

CEGEP VANIER COLLEGE

CENTRE FOR CONTINUING EDUCATION

Programming Algorithms and Patterns

420-930-VA

Teacher: Samir Chebbine

Lab 3

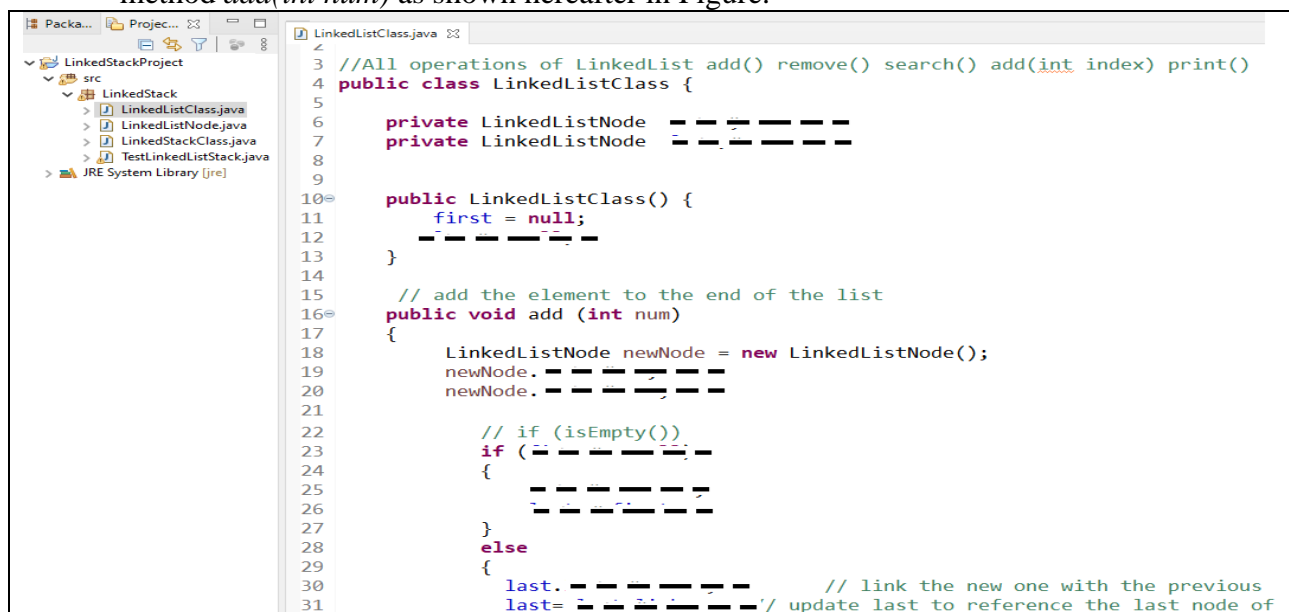
Jul 28, 2023

Lab 3: Stacks, Queues and Binary Trees

Complete all these following programs as explained during **Zoom Synchronous classes**. All *missing coding statements* were provided there with explanation. **Create and Submit** a Word file **Lab3PrograminAlgorithmsandPatternsYourName.docx** which includes **output screenshots** for every Java Project. Submit the Java projects too.

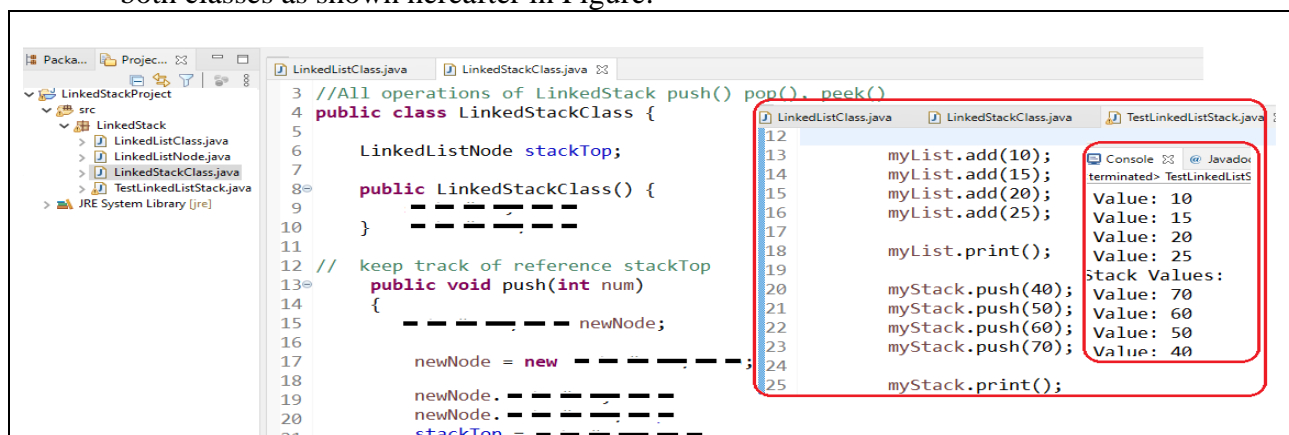
1. Stack Data Structure: Create a project named **LinkedListProject**

- a) Create *LinkedListProject* 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.



```
3 //All operations of LinkedList add() remove() search() add(int index) print()
4 public class LinkedListClass {
5
6     private LinkedListNode  _ _ _ _ _
7     private LinkedListNode  _ _ _ _ _
8
9
10    public LinkedListClass() {
11        first = null;
12    }
13
14    // add the element to the end of the list
15    public void add (int num)
16    {
17        LinkedListNode newNode = new LinkedListNode();
18        newNode._ _ _ _ _
19        newNode._ _ _ _ _
20
21
22        // if (isEmpty())
23        if ( _ _ _ _ _ )
24        {
25            _ _ _ _ _
26        }
27        else
28        {
29            last._ _ _ _ _ // link the new one with the previous
30            last= _ _ _ _ _ // update last to reference the last node of
```

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



```
3 //All operations of LinkedListStack push() pop(). peek()
4 public class LinkedListStackClass {
5
6     LinkedListNode stackTop;
7
8     public LinkedListStackClass() {
9     }
10
11    // keep track of reference stackTop
12    public void push(int num)
13    {
14        _ _ _ _ _ newNode;
15
16        newNode = new _ _ _ _ _;
17
18        newNode._ _ _ _ _
19        newNode._ _ _ _ _
20        stackTop = _ _ _ _ _
21
22
23
24
25
```

```
12    myList.add(10);
13    myList.add(15);
14    myList.add(20);
15    myList.add(25);
16
17    myList.print();
18
19    myStack.push(40);
20    myStack.push(50);
21    myStack.push(60);
22    myStack.push(70);
23
24    myStack.print();
25
```

Console terminated> TestLinkedListS

Value: 10
Value: 15
Value: 20
Value: 25
Stack Values:
Value: 70
Value: 60
Value: 50
Value: 40

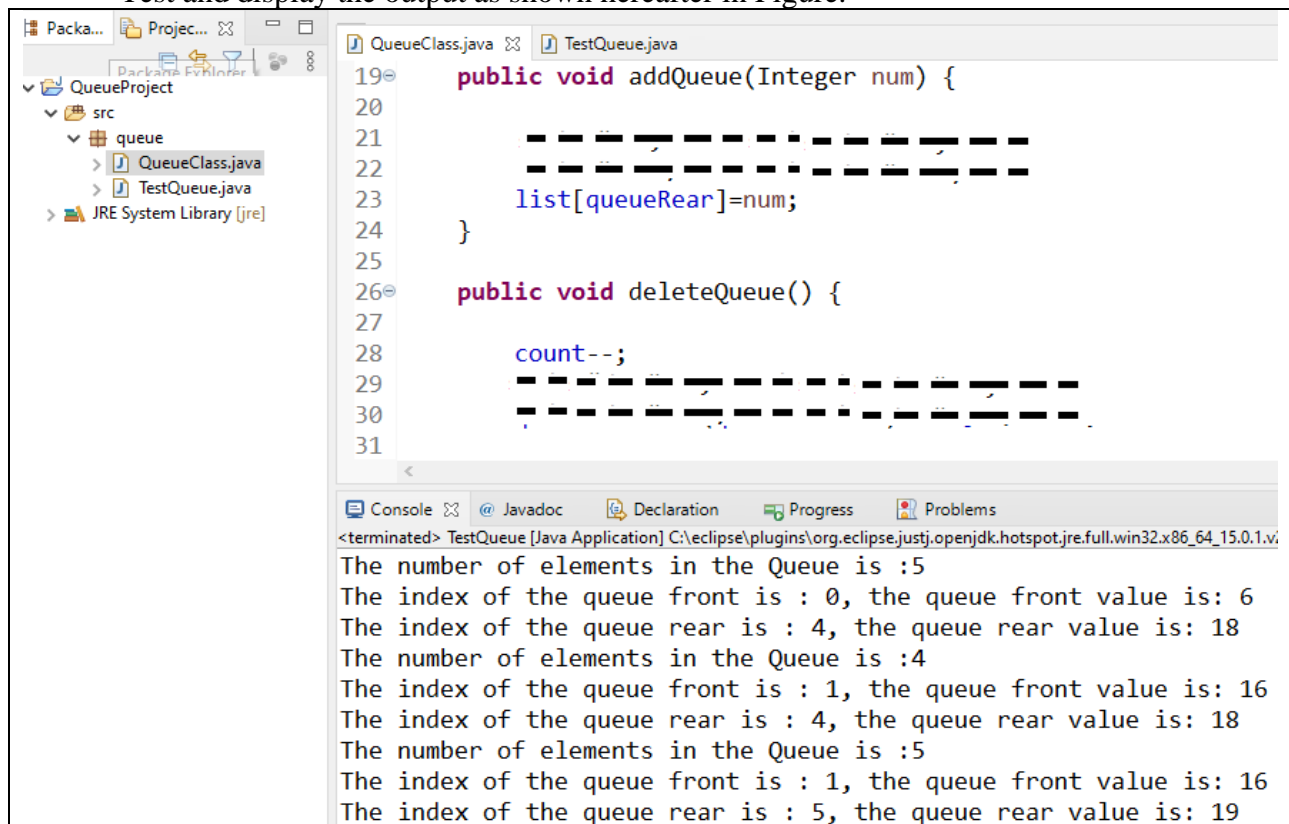
2. Queue Data Structure: Create a Project named QueueProject

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.



```
19 public void addQueue(Integer num) {
20
21     -----
22     list[queueRear]=num;
23 }
24
25
26 public void deleteQueue() {
27
28     count--;
29     -----
30     -----
31 }
```

```
<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.

```

Book.in
1 1 Paul Henry Business principles 654321 BG 12.60
2 2 Stev Jeff Java Programming 1234987 BG 42.40
3 3 Paul Tremblay Economy and Wealth 1209845 BG 43.30
4 4 James Peter PHP Programming 765432 MD 66.70
5 5 Eduard Becker History of Art 98766120 EX 202.30
6 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:

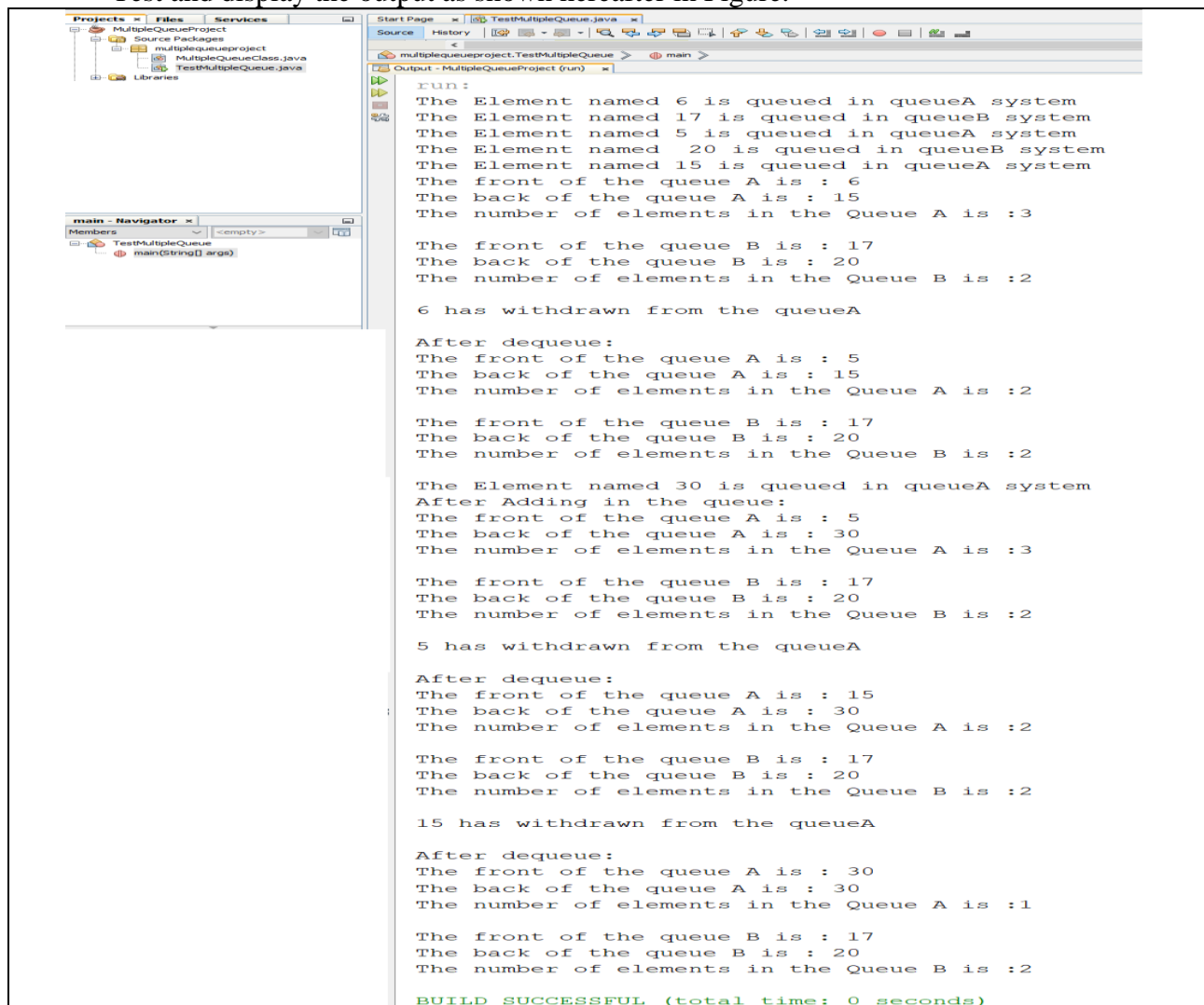
- The first element is inserted in the first queue *A*.
- A scenario when an element is already inserted in the first queue *A*, the second element *has* to be inserted in the second queue *B*.

- 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;
```

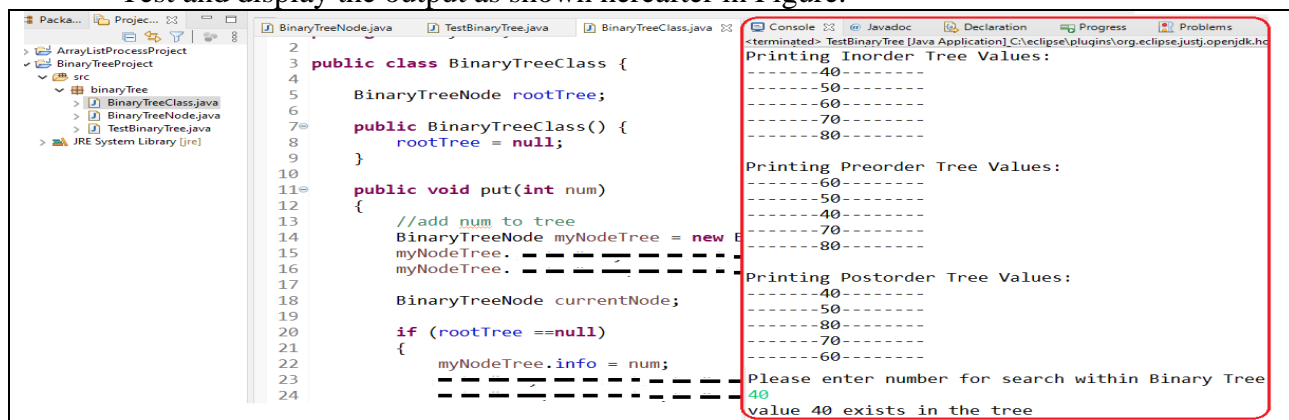
- Add a constructor to *QueueClass* in order to initialize the above data members accordingly.
- Add Java method *addQueue(Integer num)* into *MultipleQueueClass* to add an element in one of the two queues accordingly.
- Add Java method *deleteQueue()* into *MultipleQueueClass* to delete an element in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



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.



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.

