ADS Project 1 Report

Sahil Garg(UFID : 3191-8127)
(sahilgarg@ufl.edu)

Source code consists of following files:

1. **Building.java :** *Domain object defining the building specification(building no.,executedTime,totalTime)*
2. **MinHeap.java :** *Defines MinHeap dataStructure*
3. **RedBlackNode :** *Defines RedBlackTree Node*
4. **RedBlackTree :***Defines RedBlackTree dataStructure*
5. **risingCity.java :** *Defines main program defines the project workflow(main program)*

Classes Structure:

Building.java

Instance variables:

- `long buildingNumber` :*unique id which defines the building number*
- `int executedTime`:*defines the time spent on the building so far.*
- `int totalTime`:*defines the total time needed to complete the construction.*
- `RedBlackNode redBlackNode`:*pointer to the redBlackTree node representing the same building*

Functions:

- `getRedBlackNode()`:*returns the Red Black node representing the building.*
- `setRedBlackNode(RedBlackNode redBlackNode)`: *set the pointer in the Red Black tree.*
- `getBuildingNumber()`: *set the building number in the building object.*
- `getExecutedTime()`: *returns the time spent on the building so far.*
- `setExecutedTime(int executedTime)`:*updates the time spent on the building.*
- `getTotalTime()`: *returns total time needed to construct the building.*
- `compareTo(Object o)`: *To compare two buildings on the basis of executed time and if the executed time is same , compare building numbers.*

`RedBlackNode.java`

Instance variables:

- `Building data` : *defines the red black tree node data i.e building object*
- `RedBlackNode parent`: *defines the parent node of red black tree node*
- `RedBlackNode left`: *defines the left child of red black tree node*
- `RedBlackNode right`: *defines the right child of red black tree node*
- `int color`: *defines the colour of the red black tree node(1->Red,0->Black)*

Functions:

- `getData()`: *returns the data containing the red black tree node(i.e building object in our case)*
- `getLeft()`: *returns the left child of this node*
- `getRight()`: *returns the right child of this node*

`RedBlackTree.java`

This class defines the red black tree. A red-black tree follows the binary search tree properties with each node as either red or black and follows that the no. of black nodes from root to any external node node should be same and also there shouldn't be any two consecutive red nodes.

Instance variables:

- `RedBlackNode root` : *defines the root node of the red black tree*
- `RedBlackNode TNULL` : *defines the external node of the red black tree*

Functions:

- `RedBlackNode getRoot()`:*returns the root node of the red black tree.*
- `RedBlackNode getTNULL()`:*returns the external node of the red black tree.*
- `void insert(RedBlackNode node)`:*inserts the node into the red black tree by following the binary search property and making the colour of inserted node to be red .*

- **`void fixInsert(RedBlackNode insertedNode):`** *After inserting the new node , fixInsert is called in order to maintain redBlackTree property mentioned above. Following cases are handled in order to make sure the redBlackTree properties are satisfied:*
  - **Case 1:** *node's uncle is red.*
  - **Case 2:** *node's uncle is black and node is a right child.*
  - **Case 3:** *node's uncle is black and node is a left child.*
- **`RedBlackTree():`** *Default constructor of red black tree.*
- **`RedBlackNode minimum(RedBlackNode node) :`***returns the building having minimum building number .*
- **`void leftRotate(RedBlackNode x):`** *rotate the node left and change the color of the nodes accordingly.*
- **`void rightRotate(RedBlackNode x):`** *rotate the node right and change the color of node accordingly.*
- **`RedBlackNode searchTree(int k):`** *search a building with the given building number in the tree.*
- **`RedBlackNode searchTreeHelper(RedBlackNode node, int key):`***helper function to search a particular building number in the tree using binary search tree property.*
- **`void deleteNode(RedBlackNode deleteNode):`** *remove the given redblack node from the tree.*
- **`void fixDelete(RedBlackNode x):`** *After deleting the node , fix the red black tree in order to maintain red black tree property . Following cases are handled:*
  - **Case 1**: *node's sibling is red.*
  - **Case 2**: *node's sibling is black, and both of the sibling's children are black.*
  - **Case 3**: *node's sibling is black, sibling's left child is red, and sibling's right child is black.*
  - **Case 4**: *node's sibling is black, and sibling's right child is red.*
- **`void rbTransplant(RedBlackNode u, RedBlackNode v):`** *When a node is deleted, it links the node's child (if any) to node's parent (if any).*
- **`List<Building> searchTree(RedBlackNode node, long buildingNumber1, long buildingNumber2, List<Building> outputList):`** *returns the list of redblack tree nodes having building numbers between buildingNumber2 and buildingNumber1.*

`ReadFileHelper.java`

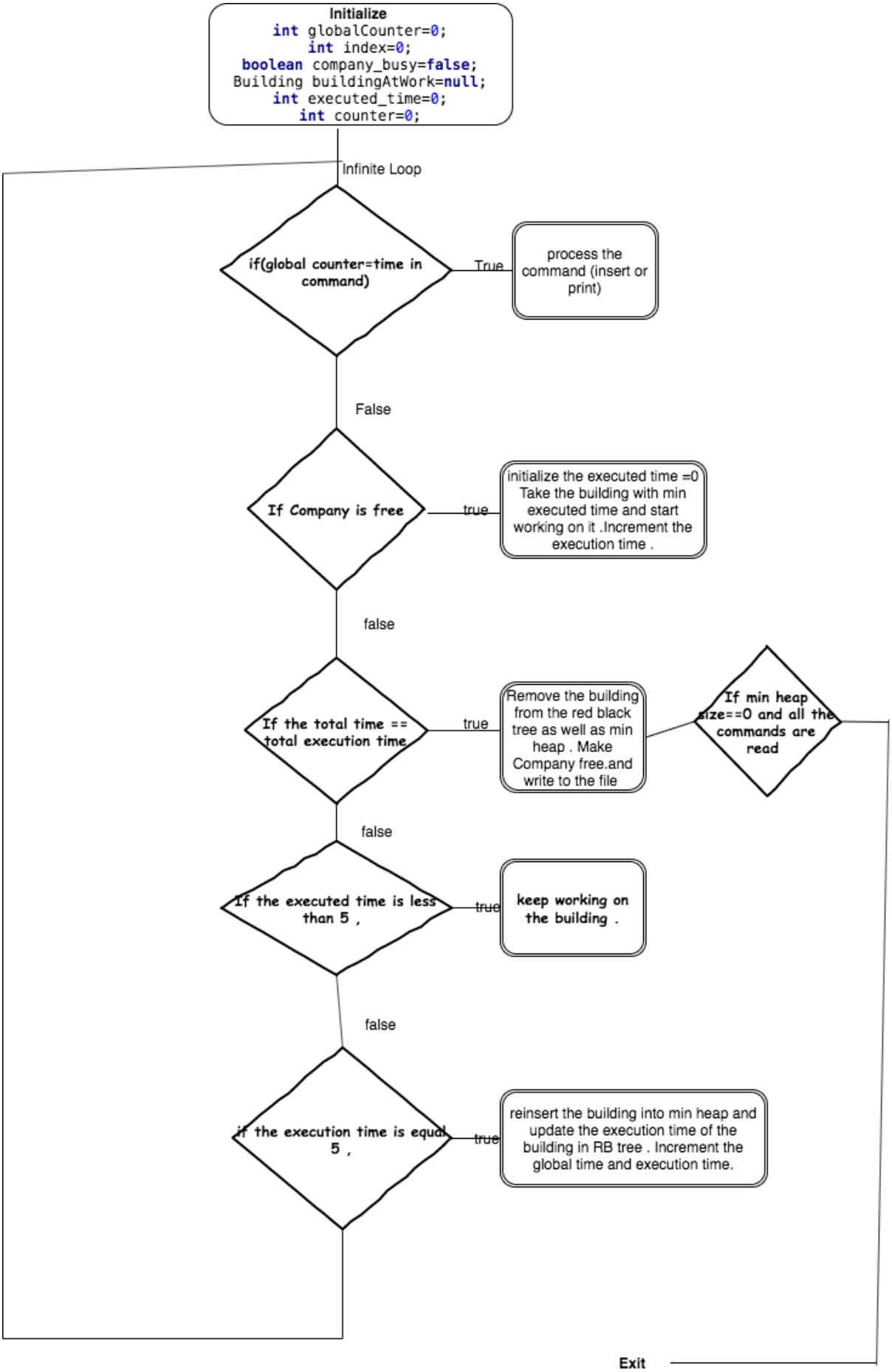
Functions:

- **`List<String> readFile(File inputFile):`***Read file function to read the input file and write the commands into a arrayList and return it to the calling function .*

`risingCity.java`

Functions:

- **`main(String[] args):`** *Main function which defines the workflow of the program ,which defines how the wayne construction company works(what process it follows in order to complete the construction process).Please follows the below flow diagram :*


Initialize
int globalCounter=0;
int index=0;
boolean company_busy=false;
Building buildingAtWork=null;
int executed_time=0;
int counter=0;
Infinite Loop
if(global counter=time in command)
True
process the command (insert or print)
False
If Company is free
true
initialize the executed time =0 Take the building with min executed time and start working on it .Increment the execution time .
false
If the total time == total execution time
true
Remove the building from the red black tree as well as min heap . Make Company free.and write to the file
If min heap size==0 and all the commands are read
false
If the executed time is less than 5 ,
true
keep working on the building .
false
if the execution time is equal 5 ,
true
reinsert the building into min heap and update the execution time of the building in RB tree . Increment the global time and execution time.
Exit

Steps to Run the project :

- Unzip the project directly and place the input file inside the folder.
- Run following commands in sequence inside the extracted folder:
  - make
  - java risingCity input_file_name

After running the above commands,a output file will be created inside the folder with name "output_file.txt"