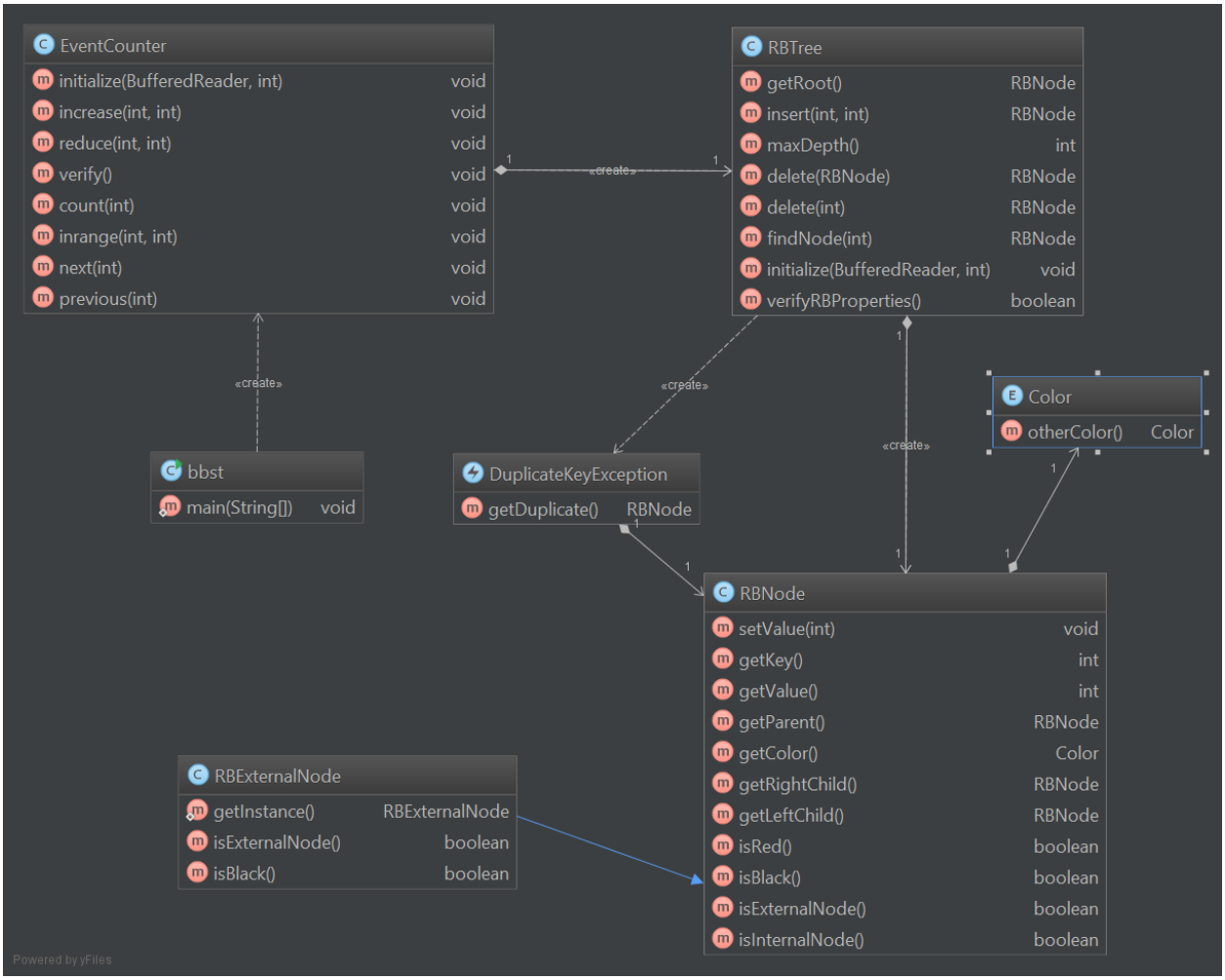
Project Report

Satish Erappa , UFID: 85 975 669

Compiler Used: 64 bit JDK version 1.8.0_65

UML Diagram representing the structure and function prototypes of the program:



Detailed description of function prototypes given in Appendix A

Program Structure:

EventCounter: Class that implements the functionality given in the project

RBTree: Class that implements the Red-Black tree.

RBNode: Class that represents the structure of a node in Red-Black tree

Color: Enumeration representing colors of Re-Black tree namely, RED and BLACK.

DuplicateKeyException: Exception that is thrown when an element with key already present in the Red-Black tree is tried to insert into the Red-Black tree.

bbst: The main class and starting point of the program.

The program begins with the main class bbst.java, it is the starting point of the program which deals with reading input file and commands. It parses the commands and calls corresponding functions in EventCounter. For example, to carry out command **increase**, the increase function in EventCounter is called.

EventCounter contains the instance to Red-Black tree (RBTree), it calls appropriate functions in the red black tree to implement the functionalities initialize, increase, reduce, count, inrange, next and previous.

The insert, delete, initialize (from a sorted list) are implemented in RBTree class. The functions increase, reduce and initialize in EventCounter use insert, delete and initialize functions in RBTree.

Note: To run the test file which contains 100 million entries, heap space for the java program should be increased to close to 8GB by using the below command. The heap space should be increased because the default heap size allocated for java programs is 1GB which is not enough memory to hold 100 million Red-Black Tree nodes.

$java **-Xmx8000m** bbst.java test-file.txt

Source files:

bbst.java

edu/ufl/ads/proj/event/*.java

edu/ufl/ads/proj/rbtree/*.java

edu/ufl/ads/proj/rbtree/generic/*.java

Appendix A: Detailed function prototypes

```java
/**
 * EventCounter class that implements the functionality given in the project
 */
public class EventCounter {

    /**
     * Initialize the event counter by reading the eventId and count from the reader
     * Complexity: O(n)
     * @param reader Reader to read the input key-value pair
     * @param size Number of key value pairs
     * @throws IOException, when read error occurs
     */
    public void initialize(BufferedReader reader, int size)throws IOException

    /**
     * Increase the count value of id by given count, if it is present otherwise
insert and print
     * the current value of count and print the current count value of the event for
the given Id.
     * @param id Id of the event
     * @param count The value by which the counter needs to be incremented.
     */
    public void increase(int id, int count)

    /**
     * Reduce the counter of given Id by count and print the current value, if the
count reduces to <=0 remove the
     * event with id from red black tree. If the node is removed print 0.
     * @param id Id of the event
     * @param count The value by which the counter needs to be decreased
     */
    public void reduce(int id, int count)

    /**
     * Prints whether the red black properties are satisfied and also prints the
maxDepth of tree
     */
    public void verify()

    /**
     * Print the count of the given Id, if the event with given id is not found print
0
     * @param id Id of the given event
     */
    public void count(int id)

    /**
     * Print the sum of count values of all the ids which are in range [id1, id2]
(inclusive)
     */
    public void inrange(int id1, int id2)

    /**
     * Prints the Id and count of the event such that Id of the event is greater than
the given id and least of all such Ids
     * Prints 0 0, if no such event is found
     * Complexity = O(lg (n))
     * @param id Id for which we need to next
     */
    public void next(int id)
```

```
    /**
     * Prints the Id and count of the event such that Id of the event is lesser than
the given id and greatest of all such Ids
     * Prints 0 0, if no such event is found
     * Complexity = O(lg (n))
     * @param id Id for which we need to find previous
     */
    public void previous(int id)
```

```
/**
 * Class to represent RedBlack tree with integer key and value types
 *
 */
public class RBTree {
    /**
     * Inserts the key-value pair into the red black tree
     * @param key Key to be inserted.
     * @param value Value to be inserted.
     * @return The the newly inserted RBNode
     * @throws DuplicateKeyException if there exists a node with same key in the tree
     */
    public RBNode insert(int key, int value) throws DuplicateKeyException

    /**
     * Returns the maximum depth of the red black tree
     * @return The maximum depth of the red black tree
     */
    public int maxDepth()

    /**
     * Delete the given RBNode
     * @param delNode Node to be deleted from redblack tree
     * @return Node deleted from reb black tree
     */
    public RBNode delete(RBNode delNode

   /**
    * Find the given key in Red-Black tree
    * @param key Key to be searched in Red-Black tree
    * @return RBNode with given key, RBExternalNode if it doesnot exist
    */
   public RBNode findNode(int key){
        return findNode(root, key);
    }

    /**
     * Initialize the RedBlack tree using a list of sorted KeyValue pairs, by building
the complete Binary Search tree in inorder
     * and coloring nodes such that only last level internal nodes are red and the
rest are black.
     * Time Complexity = O(n) { T(n) = 2T(n/2) + O(1) }
     * @param reader Reader to a stream which contains key-value pairs sorted by the
keys field.
     *               Each key-value pair is on a separate line; keys and values are
separated by a space.
     * @param size Number of key-value pairs
     */
     public void initialize(BufferedReader reader, int size)
```

```java
/**
 * Utility method to verify the properties of the RB tree
 * @return true if all the properties of RB tree hold, otherwise false;
 */
public boolean verifyRBProperties()
```