IMPORTANT SUBMISSION INSTRUCTIONS

You will be uploading your submission using the assignment server. The link is posted to cuLearn. A short video is posted there also in case you require instructions. You may submit as many times as you wish, but must wait at least 5 minutes between submissions (to protect the server). Your highest mark is kept. It would help us out if you try the upload early, even using just the unmodified assignment skeleton. That way potential problems can be found before the deadline. It will also ensure that YOU know how to submit your assignment well before the deadline. If your attempt to obtain your secret code is unsuccessful, please email me at darrylhill@cunet.carleton.ca. This may happen if you have registered late, etc.

You must adhere to the following rules *(as your submission will be subjected to automatic marking system):*

- ***Download*** *the compressed file* ***"comp2402a4.zip"*** *from cuLearn.*
- ***Retain the directory structure*** *(i.e., if you find a file in subfolder "comp2402a4", you must not remove it).*
- ***Retain the package directives*** *(i.e., if you see "package comp2402a4;" in a file, you must not remove it).*
- ***Do not rename any of the methods already present*** *in the files provided.*
- ***Do not change the visibility of any of the methods provided*** *(e.g., do not change private to public).*
- ***The main method of CountdownTree contains test code you may use and/or modify****. Do not modify any other classes or interfaces.*
- ***Upload*** *a compressed file* ***"comp2402a4.zip"*** *to the assignment server to submit your assignment as receive your mark immediately* ***(highest mark of all submissions will be your assignment mark).***

Please also note that ***your code may be marked for efficiency as well as correctness*** *– this is accomplished by placing a hard limit on the amount of time your code wil be permitted for execution. If you select/apply your data structures correctly, your code will easily execute within the time limit, but* ***if your choice or usage of a data structure is incorrect****, your code may be* ***judged to be too slow*** *and* ***it may receive a grade of zero****.*

You are expected to ***demonstrate good programming practices at all times*** *(e.g., choosing appropriate variable names, provide comments in your code, etc.) and* ***your code may be penalized if it is poorly written****. The server won't judge this, but in case of discrepancies, this will be evaluated.*

Instructions

Start by downloading the comp2402a4 Zip file from cuLearn, which contains a skeleton of the code you need to write. This assignment is about implementing a tree that supports partial rebuilding in order to keep it somewhat balanced. The folder should contain **10** files. **BinaryTree.java, BinarySearchTree.java, BinaryTreeNode.java, BSTNode.java, DefaultComparator.java** and **SSet.java** are the files needed to implement a **BinarySearchTree**. You will finish a child class of **BinarySearchTree,** the **CountdownTree** in a file **CountdownTree.java.**

In addition there are files **RangeSSet.java, SortedSSet.java,** and **Testum.java.** These classes are used in the testing code in **CountdownTree.java** in the **main** method. You should not modify these files. You may add, remove, or change the methods in the **main** method of **CountdownTree.java** if you wish.

Instructions: Exploding Trees

In this assignment, you will implement a countdown tree, in the file **CountdownTree.java**. This is a balanced binary search tree that uses partial rebuilding in a manner similar to scapegoat trees (Chapter 7 in Open Data Structures).

Every countdown tree has a float parameter **d**, defined when it is created. We call this the tree's **countdown delay factor**.

Each node **u** in a countdown tree has an int countdown timer, **u**.timer. When a new node is created, it's timer is set to **Math.ceil(d)**.

When a node **u**'s timer reaches 0, the node explodes and the entire subtree rooted at **u** is rebuilt into a perfectly balanced binary search tree (note that **u** is probably not the root of this new subtree.) Every node **v** in the rebuilt subtree has it's timer reset to **Math.ceil(d*size(v))**, where **size(v)** denotes the number of nodes in the subtree rooted at **v**.

The **add(x)** operation in a countdown tree works just like adding in a normal (unbalanced) binary search tree. A new node **u** containing **x** is added as a leaf in the tree and **u**.timer is set to **Math.ceil(d)** (implicitly, this is **Math.ceil(d*size(v)),** where **size(v)**=1). Next, every node on the path from **u** to the **root** of the tree has its timer decremented and, if any of these nodes' timers drop to 0, then their subtree explodes.

The **remove(x)** operation works just like in a normal (unbalanced) binary search tree. We first find the node **w** that contains **x**. Now, it might not be easy to remove **w** because it has two children so we try to find a node **u** that is easy to delete. If **w** has no right child, then we set **u=w**. Otherwise, we pick **u** to be the leftmost node in **w**'s right subtree'. Next we swap **u.x** and **w.x** and splice **u** out of the tree.

Finally, we walk back up the path from (the now removed) **u** to the root and decrement the timer of every node along the way. If any of these nodes' timers drop to 0, then their subtree explodes.

You don't need to do anything to implement the **find(x)** operation, it's inherited from the BinarySearchTree class and works unmodified.