Name: Vangipuram Srinivasa Sarath Chandra
Regd No: 21BCE9853
Title: Digital Assignment Java
Date: 11/June/2022

1. Develop a java program to implement List interface by using stack class and linkedlist classes. (It should include all the methods of List interface). [5M]

List of methods used in stack & Linked List :

addAll	retainAll	removeAll	containsAll
remove	clear	add	addFirst
addLast	get	set	indexOf
lastIndexOf	size	toArray	iterator

TLDR;

1. CRUD operations
2. Set operations
3. Iterator, size, toArray

Linked List

Linked List: 1: Linked List is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays. It also has a few disadvantages like the nodes cannot be accessed directly instead we need to start from the head and follow through the link to reach a node we wish to access.

```java
import java.util.*;

public class LinkedListAssign {

    // creating a linked list using generics
    static <T> LinkedList<T> createLinkedList(T... ele) {
        LinkedList<T> newList = new LinkedList<T>();
        for (T el : ele) {
            newList.add(el);
        }
        return newList;
    }

    public static void main(String[] args) {

        LinkedList<String> list1 = createLinkedList("A", "B", "C", "D", "E", "F", "G");
        LinkedList<String> list2 = createLinkedList("F", "G", "H", "I", "J");
        LinkedList<String> list3 = createLinkedList("F", "G", "I");

        System.out.println("list1: " + list1);
        System.out.println("list2: " + list2);
        System.out.println("list3: " + list3);

        // union
        list1.addAll(list2);
        System.out.println("list1 Union List2 -> list1: " + list1);

        // intersection
        list2.retainAll(list3);
        System.out.println("list2 Intersection List3 -> list2: " + list2);

        // difference
        list2.removeAll(list3);
```

```java
        System.out.println("list2 Difference List3 -> list2: " + list2);

        // subset
        System.out.println("list1 is subset of list3: " + list1.containsAll(list3));

        // remove
        list3.remove("A");
        System.out.println("Remove A from list3 -> list3: " + list3);

        // remove all
        // list3.removeAll();
        list3.clear();
        System.out.println("remove all -> list3: " + list3);

        // add
        list1.add("A");
        list1.add("B");
        System.out.println("add A and B -> list1: " + list1);

        // add at index
        list1.add(1, "C");
        System.out.println("Add at index 1 -> list1: " + list1);

        // add first and last
        list1.addFirst("F");
        System.out.println("Add first 'F'-> list1: " + list1);
        list1.addLast("G");
        System.out.println("Add last 'G'-> list1: " + list1);

        // get
        System.out.println("list1.get(0): " + list1.get(0));

        // set
        list1.set(0, "D");
        System.out.println("set method (0, 'D') -> list1: " + list1);

        // index of
        System.out.println("list1.indexOf(\"B\"): " + list1.indexOf("B"));

        // last index of
        System.out.println("list1.lastIndexOf(\"B\"): " + list1.lastIndexOf("B"));

        // size
        System.out.println("list1.size(): " + list1.size());

        // toArray
        Object[] array = list1.toArray();
        System.out.println("list1.toArray(): " + Arrays.toString(array));

        // iterate
        System.out.print("Iterate list1: ");
        for (String s : list1) {
            System.out.print(s + ' ');
        }

        // iterate using iterator
        System.out.print("\nIterate list1 using iterator: ");
        Iterator<String> itr = list1.iterator();
        while (itr.hasNext()) {
            System.out.print(itr.next() + ' ');
        }
    }
}
```

Output:

```
list1: [A, B, C, D, E, F, G]
list2: [F, G, H, I, J]
list3: [F, G, I]
list1 Union List2 -> list1: [A, B, C, D, E, F, G, F, G, H, I, J]
list2 Intersection List3 -> list2: [F, G, I]
list2 Difference List3 -> list2: []
list1 is subset of list3: true
Remove A from list3 -> list3: [F, G, I]
remove all -> list3: []
add A and B -> list1: [A, B, C, D, E, F, G, F, G, H, I, J, A, B]
Add at index 1 -> list1: [A, C, B, C, D, E, F, G, F, G, H, I, J, A, B]
Add first 'F'-> list1: [F, A, C, B, C, D, E, F, G, F, G, H, I, J, A, B]
Add last 'G'-> list1: [F, A, C, B, C, D, E, F, G, F, G, H, I, J, A, B, G]
list1.get(0): F
set method (0, 'D') -> list1: [D, A, C, B, C, D, E, F, G, F, G, H, I, J, A, B, G]
list1.indexOf("B"): 3
list1.lastIndexOf("B"): 15
list1.size(): 17
list1.toArray(): [D, A, C, B, C, D, E, F, G, F, G, H, I, J, A, B, G]
Iterate list1: D A C B C D E F G F G H I J A B G
Iterate list1 using iterator: D A C B C D E F G F G H I J A B G
```

Explanation:
As the execution of the program starts i.e the main method is called,
we create a new Linked List object and add some elements to it.
then we use different methods like: add, reomve, clear, etc to manipulate the Linked List object.

Stack

Stack: is a Linear data structure which is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek.

All the methods metioned in the above LinkedList hold true here too and are similar if not same in every aspect. Altough they acheive the same things they just go about it in a different way here are some methods which aren't implemented in LinkedList and are exclusively in stack:

- pop()
- push()
- peek()

```java
import java.util.*;

public class StackAssign {
    // creating a stack using generics
    static <T> Stack<T> createStack(T... ele) {
        Stack<T> newStack = new Stack<T>();
        for (T el : ele) {
            newStack.push(el);
        }
        return newStack;
    }

    public static void main(String[] args) {
        Stack<String> stack1 = createStack("A", "B", "C", "D", "E", "F", "G");
        Stack<String> stack2 = createStack("H", "I", "J", "K", "L", "M", "N");

        System.out.println("stack1: " + stack1);

        // peek
        System.out.println("peek: " + stack1.peek());

        // pop
        System.out.println("pop: " + stack1.pop());
        System.out.println("stack1: " + stack1);

        // push
        stack1.push("H");
        System.out.println("push H -> stack1: " + stack1);

        // add
        stack1.add("I");
```

```
        System.out.println("add I -> stack1: " + stack1);

        // remove
        stack1.remove("I");
        System.out.println("remove I -> stack1: " + stack1);

        // add all
        stack1.addAll(stack2);
        System.out.println("add all stack2 -> stack1: " + stack1);

        // get
        System.out.println("get(2): " + stack1.get(2));

        // set
        stack1.set(2, "O");

        // size
        System.out.println("size: " + stack1.size());

        // to array
        String[] array = stack1.toArray(new String[stack1.size()]);
        System.out.println("to array: " + Arrays.toString(array));

        // contains
        System.out.println("contains H: " + stack1.contains("H"));

        // iterate
        System.out.print("iterate: ");
        for (String el : stack1) {
            System.out.print(el + " ");
        }

        // iterate throught iterator
        System.out.print("\niterate through iterator: ");
        Iterator<String> it = stack1.iterator();
        while (it.hasNext()) {
            System.out.print(it.next() + " ");
        }

    }
}
```

Output:

```
stack1: [A, B, C, D, E, F, G]
peek: G
pop: G
stack1: [A, B, C, D, E, F]
push H -> stack1: [A, B, C, D, E, F, H]
add I -> stack1: [A, B, C, D, E, F, H, I]
remove I -> stack1: [A, B, C, D, E, F, H]
add all stack2 -> stack1: [A, B, C, D, E, F, H, H, I, J, K, L, M, N]
get(2): C
size: 14
to array: [A, B, O, D, E, F, H, H, I, J, K, L, M, N]
contains H: true
iterate: A B O D E F H H I J K L M N
iterate through iterator: A B O D E F H H I J K L M N
```

Explanation:
As the execution of the program starts i.e the main method is called,
we create a new Stack object and add some elements to it.
then we use different methods like: add, reomve, clear, etc to manipulate the Stack object.

2. Develp a java program to implement Set interface by using TreeSet class and EnumSet class.[10M]

List of methods used in the follwing TreeSet and EnumSet are:

addAll	retainAll	removeAll	containsAll
remove	clear	add	toArray
size	iterator		

Tree Set

Tree Set: uses tree (a non linear data structure) for storing data. The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit compartor is provided. This must be consistent with equals if it is to correctly implement the Set interface.

```
import java.util.*;

public class TreeSetAssign {

    // creating a stack using a method
    static <T> TreeSet<T> createTreeSet(T... ele) {
        TreeSet<T> newTreeSet = new TreeSet<T>();
        for (T el : ele) {
            newTreeSet.add(el);
        }
        return newTreeSet;
    }

    public static void main(String[] args) {
        TreeSet<String> treeSet1 = createTreeSet("A", "B", "C", "D", "E", "F", "G");
        TreeSet<String> treeSet2 = createTreeSet("F", "G", "H", "I", "J", "K", "L");

        System.out.println("treeSet1: " + treeSet1);
        System.out.println("treeSet2: " + treeSet2);

        // add
        treeSet1.add("H");
        System.out.println("add H -> treeSet1: " + treeSet1);

        // remove
        treeSet1.remove("H");
        System.out.println("remove H -> treeSet1: " + treeSet1);

        // add all
        TreeSet<String> addAll = new TreeSet<String>(treeSet1);
        addAll.addAll(treeSet2);
        System.out.println("addAll(treeSet1 + treeSet2): " + addAll);

        // union
        TreeSet<String> union = new TreeSet<String>(treeSet1);
        union.addAll(treeSet2);
        System.out.println("union: " + union);

        // intersection
        TreeSet<String> intersection = new TreeSet<String>(treeSet1);
        intersection.retainAll(treeSet2);
        System.out.println("intersection: " + intersection);
```

```java
        // difference
        TreeSet<String> difference = new TreeSet<String>(treeSet1);
        difference.removeAll(treeSet2);
        System.out.println("difference: " + difference);

        // subset
        System.out.println("treeSet1 is subset of treeSet2: " + treeSet1.containsAll(treeSet2));

        // clear
        treeSet2.clear();
        System.out.println("clear treeSet2: " + treeSet2);

        // size
        System.out.println("size: " + treeSet1.size());

        // to array
        String[] array = treeSet1.toArray(new String[treeSet1.size()]);
        System.out.println("to array: " + Arrays.toString(array));

        // iterate
        System.out.print("iterate: ");
        for (String el : treeSet1) {
            System.out.print(el + " ");
        }

        // iterate throught iterator
        Iterator<String> iterator = treeSet1.iterator();
        System.out.print("\niterate through iterator: ");
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }
    }
}
```

Output:

```
treeSet1: [A, B, C, D, E, F, G]
treeSet2: [F, G, H, I, J, K, L]
add H -> treeSet1: [A, B, C, D, E, F, G, H]
remove H -> treeSet1: [A, B, C, D, E, F, G]
addAll(treeSet1 + treeSet2): [A, B, C, D, E, F, G, H, I, J, K, L]
union: [A, B, C, D, E, F, G, H, I, J, K, L]
intersection: [F, G]
difference: [A, B, C, D, E]
treeSet1 is subset of treeSet2: false
clear treeSet2: []
size: 7
to array: [A, B, C, D, E, F, G]
iterate: A B C D E F G
iterate through iterator: A B C D E F G
```

Explanation:
As the execution of the program starts i.e the main method is called,
we create a new Tree Set object and add some elements to it.
then we use different methods like: add, reomve, clear, etc to manipulate the
Tree Set object.

Enum Set

Advantages of enum set:

- Due to its implementation using RegularEnumSet and JumboEnumSet all the methods in an EnumSet are implemented using bitwise arithmetic operations.
- EnumSet is faster than HashSet because we no need to compute any hashCode to find the right bucket.
- The computations are executed in constant time and the space required is very little.

Code:

```java
import java.util.*;

enum Weekday {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
};

public class EnumSetAssign {
    public static void main(String[] args) {
        EnumSet<Weekday> weekdays = EnumSet.allOf(Weekday.class);

        System.out.println("weekdays enum set: " + weekdays);

        // add
        weekdays.add(Weekday.SATURDAY);
        System.out.println("add SATURDAY -> weekdays enum set: " + weekdays);

        // remove
        weekdays.remove(Weekday.SATURDAY);
        System.out.println("remove SATURDAY -> weekdays enum set: " + weekdays);

        // contains
        System.out.println("contains SATURDAY: " + weekdays.contains(Weekday.SATURDAY));

        // size
        System.out.println("size: " + weekdays.size());

        // to array
        Weekday[] array = weekdays.toArray(new Weekday[weekdays.size()]);

        System.out.println("to array: " + Arrays.toString(array));

        // iterate
        System.out.print("iterate: ");
        for (Weekday el : weekdays) {
            System.out.print(el + " ");
        }

        // iterate using iterator
        System.out.print("\nIterate list1 using iterator: ");
        Iterator<Weekday> itr = weekdays.iterator();
        while (itr.hasNext()) {
            System.out.print(itr.next() + " ");
        }
    }
}
```

Output:

```
weekdays enum set: [MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY]
add SATURDAY -> weekdays enum set: [MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY]
remove SATURDAY -> weekdays enum set: [MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SUNDAY]
contains SATURDAY: false
size: 6
to array: [MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SUNDAY]
iterate: MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SUNDAY
Iterate list1 using iterator: MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SUNDAY
```

Explanation:
As the execution of the program starts i.e the main method is called,
we create a new Enum Set object and add some elements to it.
then we use different methods like: add, reomve, clear, etc to manipulate the Enum Set object.

3. Develp a java program to implement queue interface by using ArrayDeque class and LinkedList class and priority queue class.[5M]

List of methods used in ArrayDeque, LinkedList & PriorityQueue :

add	addAll	addFirst	addLast
removeFirst	removeLast	getFirst	getLast
iterator	clear	peek	poll

Array Deque

ArrayDeque : a linear data structure which provides a way to apply resizable-array in addition to the implementation of the Deque interface. It is also known as Array Double Ended Queue or Array Deck. This is a special kind of array that grows and allows users to add or remove an element from both sides of the queue.

```
import java.util.ArrayDeque;
import java.util.Iterator;

public class ArrayDequeAssign {
    static ArrayDeque<String> createArrayDeque(String... elements) {
        ArrayDeque<String> newArrayDeque = new ArrayDeque<String>();
        for (String el : elements) {
            newArrayDeque.add(el);
        }
        return newArrayDeque;
    }

    public static void main(String[] args) {
        ArrayDeque<String> arrayDeque = createArrayDeque("A", "B", "C", "D", "E", "F", "G");
        System.out.println("arrayDeque: " + arrayDeque);

        // add
        arrayDeque.add("H");
        System.out.println("add H -> arrayDeque: " + arrayDeque);

        // add all
        ArrayDeque<String> addAll = new ArrayDeque<String>(arrayDeque);
        addAll.addAll(createArrayDeque("F", "G", "H", "I", "J", "K", "L"));
        System.out.println("added from F-L" + addAll);

        // add first
        arrayDeque.addFirst("M");
        System.out.println("add M -> arrayDeque: " + arrayDeque);

        // add last
        arrayDeque.addLast("N");
        System.out.println("add N -> arrayDeque: " + arrayDeque);

        // remove first
        arrayDeque.removeFirst();
        System.out.println("remove first -> arrayDeque: " + arrayDeque);

        // remove last
        arrayDeque.removeLast();
        System.out.println("remove last -> arrayDeque: " + arrayDeque);

        // get first
        System.out.println("get first -> arrayDeque: " + arrayDeque.getFirst());
```

```
        // get last
        System.out.println("get last -> arrayDeque: " + arrayDeque.getLast());

        // peek
        System.out.println("peek -> arrayDeque: " + arrayDeque.peek());

        // poll
        System.out.println("poll -> arrayDeque: " + arrayDeque.poll());

        // iterate through iterator
        System.out.print("iterate: ");
        Iterator<String> iterator = arrayDeque.iterator();
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }

        // clear
        arrayDeque.clear();
        System.out.println("\nclear arrayDeque: " + arrayDeque);

    }
}
```

Output:

```
arrayDeque: [A, B, C, D, E, F, G]
add H -> arrayDeque: [A, B, C, D, E, F, G, H]
added from F-L[A, B, C, D, E, F, G, H, F, G, H, I, J, K, L]
add M -> arrayDeque: [M, A, B, C, D, E, F, G, H]
add N -> arrayDeque: [M, A, B, C, D, E, F, G, H, N]
remove first -> arrayDeque: [A, B, C, D, E, F, G, H, N]
remove last -> arrayDeque: [A, B, C, D, E, F, G, H]
get first -> arrayDeque: A
get last -> arrayDeque: H
iterate: A B C D E F G H
clear arrayDeque: []
```

Explanation:
As the execution of the program starts i.e the main method is called,
we create a new Array Deque object and add some elements to it.
then we use different methods like: add, reomve, clear, etc to manipulate the Array Deque object.

Linked List

Linked List : The explination is given above ↑

```
import java.util.Iterator;
import java.util.LinkedList;

public class LinkedListAsQueueAssign {
        // helper method to create a linked list
    static LinkedList<String> createLinkedList(String... elements) {
        LinkedList<String> newLinkedList = new LinkedList<String>();
        for (String el : elements) {
            newLinkedList.add(el);
        }
        return newLinkedList;
    }
    // main method
    public static void main(String[] args) {
        LinkedList<String> linkedList = createLinkedList("A", "B", "C", "D", "E", "F", "G");
        System.out.println("Linked List: " + linkedList);

        // add
        linkedList.add("H");
        System.out.println("add H -> Linked List: " + linkedList);
```

```java
        // add all
        LinkedList<String> addAll = new LinkedList<String>(linkedList);
        addAll.addAll(createLinkedList("F", "G", "H", "I", "J", "K", "L"));
        System.out.println("added from F-L" + addAll);

        // add first
        linkedList.addFirst("M");
        System.out.println("add M -> Linked List: " + linkedList);

        // add last
        linkedList.addLast("N");
        System.out.println("add N -> Linked List: " + linkedList);

        // remove first
        linkedList.removeFirst();
        System.out.println("remove first -> Linked List: " + linkedList);

        // remove last
        linkedList.removeLast();
        System.out.println("remove last -> Linked List: " + linkedList);

        // get first
        System.out.println("get first -> Linked List: " + linkedList.getFirst());

        // get last
        System.out.println("get last -> Linked List: " + linkedList.getLast());

        // peek
        System.out.println("peek -> Linked List: " + linkedList.peek());

        // poll
        System.out.println("poll -> Linked List: " + linkedList.poll());

        // iterate through iterator
        System.out.print("iterate: ");
        Iterator<String> iterator = linkedList.iterator();
        while (iterator.hasNext()) {
            System.out.print(iterator.next() + " ");
        }

        // clear
        linkedList.clear();
        System.out.println("\nclear LinkedList: " + linkedList);

    }
}
```

Output:

```
Linked List: [A, B, C, D, E, F, G]
add H -> Linked List: [A, B, C, D, E, F, G, H]
added from F-L[A, B, C, D, E, F, G, H, F, G, H, I, J, K, L]
add M -> Linked List: [M, A, B, C, D, E, F, G, H]
add N -> Linked List: [M, A, B, C, D, E, F, G, H, N]
remove first -> Linked List: [A, B, C, D, E, F, G, H, N]
remove last -> Linked List: [A, B, C, D, E, F, G, H]
get first -> Linked List: A
get last -> Linked List: H
peek -> Linked List: A
poll -> Linked List: A
iterate: B C D E F G H
clear LinkedList: []
```

Explanation:

As the execution of the program starts i.e the main method is called, we create a new Linked List object and add some elements to it.

then we use different methods like: add, reomve, clear, etc to manipulate the Linked List object.

Priority Queue

PriorityQueue : is used when the objects are supposed to be processed based on the priority. It is known that a Queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the PriorityQueue comes into play.

As the implementation of priority queue in java isn't done using a linear data structure we don't have the following methods:

	addFirst	addLast	removeFirst	removeLast	getFirst	getLast	

+ Used **Streams**

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class PriorityQueueAssign {
        // helper method to create a priority queue
    static PriorityQueue<Integer> createPriorityQueue(Integer... elements) {
        PriorityQueue<Integer> newPriorityQueue = new PriorityQueue<Integer>();
        for (Integer el : elements) {
            newPriorityQueue.add(el);
        }
        return newPriorityQueue;
    }

            // main method
    public static void main(String[] args) {
        PriorityQueue<Integer> priorityQueue = createPriorityQueue(2, 5, 1, 8, 3, 7, 6, 4);
        System.out.println("priorityQueue: " + priorityQueue);

        // add
        priorityQueue.add(11);
        priorityQueue.add(5);
        System.out.println("add 11 -> priorityQueue: " + priorityQueue);

        // add all
        PriorityQueue<Integer> addAll = new PriorityQueue<Integer>(priorityQueue);
        List<Integer> intList = IntStream.rangeClosed(1, 20)
                .boxed().collect(Collectors.toList());
        addAll.addAll(intList);

        System.out.println("added from 1-20 +List: " + addAll);

        // remove
        priorityQueue.remove(5);
        System.out.println("remove 5 -> priorityQueue: " + priorityQueue);

        // remove first
        Integer ele = priorityQueue.remove();
        System.out.println("remove first -> priorityQueue: " + priorityQueue + " Removed
Element: " + ele);

        // poll
        System.out.println("poll -> priorityQueue: " + priorityQueue.poll());

        // peek
        System.out.println("peek -> priorityQueue: " + priorityQueue.peek());
```

```java
            // iterate through iterator
            System.out.print("iterate: ");
            Iterator<Integer> iterator = priorityQueue.iterator();
            while (iterator.hasNext()) {
                System.out.print(iterator.next() + " ");
            }

            // iterate through stream
            System.out.print("\niterate through stream: ");
            priorityQueue.stream().forEach(System.out::print);
            System.out.println();

            // clear
            priorityQueue.clear();
            System.out.println("clear -> priorityQueue: " + priorityQueue);
        }
    }
```

Output:

```
priorityQueue: [1, 3, 2, 4, 5, 7, 6, 8]
add 11 -> priorityQueue: [1, 3, 2, 4, 5, 7, 6, 8, 11, 5]
added from 1-20 +List: [1, 1, 2, 4, 3, 2, 4, 6, 8, 5, 5, 7, 3, 6, 5, 8, 7, 11, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
remove 5 -> priorityQueue: [1, 3, 2, 4, 5, 7, 6, 8, 11]
remove first -> priorityQueue: [2, 3, 6, 4, 5, 7, 11, 8] Removed Element: 1
poll -> priorityQueue: 2
peek -> priorityQueue: 3
iterate: 3 4 6 8 5 7 11
iterate through stream: 34685711
clear -> priorityQueue: []
```

Explanation:
As the execution of the program starts i.e the main method is called,
we create a new Priority object and add some elements to it.
then we use different methods like: add, reomve, clear, etc to manipulate the Priority Queue object.