entire system based on user’s input. For eg. If user enters books, the system becomes book management system.

Some features of system include:

* Item’s detail includes Item’s Name, Item’s Type & Item’s Price.
* Insert, Delete and Search option for BST and Hash table.
* Performance of operation (returns count while searching).
* Inheritance - Class Item inherited from Class Info.
* Aggregation – Class Price used in Class Item.
* Items are stored in the node along with pointers (Node & HNode class).
* BST data structure and Hash table to store item.
* Save all the items in file using Serializable & OutputStream.
* Read all the items from the file using InputStream.
* Singleton Pattern (to open only one instance of GUI)
* Iterator Pattern (to iterate through the arrays of Items)

**CLASS DIAGRAM**

****

**Classes**

* Info - With variables Name & Type to store Item’s info
* Price - With variables euro & cents to store Item’s price
* Item - Extended from Info and added Price to create Item

For Binary Search Tree & HashTable:

* Node – Store the details of the Item.
* BinaryTree – Add, delete and list nodes in binary tree.
* BSTInven – GUI for BST operations.
* Hnode – Store the details of the Item.
* HashTable – Add, delete and list nodes in binary tree.
* HashTableInven – GUI for HashTable operations.

Interfaces

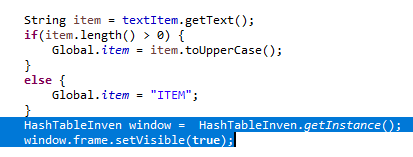
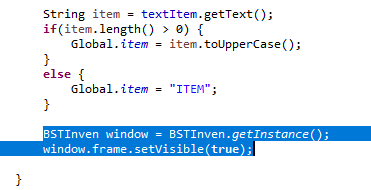
* Iterator and Container

For Iterator

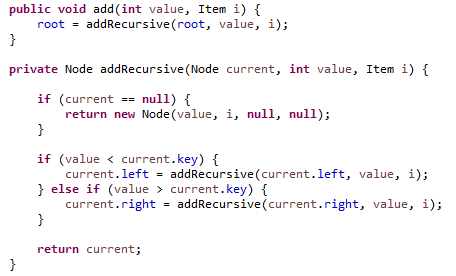
* ItemCollection –Class that implements Iterator and Container and helps us iterate through list of items.

**IMPLEMENTATION & FUNCTIONALITIES**

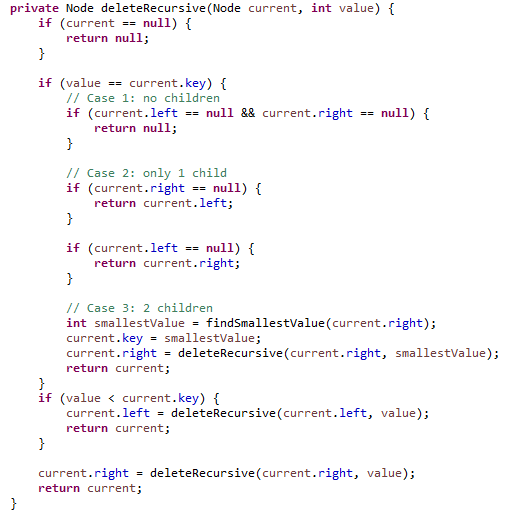
* **Load Screen for GUI:**
* Single Instance of BST & HashTable (using Singleton Pattern).



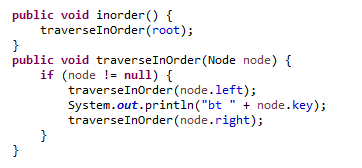
* **Binary Search Tree:**
* **Add Item as Node :** add() method takes id as value and insert it with Item.Recursively traverses the tree comparing the value, if value is less, it goes to left child otherwise goes to right child. If encounters null node, it inserts the node.



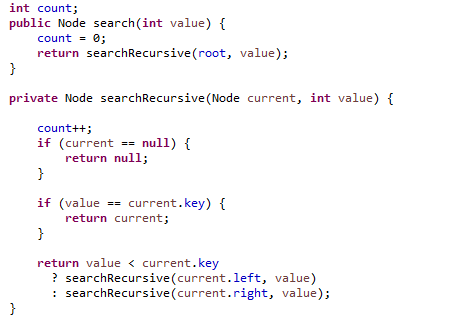
* **Delete item Node :** delete() method uses head node or current node to search node and delete it. if value is less, it goes to left child otherwise goes to right child. If encounters the node that contains the given key, it deletes that node.

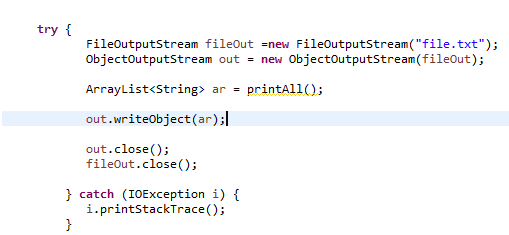


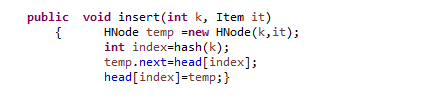
* **Display Item Nodes** : displays all nodes in the tree using Inorder Traversal of tree.



* **Search Nodes & performance** : searches recursively for the node in Tree with given value. After travelling each node, it increments count by 1 giving the no of steps taken to get the desired node. Avg Search Time Complexity : O(LogN).

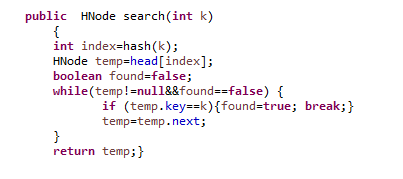


* **Save to File** (write ArrayList of Tree nodes into the file)
* **Hash Table:**
  + **Add Item as HNode:** insert the Item and key as a HNode in the hash table. First, calculate the hash of the key which gives us the position in the array where HNode is going to be placed. The first node at this position is index and all nodes in this position are stored as a linked list.



* + **Search Item** : search items in hash table using key. Finds the location in array using hash and then traverses all linked lists at that array position.

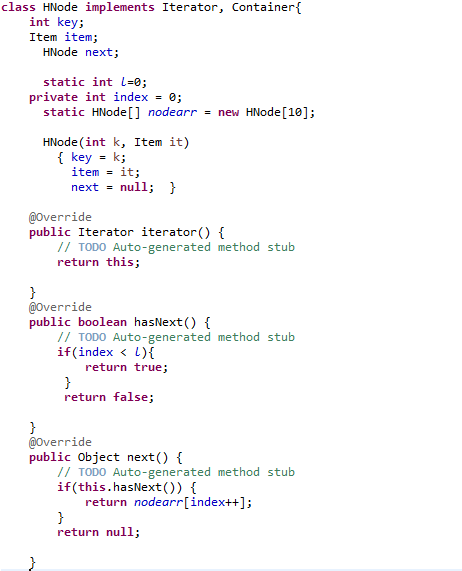
Avg Searching Time Complexity: O(1).



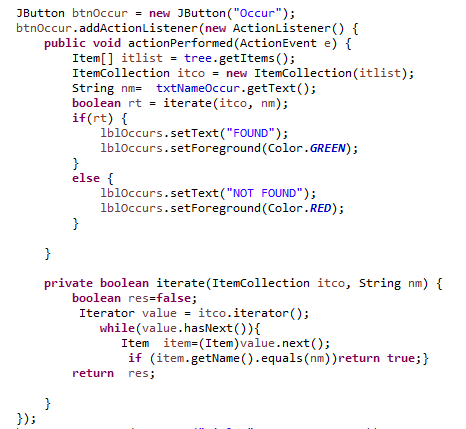
* A screenshot of a cell phone

  Description automatically generated**Delete Item** : (delete item in hash table using key)
* **Iterator Pattern to iterate through HNode of Hash Table:**

Implements Container and Iterator and provides hasNext(), next() to iterate through the HNodes in the hash table.

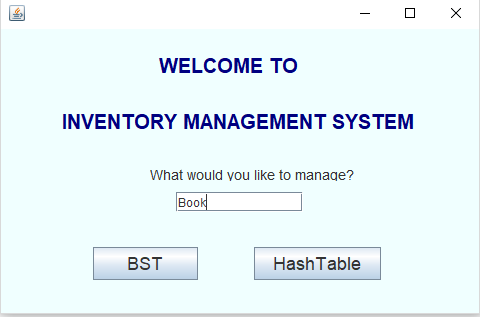


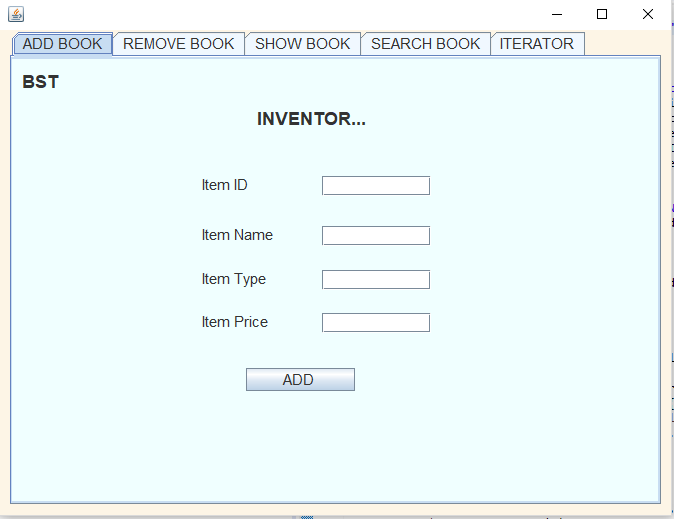
* Iterate through list and returns true if item is already present in the Binary Tree.

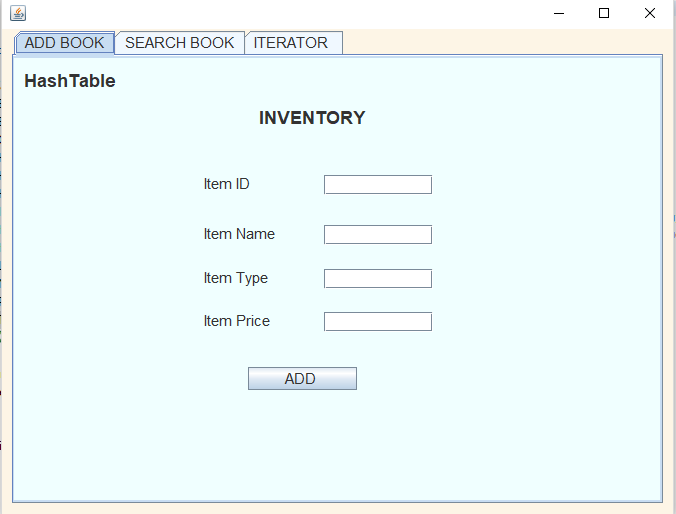


**SCREENSHOTS**

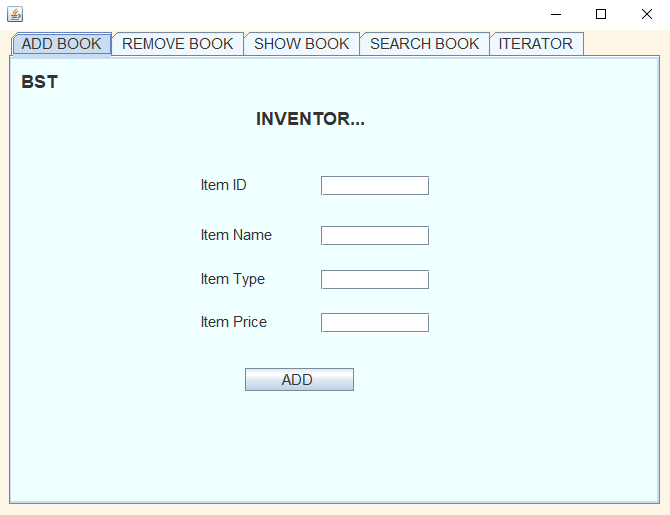
Opening Screen

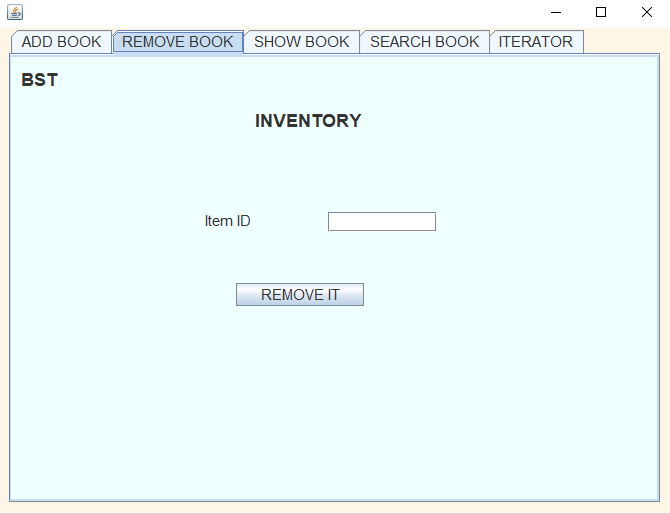
****

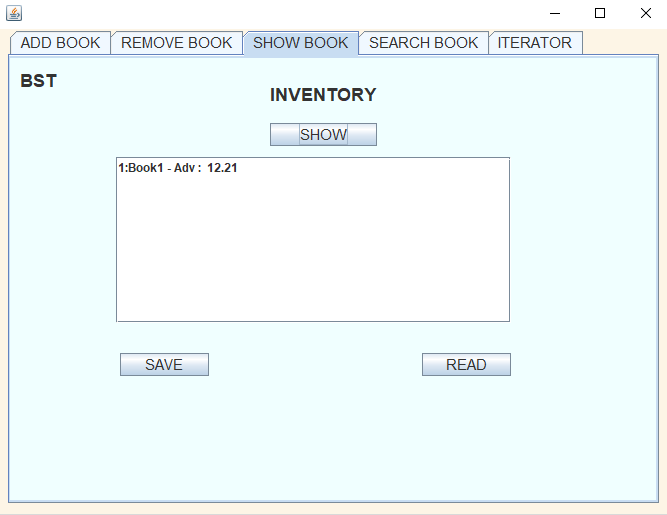
****

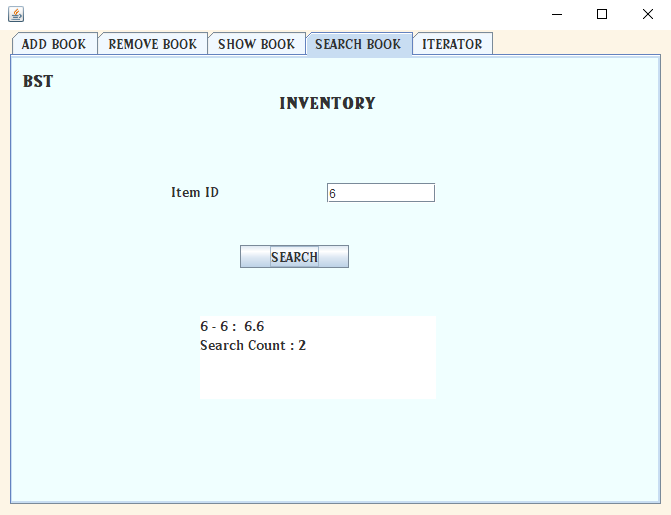
****

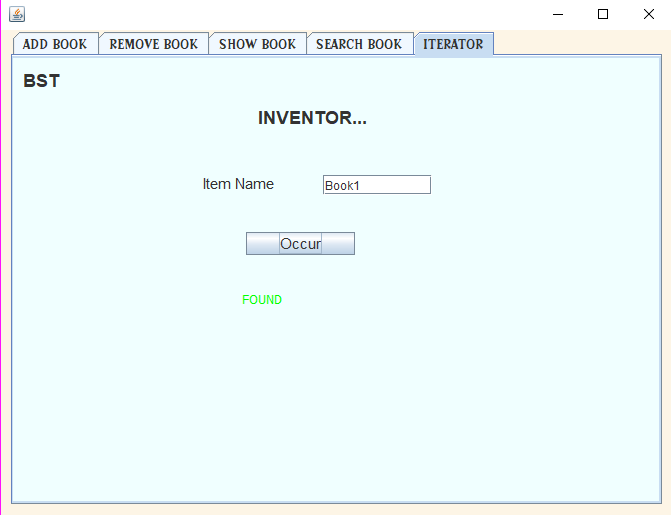
**Binary Search Tree**



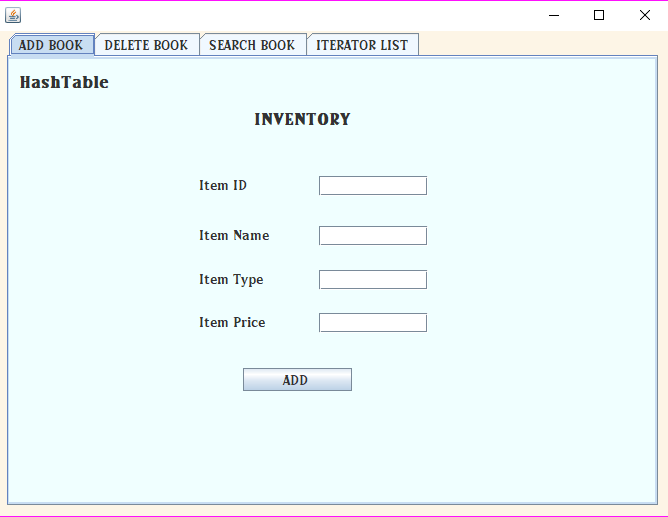


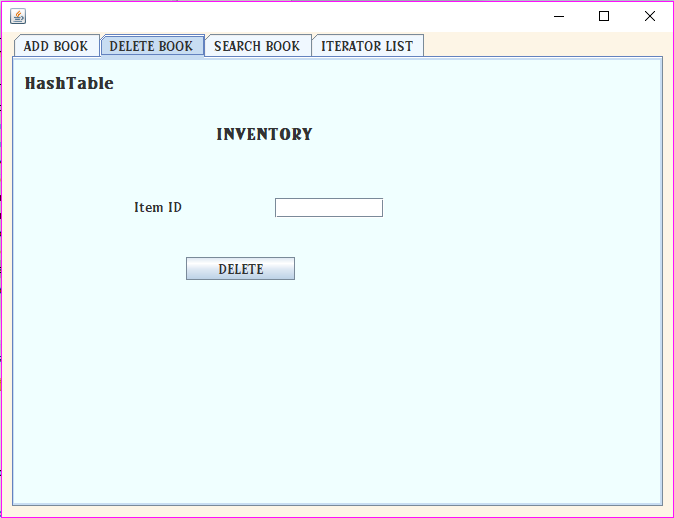


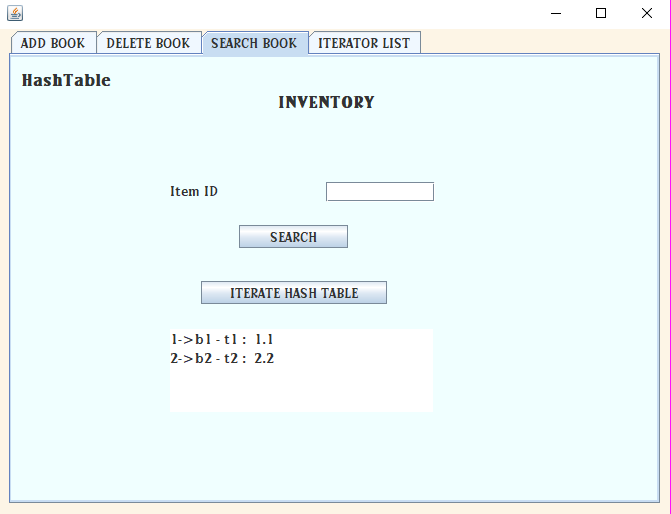


x`

**Hash Table**







**EXTRA FUNCTIONALITIES**

* Aggregation and Inheritance.
* Singleton Pattern : to open only one instance of frame at a time.
* Iterator Pattern in Hash Table: to Iterate through the HNode of Items.
* Iterator Pattern in BST: to check if item is already present in the binary search tree.
* Search Performance : gives no of steps required for searching an item.
* File IO : Saving List of Items in the file.
* File IO : Reading List of Items from the file.
* Serialization to write Nodes into the file.

**PROBLEMS ENCOUNTERED**

* Removing elements from HashTable : This was one of the major problems that I encountered. Since hash table stored the node as linked list in a specific array index. I had to traverse linked list on hashed index to delete the item.

After that, I had to make sure that item before deleted item in linked list is linked to the next item in line.

* Removing element from Binary Search Tree : It was easy to delete the last element i.e. leaf node from BST but deleting the node which had its own child was difficult task to accomplish.
* Implementing Iterator on HashTable : For this, I had to create the array of all HNode in the hash table than use iterator and collector interface to iterate through all the HNode in the array.
* Saving the Items (Serializing Object) : I wanted to store all the nodes in binary search tree into the file. For this, I traversed the binary tree in order and collected all nodes in ArrayList and saved it in the file.

**CONCLUSIONS & LEARNINGS**

I took my previous project and implemented the operations using Binary search tree and hash table. This improved my knowledge of data structures present under Collection framework.

I learned the implementation and inner working of binary search tree and hash table. Also got to learn the benefits of using JAVA design patterns like Singleton pattern and Iterator Pattern.

Performance comparison :

Hash Table is faster than BST. Time complexity for operations are

Inserting, Deleing & Searching in Hash Table is O(1) while BST is o(LogN).

This means that Hash Table is better to use over Binary search tree when we are concerned with these operations but there are also some advantages of using binary tree for e.g., sorting can be easily achieved using inorder traversal in binary tree while it is not that easy in hash table.

Overall, this project helped me a lot to improve my understanding of implementing data structures.