USN										

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC
CELEBRATING 25 YEARS

Internal Assessment Test 1 – Sept. 2017

Sub:	**Programming in JAVA(open elective)**				Sub Code:	**15CS561**	Branch:	**EEE,ECE,TCE, ME,CIV**
Date:	21/09/2017	Duration:	90 min's	Max Marks: 50	Sem / Sec:	V(A,B,C)		OBE

<u>Answer any FIVE FULL Questions</u>

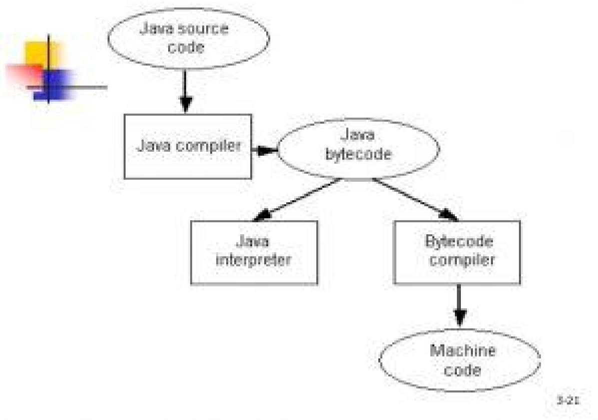
			MARKS	CO	RBT
1	(a)	"Java is platform Independent." Justify this statement.	[05]	CO1	L2
	(b)	Write a Java program to print sum of n elements using "for each" version of loop.	[05]	CO2	L3
2	(a)	Explain with the help of program: type conversion and type casting.	[10]	CO1	L2
3	(a)	Write Short note on primitive data types in Java.	[05]	CO1	L2
	(b)	Define variable and explain the declaration, scope and Lifetime of the variable.	[05]	CO1	L1
4	(a)	Explain Java Selection statements with help of example.	[10]	CO2	L2
5	(a)	Write a short note on: i. >>> ii. Short circuit logical operators.	[05]	CO2	L2
	(b)	Write a Java program to find the Largest of two numbers using ternary operator.	[05]	CO2	L3
6	(a)	Discuss three OOP principles.	[05]	CO1	L2
	(b)	Write a Java program to check if the entered number is a prime number or not.	[05]	CO2	L3
7	(a)	List and Explain the Java BUZZWORDS.	[10]	CO1	L2

1.why java is platform indepenent?

The meaning of platform independent is that, the java source code can run on all operating systems.
A program is written in a language which is a human readable language. It may contain words, phrases etc which the machine does not understand. For the source code to be understood by the machine, it needs to be in a language understood by machines, typically a machine-level language. So, here comes the role of a compiler. The compiler converts the high-level language (human language) into a format understood by the machines. Therefore, a compiler is a program that translates the source code for another program from a programming language into executable code.
This executable code may be a sequence of machine instructions that can be executed by the CPU directly, or it may be an intermediate representation that is interpreted by a virtual machine. This intermediate representation in Java is the **Java Byte Code.**

Step by step Execution of Java Program:

- Whenever, a program is written in JAVA, the javac compiles it.
- The result of the JAVA compiler is the **.class file or the bytecode** and not the machine native code (unlike C compiler).
- The bytecode generated is a non-executable code and needs an interpreter to execute on a machine. This interpreter is the JVM and thus the Bytecode is executed by the JVM.
- And finally program runs to give the desired output.



In case of C or C++ (language that are not platform independent), the compiler generates an .exe file which is OS dependent. When we try to run this .exe file on another OS it does not run, since it is OS dependent and hence is not compatible with the other OS.

Java is platform independent but JVM is platform dependent

In Java, the main point here is that the JVM depends on the operating system – so if you are running Mac OS X you will have a different JVM than if you are running Windows or some other operating system. This fact can be verified by trying to download the JVM for your particular machine – when trying to download it, you will given a list of JVM's corresponding to different operating systems, and you will obviously pick whichever JVM is targeted for the operating system that you are running. So we can conclude that JVM is platform dependent and it is the reason why Java is able to become "Platform Independent".
Important Points:

- In the case of Java, **it is the magic of Bytecode that makes it platform independent**.
- This adds to an important feature in the JAVA language termed as **portability**. Every system has its own JVM which gets installed automatically when the jdk software is installed. For every operating system separate JVM is available which is capable to read the .class file or byte code.
- An important point to be noted is that while **JAVA is platform-independent language, the JVM is platform-dependent.** Different JVM is designed for different OS and byte code is able to run on different OS.

1.b Find the sum of n numbers using for each statement?

```
class SumOfNumbers
{
    public static void main(String arg[])
    {
        int n = 5;
        int sum = 0;
         int input[];
for(int i=1;i<=n;i++)
input[i]=i;
        for(int x: input)
        {
            sum = sum + x;    // LINE A

        }


        System.out.println("Sum of numbers till " + input + " is " + sum); // LINE B

        }
}
```

2.Explaintype cating and type conversion in java?

Type conversion in Java with Examples

When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly. For example, assigning an int value to a long variable.

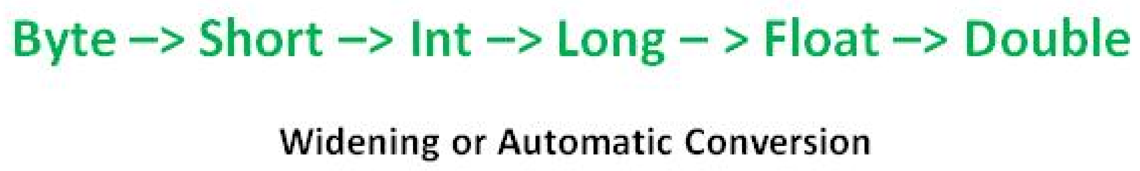
Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.

- When we assign value of a smaller data type to a bigger data type.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.


Byte –> Short –> Int –> Long – > Float –> Double
Widening or Automatic Conversion

Example:

```
class Test
{
        public static void main(String[] args)
        {
                int i = 100;

                //automatic type conversion
                long l = i;

                //automatic type conversion
                float f = l;
System.out.println("Int value "+i);
                System.out.println("Long value "+l);
                System.out.println("Float value "+f);
        }
}
```

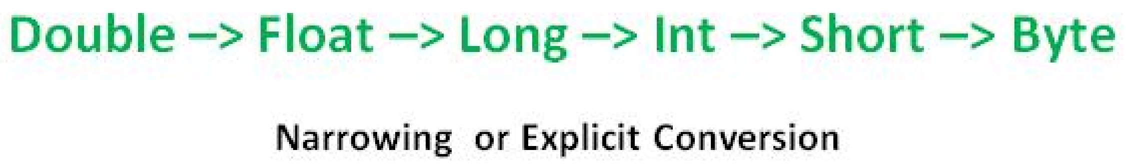
Output:

```
Int value 100
Long value 100
Float value 100.0
```

Narrowing or Explicit Conversion

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.


Double –> Float –> Long –> Int –> Short –> Byte
Narrowing or Explicit Conversion

char and number are not compatible with each other. Let’s see when we try to convert one into other.

```
//Java program to illustrate incompatible data
// type for explicit type conversion
public class Test
{
  public static void main(String[] argv)
  {
    char ch = 'c';
    int num = 88;
    ch = num;
  }
}
```

Error:

```
7: error: incompatible types: possible lossy conversion from int to char
    ch = num;
         ^
1 error
```

How to do Explicit Conversion?

Example:

```
//Java program to illustrate explicit type conversion
class Test
{
        public static void main(String[] args)
        {
                double d = 100.04;

                //explicit type casting
                long l = (long)d;

      //explicit type casting
                int i = (int)l;
                System.out.println("Double value "+d);
                //fractional part lost
                System.out.println("Long value "+l);

                //fractional part lost
                System.out.println("Int value "+i);
        }
}
```

Output:

```
Double value 100.04
Long value 100
Int value 100
```

While assigning value to byte type the fractional part is lost and is reduced to modulo 256(range of byte).
Example:

```
//Java program to illustrate Conversion of int and double to byte
class Test
{
        public static void main(String args[])
        {
                byte b;
                int i = 257;
                double d = 323.142;
            System.out.println("Conversion of int to byte.");

                //i%256
                b = (byte) i;
                System.out.println("i = b " + i + " b = " + b);
                System.out.println("\nConversion of double to byte.");

                //d%256
                b = (byte) d;
                System.out.println("d = " + d + " b= " + b);
        }
}
```

Output:

```
Conversion of int to byte.
i = 257 b = 1

Conversion of double to byte.
d = 323.142 b = 67
```

Type promotion in Expressions

While evaluating expressions, the intermediate value may exceed the range of operands and hence the expression value will be promoted. Some conditions for type promotion are:

1. Java automatically promotes each byte, short, or char operand to int when evaluating an expression.
2. If one operand is a long, float or double the whole expression is promoted to long, float or double respectively.

Example:

```
//Java program to illustrate Type promotion in Expressions
class Test
{
        public static void main(String args[])
        {
                byte b = 42;
                char c = 'a';
                short s = 1024;
                int i = 50000;
                float f = 5.67f;
                double d = .1234;

                // The Expression
                double result = (f * b) + (i / c) - (d * s);

                //Result after all the promotions are done
System.out.println("result = " + result);
        }
}
```

Output:

```
Result = 626.7784146484375
```

Explicit type casting in Expressions

While evaluating expressions, the result is automatically updated to larger data type of the operand. But if we store that result in any smaller data type it generates compile time error, due to which we need to type cast the result.
Example:

```
//Java program to illustrate type casting int to byte
class Test
{
        public static void main(String args[])
        {
   byte b = 50;

                //type casting int to byte
                b = (byte)(b * 2);
                System.out.println(b);
```

```
    }
}
```

Output

```
 100
```

NOTE- In case of single operands the result gets converted to int and then it is type casted accordingly.
Example:

```
//Java program to illustrate type casting int to byte
class Test
{
    public static void main(String args[])
    {
        byte b = 50;

        //type casting int to byte
        b = (byte)(b * 2);
System.out.println(b);
    }
}
```

Output

```
100
```

3.a)explain the primitive types of java?
The Java programming language is statically-typed, which means that all variables must first be declared before they can be used. This involves stating the variable's type and name, as you've already seen:

```
int gear = 1;
```

Doing so tells your program that a field named "gear" exists, holds numerical data, and has an initial value of "1". A variable's data type determines the values it may contain, plus the operations that may be performed on it. In addition to int, the Java programming language supports seven other *primitive data types*. A primitive type is predefined by the language and is named by a reserved keyword. Primitive values do not share state with other primitive values. The eight primitive data types supported by the Java programming language are:

- **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large arrays, where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.

- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.

- **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -2^{31} and a maximum value of $2^{31}-1$. In Java SE 8 and later, you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of $2^{32}-1$. Use the Integer class to use int data type as an unsigned integer. See the section The Number Classes for more information. Static methods like compareUnsigned, divideUnsigned etc have been added to the Integer class to support the arithmetic operations for unsigned integers.

- **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of $2^{64}-1$. Use this data type when you need a range of values wider than those provided by int. The Long class also contains methods like compareUnsigned, divideUnsigned etc to support arithmetic operations for unsigned long.

- **float**: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the Floating-Point Types, Formats, and Values section of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the java.math.BigDecimal class instead. Numbers and Strings covers BigDecimal and other useful classes provided by the Java platform.

- **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the Floating-Point Types, Formats, and Values section of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.

- **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.

- **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

b)define variable?explain the scope and lifetime of variable?

Scope and Lifetime of Variables

The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before it is destroyed. Destroying variables refers to deallocating the memory that was allotted to the variables when declaring it. We have written a few classes till now. You might have observed that not all variables are the same. The ones declared in the body of a method were different from those that were declared in the class itself. There are there types of variables: instance variables, formal parameters or local variables and local variables.

4.explain java selection statements with example?
In Java, these are used to control the flow of the program. These are the types of selection statements in Java.

- If statement
- If-else statement

- Switch statement

If Statement

In Java, "if" is a conditional statement. It will provides execution of one of two statements (or blocks), depending on the condition.

Syntax

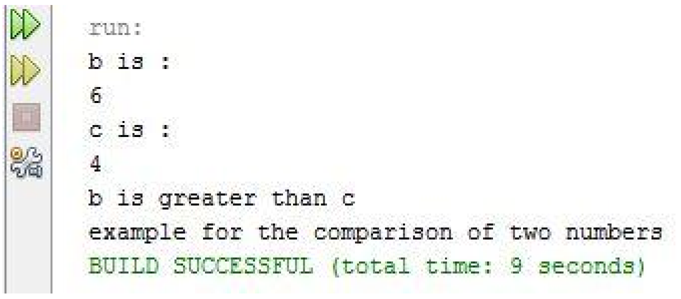
```
if(condition)
{
    statements;
    ...
    ...
}
```

If the condition is true then the statements inside the if block will be executed. The execution will be continued to the next statements after the execution of the statements in the if block. If the condition is false then the statements inside the if block will not be executed and the execution will start from the statements that are next to the if block.

Example

```
package test;
import java.util.Scanner;
publicclass Test
{
    publicstaticvoid main(String args[])
    {
        int b,c;
        Scanner scnr = new Scanner(System.in);
        System.out.println("b is : ");
        b=scnr.nextInt();
        System.out.println("c is : ");
        c=scnr.nextInt();
        if (b > c)
        {
            System.out.println("b is greater than c");
        }
        System.out.println("example for the comparison of two numbers");
    }
}
```

Output



```
run:
b is :
6
c is :
4
b is greater than c
example for the comparison of two numbers
BUILD SUCCESSFUL (total time: 9 seconds)
```

If-else Statement

If the condition is true then the statements inside the if block will be executed and if the condition is false then the statements inside the else block will be executed.

Syntax

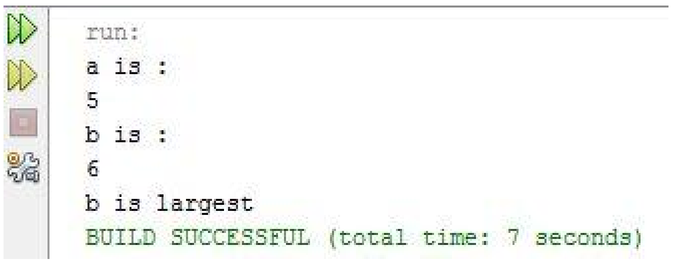
```
if (condition)
{
    first statement;
}
else
{
    second statement;
}
```

If the condition is true then the first statement will be executed and if the condition is false then the second statement will be executed.

Example

```
package demo;
import java.util.Scanner;
publicclass Demo
{
    publicstaticvoid main(String args[])
    {
        int a,b;
        Scanner scnr = new Scanner(System.in);
        System.out.println("a is : ");
        a=scnr.nextInt();
        System.out.println("b is : ");
        b=scnr.nextInt();
        if (a > b)
        {
            System.out.println("a is largest");
        }
        else
        {
            System.out.println("b is largest");
        }
    }
}
```

Output



```
run:
a is :
5
b is :
6
b is largest
BUILD SUCCESSFUL (total time: 7 seconds)
```

Switch Statement

A switch statement can be easier than if-else statements. In the switch we have multiple cases. The matching case will be executed. In the switch statement we can only use int, char, byte and short data types.

Syntax

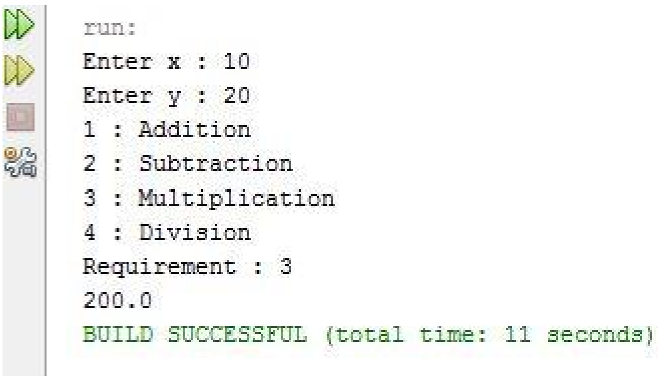
```
switch (expression)
{
    case 1:
    {
        statement;
    }
    break;
    case 2:
    {
        statement;
    }
    break;
    .
    .
    .
    case N:
    {
        statement;
    }
    break;
    default:
    {
        statement;
    }
    break;
}
```

Example

```
package demo;
import java.util.Scanner;
publicclassDemo
{
    publicstaticvoid main(String[] args)
    {
        int x,y,r;
        double z;
        Scanner scnr = new Scanner(System.in);
        System.out.print("Enter x : ");
        x=scnr.nextInt();
        System.out.print("Enter y : ");
        y=scnr.nextInt();
        System.out.println("1 : Addition");
        System.out.println("2 : Subtraction");
        System.out.println("3 : Multiplication");
        System.out.println("4 : Division");
        System.out.print("Requirement : ");
        r=scnr.nextInt();
        switch(r)
        {
            case 1:
            {
                z=x+y;
                System.out.println(z);
            }
            break;
            case 2:
            {
                z=x-y;
                System.out.println(z);
            }
            break;
            case 3:
```

```
            {
                z=x*y;
                System.out.println(z);
            }
            break;
            case 4:
            {
                z=x/y;
                System.out.println(z);
            }
            break;
            default:
            {
                System.out.println("Requirement is invalid");
            }
        }
    }
}
```

Output



```
run:
Enter x : 10
Enter y : 20
1 : Addition
2 : Subtraction
3 : Multiplication
4 : Division
Requirement : 3
200.0
BUILD SUCCESSFUL (total time: 11 seconds)
```

5a).write a noe on
i.>>>
ii.short circuit logical operators

i.The shift operators include left shift <<, signed right shift >>, and unsigned right shift >>>.
The value of n>>s is n right-shifted s bit positions with **sign-extension**.

The value of n>>>s is n right-shifted s bit positions with **zero-extension**.

```
System.out.println(Integer.toBinaryString(-1));
    // prints "11111111111111111111111111111111"
    System.out.println(Integer.toBinaryString(-1 >> 16));
    // prints "11111111111111111111111111111111"
    System.out.println(Integer.toBinaryString(-1 >>> 16));
    // prints "1111111111111111"
```

ii.
The && and || operators "short-circuit", meaning they don't evaluate the right hand side if it isn't necessary.

The & and | operators, when used as logical operators, always evaluate both sides.

There is only one case of short-circuiting for each operator, and they are:

- false && ... - it is not necessary to know what the right hand side is, the result must be false
- true || ... - it is not necessary to know what the right hand side is, the result must be true

Let's compare the behaviour in a simple example:

```
public boolean longerThan(String input, int length) {
    return input != null && input.length() > length;
}

public boolean longerThan(String input, int length) {
    return input != null & input.length() > length;
}
```

The 2nd version uses the non-short-circuiting operator & and will throw a NullPointerException if input is null, but the 1st version will return false without an exception;

6.a)discuss oops principles?

The four principles of OOP.

Dimistifying Object Oriented Programming is not easy at first, but re-read this article a few times and you'll rank up. I'm going to give you some insight into the four principles of Object Oriented Programming:

1. Encapsulation:

Encapsulation means that the internal representation of an object is generally hidden from view outside of the object's definition. Typically, only the object's own methods can directly inspect or manipulate its fields.

Encapsulation is the hiding of data implementation by restricting access to accessors and mutators.

An accessor is a method that is used to ask an object about itself. In OOP, these are usually in the form of properties, which have a get method, which is an accessor method. However, accessor methods are not restricted to properties and can be any public method that gives information about the state of the object.

A *Mutator* is public method that is used to modify the state of an object, while hiding the implementation of exactly how the data gets modified. It’s the set method that lets the caller modify the member data behind the scenes.

Hiding the internals of the object protects its integrity by preventing users from setting the internal data of the component into an invalid or inconsistent state. This type of data protection and implementation protection is called Encapsulation.

A benefit of encapsulation is that it can reduce system complexity.

2. Abstraction

Data abstraction and encapuslation are closely tied together, because a simple definition of data abstraction is the development of classes, objects, types in terms of their interfaces and functionality, instead of their implementation details. Abstraction denotes a model, a view, or some other focused representation for an actual item.

“An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of object and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.” — G. Booch

In short, data abstraction is nothing more than the implementation of an object that contains the same essential properties and actions we can find in the original object we are representing.

3. Inheritance

Inheritance is a way to reuse code of existing objects, or to establish a subtype from an existing object, or both, depending upon programming language support. In classical inheritance where objects are defined by classes, classes can inherit attributes and behavior from pre-existing classes called base classes, superclasses, parent classes or ancestor classes. The resulting classes are known as derived classes, subclasses or child classes. The relationships of classes through inheritance gives rise to a hierarchy.

Subclasses and Superclasses A subclass is a modular, derivative class that inherits one or more properties from another class (called the superclass). The properties commonly include class data variables, properties, and methods or functions. The superclass establishes a common interface and foundational functionality, which specialized subclasses can inherit, modify, and supplement. The software inherited by a subclass is considered reused in the subclass. In some cases, a subclass may customize or redefine a method inherited from the superclass. A superclass method which can be redefined in this way is called a virtual method.

4. Polymorphism

Polymorphism means one name, many forms. Polymorphism manifests itself by having multiple methods all with the same name, but slightly different functionality.

There are 2 basic types of polymorphism. Overridding, also called run-time polymorphism. For method overloading, the compiler determines which method will be executed, and this decision is made when the code gets compiled. Overloading, which is referred to as compile-time polymorphism. Method will be used for method overriding is determined at runtime based on the dynamic type of an object.

If you can grasp these four principles, OOP can be much of a breeze for you. It might take more than one read, I encourage you to practically try it.

b)largest of 2 numbers using ternary operator?

```
import java.util.Scanner;


public class JavaProgram
{
    public static void main(String args[])
    {
        int a, b, big;
        Scanner scan = new Scanner(System.in);

        System.out.print("Enter Two Number : ");
        a = scan.nextInt();
        b = scan.nextInt();


            big = (a>b)?a:b;



        System.out.print("Largest of Two Number is " +big);
    }
}
```

7. Explain java buzzwords?

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

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

Simple

- **Java was designed to be easy for professional programmer to learn and use effectively.**
- **It’s simple and easy to learn if you already know the basic concepts of Object Oriented Programming.**

- **C++ programmer can move to JAVA with very little effort to learn.**
- **In Java, there is small number of clearly defined ways to accomplish a given task002E**

Object Oriented

- **Java is true object oriented language.**
- **Almost "Everything is an Object" paradigm. All program code and data reside within objects and classes.**
- **The object model in Java is simple and easy to extend.**
- **Java comes with an extensive set of classes, arranged in packages that can be used in our programs through inheritance.**

Distributed

- **Java is designed for distributed environment of the Internet. Its used for creating applications on networks.**
- **Java applications can access remote objects on Internet as easily as they can do in local system.**
- **Java enables multiple programmers at multiple remote locations to collaborate and work together on a single project.**

Compiled and Interpreted

- **Usually a computer language is either compiled or Interpreted. Java combines both this approach and makes it a two-stage system.**
- **Compiled : Java enables creation of a cross platform programs by compiling into an intermediate representation called Java Bytecode.**
- **Interpreted : Bytecode is then interpreted, which generates machine code that can be directly executed by the machine that provides a Java Virtual machine.**

Robust

- **It provides many features that make the program execute reliably in variety of environments.**
- **Java is a strictly typed language. It checks code both at compile time and runtime.**
- **Java takes care of all memory management problems with garbage-collection.**
- **Java, with the help of exception handling captures all types of serious errors and eliminates any risk of crashing the system.**

Secure

- **Java provides a "firewall" between a networked application and your computer.**
- **When a Java Compatible Web browser is used, downloading can be done safely 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 to access other parts of the computer.**

Architecture Neutral

- **Java language and Java Virtual Machine helped in achieving the goal of "write once; run anywhere, any time, forever."**
- **Changes and upgrades in operating systems, processors and system resources will not force any changes in Java Programs.**

Portable

- **Java Provides a way to download programs dynamically to all the various types of platforms connected to the Internet.**
- **It helps in generating Portable executable code.**

High Performance

- **Java performance is high because of the use of bytecode.**
- **The bytecode was used, so that it was easily translated into native machine code.**

Multithreaded

- **Multithreaded Programs handled multiple tasks simultaneously, which was helpful in creating interactive, networked programs.**
- **Java run-time system comes with tools that support multiprocess synchronization used to construct smoothly interactive systems.**

Dynamic

- **Java is capable of linking in new class libraries, methods, and objects.**
- **It can also link native methods (the functions written in other languages such as C and C++).**