ICS1312 – JAVA PROGRAMMING LABORATORY

DATE : 2.8.2025

ASSIGNMENT : 3

TITLE : JAVA INTERFACE FOR ADT

ROLLL NO : 3122247001017

LEARNING OBJECTIVE:

- TO IMPLEMENT INTERFACE FOR THE GIVEN USE CASE
- TO CREATE STACK ADT USING INTERFACE IN JAVA
- TO CREATE SET ADT USING INTERFACE IN JAVA

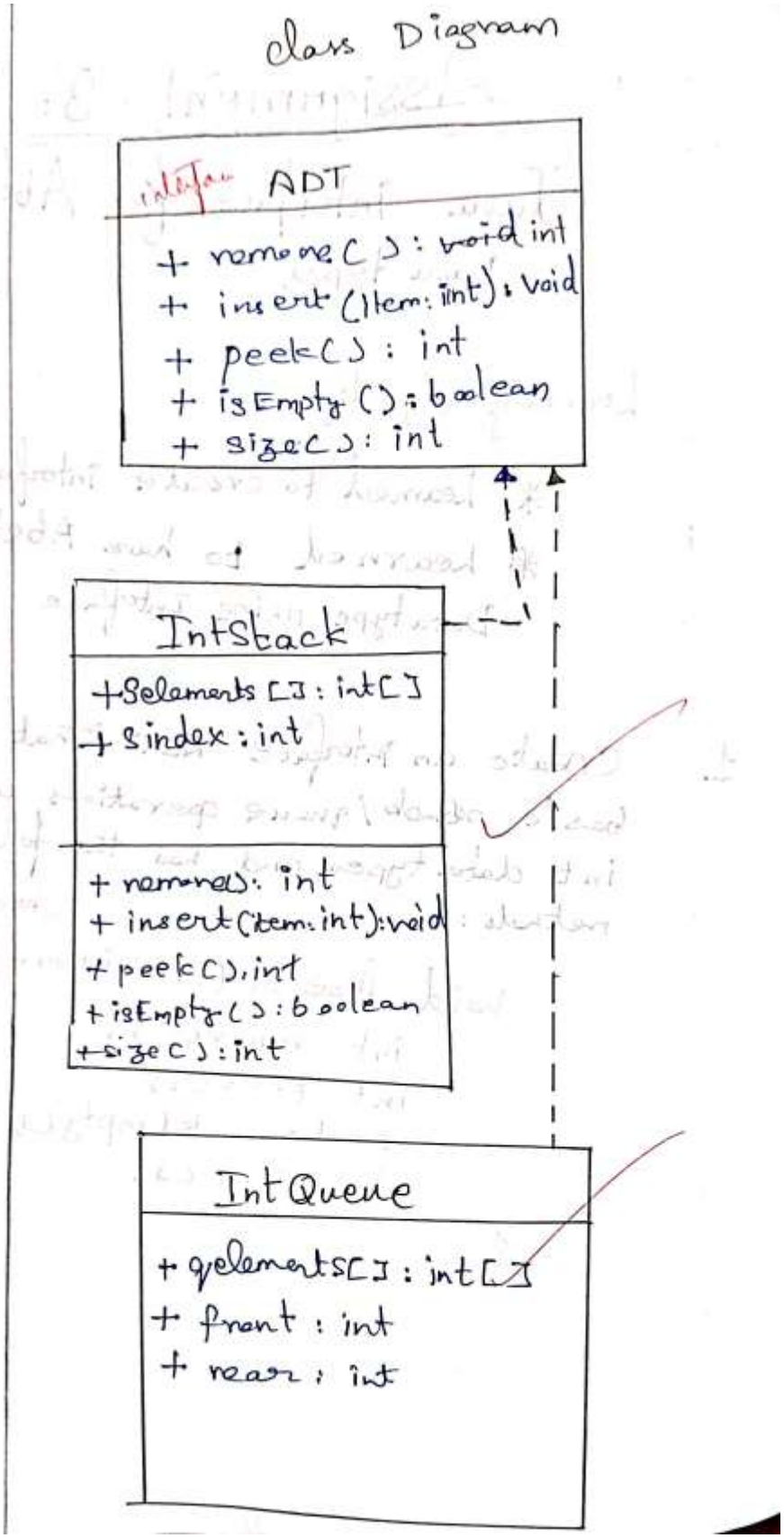
1. Create an interface ADT that defines basic stack/queue operations using `int` data type and has the following methods:

```
void insert(int item);   // Push or Enqueue
    int remove();          // Pop or Dequeue
    int peek();
    boolean isEmpty();
    int size();
}
```

   a. Create a class `IntStack` that implements the interface ADT and has variables to store:
      i. the stack elements
      ii. the index to the top element

   b. Create a class `IntQueue` that implements the interface ADT and has variables to store:
      i. the queue elements
      ii. the index to the front of the queue
      iii. the index to the rear of the queue

CLASS DIAGRAM



Class Signature :

```
interface ADT {
    void insert(int item);
    int remove();
    int peek();
    boolean isEmpty();
    int size();
}
class IntStack implements ADT {
    void insert
    int[] elements = new int[50];
    int sindex = 0;
    void insert(int item)
    {
        ---------
    }
    int remove()
    {
        ---------
    }
    int peek()
    {
        ---------
    }
```

```
boolean isEmpty{
    ---------
}
int size()
 {
    ------
   }
}
class IntQueue implements ADT {
 int qelemats [] = new qelemats[100];
 int front = 0;
 int rear = 0;
 void insert{
      ---------
    }
  int remove() {
      ---------
   }
  int peek(){
      ----
     }
   boolean isEmpty(){
      ---------
     }
   int size()
    {---------
```

}

Test cases:

```
class Main {
public static void main (String[] args)
 {
      IntStack a = new IntStack();
      a.insert(5);
      a.remove();
      a.peek();
 }    a.insert(6);
      a.insert(7);
      a.peek();
      a.isEmpty();
      a.size

      qelements

Int Queue
      IntQueue b = new IntQueue();
      b.insert(5);
      b.remove();
      b.insert(6);
      b.insert(7);
      b.isEmpty();
      b.size()
}
```

CODE :

```
interface ADT {
    void insert(int item);
    int remove();
    int peek();
    boolean isEmpty();
    int size();
}


class IntStack implements ADT {
    private int[] stack;
    private int top;

    public IntStack(int capacity) {
        stack = new int[capacity];
        top = -1;
    }

    public void insert(int item) {
        if (top < stack.length - 1) {
            stack[++top] = item;
        } else {
            System.out.println("Stack Overflow");
        }
    }

    public int remove() {
        if (!isEmpty()) {
            return stack[top--];
        }
        System.out.println("Stack Underflow");
        return -1;
    }

    public int peek() {
        if (!isEmpty()) {
            return stack[top];
        }
        return -1;
    }

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

    public int size() {
        return top + 1;
```

```
    }
}


class IntQueue implements ADT {
    private int[] queue;
    private int front, rear, count;

    public IntQueue(int capacity) {
        queue = new int[capacity];
        front = 0;
        rear = -1;
        count = 0;
    }

    public void insert(int item) {
        if (count < queue.length) {
            rear = (rear + 1) % queue.length;
            queue[rear] = item;
            count++;
        } else {
            System.out.println("Queue Overflow");
        }
    }

    public int remove() {
        if (!isEmpty()) {
            int item = queue[front];
            front = (front + 1) % queue.length;
            count--;
            return item;
        }
        System.out.println("Queue Underflow");
        return -1;
    }

    public int peek() {
        if (!isEmpty()) {
            return queue[front];
        }
        return -1;
    }

    public boolean isEmpty() {
        return count == 0;
    }

    public int size() {
        return count;
```

```
        }
    }

    public class Main {


    public static void main(String[] args) {
        System.out.println("Stack Example:");
        ADT stack = new IntStack(5);
        stack.insert(10);
        stack.insert(20);
        System.out.println("Top of stack: " + stack.peek());
        System.out.println("Removed from stack: " + stack.remove());
        System.out.println("Stack size: " + stack.size());

        System.out.println("\nQueue Example:");
        ADT queue = new IntQueue(5);
        queue.insert(100);
        queue.insert(200);
        System.out.println("Front of queue: " + queue.peek());
        System.out.println("Removed from queue: " + queue.remove());
        System.out.println("Queue size: " + queue.size());
    }
}
```

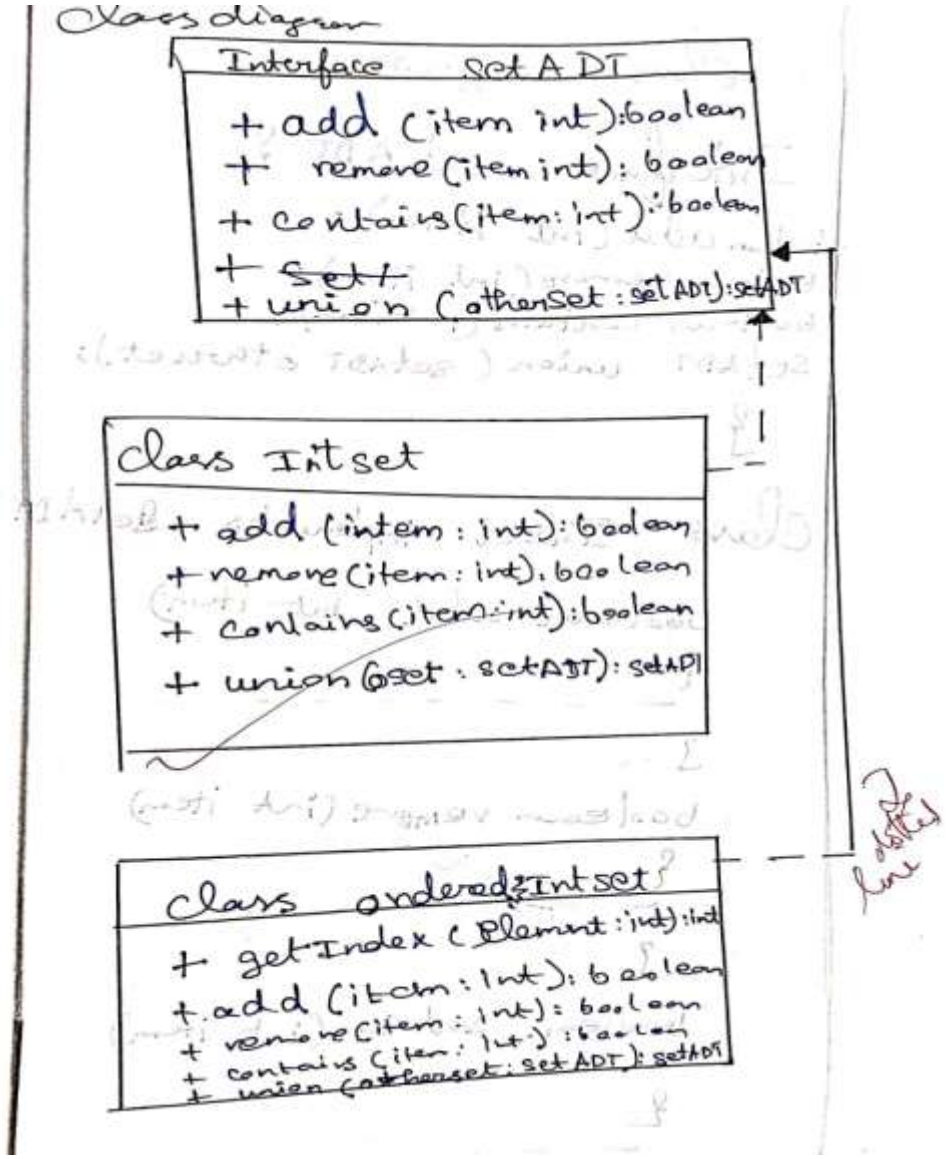
OUTPUT :

```
Stack Example:
Top of stack: 20
Removed from stack: 20
Stack size: 1

Queue Example:
Front of queue: 100
Removed from queue: 100
Queue size: 1
PS D:\Java\3>



Front of queue: 100
Removed from queue: 100
Queue size: 1
```

2. A set is a collection of unique elements. Create an interface `SetADT` that defines basic set operations using `int` data type and has the following methods:

```
boolean add(int item);
boolean remove(int item);
boolean contains(int item);
SetADT union(SetADT otherSet);
```

   a. Create a class `IntSet` that implements .`SetADT` and defines all its methods.
   b. An ordered set is a set that maintains the elements in ascending order. Create a class `OrderedIntSet` that defines an ordered set by implementing `SetADT`.



Class Signature

```
Interface setADT {
boolean add (int item);
boolean remove (int item);
boolean contain (int item);
setADT union (setADT otherset);
}

Class Intset implements setADT {
    boolean add (int item)
    {
    - - - - - - - -
    }
    boolean remove (int item)
    {
    - - - - - - - -
    }
    boolean contain (int item)
    {
    - - - - - - - -
    }
```

```
~~boo~~
setADT union (setADT otherset)
 {
   - - - - - - - .
  }
class OrdLinkedIntSet implements SetADT
 {
  int ~~check~~ getIndex(Int element)
   {
    - - - - - - .
   }
  boolean add (int item)
   {
   }
  boolean remove (int item)
   {
   }
  boolean contains (int item
   {
   }
  SetADT union(setADT otherset)
   {
     - - - - - - -
    }

 }
```

```
class main {
public static void main(String[] args)
{   Inset result = new Inset();
    Inset a2 = new Inset();
    Inset a1 = new Inset()
    a1.add(1) ;
    a1.remove();
    a1.add(5);
    a1.contains()
    a2.add(2); ;
result = a1.union(a2) ; system.out.println(result
result= a1.setADTUnion(a2) ;
    OrderedInset result = new orderIn
    OrderedInset b1 = new orderIn
    orderedInset b2 = new orderInt
    b1.add(1) ;
    b2.add(5) ;
    b1.remove();
    b1.add(2)
    b1.contains();
    b2.add(6)
result2=b1.union()
result2 =a1.setADT union();
}
}
```

Test cases:

1 added

5 added

5

2 added

CODE :

```java
interface SetADT {
    boolean add(int item);
    boolean remove(int item);
    boolean contains(int item);
    SetADT union(SetADT otherSet);
}


class IntSet implements SetADT {
    private int[] elements;
    private int size;

    public IntSet() {
        elements = new int[100];
        size = 0;
    }


    public boolean add(int item) {
        if (contains(item)) return false;
        elements[size++] = item;
        return true;
    }


    public boolean remove(int item) {
        for (int i = 0; i < size; i++) {
            if (elements[i] == item) {

                for (int j = i; j < size - 1; j++) {
                    elements[j] = elements[j + 1];
                }
                size--;
                return true;
            }
        }
        return false;
    }

    public boolean contains(int item) {
        for (int i = 0; i < size; i++) {
            if (elements[i] == item) return true;
        }
        return false;
    }
```

```
    public SetADT union(SetADT otherSet) {
        IntSet result = new IntSet();

        for (int i = 0; i < this.size; i++) {
            result.add(this.elements[i]);
        }

        if (otherSet instanceof IntSet) {
            IntSet other = (IntSet) otherSet;
            for (int i = 0; i < other.size; i++) {
                result.add(other.elements[i]);
            }
        }
        return result;
    }


    public void display() {
        System.out.print("Set: ");
        for (int i = 0; i < size; i++) {
            System.out.print(elements[i] + " ");
        }
        System.out.println();
    }
}
public class Main2 {

public static void main(String[] args) {
    IntSet set1 = new IntSet();
    set1.add(5);
    set1.add(3);
    set1.add(9);
    set1.add(5);
    System.out.println("Set 1:");
    set1.display();

    IntSet set2 = new IntSet();
    set2.add(3);
    set2.add(7);
    set2.add(1);
    System.out.println("Set 2:");
    set2.display();

    SetADT unionSet = set1.union(set2);
    System.out.println("Union of Set 1 and Set 2:");
    ((IntSet) unionSet).display();
}
```

```
}
```

OUTPUT:

```
PS D:\Java\3> javac Main2.java
PS D:\Java\3> java Main2
Set 1:
Set: 5 3 9
Set 2:
Set: 3 7 1
Union of Set 1 and Set 2:
Set: 5 3 9 7 1
```

LEARNING OUTCOMES :

- **LEARNED TO IMPLEMENT INTERFACE**
- **LEARNED TO IMPLAMENT STACK IN JAVA**
- **LEARNED TO IMPLEMENT SET IN JAVA**