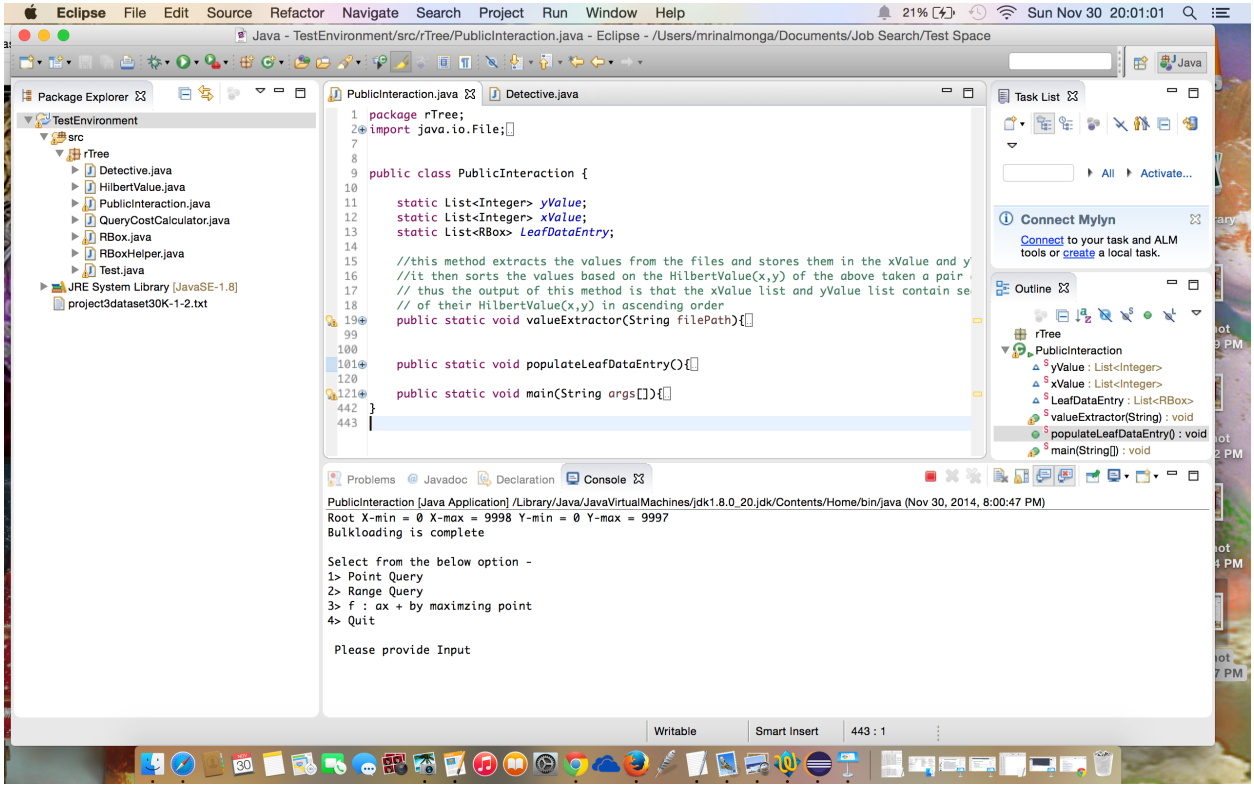
Database Implementation System
Assignment 3
Mrinal Monga

Minimum System Requirements

1. Free RAM space of at least 60 MB
2. Developed and Tested using **Java 8 SDK**

Compilation Instructions -

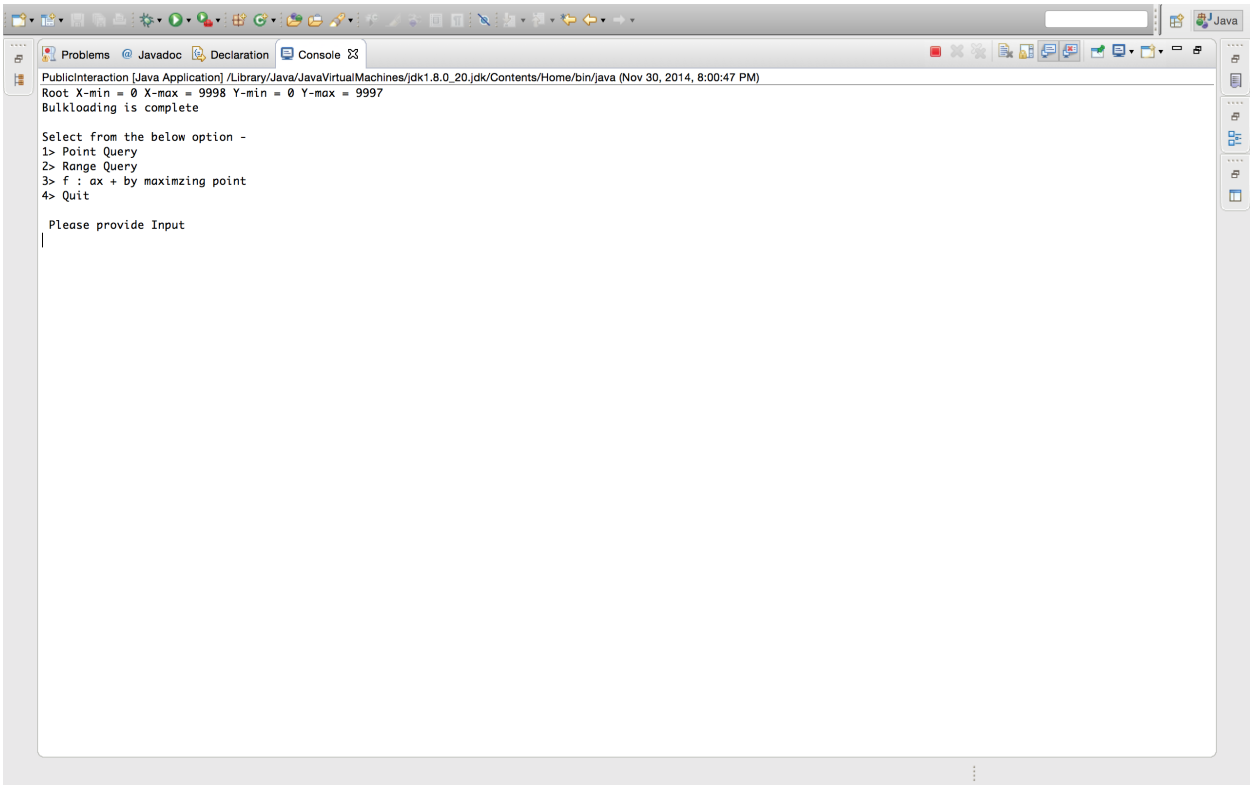
1. The zip folder is a java project. Use the java project to run the program. The class which must be run is the **PublicInteraction.java** file. This class is responsible for interacting with the user. It is also responsible for bulkloading R-Tree. Please take great care to note the location of the file project3dataset30K-1-2.txt which contains the dataset. **Please carefully note the location of project3dataset30K1-2-.txt with respect to the package rTree**.
2. Once you have a Java Project in the Eclipse as shown in the image below run the file PublicInteraction.java



3. Once you run the PublicInteraction.java program you would see the in the console the following output.

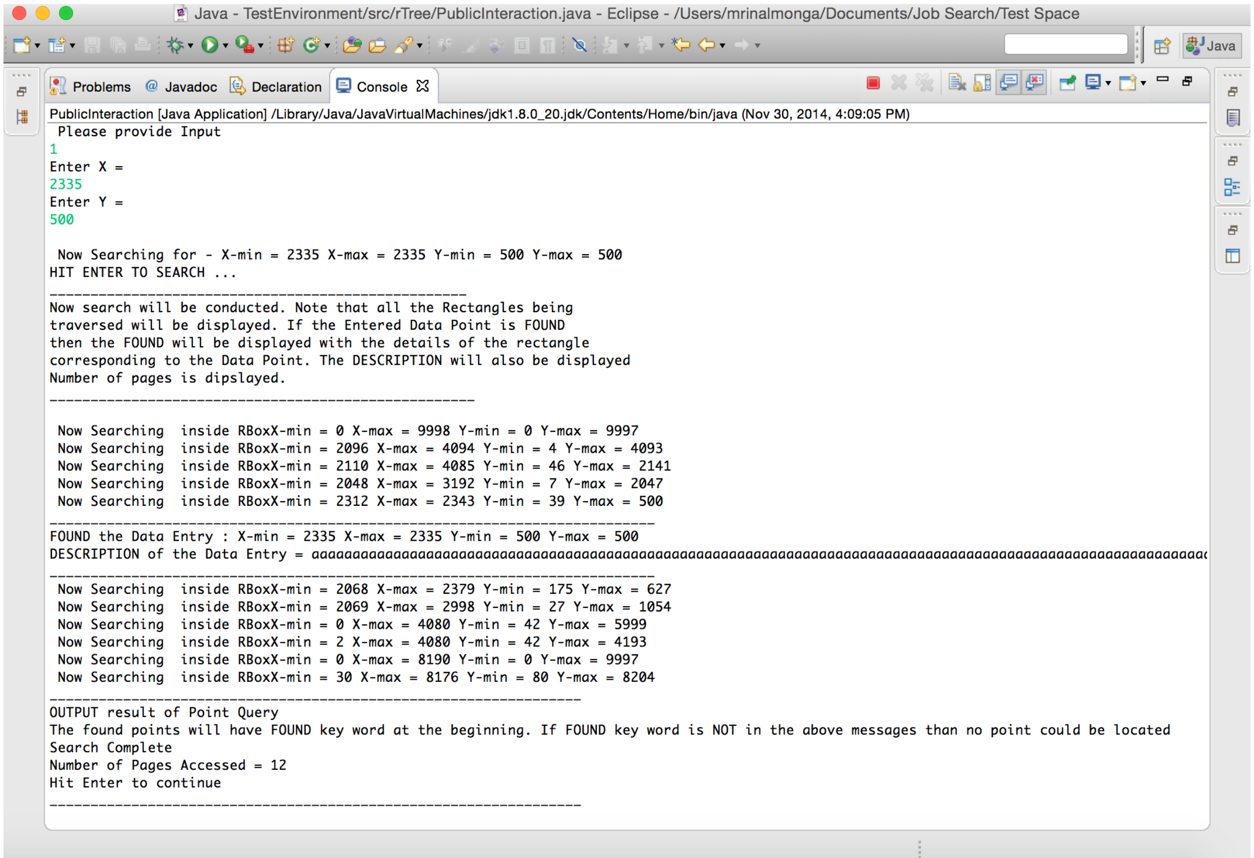
User Interaction

1. The following screen is seen



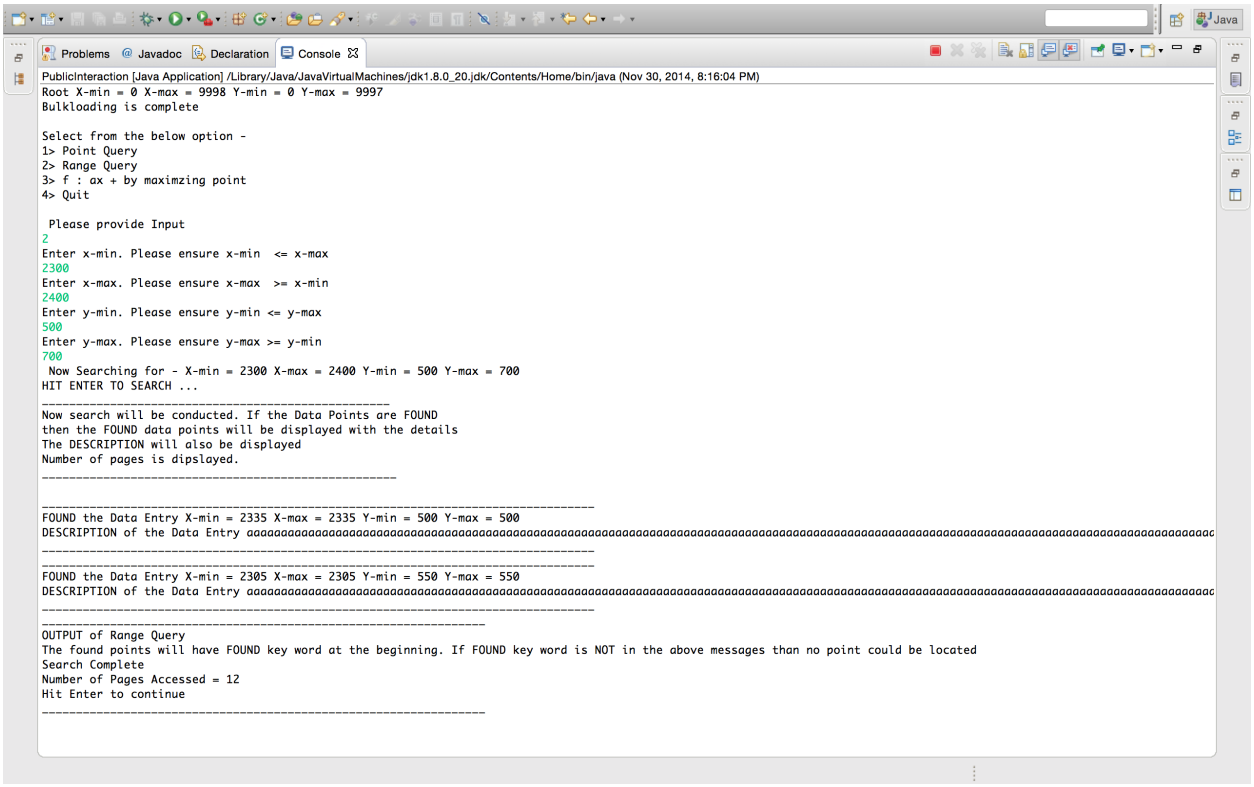
2. Note carefully the options it provides. 1 for Point Query 2 for Range Query and so on.
3. Some sample inputs outputs are given.

POINT QUERY



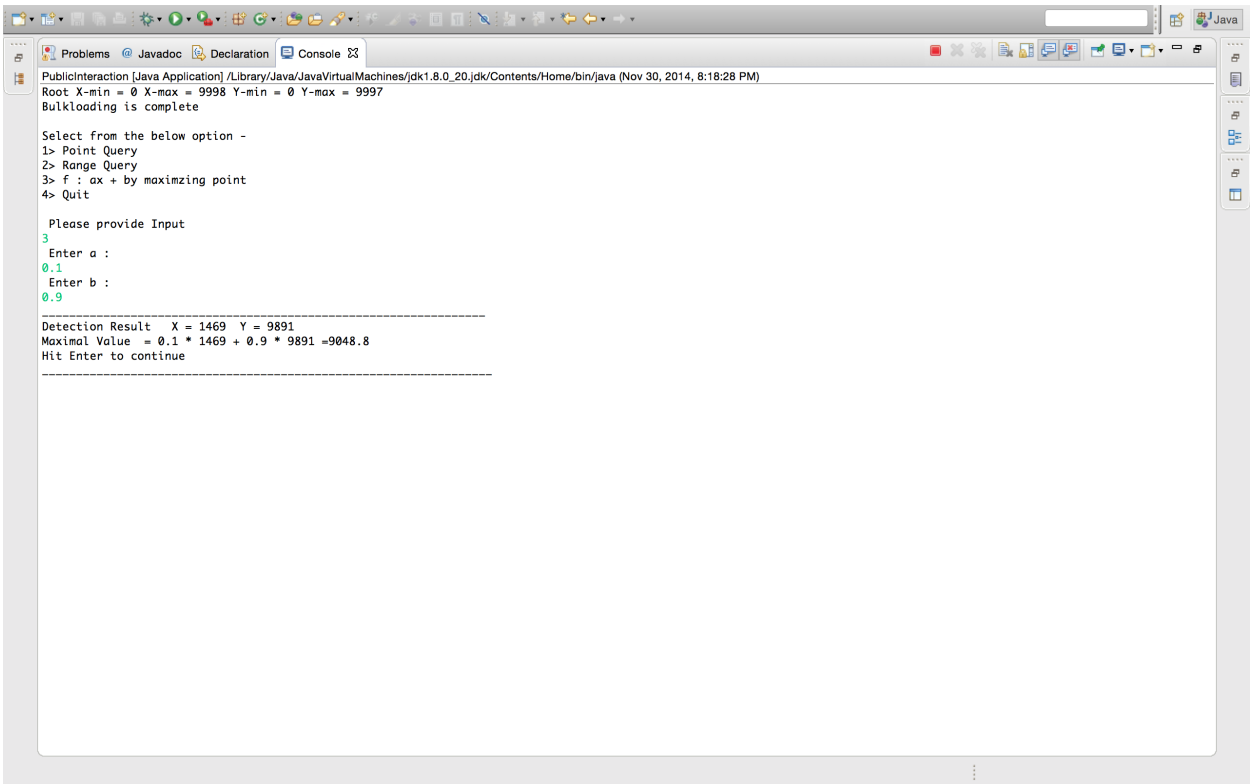
Note only if a data point is found then FOUND key word along with the Data Entry details is shown. The DESCRIPTION is also shown. If data point is not found then FOUND Key word followed by the Data Entry details will not be observed.

RANGE QUERY



Note - Here only those points are shown which exist in the given range. If no point is found then the FOUND key word followed by Data Entry details will not be observed.

f:ax + by QUERY



RTree Details

1. RBox is the class which is represents a rectangle.
2. All rectangles in this program are RBox objects.
3. Data Entry Rectangles contain description of 500 char. This variable is null for all rectangles other than the Data Entry Rectangles. This implies that the root, index nodes and leaf nodes will have description as null. The Data Entrry Rectangles are given the description while they are bulkloaded.
4. The RBox has List<RBox> Entry which points to child rectangles or RBox objects of a given RBox object.
5. For Leaf Rectangles the Entry points to data entries.
6. For Index Rectangles the Entry points to child RBox objects.
7. For Data Entry Rectangles Entry is null;
8. All data is held in the main memory.
9. Size of data entry = 4 * 4 bytes + 500* 2 bytes = 1016 bytes
   = 4 * size of an int + 500 * size of char
   Therefore 1 page of size 4096 byte has [4096/1016] = 4 data entries
   Therefore there are 7500 Leaf Rectangles
10. Size of an Index Entry = 4 * int + 4 byte for object reference or pointer
    Similarly it is seen that 1 Index Node has [4096 / 20] or 204 entries

There are 37 Index Rectangle

11. Thus in this implementation every thing is in the main memory.

Class Description

1. The name of the classes used for the implementation of R-Tree
   (1) **RBox** - The RBox class represents a class of the Rectangle Box. These are used to create rectangles. Note that due to the nature of R-Tree one rectangle can contain as its children many other rectangles. Also the data entries are also rectangles.
   (2) The data members of the RBox class are -

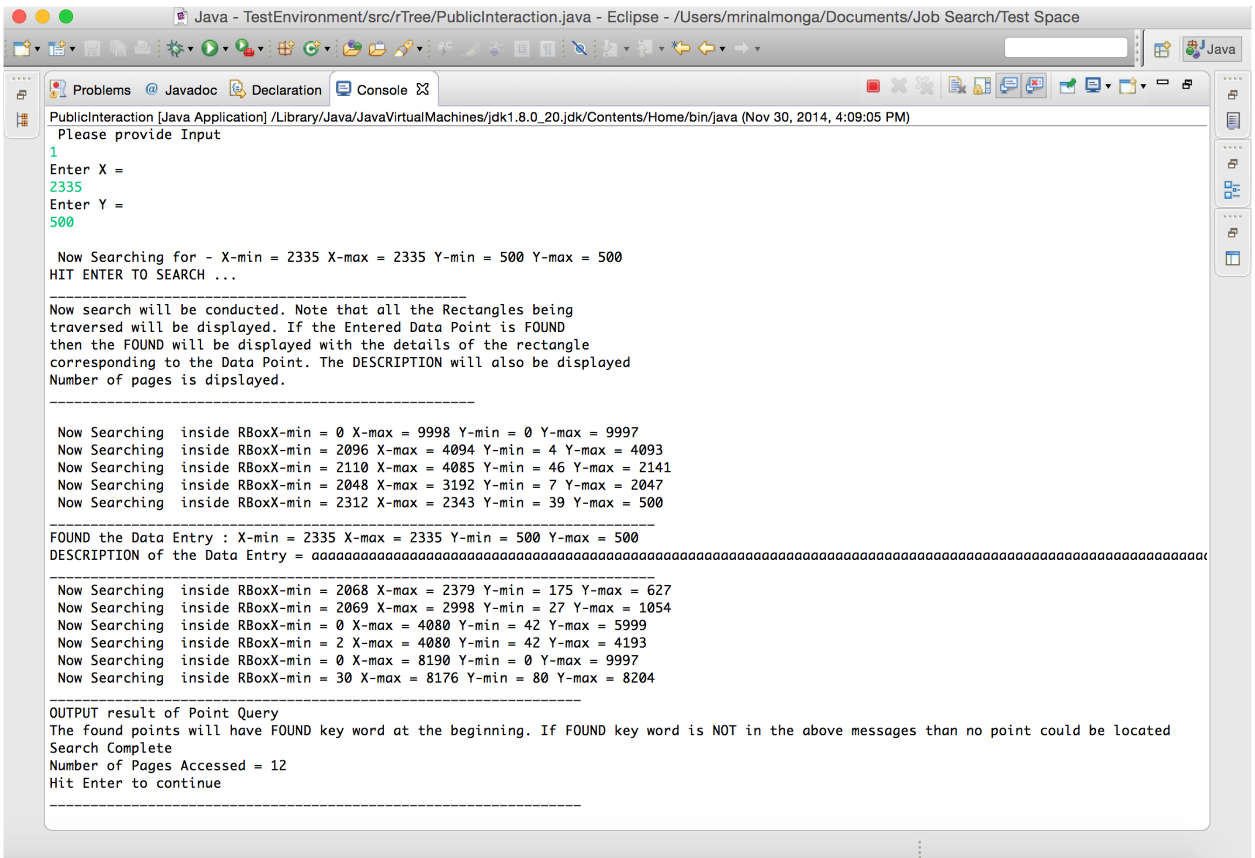
```
int xmax;
int xmin;
int ymax;
int ymin;


public char[] description; // This is used only by a Data Entry.
                           Not used by any other type of RBox.

List<RBox> Entry = null; // for data entry its value is null

List<RBox> listOfPotentialRBoxes = null;
```

   (3) Note that as the name suggests xmin, xmax, ymin, ymax contain the the minimum and maximum x and y values respectively corresponding to the rectangle. The char[] description is a data member which will ONLY be assigned value for the data entries of the leaf nodes. char[] description is null for all other types of rectangles.
   (4) List<RBox> Entry is the array list which holds the list of children rectangles pointed to by a rectangle . For DATA ENTRY rectangles Entry would be null as they don't point to anything. The LEAF NODES would point to the DATA ENTRY rectangles. They INDEX NODES would point to other INDEX NODES, LEAF NODES.
   (5) The List<RBox> listOfPotentialRBoxes is an array list which is used by the **pSearch(RBox b, QueryCostCalculator QueryCost)** and the **public rSearch(RBox b, QueryCostCalculator QueryCost)** methods and used for the Point and Range Queries respectively.
   (6) The **public void dimensionCalculation()** method is used to assign the values to the xmax, xmin, ymax, ymin variables. This method takes into consideration the xmax, xmin, ymax, ymin values of the rectangles in the Entry array list.
   (7) The **public void pSearch(RBox b, QueryCostCalculator QueryCost)** method is used for the point queries. When the user inputs a point, it is converted into an RBox object - that is a rectangle. This rectangle RBox b is sent to the method along with a QueryCostCalculator object which helps to keep track of the pages accessed during the search.

This method first check whether it is inside a rectangle which is a Leaf Data Entry or not. If it is not a Data Entry rectangle then it prepares a list of potentialRectangles which is basically those rectangles which are children of the current rectangle and which overlap the RBox -

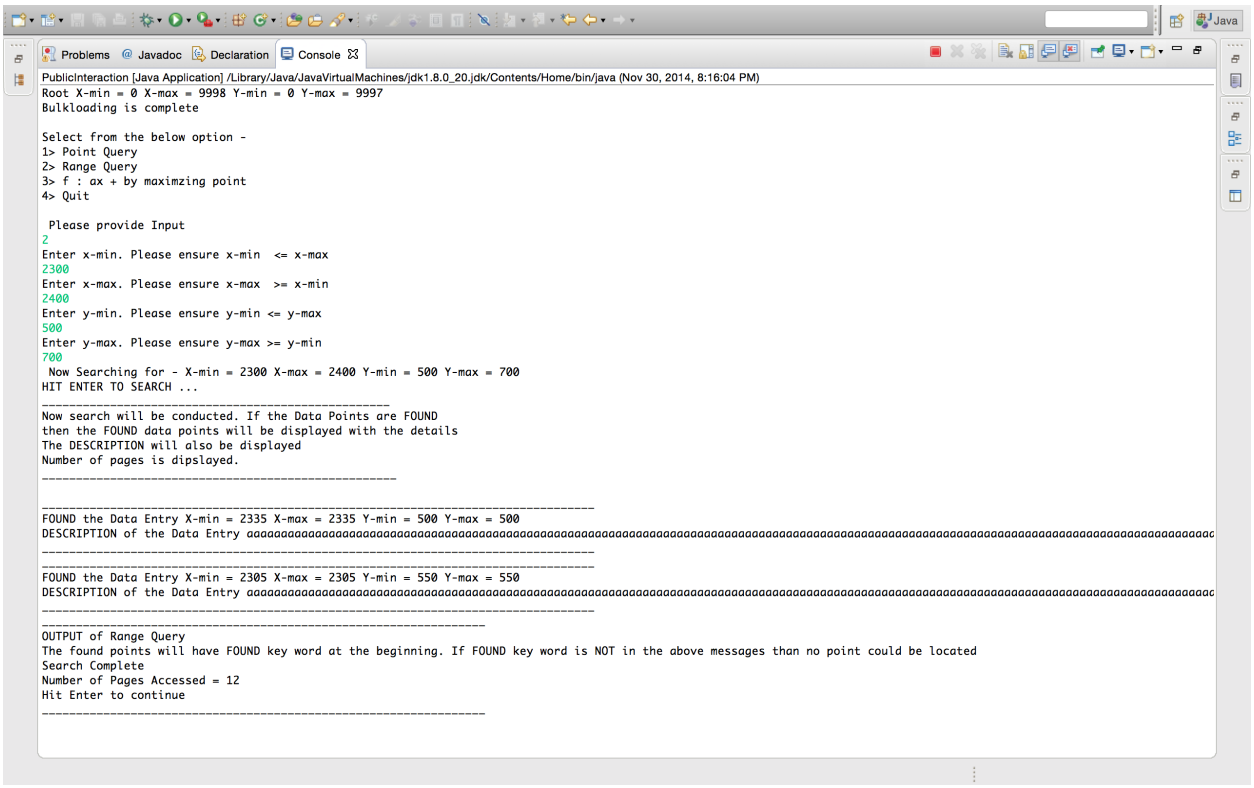
Rectangle corresponding to the user provided input. Then it searches in the sub-trees rooted at each of such rectangles which overlap the input, by calling the **pSearch(RBox b, QueryCostCalculator QueryCost)** of the child rectangles.

It displays as output all the nodes that it visits and if it is able to reach inside a Data Entry Rectangle which is the input being searched for, then it displays it with FOUND key word. If for a query no FOUND key word is in the output then that point couldn't be found. It also mentions the **Number of Pages Accessed** at the bottom.



(8) The public void rSearch(RBox b, QueryCostCalculator QueryCost) method answers the Range Queries. It is slightly different in its logic then the method for answering point queries. In this method once the list of potential rectangles which could contain the required RBoxs-Rectangles is formed, then the entries of the leaf nodes of the subtree of the potential rectangles are checked those entries are only displayed which actually are found to be CONTAINED by the rectangle corresponding to the input.

A Sample output -



Note - Only if points exist in the specified range are they displayed using the FOUND key word along with with the description.

(9) The methods public String getDescription() and public void setDescription(char[] description) are used by the Data Entries RBoxes-rectangles as they contain description.

As stated in the answer to FAQ 1. in the Assignment 3. Every thing is kept in memory. Pages are counted when they are traversed for I/O Cost.

(10) **Algorithm for finding the point maximizing the function f in D. f: ax + by, a+b =1.**
Introduction - This algorithm is based on the concept of sending a Detective Probe inside the R-Tree. The detective probe can be thought of as a probe which is initialized by providing it the a and b values for the f. It then traverses inside the R-Tree based on the algorithm in search of such a Data Entry which maximizes the value of f:ax + by.

Explanation - Every node in the R-Tree is a rectangle - here implemented by the R-Box. Each R-Box or rectangle can have children rectangles which may contain the points which could maximize the f:ax+by. So a method is there in the R-Box class and thus in each object of the class called the public Detective maximizingEntryDetector(Detective detective) method.

This method is both the point of insertion and point of extraction of the of the Detective probe into the R-Tree rooted at that R-Box or the rectangle under consideration. Once the Detective probe is initialized with a and b from the user, the R-Tree's root RBox or rectangle is asked to consume/ingest this Detective probe. The root then runs an algorithm explained below and passes this Detective probe to its such children rectangles or RBoxes which most likely contain the maximizing point. Each of the children rectangles then subsequently follow the same algorithm to identify their most likely descendant children and the process goes on till Leaf Rectangle is reached.
The Detective probe on reaching a Leaf Rectangle will scan the entire list of entries to identify such a Data Entry which maximizes the f:ax+by. It records the x, y and maximalValue. Subsequently when the Detective probe visits other leaf nodes it behaves similarly, **but** it would now record the x,y and maximal value ax+by **if and only if** the new maximal value found is **bigger** then the previous maximal value. So the Detective node in effect traverses the likely leaf rectangles based upon the algorithm, however at any point in time it would only have the x,y and maximal value corresponding to such a Data Entry which had maximized f:ax + by till that point in execution. Also once it finds that for a particular sub-tree the value f:ax+by is not greater than the maximalValue it has found till that point in execution, it would no longer go down that sub-tree. This is taken care of by the algorithm explained below.

Subsequently the Detective probe is returned by the same method public Detective maximizingEntryDetector(Detective detective) method at which it was inserted to into the root of the R-Tree, **where it was ingested by the R-Tree by for the very first time.** After extraction the Detective probe is examined to see what it found inside the R-Tree.

Algorithm which runs inside the public Detective maximizingEntryDetector(Detective detective) method

1. **Check if the existing RBox is a Leaf Node**
2. **If it is not a Leaf Node then -**

   Extract the a, b and maximalValue (found till this point in execution) from the Detective probe.

For the existing RBox find the value of ax+by by analyzing the centroids of all the children rectangles. Every RBox has a public float centroidX() and public float centroidY() method. These methods return the X and Y coordinates of the centroid of the rectangle.

For a child RBox - rectangle its centroid's coordinates are put in the f:ax+by to find the value.

Scan all the children rectangles **to observe and find out if at all** any of the child rectangles or RBoxes gives a value of f: ax+by greater than that found till this point in execution which is stored in Detective probe's variable maximalValue.

This is implemented by substituting the centroid of a RBox or rectangle in the f:ax+by and comparing it with the “maximalValue” found till now , which was extracted from the Detective probe. The detective probe on its way from other Rectangles to the current rectangle stores the x,y, and the maximalValue found till this point in execution.

The algorithm is designed in such a manner so that if it was found that the new maximalValue found was greater than that stored inside the Detective probe then again scan all the children RBoxes - rectangles to find those rectangle for which the value obtained by putting their centroid’s coordinates for x, y in f:ax+by gives a value equal to the new greater maximalValue found now.

The list of all such children rectangles the centroids of which when put in f:ax+by give the maximalValue found till now is prepared.

Thereafter again scan the entire list of children rectangles of the current Rectangle to find all such rectangles which **overlap** the list of rectangles the centroids of which give the maximalValue when substituted for x,y in the f:ax + by.

The above step is done to ensure that the rectangle the centroid of which maximizes the f:ax+by may not contain that data entry in its sub-tree which actually maximizes f:ax+by . So all those rectangles which overlap the rectangle are also added to the search space.

Prepare a list of all of the above rectangles , remove any redundancy which may be there on account of counting a rectangle more than once because of several scans.

Then send the Detective probe to every such rectangle from the list prepared.

Note - We send the Detective probe down every possible sub-tree but the probe need not actually reach the Leaf Rectangles of each of the sub-tree as the algorithm takes care to let the probe go down further if and only if the value f:ax+by obtained by scanning the children rectangles or Rboxes of a rectangle is greater than the maximalValue found till that point in execution.

This means even if we send the Detective probe down every potential sub-tree it would actually go down only if the condition new-maximal-Value-found > maximalValue-found-till-now is met. Other wise it would NO LONGER CONTINUE down that subtree.

3. If a Leaf Rectangle is reached

Scan all the Data Entries of the leaf node to check whether the new value found for f:ax+by by substituting the x,y of the Data Entry exceeds the maximalValue found till this point in execution inside Detective probe. If it does exceed record the new x, y, and maximalValue found else don’t store anything.

4. Running Time of the Algorithm -

In worst case - Height of tree levels would need to be traversed - log(n) base - branching factor. maximum data entries which could ever be scanned in a tree = n

Thus Running Time - O(n * log(n)) (ASSUME base = branching factor of R-Tree)

(11) Class RBoxHelper
This class has two methods
One is static boolean doTheseOverlap(RBox b1, RBox b2) which tells whether two rectangles overlap. The other is static boolean contains(RBox b1, RBox b2) which checks whether one rectangle contains the other.

(12) QueryCostCalculator Class
The object of this class is used to keep track of the number of pages traversed .

(13) PublicInteractionClass
This interacts with the user. This is responsible for bulk loading of the tree and interacting with the user.