JAVA PROGRAMMING

BCA 113

UNIT-1

Comparison of C++ and JAVA, JAVA and Internet, JAVA support systems, JAVA environment, JAVA program structure, Tokens, Statements, JVM, Constant and Variables, Data Types, Declaration of variables, Scope of variables, Symbolic constants, Type Casting Operators: Arithmetic, Relational, Logical assignments, Increment and Decrement, Conditional, Bitwise, Special, Expressions and its evaluation.

JAVA: Java is an Object-Oriented High-level programming language it is a platform independent, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak in 1991*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Why to Learn java Programming?

Java is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Software Development Domain. I will list down some of the key advantages of learning Java Programming:

- **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 the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it 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.
- **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** − Being architecture-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.

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts: OOP, concepts in java is to improve code readability and reusability by defining a Java program efficiently.
The main principles of object-oriented programming are **abstraction, encapsulation, inheritance, and polymorphism**. These concepts aim to implement real-world entities in programs.

Component Of Object-Oriented Programming

- Object

- Class
- Abstraction
- Inheritance
- Polymorphism
- Encapsulation

Object:-

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; colour is white, known as its state. It is used to write, so writing is its behaviour.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

CLASS: - A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**

- **Blocks**
- **Nested class and interface**

Syntax of class:-

```
1. class <class_name>{
2.     field;
3.     method;
4. }
```

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

```
1. class Subclass-name extends Superclass-name
2. {
```

3. //methods and fields
4. }

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Data abstraction: - is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either **abstract classes** or **interfaces**

Encapsulation is one of the four fundamental OOP concepts. Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java −

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

C++ vs Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy

		to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.

Structure of Java Program

Java is an object-oriented programming, platform independent, and secure programming language that makes it popular. Using the Java programming language, we can develop a wide variety of applications. So, before diving in depth, it is necessary to understand the basic structure of Java program in detail. In this section, we have discussed the basic structure of a Java program. At the end of this section, you will able to develop the Hello world Java program, easily.

Let's see which elements are included in the structure of a Java program. A typical structure of a Java program contains the following elements:

- Documentation Section
- Package Declaration
- Import Statements
- Interface Section
- Class Definition
- Class Variables and Variables
- Main Method Class
- Methods and Behaviors

Documentation Section: -

Comment in java programme

To write the statements in the documentation section, we use **comments**. The comments may be **single-line, multi-line,** and **documentation** comments.

There are 3 types of comment in java program

- **Single-line Comment:** It starts with a pair of forwarding slash **(//)**. For example:

```
1. //First Java Program
```

- **Multi-line Comment:** It starts with a **/*** and ends with ***/.** We write between these two symbols. For example:

```
1. /*It is an example of
2. multiline comment*/
```

- **Documentation Comment:** It starts with the delimiter **(/**)** and ends with ***/**. For example:

```
1. /**It is an example of documentation comment*/
```

Java Character Set: - Characters are the smallest units (elements) of Java language that are used to write Java tokens. These characters are defined by the Unicode character set. A character set in Java is a set of alphabets, letters, and some special characters that are valid in java programming language.

1. Letters: Both lowercase (a, b, c, d, e, etc.) and uppercase (A, B, C, D, E, etc.) letters.

2. Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
3. Special symbols: _, (,), {, }, [,], +, -, *, /, %, !, &, |, ~, ^, <, =, >, $, #, ?, Comma (,), Dot (.), Colon (:), Semi-colon (;), Single quote (‘), Double quote (“), Back slash (\).
4. White space: Space, Tab, New line.

Tokens are the various elements in the java program that are identified by Java compiler. A token is the smallest individual element (unit) in a program that is meaningful to the compiler.

Types of Tokens: - Java language contains five types of tokens that are as follows:

1. Keywords
2. Identifiers
3. Literals/constant,
4. Operators
5. Separators

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name. There are 48 keywords in java

Below is the list of reserved keywords in Java:

abstract	continue	for	protected	transient
Assert	Default	Goto	public	Try
Boolean	Do	If	Static	throws
break	double	implements	strictfp	Package
byte	else	import	super	Private
case	enum	Interface	Short	switch
Catch	Extends	instanceof	return	void
Char	Final	Int	synchronized	volatile
class	finally	long	throw	Date
const	float	Native	This	while

Identifiers in Java are symbolic names used for identification. They can be a class name, variable name, method name, package name, constant name, and more. However, in Java, there are some reserved words that cannot be used as an identifier.

Rules for Identifiers in Java

There are some rules and conventions for declaring the identifiers in Java. If the identifiers are not properly declared, we may get a compile-time error. Following are some rules and conventions for declaring identifiers:

- A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(_) or a dollar sign ($). for example, @javatpoint is not a valid identifier because it contains a special character which is @.
- There should not be any space in an identifier. For example, java tpoint is an invalid identifier.
- An identifier should not contain a number at the starting. For example, 123javatpoint is an invalid identifier.
- An identifier should be of length 4-15 letters only. However, there is no limit on its length. But, it is good to follow the standard conventions.
- We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.
- An identifier should not be any query language keywords such as SELECT, FROM, COUNT, DELETE, etc.

Constants in Java: -

A value which is fixed and does not change during the execution of a program is called **constants in java**. In other words, Java constants are fixed (known as immutable) data values that cannot be changed. Java supports various types of constants. They are as follows:

1. Integer Constants
2. Real Constants
3. Character Constants
4. String Constants

Integer Constants in Java

An integer constant is a sequence of digits without a decimal point. For example, 10 and -200 are integer constants. There are three types of integer constants. They are as follows:

- Decimal integer
- Octal integer
- Hexadecimal integer

Real (Floating-point) Constants in Java

Real constants consist of a sequence of digits with fractional parts or decimal points. These constants are also called floating-point constants. The valid examples of real constants are 2.3, 0.0034, -0.75, 56.7, etc.

Character Constants in Java

A single character constant (or simply character constant) is a single character enclosed within a pair of single quote. The example of single-character constants are as follows:

'5' 'x' ';' ' ' etc.

String Constants

A string constant is a sequence of characters within a pair of double-quotes. The characters can be alphabets, special characters, digits, and blank spaces. The valid examples of string constants are given below:

"Hello Java" "1924" "?…!" "2+7" "X" etc.

Variable: - A variable is a name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location; all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before use.

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

What is JVM

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include Boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include **Classes, Interfaces, and Arrays.**

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

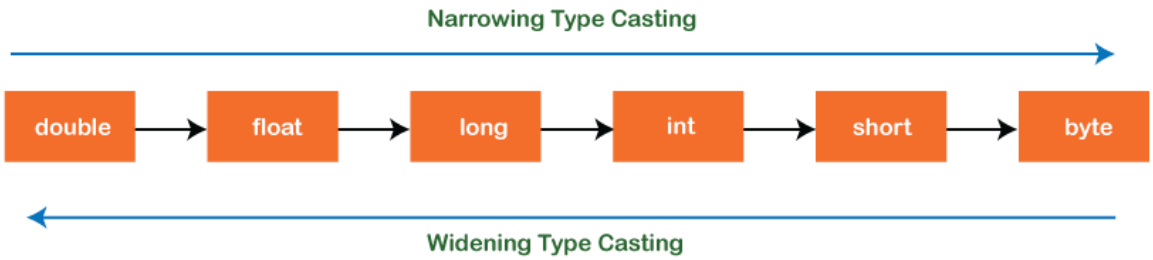
There are 8 types of primitive data types:

1. boolean data type
2. byte data type
3. char data type
4. short data type
5. int data type
6. long data type
7. float data type
8. double data type

Type	Default	Size	Example Literals
boolean	false	1 bit	true, false
byte	0	8 bits	(none)
char	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\'', '\n', 'ß'
short	0	16 bits	(none)
int	0	32 bits	-2, -1, 0, 1, 2
long	0	64 bits	-2L, -1L, 0L, 1L, 2L
float	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F
double	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d

Type Casting in Java

In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. In this section, we will discuss **type casting** and **its types** with proper examples.



Type Casting in Java

Type casting

Convert a value from one data type to another data type is known as **type casting**.

Types of Type Casting

There are two types of type casting:

- **Widening Type Casting/** implicit conversion
- **Narrowing Type Casting/** explicit conversion

Widening Type Casting

Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

- Both data types must be compatible with each other.
- The target type must be larger than the source type.

1. **byte** -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

For example, the conversion between numeric data type to char or Boolean is not done automatically. Also, the char and Boolean data types are not compatible with each other. Let's see an example.

```
1. public class WideningTypeCastingExample
2. {
3. public static void main(String[] args)
4. {
5. int x = 7;
6. //automatically converts the integer type into long type
7. long y = x;
8. //automatically converts the long type into float type
9. float z = y;
10. System.out.println("Before conversion, int value "+x);
11. System.out.println("After conversion, long value "+y);
12. System.out.println("After conversion, float value "+z);
13. }
14. }
```

Narrowing Type Casting

Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.

1. **double** -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

Let's see an example of narrowing type casting.

In the following example, we have performed the narrowing type casting two times. First, we have converted the double type into long data type after that long data type is converted into int type.

NarrowingTypeCastingExample.java

```
1. public class NarrowingTypeCastingExample
2. {
3. public static void main(String args[])
4. {
5. double d = 166.66;
6. //converting double data type into long data type
7. long l = (long)d;
8. //converting long data type into int data type
9. int i = (int)l;
10. System.out.println("Before conversion: "+d);
11. //fractional part lost
12. System.out.println("After conversion into long type: "+l);
13. //fractional part lost
14. System.out.println("After conversion into int type: "+i);
15. }
16. }
```

Operators in Java

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

- Arithmetic Operators
- Unary Operators
- Assignment Operator
- Relational Operators
- Logical Operators
- Ternary Operator
- Bitwise Operators

- Shift Operators
- instance of operator
- Precedence and Associativity
- Interesting Questions

- **Arithmetic Operators:** They are used to perform simple arithmetic operations on primitive data types.
    - *** :** Multiplication
    - **/ :** Division
    - **% :** Modulo
    - **+ :** Addition
    - **– :** Subtraction

- **Unary Operators:** Unary operators need only one operand. They are used to increment, decrement or negate a value.
    - **– :Unary minus**, used for negating the values.
    - **+ :Unary plus**, indicates positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is byte, char, or short. This is called unary numeric promotion.
    - **++ :Increment operator**, used for incrementing the value by 1. There are two varieties of increment operator.
        - **Post-Increment :** Value is first used for computing the result and then incremented.
        - **Pre-Increment :** Value is incremented first and then result is computed.
    - **— : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operator.
        - **Post-decrement :** Value is first used for computing the result and then decremented.
        - **Pre-Decrement :** Value is decremented first and then result is computed.
    - **! : Logical not operator**, used for inverting a boolean value.

- **Assignment Operator : '='** Assignment operator is used to assign a value to any variable. It has a right to left associativity, i.e value given on right hand side of operator is assigned to the variable on the left and therefore right hand side value must be declared before using it or should be a constant.
  General format of assignment operator is,

variable = value;

1. In many cases assignment operator can be combined with other operators to build a shorter version of statement called **Compound Statement**. For example, instead of a = a+5, we can write a += 5.

- **+=**, for adding left operand with right operand and then assigning it to variable on the left.
- **-=**, for subtracting left operand with right operand and then assigning it to variable on the left.
- ***=**, for multiplying left operand with right operand and then assigning it to variable on the left.
- **/=**, for dividing left operand with right operand and then assigning it to variable on the left.
- **%=**, for assigning modulo of left operand with right operand and then assigning it to variable on the left.

2. **Relational Operators :** These operators are used to check for relations like equality, greater than, less than. They return boolean result after the comparison and are extensively used in looping statements as well as conditional if else statements. General format is,

variable **relation_operator** value

- Some of the relational operators are-
    - **==, Equal to :** returns true if left hand side is equal to right hand side.
    - **!=, Not Equal to :** returns true if left hand side is not equal to right hand side.
    - **<, less than :** returns true if left hand side is less than right hand side.
    - **<=, less than or equal to :** returns true if left hand side is less than or equal to right hand side.
    - **>, Greater than :** returns true if left hand side is greater than right hand side.
    - **>=, Greater than or equal to:** returns true if left hand side is greater than or equal to right hand side.

- **Logical Operators :** These operators are used to perform "logical AND" and "logical OR" operation, i.e. the function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e. it has a short-circuiting effect. Used extensively to test for several conditions for making a decision.
  Conditional operators are-
    - **&&, Logical AND :** returns true when both conditions are true.
    - **||, Logical OR :** returns true if at least one condition is true.
    - **!** not operator

- **Ternary operator :** Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary. General format is-

condition **?** if true **:** if false

Bitwise operators in Java: -

Bitwise operators are used to performing manipulation of individual bits of a number. They can be used with any of the integral types (char, short, int, etc). They are used when performing update and query operations of Binary indexed tree.

1. **Bitwise OR (|) –**
   This operator is a binary operator, denoted by '|'. It returns bit by bit OR of input values, i.e, if either of the bits is 1, it gives 1, else it gives 0.
   For example,

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise OR Operation of 5 and 7
 0101
| 0111
 _____
 0111 = 7 (In decimal)

2. **Bitwise AND (&) –**
   This operator is a binary operator, denoted by '&'. It returns bit by bit AND of input values, i.e, if both bits are 1, it gives 1, else it gives 0.
   For example,

a = 5 = 0101 (In Binary)

b = 7 = 0111 (In Binary)

Bitwise AND Operation of 5 and 7

 0101

& 0111

 _____

 101 5 (In decimal)

3. **Bitwise XOR (^) –**
   This operator is a binary operator, denoted by '^'. It returns bit by bit XOR of input values, i.e, if corresponding bits are different, it gives 1, else it gives 0.
   For example,

a = 5 = 0101 (In Binary)

b = 7 = 0111 (In Binary)

Bitwise XOR Operation of 5 and 7

0101

^ 0111

_____

10 2 (In decimal)

4. **Bitwise Complement (~) –**
   This operator is a unary operator, denoted by '~'. It returns the one's complement representation of the input value, i.e, with all bits inverted, which means it makes every 0 to 1, and every 1 to 0.
   For example,

a = 5 = 0101 (In Binary)

Bitwise Complement Operation of 5

~ 0101

_____

1010 = 10 (In decimal)

Java other special Operators:-

The **dot (.)** operator is used to select members of a class or object instance:

```
...
int i = myObject.length;  //Retrieve the value of an instance variable
                    //(of an object) or a static variable (of a class).
myObject.someMethod( );   //A method to be invoked on an object or class
...
```

The **new** operator is used to create objects out of a class:

```
...
Object o = new Object( );         // o has reference to new object
                          // of Object type (class)
int hours = new Date().getHours(); // create a new object and invoke a
                          // method in it without assigning the object
...
```

The i**nstanceof** operator is used to test whether an object is an instance of a class:

```
...
Boolean b;
String str = "foo";
b = ( str instanceof String ); // true, str is a String
b = ( str instanceof Object ); // also true, as String is an Object
b = ( str instanceof Date );   // false, str is not a Date or subclass
                          // of Date
...
```

JAVA AND THE VIRTUAL MACHINE

- Many programming tools, like C++, need to compile the source code into an intermediary form, object-file. This process is called compiling. This object-file will then be linked with other object-files into an executable program with help of a Linker program.
- Other types of programming tools, such as JavaScript, called an interpreter, meaning that the compile and execution is done at the same time as the program reads the source code.
- Java is both a compiled and an interpreted programming language.
- **Java source code files are compiled into a format called bytecode, which can then be executed by a Java interpreter.**
- The interpreter can be run as a separate program, or it can be embedded in another software, such as a browser.
- The compiled bytecode can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist in most operating systems, including UNIX, the Macintosh OS, and Windows.

COMPILING AND RUNNING JAVA PROGRAMS:

- Java JDK is a free software that can be downloaded from Oracle's website.
- Inside the JDK-software you will find, among others, the following tools:

Tools	Use
javac	Java compiler, creates bytecode
java	Java interpreter, used to run compiled program
appletviewer	Used for displaying the applet as it would be seen by the browser
jdb	Java debugger
javap	Decompiler
jar	Create jar files that contains a lot of bytecode files
javadoc	Documentation generator

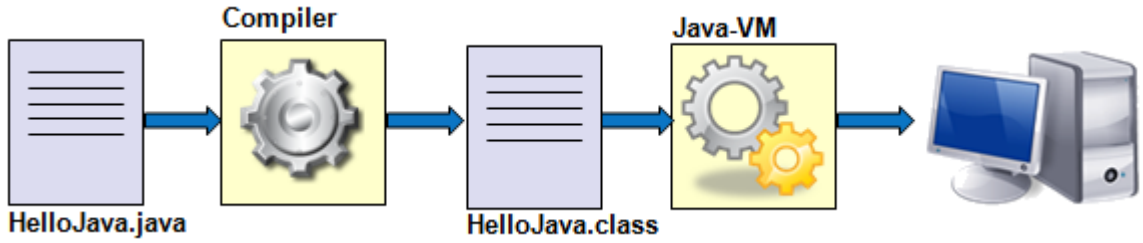
- The process of running a java application are at least:
    1. Create all Java source files where each file has a name with the .java extension.

2. Each Java source file must define at least a public class or interface with the same name as the file name (exclusive the extension).
3. If you are building a Java Application one of the Java source files must include a main() method which will bee the start point for your program.
4. We must then compile the Java source files with the **javac** tool to create .class files, which results in bytecode files.
5. A .class file will have the same name as the name of corresponding .java file.
6. At last we will run the program with the **java** interpreter.

Here is a HelloJava.java program example:

```
public class HelloJava
{
  public static void main( String[] args )
                {

                                System.out.println("Hello, Java! ");
      }
}
```



Conditional statement in java

if statement is used to test the condition. It checks boolean

condition: *true* or *false*. There are various types of if statement in Java.

1. if statement
2. if-else statement
3. if-else-if ladder
4. nested if statement

Java if Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

```
if(condition)
{
//code to be executed
}
```

Java if-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
if(condition)
{
//code if condition is true
}
else
{
//code if condition is false
}
```

Java if-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
```

```
//code to be executed if all the conditions are false
}
```

Java Nested if statement

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
   }
}
```

Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with int char

In other words, the switch statement tests the equality of a variable against multiple values.

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

Syntax

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
  code to be executed if all cases are not matched;
}
```

Java Switch Statement is fall-through

The Java switch statement is fall-through. It means it executes all statements after the first match if a break statement is not present.

Example:

```
//Java Switch Example where we are omitting the
//break statement
public class SwitchExample2 {
public static void main(String[] args) {
    int number=20;
    //switch expression with int value
    switch(number){
    //switch cases without break statements
    case 10: System.out.println("10");
    case 20: System.out.println("20");
    case 30: System.out.println("30");
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

Java Nested Switch Statement

We can use switch statement inside other switch statement in Java. It is known as nested switch statement.

Loops in Java:- The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is **fixed**, it is recommended to use for loop.

There are three types of for loops in Java.

1. for loop
2. while loop
3. do while

for loop:- for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

```
for (initialization; condition; increment/decrement)
{
   statement(s)
}
```

1. **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
2. **Testing Condition:** It is used for testing the exit condition for a loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
3. **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
4. **Increment/ Decrement:** It is used for updating the variable for next iteration.
5. **Loop termination:**When the condition becomes false, the loop terminates marking the end of its life cycle.

While loop: A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax :

```
while (boolean condition)
{
```

```
  loop statements...
}
```

- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**
- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

Do while loop:- do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

Syntax:

```
do
{
   statements..
}
while (condition);
```

1. do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
2. After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
3. When the condition becomes false, the loop terminates which marks the end of its life cycle.
4. It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

Infinite loop: non termination loop is called infinite loop