The assignment required the creator to make the **LSApp** application to read data from a text file , store it in an array then search specific data in the array and display it .The creator was also supposed to create a **LSBSTApp** application that will carry out the same function as the **LSApp** but instead of storing the data in an array it is stored in a Binary Search tree. The purpose of this assignment is to compare the efficiency of a Binary Search Tree to an Array in their search operations. Inorder to use some functions the following java libraries were imported: java.io.File , java.io.FileNotFoundException, java.util.ArrayList and java.util.Scanner.

Array

A public class **LSApp** was created which has a main method, a printAllAreas and printArea methods which are all public. The printAllAreas and printArea are all to be called into the main method to display the desired areas.

printAllAreas

this method is called into the main method to print out all areas. It is invoked if no arguments are passed to the args array as input. In this case it accepts an array of type string as its parameter which is the array that contains all the test file data. The array is going to be looped through using a for loop and all data at each array index is going to be printed at every line. This is done using the System.out.println() statement which allows data to be printed one line after the other.

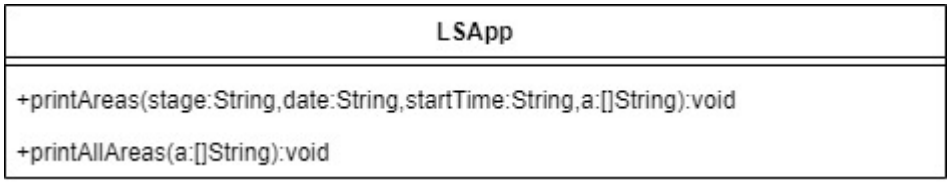
printArea

This method accepts 4 parameters in this case. 1st variable is the load shedding stage, 2nd variable is the load shedding day, the 3rd variable is the start time of the load shedding schedule and the 4th variable is the array that is storing all the text file data that is to be searched through. A variable key was created this variable stores the stage, day and startTime in the format that it is used in the text file hence the use of the concatenation function. The key variable will make the search process easier since the load shedding data is in this format “stage_day_startTime Arear” which is the format it has just been put in. A Boolean variable flag was created which is used to reflect whether the desired search data is found or not. The *String.contains()* method was used to search for the key in the array .It was checked If the LoadShedding data on that array position contains the key String if it does then the Boolean variable is set to true.If area is found flag is set to true if it is not it remains as false. An operation count variable is declared in this method to count how many operations are done in order to find a certain search area in the area. A for loop is used to loop through the array. The variable index is used to find the index of the last “_” (underscore) character so that the text line Load Shedding data can be split to access the areas only. the *String. contains* method to check if the search data is at the current array index if it is flag is set to true and the areas are printed using t*he System.out.println()* command. In order to access the area from the string line the *String. Substring* method is used. The substring index is specified to specify which position to start from. If it is not on the that position the program counties to loop through the array to the last index of the array .If the last index of array is reached the loop is terminated and the flag is checked if its false if it is the statement “Area not found” is printed to mean the search data supplied is not in the array.

Main

This method accepts a string array as a parameter. The scanner is used to read from the text file. A variable file of type scanner is created it accepts a *new file ()* as its instance variable while new file accepts the file path as its instance variable. The following format is used to read a text file using a scanncer: S*cannner variable_name=new Scanner(new File(pathname));* A try catch was used to catch any expections that might occur in reading the file example a FileNotFound Exception which normally happens when the user tried to open a file that does not exist. An error messege is displayed in the case that an error occurs. An ArrayList is created in order to find the amount of Load Shedding data in the text for the creation of the array . While loop used to store the text file data into the arraylist. After storing the data in the array list a String array is used of size arrayList size. A for loop was used to store the Load Shedding text file data from the arrrayList into the String array as per assignment requirement. If the array argument is null then the *printAllAreas()* method is called. Else the *printAreas(stage,day,startTime)* method is called.

Below is the class diagram for the LSApp :



Binary Search Tree

The next part of the assignment required the use of a binary search tree to store the text file LoadShedding data .In this case a LoadS object class was created that holds the *key (stage, day , startTime)* and the area for the load shedding schedules as instance variables . A LoadS constructor was made to initialize the object. A *toString()* method was created to enable the display of the load shedding text file data as it is instead of their memory address. The *toString*() method enables objects to be called by values not by reference. The *getKey()* method was created that returns the current LoadS object key. The *getArea()* method was created that returns the current LoadS object area. The *compareTo()* is overridden which is a feature that allows a subclass to provide specific implementation of a method that is already provided in this case it is already provided in the String class. The method is used to compare the current LoadS key and the parameter provided LoadS object key it then returns a negative 1 is it is smaller that the key if it is equal it returns a zero if it is larger it returns a positive 1.The Binary Search Tree, BinaryTreeNode and the BinaryTree classes used were downloaded from vula **.LSBSTApp** class has three methods like the **LSBTApp** the main method , *printAllAreas* and the *printArreas()* method.

printAllAreas

The printAllAreas method in this case it takes in the BinarySearchTree of type LoadS as a parameter the method you first put the BinarySearchTree in order by calling the *inOrder()* method it traverses the whole tree there by printing the each node that is visited.

printAreas()

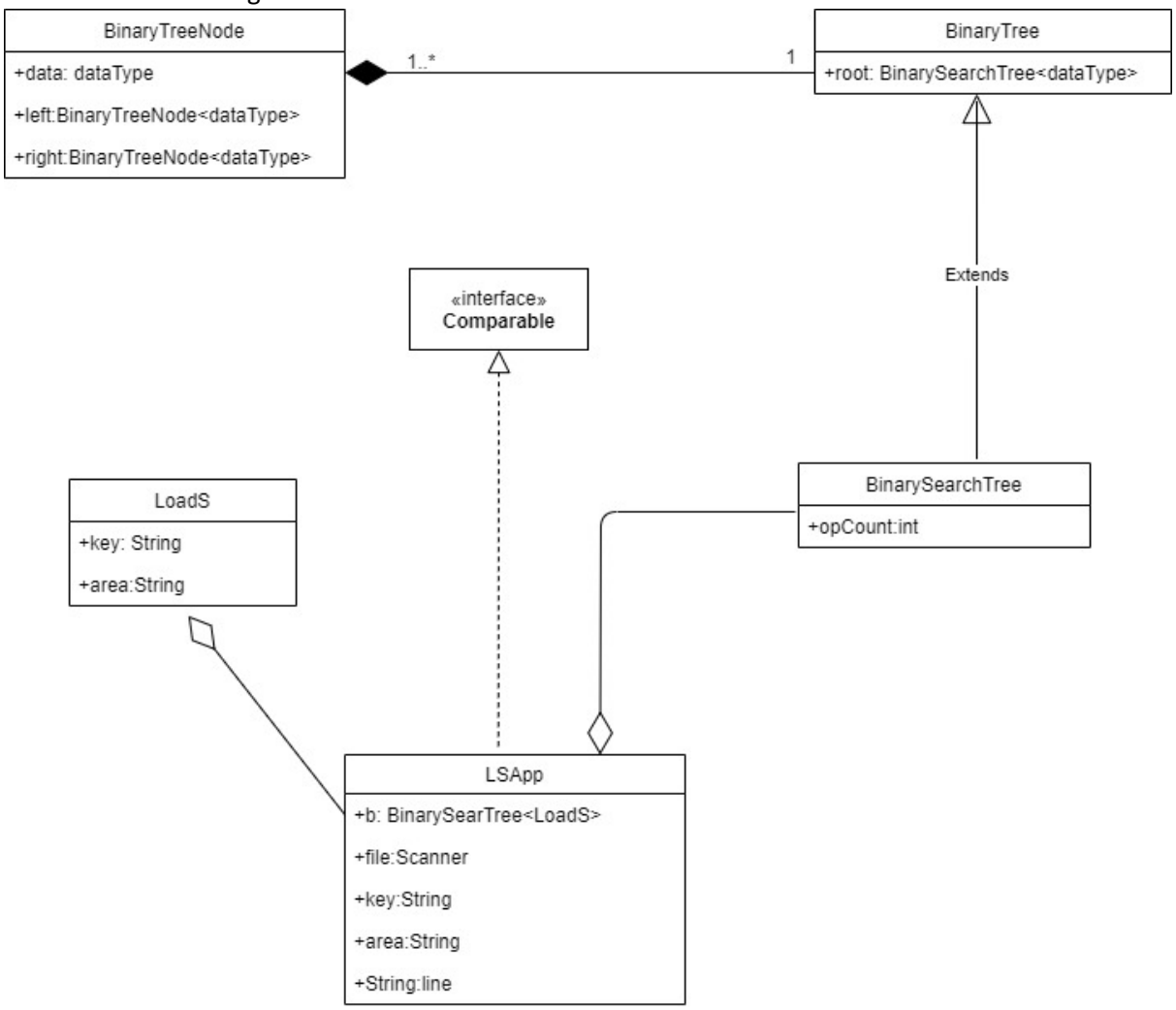
The *printAreas()* method it takes in the stage, day,startTime and BinarySerch tree of type Loads as its parameters. The key which is an instance variable is first created by assigning the String parameters in the format they are supposed to be in the LoadShedding text file. A LoadS object is created and the key created is assigned and area is left as null since our key is the search value item that was being looked for .A Boolean variable was created that is an indicator whether any *LoadS.key* values that meet the criteria are found or not. To find the areas that meet the criteria the *. find ()* method in the BinarySearchTree class is called. The methods accepts the LoadS object as its parameter then find the key that meets the search data by evoking the *compareTo()* method in the LoadS class. The method returns the data in a form of a binaryTreeNode, so if it is empty that means no data was found and the message was printed if it was not empty the areas are printed that meet the search criteria*. String.substring()* method was used to print the areas only instead of the whole Binary tree node.

Main

In the main method a binarySearchTreee of type LoadS is created. Variables necessary for the reading of the file and inserting of LoadShedding text file data are declared. The file is opened and read into a Scanner variable named file. A while loop is used to loop through the file and assign each line to a String variable line. *String.substring* is used to access the key and area which are instance variables of the LoadS class object. A LoadS class object was created, and the key and area are initialized. The created LoadS object was inserted into the BinarSearchTree. The loop keeps on iterating until there are no more text lines in the Scanner file variable. The try catch was there to catch any error that might occur in reading the file or looping through the file variable. The method then checked if there are any arguments passed to the *main method()* if there was not any the *printAllArea()* method was called to print all the LoadShedding areas that were in the text file .Else the *printAreas()* method would be called to print the areas that meet the key search criteria.The

method *getOpcount()* is called from the Binary Search Tree to display how many operations it takes to do the find operation.

Below is the class diagram of the above :



Comparing Arrays with BinarySearchTrees

Data Structure			
Binary Search Tree	Average	O(logn)	Worst Case -O(n)
Array	Unsorted	O(n)	Sorted-O(n)

Data Structure	Input	Number Of Operations	Output
Binary Search Tree	“1” “1” “00”	6500	1
Array	“1” “1” “00”	2976	1

The above results shows that the sorted array has a smaller number of compare operations than the Binary Search Tree since it only does one comparison then its done, unlike in a Binary Search Tree it first has to check the root node then traverse the tree checking each left and right node until the desired node is found.