Experiment – 1: Data structures using Java

1. A) Programs for Stack.

Prog-1:

```
import java.util.*; class StackDemo
{
static void showpush(Stack st, int a)
{
st.push(new Integer(a));
System.out.println("push(" + a + ")");
System.out.println("stack: " + st);
}
static void showpop(Stack st)
{
System.out.print("pop -> ");
Integer a =(Integer) st.pop();
System.out.println(a);
System.out.println("stack: " + st);
}
public static void main(String args[])
{
Stack st = new Stack();  System.out.println("stack: " + st);
showpush(st, 42); showpush(st, 66);  showpush(st, 99);  showpop(st);  showpop(st);
showpop(st);
 try
{
showpop(st);

} catch (EmptyStackException e)
{
System.out.println("empty stack");
}
}
}
```

Output:

The following is the output produced by the program; notice how the exceptionhandler for **EmptyStackException** is caught so that you can gracefully handle a stack underflow:

stack: [ ] push(42) stack: [42]
push(66)

stack: [42, 66]
push(99)

stack: [42, 66, 99] pop -> 99 stack: [42, 66] pop -> 66
stack: [42]

pop -> 42
stack: [ ] pop -> empty stack

Prog-2:

Stack Implementation in Java using Array

```
public class StackDemo
{
private static final int capacity = 3;
int arr[] = new int[capacity];
int top = -1;
public void push(int pushedElement)
{
if (top < capacity - 1)
{
top++;
arr[top] = pushedElement;
System.out.println("Element " + pushedElement + " is pushed toStack!");  printElements();
}
else
{
System.out.println("Stack Overflow !");
}
}
public void pop()
{
if (top >= 0)
{
top--;
System.out.println("Pop operation done !");
}
else
{
System.out.println("Stack Underflow !");
}
}
public void printElements()
{
if (top >= 0)
{
System.out.println("Elements in stack :");
for (int i = 0; i <= top; i++)
{
```

```
System.out.println(arr[i]);
}
}
}
public static void main(String[] args)
{
StackDemo stackDemo = new StackDemo();
stackDemo.pop();
stackDemo.push(23);
stackDemo.push(2);
stackDemo.push(73);
stackDemo.push(21);
stackDemo.pop();
stackDemo.pop();
stackDemo.pop();
stackDemo.pop();
}
}
```

Output:

Stack Underflow !
Element 23 is pushed to Stack !
Elements in stack :
23
Element 2 is pushed to Stack !
Elements in stack :
23 2
Element 73 is pushed to Stack !
Elements in stack :
23
2 73
Stack Overflow !
Pop operation done !
Pop operation done !
Pop operation done !
Stack Underflow !

Prog-3:

/*Stack collection implements Last-In Fist-Out behavior on items stored within it.*/

```
import java.util.*;
public class StackDemo
{
static String newLine = System.getProperty("line.separator");
public static void main(String[] args)
{
System.out.println(newLine + "Stack in Java" + newLine);
System.out.println("----------" + newLine);
System.out.println("Adding items to the Stack1" + newLine);
//Creating stack object
Stack stack = new Stack();
//you add elements to stack using push method stack.push("Java");
stack.push(".NET");
stack.push("Javascript");
stack.push("HTML5");
stack.push("Hadoop");
System.out.println(newLine + "Items in the stack..." + stack + newLine);
System.out.println("----------" + newLine);

//you remove elements from the stack using pop method
//pop would remove the last element and would return it
//its last-in first-out behavior
//so here it would return hadoop

System.out.println("removing from stack, using pop method - " + stack.pop()+  newLine);
//another pop would return HTML5

System.out.println("removing from stack, using pop method - " + stack.pop()+
                                newLine); System.out.println("----------" + newLine);

//Peek returns the top element in the stack but does not remove it
//so here after html5 and hadoop are returned, the top item is Javascript

 System.out.println("returning the top element in the stack using peekmethod - " +  stack.peek() +
                                                                                    newLine);
}
}
```

1. B) Programs for Queue.

Prog-1:

/* Queues using Java*/

```
public class QueueDemo
{
private static final int capacity = 3;
int arr[] = new int[capacity];
int size = 0;
int top = -1;
int rear = 0;
public void push(int pushedElement)
{
if (top < capacity - 1)
{
top++;
arr[top] = pushedElement;
System.out.println("Element " + pushedElement + " is pushed to Queue !"); display();
}
else
{
System.out.println("Overflow !");
}
}
public void pop()
{
if (top >= rear)
{
rear++;
System.out.println("Pop operation done !");  display();
}
else
{
System.out.println("Underflow !");
}
}
public void display()
{
if (top >= rear)
{
System.out.println("Elements in Queue : ");
```

```
for (int i = rear; i <= top; i++)
{
System.out.println(arr[i]);
}
}
}
public static void main(String[] args)
{
QueueDemo queueDemo = new QueueDemo();
queueDemo.pop();
queueDemo.push(23);
queueDemo.push(2);
queueDemo.push(73);
queueDemo.push(21);
queueDemo.pop();
queueDemo.pop();
queueDemo.pop();
queueDemo.pop();
}
}
```

Output:

```
Underflow !
Element 23 is pushed to Queue !
Elements in Queue : 23
Element 2 is pushed to Queue !
Elements in Queue :
23
2
Element 73 is pushed to Queue !
Elements in Queue :
23
2  73 Overflow !
Pop operation done !
Elements in Queue :
2
73
Pop operation done !
Elements in Queue :
73
Pop operation done !
Underflow !
```

Prog-2:

/*Queues using java */

```
import java.io.*;
class QueueAction
{
BufferedReader is = new BufferedReader(new InputStreamReader(System.in));
int items[];
int i, front = 0, rear = 0, noOfItems, item, count = 0;

void getdata()
{
try
{
System.out.println("Enter the Limit :");
noOfItems = Integer.parseInt(is.readLine());
items = new int[noOfItems];
} catch (Exception e)
{
System.out.println(e.getMessage());
}
}

void enqueue()
{
try
{
if (count < noOfItems)
{
System.out.println("Enter Queue Element :");
item = Integer.parseInt(is.readLine());
items[rear] = item;
rear++;
count++;
}
else
{
System.out.println("Queue Is Full");
}
} catch (Exception e)
{
System.out.println(e.getMessage());
```

```
}
}

void dequeue()
{
if (count != 0)
{
System.out.println("Deleted Item :" + items[front]);
front++;
count--;
}
else
{
System.out.println("Queue IS Empty");
}
if (rear == noOfItems)
{
rear = 0;
}
}

void display()
{
int m = 0;
if (count == 0)
{
System.out.println("Queue IS Empty");
}
else
{
for(i = front; m < count; i++, m++)
{
 System.out.println(" " + items[i]);
}
}
}
}

class QueueProgram
{
public static void main(String arg[])
{
DataInputStream get = new DataInputStream(System.in);
```

```
int choice;
QueueAction queue = new QueueAction();
queue.getdata();
System.out.println("Queue\n\n");
try
{
do
{
System.out.println("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
System.out.println("Enter the Choice : ");
choice = Integer.parseInt(get.readLine());
switch (choice)
{
case 1:  queue.enqueue();
break;

case 2:  queue.dequeue();
break;

case 3:  queue.display();
break;
}
} while (choice != 4);
} catch (Exception e)
{
System.out.println(e.getMessage());
}
}
}
```

Output:

```
Enter the Limit : 4

Queue

1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 1
Enter Queue Element : 45
1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 1
Enter Queue Element : 67
1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 1
Enter Queue Element : 89
1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 1
Enter Queue Element : 567
1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 1
Queue Is Full
1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter the Choice : 3 45 67 89 567
```

1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 2
Deleted Item :45
1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 2
Deleted Item :67
1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 2
Deleted Item :89
1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 2
Deleted Item :567
1.Enqueue
2.Dequeue
3.Display 4.Exit

Enter the Choice : 2
Queue IS Empty
1.Enqueue
2.Dequeue
3.Display
4.Exit

Enter the Choice : 4

1. C) Linked lists using java

Prog-1:

```
/*
 *  Java Program to Implement Singly Linked List
 */

import java.util.Scanner;

/*  Class Node  */
class Node
{
    protected int data;
    protected Node link;

    /*  Constructor  */
    public Node()
    {
        link = null;
        data = 0;
    }
    /*  Constructor  */
    public Node(int d,Node n)
    {
        data = d;
        link = n;
    }
    /*  Function to set link to next Node  */
    public void setLink(Node n)
    {
        link = n;
    }
    /*  Function to set data to current Node  */
    public void setData(int d)
    {
        data = d;
    }
    /*  Function to get link to next node  */
    public Node getLink()
    {
        return link;
    }
    /*  Function to get data from current Node  */
    public int getData()
    {
```

```
        return data;
    }
}

/* Class linkedList */
class linkedList
{
    protected Node start;
    protected Node end ;
    public int size ;

    /*  Constructor  */
    public linkedList()
    {
        start = null;
        end = null;
        size = 0;
    }
    /*  Function to check if list is empty  */
    public boolean isEmpty()
    {
        return start == null;
    }
    /*  Function to get size of list  */
    public int getSize()
    {
        return size;
    }
    /*  Function to insert an element at begining  */
    public void insertAtStart(int val)
    {
        Node nptr = new Node(val, null);
        size++ ;
        if(start == null)
        {
            start = nptr;
            end = start;
        }
        else
        {
            nptr.setLink(start);
            start = nptr;
        }
    }
    /*  Function to insert an element at end  */
    public void insertAtEnd(int val)
```

```
{
   Node nptr = new Node(val,null);
   size++ ;
   if(start == null)
   {
      start = nptr;
      end = start;
   }
   else
   {
      end.setLink(nptr);
      end = nptr;
   }
}
/*  Function to insert an element at position  */
public void insertAtPos(int val , int pos)
{
   Node nptr = new Node(val, null);
   Node ptr = start;
   pos = pos - 1 ;
   for (int i = 1; i < size; i++)
   {
      if (i == pos)
      {
         Node tmp = ptr.getLink() ;
         ptr.setLink(nptr);
         nptr.setLink(tmp);
         break;
      }
      ptr = ptr.getLink();
   }
   size++ ;
}
/*  Function to delete an element at position  */
public void deleteAtPos(int pos)
{
   if (pos == 1)
   {
      start = start.getLink();
      size--;
      return ;
   }
   if (pos == size)
   {
      Node s = start;
      Node t = start;
```

```
        while (s != end)
        {
            t = s;
            s = s.getLink();
        }
        end = t;
        end.setLink(null);
        size --;
        return;
    }
    Node ptr = start;
    pos = pos - 1 ;
    for (int i = 1; i < size - 1; i++)
    {
        if (i == pos)
        {
            Node tmp = ptr.getLink();
            tmp = tmp.getLink();
            ptr.setLink(tmp);
            break;
        }
        ptr = ptr.getLink();
    }
    size-- ;
}
/*  Function to display elements  */
public void display()
{
    System.out.print("\nSingly Linked List = ");
    if (size == 0)
    {
        System.out.print("empty\n");
        return;
    }
    if (start.getLink() == null)
    {
        System.out.println(start.getData() );
        return;
    }
    Node ptr = start;
    System.out.print(start.getData()+ "->");
    ptr = start.getLink();
    while (ptr.getLink() != null)
    {
        System.out.print(ptr.getData()+ "->");
        ptr = ptr.getLink();
```

```
      }
      System.out.print(ptr.getData()+ "\n");
   }
}

/*  Class SinglyLinkedList  */
public class SinglyLinkedList
{
   public static void main(String[] args)
   {
      Scanner scan = new Scanner(System.in);
      /* Creating object of class linkedList */
      linkedList list = new linkedList();
      System.out.println("Singly Linked List Test\n");
      char ch;
      /*  Perform list operations  */
      do
      {
         System.out.println("\nSingly Linked List Operations\n");
         System.out.println("1. insert at begining");
         System.out.println("2. insert at end");
         System.out.println("3. insert at position");
         System.out.println("4. delete at position");
         System.out.println("5. check empty");
         System.out.println("6. get size");
         int choice = scan.nextInt();
         switch (choice)
         {
         case 1 :
            System.out.println("Enter integer element to insert");
            list.insertAtStart( scan.nextInt() );
            break;
         case 2 :
            System.out.println("Enter integer element to insert");
            list.insertAtEnd( scan.nextInt() );
            break;
         case 3 :
            System.out.println("Enter integer element to insert");
            int num = scan.nextInt() ;
            System.out.println("Enter position");
            int pos = scan.nextInt() ;
            if (pos <= 1 || pos > list.getSize() )
               System.out.println("Invalid position\n");
            else
               list.insertAtPos(num, pos);
            break;
```

```
        case 4 :
           System.out.println("Enter position");
           int p = scan.nextInt() ;
           if (p < 1 || p > list.getSize() )
              System.out.println("Invalid position\n");
           else
              list.deleteAtPos(p);
           break;
        case 5 :
           System.out.println("Empty status = "+ list.isEmpty());
           break;
        case 6 :
           System.out.println("Size = "+ list.getSize() +" \n");
           break;
         default :
           System.out.println("Wrong Entry \n ");
           break;
        }
        /*  Display List  */
        list.display();
        System.out.println("\nDo you want to continue (Type y or n) \n");
        ch = scan.next().charAt(0);
      } while (ch == 'Y'|| ch == 'y');
   }
}
```

Output:

```
Singly Linked List Test

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
5
Empty status = true

Singly Linked List = empty

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
1
Enter integer element to insert
5

Singly Linked List = 5

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
```

6. get size
1
Enter integer element to insert
7

Singly Linked List = 7->5

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
2
Enter integer element to insert
4

Singly Linked List = 7->5->4

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
2
Enter integer element to insert
2

Singly Linked List = 7->5->4->2

Do you want to continue (Type y or n)

y

```
Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
1
Enter integer element to insert
9

Singly Linked List = 9->7->5->4->2

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
3
Enter integer element to insert
3
Enter position
3

Singly Linked List = 9->7->3->5->4->2

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
```

```
6. get size
3
Enter integer element to insert
2
Enter position
2

Singly Linked List = 9->2->7->3->5->4->2

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
6
Size = 7


Singly Linked List = 9->2->7->3->5->4->2

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
4

Singly Linked List = 9->2->7->5->4->2

Do you want to continue (Type y or n)
```

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
2

Singly Linked List = 9->7->5->4->2

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
1

Singly Linked List = 7->5->4->2

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty

6. get size
4
Enter position
3

Singly Linked List = 7->5->2

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
1

Singly Linked List = 5->2

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
2

Singly Linked List = 5

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
1

Singly Linked List = empty

Do you want to continue (Type y or n)

y

Singly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
5
Empty status = true

Singly Linked List = empty

Do you want to continue (Type y or n)

n

Prog-2:

```
package com.java2novice.ds.linkedlist;
public class SinglyLinkedListImpl<T>
{
private Node<T> head;
private Node<T> tail;

public void add(T element)
{
Node<T> nd = new Node<T>();
nd.setValue(element);

System.out.println("Adding: "+element);
/** * check if the list is empty */
if(head == null)
{
//since there is only one element, both head and
//tail points to the same object.
head = nd;
tail = nd;
}
else
{
//set current tail next link to new node
tail.setNextRef(nd);
//set tail as newly created node
tail = nd;
}
}

public void addAfter(T element, T after)
{
Node<T> tmp = head;
Node<T> refNode = null;
System.out.println("Traversing to all nodes..");
/** * Traverse till given element */
while(true)
{
if(tmp == null)
{
break;
}
```

```
if(tmp.compareTo(after) == 0)
{
//found the target node, add after this node
refNode = tmp; break;
}
 tmp = tmp.getNextRef();
}

if(refNode != null)
{
//add element after the target node
Node<T> nd = new Node<T>();
nd.setValue(element);
nd.setNextRef(tmp.getNextRef());
if(tmp == tail)
{
tail = nd;
}
 tmp.setNextRef(nd);
}
else
{
System.out.println("Unable to find the given element...");
}
}

public void deleteFront()
{
if(head == null)
{
System.out.println("Underflow...");
}

Node<T> tmp = head;
head = tmp.getNextRef();
if(head == null)
{
tail = null;
}

System.out.println("Deleted: "+tmp.getValue());
```

```
}

public void deleteAfter(T after)
{
Node<T> tmp = head;
Node<T> refNode = null;
System.out.println("Traversing to all nodes..");
/** * Traverse till given element */
while(true)
{
if(tmp == null)
{
break;
}

if(tmp.compareTo(after) == 0)
{
//found the target node, add after this node
refNode = tmp; break;
}

tmp = tmp.getNextRef();

}

if(refNode != null)
{ tmp = refNode.getNextRef();
refNode.setNextRef(tmp.getNextRef());
if(refNode.getNextRef() == null)
{
tail = refNode;
}

System.out.println("Deleted: "+tmp.getValue());
}
else
{
System.out.println("Unable to find the given element...");
}
}

public void traverse()
{
```

```
Node<T> tmp = head;
while(true)
{
if(tmp == null)
{
break;
}
System.out.println(tmp.getValue());
tmp = tmp.getNextRef();
}
}

public static void main(String a[])
{
SinglyLinkedListImpl<Integer> sl = new SinglyLinkedListImpl<Integer>();
sl.add(3);
sl.add(32);
sl.add(54);
sl.add(89);
sl.addAfter(76, 54);
sl.deleteFront();
sl.deleteAfter(76);
sl.traverse();
}
}

class Node<T> implements Comparable<T>
{
private T value;
private Node<T> nextRef;
public T getValue()
{
return value;
}

public void setValue(T value)
{
this.value = value;
}
public Node<T> getNextRef()
{
return nextRef;
}
```

```
public void setNextRef(Node<T> ref)
{
this.nextRef = ref;
}

//@Override

public int compareTo(T arg)
{
if(arg == this.value)
{
return 0;
 }
else
{
return 1;
}
}
}
```

1. D) SET using Java

Prog-1:

```
import java.util.*;

public class SetDemo
{
public static void main(String args[])
{
int count[] = {34, 22,10,60,30,22};

Set<Integer> set = new HashSet<Integer>();
try
{
for(int i = 0; i<5; i++)
{
set.add(count[i]);
}

System.out.println(set);
TreeSet sortedSet = new TreeSet<Integer>(set);
System.out.println("The sorted list is:");
System.out.println(sortedSet);
System.out.println("The First element of the set is: "+ (Integer)sortedSet.first());
System.out.println("The last element of the set is: "+ (Integer)sortedSet.last());
}catch(Exception e)
{
}
}
}
```

Output:-

```
[amrood]$ java SetDemo [34, 30, 60, 10, 22]
The sorted list is: [10, 22, 30, 34, 60]
The First element of the set is: 10
The last element of the set is: 60
```

Prog-2:

```
import java.util.HashSet;
import java.util.Set;
import java.util.TreeSet;

public class Main
{
public static void main(String args[])
{
Set<String> hs = new HashSet<String>();
hs.add("one");
hs.add("two");
hs.add("three");
System.out.println("Here is the HashSet: " + hs);

if (!hs.add("three"))
System.out.println("Attempt to add duplicate. " + "Set is unchanged: "+ hs);

TreeSet<Integer> ts = new TreeSet<Integer>();
ts.add(8);
ts.add(19);
ts.add(-2);
ts.add(3);

System.out.println(ts);
System.out.println("First element in ts: " + ts.first());
System.out.println("Last element in ts: " + ts.last());
System.out.println("First element > 15: " + ts.higher(15));
System.out.println("First element < 15: " + ts.lower(15));

}
}
```

Output:-

```
Here is the HashSet: [one, two, three]
Attempt to add duplicate. Set is unchanged: [one, two, three]
[-2, 3, 8, 19]
First element in ts: -2
Last element in ts: 19
First element > 15: 19
First element < 15: 8
```

1. E) Map using Java

```
/*  Simple Java HashMap example

This simple Java Example shows how to use Java HashMap. It also describes how to add
something to HashMap and how to retrieve the value added from HashMap. */

import java.util.HashMap;

public class JavaHashMapExample
{
public static void main(String[] args)
{

//create object of HashMap
 HashMap<String, Integer> hMap = new HashMap<String, Integer>();

/*Add key value pair to HashMap usingObject put(Object key, Object value) method of Java
HashMap class, where key and value both are objectsput method returns Object which is either
the value previously tied to the key or null if no value mapped to the key.*/

hMap.put("One", new Integer(1));

hMap.put("Two", new Integer(2));

/* Please note that put method accepts Objects. Java Primitive values CAN NOTbe added directly
to HashMap. It must be converted to corresponding wrapper class first.  */

//retrieve value using Object get(Object key) method of Java HashMap class

Object obj = hMap.get("One");

System.out.println(obj);

/* Please note that the return type of get method is an Object. The value must be casted to the
original class. */

}
}
```

/* Output of the program would be 1 */