

Lab 9a:

```
package Unit9;

public class MathDemo {
    public static void main ( String[] args ) {
        System.out.println("Exploring Java packages list");
        int a = 2, b = 3; // Using methods defined inside Math class of Java
        int sum = Math.addExact(a, b);
        int multiply = Math.multiplyExact(a, b);
        System.out.println("Sum is: " + sum );
        System.out.println("Multiplication is: " + multiply);

        // Using methods defined inside Integer Wrapper class of Java
        int c = 2; // Use to convert an integer to string
        System.out.println("Using toString(c): " + c);

        // Use to convert an integer to Binary String
        System.out.println("Using toBinaryString(c): " + Integer.toBinaryString(c));

        System.out.println(Math.max(5, 19));
        System.out.println(Math.pow(2, 4)); //16
    }
}
```

Lab 9b:

What is the Autoboxing in Java?

Automatic conversion of primitive data type to the object data type with corresponding Java wrapper class is called as Autoboxing.

For example convert primitive data type int to Java wrapper class Integer.

Java Autoboxing example

```
/* Java Autoboxing example convert primitive data type int
to Java wrapper class Integer
Save with file name AutoboxingExample.java */
import java.util.ArrayList;
public class AutoboxingExample
{
    public static void main(String args[])
    {
        int intval = 100;
        // AUTOBOXING
        // AUTOMATIC CONVERSION FROM PRIMITIVE DATA TYPE TO WRAPPER CLASS OBJECT
        Integer intobj = intval;
        System.out.println("Autoboxing Value : "+intobj);
        ArrayList<Integer> arrayList = new ArrayList<Integer>();
        // AUTOBOXING BECAUSE ARRAYLIST STORES OBJECTS ONLY
    }
}
```

```

        arrayList.add(101);
        arrayList.add(intobj);
        System.out.println(arrayList.get(0));
        System.out.println(arrayList.get(1));
    }
}

```

### Lab 9c:

What is the AutoUnboxing in Java?

Automatic conversion of Java wrapper class object to the primitive data type is called as Unboxing.

For example convert Java wrapper class Integer to primitive data type int.

### Java Unboxing example

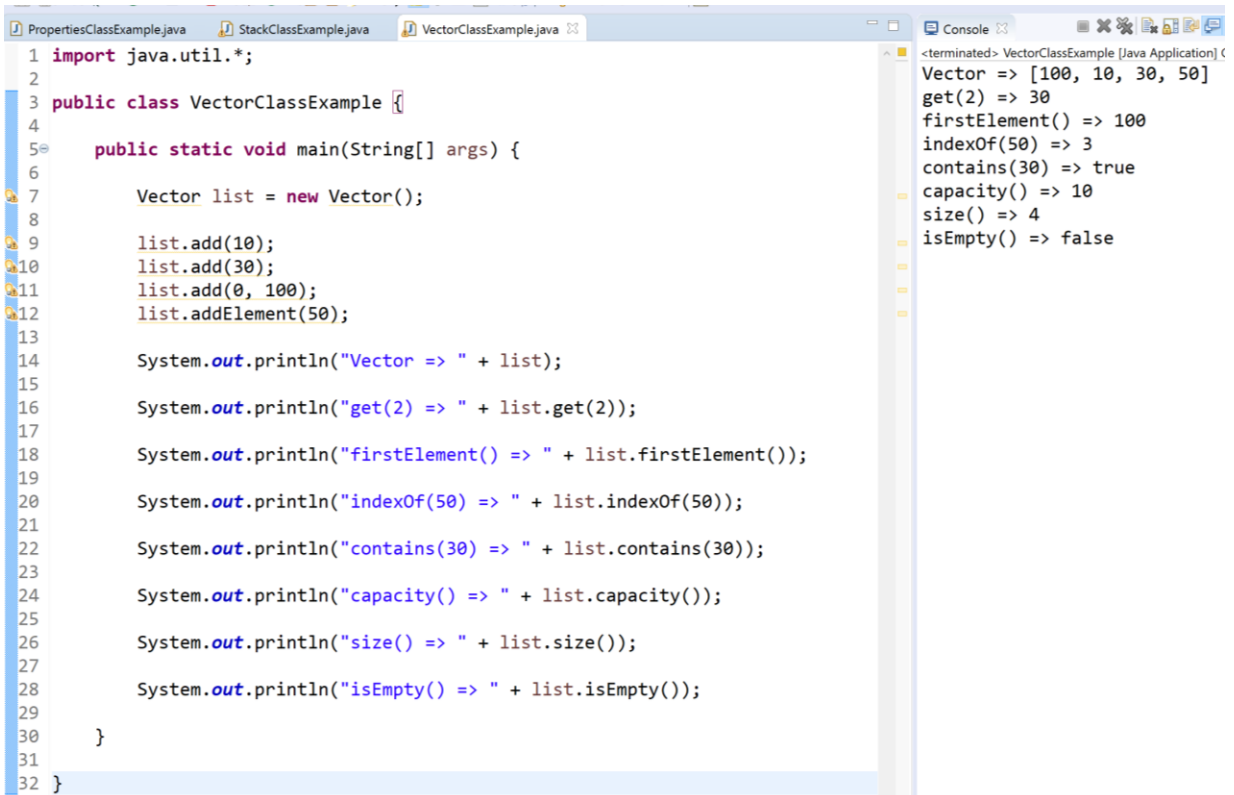
```

/* Java Unboxing example convert Java wrapper class Integer
to primitive data type int
Save with file name UnboxingExample.java */

public class UnboxingExample
{
    public static void main(String args[])
    {
        Integer intobj = new Integer(100);
        // UNBOXING
        // AUTOMATIC CONVERSION FROM WRAPPER CLASS OBJECT TO PRIMITIVE DATA TYPE
        int intval = intobj;
        System.out.println("Unboxing Value : "+intval);
    }
}

```

### Lab 9d: Vector



```
1 import java.util.*;
2
3 public class VectorClassExample {
4
5     public static void main(String[] args) {
6
7         Vector list = new Vector();
8
9         list.add(10);
10        list.add(30);
11        list.add(0, 100);
12        list.addElement(50);
13
14        System.out.println("Vector => " + list);
15
16        System.out.println("get(2) => " + list.get(2));
17
18        System.out.println("firstElement() => " + list.firstElement());
19
20        System.out.println("indexOf(50) => " + list.indexOf(50));
21
22        System.out.println("contains(30) => " + list.contains(30));
23
24        System.out.println("capacity() => " + list.capacity());
25
26        System.out.println("size() => " + list.size());
27
28        System.out.println("isEmpty() => " + list.isEmpty());
29    }
30 }
31
32 }
```

```
<terminated> VectorClassExample [Java Application] (
Vector => [100, 10, 30, 50]
get(2) => 30
firstElement() => 100
indexOf(50) => 3
contains(30) => true
capacity() => 10
size() => 4
isEmpty() => false
```

## Lab 9e: Stack

```
// StackDemo.java

import java.util.Stack;
class StackDemo {
    public static void main(String a[]){
        Stack<Integer> stack = new Stack<>();
        System.out.println("Empty stack : " + stack);
        System.out.println("Empty stack : " + stack.isEmpty());

        stack.push(100);
        stack.push(102);
        stack.push(103);
        stack.push(104);
        System.out.println("Stack elements are : " + stack);
        System.out.println("Stack: Pop: " + stack.pop());
        System.out.println("Stack: After Pop: " + stack);
        System.out.println("Stack : search(): " + stack.search(1002));
        System.out.println("If Stack is empty : " + stack.isEmpty());
    }
}
```

See the following output:

```

→ java javac StackDemo.java
→ java java StackDemo
Empty stack : []
Empty stack : true
Stack elements are : [100, 102, 103, 104]
Stack: Pop Operation : 104
Stack: After Pop Operation : [100, 102, 103]
Stack : search() Operation : -1
If Stack is empty : false
→ java []

```

The screenshot shows an IDE with two tabs: 'PropertiesClassExample.java' and 'StackClassExample.java'. The 'StackClassExample.java' tab is active, displaying the following code:

```

1 import java.util.*;
2
3 public class StackClassExample {
4
5     public static void main(String[] args) {
6
7         Stack stack = new Stack();
8
9         Random num = new Random();
10
11         for(int i = 0; i < 5; i++)
12             stack.push(num.nextInt(100));
13
14         System.out.println("Stack elements => " + stack);
15
16         System.out.println("Top element is " + stack.peek());
17
18         System.out.println("Removed element is " + stack.pop());
19
20         System.out.println("Element 50 availability => " + stack.search(50));
21
22         System.out.println("Stack is empty? - " + stack.isEmpty());
23
24     }
25
26 }
27

```

The console window on the right shows the output of the program:

```

<terminated> StackClassExample [Java Application] C:\Program Files\Java
Stack elements => [35, 54, 77, 22, 64]
Top element is 64
Removed element is 64
Element 50 availability => -1
Stack is empty? - false

```