INFORMATION TECHNOLOGY DEPARTMENT

JAVA PROGRAMMING LAB MANUAL

TABLE OF CONTENTS

Introduction TO JAVA Programming Laboratory

JAVA: Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later being renamed as Java, from a list of random words.

Laboratory Objective

Upon successful completion of this Lab the student will be able to:

1. Understand the concept of OOP as well as the purpose and usage principles of inheritance, polymorphism, encapsulation and method overloading.
2. Understand fundamentals of programming such as variables, conditional and iterative execution, methods, etc.
3. Identify classes, objects, members of a class and the relationships among them needed for a specific problem.
4. Understand fundamentals of object-oriented programming in Java, including defining classes, invoking methods, using class libraries, etc.
5. Create Java application programs using sound OOP practices (e.g., interfaces and APIs) and proper program structuring (e.g., by using access control identifies, automatic documentation through comments, error exception handling)
6. Have the ability to write a computer program to solve specified problems.
7. Develop programs using the Java Collection API as well as the Java standard class library.
8. Use the Java SDK environment to create, debug and run simple Java programs

Overview of Java

Java Is Important to the Internet, The Internet helped catapult Java to the forefront of programming, and Java, in turn, has had a profound effect on the Internet. The reason for this is quite simple: Java expands the universe of objects that can move about freely in cyberspace. In a network, two very broad categories of objects are transmitted between the server and our personal computer: passive information and dynamic, active programs.

Java can be used to create two types of programs: applications and applets. An application is a program that runs on your computer, under the operating system of that computer. An applet is an application designed to be transmitted over the Internet and executed by a Java-compatible Web browser.

Features of JAVA

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

JDK

The Java Development Kit (JDK) is an implementation of either one of the Java SE, Java EE or Java ME platforms. The JDK includes a private JVM and a few other resources to finish the development of a Java Application.

The JDK has as its primary components a collection of programming tools, including:

- appletviewer – this tool can be used to run and debug Java applets without a web browser
- apt – the annotation-processing tool.
- extcheck – a utility that detects JAR file conflicts
- idlj – the IDL-to-Java compiler. This utility generates Java bindings from a given Java IDL file.
- jabswitch – the Java Access Bridge. Exposes assistive technologies on Microsoft Windows systems.
- java – the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The old deployment launcher, jre, no longer comes with Sun JDK, and instead it has been replaced by this new java loader.
- javac – the Java compiler, which converts source code into Java bytecode
- javadoc – the documentation generator, which automatically generates documentation from source code comments
- jar – the archiver, which packages related class libraries into a single JAR file. This tool also helps manage JAR files.
- javafxpackager – tool to package and sign JavaFX applications
- jarsigner – the jar signing and verification tool
- javah – the C header and stub generator, used to write native methods
- javap – the class file disassembler
- javaws – the Java Web Start launcher for JNLP applications
- JConsole – Java Monitoring and Management Console
- jdb – the debugger
- jhat – Java Heap Analysis Tool (experimental)

- jinfo – This utility gets configuration information from a running Java process or crash dump. (experimental)
- jmap – This utility outputs the memory map for Java and can print shared object memory maps or heap memory details of a given process or core dump. (experimental)
- jmc – Java Mission Control
- jps – Java Virtual Machine Process Status Tool lists the instrumented HotSpot Java Virtual Machines (JVMs) on the target system. (experimental)
- jrunscript – Java command-line script shell.
- jstack – utility that prints Java stack traces of Java threads (experimental)
- jstat – Java Virtual Machine statistics monitoring tool (experimental)
- jstatd – jstat daemon (experimental)
- keytool – tool for manipulating the keystore
- pack200 – JAR compression tool
- policytool – the policy creation and management tool, which can determine policy for a Java runtime, specifying which permissions are available for code from various sources
- VisualVM – visual tool integrating several command-line JDK tools and lightweight performance and memory profiling capabilities
- wsimport – generates portable JAX-WS artifacts for invoking a web service.
- xjc – Part of the Java API for XML Binding (JAXB) API. It accepts an XML schema and generates Java classes.

1. Data Types used in JDK

Each row contains the data type and size and range of the data type. The list of available data types in Java is shown in table below

Name	Width	Range
byte	8	–128 to 127
short	16	–32,768 to 32,767
int	32	–2,147,483,648 to 2,147,483,647
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	32	1.4e−045 to 3.4e+038
double	64	4.9e–324 to 1.8e+308
Char	2	0 to 65,536
Boolean	1	True or false

2. **Security**

As we are likely aware, every time that we download a –normal‖ program, we are risking a viral infection. Prior to Java, most users did not download executable programs frequently, and those who did scan them for viruses prior to execution. Even so, most users still worried about the possibility of infecting their systems with a virus. In addition to viruses, another type of malicious program exists that must be guarded against. This type of program can gather private information, such as credit card numbers, bank account balances, and passwords, by searching the contents of your computer‘s local file system. Java answers both of these concerns by providing a –firewall‖ between a networked application and our computer.
When we use a Java-compatible Web browser, we can safely download Java applets without fear of viral infection or malicious intent. Java achieves this protection by confining a Java program to the Java execution environment and not allowing it access to other parts of the computer.
The ability to download applets with confidence that no harm will be done and that no security will be breached is considered by many to be the single most important aspect of Java.

The key that allows Java to solve both the security and the portability problems just described is that the output of a Java compiler is not executable code. Rather, it is bytecode. Bytecode is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreter for bytecode. This may come as a bit of a surprise. Translating a Java program into bytecode helps makes it much easier to run a program in a wide variety of environments. The reason is straightforward: only the
JVM needs to be implemented for each platform. Once the run-time package exists for a given system, any Java program can run on it. Because the execution of every Java program is under the control of the JVM, the JVM can contain the program and prevent it from generating side effects outside of the system. When a program is interpreted, it generally runs substantially slower than it would run if compiled to executable code.

3. **System Requirement**

Hardware Configuration

System Configuration: Windows XP, Intel Xeon 3.0 GHz CPU, Cache, 1GB DDR RAM

Software Configuration

JDK Software Version: - JDK1.7 Version: 7.1.7.0 40

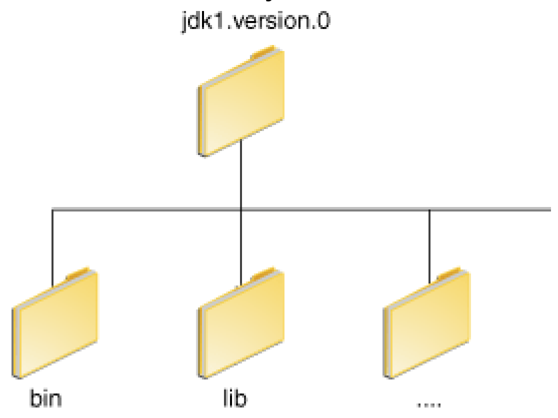
Browser: Internet Explorer / Google Chrome

4. PATH and CLASSPATH

Setting PATH and CLASSPATH environment variables on Microsoft Windows, Solaris, and Linux are as follows.

Install the Java Development Kit (JDK) software.

After installing the software, the JDK directory will have the structure shown below.



The bin directory contains both the compiler and the launcher.

Update the PATH Environment Variable (Microsoft Windows)

We can run Java applications just fine without setting the PATH environment variable. Or, we can optionally set it as a convenience.

Set the PATH environment variable if we want to be able to conveniently run the executables (javac.exe, java.exe, javadoc.exe, and so on) from any directory without having to type the full path of the command. If we do not set the PATH variable, we need to specify the full path to the executable every time we run it, such as:

C:\Java\jdk1.7.0\bin\javac MyClass.java

The PATH environment variable is a series of directories separated by semicolons (;). Microsoft Windows looks for programs in the PATH directories in order, from left to right. We should have only one bin directory for the JDK in the path at a time (those following the first are ignored), so if one is already present, we can update that particular entry.

The following is an example of a PATH environment variable: C:\Java\jdk1.7.0\bin;C:\Windows\System32\;C:\Windows\;C:\Windows\System32\Wbem

It is useful to set the PATH environment variable permanently so it will persist after rebooting. To make a permanent change to the PATH variable, use the **System** icon in the Control Panel. The precise procedure varies depending on the version of Windows:

Windows XP

1. Select **Start**, select **Control Panel**. double click **System**, and select the **Advanced** tab.
2. Click **Environment Variables**. In the section **System Variables**, find the PATH environment variable and select it. Click **Edit**. If the PATH environment variable does not exist, click New.
3. In the **Edit System Variable** (or **New System Variable**) window, specify the value of the PATH environment variable. Click **OK**. Close all remaining windows by clicking **OK**.

Program 1
Java program to display Welcome message.

Problem Definition

Syntax of the java program to display a message.

Problem Description

This program has given for understanding of the basic Java program and the execution procedure. The file name should be given as the name of the class which is having main method.

Pseudocode

```
/* This is a simple Java program. Call this file "Example.java". */
class name_of_the_class {
// Your program begins with a call to main().
public static void main(String args[]) {
// display the ouput using the following command
System.out.println("output to be displayed.");
}
}
```

Problem Validation

Compile the program using the following command

Javac Example.java

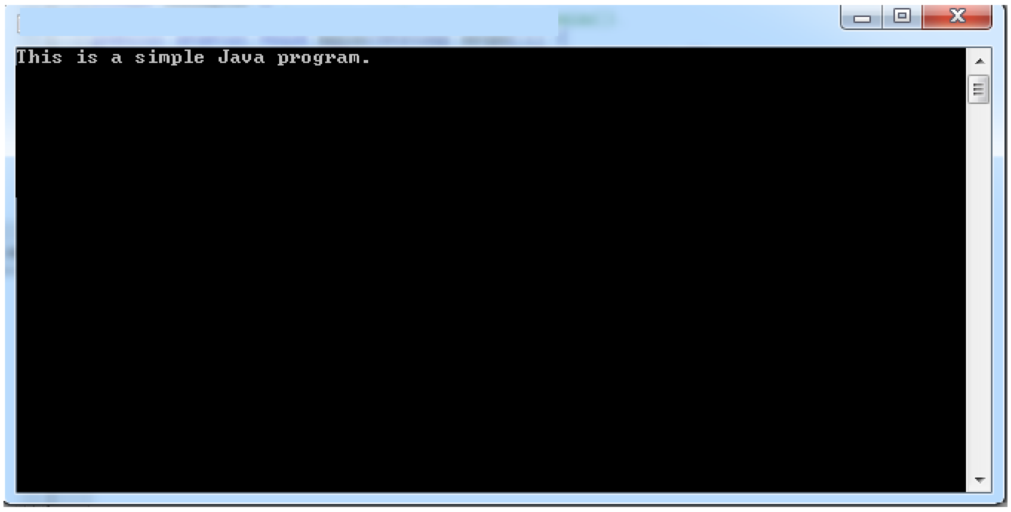
Execute the program using

Java Example

Input:

No Input

Output:



```
This is a simple Java program.
```

Program 2

Java program to demonstrate Command line arguments.

Problem Definition

Reading the data from the user.

Problem 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 and will take as String arguments through main method arguments.

Pseudocode

```
Class CommandArgs
{
Public static void main(String args[]) {
  Declare 2 variables n1, n2
read first argument in n1
Read 2nd arguent in n2
Display n1, n2
Display sum of n1 and n2
}
}
```

Problem Validation

Compile the program using the following command

```
Javac CommandArgs.java
```

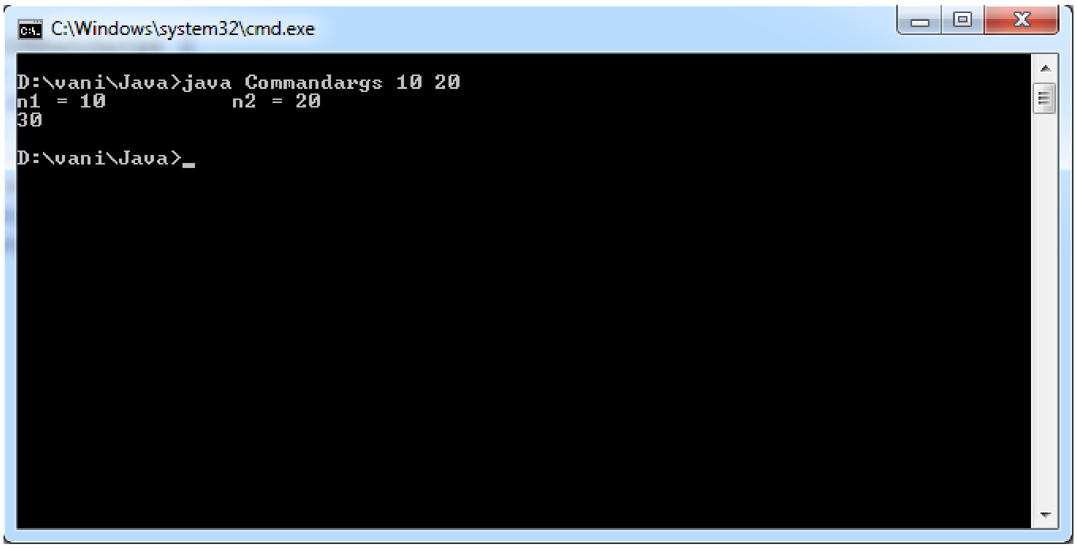
Execute the program using

```
Java CommandArgs 10 20
```

Input:

No Input

Output:



```
D:\vani\Java>java Commandargs 10 20
n1 = 10          n2 = 20
30

D:\vani\Java>_
```

Program 3

Write a Java program to demonstrate Scanner(I/O Streams)

Problem Definition

Reading the data from the user.

Problem Description

The Scanner class is a class in java.util, which allows the user to read values of various types. The Scanner looks for tokens in the input. A token is a series of characters that ends with what Java calls whitespace.

Syntax for Creating Object:

```
Scanner in = new Scanner(System.in);
```

Pseudo Code

```
import the package java.util.*
Class ScannerEx
{
Public static void main(String args[]) {
  Declare 2 variables n1, n2
Create scanner object
read n1 using Scanner Object
Read n2 using Scanner Object
Display n1, n2
Display sum of n1 and n2
}
}
```

Problem Validation

Compile the program using the following command

Javac ScannerEx.java

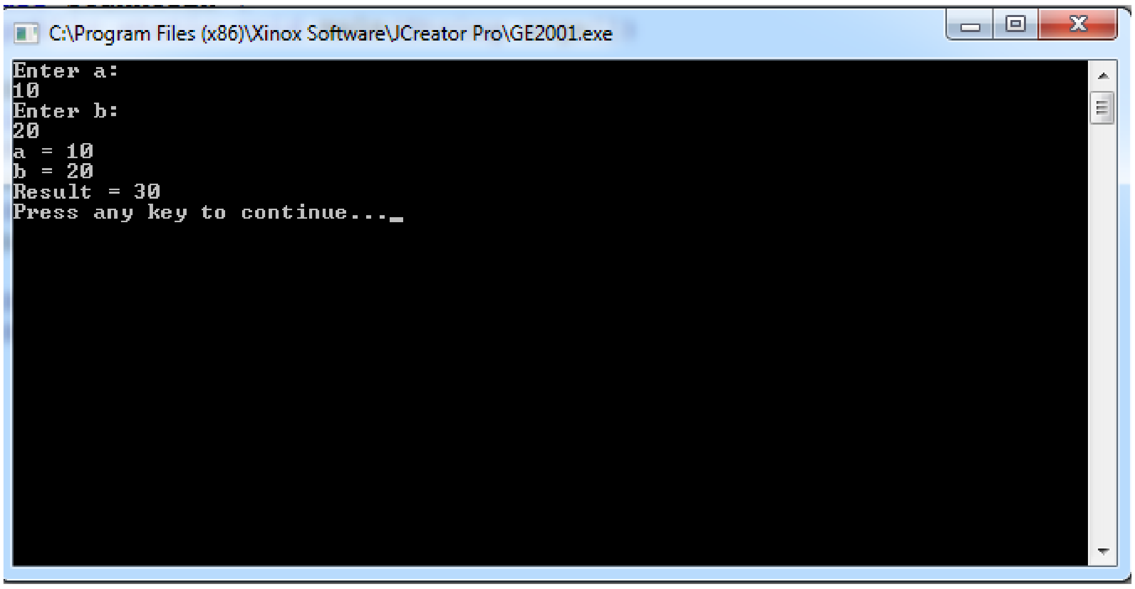
Execute the program using

Java ScannerEx

Input:

```
10
20
```

Output:


C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter a:
10
Enter b:
20
a = 10
b = 20
Result = 30
Press any key to continue...
```

Program 4
Write a Java program to demonstrate BufferedReader(I/O Streams)

Problem Definition

Reading the data from the user.

Problem Description

BufferedReader class is used to read the data from the user as well as from the file. BufferedReader improves performance by buffering input.

Syntax for Creating Object:

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

Pseudocode

```
import the package java.util.*
import java.io.*
Class BuffEx
{
Public static void main(String args[]) throws Exception{
  Declare 2 variables n1, n2
Create BufferedReader object
read n1 using BufferedReader Object and convert to Integer
Read n2 using BufferedReader Object and convert to Integer
Display n1, n2
Display sum of n1 and n2
}
}
```

Problem Validation

Compile the program using the following command

Javac BuffEx.java

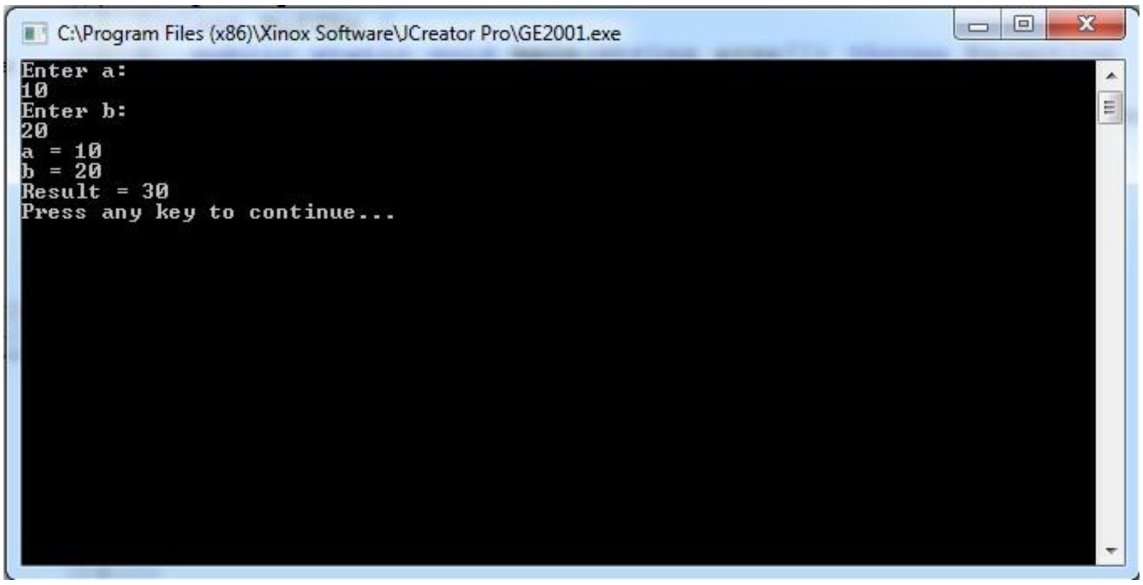
Execute the program using

Java BuffEx

Input:

```
10
20
```

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter a:
10
Enter b:
20
a = 10
b = 20
Result = 30
Press any key to continue...
```

Program 5

Write a Java program to demonstrate Arrays.

Problem Definition

Read data from the user and calculate the sum of array elements.

Problem Description

Making use of Scanner class to read the data from the user into the array and calculate the sum of array elements. Use for loop to read the elements and find the sum.

Pseudocode

```
 import the package java.util.*
Class ArrayDemo
{
Public static void main(String args[]) {
Declare size of array n
Create scanner object
Read n using Scanner Object
Create Array object with size n [an]
Read array vales using Scanner object in for loop
Calculate the sum of array elements
Display array elements and sum.
}
}
```

Problem Validation

Compile the program using the following command

Javac ArrayDemo.java

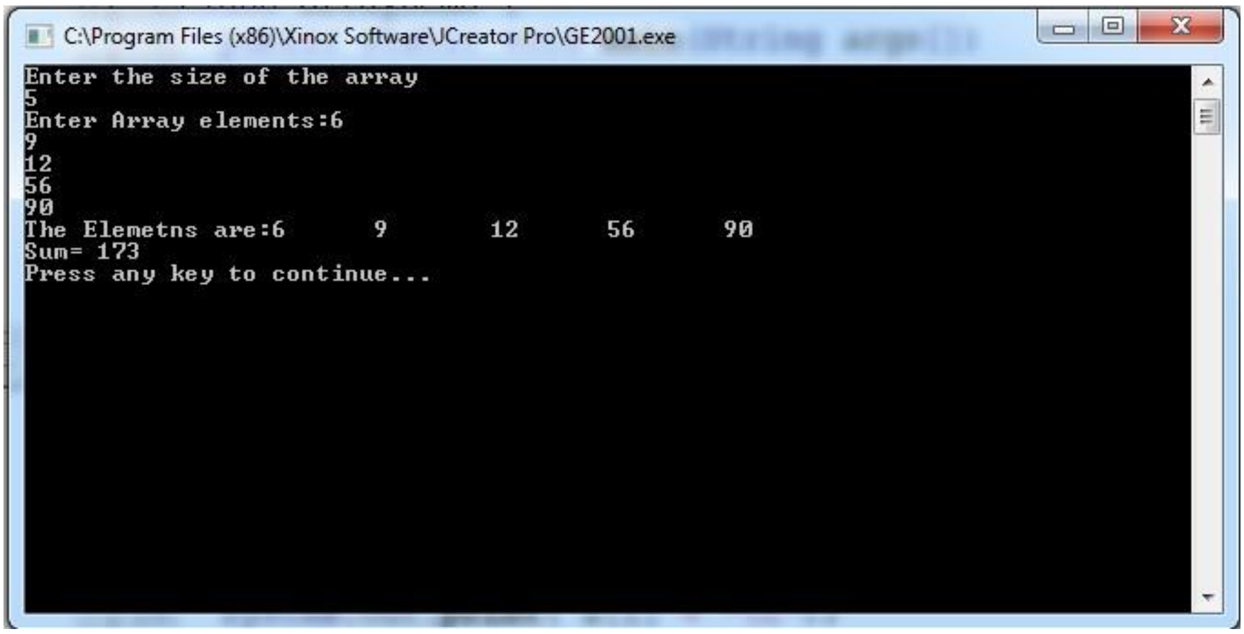
Execute the program using

Java ArrayDemo

Input:

Size 5
Elements: 6 9 12 56 90

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter the size of the array
5
Enter Array elements:6
9
12
56
90
The Elemetns are:6      9       12      56      90
Sum= 173
Press any key to continue...
```

Program 6

Write a Java program to find both the largest and smallest number in a list of integers.

Problem Definition

Read data from the user in an array and find maximum and minimum of the elements.

Problem Description

Making use of Scanner class to read the data from the user into the array and find the maximum and minimum of the elements. Use for loop to read the elements and if condition to check the maximum and minimum of the elements.

Pseudocode

```
import java.util.*;
class MaxMin {
  public static void main(String args[])
{
  Declare n, Max, Min;
  Scanner s = new Scanner(System.in);
  Read size
  Create array a[n]
  Read the elements of the array using for loop
  Max = Min = a[0];
  for(int i=0;i<n;i++) {
          if(Min > a[i] )
             Min = a[i];
          if(Max < a[i] )
             Max = a[i];
  }
  Disply array elements, maximum and minimum values
}
}
```

Problem Validation

Compile the program using the following command

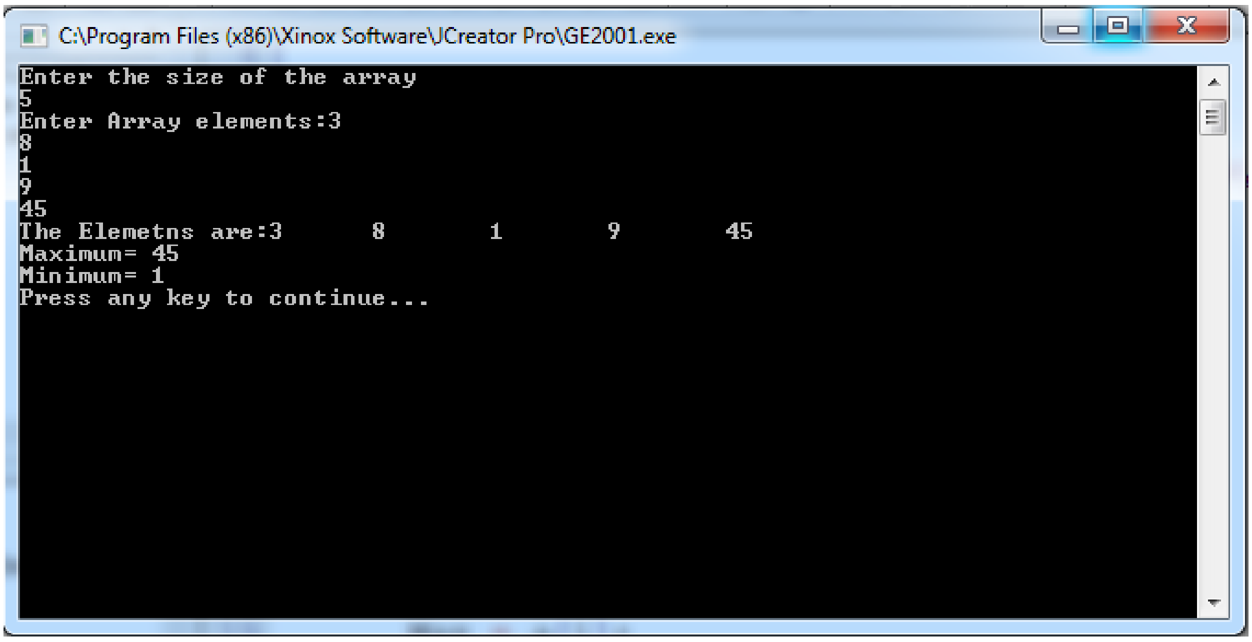
Javac MaxMin.java

Execute the program using

Java MaxMin

Input:
Size 5
Elements: 3 8 1 9 45

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter the size of the array
5
Enter Array elements:3
8
1
9
45
The Elemetns are:3       8       1       9       45
Maximum= 45
Minimum= 1
Press any key to continue...
```

Program 7

Write a Java program that prints all real solutions to the quadratic equation ax2 + bx + c = 0. Read in a, b, c and use the quadratic formula.

Problem Definition

Find the roots of the Quadratic Equation $ax^2+bx+c=0$.

Problem Description

Read the values of a, b, c values for the equation $ax^2+bx=c=0$ using BufferedReader object. Calculate DET value and bsed on the value of det, calculate and display the roots.

If the det is zero, Equation has single root. Root= -b/(2*a)

If the det is > zero, Equation has 2 different roots and they are (–b+sqrt(det))/(2*a) and (–b-sqrt(det))/(2*a)

If the det is < zero, Roots are imaginary.

Pseudocode

```
public class Quadratic
{
  public static void main(String args[])
  {
        Declare a,b,c,r1,r2;
        Scanner s = new Scanner(System.in);
        Read a,b,c values using s
        Calculate Det value d = b*b-4*a*c
        If(d==0)
        {
                Display ROOTS ARE REAL and EQUAL
                Calculate r1=r2=-b/(2*a);
                Display r1, r2
        }
        if(d > 0)
        {
                Display ROOTS ARE REAL and DISTINCT
                r1=-b+Math.sqrt(d)/(2*a);
                 r2=-b-Math.sqrt(d)/(2*a);
                 Display r1, r2
        }       else
                 Display Roots are Imaginary

  }
}
```

Problem Validation

Compile the program using the following command

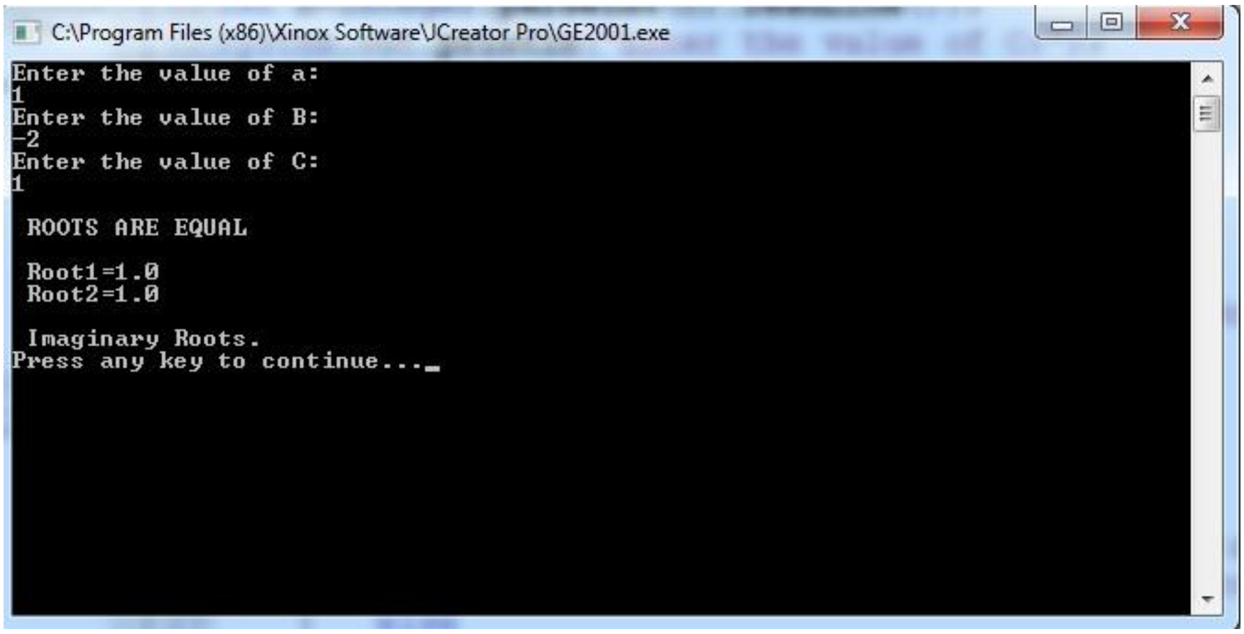
Javac Quadratic.java

Execute the program using
Java Quadratic

Input:

a=1
b=-2
c=1

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter the value of a:
1
Enter the value of B:
-2
Enter the value of C:
1

 ROOTS ARE EQUAL

 Root1=1.0
 Root2=1.0

 Imaginary Roots.
Press any key to continue..._
```

Program 8

Write a Java program that uses both recursive and non recursive functions to print the nth value in the Fibonacci sequence.

Problem Definition

Generate the Fibonacci series using both recursive and non-recursive.

Problem Description

The Fibonacci Sequence is the series of numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, .. The next number is found by adding up the two numbers before it. The 2 is found by adding the two numbers before it (1+1). Initial values in the series are taken as 0, 1.

The Rule is $x_n = x_{n-1} + x_{n-2}$

Pseudocode

a) Non- Recursive

```
import java.util.*;
public class Fibn
{
        public static void main(String args[])
{
        Declare integers f1=0,f2=1,n,i,f3;
        Create Scanner Object. Scanner s = new Scanner(System.in);
        Read the size of the series n
        Display first 2 numbers in the series
        for(i=0;i<n-2;i++)
        {
          Add f1, f2 to get f3
          Display f3
          f1=f2;
          f2=f3;
        }
}
}
```

Problem Validation

Compile the program using the following command

Javac Fibn.java

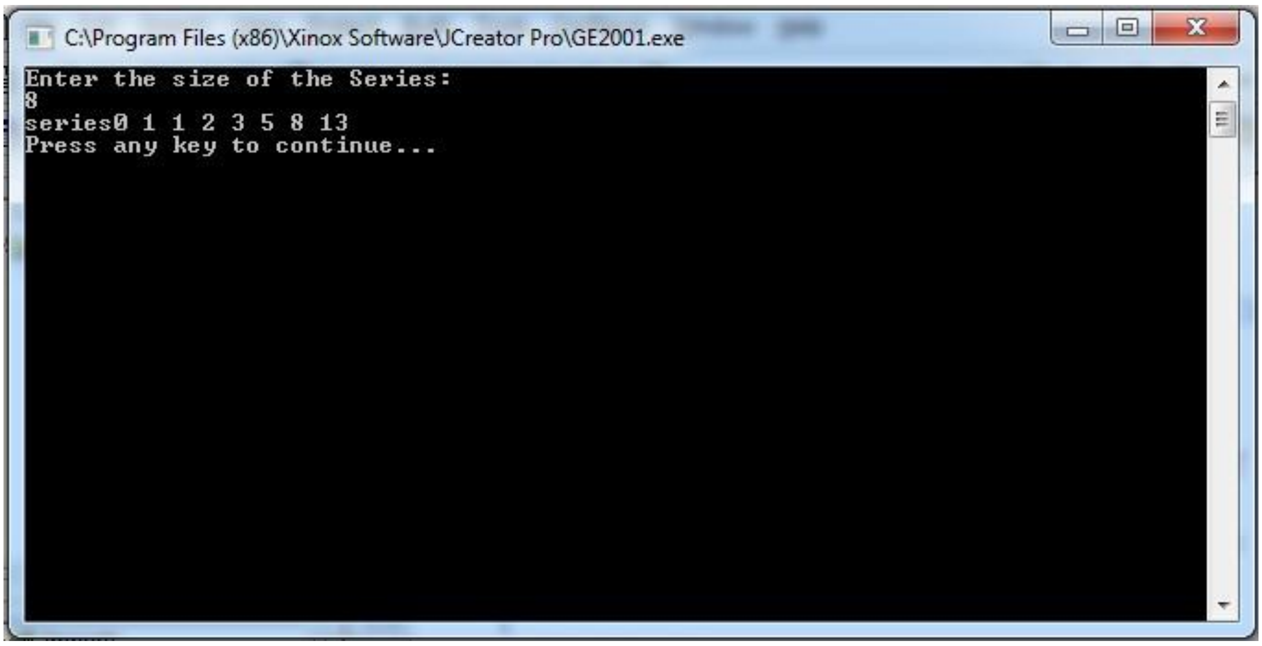
Execute the program using

Java Fibn

Input:

Size = 8

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter the size of the Series:
8
series0 1 1 2 3 5 8 13
Press any key to continue...
```

b) Recursive

```
import java.util.*;
public class Fibrn
{
public static void main(String args[])
{
Declare integers n,i,f3;
Create Scanner Object. Scanner s = new Scanner(System.in);
Read the size of the series n
for(i=0;i<n;i++)
{
Call function fib(i)
Display f3
}
}
Static fib(int d)
{
if(d==0)
    return 0;
  else if(d==1)
     return 1;
     else
     return((d-1)+(d-2));
}
}
```

Problem Validation

Compile the program using the following command

Javac Fibrn.java

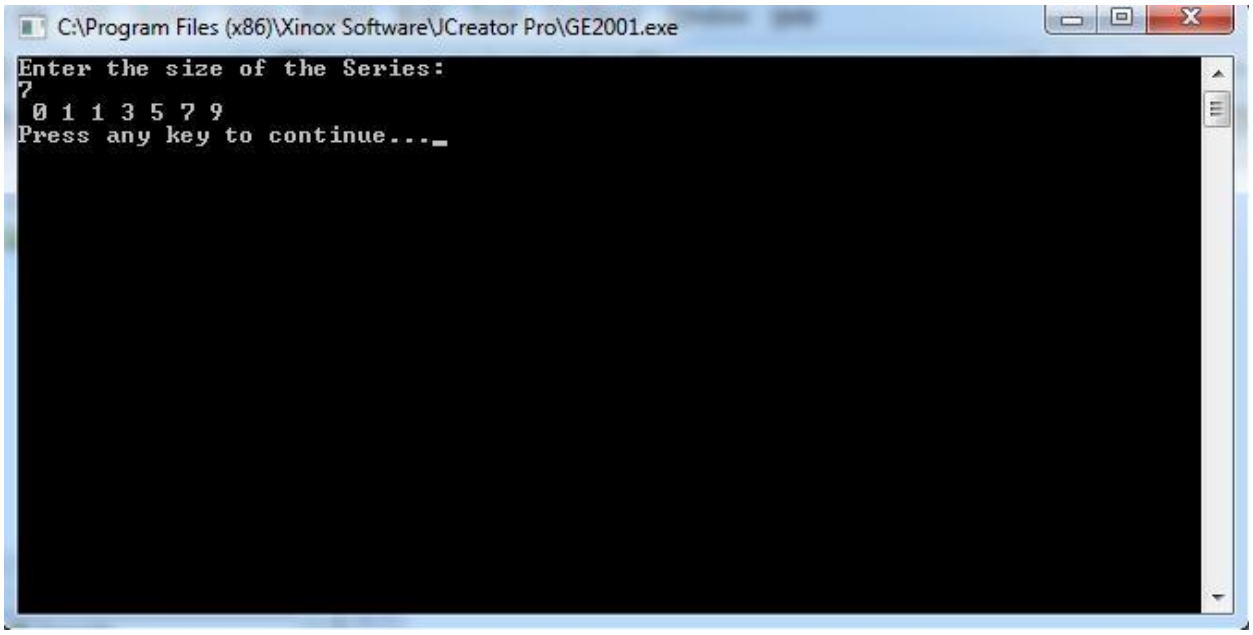
Execute the program using

Java Fibrn

Input:

Size = 7

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter the size of the Series:
7
 0 1 1 3 5 7 9
Press any key to continue...
```

Program 9

Write a Java Program that reads a line of integers, and then displays each integer, and the sum of all the integers

Problem Definition

Reading the data from the user in the form of string. Convert string into numbers and find the sum of the numbers.

Problem Description

Use the StringTokenizer class to convert String into list of numbers and find the sum after conversion. StringTokenizer uses the following methods for conversion and read token by token.

Object Creation:

```
StringTokenizer st = new StringTokenizer(String);
```

Methods:

```
hasMoreTokens()
nextToken()
```

Pseudocode

```
import java.io.*;
import java.util.*;
public class StringToken
{
        public static void main(String args[]) throws IOException
{
        Create BufferedReader Object br as follows.
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        Declare i, n, sum=0;
        Read String s
        Convert string into tokens as StringTokenizer st = new StringTokenizer(s);
        while(st.hasMoreTokens())
        {
                Take the token n and convert to integer
                Display n value
                Add n to sum
        }
        Display sum
}
}
```

Problem Validation

Compile the program using the following command

Javac StringToken.java

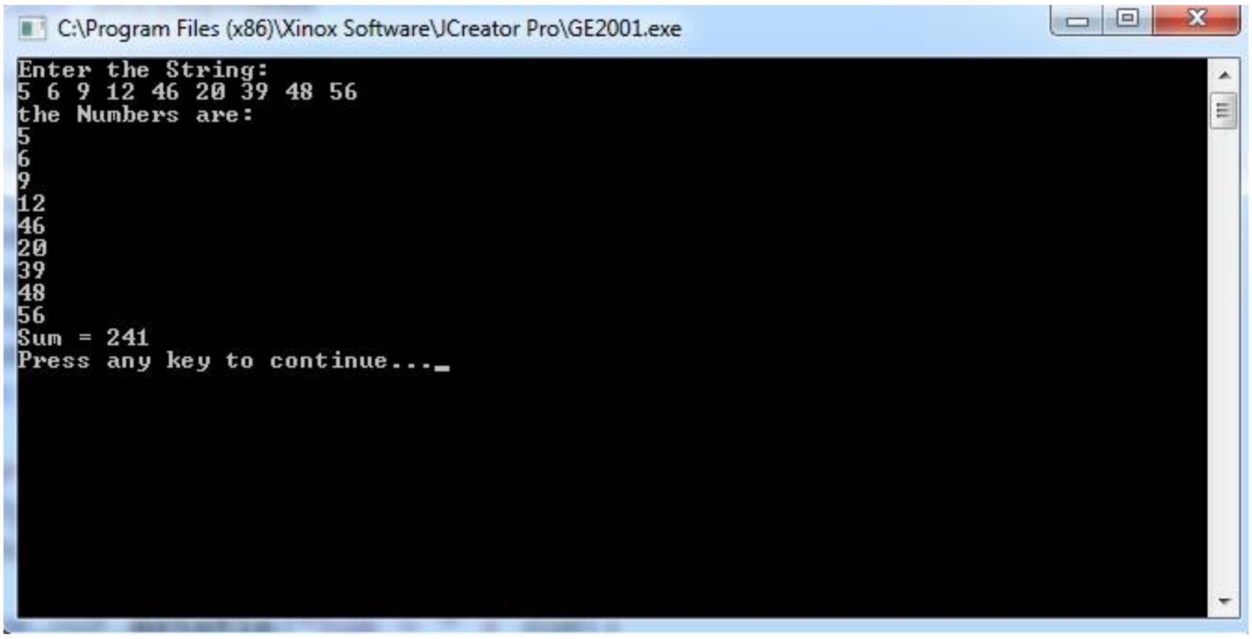
Execute the program using

Java StringToken

Input:

5 6 9 12 46 20 39 48 56

Output:


C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter the String:
5 6 9 12 46 20 39 48 56
the Numbers are:
5
6
9
12
46
20
39
48
56
Sum = 241
Press any key to continue...
```

Program 10

Write a Java program that checks whether a given string is palindrome or not.

Problem Definition

Check whether the given String is Palindrome or not.

Problem Description

A Palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward.
Here we are checking related to Strings. First Read the string and reverse it. Compare both the strings and finally display whether the given string is palindrome or not.

Pseudocode

```
import java.util.*;
public class StrPalin{
 public static void main(String[] args) {
 Declare Str, revstr;
Create Scanner Object. Scanner in = new Scanner(System.in);
Read String using in.
Reverse the string using for loop.
for(int  i=str.length()-1;i>=0;--i){
revstr +=str.charAt(i);
}
Display revstr.
 Check whether str and revstr are quual or not. If equal
 Display The string is Palindrome
else
Display The string is not Palindrome
}
}
```

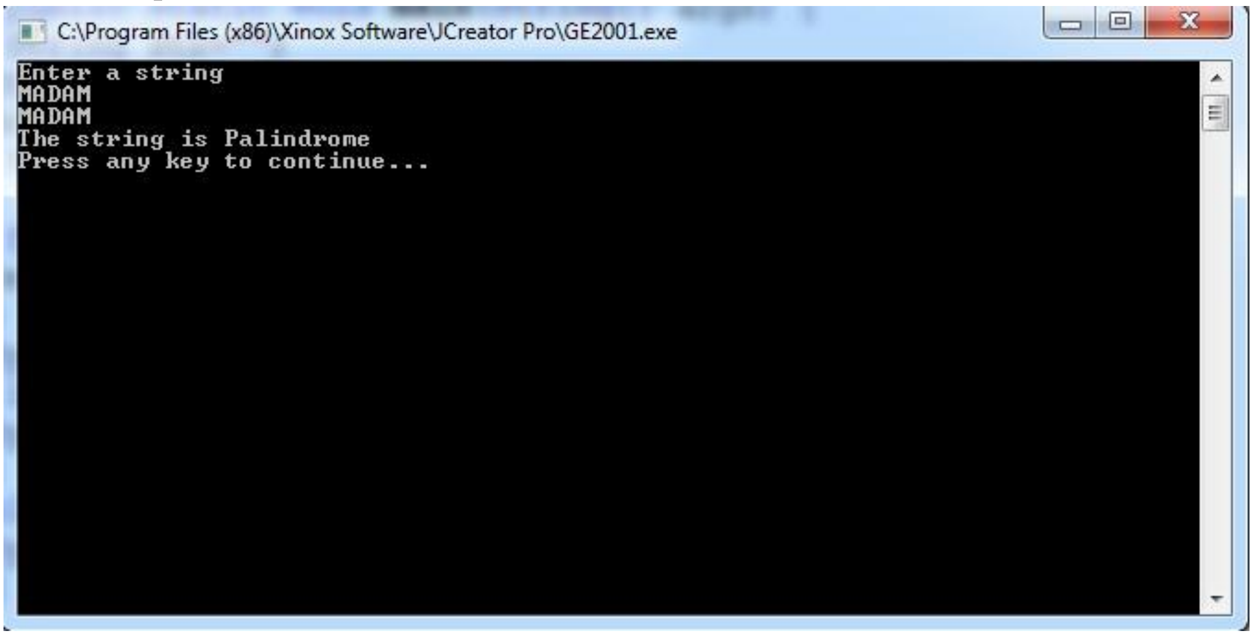
Problem Validation

Compile the program using the following command
Javac StrPalin.java
Execute the program using
Java StrPalin

Input:

Str = MADAM

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Enter a string
MADAM
MADAM
The string is Palindrome
Press any key to continue...
```

Program 11

Write a Java program to sort a list of names in ascending order.

Problem Definition

Sort the given list of names in Ascending order.

Problem Description

Read list of Strings and compare each string to put it in the alphabetical order.

Pseudocode

```
import java.io.*;
class SortStr
{
public static void main(String args[])throws IOException
{
Create BufferedReafer Object.
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
Read the number of Strings to be sorted. n
Create String Array x with size n;
for(int i=0;i<n;i++)
{
Read all the strings in x[i]
}
Create an empty string s
for(int i=0;i<n;i++)
{
for(int j=0;j<n;j++)
{
Compare the characters in 2 strings and swap if the first string is bigger.
if(x[i].compareTo(x[j])<0)
{
s=x[i];
x[i]=x[j];
x[j]=s;
}
}
}
for(int i=0;i<n;i++)
{
Display the strings after sorting
}
}
}
```

Problem Validation

Compile the program using the following command

Javac SortStr.java

Execute the program using

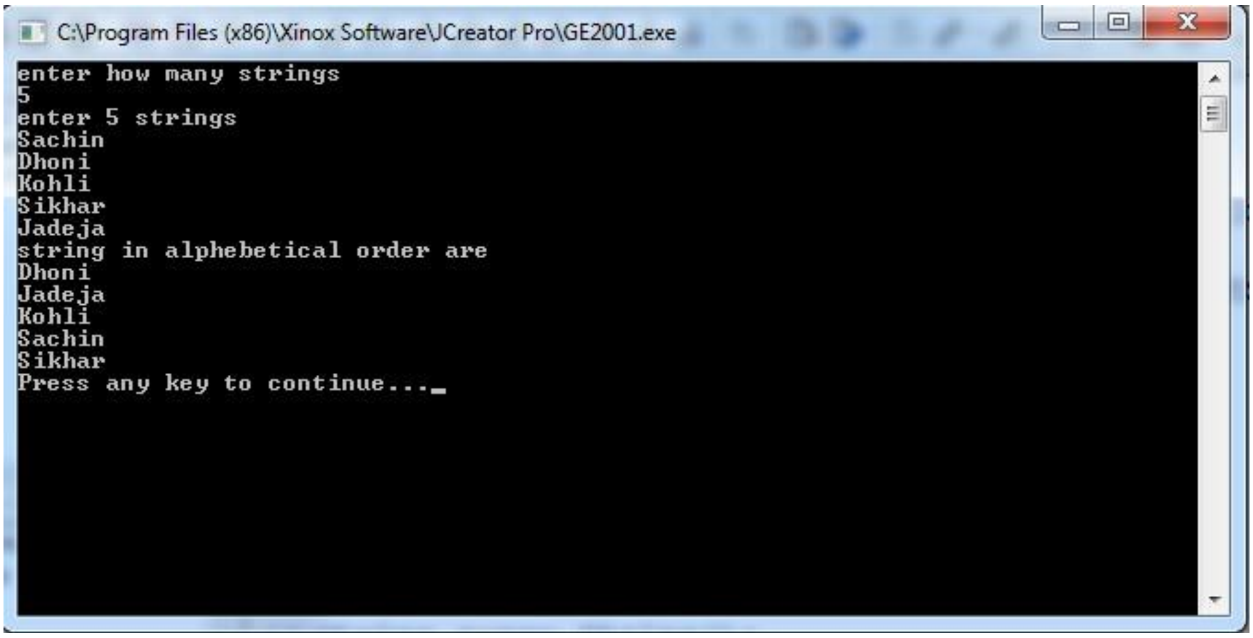
Java SortStr

Input:

Size = 5

The strings are Sachin, Dhoni, Kohli, Sikhar, Jadeja

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
enter how many strings
5
enter 5 strings
Sachin
Dhoni
Kohli
Sikhar
Jadeja
string in alphebetical order are
Dhoni
Jadeja
Kohli
Sachin
Sikhar
Press any key to continue...
```

Program 12

A program to illustrate the concept of class with Constructor overloading,

Problem Definition

Constructor overloading demonstration.

Problem Description

Constructors are used to assign initial values to instance variables of the class. A default constructor with no arguments will be called automatically by the Java Virtual Machine (JVM). Constructor is always called by new operator. Constructors are declared just like as we declare methods, except that the constructor doesn‘t have any return type. Constructor can be overloaded provided they should have different arguments because JVM differentiates constructors on the basis of arguments passed in the constructor.

Pseudocode

```
Create class Rectangle{
  Declare l, b;
  Declare p, q;
  public Rectangle(int x, int y){
  assign x to l;
  assign y to b;
  }
  public int first(){
  return(l * b);
  }
  public Rectangle(int x){
  assign x to l, b;
  }
  public int second(){
  return(l * b);
  }
  public Rectangle(float x){
  assign x to p, q;
  }
  public float third(){
  return(p * q);
  }
  public Rectangle(float x, float y){
  assign x to p;
  assign y to q;
  }
  public float fourth(){
  return(p * q);
  }
}
```

```
class ConsOverload
{
 public static void main(String args[])
 {
  Create Rectangle Class Object retangle1=new Rectangle(2,4);
  Call first method using rectangle1 and store result r1
  Display area r1
  Create Rectangle Class Object rectangle2=new Rectangle(5);
  Call second method using rectangle2 and store result r2
  Display area r2
  Create Rectangle Class Object rectangle3=new Rectangle(2.0f);
  Call third method using rectangle2 and store result r3
  Display area r3
  Create Rectangle Class Object rectangle4=new Rectangle(3.0f,2.0f);
  Call forth method using rectangle4 and store result r4
  Display area r4
 }
}
```

Problem Validation

Compile the program using the following command

Javac ConsOverload.java

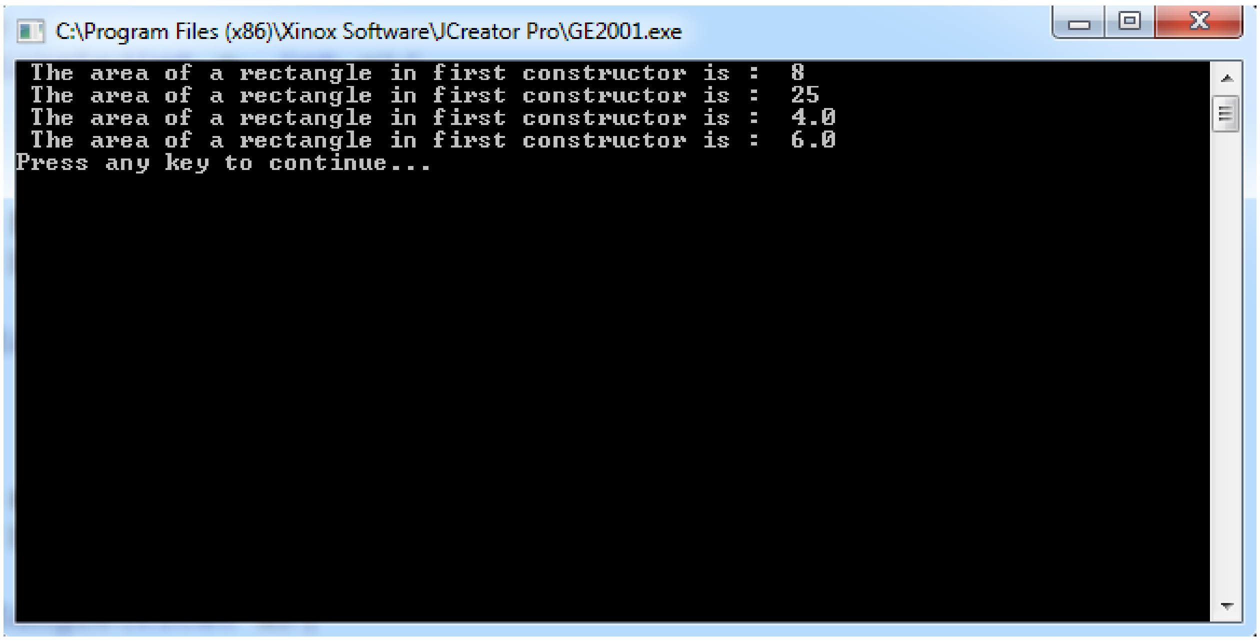
Execute the program using

Java ConsOverload

Input:

No Input

Output:


C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
 The area of a rectangle in first constructor is :  8
 The area of a rectangle in first constructor is :  25
 The area of a rectangle in first constructor is :  4.0
 The area of a rectangle in first constructor is :  6.0
Press any key to continue...
```

Program 13

A program to illustrate the concept of class with Method overloading

Problem Definition

Method overloading demonstration.

Problem Description

In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be overloaded, and the process is referred to as method overloading. Method overloading is one of the ways that Java implements polymorphism.

Pseudocode

```
Declare class OverloadDemo {
void test() {
display No parameters
}
void test(int a) {
Display the parameter a value
}
void test(int a, int b) {
Display the parameters a, b values
}
double test(double a) {
Display the parameter a value
return a*a;
}
}
class MethodOverload {
public static void main(String args[]) {
Create Object for OverloadDemo ob = new OverloadDemo();
Call ob.test();
Call ob.test(10);
Call ob.test(10, 20);
Declare result and store result = ob.test(123.25);
Display result
}
}
```

Problem Validation

Compile the program using the following command

Javac OverloadDemo.java

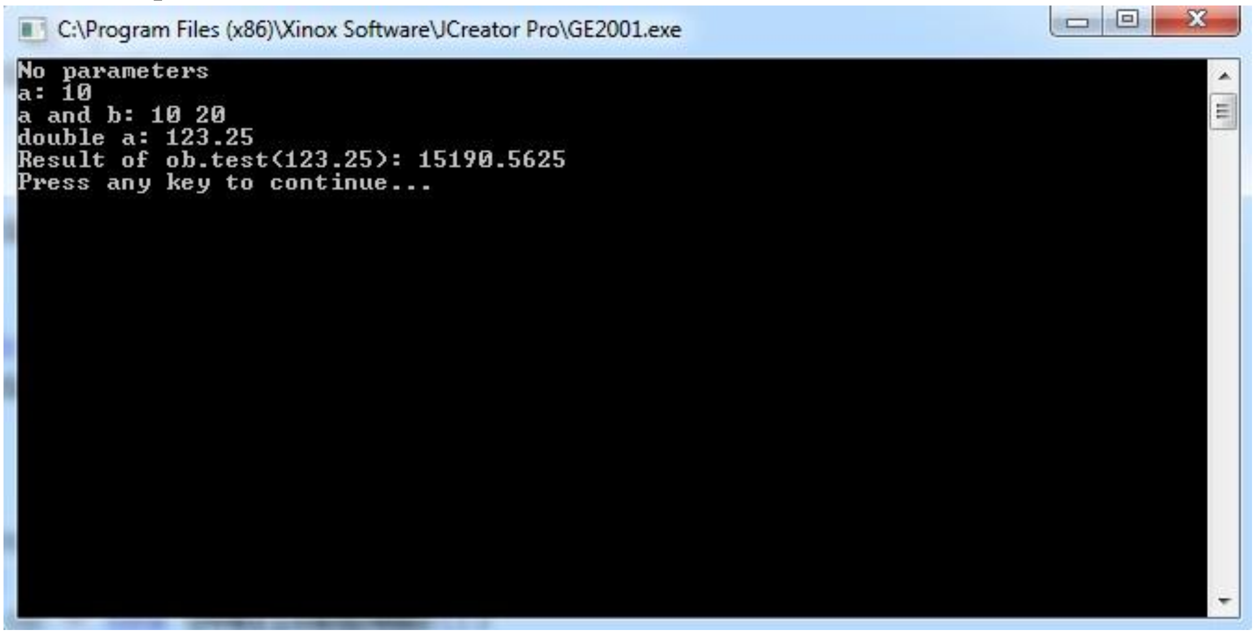
Execute the program using

Java OverloadDemo

Input:

No Input

Output:



```
C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe
No parameters
a: 10
a and b: 10 20
double a: 123.25
Result of ob.test(123.25): 15190.5625
Press any key to continue...
```

Program 14

A program to illustrate the concept of Dynamic Polymorphism.

Problem Definition

Demonstration of Dynamic or runtime Polymorphism.

Problem Description

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism. Let's begin by restating an important principle: a superclass reference variable can refer to a subclass object.

Java uses this fact to resolve calls to overridden methods at run time. Here is how. When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time. When different types of objects are referred to, different versions of an overridden method will be called.

Pseudocode

```
Declare class Figure {
Declare dim1, dim2;
  Figure(double a, double b) {
    Assign a to dim1, b to dim2
  }
  double area() {
    Display Area for Figure is undefined
    return 0;
  }
}
Declare class Rectangle extends Figure {
  Rectangle(double a, double b) {
    Call super class constructor by passing a, b
  }
  double area() {
    Display Inside Area for Rectangle
    Calculate dim1 * dim2 and return the result
  }
}
Declare class Triangle extends Figure {
  Triangle(double a, double b) {
    Call super class constructor by passing a, b
  }
  double area() {
    Display Inside Area for Triangle
    Calculate (dim1 * dim2)/2 and return the result
  }
}
Declare class FindAreas {
  public static void main(String args[]) {
    Create Object for Figure f = new Figure(10, 10);
    Create Object for Rectangle r = new Rectangle(9, 5);
    Create Object for Triangle t = new Triangle(10, 8);

    Create a reference for Figure figref;
    Assign r to reference figref = r;
    Display area
    Assign t to reference figref = t;
    Display area
    Assign f to reference figref = f;
    Display area
  }
}
```

Problem Validation

Compile the program using the following command

Javac FindAreas.java

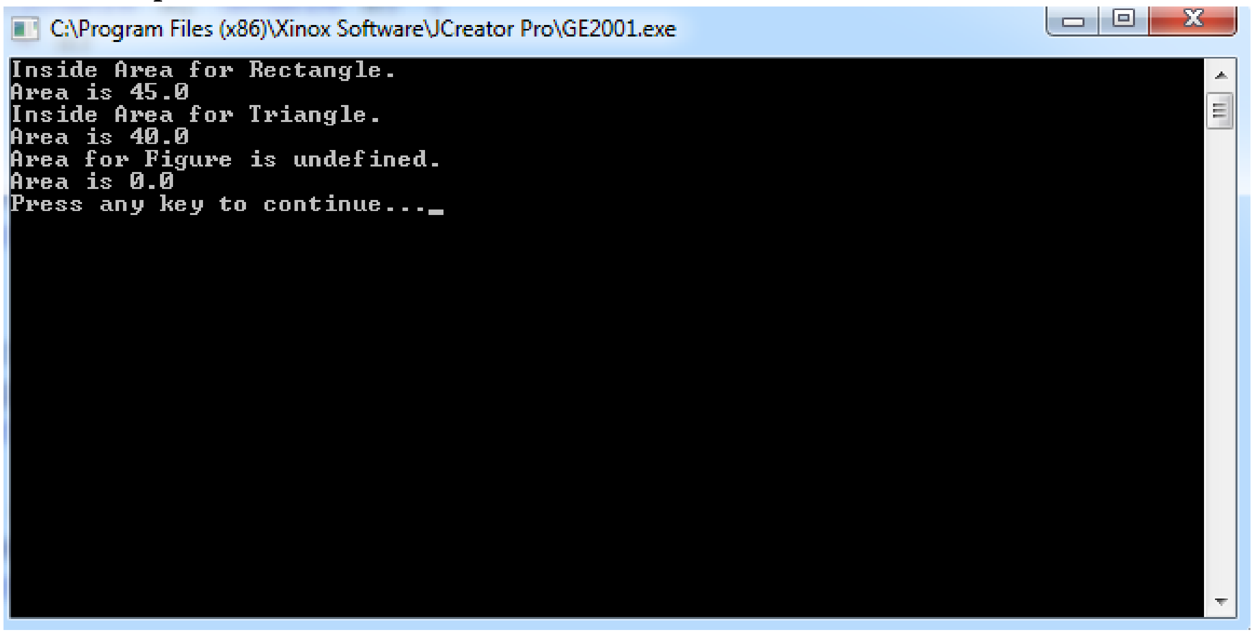
Execute the program using

Java FindAreas

Input:

No Input

Output:


C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Inside Area for Rectangle.
Area is 45.0
Inside Area for Triangle.
Area is 40.0
Area for Figure is undefined.
Area is 0.0
Press any key to continue...
```

Program 15

A program to illustrate the concept of Single inheritance

Problem Definition

Demonstration of Single Inheritance.

Problem Description

The concept of inheritance is used to make the things from general to more specific e.g. When we hear the word vehicle then we got an image in our mind that it moves from one place to another place it is used for traveling or carrying goods but the word vehicle does not specify whether it is two or three or four wheeler because it is a general word.

When a subclass is derived simply from it‘s parent class then this mechanism is known as simple inheritance. In case of simple inheritance there is only a sub class and it's parent class. It is also called single inheritance or one level inheritance.

Pseudocode

```
Declare class Box {
Declare width, height, depth
Box(Box ob) {
Assign ob.width to width
height = ob.height;
depth = ob.depth;
}
Box(double w, double h, double d) {
Assign w to width
Assign h to height
Assign d to depth
}
Box() {
Assign width = height = depth = -1
}
Box(double len) {
Assign len to width, height, depth
}
double volume() {
return width * height * depth;
}
}
class BoxWeight extends Box {
double weight;
BoxWeight(double w, double h, double d, double m) {
Call the super class constructor by passing parameters w,h,d
Assign m to weight
}
}

class SingleInh {
public static void main(String args[]) {
Create Object for BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
Create Object for BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
Call volume() on mybox1 object and store result in vol
Display volume vol
Display weight on mybox1
Call volume() on mybox2 object and store result in vol
Display volume vol
Display weight on mybox2
}
}
```

Problem Validation

Compile the program using the following command

Javac SingleInh.java

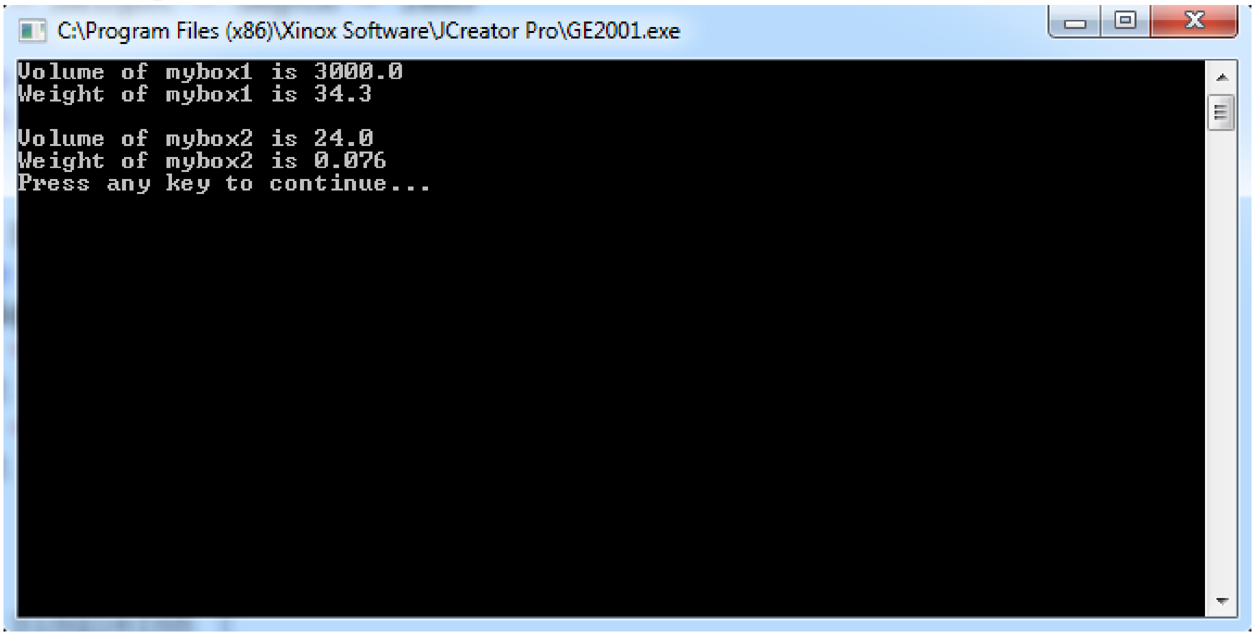
Execute the program using

Java SingleInh

Input:

No Input

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Volume of mybox1 is 3000.0
Weight of mybox1 is 34.3

Volume of mybox2 is 24.0
Weight of mybox2 is 0.076
Press any key to continue...
```

Program 16

A program to illustrate the concept of Multi level inheritance

Problem Definition

Demonstration of Multilevel Inheritance

Problem Description

It is the enhancement of the concept of inheritance. When a subclass is derived from a derived class then this mechanism is known as the multilevel inheritance. The derived class is called the subclass or child class for it's parent class and this parent class works as the child class for it's just above (parent) class. Multilevel inheritance can go up to any number of level.

Pseudocode

```
Declare class Box {
Declare width, height, depth
Box(Box ob) {
Assign ob.width to width
height = ob.height;
depth = ob.depth;
}
Box(double w, double h, double d) {
Assign w to width
Assign h to height
Assign d to depth
}
Box() {
Assign width = height = depth = -1
}
Box(double len) {
Assign len to width, height, depth
}
double volume() {
return width * height * depth;
}
}
class BoxWeight extends Box {
double weight;
BoxWeight(double w, double h, double d, double m) {
Call the super class constructor by passing parameters w,h,d
Assign m to weight
}
}
Declare class Shipment extends BoxWeight {
double cost;
Shipment(Shipment ob) {
Call super class constructor by passing object ob
Assign object cost value to cost
}
Shipment(double w, double h, double d,double m, double c) {
Call super class constructor by passing w, h, d, m
Assign c to cost
}
Shipment() {
Call super class default constructor
Assign cost = -1;
}
Shipment(double len, double m, double c) {
Call super class constructor by passing len, m
Assign c to cost
}
}
class MultiInh {
public static void main(String args[]) {
Create Object for Shipment shipment1 = new Shipment(10, 20, 15, 10, 3.41);
Create Object for Shipment shipment2 = new Shipment(2, 3, 4, 0.76, 1.28);
Call volume method on shipment1 and store result in vol
Display vol
Display weight on shipment1
Display cost on shipment1
Call volume method on shipment2 and store result in vol
Display vol
Display weight on shipment2
Display cost on shipment2
}
}
```

Problem Validation

Compile the program using the following command

Javac MultiInh.java

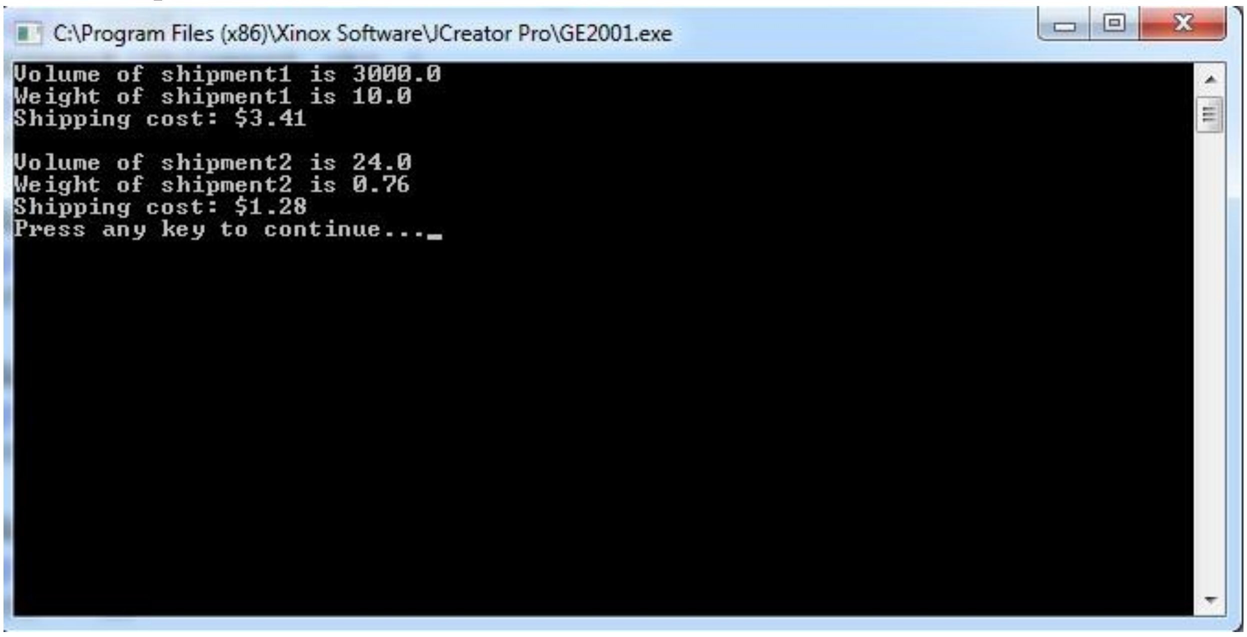
Execute the program using

Java MultiInh

Input:

No Input

Output:


C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Volume of shipment1 is 3000.0
Weight of shipment1 is 10.0
Shipping cost: $3.41

Volume of shipment2 is 24.0
Weight of shipment2 is 0.76
Shipping cost: $1.28
Press any key to continue...
```

Program 17

A program to illustrate user defined packages.

Problem Definition

Creating and using Packages.

Problem Description

Packages in Java is a mechanism to encapsulate a group of classes, interfaces and sub packages. Some of the existing packages in Java are:

- java.lang - bundles the fundamental classes
- java.io - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related.

Steps for Creating Package:

1. Create a package with a .class file.
2. set the classpath from the directory from which you would like to access. It may be in a different drive and directory. Let us call it as a target directory.
3. Write a program and use the file from the package.

Pseudocode

Creating the Package.

```
package animals;
public interface Animal {
 declare eat() as public
 declare travel() as public
}
```

Importing Package

```
Import animals.*;
public class MammalInt implements Animal{

Implement eat()
Implement travel()

  public int noOfLegs(){
    return 0;
  }

  public static void main(String args[]){
    Create object for the class MammalInt m = new MammalInt();
    Call eat()
Call travel()
  }
}
```

Problem Validation

Compile the program using the following command

Javac –d . Animal.java

Javac MammalInt.java

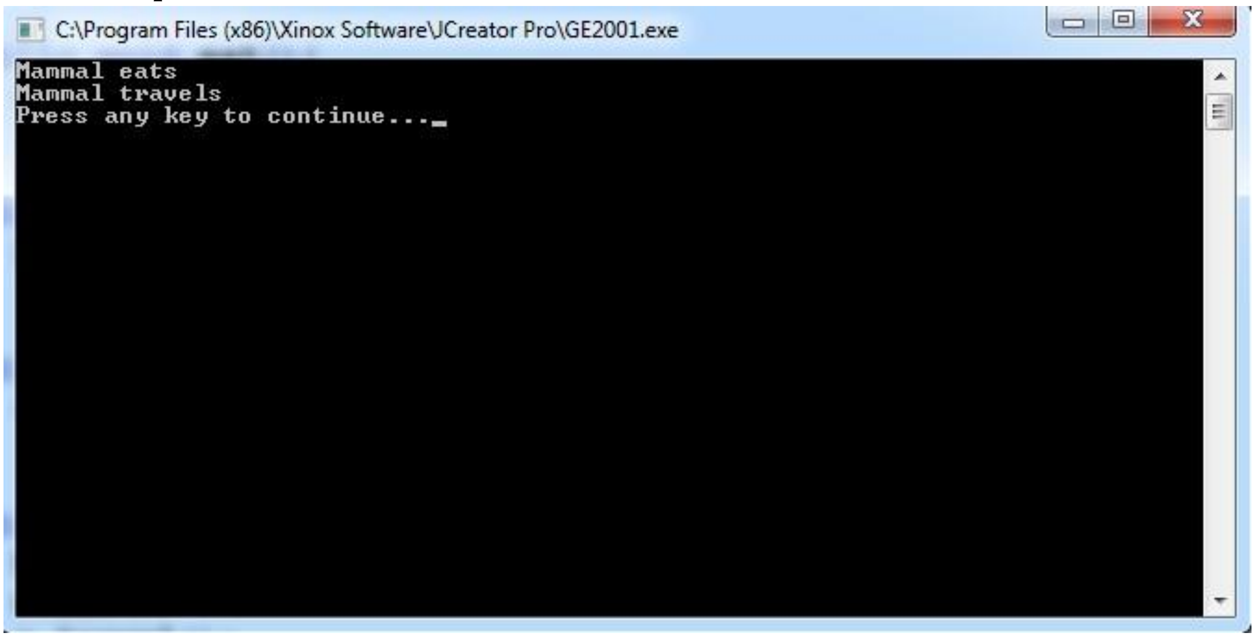
Execute the program using

Java MammalInt

Input:

No Input

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Mammal eats
Mammal travels
Press any key to continue...
```

Program 18

A program to illustrate the concept of Abstract Classes.

Problem Definition

Demonstration on creation of Abstract Class.

Problem Description

An abstract class is a class that is declared by using the abstract keyword. It may or may not have abstract methods. Abstract classes cannot be instantiated, but they can be extended into sub-classes.

Java provides a special type of class called an abstract class, which helps us to organize our classes based on common methods. An abstract class lets you put the common method names in one abstract class without having to write the actual implementation code.

An abstract class can be extended into sub-classes; these sub-classes usually provide implementations for all of the abstract methods.

- Abstract class contains abstract methods.
- Program can't instantiate an abstract class.
- Abstract classes contain mixture of non-abstract and abstract methods.
- If any class contains abstract methods then it must implements all the abstract methods of the abstract class.

Pseudocode

```
Create abstract class Figure {
declare dim1, dim2
Figure(double a, double b) {
Assign a to dim1
Assign b to dim2
}
Declare abstract double area();
}
class Rectangle extends Figure {
Rectangle(double a, double b) {
Class the super class constructor by passing the parameters a, b
}
double area() {
Display Inside Area for Rectangle
Calculate and return dim1 * dim2;
}
}
class Triangle extends Figure {
```

```
Triangle(double a, double b) {
Class the super class constructor by passing the parameters a, b
}
double area() {
Display Inside Area for Triangle
Calculate and return dim1 * dim2/2;
}
}
class AbstractAreas {
public static void main(String args[]) {
Create Object for the class Rectangle r = new Rectangle(9, 5);
Create Object for the class Triangle t = new Triangle(10, 8);
Create a reference for Figure figref; // this is OK, no object is created
Assign r to figref
Call area() using figref and display the result
Assigntr to figref
Call area() using figref and display the result
}
}
```

Problem Validation

Compile the program using the following command

Javac AbstractAreas.java

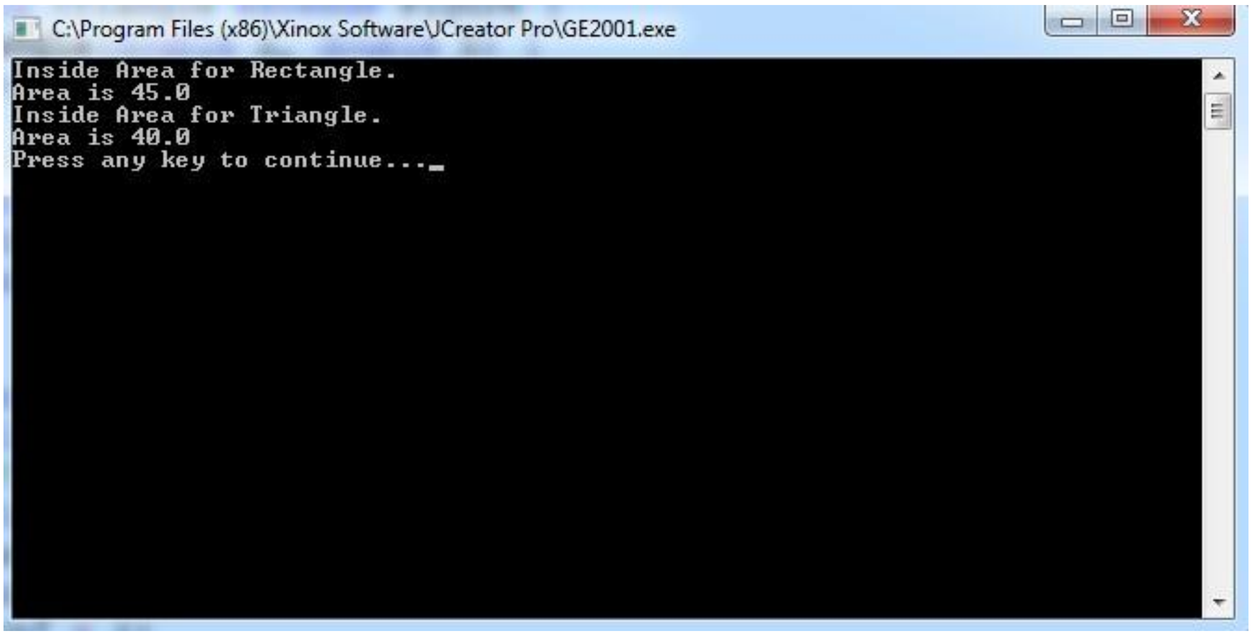
Execute the program using

Java AbstractAreas

Input:

No Input

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Inside Area for Rectangle.
Area is 45.0
Inside Area for Triangle.
Area is 40.0
Press any key to continue...
```

Program 19

A program using Interfaces.

Problem Definition

Demonstration of Interface.

Problem Description

Using the keyword interface, you can fully abstract a class‘ interface from its implementation. That is, using interface, you can specify what a class must do, but not how it does it. Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body. To implement an **interface**, a class must create the complete set of methods defined by the interface. However, each class is free to determine the details of its own implementation. By providing the **interface** keyword, Java allows you to fully utilize the ―one interface, multiple methods‖ aspect of polymorphism.

Implementation of Stack program using Interfaces.

Pseudocode

Declare an Interface

```
Declare interface printable{
Declare the method print()
}
```

Create the class and call the interface.

```
class IntDemo implements printable{
Implement the print method
public static void main(String args[]){
Create object for Class IntDemo obj = new IntDemo();
Call Print method
}
}
```

Problem Validation

Compile the program using the following command

Javac IntDemo .java

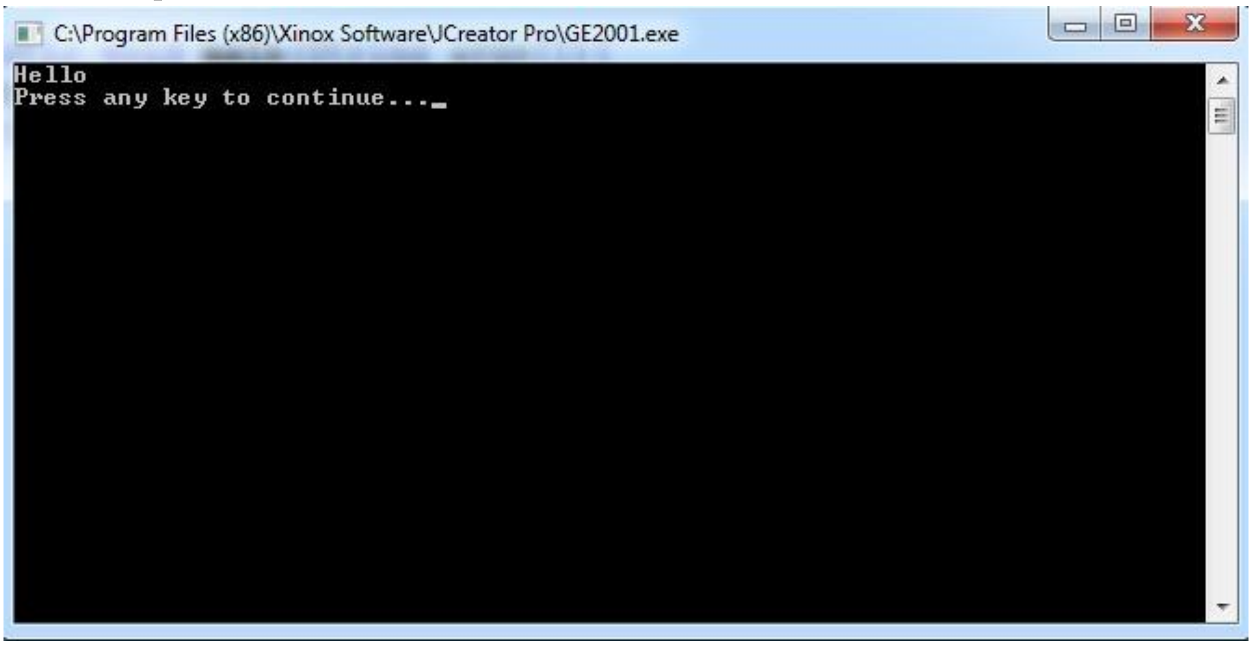
Execute the program using

Java IntDemo

Input:

No Input

Output:



C:\Program Files (x86)\Xinox Software\JCreator Pro\GE2001.exe

```
Hello
Press any key to continue...
```

Program 20

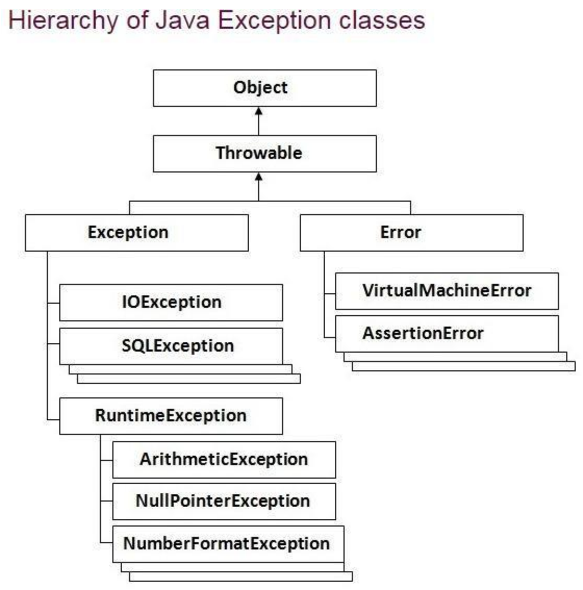
Write a Java program to implement the concept of exception handling

Problem Definition

Java program to implement the concept of exceptional handling.

Problem Description

The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.
In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.
Advantage of Exception Handling
The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling.



Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. There are three types of exceptions:

- Checked Exception
- Unchecked Exception
- Error

Difference between checked and unchecked exceptions

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Pseudocode

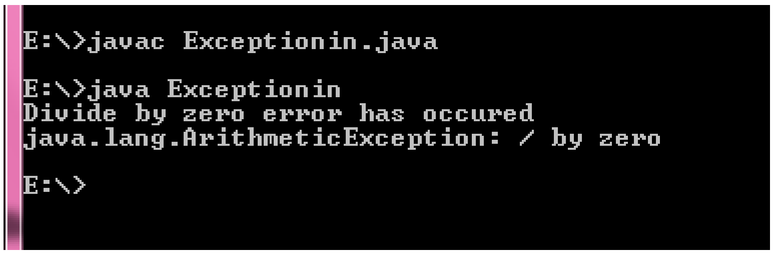
```
1. Create a class and initialize class members and methods.
2. Read input values a,b,c
3. Compute d=a/(b-c) in try block
4. Provide catch(Exception e) block
5. Display –Divide by zero error‖ message
6. End.
```

Problem Validation

Compile the program using the following command
Javac ExceptionChk.java
Execute the program using
Java ExceptionChk

Input: a=10,b=5,c=5

Output:



```
E:\>javac Exceptionin.java

E:\>java Exceptionin
Divide by zero error has occured
java.lang.ArithmeticException: / by zero

E:\>
```

Program 21

A program to illustrate the concept of threading using Thread Class

Problem Definition

A program to illustrate concept of threading using thread class.

Problem Description

Java is a *multi threaded programming language* which means we can develop multi threaded program using Java. A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources especially when your computer has multiple CPUs.

By definition multitasking is when multiple processes share common processing resources such as a CPU. Multi threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Create Thread by Extending Thread Class:

The second way to create a thread is to create a new class that extendsThread class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

Step 1

You will need to override run( ) method available in Thread class. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run() method:

public void run( )

Step 2

Once Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method. Following is simple syntax of start() method:

void start( );

Pseudocode

1. Create a class and extend –Thread ‖ class

2. Now override the –public void run()‖ method and write the logic there (This is the method which will be executed when this thread is started)

That‘s it, now start this thread as given below

1. Create an object of the above class

2. Call the method ―start‖ on the object created. Now our thread will start its execution in parallel

Problem Validation

Compile the program using the following command

Javac FileName.java

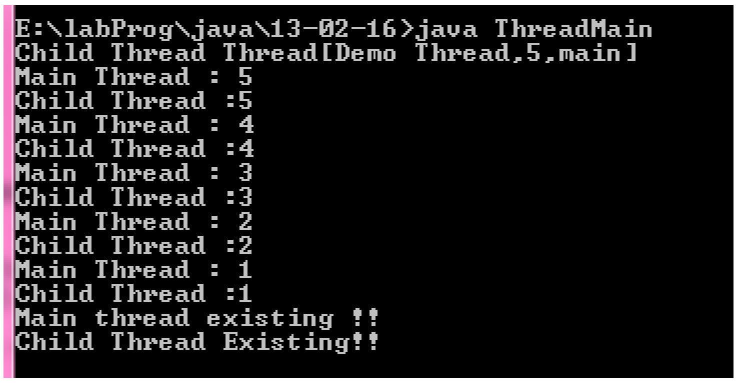
Execute the program using

Java MainClassName

Input:

No Input

Output:



```
E:\labProg\java\13-02-16>java ThreadMain
Child Thread Thread[Demo Thread,5,main]
Main Thread : 5
Child Thread :5
Main Thread : 4
Child Thread :4
Main Thread : 3
Child Thread :3
Main Thread : 2
Child Thread :2
Main Thread : 1
Child Thread :1
Main thread existing !!
Child Thread Existing!!
```

Program 22

A program to illustrate the concept of threading using runnable Interface.

Problem Definition

A program to illustrate the concept of threading using runnable interface.

Problem Description

Create Thread by Implementing Runnable Interface:

If your class is intended to be executed as a thread then you can achieve this by implementing Runnable interface. You will need to follow three basic steps:

Step 1:

As a first step you need to implement a run() method provided by Runnableinterface. This method provides entry point for the thread and you will put you complete business logic inside this method. Following is simple syntax of run() method:

public void run( )

Step 2:

At second step you will instantiate a Thread object using the following constructor:

Thread(Runnable threadObj, String threadName);

Where, threadObj is an instance of a class that implements the Runnableinterface and threadName is the name given to the new thread.

Step 3

Once Thread object is created, you can start it by calling start( ) method, which executes a call to run( ) method. Following is simple syntax of start() method:

void start( );

Pseudocode

1. Create a class implement ―Runnable‖ interface.

2. Now override the –public void run()‖ method and write the logic there (This is the method which will be executed when this thread is started).

That‘s it, now start this thread as given below

1. Create an object of the above class

2. Allocate a thread object for our thread

3. Call the method —start‖ on the allocated thread object.

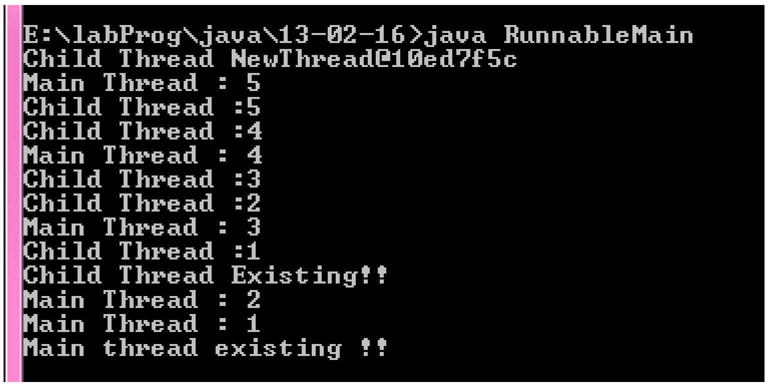
Problem Validation

Compile the program using the following command
Javac FileName.java
Execute the program using
Java MainClassName

Input:

No Input

Output:



```
E:\labProg\java\13-02-16>java RunnableMain
Child Thread NewThread@10ed7f5c
Main Thread : 5
Child Thread :5
Child Thread :4
Main Thread : 4
Child Thread :3
Child Thread :2
Main Thread : 3
Child Thread :1
Child Thread Existing!!
Main Thread : 2
Main Thread : 1
Main thread existing !!
```

Program 23

A program to illustrate the concept of multi-threading that creates three threads. First thread displays “Good Morning” every one second, the second thread displays “Hello” every two seconds and the third thread displays “Welcome” every three seconds.

Problem Definition

Program to illustrate the concept of multithreading that creates three threads. First thread displays ―Good Morning‖ every second**,** the second thread displays ―Hello‖ every two seconds and the third thread displays ―Welcome‖ every three seconds.

Problem Description

Create three separate threads that will calculate the average, minimum and maximum of a series of numbers that is passed to the program. The values will be stored globally in the program. The three threads will return the three values respectively to the main program where it will be output to the user.

Pseudocode

```
1.  Create a class and initialize class members and methods.
2.  Read input values
3.  Using threads implement 3 threads methods to display messages like –GOOD
    MORNING‖, – HELLO‖, –WELCOME‖ in specified time intervals.
4.  Display output.
5.  End.
```

Problem Validation

Compile the program using the following command

Javac <Programname.java>

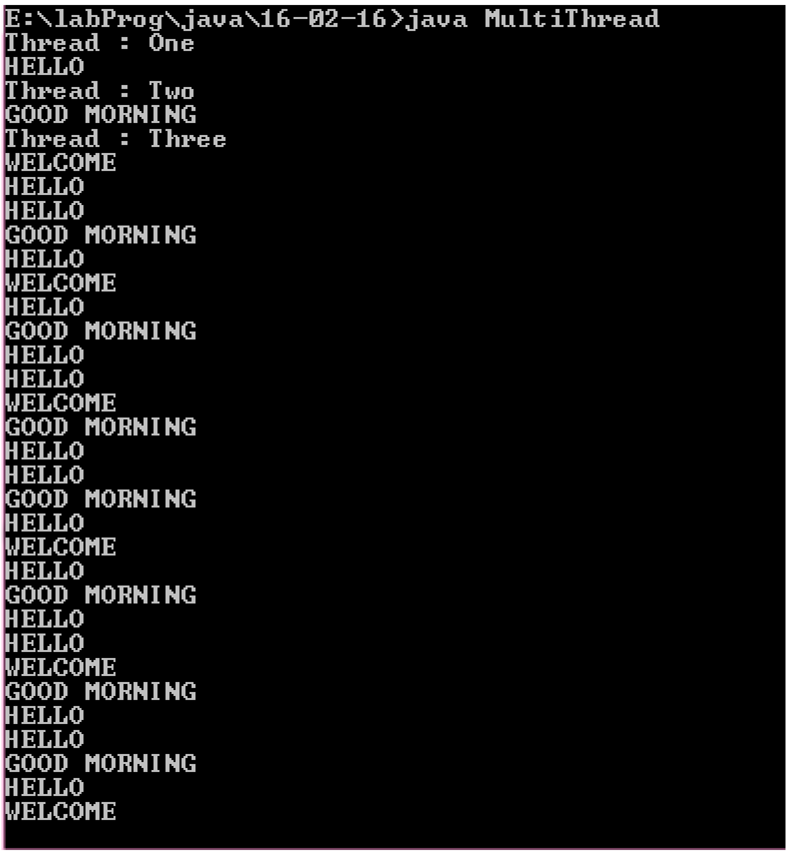
Execute the program using

Java <MainClassName>

Input:

No Input

Output:



```
E:\labProg\java\16-02-16>java MultiThread
Thread : One
HELLO
Thread : Two
GOOD MORNING
Thread : Three
WELCOME
HELLO
HELLO
GOOD MORNING
HELLO
WELCOME
HELLO
GOOD MORNING
HELLO
HELLO
WELCOME
GOOD MORNING
HELLO
HELLO
GOOD MORNING
HELLO
WELCOME
HELLO
GOOD MORNING
HELLO
HELLO
WELCOME
GOOD MORNING
HELLO
HELLO
GOOD MORNING
HELLO
WELCOME
```

Program 24

A program to illustrate the concept of Thread synchronization.

Problem Definition

A program to illustrate the concept of Thread synchronization.

Problem Description

Synchronization in java is the capability to control the access of multiple threads to any shared resource. Java Synchronization is better option where we want to allow only one thread to access the shared resource. The synchronization is mainly used to

To prevent thread interference.

To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

- Process Synchronization
- Thread Synchronization

<u>Thread Synchronization</u>

There are two types of thread synchronization mutual exclusive and inter-thread communication.

Mutual Exclusive

Synchronized method.

Synchronized block.

static synchronization.

Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

by synchronized method

by synchronized block

by static synchronization

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

Pseudocode

//example of java synchronized method

```
o  Create a class with a synchronized
o  Create one thread and pass the shared class object.
o  Create second thread and pass the shared class object.
o  Create a main tester class and run both the threads
```

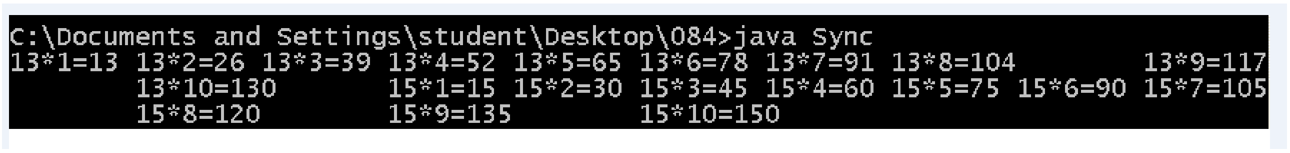
Problem Validation

Compile the program using the following command
Javac <Programname.java>
Execute the program using
Java <MainClassName>

Input:

No Input

Output:



```
C:\Documents and Settings\student\Desktop\084>java Sync
13*1=13 13*2=26 13*3=39 13*4=52 13*5=65 13*6=78 13*7=91 13*8=104        13*9=117
        13*10=130       15*1=15 15*2=30 15*3=45 15*4=60 15*5=75 15*6=90 15*7=105
        15*8=120        15*9=135        15*10=150
```

Program 25

Write a Java program to implement serialization concept

Problem Definition

Java program to implement serialization concept

Problem Description

Serialization is a process by which we can store the state of an object into any storage medium. We can store the state of the object into a file, into a database table etc. Deserialization is the opposite process of serialization where we retrieve the object back from the storage medium.

Eg1: Assume you have a Java bean object and its variables are having some values. Now you want to store this object into a file or into a database table. This can be achieved using serialization. Now you can retrieve this object again from the file or database at any point of time when you need it. This can be achieved using deserialization

Pseudocode

```
1. Create an –Emloyee‖ Class which has two member variables ―employeeNumber‖ and
–employeeName‖.

2. Then create an object of –Emloyee‖ Class, say –employee1‖ and set values for the member
variables –employeeNumber‖ and –employeeName‖.

3. Serialize this employee object –employee1‖ and store it into a file

4.Retrieve the object saved into the file
```

Problem Validation

Compile the program using the following command
Javac <Programname.java>
Execute the program using
Java <MainClassName>

Input:

No Input

Output: Objects are stored from the class into the file.

Program 26

Write a Java program that reads a file name from the user, and then displays information about whether the file exists, whether the file is readable,

Problem Definition

Reading the data from the user.

Problem Description

There is a useful class in Java called File that we can use for checking some properties of the files. What you usually have to do is to declare an object of type File whose constructor accepts a file name, and apply to it some of the methods defined for that class. Once you know the properties of the file (it exists and it can be read, for example), you declare a stream of one of the types explained in the previous sections to read from and to write to. Some methods that you can apply to a File object are:

• public boolean exists() Checks whether a file exists with the name associated to the object when it was created.

• public boolean canRead() Checks whether the program can read from the file.

• public boolean canWrite() Checks whether the program can write to the file.

• public boolean delete() Tries to delete the file and returns the result of the operation (success or fail).

• public long length() Returns the length in bytes of the file.

• public String getName() Returns the name of the file.

• public String getPath() Returns the path where the file is located. Modify the program that asks the user the temperatures of the week and that saves them in a file. Check if the file that the user indicates already exists. If so, ask the user if she/he wants to overwrite it. If her/his answer is negative, then the program ends.

Pseudocode

```
1. Start the program, import the packages.

2. create an object of _fi‘ using fileinputstream class

3. if check the file exit then

     print _file is exit‘

     else

     print _file does not exit

4. Print the number of bytes of given files is using fi.length() function
```

5. End

Problem Validation

Compile the program using the following command

Javac <Programname.java>

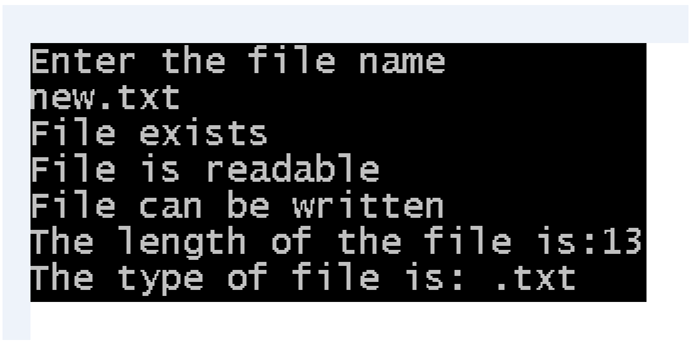
Execute the program using

Java <MainClassName>

Input:

No Input

Output:



```
Enter the file name
new.txt
File exists
File is readable
File can be written
The length of the file is:13
The type of file is: .txt
```

Program 27
Write a Java program that reads a file and displays the file on the screen, with a line number before each line.

Problem Definition

Java program that reads a file and displays the file on the screen with a line number before each line.

Problem Description

Java has a concept of working with streams of data. You can say that a Java program reads sequences of bytes from an input stream (or writes into an output stream): byte after byte, character after character, primitive after primitive. Accordingly, Java defines various types of classes supporting streams, for example InputStream or OutputStream. There are classes specifically meant for reading character streams such as Reader and Writer.

Before an application can use a data file, it must open the file. A Java application opens a file by creating an object and associating a stream of bytes with that object. Similarly, when you finish using a file, the program should close the file—that is, make it no longer available to your application.

To read file content line by line use BufferedReader object. By calling readLine() method you can get file content line by line. readLine() returns one line at each iteration, we have to iterate it till it returns null.

Pseudocode

```
1. Start the program, import the packages.
2. create an object of _fi‘ using fileinputstream class
3. compute count=0
4. repeate _4‘ until file is empty
5. if the new line is arrived then
6. printf counter,count++
7. End
```

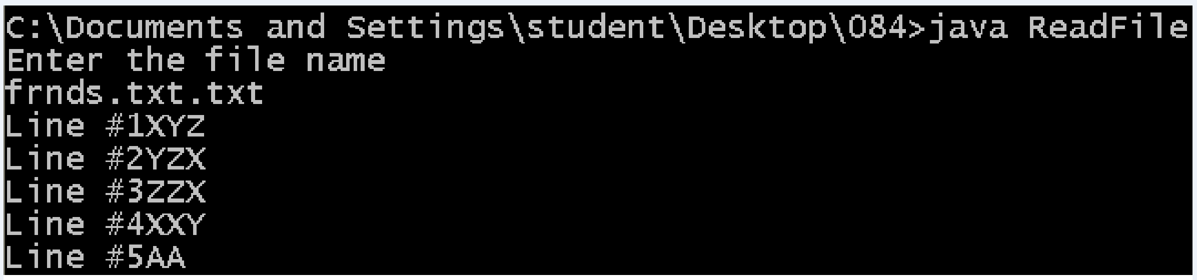
Problem Validation

Compile the program using the following command

Javac <Programname.java>
Execute the program using
Java <MainClassName>

Input:
No Input

Output:



```
C:\Documents and Settings\student\Desktop\084>java ReadFile
Enter the file name
frnds.txt.txt
Line #1XYZ
Line #2YZX
Line #3ZZX
Line #4XXY
Line #5AA
```

Program 28
Write a Java program that displays the number of characters, lines and words in a text file.

Problem Definition

Java program to display the number of characters, lines and words in a text file.

Problem Description

In Java, FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in java. Java FileOutputStream is an output stream for writing data to a file.If you have to write primitive values then use FileOutputStream.Instead, for character-oriented data, prefer FileWriter.But you can write byte-oriented as well as character-oriented data.

Pseudocode

```
1. Start the program, import the packages.

2. create an object of _fi‘ using fileinputstream class

3. compute count=0,word=0,line=1,space=0

4. repeate _4‘ until file is empty

5.  if fi.read() equal to new line  then
            increment line
        else
        fi.read() equal to space then
            increment word
        else
            increment char
6. print word,line,char

7. End
```

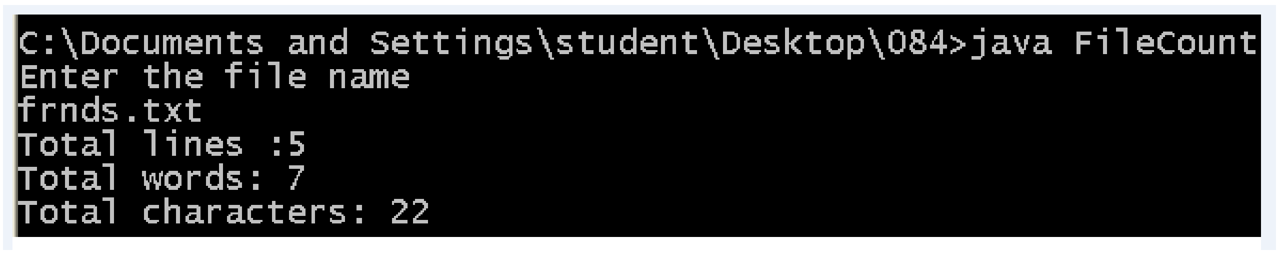
Problem Validation

Compile the program using the following command
Javac <Programname.java>
Execute the program using
Java <MainClassName>

Input:

Input is the name of a text file.

Output:



```
C:\Documents and Settings\student\Desktop\084>java FileCount
Enter the file name
frnds.txt
Total lines :5
Total words: 7
Total characters: 22
```

Program 29

Write a Java program to illustrate collection classes like (i) Array List, (ii) Iterator, (iii)Hash set.

Problem Definition

Java program to illustrate collection classes like (i) Array List, (ii) Iterator, (iii)Hash set.

Problem Description

Arraylist

Java ArrayList class uses a dynamic array for storing the elements.It extends AbstractList class and implements List interface.Java ArrayList class can contain duplicate elements.Java ArrayList class maintains insertion order.Java ArrayList class is non synchronized.Java ArrayList allows random access because array works at the index basis.In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list

Iterator

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

Commonly used methods of List Interface:

public void add(int index,Object element);

public boolean addAll(int index,Collection c);

public object get(int Index position);

public object set(int index,Object element);

public object remove(int index);

public ListIterator listIterator();

public ListIterator listIterator(int i);

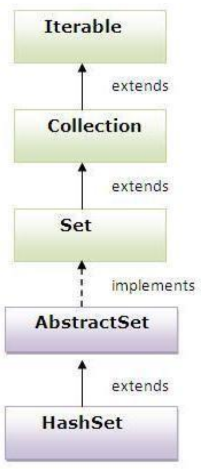
Hashset

Java HashSet class uses hashtable to store the elements.It extends AbstractSet class and implements Set interface. It contains unique elements only.

Difference between List and Set:

List can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class:



Pseudocode

c) ArrayList

```
1. import java.util.* package
2. create a class
3. ArrayList al=new ArrayList();
4. add(100) to arraylist;
5. add("abc") to arraylist;
6. add(10.84) to arraylist;
7. add(2,5) to arraylist;
8. remove(10.84) from araylist;
9. print the "Array list‖
10. end
```

d) Iterator

```
    1. import java.util.* package
    2. create a class
    3. ArrayList al=new ArrayList();
    4. add(100) to arraylist;
    5. add("abc") to arraylist;
    6. add(10.84) to arraylist;
    7. add(2,5) to arraylist;
    8. remove(10.84) from arraylist;
    9. use Iterator it=al.iterator() to move ahead in the list;
    10. print "List";
    11. use ListIterator lit=al.listIterator() to move ahead in list;
    12. move until no element found (lit.hasNext())
    13. Add –+‖ to the existing element using set() method;
    14. Print "Modified List: "
    15. Print "Modified list backwards";
    16. Move until first element using hasPrevious() method
    17. Print the element
    18. end
```

e) Hashset

```
1. import java.util.* package
2. create a class
```

```
3. Create HashSet hs=new HashSet();
4. Add (244) to hashset;
5. Add (21) to hashset;
6. Add ("Xyz") to hashset;
7. remove(21) from hashset;
8. print the Hashset  hs;
9. end
```

Problem Validation

Compile the program using the following command

Javac <Programname.java>

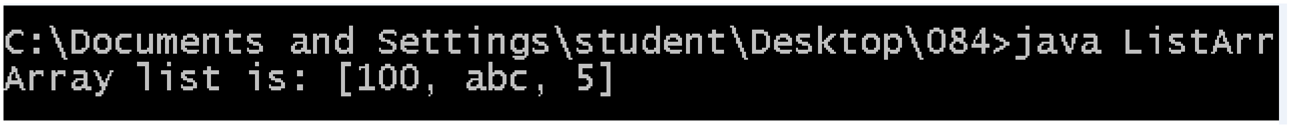
Execute the program using

Java <MainClassName>

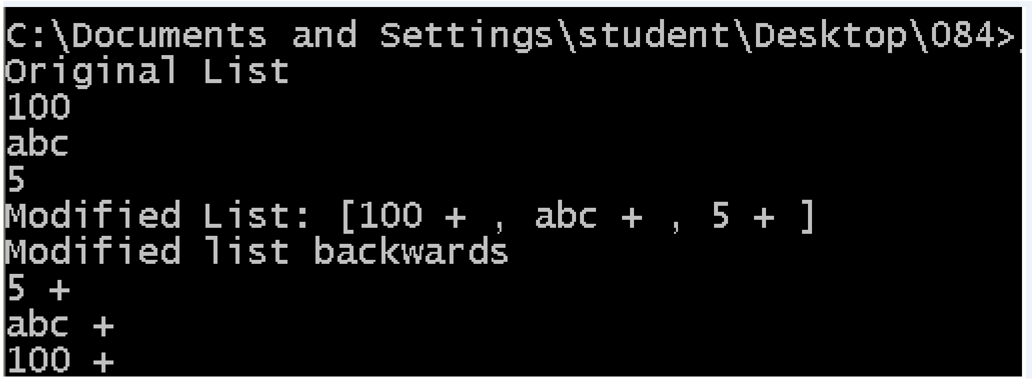
Input:

No Input

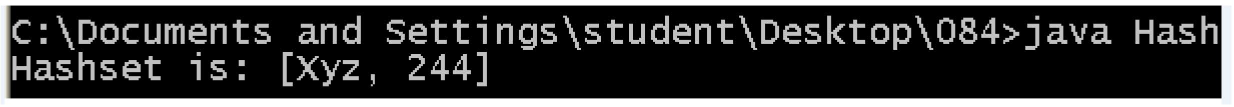
Output:



```
C:\Documents and Settings\student\Desktop\084>java ListArr
Array list is: [100, abc, 5]
```



```
C:\Documents and Settings\student\Desktop\084>
Original List
100
abc
5
Modified List: [100 + , abc + , 5 + ]
Modified list backwards
5 +
abc +
100 +
```



```
C:\Documents and Settings\student\Desktop\084>java Hash
Hashset is: [Xyz, 244]
```

Program 30
Write a Java program for handling Key events

Problem Definition

Java program to handle keyboard events.

Problem Description

On entering the character the Key event is generated.There are three types of key events which are represented by the integer constants. These key events are following

KEY_PRESSED

KEY_RELASED

KEY_TYPED

The class which processes the KeyEvent should implement this interface. The object of that class must be registered with a component. The object can be registered using the addKeyListener() method.

Pseudocode

```
o   Import the packages of applet, awt, awt.event.
o   Create a classes, methods.
o   Keyboard events like keyTyped, key Pressed, keyReleased.
o   g.drawString() application of Graphical User Interface.
o   The keyboard event arguments execution.
o   Printing in the separated Applet viewer window.
o   End
```

Problem Validation

Compile the program using the following command

Javac <Filename.java>

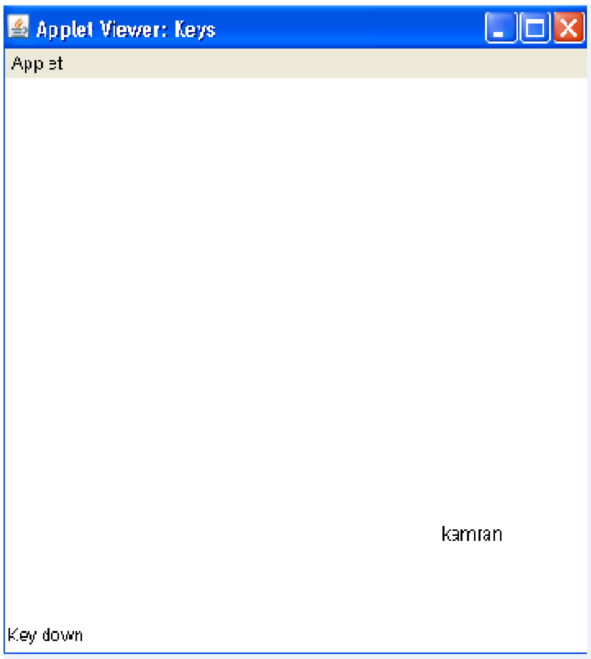
Execute the program using

appletviewer <Filename.java>

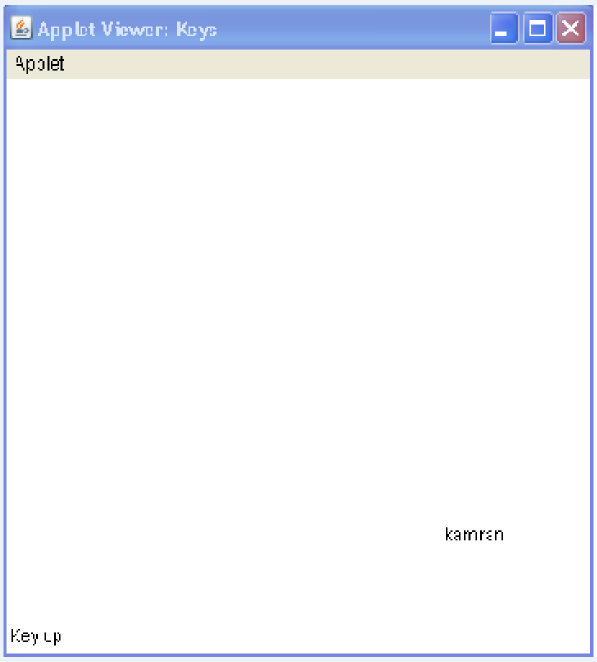
Input:

Type keys like up arrow and down arrow or any character or number

Output:



```
kamran
Key down
```



```
kamran
Key up
```

Program 31

Write a Java program for handling Mouse events

Problem Definition

Develop a Java Program for handling Mouse Events.

Problem Description

Pseudocode

```
1. Import the packages of applet, awt, awt.event.

2. Create a classes, methods.

3. Mouse moments, mouse Clicked, mouse Pressed, mouse Released, mouse Entered,
   mouse Exited, mouse Dragged events args.

4. g.drawString() application of Graphical User Interface.

5. while rotating mouse event args.

6. The mouse event arguments execution.

7. Printing in the separated Applet viewer window.

8. End
```

Problem Validation

Compile the program using the following command

Javac <Filename.java>

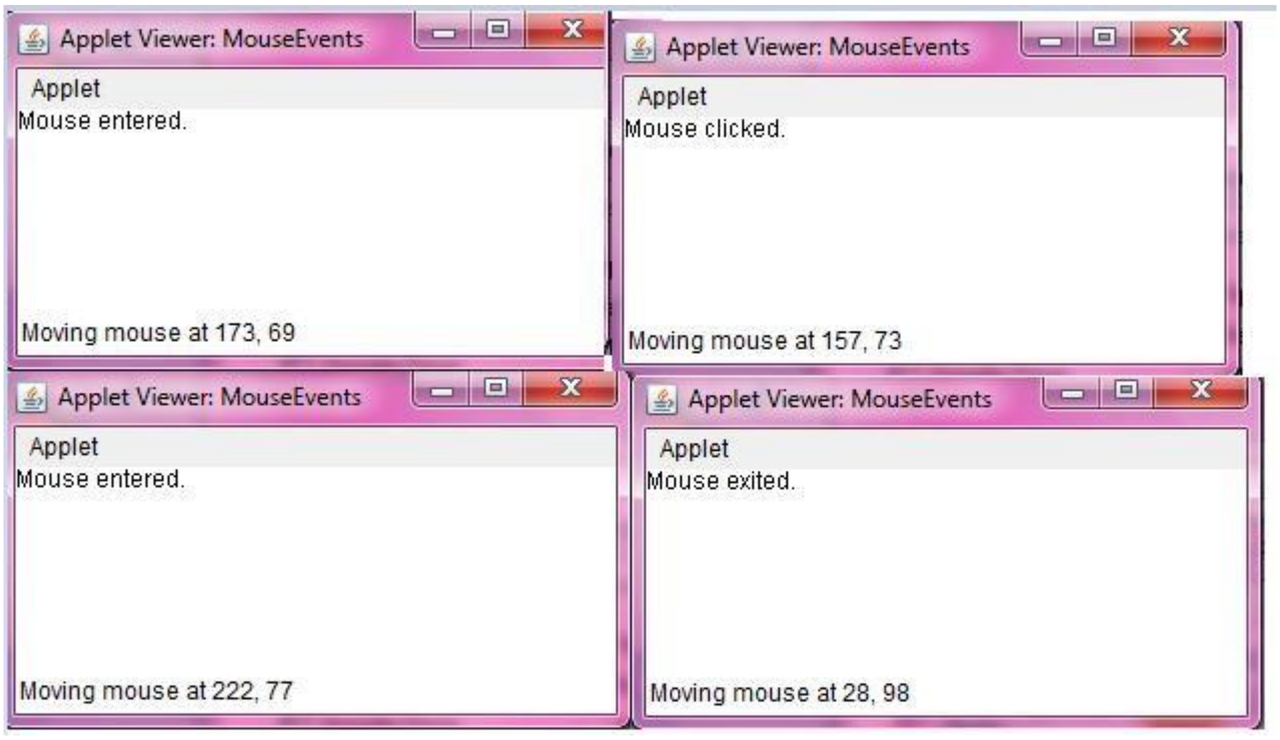
Execute the program using

appletviewer <Filename.java>

Input:

No Input

Output:


Applet Viewer: MouseEvents
Applet
Applet Viewer: MouseEvents
Applet
Applet Viewer: MouseEvents
Applet
Applet Viewer: MouseEvents
Applet

```
Mouse entered.

Moving mouse at 173, 69
```

```
Mouse clicked.

Moving mouse at 157, 73
```

```
Mouse entered.

Moving mouse at 222, 77
```

```
Mouse exited.

Moving mouse at 28, 98
```

Program 32

Develop an applet that displays a simple message

Problem Definition

Develop an applet that displays a simple message

Problem Description

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules.

Pseudocode

```
1. import java.applet.Applet, java.awt.* packages
2. Include the /*<Applet code="Simple" height=500 width=500>  </Applet>*/
3. Create class Simple extends Applet
4. Declare void paint(Graphics g)
5. g.drawString("Hello World",10,20);
6. End
```

Problem Validation

Compile the program using the following command

Javac <Filename.java>

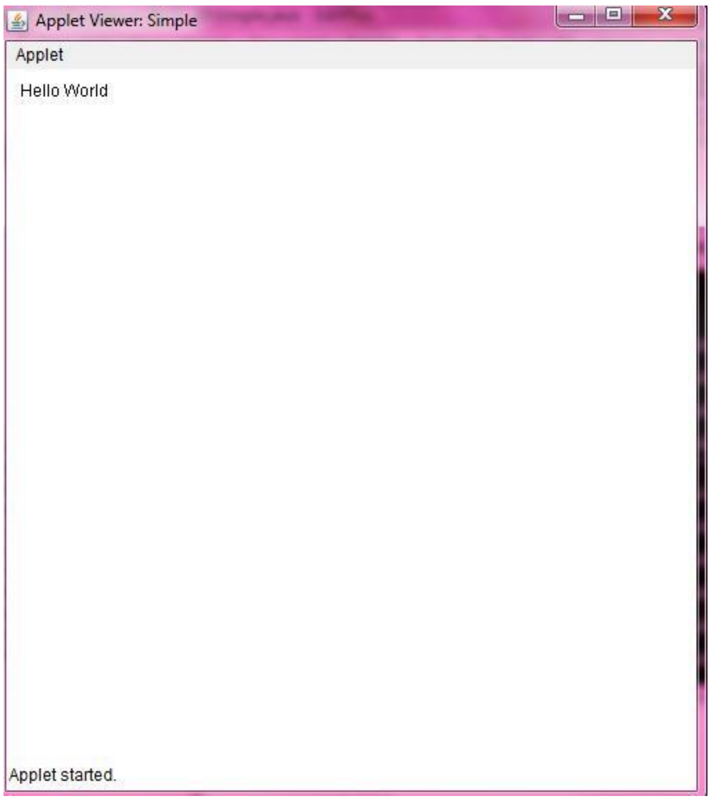
Execute the program using

appletviewer <Filename.java>

Input:

No Input

Output:



```
Hello World
```

```
Applet started.
```

Program 33
Develop an applet that displays lines, rectangles, ovals, square etc.

Problem Definition

Develop an applet that displays lines, rectangles, ovals, square etc.

Problem Description

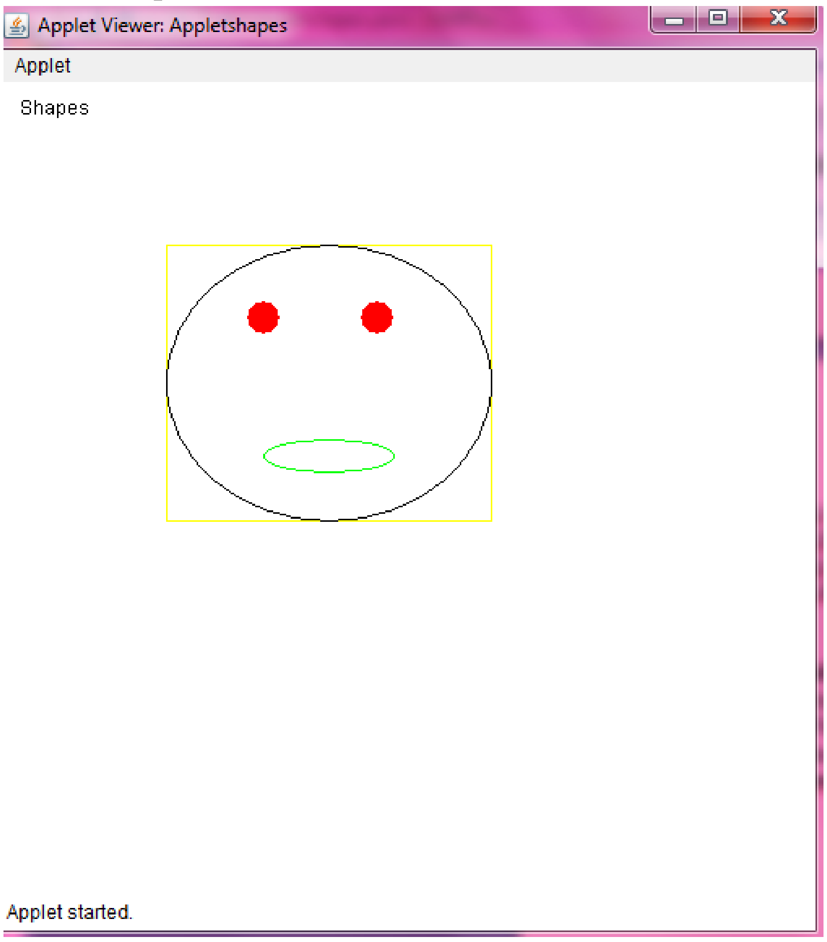
Pseudocode

```
1. Import the packages of applet,awt,awt.event.

2. Create a classes: public void paint(Graphics g).

3. Assume the values of string, color and shape.

4. g.drawString() application of Graphical User Interface.

5. Printing in the separated Applet viewer window.

6. End
```

Problem Validation

Compile the program using the following command
Javac <Filename.java>
Execute the program using
appletviewer <Filename.java>
Input:
No Input

Output:



Program 34
Java program to illustrate GUI Components using AWT.

Problem Definition

Java applet to demonstrate the various GUI components in AWT package.

Problem Description

Java Graphics APIs - AWT and Swing - provide a huge set of reusable GUI components, such as button, text field, label, choice, panel and frame for building GUI applications.

AWT is huge! It consists of 12 packages (Swing is even bigger, with 18 packages as of JDK 1.8). Fortunately, only 2 packages - java.awt and java.awt.event - are commonly-used.

The java.awt package contains the core AWT graphics classes:

- GUI Component classes (such as Button, TextField, and Label),
- GUI Container classes (such as Frame, Panel, Dialog and ScrollPane),
- Layout managers (such as FlowLayout, BorderLayout and GridLayout),
- Custom graphics classes (such as Graphics, Color and Font).

The java.awt.event package supports event handling:

- Event classes (such as ActionEvent, MouseEvent, KeyEvent and WindowEvent),
- Event Listener Interfaces (such as ActionListener, MouseListener, KeyListener and WindowListener),

Pseudocode

```
<import relevant packages>
Add the applet code
/* <applet code="" height=600 width=600>
</applet> */
Declare class GuiWindow1 extends Applet implements ActionListener,ItemListener
{
        Declare msg,s1,s2,s3,s4,s5,s6 as String;
        Declare ent,cnl as Button;
        Declare fname,lname as TextField;
        Declare addr as TextArea;
        Declare n1,l1,a1,gend,qal,jbl as Label;
        Declare c1,c2 as Checkbox;
        Declare cbg as CheckboxGroup;
        Declare quali,job as Choice;
        public void init()
        {
                setLayout(new FlowLayout(FlowLayout.LEFT,40,20));
                msg=" ";
                s1="";
```

```
s2="";
s3="";
s4="";
s5="";
s6="";
create Label("First Name:",Label.RIGHT);
add  TextField(12);

create Label("Last Name:",Label.RIGHT);
add TextField(12);

create new Label("Address:",Label.RIGHT);
add TextArea(5,20);


create Label("Gender:",Label.RIGHT);
add(gend);
create CheckboxGroup();
create Checkbox("MALE",true,cbg);
create Checkbox("FEMALE",false,cbg);
add(c1);
add(c2);
c1.addItemListener(this);
c2.addItemListener(this);
create Label("Qualification:",Label.RIGHT);
add(qal);
quali=new Choice();
quali.add("SSC");
quali.add("Inter");
quali.add("Under Graduate");
quali.add("Graduate");
quali.add("Post Graduate");
add(quali);
quali.addItemListener(this);
create Label("Occupation:",Label.RIGHT);
add(jbl);
job=new Choice();
job.add("Engineer(Student)");
job.add("Software Engineer");
job.add("Lawyer");
job.add("Medical Profestional");
job.add("Self Employed");
job.add("Lecturer");
job.add("Others");
add(job);
job.addItemListener(this);
create Button("Enter");
add(ent);
ent.addActionListener(this);
create Button("Cancel");
```

```
                add(cnl);
                cnl.addActionListener(this);
        }
        public void itemStateChanged(ItemEvent ie)
        {
                repaint();
        }
        public void actionPerformed(ActionEvent ae)
        {
                repaint();
                String arg=ae.getActionCommand();
                if(arg.equals("Enter"))

                        s1+=fname.getText();
                        s2+=lname.getText();
                        s3+=addr.getText();
                        s4+=cbg.getSelectedCheckbox().getLabel();
                        s5+=quali.getSelectedItem();
                        s6+=job.getSelectedItem();

                else

                        fname.setText("");
                        lname.setText("");
                        addr.setText("");
                        s1=s2=s3=s4=s5=s6="";
                        cbg.setSelectedCheckbox(c1);

        }
        public void paint(Graphics g)
        {
                Display (msg);
                Display ("First Name:"+s1,20,410);
                Display ("Last Name:"+s2,20,430);
                Display ("Address:"+s3,20,450);
                Display ("Gender:"+s4,20,470);
                Display ("Qualification:"+s6,20,510);
                using  g.drawString
        }
End the class
```

Problem Validation

Compile the program using the following command

Javac <Filename.java>

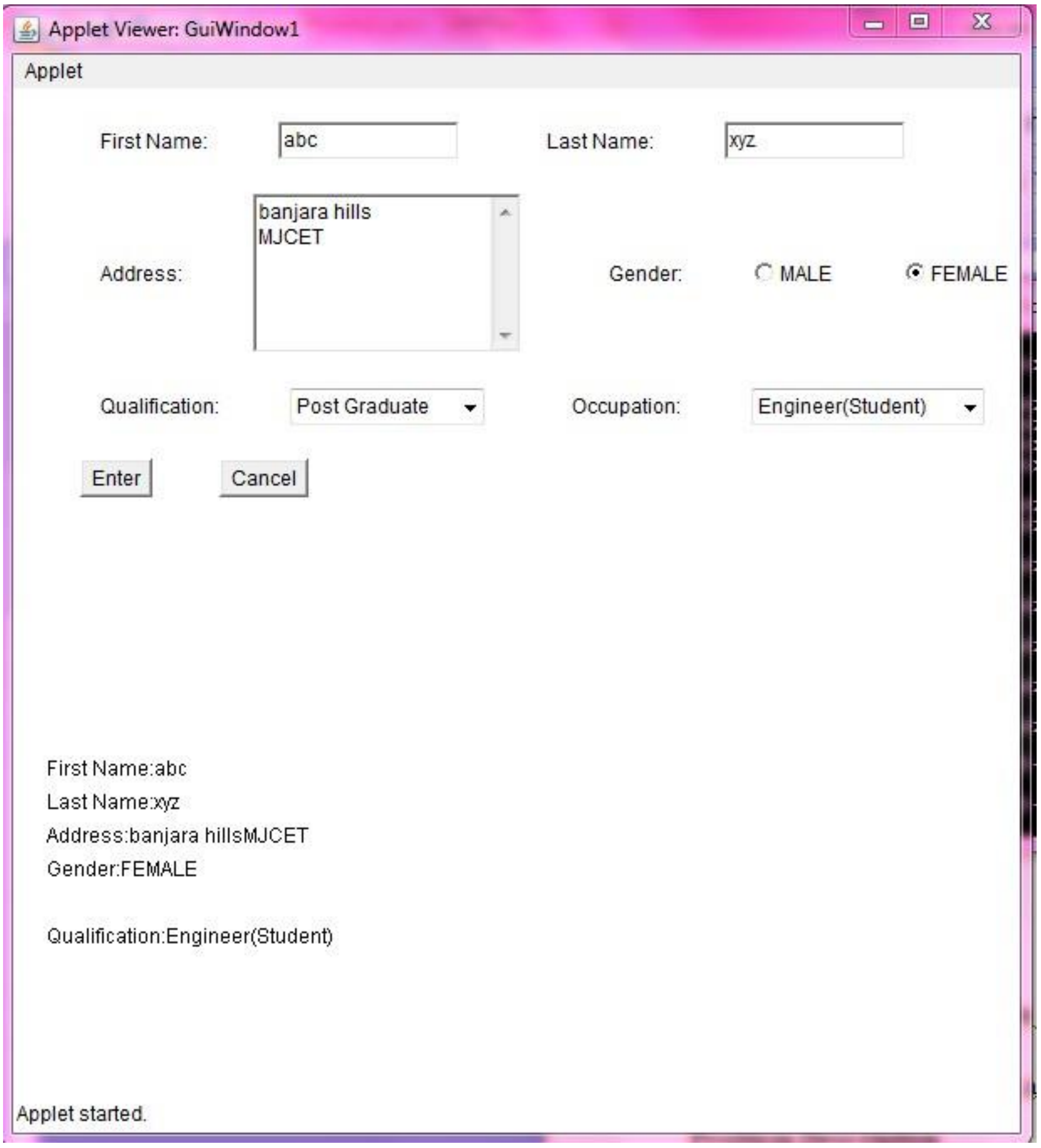
Execute the program using

appletviewer <Filename.java>

Input:

Input is given in the form at the GUI window

Output:



First Name: abc

Last Name: xyz

Address: banjara hills
MJCET

Gender: ( ) MALE [✓ FEMALE]

Qualification: Post Graduate

Occupation: Engineer(Student)

```
First Name:abc
Last Name:xyz
Address:banjara hillsMJCET
Gender:FEMALE

Qualification:Engineer(Student)
```

Applet started.

Program 35
Java program to change a specific character in a file.

Problem Definition
Java program to change a specific character in a file with a new character.

Problem Description
This method replaces each substring of this string that matches the given regular expression with the given replacement.

String replace(char oldChar, char newChar)

Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

String replaceAll(String regex, String replacement

Replaces each substring of this string that matches the given regular expression with the given replacement.

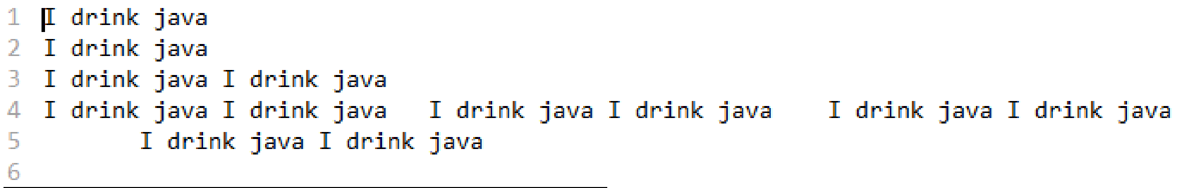
Pseudocode

```
1. <import relevant packages>
2. Declare a class
3. Read a file name
4. Read every line of a file
5. Read the existing character in the file using Scanner object
6. Read the new character to be replaced using Scanner object
7. Use the method
8. replaceAll(old_char,new_char);
9. write the data to the file using FileWriter
10. End
```

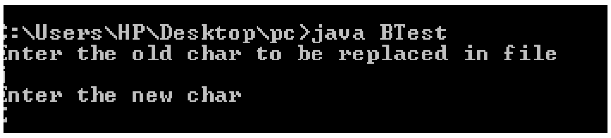
Problem Validation
Compile the program using the following command
Javac <Filename.java>
Execute the program using
Java <main_Class_Name>
Input:

Preexisting character in the file.
Character to be replaced

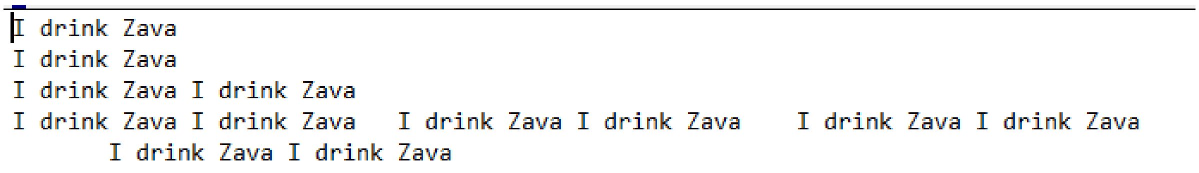
Output:



```
I drink java
 I drink java
 I drink java I drink java
 I drink java I drink java   I drink java I drink java    I drink java I drink java
        I drink java I drink java

```



```
:\Users\HP\Desktop\pc>java BTest
nter the old char to be replaced in file

nter the new char

```



```
I drink Zava
I drink Zava
I drink Zava I drink Zava
I drink Zava I drink Zava   I drink Zava I drink Zava    I drink Zava I drink Zava
       I drink Zava I drink Zava
```

Program 36
Java program to implements producer consumer problem

Problem Definition

Java program to implements producer consumer problem using the concept of inter thread communication.

Problem Description

Multithreading replaces event loop programming by dividing tasks into discrete, logical units. Threads also provide a secondary benefit: they do away with polling. Polling is usually implemented by a loop that is used to check some condition repeatedly. Once the condition is true, appropriate action is taken. This wastes CPU time for example, consider the classic queuing problem, where one thread is producing some data and another is consuming it. To make the problem more interesting, suppose that the producer has to wait until the consumer is finished before it generates more data. In a polling system, the consumer would waste many CPU cycles while it waited for the producer to produce. Once the producer was finished, it would start polling, wasting more CPU cycles waiting for the consumer to finish, and so on. Clearly, this situation is undesirable.
To avoid polling, Java includes an elegant interprocess communication mechanism via the methods:

- wait( ) tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify( ).
- notify( ) wakes up a thread that called wait( ) on the same object.
- notifyAll( ) wakes up all the threads that called wait( ) on the same object. One of the threads will be granted access.

Pseudocode

1. Create a class and initialize class members and methods.
2. Read input values
3. Implement 2 different threads for allocating the resources and consuming the resources.
4. Provide inter thread communication between them to resolve the problem.
5. Display output.
6. End.

```
int itemCount = 0;

procedure producer() {
   while (true) {
      item = produceItem();

      if (itemCount == BUFFER_SIZE) {
         sleep();
      }
```

```
        putItemIntoBuffer(item);
        itemCount = itemCount + 1;

        if (itemCount == 1) {
            wakeup(consumer);
        }
    }
}

procedure consumer() {
    while (true) {

        if (itemCount == 0) {
            sleep();
        }

        item = removeItemFromBuffer();
        itemCount = itemCount - 1;

        if (itemCount == BUFFER_SIZE - 1) {
            wakeup(producer);
        }

        consumeItem(item);
    }
}
```

Problem Validation

Compile the program using the following command

Javac <Filename.java>

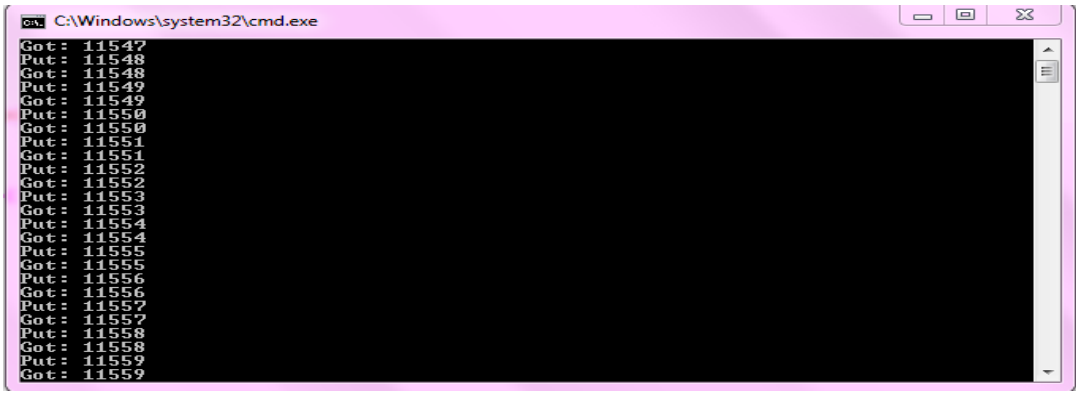
Execute the program using

Java <main_Class_Name>

Input:

No Input

Output:



```
Got: 11547
Put: 11548
Got: 11548
Put: 11549
Got: 11549
Put: 11550
Got: 11550
Put: 11551
Got: 11551
Put: 11552
Got: 11552
Put: 11553
Got: 11553
Put: 11554
Got: 11554
Put: 11555
Got: 11555
Put: 11556
Got: 11556
Put: 11557
Got: 11557
Put: 11558
Got: 11558
Put: 11559
Got: 11559
```

Annexure – I

List of programs according to O.U. curriculum

Code: BIT 282 **JAVA PROGRAMMING LAB**

Instruction week	3	Periods per
Duration of University Examination	3	Hours
University Examination	50	Marks
Sessional	25	Marks

- Write a Java Program that reads a line of integers, and then displays each integer, and the sum of all the integers (Use String Tokenizer class of java. util)
- Write a Java program to illustrate the concept of class with method overloading.
- Write a Java program to illustrate the concept of Single level and Multi level Inheritance.
- Write a Java program to illustrate the concept of Dynamic Polymorphism.
- Write a Java program to demonstrate the Interfaces & Abstract Classes.
- Write a Java program to implement the concept of exception handling.
- Write a Java program to illustrate the concept of threading using Thread Class and runnable Interface.
- Write a Java program to illustrate the concept of multi-threading that creates three threads. First thread displays ―Good Morning‖ every one second, the second thread displays ―Hello‖ every two seconds and the third thread displays ―Welcome‖ every three seconds.
- Write a Java program to implement serialization concept
- Write a Java program to illustrate the concept of Thread synchronization.
- Write a Java program that correctly implements producer consumer problem using the concept of inter thread communication.
- Write a Java program that reads a file name from the user, and then displays information about whether the file exists, whether the file is readable, whether the file is writable, the type of file and the length of the file in bytes.
- Write a Java program that reads a file and displays the file on the screen, with a line number before each line.
- Write a Java program that displays the number of characters, lines and words in a text file.
- Write a Java program to change a specific character in a file.
- Note: Filename, number of the byte in the file to be changed and the new character are specified on the command line.
- Write a Java program to illustrate collection classes like Array List, Iterator, Hash map etc.
- Write a Java program for handling mouse & key events.

- A program to illustrate the concept of I/O Streams
- Write a Java program that works as a simple calculator. Use a grid layout to arrange buttons for the digits and for the +, -,*, % operations. Add a text field to display the result.

Suggested Reading:

1. Herbert Scheldt, –The Complete Reference Java, 7th Edition, Tata McGraw Hill, 2006.
2. James M Slack, Programming and Problem Solving with JAVA, Thomson Learning, 2002.
3. C Thomas Wu, An Introduction to Object Oriented Programming with Java 5th Edition, McGraw Hill Publishing, 2010.
4. H. M. Dietel and P. J. Dietel, Java How to Program, Sixth Edition, Pearson Education / PHI