What is 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.

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suite various types of platforms. Ex: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**

Java is:

* **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
* **Platform independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
* **Simple:** Java is designed to bke easy to learn. If you understand the basic concept of OOP Java would be easy to master.
* **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
* **Architectural-neutral:** Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
* **Portable:** Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary which is a POSIX subset.
* **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
* **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.
* **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.
* **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.
* **Distributed:** Java is designed for the distributed environment of the internet.
* **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

History of Java:

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.

**Sun** released the first public implementation as Java 1.0 in **1995**. It promised **Write Once, Run Anywhere**(WORA), providing no-cost run-times on popular platforms.

On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

# Process flow of execution

In the Java programming language, all source code is first written in plain text files ending with the .java extension.

Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine[1](https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html#FOOT) (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.

A diagram of a diagram of a program

Description automatically generated

An overview of the software development process.

Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS

A diagram of a computer program

Description automatically generated

Through the Java VM, the same application is capable of running on multiple platforms.

What is JVM?

JVM stands for ***Java Virtual Machine***. It is an abstract computing machine that is **responsible for executing Java programs**. When you write a Java program, the source code is compiled into byte code which is understandable by the JVM. Upon execution, the JVM translates byte code into machine code of the target operating system.

The JVM is the cornerstone of the Java programming language. It is responsible for the very well-known feature of Java: cross-platform. That means you can write a Java program once and run it anywhere: Windows, Linux, Mac and Solaris, as long as JRE is installed on the host operating system.

Every time you run a Java program, the JVM is started to execute and manage the program’s execution. The JVM is running in two modes: client (default) and server.

***An Oracle’s implementation for JVM is called Java HotSpot VM***

What is JRE?

JRE stands for ***Java Runtime Environment***. It provides the **libraries**, JVM and other components necessary for you to run applets and applications written in the Java programming language.

The JRE contains standard tools such as java, keytool, policytoo, but it doesn’t contain compilers or debuggers for developing applets and applications.

When you deploy your Java applications on client’s computer, the client needs a JRE to be installed.

What is JDK?

JDK stands for ***Java Development Kit***. It’s a superset of JRE. The JDK includes the JRE plus command-line development tools such as compilers (javac) and debuggers (jdb) and others (jar, javadoc, etc) that are necessary or useful for developing applets and applications.

Therefore, as a Java programmer, you have to install JDK as a minimum requirement for theh development environment.

Summary:

* **JVM** = JVM is the Virtual Machine that runs Java applications. The JVM makes Java platform independence
* **JRE** = JVM + standard libraries: provides environment for executing Java applications ▪ **JDK** = JRE + development tools for compiling and debugging Java applications

Tips:

* You should have both JRE and JDK installations (setup) on your computer. You will need both during the development process.
* You should have multiple versions of JDK and JRE installed: 1.5, 1.6, 1.7 and 1.8 for different testing purposes in the future.
* You should install both 32-bit and 64-bit versions.
* When installing the JDK, remember to check ‘Install Demos and Samples’. Then you can explore various interesting examples in the **demo** directory under JDK’s installation path.
* Only the JDK includes source code of the Java runtime libraries. You can discover the source code in the **src.zip** file which can be found under JDK’s installation directory.

First Java Program:

public class MyFirstJavaProgram {

/\* This is my first java program.

\* This will print 'Hello World' as the output

\*/

public static void main(String []args) {

System.out.println("Hello World"); // prints Hello World

}

}

Let's look at how to save the file, compile and run the program. Please follow the steps given below:

1. Open notepad and add the code as above.
2. Save the file as: MyFirstJavaProgram.java.
3. Open a command prompt window and go to the directory where you saved the class. Assume it's C:\.
4. Type ' javac MyFirstJavaProgram.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption : The path variable is set).
5. Now, type ' java MyFirstJavaProgram ' to run your program.
6. You will be able to see ' Hello World ' printed on the window.

C:\> javac MyFirstJavaProgram.java

C:\> java MyFirstJavaProgram

Hello World

Basic Syntax:

About Java programs, it is very important to keep in mind the following points.

* **Case Sensitivity -** Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
* **Class Names -** For all class names the first letter should be in Upper Case.

If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Example *class MyFirstJavaClass*

* **Method Names -** All method names should start with a Lower Case letter.

If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

Example *public void myMethodName()*

* **Program File Name -** Name of the program file should exactly match the class name. When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match your program will not compile). Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*
* **public static void main(String args[]) -** Java program processing starts from the main() method which is a mandatory part of every Java program.

Java Identifiers:

All Java components require names. Names used for classes, variables and methods are called identifiers.

In Java, there are several points to remember about identifiers. They are as follows:

* All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (\_).
* After the first character identifiers, can have any combination of characters.
* A keyword cannot be used as an identifier.
* Most importantly identifiers are case sensitive.
* Examples of legal identifiers: age, $salary, \_value, \_\_1\_value
* Examples of illegal identifiers: 123abc, -salary

Java Keywords:

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | Assert | boolean | break |
| byte | case | catch | char |
| class | const | continue | default |
| do | double | else | enum |
| extends | Final | finally | float |
| for | Goto | if | implements |
| import | instanceof | int | interface |
| long | native | new | package |
| private | protected | public | return |
| short | Static | strictfp | super |
| switch | synchronized | this | throw |
| throws | transient | try | void |
| volatile | While |  |  |

# Comments in Java

Java supports single-line and multi-line comments very similar to c and c++. All characters available inside any comment are ignored by Java compiler.

public class MyFirstJavaProgram{

/\* This is my first java program.

\* This will print 'Hello World' as the output \* This is an example of multi-line comments. \*/

public static void main(String []args){

// This is an example of single line comment

/\* This is also an

example of single line comment. \*/

System.out.println("Hello World");

}

}

# Datatypes

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java:

* Primitive Data Types
* Reference/Object Data Types

Primitive Data Types:

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword. Let us now look into detail about the eight primitive data types.

The Java language has 8 primitive types: **boolean**, **byte**, **char**,**double**, **float**, **int**, **long**, and **short**.

A **boolean** type represents either *true* or *false* value.

A **char** type represents a single character, such as 'a', 'B', 'c', ...Actually char type is 16-bit integer number (un-signed).

The others are numeric types. The following table lists the primitive types which represent numbers in the Java language (the char type is also included because it is actually a number type):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Type** | **Bits** | **Bytes** | **Minimum value** | **Maximum value** |
| ***Integer numbers*** | **byte** | 8 | 1 | **-128** (-27) | **127** (27-1) |
| **char** | 16 | 2 | **0** | **65,535** |
| **short** | 16 | 2 | **-32,768** (-215) | **32,767** (215-1) |
| **int** | 32 | 4 | **-2,147,483,648** (-231) | **2,147,483,647** (231-1) |
| **long** | 64 | 8 | approx. -9,2 billions of billion (-  263) | approx. 9,2 billions of billion (-2631) |
| ***Floating point***  ***numbers*** | **float** | 32 | 4 | N/A | N/A |
| **Double** | 64 | 8 | N/A | N/A |

Reference Data Types:

* Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
* Class objects, and various type of array variables come under reference data type.
* Default value of any reference variable is null.
* A reference variable can be used to refer to any object of the declared type or any compatible type.
* Example: Animal animal = new Animal("giraffe");

# Variable Types

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. The basic form of a variable declaration is shown here:

data type variable [ = value][, variable [= value] ...] ;

Here *data type* is one of Java's datatypes and *variable* is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java: int a, b, c; // Declares three ints, a, b, and c.

int a = 10, b = 10; // Example of initialization byte B = 22; // initializes a byte type variable B. double pi = 3.14159; // declares and assigns a value of PI.

char a = 'a'; // the char variable a iis initialized with value 'a'

This chapter will explain various variable types available in Java Language. There are three kinds of variables in Java:

1. Local variables
2. Instance variables
3. Class/static variables

Local variables:

* Local variables are declared in methods, constructors, or blocks.
* Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
* Access modifiers cannot be used for local variables.
* Local variables are visible only within the declared method, constructor or block.
* Local variables are implemented at stack level internally.
* There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

Example:

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to this method only.

public class Test{ public void pupAge(){ int age = 0; age = age + 7;

System.out.println("Puppy age is : " + age);

}

public static void main(String args[]){ Test test = new Test(); test.pupAge();

} }

|  |
| --- |
| **Local Variable Demo 1** |
| public class LocalVarDemo\_2  {  public static void main(String[] args) { |

Program N0. 2

|  |  |
| --- | --- |
| }  } | int age=21; //local variables  System.out.println("\n\t Value of age = " + age); |

|  |
| --- |
| **Local Variable Demo 2** |
| public class LocalVarDemo\_3  {  public static void main(String[] args)  {  char ch; float avg; String str;    ch = 'A'; avg = 45.67f;  str = "Welcome";    System.out.println("\n\t Value of ch = " + ch);  System.out.println("\n\t Value of avg = " + avg); System.out.println("\n\t Value of str = " + str); }  } |

Program N0. 3

|  |
| --- |
| **Local Variable Demo 3** |
| public class LocalVarDemo\_4  {  static void showvalue()  {  int age = 32; //local variable  System.out.println("\n\t Value of age = " + age);  }  public static void main(String[] args)  {  showvalue();  }  } |

Program N0. 4

Instance variables:

* Instance variables are declared in a class, but outside a method, constructor or any block.
* When a space is allocated for an object in the heap, a slot for each instance variable value is created.
* Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
* Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
* Instance variables can be declared in class level before or after use.
* Access modifiers can be given for instance variables.
* The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However visibility for subclasses can be given for these variables with the use of access modifiers.
* Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor.
* Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class ( when instance variables are given accessibility) should be called using the fully qualified name . *ObjectReference.VariableName*.

Example:

import java.io.\*;

public class Employee{

// this instance variable is visible for any child class. public String name;

// salary variable is visible in Employee class only. private double salary;

// The name variable is assigned in the constructor. public Employee (String empName){ name = empName;

}

// The salary variable is assigned a value.

public void setSalary(double empSal){ salary = empSal;

}

// This method prints the employee details. public void printEmp(){

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

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

}

public static void main(String args[]){ Employee empOne = new Employee("Ransika"); empOne.setSalary(1000); empOne.printEmp();

} }

|  |
| --- |
| **Instance Variable Class Student** |
| class Student  {  public int roll; // instance varialbe  String name; // data members / fields    public void showStud()  {  System.out.println("\n\t Roll No :" + roll);  System.out.println("\n\t Name :" + name);  }  }  public class InstanceVarDemo\_5  {  public static void main(String[] args)  {  Student stud = new Student();  stud.roll = 101; stud.name = "Chetan";  stud.showStud();  }  } |

Program N0. 5

|  |
| --- |
| **Instance Variable Default Values** |
| class Demo  {  int i; byte b; |

Program N0. 6

|  |
| --- |
| String str; boolean bl;  }  public class InstanceVarDemo\_6  {  public static void main(String[] args)  {  Demo d1 = new Demo();  System.out.println("Default value of int i = " + d1.i);  System.out.println("Default value of byte b = " + d1.b);  System.out.println("Default value of String str = " + d1.str); System.out.println("Default value of boolean bl = " + d1.bl); }  } |

|  |
| --- |
| **Class marketing dept** |
| class MktDept  { int empid; int sal;    static String deptName;  }  public class StaticVarDemo\_7  {  public static void main(String[] args)  {  //MktDept m1 = new MktDept();  //m1.empid = 101;    MktDept.deptName = "Marketing Department";  System.out.println("\n\t Dept name = " + MktDept.deptName);  }  } |

Program N0. 7

# Assignment

|  |
| --- |
| **Class Product** |
|  |

Program N0. 8

Class/static variables:

* Class variables also known as static variables are declared with the ***static*** keyword in a class, but outside a method, constructor or a block.
* There would only be one copy of each class variable per class, regardless of how many objects are created from it.
* Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final and static. Constant variables never change from their initial value.
* Static variables are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.
* Static variables are created when the program starts and destroyed when the program stops.
* Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
* Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
* Static variables can be accessed by calling with the class name*ClassName.VariableName*.
* When declaring class variables as public static final, then variables names (constants) are all in upper case. If the static variables are not public and final the naming syntax is the same as instance and local variables.

Example:

public class Employee{

// salary variable is a private static variable private static double salary; // DEPARTMENT is a constant public static final String DEPARTMENT = "Development "; public static void main(String args[]){ salary = 1000;

System.out.println(DEPARTMENT + "average salary:" + salary);

}

}

**Note:** If the variables are access from an outside class the constant should be accessed as Employee.DEPARTMENT

# Java Modifier

* Modifiers are keywords that you add to those definitions to change their meanings.
* The Java language has a wide variety of modifiers, including the following: o Java Access Modifiers o Non Access Modifiers

# Java Access Modifiers

Default Access Modifier

* Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
* A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default

public.

|  |
| --- |
| **Class Student** |
| class Student  {  int sid; //default members  String sname;    void showStud()  {  System.out.println("\n\t Roll No :" + sid);  System.out.println("\n\t Name :" + sname);  } }  public class DefaultAccessDemo\_9  {  public static void main(String[] args)  {  Student s1 = new Student();  s1.sid = 101; |

Program N0. 9

|  |  |  |
| --- | --- | --- |
|  |  | s1.sname = "Prakash"; s1.showStud(); |
| } | } |  |

## Public Access Modifier

* A class, method, constructor, interface etc declared public can be accessed from any other class.
* Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

|  |
| --- |
| **Class Employee** |
| class Employee  { public int empid; public String ename;    private void showEmp()  {  System.out.println("\n\t Empid :" + empid);  System.out.println("\n\t Emp Name :" + ename); }  }  public class PublicModifierDemo  {  public static void main(String[] args)  {  Employee e1 = new Employee();  e1.empid = 101; e1.ename = "Sneha";  e1.showEmp();  }  } |

**Program N0.**

**10**

Private Access Modifier

* Methods, Variables and Constructors that are declared private can only be accessed within the declared class

itself

* Variables that are declared private can be accessed outside the class if public getter methods are present in the class.

Protected Access Modifier

* Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

# Java Non Access Modifiers

Static Variables:

* The static key word is used to create variables that will exist independently of any instances created for the class.
* Only one copy of the static variable exists regardless of the number of instances of the class

|  |
| --- |
| **Static Variable Demo** |
| class Demo  {  int x, y; static int count; |

🞆 Static variables are also known as class variables. Local variables cannot be declared static **Program N0. 11**

|  |
| --- |
| void setxy()  {  X = 1; Y = 1; count++;  } }  public class StaticDemo1  {  public static void main(String[] args)  {  System.out.println(“\n\t No. of Objects = “ +  Demo.count);    Demo d1 = new Demo(); d1.setxy();    Demo d2 = new Demo(); d2.setxy();    System.out.println(“\n\t No. of Objects = “ +  Demo.count);  }  } |

Static Methods

* The static key word is used to create methods that will exist independently of any instances created for the class
* Static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables
* Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method

|  |
| --- |
| **Class Product (Static data members and static methods)** |
| class Product  { int pid, price, qty; static double taxRate;    static void setTax()  {  pid = 101;  taxRate = 5.00;  }  static double getTax()  {  return taxRate;  } }  public class StaticDemo2  {  public static void main(String[] args)  {  Product.setTax();  System.out.println("Tax = " + Product.getTax());  }  } |

**Program N0.**

**12**

# Final Modifier

Final Variables

* A final variable can be explicitly initialized only once.
* A reference variable declared final can never be reassigned to refer to a different object
* However the data within the object can be changed. So the state of the object can be changed but not the reference

|  |
| --- |
| **Class Circle** |
| class Circle  {  double rad;  final double PI=3.14;    void setRadius(double r)  {  rad = r;  }  double getArea()  {  return PI\*rad\*rad;  }  }  public class FinalVarDemo1\_13  {  public static void main(String[] args)  {  Circle cir1 = new Circle(); cir1.setRadius(4.30);  System.out.println("\n\t Area of cir1 = " + cir1.getArea());    }  } |

**Program N0.**

**13**

|  |
| --- |
| **Final Object** |
| public class FinalVarDemo\_14  {  public static void main(String[] args)  {  String str1 = "Welcome";    System.out.println("\n\t str1 = " + str1);    final String str2 = "Hello World";    System.out.println("\n\t str2 = " + str2);    str1 = "Welcome to Java";  System.out.println("\n\t str1 = " + str1);    str2 = "This is New String";  System.out.println("\n\t str2 = " + str2);  }  } |

**Program N0.**

**14**

## Final Methods

* A final method cannot be overridden by any subclasses. As mentioned previously the final modifier prevents a method from being modified in a subclass
* The main intention of making a method final would be that the content of the method should not be changed by any outsider

## Final Classes

▪ The main purpose of using a class being declared as final is to prevent the class from being sub classed ▪ If a class is marked as final then no class can inherit any feature from the final class

# Scanner Class

* Scanner class is defined in **java.util** package, it used to perform input operations
* The Scanner class breaks the input into tokens using a delimiter which is whitespace by default ▪ It provides many methods to read and parse various primitive values Commonly used methods of Scanner class:

|  |  |
| --- | --- |
| **Method** | **Description** |
| **pubic String next()** | Returns the next token from the scanner |
| **public String nextLine()** | Moves the scanner position to the next line and returns the value as a string. |
| **public byte nextByte()** | Scans the next token as a byte |
| **public short nextShort()** | Scans the next token as a short value |
| **public int nextInt()** | Scans the next token as an int value |
| **public long nextLong()** | Scans the next token as a long value |
| **public float nextFloat()** | Scans the next token as a float value |
| **public double nextDouble()** | Scans the next token as a double value |

|  |
| --- |
| **Input Age using Scanner Class** |
| import java.util.Scanner; public class ScannerDemo1\_15  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in);    int age;    System.out.print("\n\t Enter your Age :");  age = scan.nextInt();    System.out.println("\n\t Your Age :" + age); }  } |

**Program N0.**

**15**

|  |
| --- |
| **Input average marks, name, character using scanner class** |
| import java.util.Scanner; public class ScannerDemo2\_16  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in); |

**Program N0. 16**

|  |  |
| --- | --- |
| }  } | float avg; char ch; String sname;  System.out.print("\n\t Enter Average Marks :"); avg = scan.nextFloat();  System.out.print("\n\t Enter Any Character :"); ch = scan.next().charAt(0);  System.out.print("\n\t Enter Your Name :"); sname = scan.next();  System.out.println("\n\t Average Marks :" + avg);  System.out.println("\n\t Character :" + ch);  System.out.println("\n\t Name :" + sname); |

# Assignment

**Program N0. 17:** Sum of two numbers

**Program N0. 18:** Input roll no, name marks of 3 subjects and calculate total marks, Average marks of a student **Program N0. 19:** Input radius of a circle and calculate area and circumference

# Command Line Arguments

* A Java application can accept any number of arguments from the command line. This allows the user to specify configuration information when the application is launched.
* The user enters command-line arguments when invoking the application and specifies them after the name of the class to be run.
* For example, suppose a Java application called Sort sorts lines in a file. To sort the data in a file named friends.txt, a user would enter: **java Sort friends.txt**
* When an application is launched, the runtime system passes the command-line arguments to the application's main method via an array of Strings

|  |
| --- |
| **Command line arguments display length of arguments** |
| public class CmdLIneArgsDemo1  {  public static void main(String[] a)  {  System.out.println("\n\t No. of arguments :" + a.length); }  } |

**Program N0.**

**20**

|  |
| --- |
| **Command line arguments – pass two arguments and display the same** |

**Program N0. 21**

public class CmdLIneArgsDemo2

{

public static void main(String[] a)

{

System.out.println("\n\t No. of arguments :" + a.length);

System.out.println("\n\t 1st Argument :" + a[0]);

System.out.println("\n\t 2nd Argument :" + a[1]);

} }

|  |
| --- |
| **Command line arguments – display all arguments using loop** |
| public class CmdLIneArgsDemo3  {  public static void main(String[] a)  {  System.out.println("\n\t No. of arguments :" + a.length);    for(int i=0; i<a.length; i++)  {  System.out.println("\n\t a["+i+"]:" + a[i]);  }  }  } |

**Program N0.**

**22**

▪ If an application needs to support a numeric command-line argument, it must convert a String argument that represents a number, such as "34", to a numeric value.

|  |
| --- |
| **Parsing Numeric Command-Line Arguments** |
| public class CmdLIneArgsDemo4  {  public static void main(String[] args)  {  int n1, n2;  n1 = Integer.parseInt(args[0]); n2 = Integer.parseInt(args[1]);    int sum = n1+n2;    System.out.println("\n\t n1 = " + n1);  System.out.println("\n\t n2 = " + n2);  System.out.println("\n\t sum = " + sum);  }  } |

**Program N0.**

**23**

# Decision Making Statements

* The program executes instructions sequentially. Sometimes, a program requires checking of certain conditions in program execution
* Java provides various conditional statements to check condition and execute statements according conditional

criteria

* Followings are the different conditional statements used in Java
* If Statement
* If-Else Statement
* Nested If-Else Statement
* Switch Case

## The if Statement

* An if statement consists of a Boolean expression followed by one or more statements
* If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not the first set of code after the end of the if statement (after the closing curly brace) will be executed. **The syntax of an if statement is:**

if(Boolean\_expression)

{

//Statements will execute if the Boolean expression is true

}

|  |
| --- |
| **Input two numbers and display the max** |
| import java.util.Scanner; public class IfDemo1  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in);  int a, b;    System.out.print("\n\t Enter a :");   1. = scan.nextInt();     System.out.print("\n\t Enter b :");   1. = scan.nextInt();     if(a>b)  System.out.println("\n\t a is Max");    if(b>a)  System.out.println("\n\t b is Max");  }  } |

**Program N0.**

**24**

|  |
| --- |
| **Input Average marks and display the result pass if avg is greater than or equal to 35, otherwise result is fail** |
| import java.util.Scanner; public class IfDemo2  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in);  float avg;    System.out.print("\n\t Enter Average Marks :");  avg = scan.nextFloat();    if(avg>=35)  {  System.out.println("\n\t You Are Pass");  }  else  {  System.out.println("\n\t You Are Fail");  }  }  } |

**Program N0.**

**25**

|  |
| --- |
| **Input a number and display whether it is positive, negative or zero** |
| import java.util.Scanner; public class IfDemo3  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in);  int no;    System.out.print("\n\t Enter a no. :");  no = scan.nextInt();    if(no>0)  System.out.println("\n\t number is Positive"); else if(no==0)  System.out.println("\n\t number is Zero");  else  System.out.println("\n\t number is Negative");  }  } |

**Program**

### **N0. 26**

|  |
| --- |
| **Input Average marks and display the appropriate grades** |
| import java.util.Scanner; public class IFDemo4\_27  {  public static void main(String[] args)  {  Scanner sc = new Scanner(System.in);  float avg;    System.out.print("\n\t Enter average marks :");  avg = sc.nextFloat();    if(avg>=75)  System.out.println("\n\t Distinction"); else if(avg>=60)  System.out.println("\n\t First Class"); else if(avg>=45)  System.out.println("\n\t Second Class");  else if(avg>=35)  System.out.println("\n\t Pass");  else  System.out.println("\n\t Fail");  }  } |

**Program**

### **N0. 27**

|  |
| --- |
| **Input gender and age of an employee. An employee is eligible for insurance in following conditions**   1. **Employee is male and age is greater than 30** 2. **Employee is Female and age is greater that 25** |
| import java.util.Scanner; public class IFDemo5\_28 {  public static void main(String[] args){  Scanner sc = new Scanner(System.in);  String gender;  int age;    System.out.print("\n\t Enter Gender :");  gender = sc.next();    System.out.print("\n\t Enter Age :");  age = sc.nextInt();    if(gender.toLowerCase().equals("male"))  { |

**Program**

### **N0. 28**

if(age>=30)

System.out.println("\n\t Employee is Eligible for

Insurance");

else

System.out.println("\n\t Not Eligible");

}

else if(gender.equalsIgnoreCase("female"))

{

if(age>=25)

System.out.println("\n\t Employee is Eligible for

Insurance");

else

System.out.println("\n\t Not Eligible");

}

else

{

System.out.println("\n\t Wrong input");

}

}

}

# Assignments

**Input price and qty, calculate bill amount with discount of 5.00% if price is greater than or equal**

**Program N0. 29** **to 50**

**Program N0. 30** **Input a number and display whether it is Odd or Even**

**Program N0. 31** **Input an alphabet and check whether it is vowel or consonant**

# Logical Operators

Logical AND (&&)

* This operator is used to evaluate 2 or more conditions or expressions with relational operators simultaneously. If all expressions to the left and right of the logical expression are TRUE then the whole compound expression is TRUE

|  |
| --- |
| **Input three numbers and display max** |
| import java.util.Scanner;    public class IFDemo5\_28  {  public static void main(String[] args)  {  Scanner sc = new Scanner(System.in);    int a, b, c;    System.out.print("\n\t Enter a :");   1. = sc.nextInt();     System.out.print("\n\t Enter b :");   1. = sc.nextInt();     System.out.print("\n\t Enter c :");   1. = sc.nextInt();     if(a>b && a>c)  {  System.out.println("\n\t a is max");  }  else if(b>a && b>c) |

**Program**

**N0. 32**

|  |  |  |
| --- | --- | --- |
|  |  | {  System.out.println("\n\t b is max");  } else  {  System.out.println("\n\t c is max"); } |
| } | } |  |

Logical OR(||)

* This operator is used to evaluate or combine 2 or more expressions or conditions, and evaluates to TRUE if any one of all the expressions is true

|  |
| --- |
| **Input any alphabet and display whether it is a vowel or a consonant** |
| import java.util.Scanner;    public class IfDemo7  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in);  char ch;  System.out.print("\n\t Enter any alphabet :");  ch = scan.next().charAt(0);    if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u') System.out.println("\n\t It is a Vowel");  else  System.out.println("\n\t It is Consonant");  }  } |

**Program**

**N0. 33**

# Switch Statement

* The Switch Case Statement is used to make a decision from the number of choices
* The expression following the keyword switch is any C expression that evaluates an integer or a char value
* First, the expression following the keyword switch is evaluated. The value is then matched, one by one, against the constant values that follow the case statements. When a match is found, the program executes the statements following that case, and all subsequent case and default statements as well.
* If no match is found with any of the case statements, only the statements following the default are executed. A few examples will show how this control structure works.
* Expressions can also be used in cases provided they are constant expressions. Thus **case 3 + 7** is correct, however, **case a + b** is incorrect
* The **break** statement when used in a **switch** takes the control outside the **switch**. However, use of **continue** will not take the control to the beginning of **switch** as one is likely to believe

|  |
| --- |
| **Switch Demo** |
| public class SwitchDemo1 {  public static void main(String[] args) {  int no; no = 2;  switch(no)  {  case 1:  System.out.println("\n\t It is Number 1"); |

**Program N0. 34**

|  |
| --- |
| break; case 2:  System.out.println("\n\t It is Number 2");  break; default:  System.out.println("Invalid");  }  }  } |
| **Addition..Multiplication.. Menu Program** |
| import java.util.Scanner; public class SwitchDemo2 { public static void main(String[] args) { Scanner scan = new Scanner(System.in); int num1, num2, ans,choice;  System.out.print("\n\t Enter Num1 :"); num1 = scan.nextInt();  System.out.print("\n\t Enter Num2 :");  num2 = scan.nextInt();  System.out.println("\n\t\t1. Addition \n\t\t 2. Subtraction  \n\t\t 3. Multiplication \n\t\t 4. Division"); System.out.print("\n\t Enter a choice : ");  choice = scan.nextInt(); switch(choice)  {  case 1:  ans = num1+num2;  System.out.println("\n\t Addition = " + ans);  break; case 2:  ans = num1-num2;  System.out.println("\n\t Subtraction = " + ans);  break; case 3:  ans = num1\*num2;  System.out.println("\n\t Multiplication = " + ans);  break; case 4:  ans = num1/num2;  System.out.println("\n\t Division = " + ans);  break; default:  System.out.println("\n\t Wrong Input");  }  }  } |

Program N0.

35

|  |
| --- |
| **Pizza Order** |
| import java.util.Scanner; public class SwitchDemo3 {  public static void main(String[] args) { Scanner scan = new Scanner(System.in);  char pizzasize; |

Program N0.

36

|  |  |  |
| --- | --- | --- |
|  |  | System.out.println("\n\t Enter Pizza Size[s/m/l]:"); pizzasize = scan.next().charAt(0);    switch(pizzasize)  { case 's':  System.out.println("\n\t Pizza size : Small");  break; case 'm':  System.out.println("\n\t Pizza size : Medium");  break; case 'l':  System.out.println("\n\t Pizza size : Large");  break; default:  System.out.println("\n\t Invalid Input"); } |
| } | } |  |

# Assignments

|  |  |  |
| --- | --- | --- |
| **Program No. 37** | If cost price and selling price of an item is input through the keyboard, write a program to determine whether the seller has made profit or incurred loss. Also determine how much profit he made or loss he incurred. | |
| **Program No. 38** | Any year is input through the keyboard. Write a program to determine whether the year is a leap year or not. (Hint: Use the % (modulus) | |
| **Program No. 39** | Write a program to check whether a triangle is valid or not, when the three angles of the triangle are entered through the keyboard. A triangle is valid if the sum of all the three angles is equal to 180 degrees | |
| **Program No. 40** | A certain grade of steel is graded according to the following conditions:   1. Hardness must be greater than 50 2. Carbon content must be less than 0.7 (iii) Tensile strength must be greater than 5600 The grades are as follows:   Grade is 10 if all three conditions are met  Grade is 9 if conditions (i) and (ii) are met  Grade is 8 if conditions (ii) and (iii) are met  Grade is 7 if conditions (i) and (iii) are met  Grade is 6 if only one condition is met  Grade is 5 if none of the conditions are met  Write a program, which will require the user to give values of hardness, carbon content and tensile strength of the steel under consideration and output the grade of the steel | |
| **Program No. 41** | A library charges a fine for every book returned late. For first 5 days the fine is 50 paise, for 6-10 days fine is one rupee and above 10 days fine is 5 rupees. If you return the book after 30 days your membership will be cancelled. Write a program to accept the number of days the member is late to return the book and display the fine or the appropriate message. | |
| **Program No. 42** | The policy followed by a company to process customer orders is given by the following rules:   1. If a customer order is less than or equal to that in stock and has credit is OK, supply has requirement. 2. If has credit is not OK do not supply. Send him intimation. | |
|  | c. | If has credit is Ok but the item in stock is less than has order, supply what is in stock.  Intimate to him data the balance will be shipped.  Write a C program to implement the company policy. |

# Java Loops

* There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop.
* Java has very flexible three looping mechanisms. You can use one of the following three loops:
  + while Loop
  + do...while Loop
  + for Loop
  + As of Java 5, the enhanced for loop was introduced. This is mainly used for Arrays.

# The while Loop

|  |
| --- |
| **Syntax:**  **while(Boolean\_expression)**  **{**  **//Statements**  **}** |

* A while loop is a control structure that allows you to repeat a task a certain number of times.
* When executing, if the boolean\_expression result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.
* Here, key point of the while loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

|  |
| --- |
| **Print helloworld 5 times** |
| public class WhileLoopDemo1 {  public static void main(String[] args) {  int i;    i = 1;  while(i<=5)  {  System.out.println("\n\t Hello World"); i++;  }  }  } |

**Program N0.**

**43**

|  |
| --- |
| **Input 10 Number using While loop** |
| import java.util.Scanner; public class WhileDemo\_44  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in);    int no, i; i = 1;  while(i<=10)  { |

**Program N0. 44**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | } | System.out.print("\n\t Enter no :"); no = scan.nextInt(); i++; |
| } | } |  |  |

|  |
| --- |
| **Input 10 numbers and display count of positive, negative, zeros** |
| import java.util.Scanner; public class WhileDemo\_45  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in); int no, i, pos=0, neg=0, zero=0;  i = 1;    while(i<=10)  {    System.out.print("\n\t Enter no :");  no = scan.nextInt();  if(no>0)  pos++;  else if(no==0)  zero++; else neg++;  i++;  }  System.out.println("\n\t Total Positives :" + pos);  System.out.println("\n\t Total Negatives :" + neg);  System.out.println("\n\t Total Zeros :" + zero);  }  } |

**Program N0.**

**45**

# do...while Loop

* A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time
* Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

|  |
| --- |
| **Syntax:**  **Do**  **{**  **//Statements**  **} while(Boolean\_expression);** |

* If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

▪

|  |
| --- |
| **Do..while demo** |
| public class DoWhileLoopDemo1  { public static void main(String[] args) { int i = 10;    do  {  System.out.println("Value of i = " + i); |

**Program N0. 46**

|  |  |  |
| --- | --- | --- |
|  |  | i++;  }while(i<=5); |
| } | } |  |

7

|  |
| --- |
| **Input Record of Student until user enters “stop”** |
| import java.util.Scanner;    public class WhileDemo\_47  {  public static void main(String[] args)  {  Scanner scan = new Scanner(System.in);  int roll;  choice;    do  {  System.out.print("\n\n\t Enter Roll No. :");  roll = scan.nextInt();    System.out.print("\n\n\t Enter Name :");  name = scan.next();    System.out.print("\n\n\t Next Record? [ok=Next |stop = Cancel] :" );  choice = scan.next();    }while(!choice.equalsIgnoreCase("stop"));  }  } |

**Program N0.**

**47**

# For loop

|  |
| --- |
| **Syntax**: for(initialization; Boolean\_expression; update)  {  //Statements } |

* A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
* A for loop is useful when you know how many times a task is to be repeated.
* The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
* Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
* After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.

|  |
| --- |
| **For Loop Demo** |
| public class ForDemo\_48  {  public static void main(String[] args)  {  for(int i=0; i<5; i++)  { |

* The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates **Program N0. 48**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | } | System.out.println("i = " + i); |
| } | } |  |  |

## Enhanced for loop in Java

|  |
| --- |
| **Syntax:**  for(declaration : expression)  {  //Statements  } |

* As of Java 5, the enhanced for loop was introduced. This is mainly used for Arrays.
* **Declaration:** The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.
* **Expression:** This evaluates to the array you need to loop through.

The expression can be an array variable or method call that returns an array.

|  |
| --- |
| **Enhanced For loop Demo** |
| public class ForDemo\_49{  public static void main(String[] args)  {  System.out.println("\n\t Command line arguments are :");    for(String item : args)  {  System.out.println(item);  }  }  } |

**Program N0.**

**49**

### The break Keyword

▪ The break keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch

A diagram of a break

Description automatically generated

statement.

|  |
| --- |
| **Break Statement** |
|  |

Program N0.

50

## The continue Keyword

▪ The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop

A diagram of a break

Description automatically generated

|  |
| --- |
| **Continue Statement** |
| /\* Java49. WAP to display the numbers upto 100 by using continue keyword to continue or break \*/    import java.util.Scanner; class Java49  {  public static void main(String []args)  {  Scanner scan = new Scanner(System.in);    int cnt,i;  String result;    cnt=0;  for(i=1;i<=100;i++)  {  System.out.print(" " + i);  cnt++;  if(cnt==10)  {  cnt=0;  System.out.println("\n\t Do you to Continue  [yes/no] :");  result = scan.next();    if(result.equals("yes"))  {  continue;  } else  {  break;  }  }  }  }  } |

Program N0.

51