Experiment No. 1

Aim: Programs to accept command line arguments.

Description:

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

Output:

```
D:\SE3_25_java>javac HelloWorld.java

D:\SE3_25_java>java HelloWorld Swayam Dilesh Raut
Swayam dilesh raut

D:\SE3_25_java>
```

Conclusion: we learn to implement an program that can accept command line arguments

code :-

```
class HelloWorld
{
  public static void main (String args[])
  {
    System.out.println ("No of arguments :" + args.length);

    for(int i=0; i<args.length; i++)
    {
      System.out.println ("Argument" +i + ":" + args[i]);
    }
  }
}
```

Experiment No. 2

Aim: Program on accepting input through keyboard as well as users.

Description:

- `import java.util.Scanner;` - imports the class `Scanner` from the library `java.util`
- `Scanner scanner = new Scanner(System.in);` - creates a new `Scanner` object, that is connected to standard input (the keyboard)
- `String inputString = scanner.nextLine();`
  1. temporarily suspends the execution of the program, waiting for input from the keyboard;
  2. the input is accepted as soon as the user digits a carriage return;
  3. (a reference to) the string entered by the user (up to the first newline) is returned by the `nextLine()` method;
  4. (the reference to) the string is assigned to the variable `inputString`.

Output:

```
D:\SE3_25_java>javac Hello.java

D:\SE3_25_java>java Hello
Enter the salary:
65000
Enter the name:
Swayam
Enter the age:
19
Name is: Swayam
Age is: 19
Salary is: 65000
```

Conclusion:

We got to learn about Scanner class an implement the program on accepting input through user

```
import java.util.Scanner;
class HelloWorld
{
public static void main (String args[])
 {
  Scanner sc = new Scanner(System.in);

  System.out.println("Enter the salary:");
  int salary = sc.nextInt();

  sc.nextLine();

  System.out.println("Enter the Name:");
  String name = sc.NextLine();

  System.out.println("enter age :");
  int age = sc.nextInt();

  System.out.println("Name is:" +name);
  System.out.println("Age is:" +age);
  System.out.println("Salary is:" +salary);

  sc.close();
 }
}
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

class InputDemo2{
    Public static void main(String[]args){
    int num;
    BufferedReader reader = new BufferedReader(System.in);
     try{
       System.out.println("Please enter the number:");
       num = Integer.parseINt(reader.readline());
       System.out.println("The number entered is:"+num);
        } catch(IOException e){
           System.out.println("IO Error."+ e.getMessage());
           } finally {
              System.out.println("End of program");
           }
      }
    }
```

```
D:\SE3_25_java>java InputDemo2
Please enter the number:
25
The number entered is: 25
End of program

D:\SE3_25_java>
```

Experiment No. 3

Aim: Programs on class and objects. Program to display the class name at dynamic run time

Description:

```
String cmp = "Emp" // this is the classname
Object obj = class.forName().newInstance();


public static MyClass getFromString(String myClassName) {
    Object result = null;
    MyClass final_result = null;

    try{
        result = Class.forName(myClassName).getConstructor().newInstance();
        final_result= (MyClass)result;
    }catch(ClassNotFoundException e){} //you must specify concrete Exception here

    return final_result;
    }
```

Output:

```
D:\SE3_25_java>javac Test.java

D:\SE3_25_java>java Test
Name of the class running at runtime = Test

D:\SE3_25_java>
```

Conclusion: we learn about class and object as well as display class name at dynamic runtim

code

```
public class Test
{
  public String getClassname()
  {
   String classname = this.getclass().getSimpleName();
   System.out.println("Name of the class running at runtime
   = "+ className);
   return classname;
  }
  public static void main (String ~~args~~ swayam)
  {
    Test t = new Test();
    t.getClassname();
  }
}
```

Experiment No. 4

Aim: Program to open windows processes at run time.

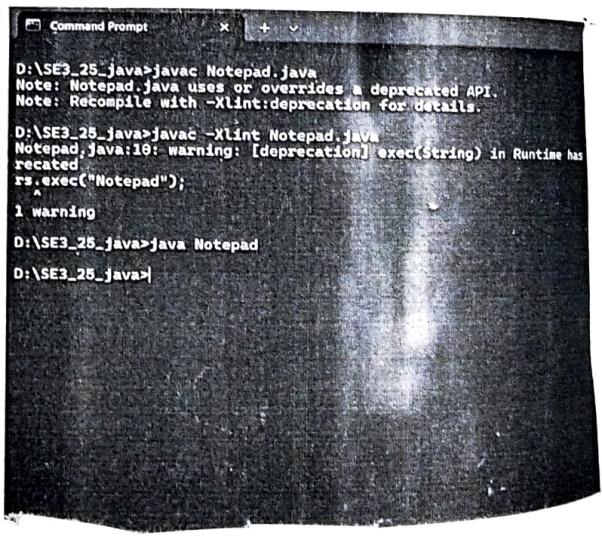
Description:

```
String command = "notepad.exe";
Runtime run = Runtime.getRuntime();
run.exec(command);
```

OR

```
Process p  = Runtime.getRuntime().exec("cmd /c START /MAX notepad.exe");
```

Output:



Conclusion:

we used java Library & windows 32 commands to execute command in & process windows at runtime

```
import java.util.*;
import java.io.*;

class Notepad
{
public static void main (String[] args)
 {
  Runtime rs = Runtime.getRuntime();

  try
  {
   rs.exec ("Notepad");
  }
  catch (IOException e)
  {
   System.out.println(e);
  }
 }

}
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:

```
D:\SE3_25_java>javac Demo2.java

D:\SE3_25_java>java Demo2
 The The string is = SE3 Students
 The sum of the Digits is =30
```

```
D:\SE3_25_java>javac Demo.java

D:\SE3_25_java>java Demo
 The  string is = SE3 Students
 The sum of the Number is =70
```

Conclusion: we implemented method & constructor overloading by printing the same function but with different parameters

```
class Demo2
{
 public void Demo()
 {
  String msg = "SE3 Students";
  System.out.println("The String is =" + msg);
 }
 public void Demo (int i, int j)
 {
  int k = i+j;
  System.out.Println("The sum of the Digit is:"+
 }
  public static void main (String args[])
  {
   Demo2 d1 = new Demo2();
   d1.Demo();
   d1.Demo(10,20);
  }
}
```

```
class Demo
{
 public Demo()
 {
  String msg = "SE3 Students";
  System.out.println("The string is = " + msg);
 }
 Public Demo (int i, int j)
 {
  int k = i+j
  System.out.println("The sum of the Number is = " + k);
 }
 public static void main(String args[])
 {
  Demo d1 = new Demo();
  Demo d2 = new Demo(10,20);
 }
}
```

```
package myPackage;
    Public class MyClass
    {
      public void getName (String [] args)
        {
          System.out.println (args);
        }
    }

import myPackage.MyClass;
    public class PrintName {
    Public Static Void main (String args[])
      {
        String name = " Hello";
        Myclass obj = new MyClass();
        obj.getName(name);
      }
}
```

pac

Experiment No. 6

Aim: Program on Packages.

Description:

There are two types of packages in java:

1. User-defined Package (Create Your Own Package's)
2. Built-in packages are packages from the java application programming interface that are the packages from Java API for example such as **swing, util, net, io, AWT, lang, javax**, etc.

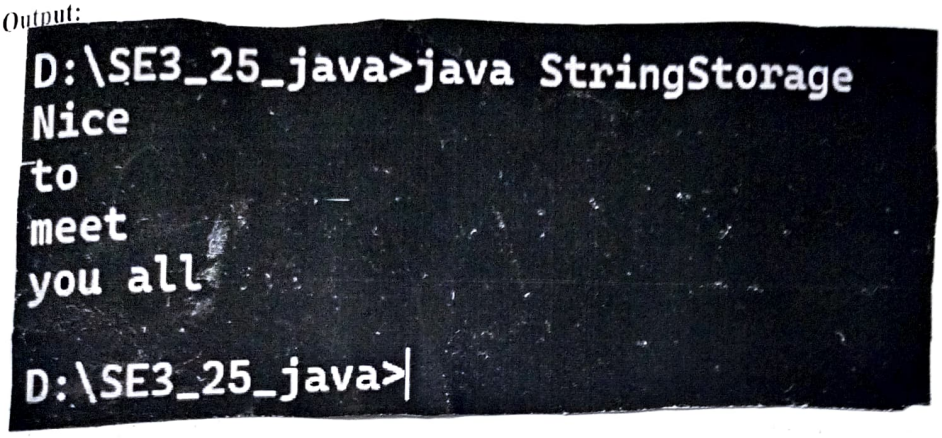
Output:

```
D:\SE3_25_java>java PrintName
Hello
inside my package

D:\SE3_25_java>
```

Conclusion: we learn to create our own package in java

```
package myPackage;
    Public class MyClass
    {
      public void getName (String [] args)
       {
         System.out.println(args);
       }
    }

import myPackage.MyClass;
    public class PrintName{
    Public Static void main (String args[])
      {
        String name = "Hello";
        Myclass obj = new MyClass();
        obj.getName(name);
      }
}
```

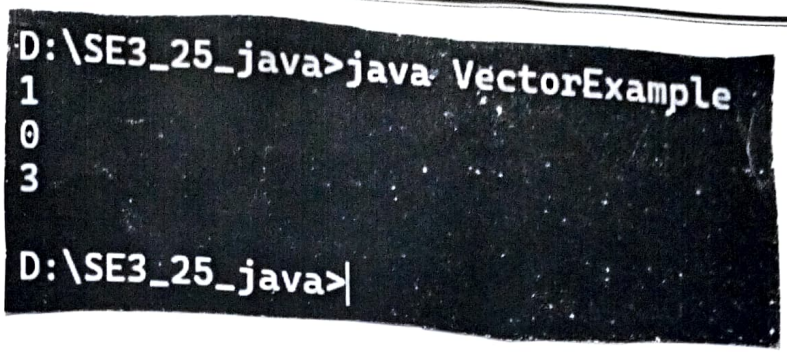
Output:



Conclusion:

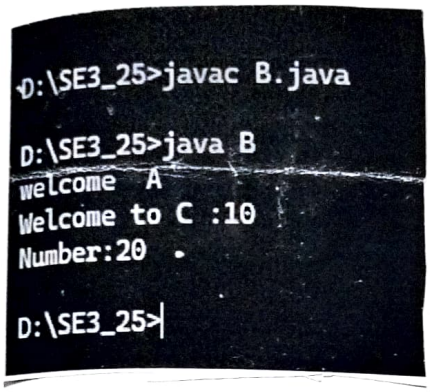
we Learn to implement various String Library and vectors with their Functions

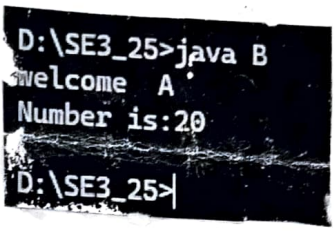
```
class StringStorage
{
    public static void main (String args[])
    {
        String S1 = "Nice";
        String S2 = new String("to");
        StringBuffer S3 = new StringBuffer("meet");
        StringBuilder S4 = new StringBuilder("you");
        S4 = S4.append("all");
        System.out.println(S1);
        System.out.println(S2);
        System.out.println(S3);
        System.out.println(S4);
    }
}
```




```
import java.util.Vector;

public class VectorExample {
    ~~class~~ ~~Aar~~
    Public static void main (String [] args)
    {
        Vector <Integer> V = new Vector <> (3, 2);
        V.addElement (1);
        V.addElement (2);
        V.addElement (3);
        V.insertElementAt (0, 1);
        V.removeElementAt (2)
        For (int i : V) {
            System.out.println (i);
        }
    }
}
```

Output:





Conclusion: we learn about two types of inheritance in java

```
class A
{
  protected int num = 20;
  void welcome()
  {
    System.out.println("welcome A");
  }
}
class B extends A
 {
    Public static void main (String args[])
     {
       A a1 = new A();
       a1.welcome();
       System.out.println("Number is:" + a1.num);
}   }
```

```
class A
{
  protected int num = 20;
  void welcome()
  {
    System.out.println("welcome A");
  }
class C extends A
  {
    void welcome(int a)
    {
      System.out.println("welcome to C:"+a);
    }
  }
class B extends C
  {
    public static void main(String args[])
    {
      B b1 = new B();
      b1.welcome();
      b1.welcome(10);
      System.out.println("Number:" +b1.num);
    }
  }
```

Experiment No. 9

m: Program on Multiple Inheritances.

escription:

Java doesn't support Multiple Inheritance.

Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with the same signature in both the superclasses and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

Java interfaces

A Java interface is a group of abstract methods that specify the behavior that implementing classes must follow. It serves as a class blueprint by outlining each class's methods. Interfaces offer a degree of abstraction for specifying behaviors but cannot be instantiated like classes. In Java, a class can successfully implement several interfaces to achieve multiple inheritance.

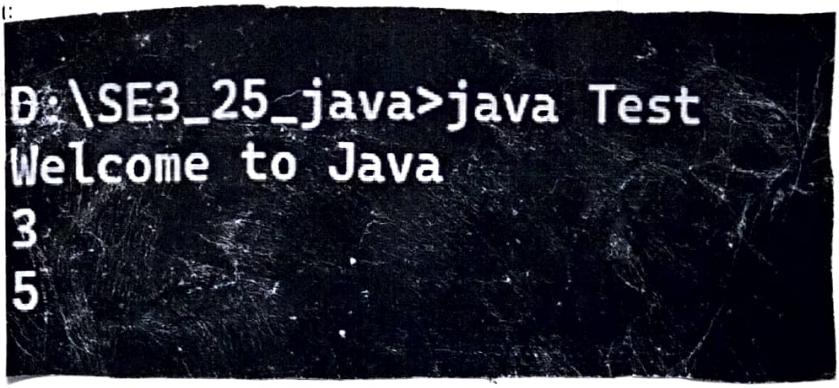
Output:

```
D:\SE3_25_java>java CI
Welcome A
Welcome B = 10
```

Conclusion: we learn to perform multiple inheritances using interface

```
interface B {
    void Welcome(int a);
}

class A {
    public void Welcome() {
        System.out.println("welcome A");
    }
}

class CI extends A implements B {
    @Override
    public void Welcome(int a) {
        System.out.println("welcome B = " + a);
    }
    public static void main(String[] args) {
        CI CI = new CI();
        CI.welcome();
        CI.welcome(10);
    }
}
```

Output:



Conclusion: we learn to handle system Exceptions in java

```
import java.lang.Throwable;
  Public class Test
  {
      public static void main (String [] args)
      {
        try
        {System.out.println (" welcome tojava ");
         int sum = a/0;
         System.out.println(" 2");
        }
       catch(ArithmeticException e)
        { System.out.println ("3");
        }
```

```
Catch (Exception e)
  {
      System.out.println("4");
  }
  Finally
  {
    System.out.println("5");
  }
}
```

Experiment No. 11

Aim: User defined exception handling implementation.

Description:

To write a Java program to implement user defined exception handling.

ALGORITHM:

1. Start
2. Create a class NegativeAmtException which extends Exception class.
3. Create a constructor which receives the string as argument.
4. Get the Amount as input from the user.
5. If the amount is negative, the exception will be generated.
6. Using the exception handling mechanism , the thrown exception is handled by the catch construct.
7. After the exception is handled , the string "invalid amount " will be displayed.
8. If the amount is greater than 0, the message "Amount Deposited " will be displayed
9. Stop

Output:

```
D:\SE3_25_java>javac ExceptionHandling.java

D:\SE3_25_java>java ExceptionHandling
Enter Your Age
19

D:\SE3_25_java>java ExceptionHandling
Enter Your Age
-19
Age can not be negative

D:\SE3_25_java>
```

Conclusion: we learn to handle user defined in java