Day 5:

Task 1: Implementing a Linked List

1) Write a class CustomLinkedList that implements a singly linked list with methods for InsertAtBeginning, InsertAtEnd, InsertAtPosition, DeleteNode, UpdateNode, and DisplayAllNodes. Test the class by performing a series of cases insertions, updates and deletions.

- Link list is linear data structure that have data field and address field.
- Searching from first node to last node
- Useful only deletion and insertion in middle of elements;
-

ge ...

ay_Assignment
kProject
lections
a_Algorithums
JRE System Library [J
src
Linked_List
Linked_list.java
module-info.java

IBank.java | HashTableDem... | Doctor.java | Recurtion.java | fibonacci_s... | Linked_list....

collections/src/MapDemo/Doctor.java

```java
package Linked_List;

public class Linked_list {

	private Node head;

	 private static class Node {
	     int data;
	     Node next;

	     public Node(int data) {
	         this.data = data;
	         this.next = null;
	     }
	 }

	 public void insertAtBeginning(int data) {
	     Node newNode = new Node(data);
	     newNode.next = head;
	     head = newNode;
	 }

	 public void insertAtEnd(int data) {
	     Node newNode = new Node(data);
	     if (head == null) {
	         head = newNode;
	         return;
	     }
	     Node current = head;
	     while (current.next != null) {
	         current = current.next;
	     }
	     current.next = newNode;
	 }
```

•

IBank.java | HashTableDem... | Doctor.java | two_sum_Prob... | Recurtion.java | fibonacci_s

```java
        }

        public void insertAtPosition(int data, int position) {
            if (position <= 0) {
                throw new IllegalArgumentException("Position must be greater than zero");
            }
            if (position == 1) {
                insertAtBeginning(data);
                return;
            }
            Node newNode = new Node(data);
            Node current = head;
            for (int i = 1; i < position - 1 && current != null; i++) {
                current = current.next;
            }
            if (current == null) {
                throw new IndexOutOfBoundsException("Position is out of bounds");
            }
            newNode.next = current.next;
            current.next = newNode;
        }

        public void deleteNode(int data) {
            if (head == null) {
                throw new IllegalStateException("List is empty");
            }
            if (head.data == data) {
                head = head.next;
                return;
            }
            Node current = head;
            while (current.next != null && current.next.data != data) {
                current = current.next;
            }
            if (current.next == null) {
                throw new IllegalArgumentException("Data not found in the list");
```

```java

        public void updateNode(int oldData, int newData) {
            if (head == null) {
                throw new IllegalStateException("List is empty");
            }
            if (head.data == oldData) {
                head.data = newData;
                return;
            }
            Node current = head;
            while (current.next != null && current.next.data != oldData) {
                current = current.next;
            }
            if (current.next == null) {
                throw new IllegalArgumentException("Old data not found in the list");
            }
            current.next.data = newData;
        }

        public void displayAllNodes() {
            Node current = head;
            while (current != null) {
                System.out.print(current.data + " -> ");
                current = current.next;
            }
            System.out.println("null");
        }

        public static void main(String[] args) {
            Linked_list list = new Linked_list();
            list.insertAtBeginning(3);
            list.insertAtBeginning(2);
            list.insertAtBeginning(1);
            list.displayAllNodes();
            list.insertAtEnd(4);
            list.insertAtEnd(5);
```

```java
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        Linked_list list = new Linked_list();
        list.insertAtBeginning(3);
        list.insertAtBeginning(2);
        list.insertAtBeginning(1);
        list.displayAllNodes();
        list.insertAtEnd(4);
        list.insertAtEnd(5);
        list.displayAllNodes();
        list.insertAtPosition(0, 1);
        list.insertAtPosition(6, 7);
        list.displayAllNodes();
        list.deleteNode(0);
        list.deleteNode(6);
        list.displayAllNodes();
        list.updateNode(1, 10);
        list.updateNode(5, 20);
        list.displayAllNodes();
    }
```

Problems @ Javadoc Declaration Console

<terminated> Linked_list [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.wir

```
1 -> 2 -> 3 -> null
1 -> 2 -> 3 -> 4 -> 5 -> null
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
1 -> 2 -> 3 -> 4 -> 5 -> null
10 -> 2 -> 3 -> 4 -> 20 -> null
```

Task 2: Stack and Queue Operations

1) Create a Custom Stack class with operations Push, Pop, Peek, and IsEmpty Demonstrate its LIFO behavior by pushing integers onto the stack, then popping and displaying them until the stack is empty.

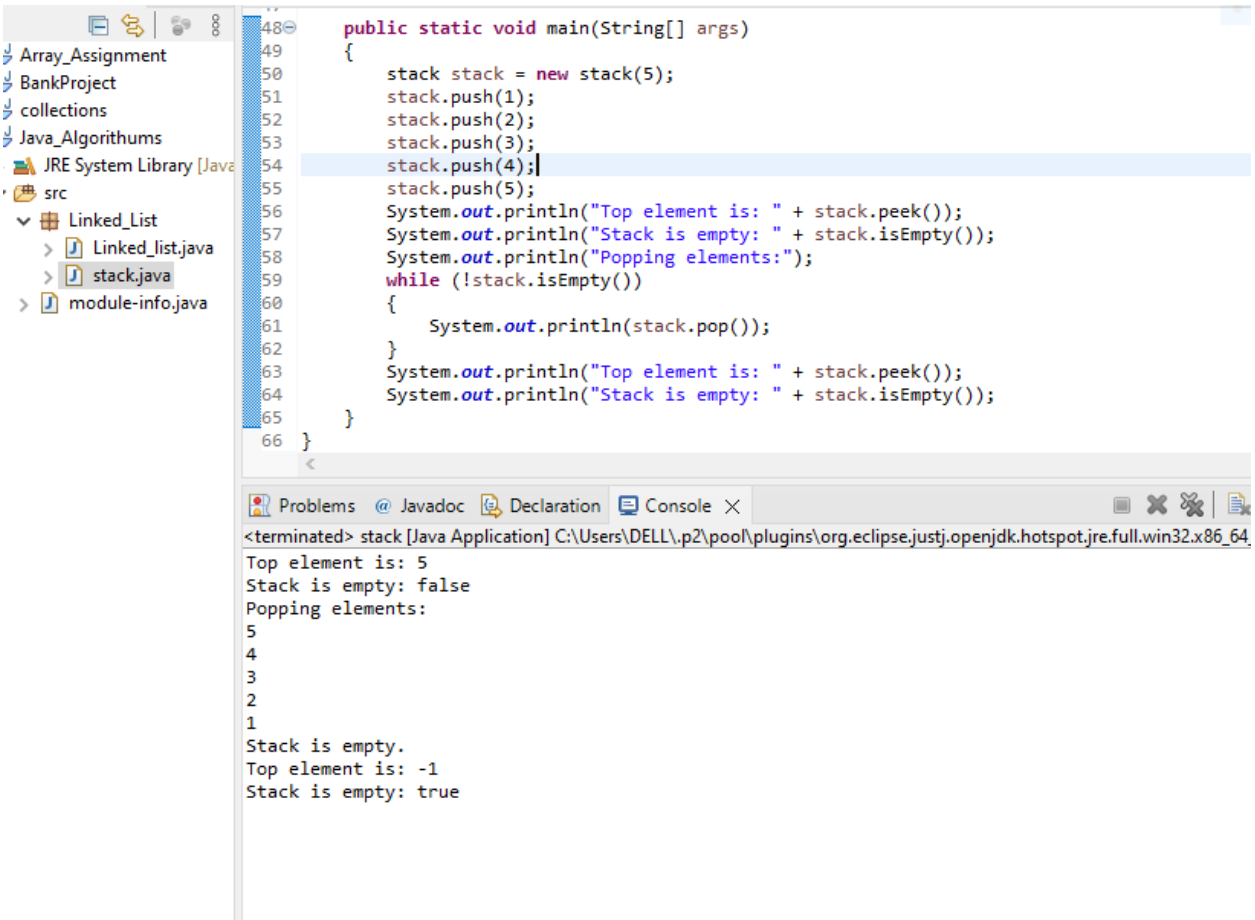
- Stack:it is linear data strcture , if fallows fifo concept,
- Initially top have -1
- It has push(), pop(), peek(), isEmplty() functions
- Use for reverse the element, reverse expression
- It can be implemented by linklist, Array ;
- Following stack implemented by using array data strcture;

IBank.java | HashTableDem... | Doctor.java | Recurtion.java | fibonacci_s... | Link

```java
package Linked_List;

public class stack
{
    private int maxSize;
    private int top;
    private int[] stackArray;

    public stack(int size)
    {
        maxSize = size;
        stackArray = new int[maxSize];
        top = -1;
    }

    public void push(int data)
    {
        if (top >= maxSize - 1)
        {
            System.out.println("Stack is full. Can't push " + data);
            return;
        }
        stackArray[++top] = data;
    }

    public int pop() {
        if (top < 0) {
            System.out.println("Stack is empty. Can't pop.");
            return -1;
        }
        return stackArray[top--];
    }
```

Problems | Javadoc | Declaration | Console

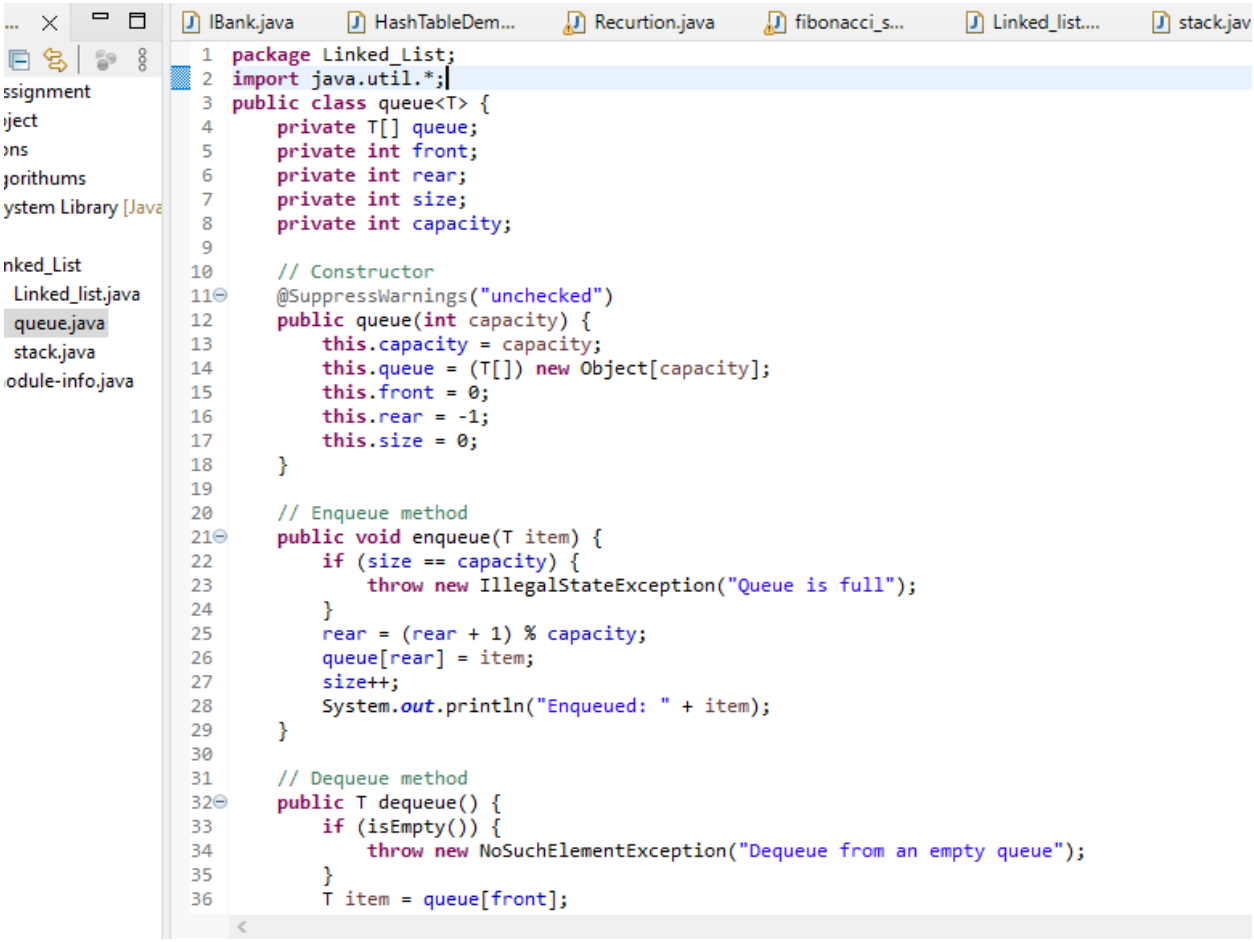
```java
        return stackArray[top--];
    }

    public int peek()
    {
        if (top < 0) {
            System.out.println("Stack is empty.");
            return -1;
        }
        return stackArray[top];
    }

    public boolean isEmpty()
    {
        return (top == -1);
    }

    public static void main(String[] args)
    {
        stack stack = new stack(5);
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        System.out.println("Top element is: " + stack.peek());
        System.out.println("Stack is empty: " + stack.isEmpty());
        System.out.println("Popping elements:");
        while (!stack.isEmpty())
        {
            System.out.println(stack.pop());
        }
        System.out.println("Top element is: " + stack.peek());
        System.out.println("Stack is empty: " + stack.isEmpty());
    }
}
```



2) Develop a CustomQueue class with methods for Enqueue, Dequeue, Peek, and IsEmpty. Show how your queue can handle different data types by enqueuing strings and integers, then dequeuing and displaying them to confirm FIFO order.

-> Generics means parameterized types.
-> The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create

classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.
-> We can also write generic functions that can be called with different types of arguments based on the type of arguments passed to
the generic method. The compiler handles each method.




```java
package Linked_List;
import java.util.*;
public class queue<T> {
    private T[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

    // Constructor
    @SuppressWarnings("unchecked")
    public queue(int capacity) {
        this.capacity = capacity;
        this.queue = (T[]) new Object[capacity];
        this.front = 0;
        this.rear = -1;
        this.size = 0;
    }

    // Enqueue method
    public void enqueue(T item) {
        if (size == capacity) {
            throw new IllegalStateException("Queue is full");
        }
        rear = (rear + 1) % capacity;
        queue[rear] = item;
        size++;
        System.out.println("Enqueued: " + item);
    }

    // Dequeue method
    public T dequeue() {
        if (isEmpty()) {
            throw new NoSuchElementException("Dequeue from an empty queue");
        }
        T item = queue[front];
```

age Ex...

rray_Assignment
ankProject
ollections
ıva_Algorithums
JRE System Library [Java
src
Linked_List
Linked_list.java
queue.java
stack.java
module-info.java

IBank.java | HashTableDem... | Recurtion.java | fibonacci_s...

```java
    // IsEmpty method
    public boolean isEmpty() {
        return size == 0;
    }

    public static void main(String[] args) {
        // Demonstrating the queue with different data types
        queue<Object> queue = new queue<>(5);

        // Enqueue integers and strings
        queue.enqueue(10);
        queue.enqueue("hello");
        queue.enqueue(20);
        queue.enqueue("world");

        // Dequeue elements to confirm FIFO order
        while (!queue.isEmpty()) {
            queue.dequeue();
        }
    }
}
```

Console

<terminated> queue [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.op

```
Enqueued: 10
Enqueued: hello
Enqueued: 20
Enqueued: world
Dequeued: 10
Dequeued: hello
Dequeued: 20
Dequeued: world
```

IBank.java | HashTableDem... | Recurtion.java | fibonacci_s... | Linked_list.... | stack.jav

nent

ıms
Library [Java

List
ed_list.java
e.java
.java
-info.java

```java
    // Dequeue method
    public T dequeue() {
        if (isEmpty()) {
            throw new NoSuchElementException("Dequeue from an empty queue");
        }
        T item = queue[front];
        front = (front + 1) % capacity;
        size--;
        System.out.println("Dequeued: " + item);
        return item;
    }

    // Peek method
    public T peek() {
        if (isEmpty()) {
            throw new NoSuchElementException("Peek from an empty queue");
        }
        return queue[front];
    }

    // IsEmpty method
    public boolean isEmpty() {
        return size == 0;
    }

    public static void main(String[] args) {
        // Demonstrating the queue with different data types
        queue<Object> queue = new queue<>(5);

        // Enqueue integers and strings
        queue.enqueue(10);
        queue.enqueue("hello");
        queue.enqueue(20);
        queue.enqueue("world");

        // Dequeue elements to confirm FIFO order
```

Task 3: Priority Queue Scenario:

Implement Priority Queue to manage the emergency room admission in the hospital.

Patients with higher urgency should be served before those with lower urgency.

Comparator Interface:

* The Comparator interface in Java is used to order the objects of a user-defined class.

* It includes the compare method that must be implemented to define the sorting logic.

Anonymous Class:

* new Comparator<Patient>() { ... } creates an anonymous class that implements the Comparator<Patient> interface.

* Inside the curly braces { ... }, you provide the implementation of the compare method.

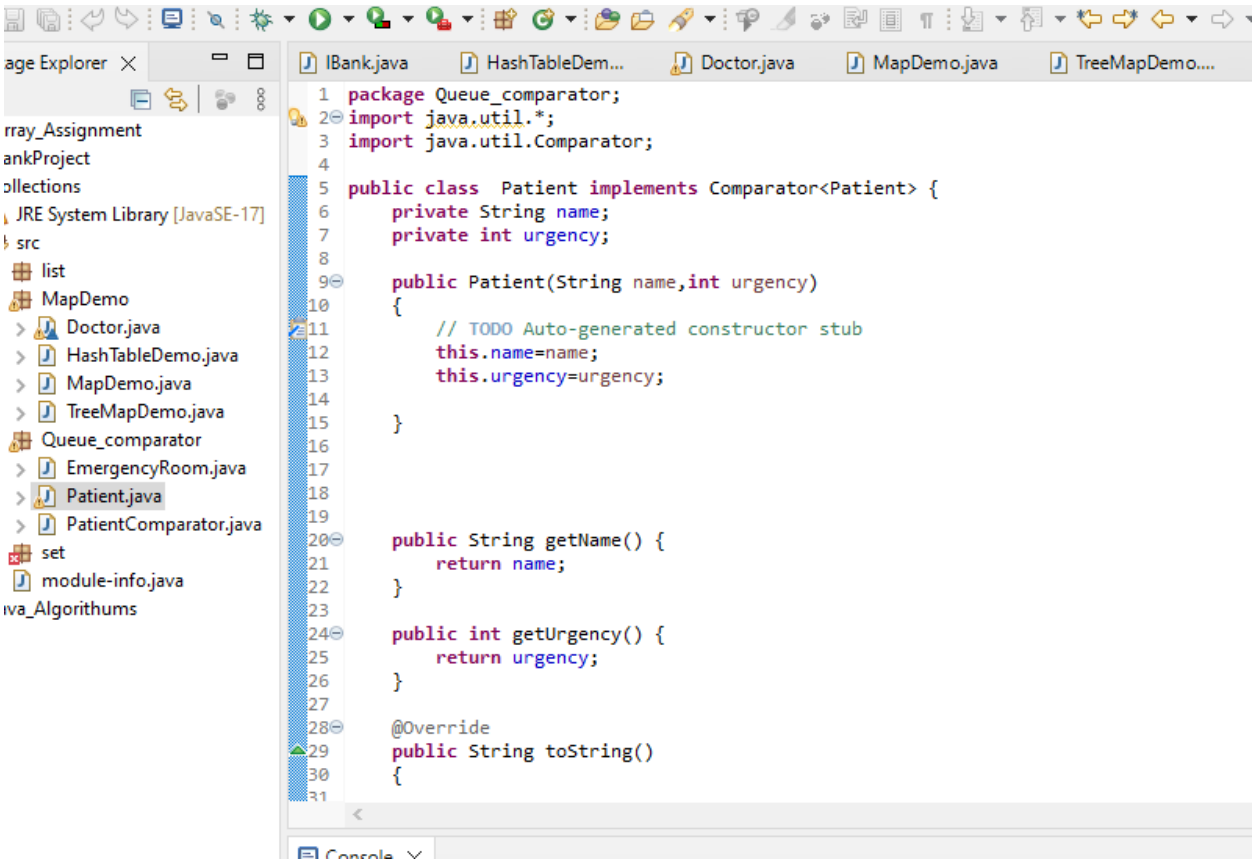
compare Method:

@ The compare method takes two Patient objects and returns an integer:

@ A negative integer if the first argument is less than the second.

@ Zero if the first argument is equal to the second.

@ A positive integer if the first argument is greater than the second.


age Explorer
rray_Assignment
ankProject
ollections
JRE System Library [JavaSE-17]
src
list
MapDemo
Doctor.java
HashTableDemo.java
MapDemo.java
TreeMapDemo.java
Queue_comparator
EmergencyRoom.java
Patient.java
PatientComparator.java
set
module-info.java
ıva_Algorithums
IBank.java
HashTableDem...
Doctor.java
MapDemo.java
TreeMapDemo....
package Queue_comparator;
import java.util.*;
import java.util.Comparator;
public class Patient implements Comparator<Patient> {
private String name;
private int urgency;
public Patient(String name,int urgency)
{
// TODO Auto-generated constructor stub
this.name=name;
this.urgency=urgency;
}
public String getName() {
return name;
}
public int getUrgency() {
return urgency;
}
@Override
public String toString()
{
Console

IBank.java | HashTableDem... | Doctor.java | MapDemo.java | TreeMapDemo....

```java
        return urgency;
    }

    @Override
    public String toString()
    {

        return name + ":" +urgency;
    }




    public static void main(String[] args)
    {
        // TODO Auto-generated method stub

    }



    @Override
    public int compare(Patient p1, Patient p2)
    {
        // TODO Auto-generated method stub
        return Integer.compare(p2.getUrgency(), p1.getUrgency());
    }

}
```

Console

<terminated> Patient [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x

Package Explorer

- Array_Assignment
- BankProject
- collections
  - JRE System Library [JavaSE-17]
  - src
    - list
    - MapDemo
      - Doctor.java
      - HashTableDemo.java
      - MapDemo.java
      - TreeMapDemo.java
    - Queue_comparator
      - EmergencyRoom.java
      - Patient.java
      - PatientComparator.java
    - set
    - module-info.java
- Java_Algorithums

IBank.java | HashTableDem... | Doctor.java | MapDemo.java | TreeMapDemo.... | Patient.java

```java
package Queue_comparator;
import java.util.*;
public class EmergencyRoom {

    public static void main(String[] args)
    {
        // TODO Auto-generated method stub

        PriorityQueue<Patient> queue = new PriorityQueue<>(new PatientComparator());

        queue.add(new Patient("Narayan", 3));
        queue.add(new Patient("vitthal", 1));
        queue.add(new Patient("More", 2));

        while (!queue.isEmpty())
        {
            Patient patient = queue.poll();
            System.out.println("Patient: " + patient);
        }
```

Console

```
<terminated> EmergencyRoom [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_6
Patient: Narayan:3
Patient: More:2
Patient: vitthal:1
```