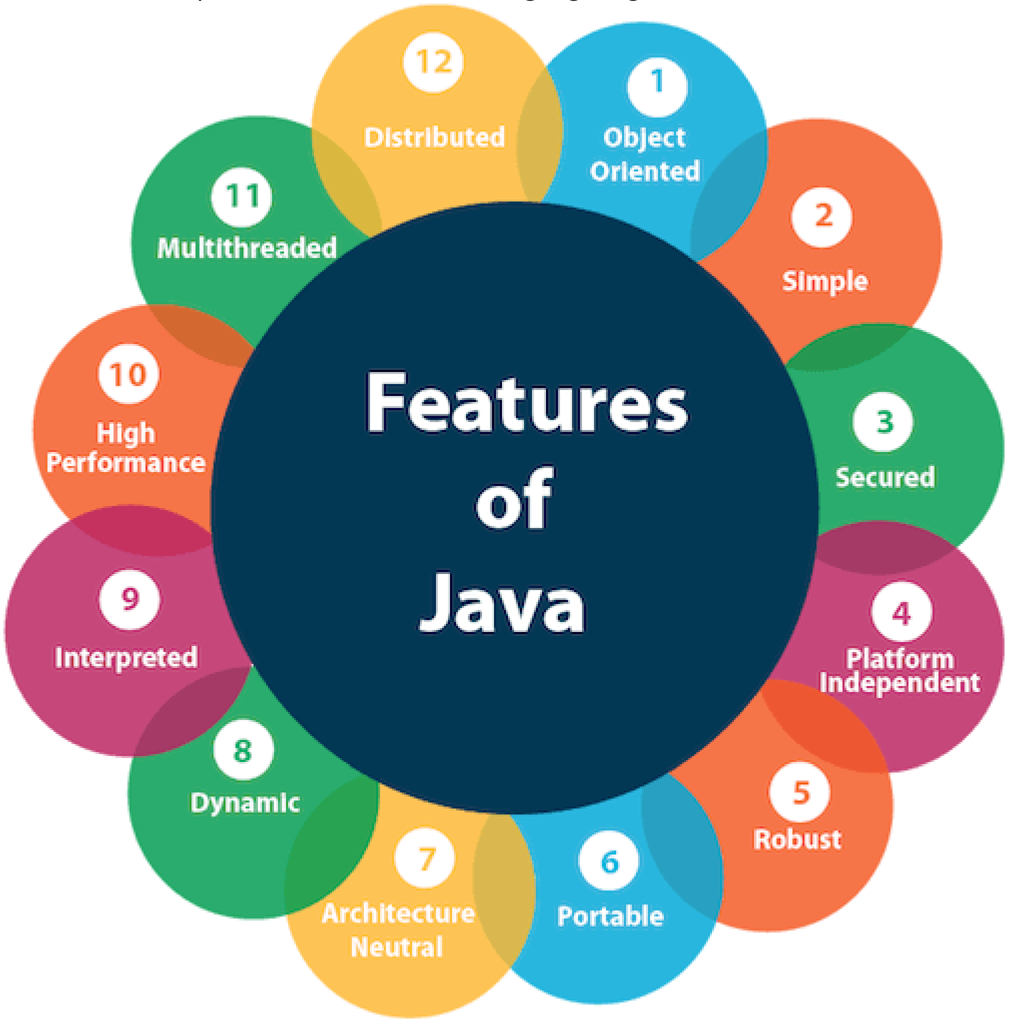
JAVA NOTES

Features of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java *buzzwords*.

A list of most important features of Java language is given below.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

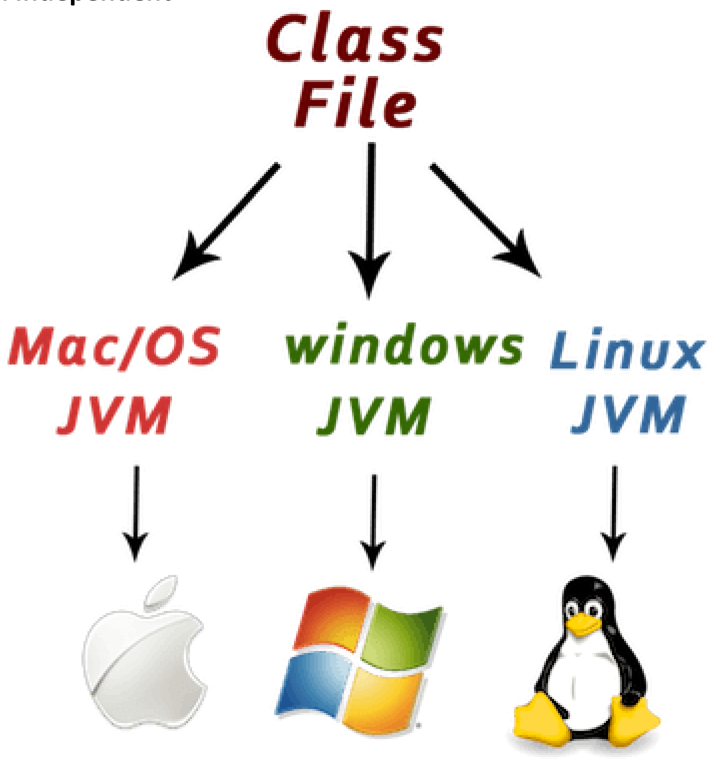
Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent



Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

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).

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**

- **Classloader:** Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java 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.

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

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

Data Types in Java

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

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

Java Primitive Data Types

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

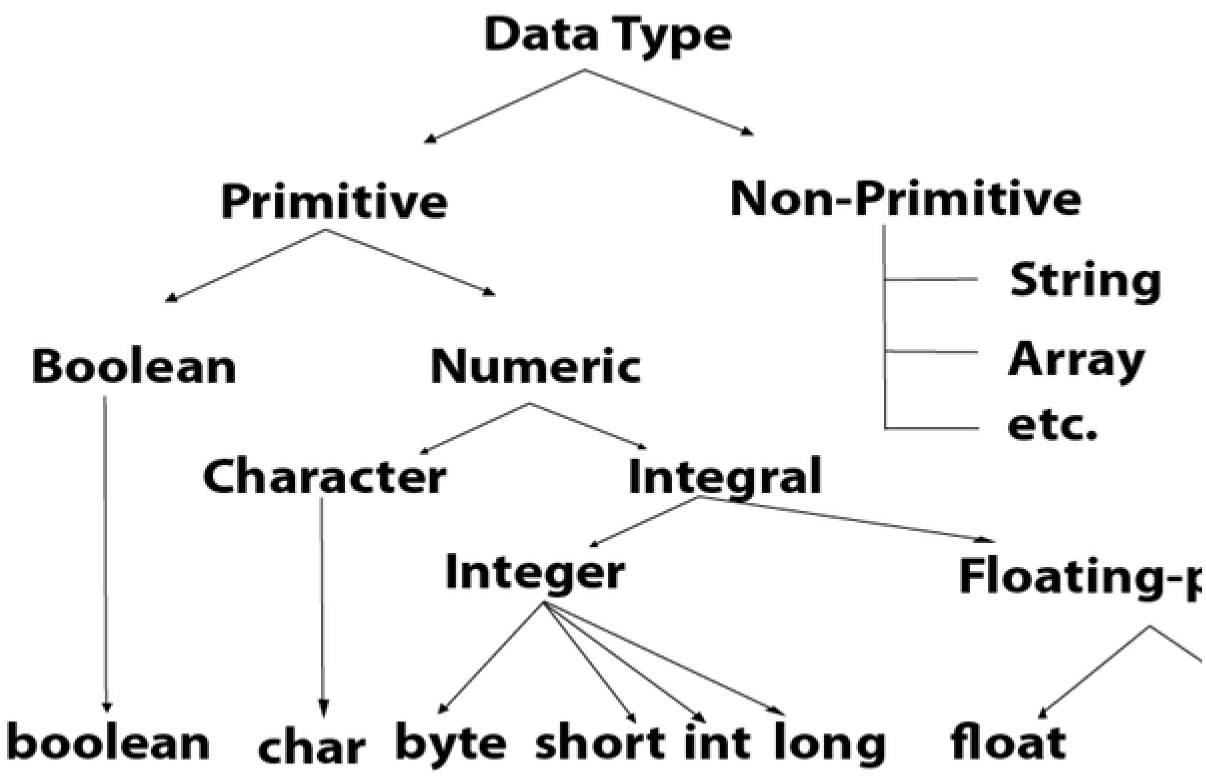
Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type

long data type

- float data type
- double data type



Data Type	Default Value	Default size
boolean	false	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
double	0.0d	8 byte

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between - 9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

❖ Java Variables and (Primitive) Data Types

➢ Java Variables

There are 4 types of variables in Java programming language:

1.Instance Variables (Non-Static Fields)
2.Class Variables (Static Fields)
3.Local Variables
4.Parameters

➢ Java Primitive Data Types

As mentioned above, Java is a statically-typed language. This means that, all variables must be declared before they can be used.
int speed;
Here, speed is a variable, and the data type of the variable is int. The int data type determines that the speed variable can only contain integers.
In simple terms, a variable's data type determines the values a variable can store. There are 8 data types predefined in Java programming language, known as primitive data types.
In addition to primitive data types, there are also referenced data types in Java (you will learn about it in later chapters).

8 Primitive Data Types

1.Boolean:

The boolean data type has two possible values, either true or false.
Default value: false.
They are usually used for true/false conditions.
Example:

```
class BooleanExample {
public static void main(String[] args) {
boolean flag = true;
System.out.println(flag);
}
}
```

2.Byte:

The byte data type can have values from -128 to 127 (8-bit signed two's complement integer).
It's used instead of int or other integer data types to save memory if it's certain that the value of a variable will be within [-128, 127].
Default value: 0

Example:

```
class ByteExample {
public static void main(String[] args) {
byte range;
range = 124;
System.out.println(range);
}
```

3.short:

The short data type can have values from -32768 to 32767 (16-bit signed two's complement integer).
It's used instead of other integer data types to save memory if it's certain that the value of the variable will be within [-32768, 32767].
Default value: 0

Example:

```
class ShortExample {
public static void main(String[] args) {
short temperature;
temperature = -200;
System.out.println(temperature);
}
}
```

4.int:

The int data type can have values from -231 to 231-1 (32-bit signed two's complement integer).
If you are using Java 8 or later, you can use unsigned 32-bit integer with minimum value of 0 and maximum value of 232-1. If you are interested in learning more about it, visit: How to use the unsigned integer in java 8?
Default value: 0

Example:

```
class IntExample {
public static void main(String[] args) {
int range = -4250000;
System.out.println(range);
}
}
```

5.long:

The long data type can have values from -263 to 263-1 (64-bit signed two's complement integer).
If you are using Java 8 or later, you can use unsigned 64-bit integer with minimum value of 0 and maximum value of 264-1.
Default value: 0

Example:

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

```
long range = -42332200000L;
System.out.println(range);
}
}
```

6.double:

The double data type is a double-precision 64-bit floating point.
It should never be used for precise values such as currency.
Default value: 0.0 (0.0d)
Example:

```
class DoubleExample {
public static void main(String[] args) {
double number = -42.3;
System.out.println(number);
}
}
```

7.Float:

The float data type is a single-precision 32-bit floating point. Learn more about single precision and double precision floating point if you are interested.
It should never be used for precise values such as currency.
Default value: 0.0 (0.0f)
Example:

```
class FloatExample {
public static void main(String[] args) {
float number = -42.3f;
System.out.println(number);
}
}
```

❖ Operators in java

There are many types of operators in java which are given below:

a) Unary Operator,
b) Arithmetic Operator,
c) Shift Operator,
d) Relational Operator,
e) Bitwise Operator,
f) Logical Operator,
g) Ternary Operator and
h) Assignment Operator.

➢ Java Operator Precedence:

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=

Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

- **Briefly Describe of the operator**

1.Assignment Operator:

Assignment operators are used in Java to assign values to variables. For example,
int age;
age = 5;
The assignment operator assigns the value on its right to the variable on its left. Here, 5 is assigned to the variable age using = operator.
There are other assignment operators too. However, to keep things simple, we will learn other assignment operators later in this article.

Example of Assignment Operator:

```
 class AssignmentOperator {
public static void main(String[] args) {
     int number1, number2;
  // Assigning 5 to number1
    number1 = 5;
    System.out.println(number1);
// Assigning value of variable number2 to number1
    number2 = number1;
    System.out.println(number2);
}
}
```

2.Arithmetic Operators:

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Java Arithmetic Operators Operator Meaning

+	Addition (also used for string concatenation)
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
%	Remainder Operator

Example of Arithmetic Operator:

```
class ArithmeticOperator {
public static void main(String[] args) {
 double number1 = 12.5, number2 = 3.5, result;
```

```
        // Using addition operator
        result = number1 + number2;
        System.out.println("number1 + number2 = " + result);

        // Using subtraction operator
        result = number1 - number2;
        System.out.println("number1 - number2 = " + result);

        // Using multiplication operator
        result = number1 * number2;
        System.out.println("number1 * number2 = " + result);
        // Using division operator
        result = number1 / number2;
        System.out.println("number1 / number2 = " + result);

        // Using remainder operator
        result = number1 % number2;
        System.out.println("number1 % number2 = " + result);
    }
}
```

3.Unary Operators:

Unary operator performs operation on only one operand.

Operator	Meaning
+	Unary plus (not necessary to use since numbers are positive without using it)
-	Unary minus; inverts the sign of an expression
++	Increment operator; increments value by 1
--	decrement operator; decrements value by 1
!	Logical complement operator; inverts the value of a boolean

Example of Unary Operator:

```
class UnaryOperator {
        public static void main(String[] args) {
        double number = 5.2, resultNumber;
        boolean flag = false;
        System.out.println("+number = " + +number);
        // number is equal to 5.2 here.
        System.out.println("-number = " + -number);
        // number is equal to 5.2 here.
        // ++number is equivalent to number = number + 1
        System.out.println("number = " + ++number);
        // number is equal to 6.2 here.
        // -- number is equivalent to number = number - 1
        System.out.println("number = " + --number);
        // number is equal to 5.2 here.
        System.out.println("!flag = " + !flag);
        // flag is still false.
    }
}
```

4.Equality and Relational Operators:

The equality and relational operators determines the relationship between two operands. It checks if an operand is greater than, less than, equal to, not equal to and so on. Depending on the relationship, it results to either true or false.

Java Equality and Relational Operators Operator	Description	Example
==	equal to	5 == 3 is evaluated to false
!=	not equal to	5 != 3 is evaluated to true
>	greater than	5 > 3 is evaluated to true
<	less than	5 < 3 is evaluated to false
>=	greater than or equal to	5 >= 5 is evaluated to true
<=	less then or equal to	5 <= 5 is evaluated to true

Example of Equality and Relational Operator:

```
  class RelationalOperator {
      public static void main(String[] args) {
          int number1 = 5, number2 = 6;
          if (number1 > number2)
          {
                  System.out.println("number1 is greater than number2.");
          }
          else
          {
                  System.out.println("number2 is greater than number1.");
          }
      }
  }
```

In addition to relational operators, there is also a type comparison operator instanceof which compares an object to a specified type. For example,

- **instanceof Operator**

  Here's an example of instanceof operator.

```
     class instanceofOperator {
    public static void main(String[] args) {
       String test = "asdf";
        boolean result;
        result = test instanceof String;
        System.out.println(result);
    }
  }
```

When you run the program, the output will be true. It's because test is the instance of String class. You will learn more about instanceof operator works once you understand Java Classes and Objects.

5.Logical Operators:

The logical operators || (conditional-OR) and && (conditional-AND) operates on boolean expressions. Here's how they work.

Java Logical Operators Operator	Description	Example
||	conditional-OR; true if either of the boolean expression is true	false || true is evaluated to true

&& conditional-AND; true if all boolean expressions are true false && true is evaluated to false

Example 8: Logical Operators

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

         int number1 = 1, number2 = 2, number3 = 9;
         boolean result;

         // At least one expression needs to be true for result to be true
         result = (number1 > number2) || (number3 > number1);
         // result will be true because (number1 > number2) is true
         System.out.println(result);

         // All expression must be true from result to be true
         result = (number1 > number2) && (number3 > number1);
         // result will be false because     (number3 > number1) is false
         System.out.println(result);
     }
  }
```

6.Ternary Operator:

The conditional operator or ternary operator ?: is shorthand for if-then-else statement. The syntax of conditional operator is:

variable = Expression ? expression1 : expression2

Here's how it works.

If the Expression is true, expression1 is assigned to variable.

If the Expression is false, expression2 is assigned to variable.

Example of Ternary Operator:

```
 class ConditionalOperator {
     public static void main(String[] args) {
        int februaryDays = 29;
         String result;
        result =  (februaryDays == 28) ? "Not a leap year" : "Leap year";
         System.out.println(result);
     }
  }
```

7.Bitwise and Bit Shift Operators:

To perform bitwise and bit shift operators in Java, these operators are used.

Java Bitwise and Bit Shift Operators Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise inclusive OR

These operators are not commonly used. Visit this page to learn more about bitwise and bit shift operators.

8.More Assignment Operators:

We have only discussed about one assignment operator = in the beginning of the article. Except this operator, there are quite a few assignment operators that helps us to write cleaner code.

Java Assignment Operators Operator Example Equivalent to

+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x / 5
<<=	x <<= 5	x = x << 5
>>=	x >>= 5	x = x >> 5
&=	x &= 5	x = x & 5
^=	x ^= 5	x = x ^ 5
|=	x |= 5	x = x | 5

❖ Java Keywords

Java keywords are also known as **reserved words**. Keywords are particular words which acts as a key to a code. These are predefined words by Java so it cannot be used as a variable or object name.

List of Java Keywords:

A list of Java keywords or reserved words are given below:

abstract: Java abstract keyword is used to declare abstract class. Abstract class can provide the implementation of interface. It can have abstract and non-abstract methods.

boolean: Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.

break: Java break keyword is used to break loop or switch statement. It breaks the current flow of the program at specified condition.

byte: Java byte keyword is used to declare a variable that can hold an 8-bit data values.

case: Java case keyword is used to with the switch statements to mark blocks of text.

catch: Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.

char: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters

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

continue: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

default: Java default keyword is used to specify the default block of code in a switch statement.

do: Java do keyword is used in control statement to declare a loop. It can iterate a part of the program several times.

double: Java double keyword is used to declare a variable that can hold a 64-bit floating-point numbers.

else: Java else keyword is used to indicate the alternative branches in an if statement.

enum: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.

extends: Java extends keyword is used to indicate that a class is derived from another class or interface.

final: Java final keyword is used to indicate that a variable holds a constant value. It is applied with a variable. It is used to restrict the user.
finally: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether exception is handled or not.
float: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
for: Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some conditions become true. If the number of iteration is fixed, it is recommended to use for loop.
if: Java if keyword tests the condition. It executes the if block if condition is true.
implements: Java implements keyword is used to implement an interface.
import: Java import keyword makes classes and interfaces available and accessible to the current source code.
instanceof: Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
int: Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
interface: Java interface keyword is used to declare an interface. It can have only abstract methods.
long: Java long keyword is used to declare a variable that can hold a 64-bit integer.
native: Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
new: Java new keyword is used to create new objects.
null: Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
package: Java package keyword is used to declare a Java package that includes the classes.
private: Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
protected: Java protected keyword is an access modifier. It can be accessible within package and outside the package but through inheritance only. It can't be applied on the class.
public: Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
return: Java return keyword is used to return from a method when its execution is complete.
short: Java short keyword is used to declare a variable that can hold a 16-bit integer.
static: Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is used for memory management mainly.
strictfp: Java strictfp is used to restrict the floating-point calculations to ensure portability.
super: Java super keyword is a reference variable that is used to refer parent class object. It can be used to invoke immediate parent class method.
switch: The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
synchronized: Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
this: Java this keyword can be used to refer the current object in a method or constructor.
throw: The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exception. It is followed by an instance.
throws: The Java throws keyword is used to declare an exception. Checked exception can be propagated with throws.
transient: Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
try: Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
void: Java void keyword is used to specify that a method does not have a return value.
volatile: Java volatile keyword is used to indicate that a variable may change asynchronously.

while: Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed.

❖ JAVA FLOW CONTROL:

Java if (if-then) Statement

The syntax of if-then statement in Java is:

```
if (expression) {
   // statements
}
```

NB->Here expression is a boolean expression (returns either true or false).

Example of Java if Statement:

```
         class IfStatement {
       public static void main(String[] args) {
             int number = 10;
             if (number > 0) {
                          System.out.println("Number is positive.");
             }
             System.out.println("This statement is always executed.");
       }
   }
```

Java if...else (if-then-else) Statement:

The if statement executes a certain section of code if the test expression is evaluated to true. The if statement may have an optional else block. Statements inside the body of else statement are executed if the test expression is evaluated to false.

The syntax of if-then-else statement is:

```
if (expression) {
  // codes
}
else {
 // some other code
}
```

Example of Java if else Statement:

```
 class IfElse {
      public static void main(String[] args) {
         int number = 10;
         if (number > 0) {
           System.out.println("Number is positive.");
         }
         else {
           System.out.println("Number is not positive.");
         }

         System.out.println("This statement is always executed.");
      }
   }
```

Java if..else..if Statement:

In Java, it's possible to execute one block of code among many. For that, you can use if..else...if ladder.

Syntax:

```
if (expression1)
{
  // codes
}
else if(expression2)
{
  // codes
}
else if (expression3)
{
  // codes
}
.
.
else
{
  // codes
}
```

Example of Java if..else..if Statement

```
 class Ladder {
     public static void main(String[] args) {
        int number = 0;
         if (number > 0) {
          System.out.println("Number is positive.");
        }
        else if (number < 0) {
          System.out.println("Number is negative.");
        }
        else {
          System.out.println("Number is 0.");
        }
     }
   }
```

Java Nested if..else Statement:

It's possible to have if..else statements inside a if..else statement in Java. It's called nested if...else statement.

Here's a program to find largest of 3 numbers:

Example of Nested if...else Statement

```
   class Number {
      public static void main(String[] args) {
         Double n1 = -1.0, n2 = 4.5, n3 = -5.3, largestNumber;
         if (n1 >= n2) {
            if (n1 >= n3) {
```

```
            largestNumber = n1;
         } else {
            largestNumber = n3;
         }
      } else {
         if (n2 >= n3) {
            largestNumber = n2;
         } else {
            largestNumber = n3;
         }
      }
      System.out.println("Largest number is " + largestNumber);
   }
}
```

Java switch Statement

In this article, you will learn to use switch statement to control the flow of your program's execution

In Java, the if..else..if ladder executes a block of code among many blocks. The switch statement can a substitute for long if..else..if ladders which generally makes your code more readable.

The syntax of switch statement is:

```
switch (variable/expression) {
case value1:
  // statements
  break;
case value2:
  // statements
  break;
  .. .. ...
  .. .. ...
default:
  // statements
}
```

The switch statement evaluates it's expression (mostly variable) and compares with values(can be expression) of each case label.

The switch statement executes all statements of the matching case label.

Suppose, the variable/expression is equal to value2. In this case, all statements of that matching case is executed.

Notice, the use of break statement. This statement terminates the execution of switch statement. The break statements are important because if they are not used, all statements after the matching case label are executed in sequence until the end of switch statement.

Java for Loop

Loop is used in programming to repeat a specific block of code. In this article, you will learn to create a for loop in Java programming.

Loop is used in programming to repeat a specific block of code until certain condition is met (test expression is false).

Java for Loop

The syntax of for Loop in Java is:

```
for (initialization; testExpression; update)
{
   // codes inside for loop's body}
```

Java Inheritance

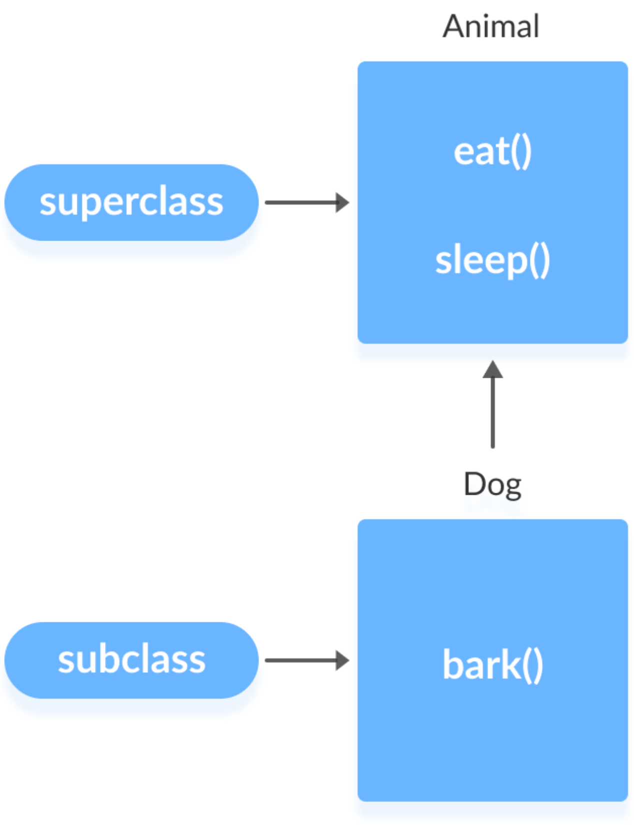
In this tutorial, we will learn about inheritance in Java with the help of examples.

Inheritance is one of the key features of OOP (Object-oriented Programming) that allows us to define a new class from an existing class. For example,

```
class Animal
{
   // eat() method
   // sleep() method
}
class Dog extends Animal
{
   // bark() method
}
```

In Java, we use the extends keyword to inherit from a class. Here, we have inherited the Dog class from the Animal class.

The Animal is the superclass (parent class or base class), and the Dog is a subclass (child class or derived class). The subclass inherits the fields and methods of the superclass.



---

is-a relationship

Inheritance is an **is-a** relationship. We use inheritance only if an **is-a** relationship is present between the two classes.

Here are some examples:

- A car is a vehicle.
- Orange is a fruit.
- A surgeon is a doctor.

- A dog is an animal.

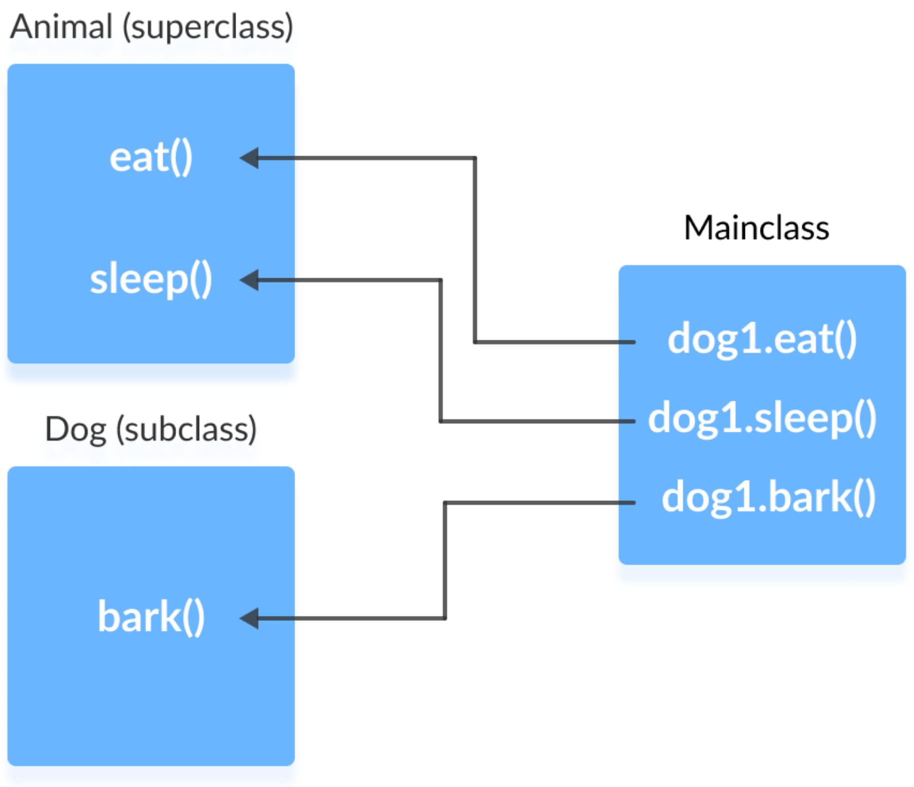
Example 1: Java Inheritance

```
class Animal {

  public void eat() {
    System.out.println("I can eat");
  }

  public void sleep() {
    System.out.println("I can sleep");
  }
}

class Dog extends Animal {
  public void bark() {
    System.out.println("I can bark");
  }
}

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

    Dog dog1 = new Dog();

    dog1.eat();
    dog1.sleep();

    dog1.bark();
  }
}
```

Output

```
I can eat
I can sleep
I can bark
```

Here, we have inherited a subclass Dog from superclass Animal. The Dog class inherits the methods eat() and sleep() from the Animal class.
Hence, objects of the Dog class can access the members of both the Dog class and the Animal class.



protected Keyword

We learned about private and public access modifiers in previous tutorials.

- private members can be accessed only within the class
- public members can be accessed from anywhere

You can also assign methods and fields protected. Protected members are accessible

- from within the class
- within its subclasses
- within the same package

Here's a summary of from where access modifiers can be accessed.

	Class	Package	subclass	World
public	Yes	Yes	Yes	Yes
private	Yes	No	No	No
protected	Yes	Yes	Yes	No

Example 2: protected Keyword

```
class Animal {
  protected String type;
  private String color;

  public void eat() {
    System.out.println("I can eat");
  }

  public void sleep() {
    System.out.println("I can sleep");
  }

  public String getColor(){
    return color;
  }

  public void setColor(String col){
    color = col;
  }
}

class Dog extends Animal {
  public void displayInfo(String c){
    System.out.println("I am a " + type);
    System.out.println("My color is " + c);
  }
  public void bark() {
    System.out.println("I can bark");
  }
}

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

    Dog dog1 = new Dog();
    dog1.eat();
    dog1.sleep();
    dog1.bark();

    dog1.type = "mammal";
    dog1.setColor("black");
    dog1.displayInfo(dog1.getColor());
  }
}
```
Output

```
I can eat
I can sleep
I can bark
I am a mammal
My color is black
```
Here, the type field inside the Animal class is protected. We have accessed this field from the Main class using

```
dog1.type = "mammal";
```
It is possible because Animal and Main class are in the same package (same file).

Java Method overriding

From the above examples, we know that objects of a subclass can also access methods of its superclass.

What happens if the same method is defined in both the superclass and subclass?

Well, in that case, the method in the subclass overrides the method in the superclass. For example,

Example 3: Method overriding Example

```
class Animal {
  protected String type = "animal";

  public void eat() {
    System.out.println("I can eat");
  }

  public void sleep() {
    System.out.println("I can sleep");
  }
}

class Dog extends Animal {

  @Override
  public void eat() {
    System.out.println("I eat dog food");
  }

  public void bark() {
    System.out.println("I can bark");
  }
}

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

    Dog dog1 = new Dog();
    dog1.eat();
    dog1.sleep();
    dog1.bark();
  }
}
```

Output

```
I eat dog food
I can sleep
I can bark
```

Here, eat() is present in both the superclass Animal and subclass Dog. We created an object dog1 of the subclass Dog.

When we call eat() using the dog1 object, the method inside the Dog is called, and the same method of the superclass is not called. This is called method overriding.

In the above program, we have used the @Overrideannotation to tell the compiler that we are overriding a method. However, it's not mandatory. We will learn about *method overriding* in detail in the next tutorial.

If we need to call the eat() method of Animal from its subclasses, we use the super keyword.

Example 4: super Keyword

```
class Animal {
  public Animal() {
    System.out.println("I am an Animal");
  }

  public void eat() {
    System.out.println("I can eat");
  }
}

class Dog extends Animal {
  public Dog(){
    super();
    System.out.println("I am a dog");
  }

  @Override
  public void eat() {
    super.eat();
    System.out.println("I eat dog food");
  }

  public void bark() {
    System.out.println("I can bark");
  }
}

class Main {
  public static void main(String[] args) {
    Dog dog1 = new Dog();

    dog1.eat();
    dog1.bark();
  }
}
```

Output

```
I am an Animal
I am a dog
I can eat
I eat dog food
I can bark
```

Here, we have used the super keyword to call the constructor using super(). Also, we have called the eat() method of Animal superclass using super.eat().
Note the difference in the use of super while calling constructor and method. To learn more, visit the *super keyword***.**

Types of inheritance

There are five types of inheritance.

- **Single inheritance** - Class B extends from class A only.
- **Multilevel inheritance** - Class B extends from class A; then class C extends from class B.
- **Hierarchical inheritance** - Class A acts as the superclass for classes B, C, and D.

- **Multiple inheritance** - Class C extends from interfaces A and B.
- **Hybrid inheritance** - Mix of two or more types of inheritance.

Java doesn’t support multiple and hybrid inheritance through classes. However, we can achieve multiple inheritance in Java through interfaces. We will learn about interfaces in later chapters.

Why use inheritance?

- The most important use is the reusability of code. The code that is present in the parent class doesn’t need to be written again in the child class.
- To achieve runtime polymorphism through method overriding. We will learn more about polymorphism in later chapters.