Introduction to OOPS: Introduction, Need of OOP, Principles of Object Oriented Languages, Procedural languages vs OOP, Applications of OOP, History of Java, JVM, Java Features, Programming Style, Command Line Arguments, Escape Sequence Comments
Data Types, Variables, Operators and Flow of Control: Variables, Primitive Data types, Constants, Identifiers- Naming Conventions, Keywords, Literals, Operators- Binary, Unary, Ternary, Expressions, Precedence rules and Associativity, Primitive Type Conversion and casting, Flow of Control- Branching, Conditional Loops.

Introduction

Java is a widely used programming language and is designed for the distributed environment of internet. It is a general-purpose programming language that is concurrent, class-based, and object-oriented. It is free to access and we can run it on all the platforms. Java follows the principle of WORA (Write Once, Run Anywhere), and is platform-independent. It is also simple and easy to learn.

Need of OOP, Principles of Object Oriented Languages

Some languages like C, Fortran and Pascal are not object-oriented programming languages and are called as procedural programming languages. They were having some disadvantages.

1) They were less secure.
2) There are no access modifiers in procedural programming.
3) Procedural programming does not have the concept of inheritance.
4) It gives importance to functions over data.
5) There is not any proper way for data hiding.
6) Code reusability is difficult in procedural programming.
7) There is no concept of virtual class.
8) Polymorphism, Abstraction and Encapsulation were not present in procedural languages.

What are the Principles of Object-Oriented Languages?

Abstraction

Abstraction is first of the object orientation principles. It is defined as hiding the complex details and presenting only with the relevant or necessary details. Abstraction is used to manage complexity while writing programs. Encapsulation and inheritance are used to implement abstraction while writing programs. Abstraction represents the external behavior of the object (what the object does) and encapsulation represents how it is implemented internally by the object.
As a real-world example for abstraction let's consider the car example:When you want to purchase a car, the car dealer gives you the details of available colors, seating capacity, engine, gearbox, type of steering

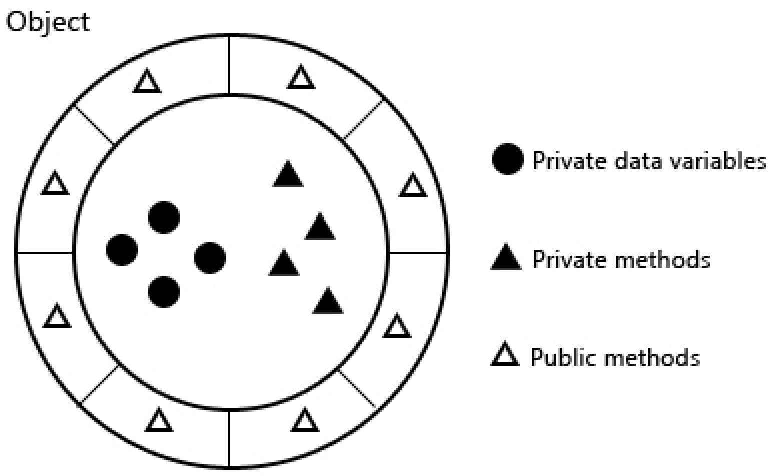
and other kinds of general details. The car dealer never tellsyou specific (complex) details of material being used for the hood,nuts and bolts and such.

Inst Panel & Interior parts
Rear Inner Fender(L&R)
Garnish molding
Roof Rail
Fr Bumper
Splash Shield
Front Pad Roof(L&R)
Cowl Top
Rear Tray
Rear Pad Roof(L&R)
Rr. Bumper
Under Cover (Device & Assist)
Enclosure(L&R)
Fuel Pipe Cover
Inner fender
Air duct parts

The above details are not needed. These details are hidden by the car dealer. Like this, you can think of other examples in the real world. Abstraction is everywhere.

Encapsulation

Encapsulation is the fundamental principle to achieve abstraction in object -oriented programming. Abstraction is conceptual and encapsulation is a way to implement abstraction. Wherever you can find abstraction you can say that encapsulation is there but encapsulation may not always provide abstraction. Encapsulation is defined as wrapping up or combining the data and code into a single unit. In the definition data refers to variables and code refers to methods (functions). Every object contains private data (variables) and public methods which allows other code to access the private data. Private methods are optional. We can view an object in OOP as shown below:



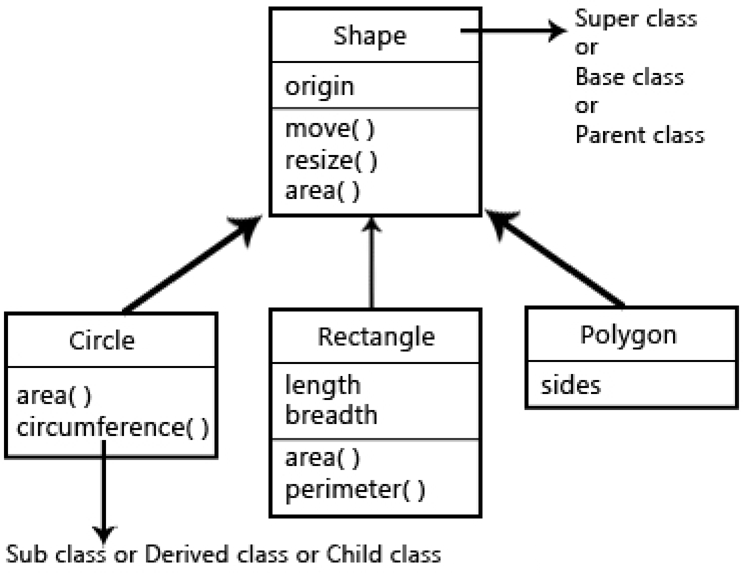
As a real-world example for encapsulation let's consider a television. We access the television by using a remote which provides an interface to communicate with the television. Television manufacturer is hiding its components (data) under the shell and is providing public interface (remote control) to access its components.

In our real-world example, television components are analogous to private member variables and public methods are analogous to buttons on remote control of the television.

Data hiding or information hiding is the side effect of encapsulation. Whenever we use encapsulation, we are implicitly hiding the data of an object which is known as data hiding. Encapsulation helps programmers to write code which is extensible.

Inheritance

Another way to implement abstraction is by using inheritance. In inheritance, entities are arranged as hierarchies. You might come across situations where even though objects have similar properties are quite different. In such cases, it is better to move all the common properties and behavior into one common entity and the specific properties and behavior into special or specific entity. Inheritance is defined as one object derives or inherits the properties and behavior of another object. Common properties and behavior are moved into a common class also known as super class or base class or parent class. Specific properties and behavior are moved into specialized classes also known as sub class or derived class or child class.



As a real-world example, consider a child inheriting the properties (land and other assets) from their parents. As another example, let's consider that we are developing a program to work with geometrical objects like circle, rectangle etc. In the application I can create these objects and perform operations like move, re size etc. These operations are common for all the objects. So, we can use inheritance in this context. It provides code reusability. The resultant hierarchy will be as shown below:

Polymorphism

Polymorphism means many forms. More generally it is defined as one interface multiple implementations. It means that based on the situation or different factors, the same thing exhibits multiple behaviors or the same thing can be used to carry out different tasks. Here thing is used as a general element.

As a real-world example for polymorphism, let's consider the animal Chameleon. This animal changes its color to reflect the surroundings to hide from its prey. It is really cool! Based on the color of the surroundings the same thing (Chameleon) can change its color from one to another. In every well written java program, we can find encapsulation, inheritance and polymorphism.

Procedural languages vs OOP:

Parameter	Procedural Programming	Object Oriented Programming
Definition	This programming language makes use of a step by step approach for breaking down a task into a collection of routines (or subroutines) and variables by following a sequence of instructions. It carries out each step systematically in order so that a computer easily gets to understand what to do.	This programming language uses objects and classes for creating models based on the real-world environment. This model makes it very easy for a user to modify as well as maintain the existing code while new objects get created by inheriting the characteristics of the present ones.
Security	Procedural Programming does not offer any method of hiding data. Thus, it is less secure when compared to Object Oriented Programming.	Hiding data is possible with Object Oriented Programming due to the abstraction. Thus, it is more secure than the Procedural Programming.
Division of Program	Procedural Programming divides the program into small programs and refers to them as functions.	Object Oriented Programming divides the program into small parts and refers to them as objects.
Approach	The Procedural Programming follows a Top-Down approach.	The Object Oriented Programming follows a Bottom-Up approach.
Inheritance	It does not provide any inheritance.	It achieves inheritance in three modes- protected, private, and public.
Overloading	The case of overloading isn't possible in the case of Procedural Programming.	Overloading is possible in the form of operator overloading and function overloading in the case of Object Oriented Programming.
Reusability of Code	No feature of reusing codes is present in Procedural Programming.	Object Oriented Programming offers the feature to reuse any existing codes in it by utilizing a feature known as inheritance.
Examples	Some common examples of Procedural Programming are C, Fortran, VB, and Pascal.	The examples of Object Oriented Programming languages are Java, C++, VB.NET, Python, and C#.NET.

History of Java

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". Java was developed by James Gosling, who is known as the father of Java, in 1995.

1) James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
2) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
3) Firstly, it was called "Greentalk" by James Gosling, and the file extension was .gt.
4) After that, it was called Oak and was developed as a part of the Green project.
5) Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
6) In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.
7) Java is an island in Indonesia where the first coffee was produced (called Java coffee).
8) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
9) JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

*Note: Below information is just for your reference.

Version	Year	Major Updates
JDK 1.0	January 1996	1st version
JDK 1.1	February 1997	Java Database connectivity (JDBC), Java Beans, RMI (Remote method Invocation)
J2SE 1.2	December 1998	Collection framework, swing graphical API was integrated into the core classes., Audio support in Applets.
J2SE 1.3	May 8, 2000	Jar Indexing, Java Platform Debugger Architecture (JPDA)
J2SE 1.4	February 2002	Integrated XML parser and XSLT processor (JAXP), JDBC 3.0 API, Image I/O API for reading and writing images in formats like JPEG and PNG, Regular Expressions, Internet Protocol version 6 (IPv6) support
J2SE 1.5	September 2004	Enhanced for Loop, Enumerations, Metadata

		(Annotations)
Java SE 6	December 2006	JDBC 4.0 API, Scripting Language Support, Improved Web Service support through JAX-WS
Java SE 7	July 2011	Strings in switch Statement, Catching multiple exception types, Timsort is used to sort collections and arrays of objects instead of merge sort
Java SE 8	March 2014	Lambda Expressions, Date and Time API
Java SE 9	September 2017	The Java Platform module system, jlink: The Java Linker, jshell: The Java Shell (Read-Eval-Print Loop)
Java SE 10	March 2018	Garbage Collector Interface, Heap Allocation on Alternative Memory Devices, Bytecode Generation for Enhanced for Loop
Java SE 11	September 2018	Enhanced KeyStore Mechanisms, Transport Layer Security (TLS) 1.3, Removal of Java Deployment Technologies
Java SE 12	March 2019	String Class New Methods, Compact Number Formatting
Java SE 13	September 2019	ZGC: Uncommit Unused Memory, Dynamic CDS Archives
Java SE 14	Mar 2020	Helpful NullPointerExceptions, Deprecate the Solaris and SPARC Ports
Java SE 15	September 2020	Sealed Classes, Pattern Matching Type Checks, Foreign Memory API
Java SE 16	March 2021	New Packaging Tool, Improved memory management, UNIX-Domain Socket Channels
Java SE 17	September 2021	Context-specific deserialization filters, Deprecation of the Security Manager, Removal of the experimental AOT and JIT compiler

---

Applications of OOP:

Java is widespread; the following are some of the areas in which we find java usable:

Desktop applications
Web applications
Mobile applications (Android)
Cloud computing

Enterprise applications
Scientific applications
Operating Systems
Embedded systems
Real-time software
Cryptography
Smart cards
Computer games
Web servers and application servers

History of Java

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". Java was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.
Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc.
Following are given significant points that describe the history of Java.
1) James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
2) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
3) Firstly, it was called "Greentalk" by James Gosling, and the file extension was .gt.
4) After that, it was called Oak and was developed as a part of the Green project.

Why Java was named as "Oak"?

5) Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.
6) In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.
7) The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say.

According to James Gosling, "Java was one of the top choices along with Silk". Since Java was so unique, most of the team members preferred Java than other names.
8) Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.
9) JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

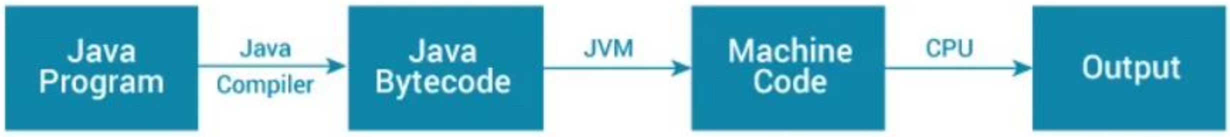
Java JDK, JRE and JVM

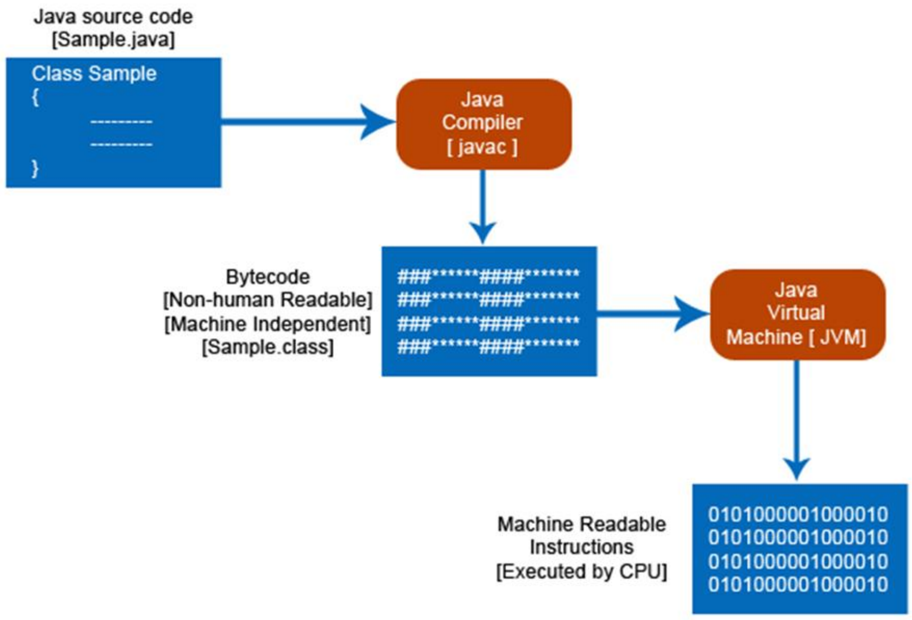
What is JVM?

JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.

When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

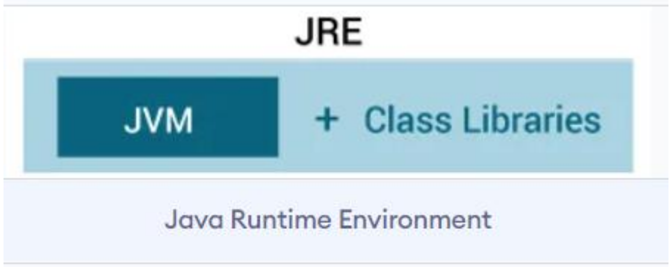
Java is a platform-independent language. It's because when you write Java code, it's ultimately written for JVM but not your physical machine (computer). Since JVM executes the Java bytecode which is platform-independent, Java is platform-independent.




Java source code [Sample.java]
Class Sample { --------- --------- }
Java Compiler [javac]
Bytecode [Non-human Readable] [Machine Independent] [Sample.class]
###******####*******
Java Virtual Machine [JVM]
Machine Readable Instructions [Executed by CPU]
0101000001000010

What is JRE?

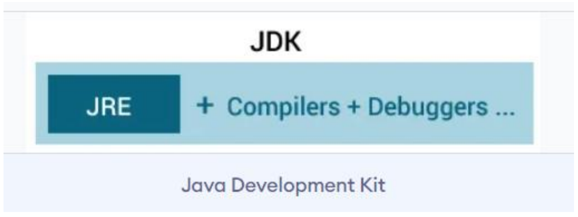
JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.



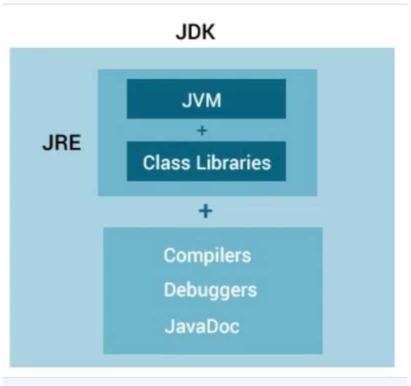
What is JDK?

JDK (Java Development Kit) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.

In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).



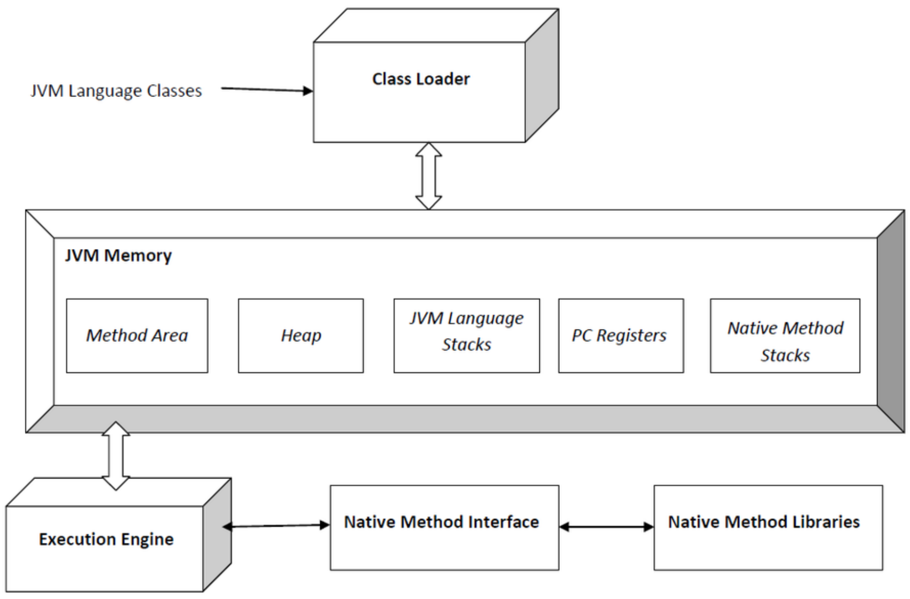
Relationship between JVM, JRE, and JDK.



Relationship between JVM, JRE, and JDK

JVM Architectute: refer to below link
https://dzone.com/articles/jvm-architecture-explained

Architecture of JVM.



1) **ClassLoader:** The class loader is a subsystem used for loading class files. It performs three major functions viz. Loading, Linking, and Initialization.
2) **Method Area:** JVM Method Area stores class structures like metadata, the constant runtime pool, and the code for methods.
3) Heap: All the Objects, their related instance variables, and arrays are stored in the heap. This memory is common and shared across multiple threads.
4) JVM language Stacks: Java language Stacks store local variables, and it's partial results.
5) PC Registers: PC register store the address of the Java virtual machine instruction which is currently executing.
6) Native Method Stacks: Native method stacks hold the instruction of native code depends on the native library.
7) Execution Engine: It is a type of software used to test hardware, software, or complete systems.
8) Native Method interface: The Native Method Interface is a programming framework. It allows Java code which is running in a JVM to call by libraries and native applications.
9) Native Method Libraries: Native Libraries is a collection of the Native Libraries (C, C++) which are needed by the Execution Engine.

How to execute a java program on a computer?

After we install Java on the computer, Let's learn how to execute a java program on a computer.
First step is to type the Java source code into a text editor and save it with .java extension.

```
class Hello
{
        public static void main(String[] args)
        {
                System.out.println("Hello World");
        }
`
```

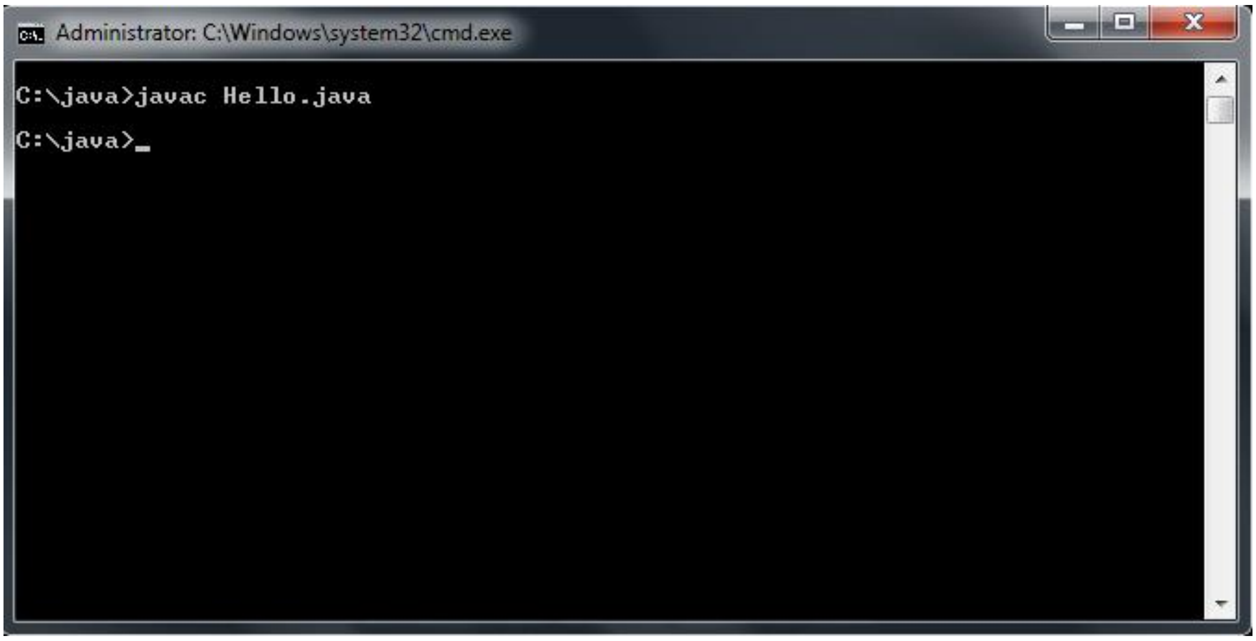
Above Java program prints Hello World on to the console (command prompt in Windows). After typing the above program save the file as Hello.java. Note that file name Hello is same as the class name.

Let's assume that the file is saved in the path C:/java. So, full path to the source file is C:/java/Hello.java.

Next step is to compile the source code file. Open the command prompt (black window) and navigate to the location C:/java. Type the following command to compile the java file:

javac Hello.java

If there are no errors in the Java program, the program compiles successfully and you will see the prompt again as shown below:



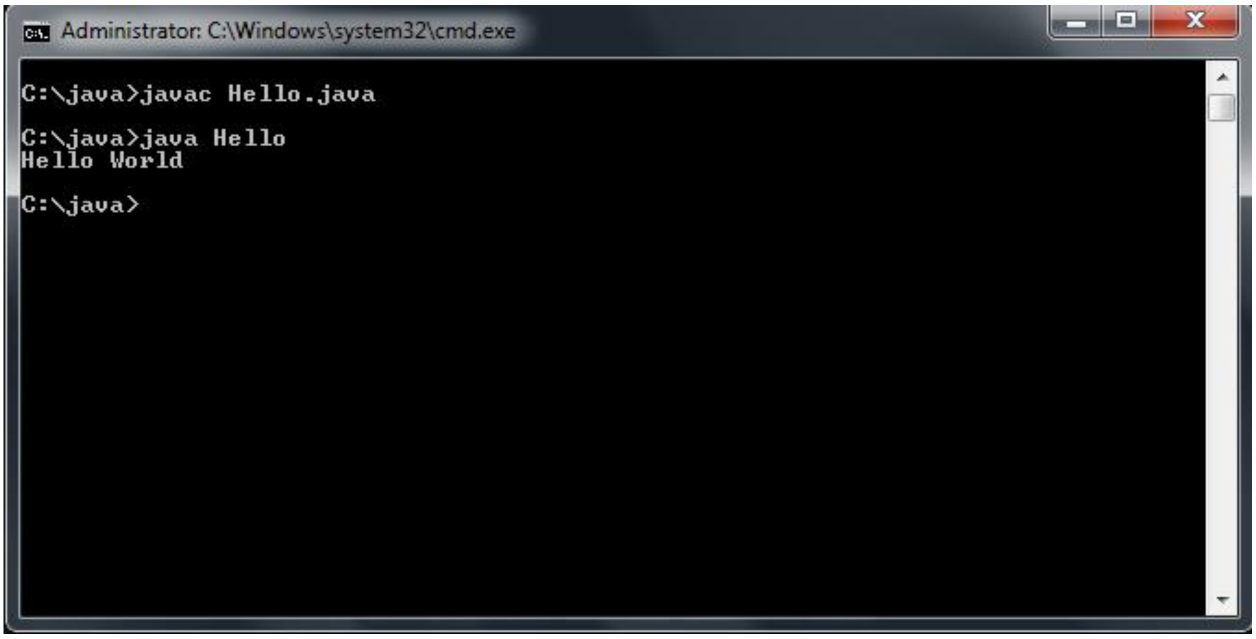
```
C:\java>javac Hello.java

C:\java>_
```

After compiling the program, you can see the generated file Hello.class which contains the bytecode in the location C:/java.

Next step is to execute the program. To execute the program, type the following command in the command prompt:

java Hello

Take care that .class is not added after the class name Hello. You can see the output Hello World as shown below:



```
C:\java>javac Hello.java

C:\java>java Hello
Hello World

C:\java>
```

Now that we have executed our first Java program.

What are the various elements in a java program? (or) Explain the structure of a java program

Public: It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program. If we use private, protected, and default before the main() method, it will not be visible to JVM.

```
class Hello
{
        public static void main(String[] args)
        {
                System.out.println("Hello World");
        }
}
```

static: You can make a method static by using the keyword static. We should call the main() method without creating an object.

Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.

Void: In Java, every method has the return type. Void keyword acknowledges the compiler that main() method does not return any value.

String[] args: The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array. It is used to hold the command line arguments in the form of string values.

Here, agrs[] is the array name, and it is of String type. It means that it can store a group of string. Remember, this array can also store a group of numbers but in the form of string only. Values passed to the main() method is called arguments. These arguments are stored into args[] array, so the name args[] is generally used for it.

Class: Class keyword is used to declare a class in Java.

System.out.println:It's normal in a program to do computations, reading data from the user and printing the results. Reading data from the user in the console window (command prompt) is called Standard Input and printing/writing data to the console is called Standard Output.

In the above Hello World program, I am printing to the console by using printlnmethod which is provided by the out object of PrintStream class and the out object is made accessible by System class. Let's say that whenever we want to print something on to the console, we have to use System.out.println.

System and PrintStream are predefined classes which are provided by Java. All these predefined classes come bundled in packages. A package is a group of related classes. System class is available in java.lang package and PrintStream is available in java.io package.

Features of Java Programming

Simple
Object-oriented
High-Performance
Secure
Architecture – Neutral
Portable
Platform Independent
Robust
Dynamic
Multithreaded
Distributed

1)Simple:

Java designers built the language in such way that it is very easy to write programs using Java's syntax.It is simple and easy to read and write.

2) Object-oriented

Java is a object oriented programming language which allows the users to solve the given problem in terms of classes and objects. Writing the solution in terms of classes and objects produces code which is easier to write, maintain and reuse.

3)High-Performance

Many virtual machines use a JIT (Just-In-Time) compiler that converts the bytecode into platform-specific instructions as the program executes. This helps to improve the performance of the Java programs.

4)Secure

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:There is No explicit pointer,Java Programs run inside a virtual machine sandbox. This prevents malicious programs to gain access to sensitive information available on the user's system.

5)Architecture – Neutral

Java compiler generates bytecode which when run on any machine gives the same output. Java's goal is to enable programmers to develop programs which has WORA (Write Once, Run Anywhere) property.

6)Portable

Java is portable because Java code can be executed on all the major platforms. Once you've compiled your Java source code to bytecode, those files can be used on any Java-supported platform without modification

7)Platform Independent

Platform independent means that the java source code can run on multiple operating systems. Java code should run on any machine that doesn't need any special software to be installed. Java code is compiled into bytecode, which is platform-independent. we can run it on Windows, Linux, Mac OS, etc.

8)Robust

Java is robust because:

- ✓ It uses strong memory management.
- ✓ There is a lack of pointers that avoids security problems.
- ✓ Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- ✓ There are exception handling and the type checking mechanism in Java. All these points make Java robust.

9)Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.
Java supports dynamic compilation and automatic memory management (garbage collection).

10)Multithreaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

11)Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications.
Remote Method Invocation (RMI) which enables a program to access methods across a network.

Comments

Writing comments is considered a good programming practice. They will help the client programmers to understand the programs developed by you.
Java comments can be a single line or multiple lines of text which are written by the programmers for documentation purpose. A programmer can write comments to mention the purpose of the program or a part of the program or to provide authorship details.

Need for comments

Following are some of the famous reasons for writing comments in programs:

- Provide the purpose of the program along with sample input and output of the program.
- Provide authorship details.
- Mention date and time of creation of the program and when it was last modified.
- Provide details about several parts of the program.
- For generating documentation which will be helpful for other programmers.

Types of comments

In Java, there are three types of comments. They are:

- Single line comments
- Multi-line comments
- javadoc comments

A single line comment as the name implies is a comment containing a single line of text. A single line comment starts with //. All the characters after the // to the end of the line is treated as a comment.
Syntax for single line comment is as shown below:

```
// This is a single line comment
```

To write a comment containing more than one line, we can use a multi-line comment. A multi-line comment starts with /* and ends with */.
Syntax for writing a multi-line comment is as shown below:

```
/*  This is a
multi-line
comment */
```

We can use a multi-line comment for writing a single line comment as shown below:

```
/* This is a single line comment. */
```

Both single line comments and multi-line comments can be written anywhere in the program.
The third type of comments called javadoc comments can be used to generate HTML documentation which can be viewed using a browser. This can help the client programmers to understand the classes that you create. A javadoc comment starts with /** and ends with */. Syntax for writing a javadoc comment is as shown below:

```
/**
* This is a
* javadoc comment
*/
```

Note that * is not necessary before each line of text. It is widely accepted usage of javadoc comments by Java developers. Generally javadoc comments are written at the top of the program.

Java Variables

A variable is a location in memory (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier).

It is common in programs to store data and process the data to get the required outcome.

Creating or declaring variables in Java:

A variable can be created or declared in Java by using the below syntax:

datatype variable_name;

A variable_name can be any valid identifier in Java. Before using any variable in a Java program, you have to declare it first as per the syntax given above. The datatype specifies the type of value you are going to store in the variable.

Example for declaring a variable in Java is shown below:

int x;

In the above example, int is a Java's primitive data type and x is the name of the variable.

Initializing variables in Java:

After declaring variables, we can initialize them as shown below:

datatype variable_name;

variable_name = value;

So, we can initialize the variable x as shown below:

```
int  x;
x = 10;
```

For initializing a variable, we use the assignment (=) operator. We can combine variable declaration and initialization into a single line as shown below:

```
int  x = 10;
```

Initializing variables with literal values like 10, 2.5, etc… is known as **static initialization.** We can also assign an expression to a variable as shown below:

```
int a = 10;
int b=20;
int c = a+b;
```

The expression a+b will be evaluated at runtime (execution of the program) and then it will be assigned to the variable c. This type of initialization is know as **dynamic initialization.**

We can declare multiple variables of the same type as shown below:

```
int  x, y, z;
```

```
//Program to demonstrate variables
import java.util.Scanner;
class Sample
{
   public static void main(String[] args)
        {
              int x, y;
              Scanner input = new Scanner(System.in);
              System.out.println("Enter value of x: ");
              x = input.nextInt();
              System.out.println("Enter value of y: ");
              y = input.nextInt();
              System.out.println("Sum of x and y is: "+(x+y));
        }
}
```

- ✓ In the above program Scanner is a utility class available in java.util package to read data from various sources like standard input, files etc…
- ✓ To read input from console (standard input) we provide System.in as a parameter to the Scanner's constructor.
- ✓ nextInt() method available in the Scanner class returns the input entered by the user as an integer.
- ✓ The variables x and y are dynamically initialized.

Output of the above program is:

```
Enter value of x:
10
Enter value of y:
20
Sum of x and y is: 30
```

Types of variables:

Based on the location where the variable is declared and how it is declared, variables in Java are divided into three types. They are:

- Instance variables
- Class variables
- Local variables

Instance Variables: A variable which is declared inside a class and outside all the methods, constructors and blocks is known as an instance variable.

Class Variables: A variable which is declared inside a class and outside all the methods, constructors, blocks and is marked as static is known as a class variable. More on static keyword in another article.

Local Variables: Any variable which is declared inside a class and inside a block, method or a constructor is known as a local variable.

Following Java example program demonstrates all the three kinds of variables:

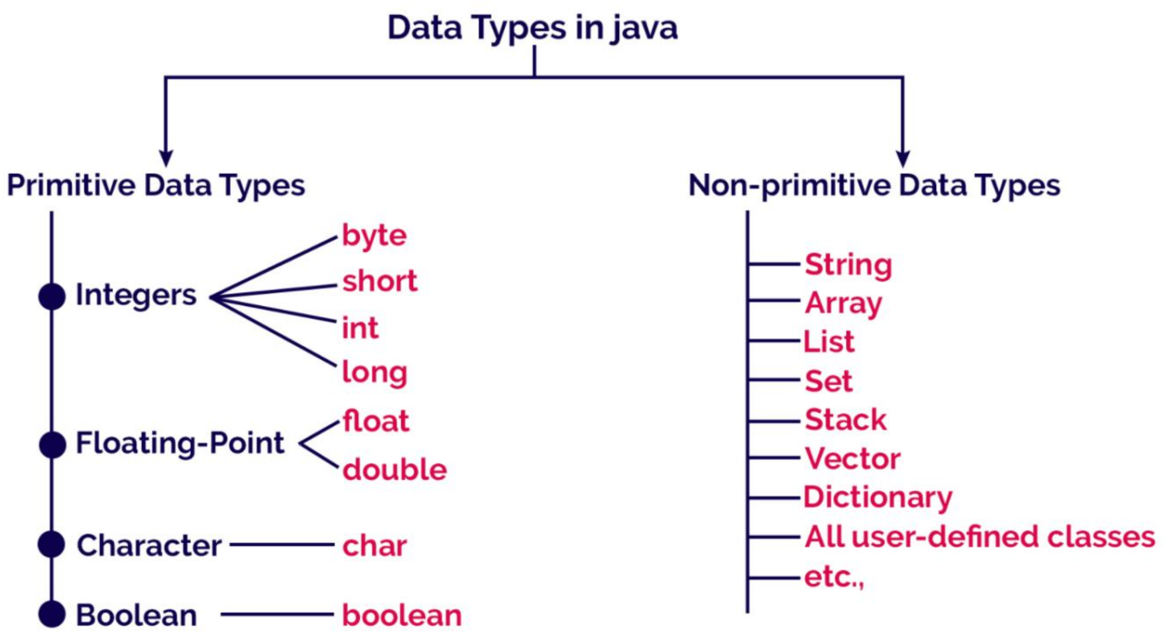
```
class Sample
{
        int x, y;
        static int result;
        void add(int a, int b)
        {
                x = a;
                y = b;
                int sum = x+y;
                System.out.println("Sum = "+sum);
        }
   public static void main(String[] args)
        {
                Sample obj = new Sample();
                obj.add(10,20);
        }
}
```

In the above program x and y are instance variables, result is a class variable; a, b, sum and args are local variables.

One important point to remember is, every object maintains its own copy of each instance variable and a shared copy of each class variable.

Data Types

A data type specifies the type of value a variable can store or the type of an expression. Java is a strongly typed language means that you should specify the type of a variable before using it in a Java program.



However the data types are mainly of two categories:

a. Primitive Data Types- Java primitive data types are the ones which are predefined by the programming language which in this case is Java. Without primitive data types it would be impossible to frame programs. Primitive data types are also the building blocks of Non-primitive data types. Examples are- int,float etc.

b. Non-Primitive Data Types- These data types are special types of data which are user defined, Some examples are- classes, interfaces etc.

Primitive Data Types

There are eight primitive types in Java namely: byte, short, long, int, float, double, char and boolean.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Command Line Arguments

Sometimes we might want to pass extra information while running a Java program. This extra information passed along with the program name are known as command line arguments.
These command line arguments are separated by white spaces.
Command line arguments can be accessed using the string array specified in the main function's signature.
For example, if the array name is args, then the first command line argument can be accessed as args[0] and the second command line argument can be accessed as args[1] and so on.
Let's see the following Java program which access two command line arguments, multiplies them and displays the result to the user:

```
class CommandArgs
{
public static void main(String[] args)
{
int x = Integer.parseInt(args[0]);
int y = Integer.parseInt(args[1]);
int mult = x * y;
System.out.println("Multiplication of the command line arguments is: " + mult);
}
}
```

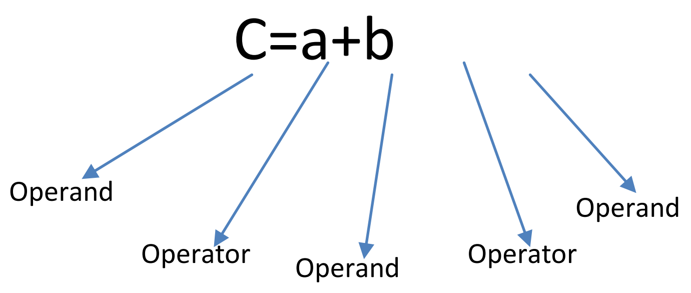
Output:

```
javac  CommandArgs.java
java CommandArgs 2 4
Multiplication of the command line arguments is:8
```

What do you mean by operators? What are different kinds of operators in java?

Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

In other words, an operator allows the programmer or the computer to perform an operation on the operands. An operand can be a literal, variable or an expression.



Arithmetic Operator:

They are used to perform basic mathematical operations like addition, subtraction, multiplication and division.

Some of the arithmetic operators are	+ , - , * , / , % , ++ , --

Points to note:

1) % is for reminder division and returns the reminder part.

2) ++x is pre-increment and x++ is post increment

3) Similarly. --x is pre-decrement and x-- is post decrement.

1) Post-Increment (y++): we use y++ in our statement if we want to use the current value, and then we want to increment the value of y by 1.

2) Pre-Increment(++x): We use ++x in our statement if we want to increment the value of x by 1 and then use it in our statement.

Similarly for decrement operator too.

```
public class Main
{
  public static void main(String[] args)
  {
    int x = 5;
    int y = 3;
    int x1=5;
    int y1=3;
System.out.println("Addition is "+(x + y));
System.out.println("Substraction "+(x - y));
System.out.println("Multiplication "+(x * y));
System.out.println("Division "+(x / y));
System.out.println("Reminder division "+(x % y));
System.out.println("pre-Increment "+ (++x));
System.out.println("Post- Increment"+(y++));
System.out.println("pre-Decrement "+ (--x1));
System.out.println("Post- decrement "+(y1--));
  }
}
```

Output of the above code is:

```
Addition is 8
Substraction 2
Multiplication 15
Division 1
Reminder division 2
pre-Increment 6
Post- Increment3
pre-Decrement 4
Post- decrement 3
```

Assignment Operators

Assignment operators are used to assign values to variables. In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

```
public class AO {
  public static void main(String[] args) {
    int x = 10;
System.out.println(x);
  }
}
```

Output of the above code is

```
10
```

We can write short hand assignment operator as follows:

Operator	Example	Same As

Operator	Name	Example
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

A Sample program demonstrating shorthand assignment operators.

```
public class Short_Assign
{
  public static void main(String[] args)
{
    int x = 10;
    x += 5;
System.out.println(x);
  }
}
```

Output of above code is:

```
15
```

Java Comparison/ Relational Operators

Comparison operators are used to compare two values.The result of comparing two values is always a boolean value true or false.Unlike C language, true doesn't refer any positive value other than zero and false doesn't refer zero.Comparison operators in java are:

==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Program demonstration operators in java are:

```
public class Main{
          public static void main(String[] args){
          int x=3;
      int y=5;
System.out.println(x==y); // Equal to
System.out.println(x!=y); // Not equal to
System.out.println(x<y); // Less than
System.out.println(x>y); // Greater than
System.out.println(x<=y); // less than or equal to
System.out.println(x>=y);// Greater than or equal to
      }
    }
```

Output of the above code is :

```
false
true
true
false
true
false
```

Java Logical Operators

Logical operators are used to check whether an expression is true or false. They are used in decision making.

These operators are used to perform logical "AND", "OR" and "NOT" operation.

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x< 10
| |	Logical or	Returns true if one of the statements is true	x < 5 | | x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Note that the logical && operator doesn't check the second condition if the first condition is false.

Sample program demonstrating logical operators:

```
public class Logical_operator
{
          public static void main(String[] args)
{
   int x=3;
booleana,b,c;
   a=x<5 && x>2;//logical AND
   b=x>5 || x>2; //logical NOT
   c=!(x<5 && x>2); // logical NOT
System.out.println(a);
System.out.println(b);
System.out.println(c);
   }
  }
```

```
true
false
```

Java Bitwise operator

Bitwise operators are used to performing manipulation of individual bits of a number.

Bitwise operator works on bits and performs the bit-by-bit operation. Assume if a = 5 and b = 7; now in binary format they will be as follows.

```
a=0101
b=0111
-----------------
a&b0101=5
a|b    0111    =7
a^b0010    =2
~b     1000    =-8
<<a    1010    =a (10)
>>a    0010    =2
```

Operator	Description
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.

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

    int number1 = 5, number2 = 7, result,result1, result2, result3, result4, result5;


    result = number1 | number2; //OR
    result1=number1 & number2; //AND
    result2=number1 ^ number2; //XOR
    result3= ~number2; //Complement
    result4= number1 << 1; //Left shift
    result5=number1 >> 1; //Right shift
System.out.println(result);
System.out.println(result1);
System.out.println(result2);
System.out.println(result3);
System.out.println(result4);
System.out.println(result5);

  }
}
```

Output:

```
7
5
2
-8
10
2
```

Operators can also be divided based on the number of operands on which they operate.

1) Unary operators:

The Java unary operators require only one operand.

2) Binary operators

The java binary operator requires 2 operands

3) Ternary operators

Java ternary operator is the only conditional operator that takes three operands. It's a one-line replacement for if-then-else statement and used a lot in Java programming.

Operator type	Operator	Example
Unary operators	++ (Increment) -- (Decrement) ! (Logical complement operator) ~ (Bitwise complement- Negation) << and >> (Bitwise Shift operator)	++a b-- !a ~a <<a
Binary operators	+,-,*,/ (Arithmetic operators) <,><=,>=,== (Relational operators) && , || (Logical operator &,|, ^ (Bitwise operators)	a+b a<b A&&b A&b
Ternary operator	? , :	(a<b)? "hi": "go"

What is Primitive Type Conversion and Casting? (or) What is the difference between implicit and explicit type conversion? (or) what is the difference between widening and Narrowing type casting?

Converting a value from one data type to another data type is known as type conversion.

There are 2 types of type conversion/casting:

1) Implicit type conversion/Automatic conversion/coercion/Widening Casting

2) Explicit type conversion/ Manual conversion/casting/narrowing casting.

Implicit type conversion/Automatic conversion /casting down / Widening 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.

byte -> short -> char -> int -> long -> float -> double

Lower datatype — Higher datatype

for example, a variable of **int** datatype is automatically converted into float.

```
public class Main
{
        public static void main(String[] args)
{
        int myint=3;
        float myfloat=myint;
        System.out.println(myint);
        System.out.println(myfloat);
  }
}
```

Output of above code is:

```
3
3.0
```

Explicit type conversion/ Manual conversion /casting up / narrowing 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. Syntax for explicit type conversion is,

(Destination datatype) Variable

double -> float -> long -> int -> char -> short -> byte

Higher Datatype

Lower datatype

Here, a variable of type **double**is converted into int.

In the below example, a variable **mydouble**of type doubleis forcefully converted into integer datatype. This is done by specifying the desired datatype (in our case int) in brackets before the variable (my double of type double).

```
public class Main
{
        public static void main(String[] args)
{
                double mydouble=9.9999948392d;
                int myint;
                myint=(int)mydouble;
                System.out.println(mydouble);
                System.out.println(myint);
  }
}
```

Output of above code is:

```
9.9999948392
9
```

Why type conversions are named widening and narrowing?

Since a smaller range type is converted into a larger range type this conversion is also known as widening conversion.

Since we are converting a source type having larger size into a destination type having less size, such conversion is known as narrowing conversion.

Expressions, Precedence rules and Associativity

What do you mean by java expressions? Explain with an example

An expression is made up of literals (Constants), variables, and operators.Every expression consists of at least one operator and an operand. Operand can be either a literal or a variable.

Consider the below code:

```
public class Main
{
        public static void main(String[] args)
  {
  int a=5, b=6,c;
  c= (a*a)+(b*a)+(a+b)-a;
System.out.println("The result is");
System.out.println(c);
  }
}
```

The output of the above code:

```
The result is
61
```

In the above code, ***(a*a)+(b*a)+(a+b)-a*** is an expression.

a and **b** are **operands** and **+,-,*** are **operators**

How are expressions evaluated?

The expressions are evaluated based on:

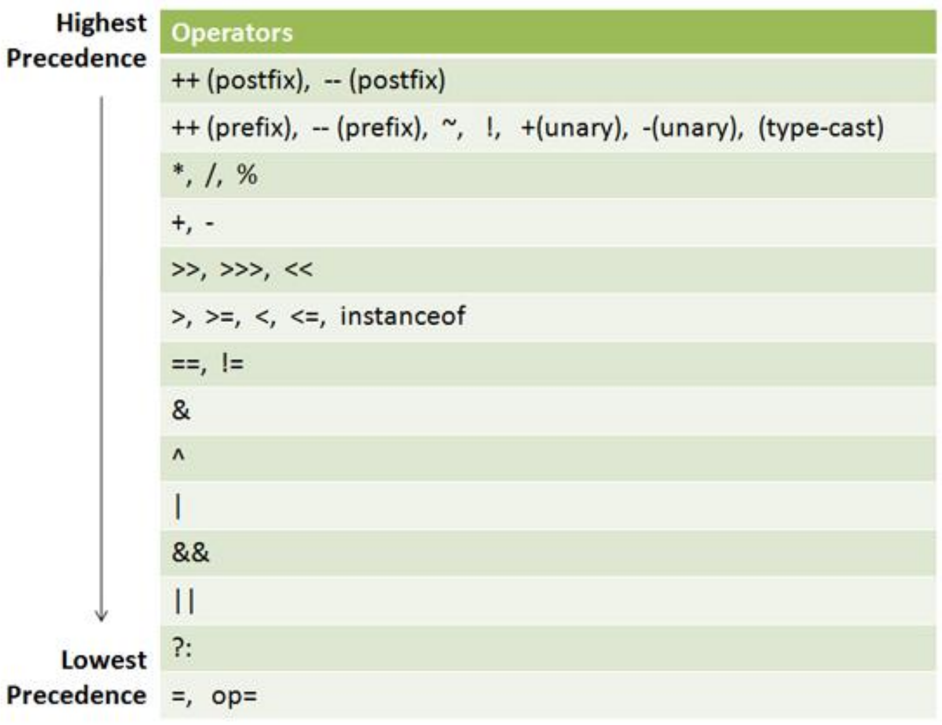
- Type promotion rules
- Operator precedence
- Associativity rules

Type promotion rules means type casting. (Note: type casting is explained in previous question)

Operator precedence

Operator precedence determines the order in which the operators in an expression are evaluated.

All the operators in Java are divided into several groups and are assigned a precedence level. The operator precedence chart for the operators in Java is shown below:



Operators
++ (postfix), -- (postfix)
++ (prefix), -- (prefix), ~, !, +(unary), -(unary), (type-cast)
*, /, %
+, -
>>, >>>, <<
>, >=, <, <=, instanceof
==, !=
&
^
|
&&
||
?:
=, op=

Highest Precedence
Lowest Precedence

Now let's consider the following expression.

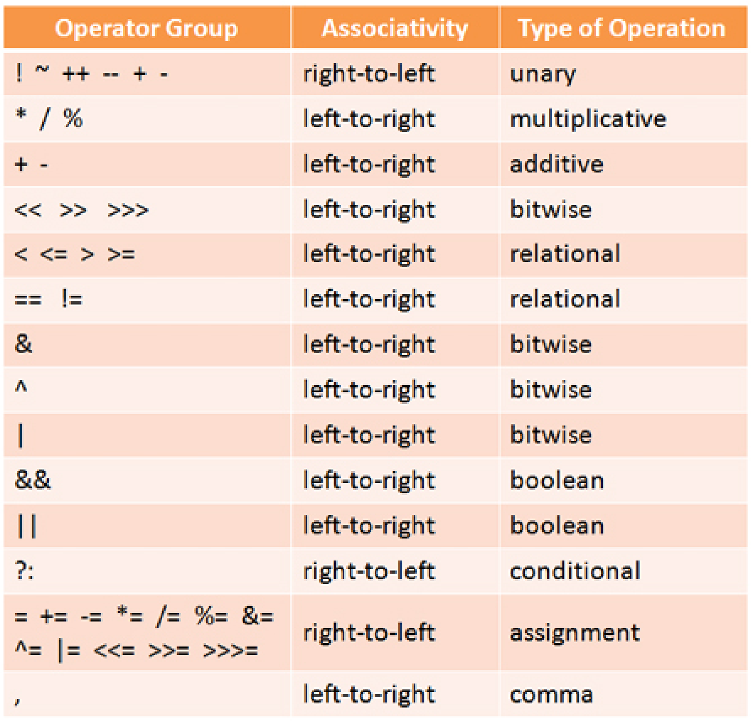
10 – 2 * 5

Based on the operator precedence chart shown above, * has higher precedence than +. So, 2* 5 is evaluated first which gives 10 and then 10 – 10 is evaluated which gives 0.

Associativity Rules

What if the expression contains two or more operators from the same group? Such ambiguities are solved using the associativity rules.

When an expression contains operators from the same group, associativity rules are applied to determine which operation should be performed first. The associativity rules of Java are shown below:



Operator Group	Associativity	Type of Operation
! ~ ++ -- + -	right-to-left	unary
* / %	left-to-right	multiplicative
+ -	left-to-right	additive
<< >> >>>	left-to-right	bitwise
< <= > >=	left-to-right	relational
== !=	left-to-right	relational
&	left-to-right	bitwise
^	left-to-right	bitwise
|	left-to-right	bitwise
&&	left-to-right	boolean
||	left-to-right	boolean
?:	right-to-left	conditional
= += -= *= /= %= &= ^= |= <<= >>= >>>=	right-to-left	assignment
,	left-to-right	comma

Now, let's consider the following expression:

10-6+2

In the above expression, the operators + and – both belong to the same group in the operator precedence chart. So, we have to check the associativity rules for evaluating the above expression. Associativity rule for + and – group is left-to-right i.e, evaluate the expression from left to right. So, 10-6 is evaluated to 4 and then 4+2 is evaluated to 6.

Use of parenthesis in expressions

Let’s look at our original expression example:

(20 * 5) + (10 / 2) – (3 * 10)

You might think that, what is the need of parenthesis (and) in the above expression. The reason we had included them is, parenthesis have the highest priority (precedence) over all the operators in Java.

So, in the above expression, (20*5) is evaluated to 100, (10/2) is evaluated to 5 and (3*10) is evaluated to 30. Now, our intermediate expression looks like:

100 + 5 – 30

Now, we can apply the associativity rules and evaluate the expression. The final answer for the above expression is 75.

Flow of Control-Branching, Conditional Loops.

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

1. Decision Making statements
   - Simple if statement
   - if-else statement
   - if-else-if ladder
   - Nested if-else statements
   - Ternary operator
   - switch statement
2. Loop statements
   - do while loop
   - while loop
   - for loop
   - for-each loop
3. Jump statements

- break statement
- continue statement

Decision making statements

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false.

Simple if Statement

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
if(condition) {
statement 1; //executes when condition is true
}
```

Consider the following example in which we have used the if statement in the java code.

```
public class Abc
{
        public static void main(String[] args)
        {
                int a=8;
                if(a%2==0)
                {
                        System.out.println("Hello");
                }
        }
}
```

Output of above code:

```
Hello
```

In the above code, print statement is executed only if ***if***condition is satisfied.

The if-else statement

It is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax of if-else statement is given below:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
```

}

Consider the following example in which we have used the if-else statement in the java code.

```
public class Abc
{
        public static void main(String[] args)
   {
        int a=7;
        if(a%2==0)
         {
                System.out.println("Given number is even");
         }
        else
        {
                System.out.println("Given number is odd");
         }

    }
}
```

Output of the above code is:

```
Given number is odd
```

In the above code, the ***if*** part will not be executed. Instead, else part is executed.

if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements where the program may enter in the block of code where the condition is true.

Syntax of if-else-if statement is given below.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

Consider the following example in which we have used the if-else-if statement in the java code.

```
public class Main
{
        public static void main(String[] args)
  {
                int a=5;
                if(a<5)
                {
                        System.out.println("a is less than 5");
                }
                else if(a==5)
                {
                        System.out.println("a is equal to 5");
                }
                else
                {
                        System.out.println("a is greater than 5");
                }
   }
}
```

Output of above code is:

```
"a is equal to 5"
```

Nested if

In nested if-statements, the if statement can contain a if or if-else statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
if(condition 1)
{
        statement 1; //executes when condition 1 is true
        if(condition 2)
        {
                statement 2; //executes when condition 2 is true
        }
        else
        {
        statement 2; //executes when condition 2 is false
        }
}
```

Consider the following example in which we have used the Nested if-else statement in the java code.

```
public class Main
{
   public static void main(String args[])
   {
      int i = 10;

      if (i == 10)
      {
         if (i< 15)
                  System.out.println("i is smaller than 15");

                  if (i< 12)
                           System.out.println("i is smaller than 12 too");
                  else
                           System.out.println("i is greater than 15");
      }
   }
}
```

Output of the above code is:

```
i is smaller than 15
i is smaller than 12 too
```

Short Hand If...Else (Ternary Operator)

There is also a short-hand if else, which is known as the ternary operator because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

Syntax of short hand if..else:

variable = (condition) ?expressionTrue : expressionFalse;

Consider the following example in which we have used short hand if.. else in the java code.

```
public class Main
{
         public static void main(String[] args)
   {
         int a=20;
         String answer;
    answer=(a<10)? "Number less than 10": "Number greater than 10";
         System.out.println(answer);
   }
}
```

Output of above code is:

```
"Number greater than 10"
```

Switch Statement

Switch statement is used to select one of many code blocks to be executed.

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- When Java reaches a break keyword, it breaks out of the switch block.

The syntax to use the switch statement is given below.

```
switch (expression){
   case value1:
    statement1;
    break;
   .
   .
   case valueN:
statementN;
    break;
   default:
    default statement;
}
```

Consider the following example in which we have used the switch statement in the java code.

```
public class Main
{
            public static void main(String[] args)
  {
  float a=5.0f;
  float b=6.0f;
  int operation=5;

  switch(operation)
  {
            case 1:
System.out.println("addition is "+ (a+b));
      break;

      case 2:
System.out.println("Substraction is " + (a-b));
      break;

      case 3:
System.out.println("Multiplication is "+ (a*b));
      break;

      case 4:
System.out.println("Division is "+ (a/b));
      break;
      case 5:
System.out.println("Reminder division "+ (a%b));
      break;
      default:
System.out.println (" Give correct choice");
  }
  }
}
```

Output of above code is:

Reminder division 5.0

Loop statements/Iterative Statements

Loops can execute a block of code as long as a specified condition is reached. Loops are handy because they save time, reduce errors, and they make code more readable.

Java While Loop

The while loop loops through a block of code as long as a specified condition is true:

Syntax:

```
while (condition){
//code to be executed
Increment / decrement statement
}
```

Example of while loop

```
public class Main
{
            public static void main(String[] args)
    {
                        int i=0;
        while (i<=5)
        {
            System.out.println("The Number is "+ i);
i++;
        }
    }
}
```

Output of above code is:

```
The Number is 0
The Number is 1
The Number is 2
The Number is 3
The Number is 4
The Number is 5
```

As long as the condition is true, the statements execute again and again. If the statement to be executed in any loop is only one, you can omit the braces. As the while loop checks the condition at the start of the loop, statements may not execute even once if the condition fails. If you want to execute the body of a loop atleast once, use do-while loop.

Note: Variable used in the condition should be increased, otherwise the loop will never end!

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

```
do
{
        Statements;
} while(condition);
```

Example of do-while loop

```
public class Main
{
        public static void main(String[] args)
        {
                int i=20;
                do
                {
                        System.out.println(i);
                        i++;
                }
                while(i<25);
        }
}
```

Output of above code is:

```
20
21
22
23
24
```

for loop

In a ***for*** statement, initialization of the loop control variable, condition and modifying the value of the loop control variable are combined into a single line. When we know exactly how many times you want to loop through a block of code, the for loop is used instead of a while loop:

Syntax:

```
for(initialization; condition; iteration)
{
        Statements;
}
```

In the above syntax, the initialization is an assignment expression which initializes the loop control variable and/or other variables. The initialization expression evaluates only once. The condition is a boolean expression. If the condition evaluates to true, the body of the loop is executed. Else, the control quits the loop.

Every time after the body executes, the iteration expression is evaluated. Generally, this is an increment or decrement expression which modifies the value of the control variable.

Example of for loop

```
public class Main
{
	public static void main(String[] args)
   {
   int i;
			for(i=0;i<5;i++)
			{
				System.out.println(i);
			}
   }
  }
```

```
1
2
3
4
```

for-each Statement

The for-each statement is also known as enhanced for statement. It is a simplification over the **for** statement and is generally used when you want to perform a common operation sequentially over an array. The syntax of for-each statement is as shown below:

Syntax:

```
for(type  var : array name)
{
	Statements;
}
```

The data type of var must be same as the data type of the array. The above syntax is can be read as, for each value in array, execute the statements. Starting from the first value in the array, each value is copied into var and the statements are executed. The loop executes until the values in the array completes.

Example of for-each loop

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

String[] branches={"CSE","ECE","MECH","EEE"};
for(String i: branches)
   {
	System.out.println(i);
   }
  }
 }
```

Output of above code is:

```
CSE
ECE
MECH
EEE
```

Java Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Example program demonstrating break.

```
public class Main
{
          public static void main(String[] args)
   {
          int i;
for(i=0; i<10; i++)
          {
                    if(i==5)
          {
                    break;
          }
System.out.println("Now the value of i is "+ i);

      }
   }
}
```

Output of above code is:

```
Now the value of i is 0
Now the value of i is 1
Now the value of i is 2
Now the value of i is 3
Now the value of i is 4
```

Note that loop is not executed till 10.

Java Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Let’s execute the above code by just changing “break” with “continue”.

Syntax for continue is :

```
Loop
{
        //Statements;
        if(condition)
                continue;
        //Statements after continue;
}
```

Example of continue is:

```
public class Main
{
        public static void main(String[] args)
   {
        int i;
for(i=0; i<10; i++)
        {
                if(i==5)
        {
                continue;
        }
System.out.println("Now the value of i is "+ i);

      }
   }
}
```

Output:

```
Now the value of i is 0
Now the value of i is 1
Now the value of i is 2
Now the value of i is 3
Now the value of i is 4
Now the value of i is 6
Now the value of i is 7
Now the value of i is 8
Now the value of i is 9
```

Notice that, when the loop didn't execute if the value of **i** is 5.