History of Java

- Java is an object-oriented programming language developed by James Gosling and colleagues at Sun Microsystems in the early 1990s. Unlike conventional languages which are generally designed either to be compiled to native (machine) code, or to be interpreted from source code at runtime, Java is intended to be compiled to a bytecode, which is then run (generally using JIT compilation) by a Java Virtual Machine.
- Java was started as a project called "Oak" by James Gosling in June 1991. Gosling's goals were to implement a virtual machine and a language that had a familiar C-like notation but with greater uniformity and simplicity than C/C++.
- The first public implementation was Java 1.0 in 1995. It made the promise of "Write Once, Run Anywhere", with free runtimes on popular platforms. It was fairly secure and its security was configurable, allowing for network and file access to be limited.
- The major web browsers soon incorporated it into their standard configurations in a secure "applet" configuration. New versions for large and small platforms (J2EE and J2ME) soon were designed with the advent of "Java 2".

Features of Java OR Properties of Java

1. Simple, small and familiar
2. Object Oriented
3. Compiled and Interpreted
4. Platform – independent (Architecture Nuetral)
5. Portable
6. Robust
7. Secure
8. Distributed

9. Multithreaded and Interactive
10. Dynamic

Simple, small and familiar

- Java is a small and simple language. Many features of C and C++ that are either redundant or sources of unreliable code are not part of Java.
- E.g. Java does not use pointer, pre-processor header files, goto statement and many others. It also eliminates operator overloading and multiple inheritance.
- Familiarity is another striking feature of Java. Java is very close to C and C++. Java uses many constructs of C and C++ and therefore java look like C.

Object Oriented

- Java is true object oriented language. Almost everything in Java is an object.
- All program code and data reside within objects and classes.
- Basic concepts of OOPs in java
    - Object
    - Class
    - Inheritance
    - Polymorphism
    - Abstraction
    - Encapsulation

Compiled and Interpreted

- Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making java a two stage system.
- First java compiler translates source code into byte code instructions. Byte code are not machine instructions and therefore, in the second stage, Java

interpreter generates machine code that can be directly executed by the machine that is running the Java program.

Platform – independent (Architecture Nuetral)

- Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode.
- This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Portable

- Java has platform independent feature. So it provide the concept of porting from one platform to another.
- Java programmes can be easily moved from one computer system to another, anytime anywhere.
- Java ensures portability in two ways:
    - Java compiler generates bytecode instructions that can be implemented on any machine.
    - The size of primitive datatypes are machine independent.

Robust

- Java is a robust language. It means there is less chance of crashing the computer and more chance of bug free software.
- It has strict compile time checking for data types.
- It is designed as a garbage-collected language relieving the programmers virtually all memory management problem.

- Java also incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system.

Secure

- Security becomes an important issue for a language that is used for programming on internet.
- Java system not only verify all memory access but also ensure that no viruses are communicated with an applet.
- The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

Distributed

- Java is designed as a distributed language for creating applications on network. It has the ability to share both data and programs.
- Java applications can open and access remote objects on internet as easily as they can do in a local system.
- This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Multithreaded and Interactive

- Multithreaded means handling multiple task simultaneously. This means that we need not wait for the application to finish one task before beginning another.
- For example, we can listen to an audio clip while scrolling a page and at the same time download an applet from a distance computer.
- This feature greatly improves the interactive performance of graphical applications.

Dynamic

- Java is dynamic language. Java is capable of dynamically linking in new class libraries, methods and objects.
- Java programs support functions written in other language such as C and C++. These functions are known as native methods. Native methods are linked dynamically at runtime.

Comparison of Java with C++

(MarApr2017) (MarApr2016)

Comparison Index	C++	Java
Platform Independent	C++ is platform dependent.	Java is platform independent.
Object Oriented	C++ is partial object oriented programming.	Java is truly object oriented programming.
Use	C++ is mainly used for system programming.	Java is mainly used for application programming. it is widely used in window, web based and mobile application.
go to	C++ supports the goto statement	Java doesn't support the goto statement.
Multiple Inheritance	C++ supports multiple inheritance	Java doesn't upport multiple inheritance through class. it can be achieved by interfaces in Java
Operator Overloading	C++ supports operator overloading	Java doesn't support operator overloading
Pointer	C++ supports pointers. you can write 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 ony. c++ is compiled and run using the compiler which converts source code into machine code. so C++ is platform dependent.	Java uses compiler and interpreter both. java source code is converted into bytecode at compilation time. the interpreter executed this bytecode at runtime and produces output. java is interpreted that is why it is platform independent.
Call by value and call by reference	C++ supports both call by value and call by reference	Java supports call by value only. there is no call by reference in java.
Structure and Union	C++ supports structures and unions	Java doesn't support structure and unions
Thread Support	C++ doesn't have built-in support for threads. it relies on third party libraries for thread support	Java has built in support
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function	Java has no virtual keyword. we can override all non-static methods are virtual by default.
Using right shift >>>	C++ doesn't support >>> operator	Java supports unsigned right shift >>> operator that fill zero at the top for the negative numbers. for positive numbers, it works same like >> operator.

Explain structure of JVM (Mar2016)

OR

Java Interpreter and Java Compiler (Mar2018)

- All language compilers translate source code into machine code for a specific computer. Java compiler also does the same thing. But java achieve architecture neutrality because Java compiler produces an intermediate code known as bytecode for a machine. This machine is called Java Virtual Machine and it exists only inside the computer memory.
- Following figure illustrates the process of compiling a java program into bytecode which is also referred to as virtual machine code.

- The virtual machine code is not machine specific. The machine specific code known as machine code that is generated b the Java interpreter by acting as an intermediary between the virtual machine and the real machine as shown in following diagram.

- Interpreter is different for different machines.

Understanding significance of using public static void main(String args[])

In Java programs, the point from where the program starts its execution or simply the entry point of Java programs is the **main()** method.

- **Public:** It is an Access modifier, which specifies from where and who can access the method. Making the main() method public makes it globally available. It is

made public so that JVM can invoke it from outside the class as it is not present in the current class.

- **Static:** It is a keyword which is when associated with a method, makes it a class related method. The main() method is static so that JVM can invoke it without instantiating the class.

- **Void:** It is a keyword and used to specify that a method doesn't return anything. As main() method doesn't return anything, its return type is void. As soon as the main() method terminates, the java program terminates too. Hence, it doesn't make any sense to return from main() method as JVM can't do anything with the return value of it.

- **main:** It is the name of Java main method. It is the identifier that the JVM looks for as the starting point of the java program. It's not a keyword.

- **String[] args:** It stores Java command line arguments and is an array of type java.lang.String class. Here, the name of the String array is args but it is not fixed and user can use any name in place of it.

Explain JDK and its uses.

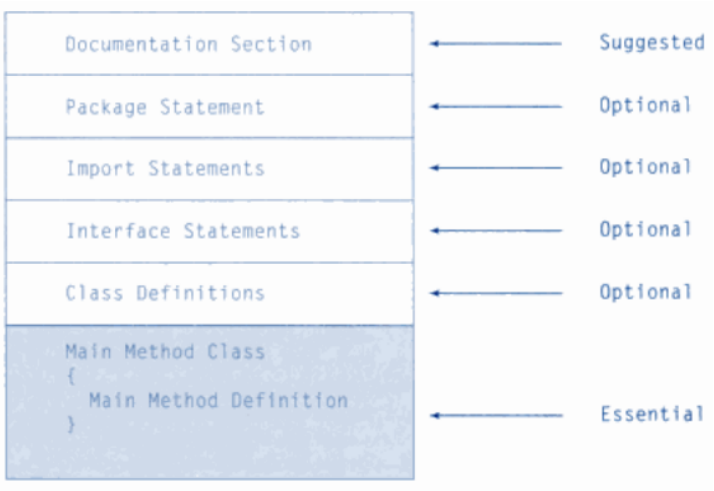
- The Java Development Kit (JDK) is one of three core technology packages used in Java programming, along with the JVM (Java Virtual Machine) and the JRE (Java Runtime Environment). It's important to differentiate between these three technologies, as well as understanding how they're connected:
    - The JVM is the Java platform component that executes programs.
    - The JRE is the on-disk part of Java that creates the JVM.

- The JDK allows developers to create Java programs that can be executed and run by the JVM and JRE.

- Developers new to Java often confuse the Java Development Kit and the Java Runtime Environment. The distinction is that the JDK is a package of tools for developing Java-based software, whereas the JRE is a package of tools for running Java code.
- The JRE can be used as a standalone component to simply run Java programs, but it's also part of the JDK. The JDK requires a JRE because running Java programs is part of developing them.

Explain Java program structure.

- A Java program may contain one or more sections as shown in following diagram.



- **Documentation Section:** The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. Comments must explain why and what of classes and how of algorithm.
- **Package statement:** The first statement allowed in a Java file is package statement. This statement declared a package name and informs the compiler that the classes defined here belong to this package.

- Package statement is optional.
- **Import statement:** The next thing after a package statement may be a number of import statements. This is similar to the #include statement in C.
- Interface statement: An interface is like a class but includes a group of method declarations. This is also an optional section and is used only when we wish to implement the multiple inheritance feature in the program.
- **Class Definitions:** A java program may contain multiple class definitions. Classes are the primary and essential elements of a Java program. These classes are used to map the objects of real -world problems.
- **Main method class:** Since every Java stand-alone program requires a main method as its starting point, this class is the essential part of a java program.

Explain Java Programming environment. OR Java Runtime environment.

- A runtime environment is a piece of software that is designed to run other software. As the runtime environment for Java, the JRE contains the Java class libraries, the Java class loader, and the Java Virtual Machine. In this system:
- The class loader is responsible for correctly loading classes and connecting them with the core Java class libraries.
- The JVM is responsible for ensuring Java applications have the resources they need to run and perform well in your device or cloud environment.
- The JRE is mainly a container for those other components, and is responsible for orchestrating their activities.

Explain command line argument with example.

- There may be occasions when we my like our program to act in a particular way depending on the input provided at the time of execution. This is achieved in Java programs by using what are known as command line arguments.
- Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.
- Example:

```
import java.io.*;
class ComLine
{
  public static void main(String args[])
  {
      int count, i=0;
      count = args.length;
      System.out.println("Number of arguement ="+count);
      while(i<count)
      {
            System.out.println(args[i]);
            i=i+1;
      }
  }
}
Output :
         Number of argument = 4
         Java
         Basic
         C
         CPP
```

- Here is a program that can receive and use the arguments provided in the command line. In the program args is declared as an array of strings. Any arguments provided in the command line are passed to the array args as its elements.
- We can simply access array elements and use them in the program.
- In following way we can supply argument when we execute program
- Example

  Java ComLine Java Basic C CPP

- This command line contains four arguments. These are assigned to the array args as follows:

  Java --------> args[0]

  Basic --------> args[1]

  C ----------> args[2]

  CPP ----------> args[3]

Java Tokens

- Smallest individual units in a program are known as tokens.
- Five types of tokens :
  - Reserved Keywords
  - Identifiers
  - Literals
  - Operators
  - Seprators

Keywords

- Keywords have specific meaning in java. We can not use them as names for variable, classes, methods and so on.
- All keywords are to be written in lower-case letter.

Identifiers

- Identifiers are programmer designed tokens. They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program.
- Rules :
  - They can have alphabets, digits and the underscore and dollar sign characters.

- They must not begin with a digit.
- Uppercase and lowercase letters are distinct.
- They can be of any length.

Literals

- Literals in java are sequence of characters that represent constant value to be stored in variable.
- Types:
  - Integer literals
  - Floating point literals
  - Character literals
  - String literals
  - Boolean literals

Operators

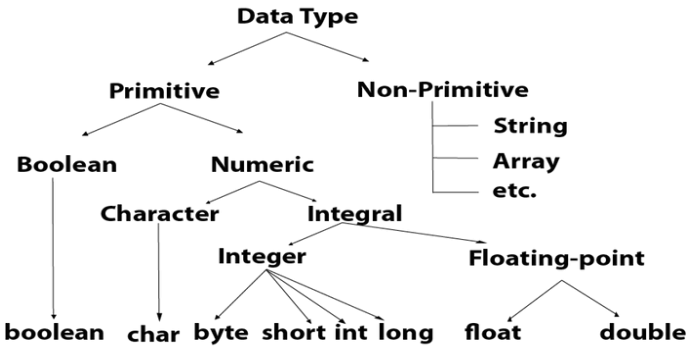
- An operators is a symbol that takes one or more arguments and operate on them to produce a result.
- Types:
  - Arithmetic operator
  - Relational operator
  - Logical operator
  - Assignment operator
  - Increment and Decrement operator
  - Conditional operator
  - Bitwise operator
  - Special operator

Separators

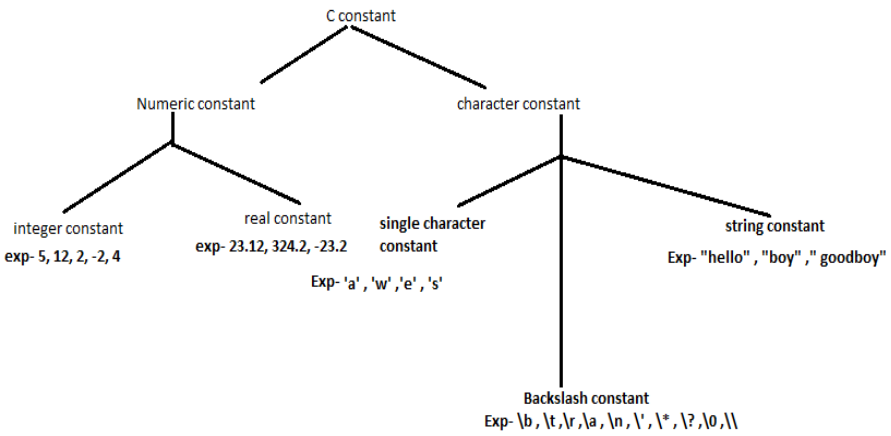
- Separators are symbol used to indicate where groups of code are divided and arranged.

Symbol	Name	Purpose
()	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expression in control statements and surrounding cast types.
{}	Braces	Used to contain the value of automatically initialized arrays. Also used to define a block of code, for classes, methods and local scopes.
[]	Brackets	Used to declare array types. Also used when dereferencing array value.
;	Semicolon	Terminates statements
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a for statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

Data Types



- **Constant**



Operators

1. Arithmetic operators

Operator	Description
+	Add two operands
-	Subtracts second operand from the first
*	Multiplies both operands
/	Divides numerator by de-numerator
%	Modulus Operator and remainder of after an integer division

2. Relational Operators

Operator	Use	Description
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	Op1 and op2 are not equal

3. Logical Operators

Operator	Meaning	Example	Result
&&	Logical AND	(5<2) && (5>3)	False
||	Logical OR	(5<2) ||(5>3)	True
!	Logical NOT	!(5<2)	True

4. Assignment Operators

Operator	Description	Example	Operation	Output
=	Assignment operator is used to assign value from right to left	X=Y	X=Y	X=5
+=	Add & assignment operator is used to add first and then assign the result to left	X+=Y	X=X+Y	X=15
-=	Subtract & assignment operator is used to subtract first and then assign the result to left	X-=Y	X=X-Y	X=5
*=	Multiply & assignment operator is used to multiply first and then assign the result to left	X*=Y	X=X*Y	X=50
/=	Divide & assignment operator is used to divide first and then assign the result to left	X/=Y	X=X/Y	X=2
%=	Modulus & assignment operator is used to modulus first and then assign the result to left	X%=Y	X=X%Y	X=0

5. Increment and Decrement operators

Operator	Description
++	Increment operator, increases integer value by one
--	Decrement operator, decreases integer value by one

6. Conditional Operators

- The character pair **? :** is a ternary operator available in Java.
- Syntax :

  exp1 ? exp2 : exp3

- Where exp1, exp2 and exp3 are expressions.
- The operator ?: works as follow:
  - exp1 is evaluated first.
  - If it is nonzero(true), then the expression exp2 is evaluated and becomes the value of the conditional expression.
  - If exp1 is false, exp3 is evaluated and its value becomes the value of the conditional expression.
- Example,

  A = 10; B = 15;

  X = (A > B) ? A : B ;

- In this example, X will be assigned the value of B. This can be achieved using if...else statement also.

7. Bitwise Operators

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	One's complement
<<	Shift left
>>	Shift right
>>>	Shift right with zero fill

Bitwise AND (&)

- This operator is 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

  _____

  0101 = 5 (In decimal)

Bitwise OR (|)

- This operator is 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)

Bitwise exclusive OR (^)

- This operator is 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

  0010 = 2 (In decimal)

One's Complement (~)

- This operator is unary operator, denoted by '~'. It returns the one's compliment representation of the input value, i.e, with all bits inversed, means it makes every 0 to 1, and every 1 to 0.
- For example,

  a = 5 = 0101 (In Binary)

- Bitwise Compliment Operation of 5

  ~ 0101

  1010 = 10 (In decimal)

Shift left

- Shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.
- For example,

  a = 5 = 0000 0101

  b = -10 = 1111 0110

  a << 1 = 0000 1010 = 10

  a << 2 = 0001 0100 = 20

  b << 1 = 0000 1010 = -20

  b << 2 = 0001 0100 = -40

Shift right

- Shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of initial number. Similar effect as of dividing the number with some power of two.
- For example,

  Example 1:

  a = 10

  a>>1 = 5

  Example 2:

  a = -10

  a>>1 = -5

- We preserve the sign bit.

Shift right with zero fill

- Shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0. (>>>) is unsigned-shift; it'll insert 0. (>>) is signed, and will extend the sign bit.
- For example,

  Example 1:

  a = 10

  a>>>1 = 5

  Example 2:

  a = -10

  a>>>1 = 2147483643
- DOES NOT preserve the sign bit.

8. Special Operator

- Java support some special operators such as instanceof operator and member selection operator (.).

- **Instanceof Operator :** The instanceof is an object reference operator and return true if the object on the left hand side is an instance of the class given on the right-hand side. This operator allows us to determin whether the object belongs to a particular class or not.
- Example

  person instanceof Student

  is true if the object person belongs to the class student otherwise it is false.

- **Dot Operator :** The dot operator(.) is used to access the instance variables and methods of class objects.

- Example

```
person1.age //Reference to the variable age
person1.salary()    //reference to the method salary()
```

- It is also used to access classes and sub-packages from a package.

Branching Statement

- In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled.
- A programming language uses control statements to control the flow of execution of program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.
- Java's Selection statements:
  - if
  - if-else
  - nested-if
  - if-else-if
  - switch-case
- These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

if:

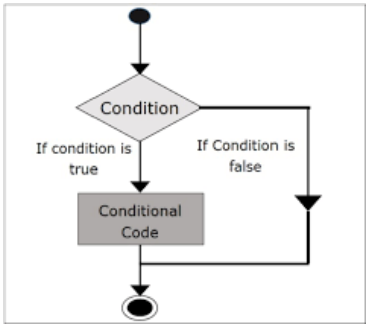
- if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.
- Syntax:

```
if(condition)
{
   // Statements to execute if
   // condition is true
```

```
}
```

- Here, condition after evaluation will be either true or false. if statement accepts boolean values – if the value is true then it will execute the block of statements under it.
- If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement will consider the immediate one statement to be inside its block. For example,

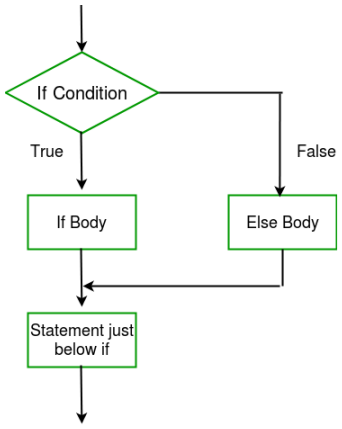
```
if(condition)
   statement1;
   statement2;

// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
```



if-else:

- The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.
- **Syntax**:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```



nested-if:

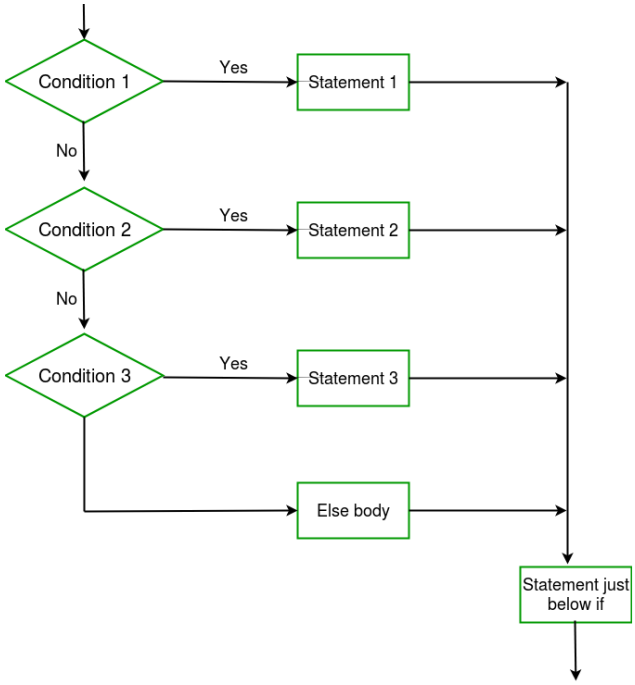
- A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, java allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.
- **Syntax:**

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```

if-else-if ladder:

- Here, a user can decide among multiple options.The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```



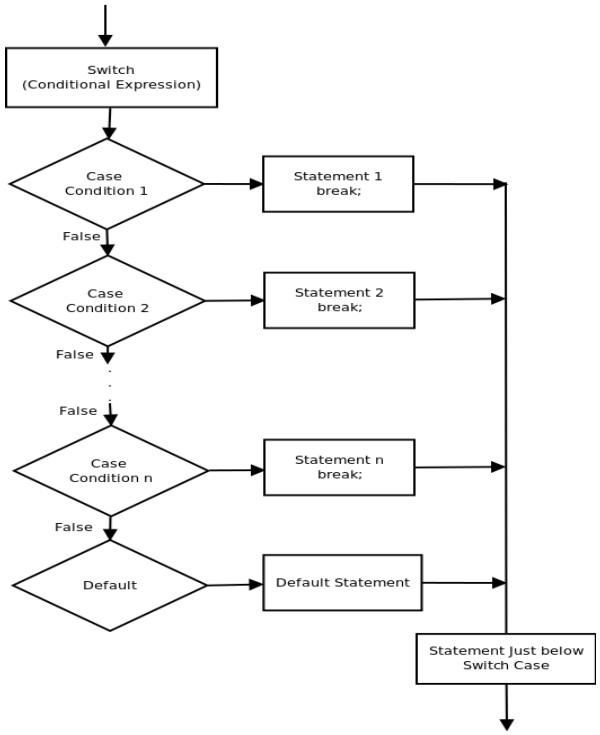
switch-case

- The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.
- **Syntax:**

```
switch (expression)
{
  case value1:
    statement1;
    break;
  case value2:
    statement2;
    break;
  .
  .
  case valueN:
    statementN;
    break;
  default:
    statementDefault;
}
```

- Expression can be of type byte, short, int char or an enumeration. Beginning with JDK7, expression can also be of type String.
- Duplicate case values are not allowed.

- The default statement is optional.
- The break statement is used inside the switch to terminate a statement sequence.
- The break statement is optional. If omitted, execution will continue on into the next case.



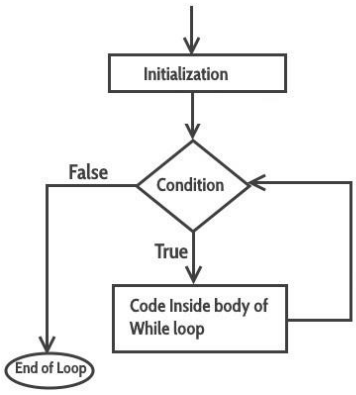
Loops in Java

- In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in java.
  1. for loop
  2. while loop
  3. do-while loop

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...
}
```



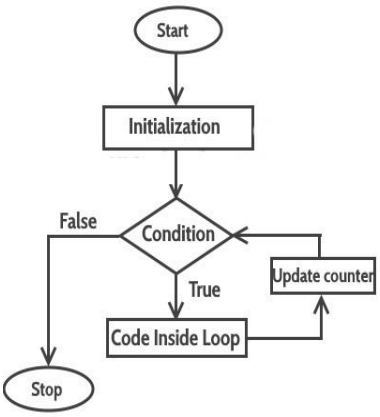
- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called Entry control loop
- Once the condition is evaluated to true, the statements in the loop 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.

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.

- **Syntax:**

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



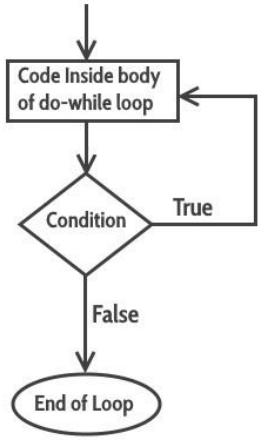
- **Initialization condition:** Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- **Testing Condition:** It is used for testing the exit condition for a loop. It must return a boolean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

do while:

- 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);
```



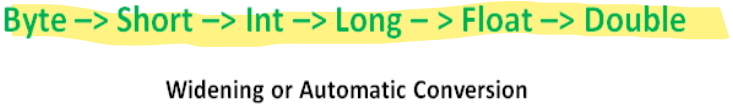
- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- 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.

Type Casting

- When you assign value of one data type to another, the two types might not be compatible with each other.
- In java, type casting is classified into two types,
  - Widening Casting (implicit)
  - Narrowing Casting (Explicit)

Widening or Automatic Type Conversion

- Widening conversion takes place when two data types are automatically converted. This happens when:
  - The two data types are compatible.
  - When we assign value of a smaller data type to a bigger data type.
- For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.



Widening or Automatic Conversion

- **Example,**

```
class MyClass
{
  public static void main(String[] args)
  {
       int myInt = 9;
  double myDouble = myInt;
  System.out.println(myInt); // Outputs 9
  System.out.println(myDouble); // Outputs 9.0
      }
}
```

Narrowing or Explicit type conversion

- When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

- Narrowing casting must be done manually by placing the type in parentheses in front of the value:
- **Syntax:**

  type variable1=(type) variable2;

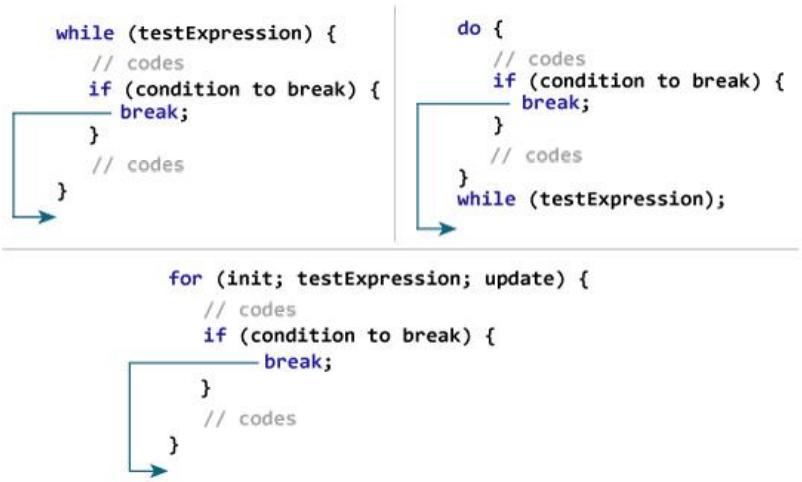
- **Example,**

```
public class MyClass
{
  public static void main(String[] args)
  {
   double myDouble = 9.78;
   int myInt = (int)myDouble;
   System.out.println(myDouble);//Outputs 9.78
   System.out.println(myInt); // Outputs 9
  }
}
```

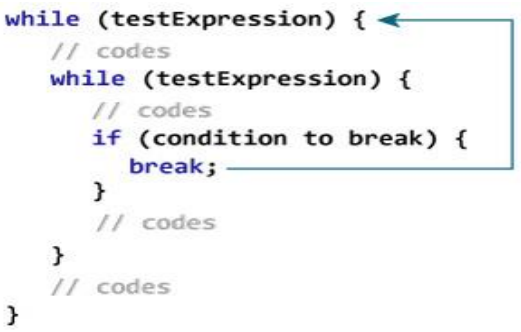
Jumps in Loop

Jumping out of the Loop : Break

- An early exit from a loop can be accomplished by using the break statement.
- When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.
- When the loops are nested, the break would only exit from the loop containing it. That is the break will exit only single loop.



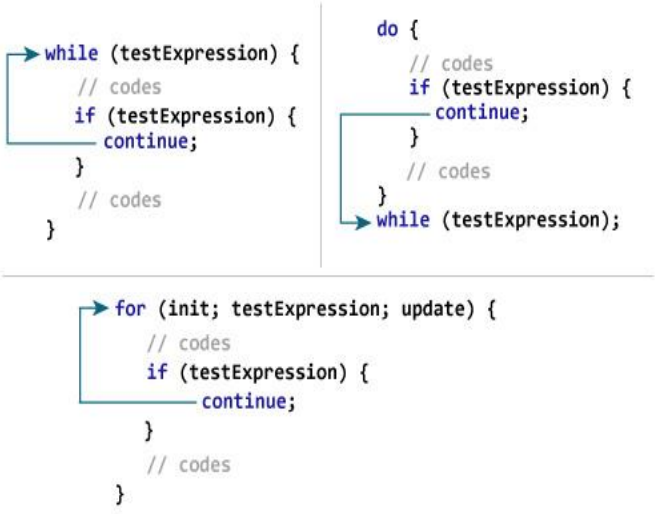
In case of nested Loops



Skipping a part of a loop : Continue

- Java supports another similar statement called the continue statement. However, unlike the break which cause them to be terminates, the continue causes the loop to be continued with the next iteration after skipping any statements in between.
- The continue statement tells the compiler, "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION."
- The format of the continue statement is simply,

  continue;
- in while and do loops, continue causes the control to go directly to the test condition and then to continue the iteration process.
- In the cause of for loop, the increment section of the loop is executed before the test condition is evaluated.



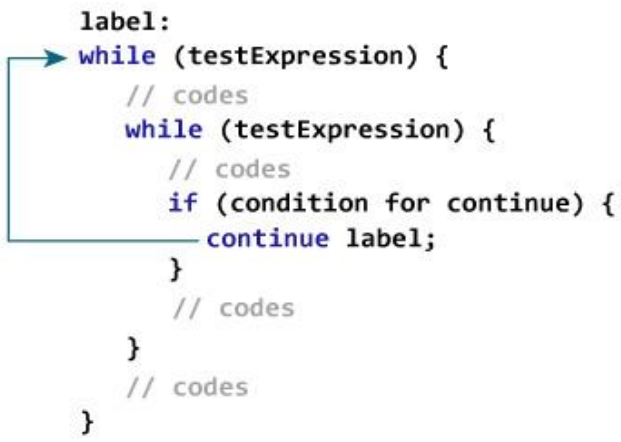
In case of Nested Loops

```
while(testExpresison) {
   // codes
   while (testExpression) {
      // codes
      if (condition for continue) {
         continue;
      }
      // codes
   }
   // codes
}
```

Labeled Loop

- In java, we can give a label to a block of statements.
- A label is any valid java variable name. to give label to a loop, place it before the loop with a colon at the end.
- Simple break statement causes the control to jump outside the nearest loop and a simple continue statement restart the current loop.
- If we want to jump outside a nested loops or to continue a loop that is outside the current one, then we may have to use the labelled break and labelled continue statements.

```
label:
for (int; testExpresison, update) {
   // codes
   for (int; testExpression; update) {
      // codes
      if (condition to break) {
         break label;
      }
      // codes
   }
   // codes
}
```



What is bytecode?

- Java bytecode is the instruction set for the Java Virtual Machine. A java program is compiled, java bytecode is generated. Java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.

Write down the significance of CLASSPATH.

- **CLASSPATH** is actually an environment variable in Java, and tells Java applications and the Java Virtual Machine (JVM) where to find the libraries of classes. These include any that you have developed on your own.
- An environment variable is a global system variable, accessible by the computer's operating system (e.g., Windows). In Java, CLASSPATH holds the list of Java class file directories and the JAR file, which is Java's delivered class library file.

Java is robust – justify.

- Java is a robust language. It means there is less chance of crashing the computer and more chance of bug free software.
- It has strict compile time checking for data types.

- It is designed as a garbage-collected language relieving the programmers virtually all memory management problem.
- Java also incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system.

Write a full form of the following : EJB, JFC, JSDK, AWT

- EJB: Enterprise Java Bean
- JFC: Java Foundation Classes
- JSDK: Java Servlet Developers Kit
- AWT: Abstract Window Toolkit

What is JVM?

- A Java virtual machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode.

Explain working of JIT.

- The JIT compiler is enabled by default, and is activated when a Java method is called. The JIT compiler compiles the bytecode of that method into native machine code, compiling it "just in time" to run. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it

Give the use of import statement.

- In Java, the import statement is used to bring certain classes or the entire packages, into visibility. As soon as imported, a class can be referred to directly by using only its name.

What is native code?

- Native code is computer programming (code) that is compiled to run with a particular processor (such as an Intel x86-class processor) and its set of instructions.
- This term is sometimes used in places where *machine code* (see above) is meant.