Report for ADS Project

Name: Jiajing Liao
UFID: 01469951
UF Email: jiajingliao@ufl.edu

File and class preview:

MinHeap.java
- HeapNode
- MinHeap

RedBlackTree.java
- RedBlackTree
- TreeNode
- Color

risingCity.java
- risingCity

Steps or Ideas to solve this Project:

In this project, I implemented a MinHeap to extract min executed building and a Red Black Tree for PrintBuilding, which is a range query.
I implement the MinHeap in the MinHeap.java.
I implement the RBT in the RedBlackTree.java.
In the risingCity.java, it's the logic of this project. It reads the input and process it. It get the active building, and add executed Building by 1, update the active building every 5 days or whenever a building is finished. It also process the input into simple int, which is easy to process.

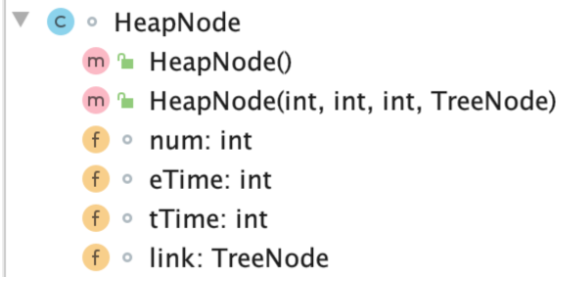
Class and function detailed Explanation:

MinHeap.java

Ideas:

1. using a MinHeap to Extract the minimal executed building
2. Implemented MinHeap by an array, finding child or parent by indexing
3. using heapify() to fix the MinHeap whenever ExtractMin()
4. there is a link in the node in MinHeap, which point to the node in RBT, whenever node in the MinHeap changes, the corresponding node in the RBT changes.
5. using a random array to test the correctness of MinHeap

HeapNode

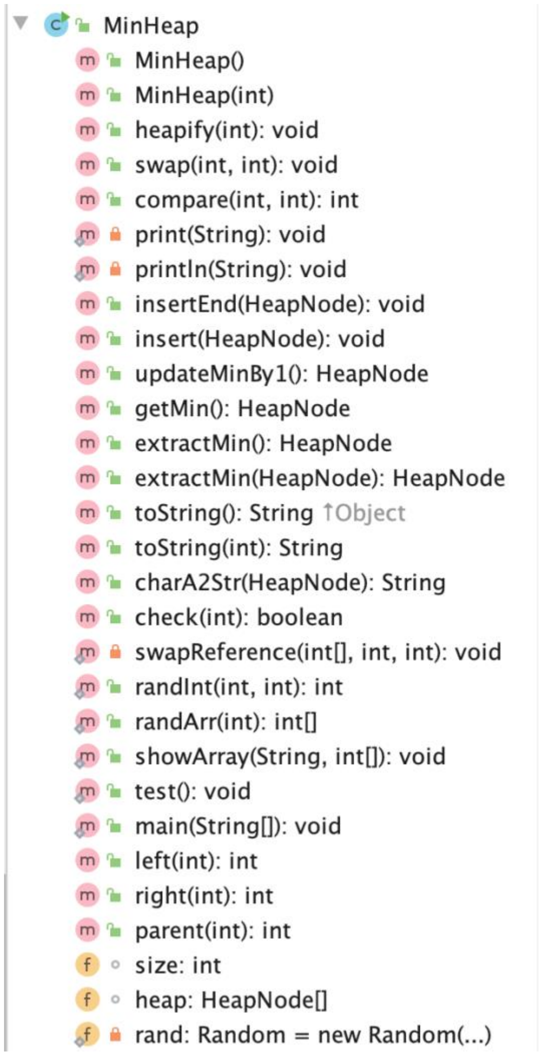


```
HeapNode
    HeapNode()
    HeapNode(int, int, int, TreeNode)
    num: int
    eTime: int
    tTime: int
    link: TreeNode
```

* HeapNode is a class for Node in the Heap
* it has 2 construction function
 * it has 4 fields:
 * num,
 * eTime is extended time,
 * tTime is total time
 * link is a variable point to a RedBlack Tree

Each building will be modeled as a HeapNode, then inserted into the Heap

MinHeap



```
MinHeap
    MinHeap()
    MinHeap(int)
    heapify(int): void
    swap(int, int): void
    compare(int, int): int
    print(String): void
    println(String): void
    insertEnd(HeapNode): void
    insert(HeapNode): void
    updateMinBy1(): HeapNode
    getMin(): HeapNode
    extractMin(): HeapNode
    extractMin(HeapNode): HeapNode
    toString(): String ↑Object
    toString(int): String
    charA2Str(HeapNode): String
    check(int): boolean
    swapReference(int[], int, int): void
    randInt(int, int): int
    randArr(int): int[]
    showArray(String, int[]): void
    test(): void
    main(String[]): void
    left(int): int
    right(int): int
    parent(int): int
    size: int
    heap: HeapNode[]
    rand: Random = new Random(...)
```

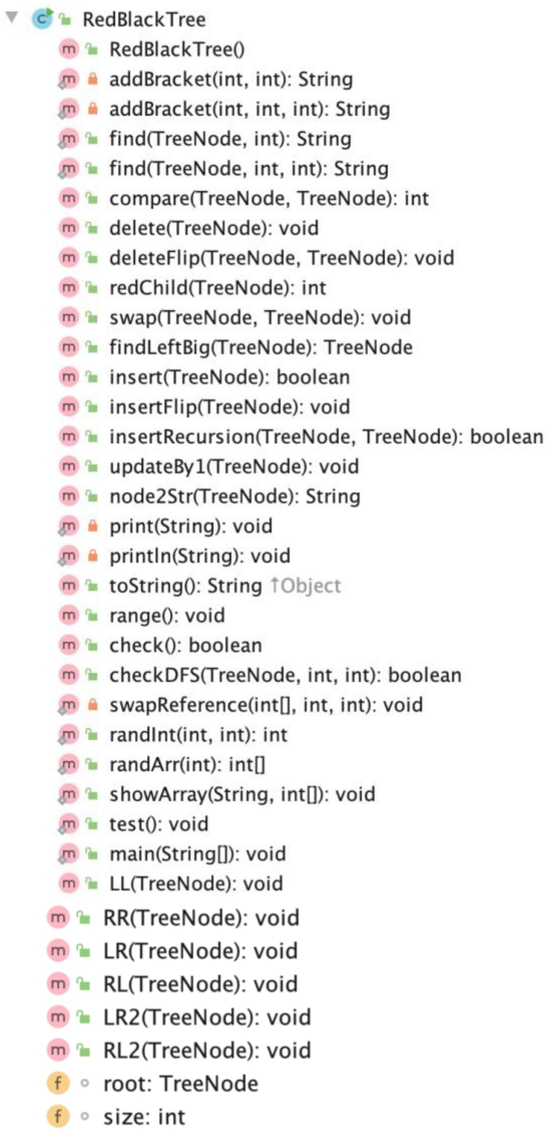
insert() can insert a building into the Heap, then the extractMin will get the currently active building,
extractMin() can get the minimal executed building.
test() using a random array to test the correctness of MinHeap

RedBlackTree.java

Ideas:

1. using a Red Black Tree to implement the function of PrintBuilding()
2. Implemented Red Black Tree by extending classic Tree Implementation.
3. add a field named color, which can be red or black.
4. insert a node as red, the fix the RBT by rotating and color fliping
5. using rotating and color fliping when deleting to fix the RBT
6. using random array to test the correctness of RBT

RedBlackTree



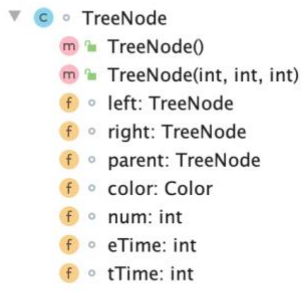
```
RedBlackTree
    RedBlackTree()
    addBracket(int, int): String
    addBracket(int, int, int): String
    find(TreeNode, int): String
    find(TreeNode, int, int): String
    compare(TreeNode, TreeNode): int
    delete(TreeNode): void
    deleteFlip(TreeNode, TreeNode): void
    redChild(TreeNode): int
    swap(TreeNode, TreeNode): void
    findLeftBig(TreeNode): TreeNode
    insert(TreeNode): boolean
    insertFlip(TreeNode): void
    insertRecursion(TreeNode, TreeNode): boolean
    updateBy1(TreeNode): void
    node2Str(TreeNode): String
    print(String): void
    println(String): void
    toString(): String ↑Object
    range(): void
    check(): boolean
    checkDFS(TreeNode, int, int): boolean
    swapReference(int[], int, int): void
    randInt(int, int): int
    randArr(int): int[]
    showArray(String, int[]): void
    test(): void
    main(String[]): void
    LL(TreeNode): void
    RR(TreeNode): void
    LR(TreeNode): void
    RL(TreeNode): void
    LR2(TreeNode): void
    RL2(TreeNode): void
    root: TreeNode
    size: int
```

find() is the range query for PrintBuilding

insert(), insert a node into RBT, it will then call insertRecursion for recursively insert and insertFlip for rotating and color fliping to fix RBT

delete() delete a node from the Tree, it will then call deleteFlip() to fix the RBT by rotating and color flip.
test() using a random array to test the correctness of MinHeap

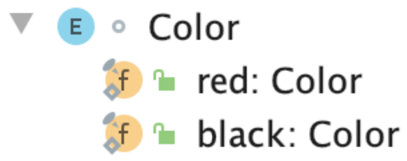
TreeNode



```
TreeNode
    TreeNode()
    TreeNode(int, int, int)
    left: TreeNode
    right: TreeNode
    parent: TreeNode
    color: Color
    num: int
    eTime: int
    tTime: int
```

TreeNode is a extended Node for RBT, it has 2 construction functions

Color



```
Color
    red: Color
    black: Color
```

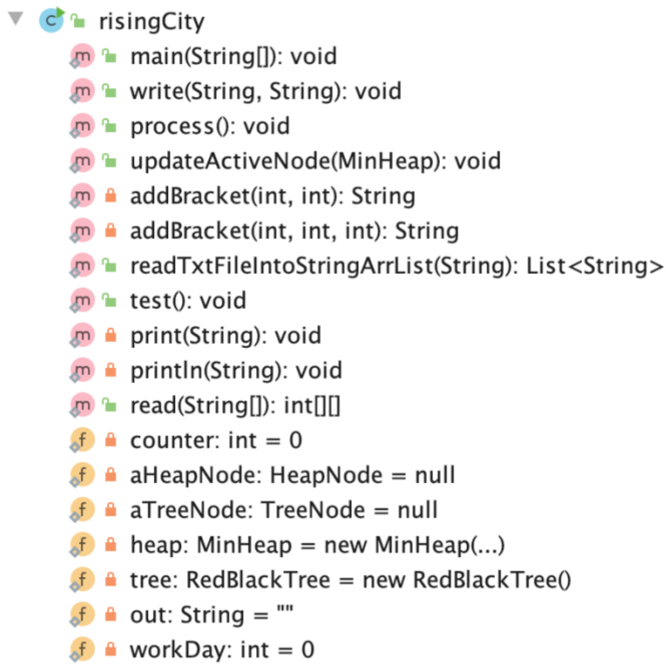
Color defines 2 color the RBT can have: red or black

risingCity.java

Idea:

1. It reads the input and process it.
2. It get the active building, and add executed Building by 1,
3. update the active building every 5 days or whenever a building is finished.
4. It also process the input into simple int, which is easy to process.
5. even if there is no input, the process will output all the building until all of then are finished.
6. after everything is done, it will output the file.

risingCity



```
risingCity
    main(String[]): void
    write(String, String): void
    process(): void
    updateActiveNode(MinHeap): void
    addBracket(int, int): String
    addBracket(int, int, int): String
    readTxtFileIntoStringArrList(String): List<String>
    test(): void
    print(String): void
    println(String): void
    read(String[]): int[][]
    counter: int = 0
    aHeapNode: HeapNode = null
    aTreeNode: TreeNode = null
    heap: MinHeap = new MinHeap(...)
    tree: RedBlackTree = new RedBlackTree()
    out: String = ""
    workDay: int = 0
```

write() is for writing file
main() contains the main part of processing
process() is everyday process
updateActiveNode() can update active node after 5 days work or whenever a building is finished.
read() for reading and processing the input data.