CEGEP VANIER COLLEGE
CENTRE FOR CONTINUING EDUCATION
Programming Algorithms and Patterns
420-930-VA

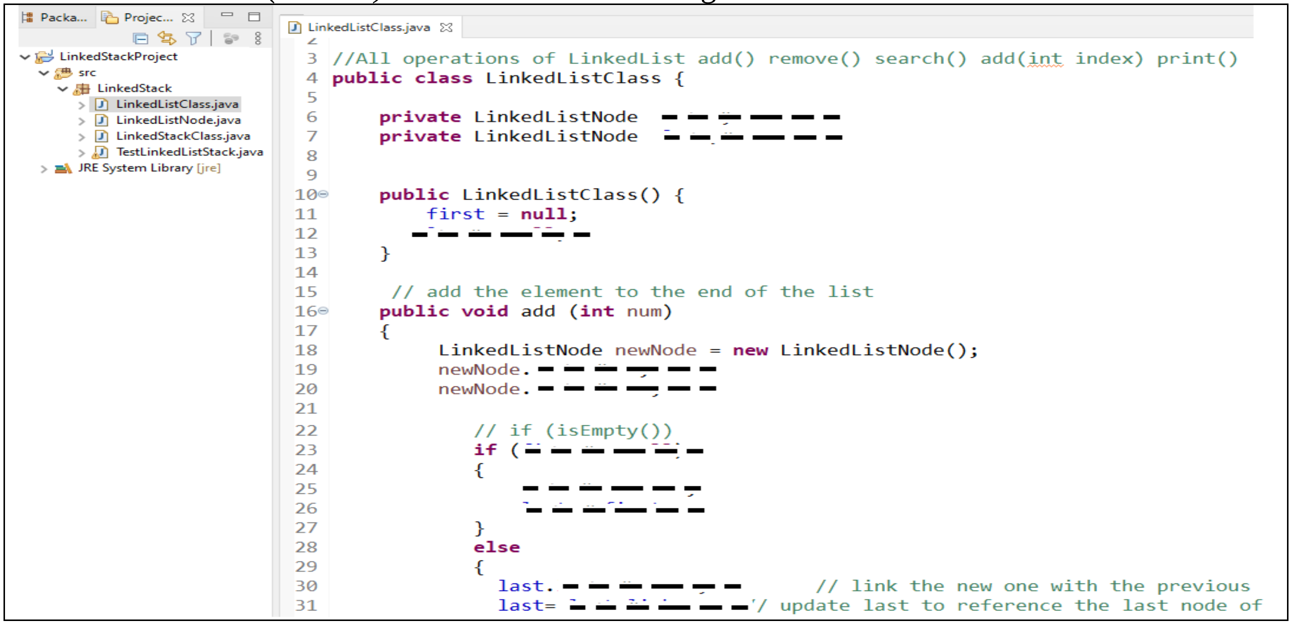
Teacher: Samir Chebbine **Lab 3** **Jun 07, 2024**

Lab 3: Stacks, Queues and Binary Trees

Complete all these following programs as explained during **classes.** All *missing coding statements* were provided there with explanation. **Create and Submit** a Word file ***Lab3ProgramminAlgorithmsandPatternsYourName.docx*** which includes **output screenshots** for every Java Project. Submit the Java projects too.

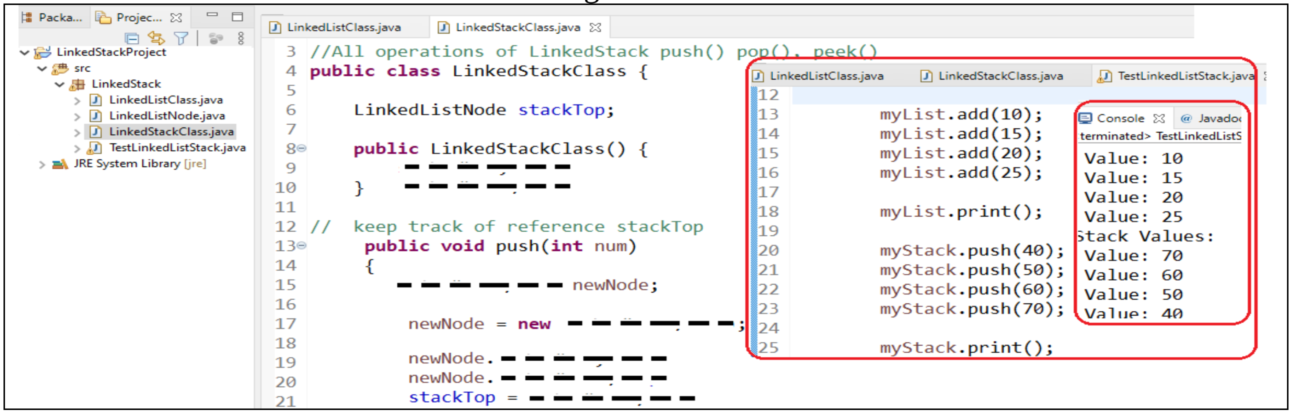
1. Stack Data Structure: Create a project named LinkedStackProject

a) Create *LinkedStackProject* using Eclipse IDE for developing *user-defined methods* of Stack and linked list operations. Create *LinkedListClass.java* that includes user-defined method *add(int num)* as shown hereafter in Figure.



```
//All operations of LinkedList add() remove() search() add(int index) print()
public class LinkedListClass {

    private LinkedListNode [redacted]
    private LinkedListNode [redacted]


    public LinkedListClass() {
        first = null;
      [redacted]
    }

     // add the element to the end of the list
    public void add (int num)
    {
          LinkedListNode newNode = new LinkedListNode();
          newNode.[redacted]
          newNode.[redacted]

              // if (isEmpty())
              if ([redacted]
              {
                  [redacted]
                  [redacted]
              }
              else
              {
                last.[redacted]        // link the new one with the previous
                last= [redacted]/ update last to reference the last node of
```

b) Create *LinkedStackClass.java* that includes user-defined method *push(int num)* and test both classes as shown hereafter in Figure.



```
//All operations of LinkedStack push() pop(), peek()
public class LinkedStackClass {

    LinkedListNode stackTop;

    public LinkedStackClass() {
        [redacted]
    }   [redacted]

//   keep track of reference stackTop
     public void push(int num)
     {
         [redacted] newNode;

          newNode = new [redacted];

          newNode.[redacted]
          newNode.[redacted]
          stackTop = [redacted]
```

```
        myList.add(10);
        myList.add(15);
        myList.add(20);
        myList.add(25);

        myList.print();

        myStack.push(40);
        myStack.push(50);
        myStack.push(60);
        myStack.push(70);

        myStack.print();
```

```
terminated> TestLinkedListS
 Value:  10
 Value:  15
 Value:  20
 Value:  25
Stack Values:
 Value:  70
 Value:  60
 Value:  50
 Value:  40
```

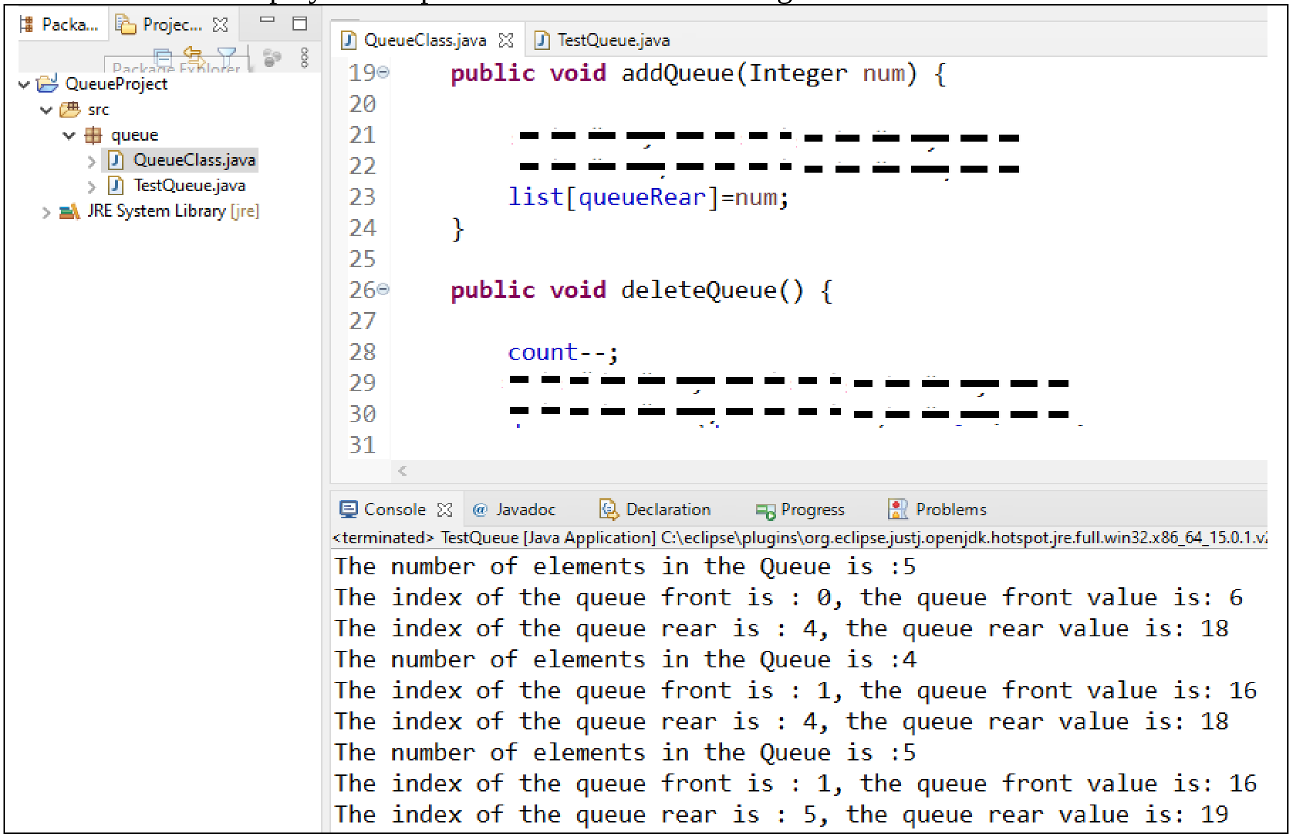
2. Queue Data Structure: Create a Project named QueueProject

Create a Java Project named QueueProject to add element to a given queue and removing an element from a queue.

- Create a class called *QueueClass* which contains the following data members:

```
public int queueFront;    // keeps track of the first element
public int queueRear;     // keeps track of the last element
public int maxQueueSize;  // specifies the maximum size of the queues
public int count;         // number of element in the queue
public Integer[]  list;
```

- Add a constructor to *QueueClass* in order to initialize *the above data members accordingly*.
- Add Java method *addQueue(Integer num)* into *QueueClass* to add an element in the queue.
- Add Java method *deleteQueue ()* into *QueueClass* to delete an element in the queue.
- Test and display the output as shown hereafter in Figure.



```
    public void addQueue(Integer num) {

        [redacted]
        list[queueRear]=num;
    }

    public void deleteQueue() {

        count--;
        [redacted]
```

<terminated> TestQueue [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v

```
The number of elements in the Queue is :5
The index of the queue front is : 0, the queue front value is: 6
The index of the queue rear is : 4, the queue rear value is: 18
The number of elements in the Queue is :4
The index of the queue front is : 1, the queue front value is: 16
The index of the queue rear is : 4, the queue rear value is: 18
The number of elements in the Queue is :5
The index of the queue front is : 1, the queue front value is: 16
The index of the queue rear is : 5, the queue rear value is: 19
```

3. Stack List for Book records

a) Create a Java Project ***StackBookProject*** to assign the instance data of all *records* read from the input file *Book.in* into a *stack List* to print records in the `descending order` with respect to the amount of book price. You should use user-defined method *push()* implemented in the user-defined Linked Stack class.

b) Create a class to define data structure type, called *Book*, which is designed to group data and functions into a single unit that *represents* a template of the fields used in `Book.in` as shown in the following Figure.

c) Each line within *Book.in* represents a book's record stored in book price `ascending order` with the following fields: `(b_id (int),b_author(string),b_title(string),b_isbn(string),` b_type(string), b_price(double)).

d) Add **default constructor** (b_id=0,b_pid=0,b_author="",b_title="",b_isbn="", b_type="", b_price=0) and **constructor with parameters** within the Book class in order to initialize the data members (b_id, b_pid, b_author, b_title, b_isbn, b_type, b_price) of every object.

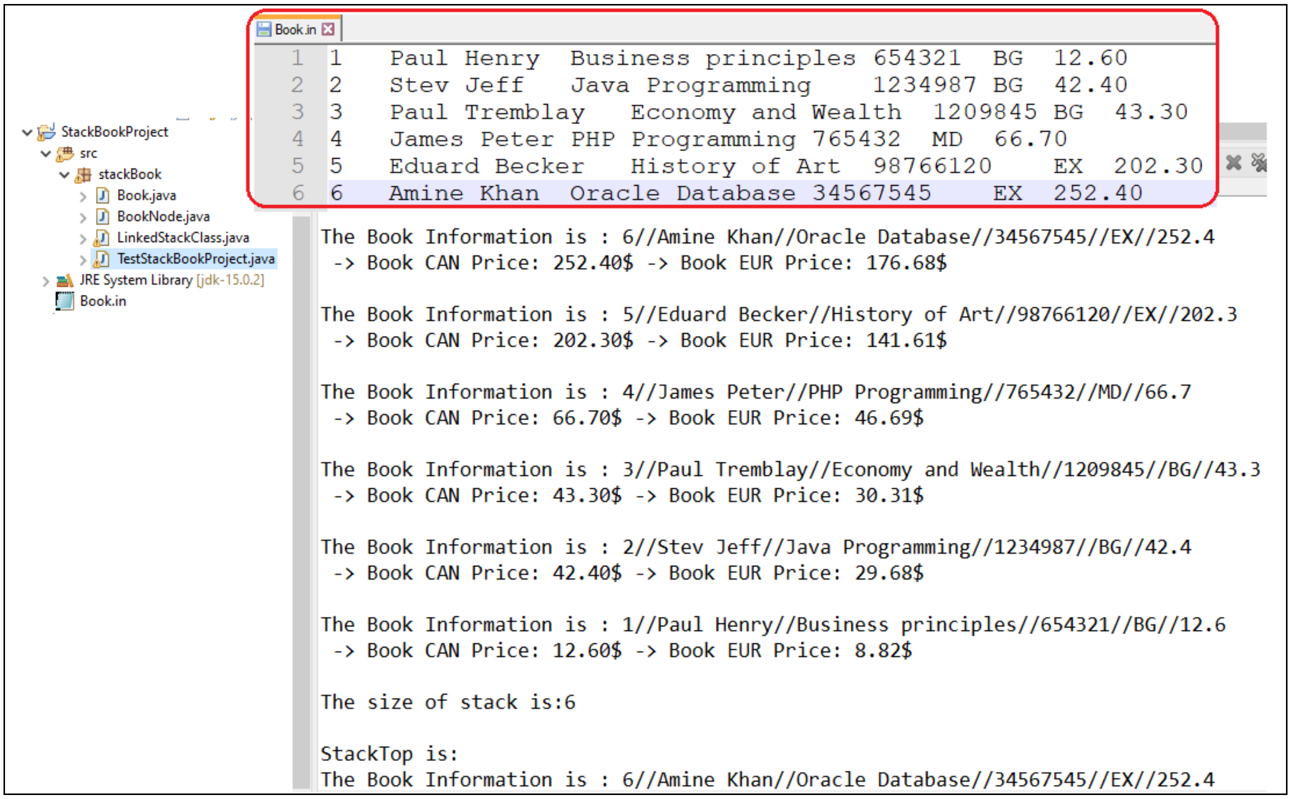
e) Add public **Mutator** **(setter)** methods (setBook_id(),setB_Author(),setB_Title(), setB_Isbn(),setB_Type(),setB_Price()) in Book class to modify the values of private members.

f) Add public **Accessor** **(getter)** methods (getBook_id(),getB_Author(),getB_Title(), getB_Isbn(),getB_Type(),getB_Price()) in Book class to read the values of private members.

g) Add a method called public String toString() in Book class to print the Book information in the form of "The Book Information is : " + b_id + "//"+ b_author + "//" + b_title + "//" + b_isbn + "//" + b_type+ "//" + b_price

h) Add a method called public calculate_Price_Euro() in Book class to calculate the price in euro using the following formula: b_price*0.7

i) You need to instantiate in the main method objects of Book class type, and display the output as shown hereafter.



```
1    Paul Henry  Business principles 654321  BG  12.60
2    Stev Jeff   Java Programming    1234987 BG  42.40
3    Paul Tremblay   Economy and Wealth  1209845 BG  43.30
4    James Peter PHP Programming 765432  MD  66.70
5    Eduard Becker   History of Art  98766120    EX  202.30
6    Amine Khan  Oracle Database 34567545    EX  252.40
```

```
The Book Information is : 6//Amine Khan//Oracle Database//34567545//EX//252.4
 -> Book CAN Price: 252.40$ -> Book EUR Price: 176.68$

The Book Information is : 5//Eduard Becker//History of Art//98766120//EX//202.3
 -> Book CAN Price: 202.30$ -> Book EUR Price: 141.61$

The Book Information is : 4//James Peter//PHP Programming//765432//MD//66.7
 -> Book CAN Price: 66.70$ -> Book EUR Price: 46.69$

The Book Information is : 3//Paul Tremblay//Economy and Wealth//1209845//BG//43.3
 -> Book CAN Price: 43.30$ -> Book EUR Price: 30.31$

The Book Information is : 2//Stev Jeff//Java Programming//1234987//BG//42.4
 -> Book CAN Price: 42.40$ -> Book EUR Price: 29.68$

The Book Information is : 1//Paul Henry//Business principles//654321//BG//12.6
 -> Book CAN Price: 12.60$ -> Book EUR Price: 8.82$

The size of stack is:6

StackTop is:
The Book Information is : 6//Amine Khan//Oracle Database//34567545//EX//252.4
```

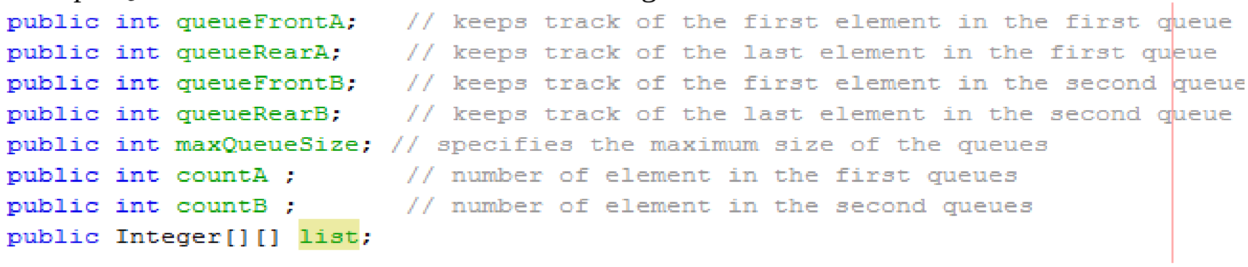
4. Queue with two lines

Create a Java Project named *MultipleQueueProject* in order to implement a new data structure called *MultipleQueueClass* for maintaining two queues *A* and *B* at the same time (in case you have two counter desks). You need to take into account the following requirements:

- The first element is inserted in the first queue *A*.
- A scenario when an element is already inserted in the first queue *A*, the second element *has* to be inserted in the second queue *B*.

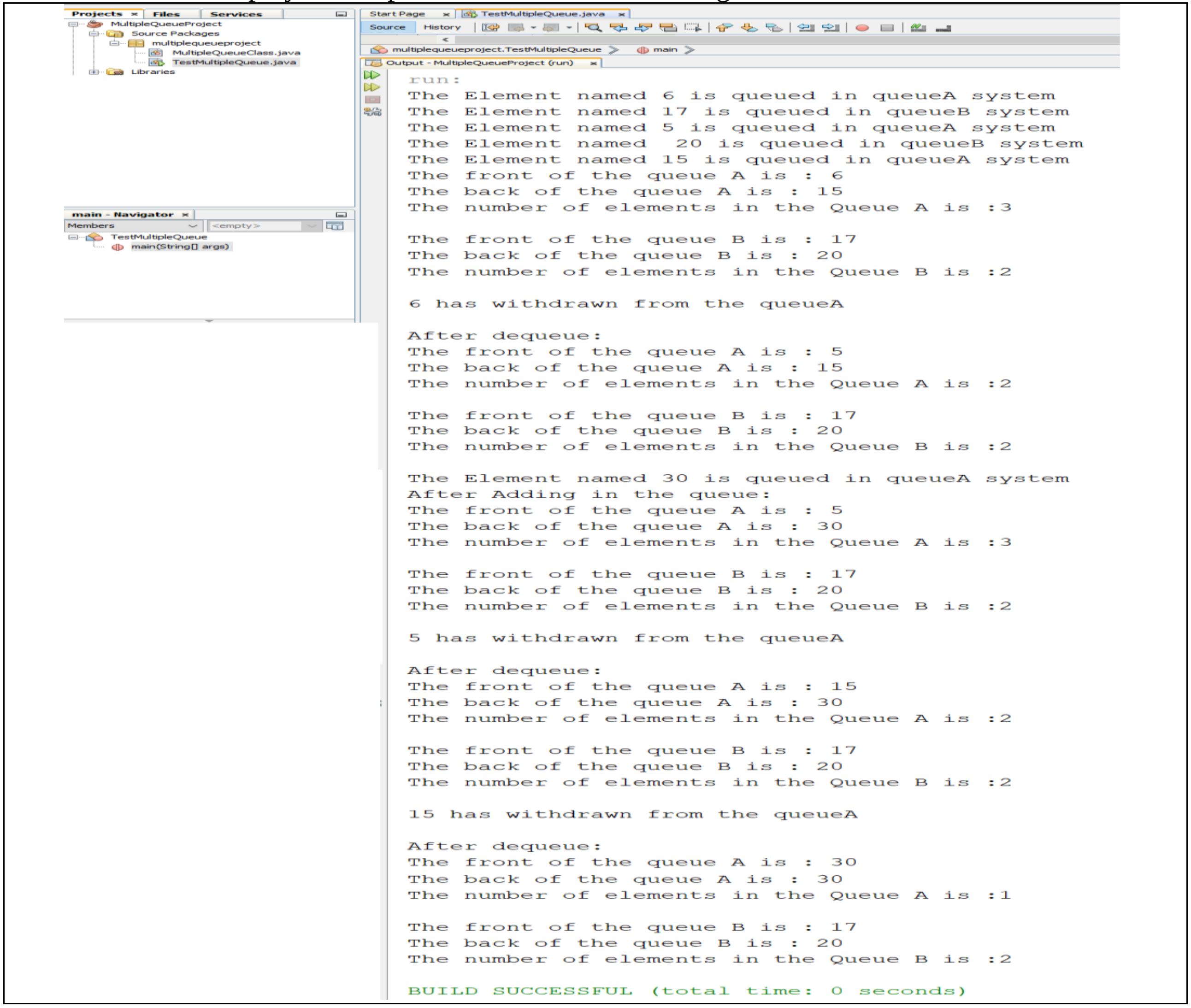
- A new element is inserted in queue *A* if number of elements is the same in both queues.
- You need to delete an element from a given queue which has higher size of elements. You delete an element from the queue *A* if the size is the same in the two queues.
- You need to maintain the front and the rear of each queue.
- Maintain the elements of the two queues in two dimensional array *list[ ][ ]*

Implement *MultipleQueueClass* using arrays to simulate two lines of queues in Counter waiting line. *MultipleQueueClass* should contains the following data members:



```
public int queueFrontA;   // keeps track of the first element in the first queue
public int queueRearA;    // keeps track of the last element in the first queue
public int queueFrontB;   // keeps track of the first element in the second queue
public int queueRearB;    // keeps track of the last element in the second queue
public int maxQueueSize; // specifies the maximum size of the queues
public int countA ;       // number of element in the first queues
public int countB ;       // number of element in the second queues
public Integer[][] list;
```

- Add a constructor to *QueueClass* in order to initialize *the above data members accordingly*.
- Add Java method *addQueue(Integer num)* into *MultipleQueueClass* to add an element in one of the two queues accordingly.
- Add Java method *deleteQueue()* into *MultipleQueueClass* to delete an element in one of the two queues accordingly.
- Test and display the output as shown hereafter in Figure.

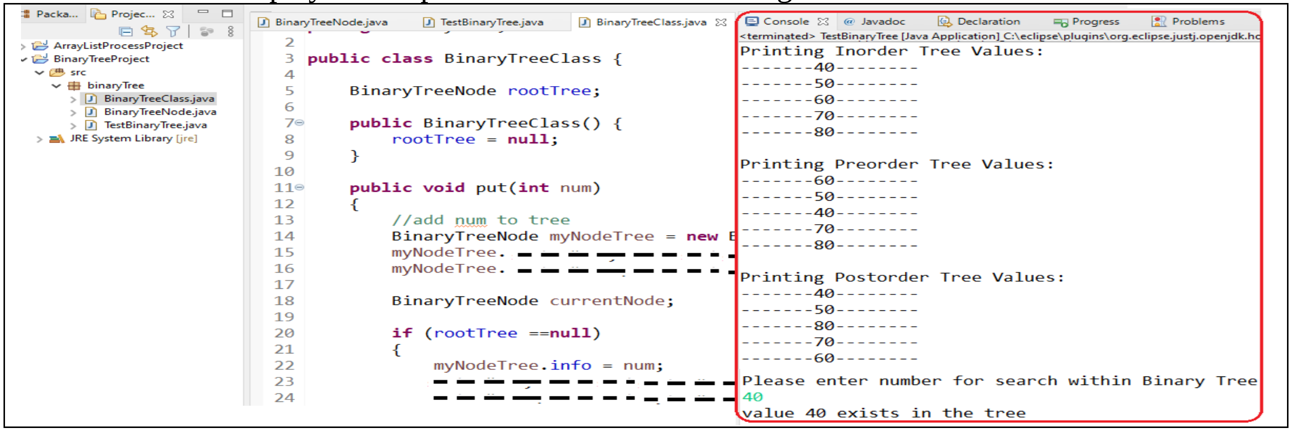


```
run:
The Element named 6 is queued in queueA system
The Element named 17 is queued in queueB system
The Element named 5 is queued in queueA system
The Element named  20 is queued in queueB system
The Element named 15 is queued in queueA system
The front of the queue A is : 6
The back of the queue A is : 15
The number of elements in the Queue A is :3

The front of the queue B is : 17
The back of the queue B is : 20
The number of elements in the Queue B is :2

6 has withdrawn from the queueA

After dequeue:
The front of the queue A is : 5
The back of the queue A is : 15
The number of elements in the Queue A is :2

The front of the queue B is : 17
The back of the queue B is : 20
The number of elements in the Queue B is :2

The Element named 30 is queued in queueA system
After Adding in the queue:
The front of the queue A is : 5
The back of the queue A is : 30
The number of elements in the Queue A is :3

The front of the queue B is : 17
The back of the queue B is : 20
The number of elements in the Queue B is :2

5 has withdrawn from the queueA

After dequeue:
The front of the queue A is : 15
The back of the queue A is : 30
The number of elements in the Queue A is :2

The front of the queue B is : 17
The back of the queue B is : 20
The number of elements in the Queue B is :2

15 has withdrawn from the queueA

After dequeue:
The front of the queue A is : 30
The back of the queue A is : 30
The number of elements in the Queue A is :1

The front of the queue B is : 17
The back of the queue B is : 20
The number of elements in the Queue B is :2

BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Building Binary Tree class

Create project BinaryTreeProject to implement Binary search tree. Create a class *called BinaryTreeClass* which includes an attribute *rootTree* of *BinaryTreeNode* class type as done during Zoom synchronous class.

- Add a constructor to *BinaryTreeClass* in order to initialise *rootTree* to null.
- Add Java method *put()* into *BinaryTreeClass* to insert an element *num* in the Binary Search Tree.
- Add Java method *search()* into *BinaryTreeClass* to search an element *num* in the Binary Search Tree.
- Add Java method *InorderTraversal()* into *BinaryTreeClass* to print all elements of Binary Search Tree.
- Add Java method *PreorderTraversal ()* into *BinaryTreeClass* to print all elements of Binary Search Tree.
- Add Java method *PostorderTraversal ()* into *BinaryTreeClass* to print all elements of Binary Search Tree.
- Test and display the output as shown hereafter in Figure.



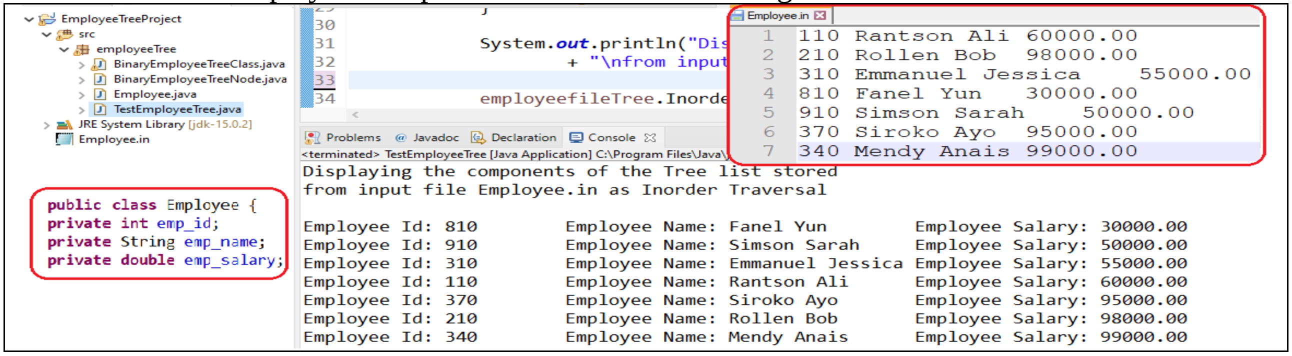
```
public class BinaryTreeClass {

    BinaryTreeNode rootTree;

    public BinaryTreeClass() {
        rootTree = null;
    }

    public void put(int num)
    {
        //add num to tree
        BinaryTreeNode myNodeTree = new B
        myNodeTree. [redacted]

        BinaryTreeNode currentNode;

        if (rootTree ==null)
        {
            myNodeTree.info = num;
            [redacted]
```

```
<terminated> TestBinaryTree [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.ho
Printing Inorder Tree Values:
-------40--------
-------50--------
-------60--------
-------70--------
-------80--------

Printing Preorder Tree Values:
-------60--------
-------50--------
-------40--------
-------70--------
-------80--------

Printing Postorder Tree Values:
-------40--------
-------50--------
-------80--------
-------70--------
-------60--------
Please enter number for search within Binary Tree
40
value 40 exists in the tree
```

6. Binary Tree for Employee records

Create a Java Project *EmployeeTreeProject* to assign the instance data of all *records* read from the input file *Employee.in* into a *Binary Search Tree* such that *emp_salary* attribute *in each node of Binary Search Tree is **Larger** than the emp_salary in its left child and **Smaller** than the emp_salary in its right child* as shown in Figure.

- Create class *Employee* that defines data attributes of every employee record (emp_id, emp_name, emp_salary).
- Create class *BinaryEmployeeTreeNode* that defines every node in tree where data info is of *Employee* Data type.
- Create *class BinaryEmployeeTreeClass* that defines *put (BinaryEmployeeTreeNode obj)* and *InorderTraversal()* operations related to binary tree.
- Test and display the output as shown hereafter in Figure.



```
public class Employee {
private int emp_id;
private String emp_name;
private double emp_salary;
```

```
        }

            System.out.println("Dis
                + "\nfrom input

            employeefileTree.Inorde
```

Employee.in

```
110 Rantson Ali 60000.00
210 Rollen Bob  98000.00
310 Emmanuel Jessica     55000.00
810 Fanel Yun    30000.00
910 Simson Sarah      50000.00
370 Siroko Ayo   95000.00
340 Mendy Anais 99000.00
```

```
<terminated> TestEmployeeTree [Java Application] C:\Program Files\Java\
Displaying the components of the Tree list stored
from input file Employee.in as Inorder Traversal

Employee Id: 810        Employee Name: Fanel Yun        Employee Salary: 30000.00
Employee Id: 910        Employee Name: Simson Sarah     Employee Salary: 50000.00
Employee Id: 310        Employee Name: Emmanuel Jessica Employee Salary: 55000.00
Employee Id: 110        Employee Name: Rantson Ali      Employee Salary: 60000.00
Employee Id: 370        Employee Name: Siroko Ayo       Employee Salary: 95000.00
Employee Id: 210        Employee Name: Rollen Bob       Employee Salary: 98000.00
Employee Id: 340        Employee Name: Mendy Anais      Employee Salary: 99000.00
```