

What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language. Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Java Example

```
1. class Simple{
2.     public static void main(String args[]){
3.         System.out.println("Hello Java");
4.     }
5. }
```

Application

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming

1) **Standalone Application**

Standalone applications are also known as desktop applications or window-based applications.

Media player, antivirus, etc

2) **Web Application**

An application that runs on the server side and creates a dynamic page is called a web application.

3) **Enterprise Application**

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security,

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications

Features of Java

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,

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 incorporate 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](#),

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

It has two components:

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

Java code can be executed 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

Bytecode Verifier:

Security Manager

C++ vs Java

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language .	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.

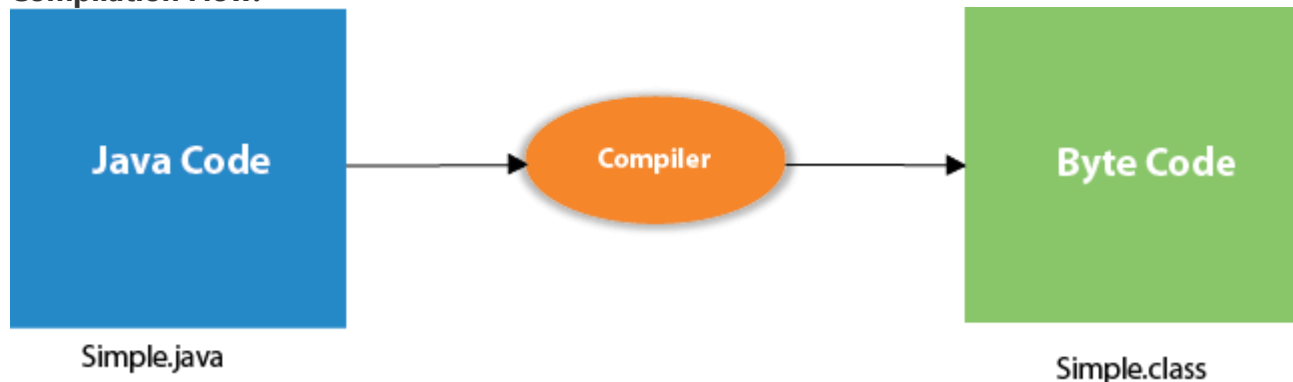
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java .
Operator Overloading	C++ supports operator overloading .	Java doesn't support operator overloading.
Pointers	C++ supports pointers . You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
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 structures 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 thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (<code>/** ... */</code>) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the inheritance .

		tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an <u>object-oriented</u> language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Note

- Java doesn't support default arguments like C++.
- Java does not support header files like C++. Java uses the import keyword to include different classes and methods.

Compilation Flow:



Parameters used in First Java Program

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

Difference between JDK, JRE, and JVM

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

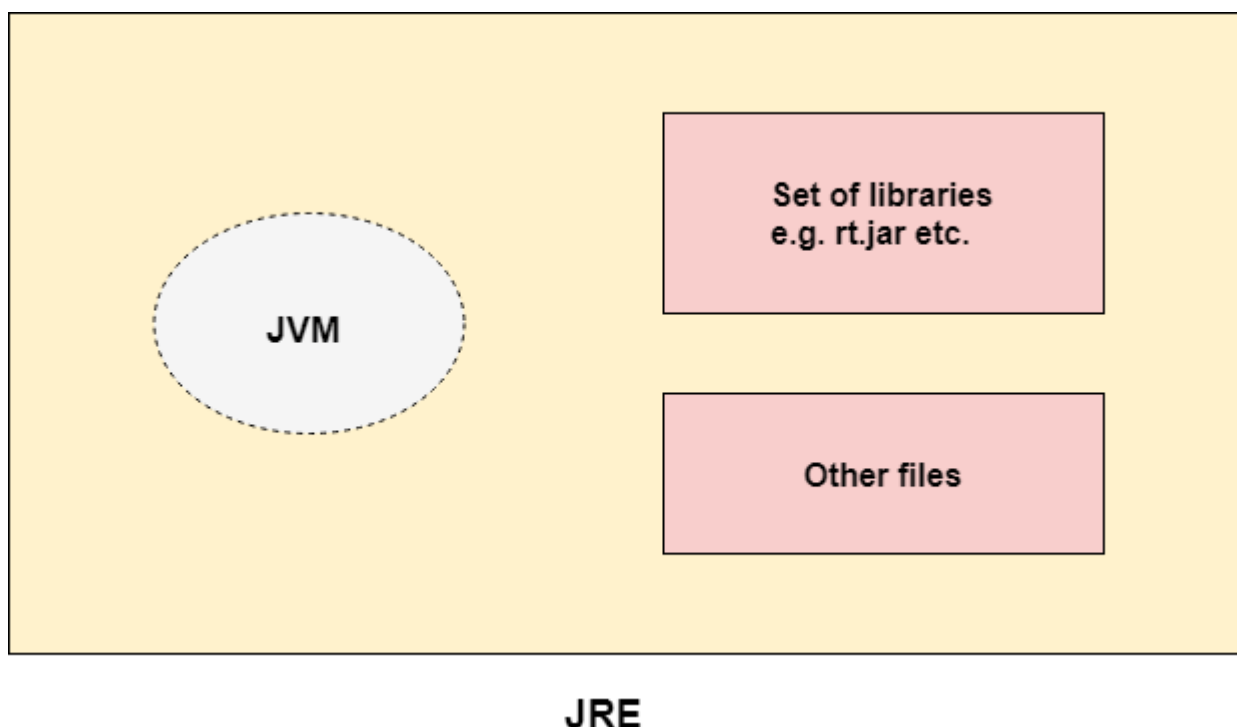
JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each [OS](#) is different from each other. However, Java is platform independent. There are three notions of the JVM: *specification*, *implementation*, and *instance*.

The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

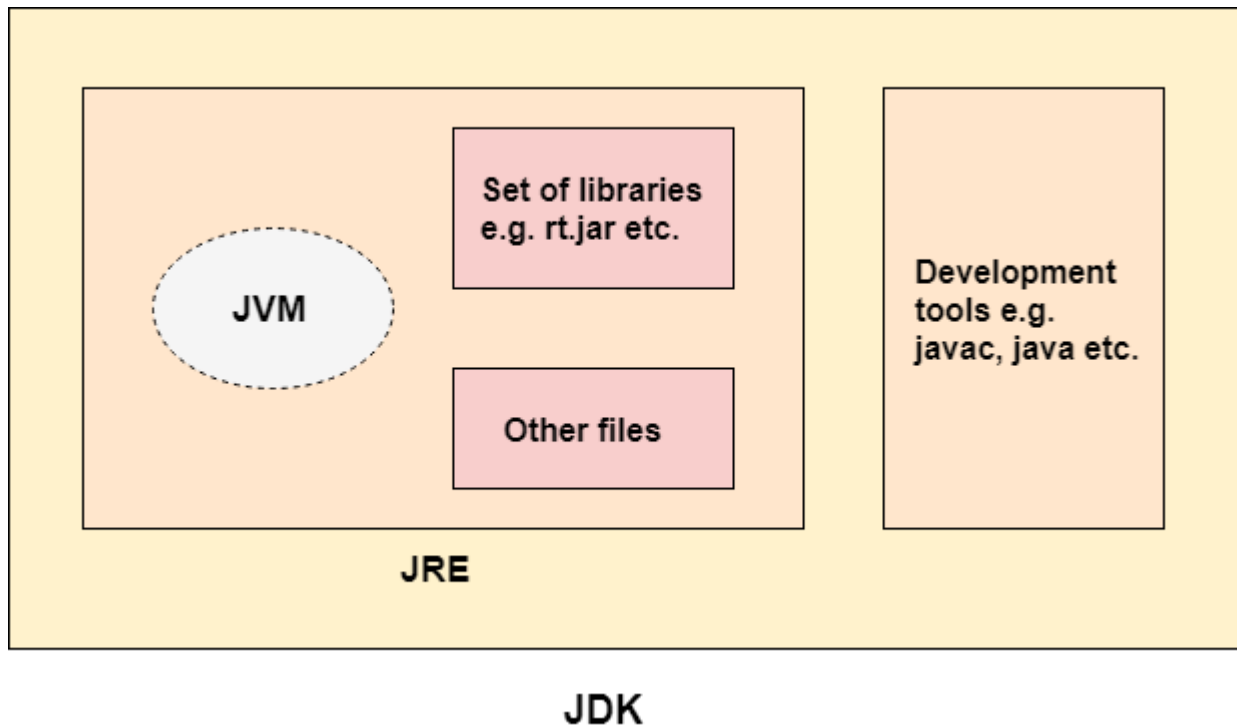
JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.



JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](#). It physically exists. It contains JRE + development tools.

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



Java Variables

A variable is a container which holds the value while the [Java program](#) is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of [data types in Java](#)

: primitive and non-primitive.

Types of Variables

1) Local Variable

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

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

2) Instance Variable

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

.

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

3) Static variable

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

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:

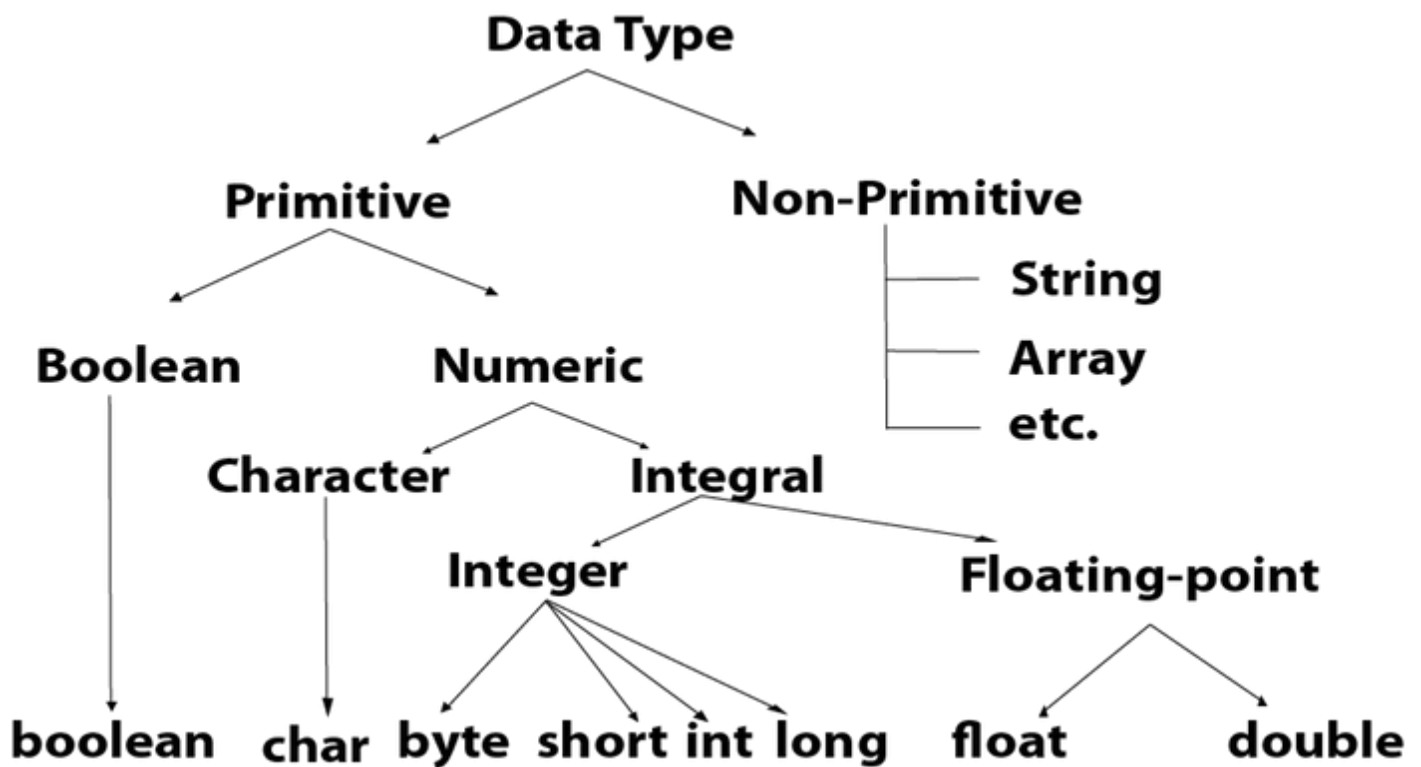
1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](#), [Interfaces](#), and [Arrays](#)

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](#)

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.

Byte Data Type

The byte data type is an example of primitive data type. It is an 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.

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.

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 -2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.

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 -9,223,372,036,854,775,808 and 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.

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

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.

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.

Operators in Java

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

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

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	<i>expr++ expr--</i>
	prefix	<i>++expr --expr +expr -expr ~ !</i>
Arithmetic	multiplicative	<i>* / %</i>
	additive	<i>+ -</i>
Shift	shift	<i><< >> >>></i>
Relational	comparison	<i>< > <= >= instanceof</i>
	equality	<i>== !=</i>
Bitwise	bitwise AND	<i>&</i>
	bitwise exclusive OR	<i>^</i>
	bitwise inclusive OR	<i> </i>
Logical	logical AND	<i>&&</i>
	logical OR	<i> </i>
Ternary	ternary	<i>? :</i>

Java Keywords

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

List of Java Keywords

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

1. **abstract:** Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. **boolean:** Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break:** Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. **byte:** Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. **case:** Java case keyword is used with the switch statements to mark blocks of text.
6. **catch:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. **char:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class:** Java class keyword is used to declare a class.
9. **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.
10. **default:** Java default keyword is used to specify the default block of code in a switch statement.
11. **do:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. **double:** Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. **else:** Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.

16. **final:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. **float:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. **for:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. **if:** Java if keyword tests the condition. It executes the if block if the condition is true.
21. **implements:** Java implements keyword is used to implement an interface.
22. **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. **instanceof:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface:** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new:** Java new keyword is used to create new objects.
29. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package:** Java package keyword is used to declare a Java package that includes the classes.
31. **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.
32. **protected:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. **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.
34. **return:** Java return keyword is used to return from a method when its execution is complete.
35. **short:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. **static:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. **strictfp:** Java strictfp is used to restrict the floating-point calculations to ensure portability.

38. **super:** Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. **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.
40. **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. **this:** Java this keyword can be used to refer the current object in a method or constructor.
42. **throw:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. **throws:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. **transient:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. **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.
46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. **volatile:** Java volatile keyword is used to indicate that a variable may change asynchronously.
48. **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, it is recommended to use the while loop.

Java Control Statements | Control Flow in Java

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.

Java provides three types of control flow statements.

1. Decision Making statements
 - if statements
 - 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:

1) If Statement:

1. Simple if statement

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

2. if-else statement

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

3. if-else-if ladder

```
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  
}
```

4. Nested if-statement

```
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  
    }  
}
```

Switch Statement:

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

Loop Statements

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

Java for loop

```
for(initialization, condition, increment/decrement) {  
  //block of statements  
}
```

Java while loop

```
while(condition){  
  //looping statements  
}
```

Java do-while loop

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

Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

Java break statement

As the name suggests, the [break statement](#) is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

BreakExample.java

```
public class BreakExample {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        for(int i = 0; i <= 10; i++) {  
            System.out.println(i);  
            if(i == 6) {  
                break;  
            }  
        }  
    }  
}
```

Java continue statement

Unlike break statement, the [continue statement](#) doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

```
public class ContinueExample {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        for(int i = 0; i <= 2; i++) {  
  
            for (int j = i; j <= 5; j++) {  
  
                if(j == 4) {  
                    continue;  
                }  
                System.out.println(j);  
            }  
        }  
    }  
}
```

```
}
```

```
}
```

Java Comments

The [Java](#) comments are the statements in a program that are not executed by the compiler and interpreter.

Types of Java Comments

There are three types of comments in Java.

1. Single Line Comment

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

2. Multi Line Comment

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

Java OOPs Concepts

In this page, we will learn about the basics of OOPs. Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

[Object](#)

Class

[Inheritance](#)

[Polymorphism](#)

[Abstraction](#)

Encapsulation

Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

- One to One
- One to Many
- Many to One, and
- Many to Many

Advantage of OOPs over Procedure-oriented programming language

- 1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.
- 2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.
- 3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

What is the difference between an object-oriented programming language and object-based programming language?

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

Naming Conventions of the Different Identifiers

The following table shows the popular conventions used for the different identifiers.

Identifiers Type	Naming Rules	Examples
Class	It should start with the uppercase letter. It should be a noun such as Color, Button, System, Thread, etc. Use appropriate words, instead of acronyms.	public class Employee { //code snippet }
Interface	It should start with the uppercase letter. It should be an adjective such as Runnable, Remote, ActionListener. Use appropriate words, instead of acronyms.	interface Printable { //code snippet }
Method	It should start with lowercase letter. It should be a verb such as main(), print(), println(). If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().	class Employee { // method void draw() { //code snippet } }

Variable	<p>It should start with a lowercase letter such as id, name. It should not start with the special characters like & (ampersand), \$ (dollar), _ (underscore). If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.</p> <p>Avoid using one-character variables such as x, y, z.</p>	<pre>class Employee { // variable int id; //code snippet }</pre>
Package	<p>It should be a lowercase letter such as java, lang. If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.</p>	<pre>//package package com.javatpoint; class Employee { //code snippet }</pre>
Constant	<p>It should be in uppercase letters such as RED, YELLOW. If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY. It may contain digits but not as the first letter.</p>	<pre>class Employee { //constant static final int MIN_AGE = 18; //code snippet }</pre>

Objects and Classes in Java

What is an object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical

What is a class in Java

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

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Syntax to declare a class:

```
class <class_name>{  
    field;  
    method;  
}
```

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Object and Class Example: main outside the class

```
class Student{  
    int id;  
    String name;  
}  
class TestStudent1{  
    public static void main(String args[]){  
        Student s1=new Student();  
        System.out.println(s1.id);  
        System.out.println(s1.name);  
    }  
}
```

3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructo

What are the different ways to create an object in Java?

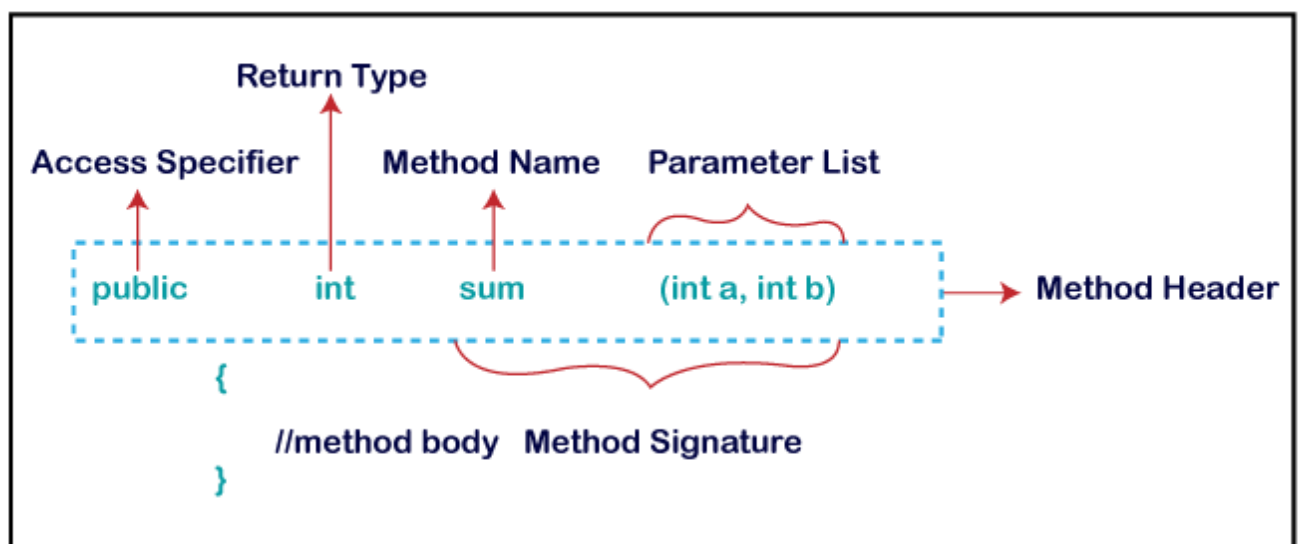
There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By deserialization
- By factory method etc

What is a method in Java?

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

Method Declaration



Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.

- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length()**, **equals()**, **compareTo()**, **sqrt()**, etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

User-defined Method

The method written by the user or programmer is known as a **user-defined** method. These methods are modified according to the requirement.

How to Create a User-defined Method

```
public static void findEvenOdd(int num)
{
    if(num%2==0)
        System.out.println(num+" is even");
    else
        System.out.println(num+" is odd");
}
```

How to Call or Invoke a User-defined Method

```
import java.util.Scanner;
public class EvenOdd
{
    public static void main (String args[])
```



```

{
//creating Scanner class object
Scanner scan=new Scanner(System.in);
System.out.print("Enter the number: ");
//reading value from the user
int num=scan.nextInt();
//method calling
findEvenOdd(num);
}

```

Static Method

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

Example of static method

```

public class Display
{
    public static void main(String[] args)
    {
        show();
    }
    static void show()
    {
        System.out.println("It is an example of static method.");
    }
}

```

Instance Method

The method of the class is known as an **instance method**. It is a **non-static** method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class. Let's see an example of an instance method.

Abstract Method

The method that does not has method body is known as abstract method. In other words, without an implementation is known as abstract method. It always declares in the **abstract class**. It means the class itself must be abstract if it has abstract method. To create an abstract method, we use the keyword **abstract**.

Syntax

```
abstract void method_name();
```

Example of abstract method

```
abstract class Demo
{
    abstract void display();
}

public class MyClass extends Demo
{
    void display()
    {
        System.out.println("Abstract method?");
    }

    public static void main(String args[])
    {
        Demo obj = new MyClass();
        obj.display();
    }
}
```

Factory method

It is a method that returns an object to the class to which it belongs. All static methods are factory methods. For example, **NumberFormat obj = NumberFormat.getNumberInstance();**

Constructors in Java

In [Java](#)

, a constructor is a block of codes similar to the method. It is called when an instance of the [class](#) is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type

3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>(){}
```

Example of default constructor

```
class Bike1{
    Bike1(){System.out.println("Bike is created");}
    public static void main(String args[]){
        Bike1 b=new Bike1();
    }
}
```

Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

1. **class** Student4{
2. **int** id;
3. String name;
4. //creating a parameterized constructor
5. Student4(**int** i,String n){
6. id = i;

```

7.     name = n;
8.     }
9.     //method to display the values
10.    void display(){System.out.println(id+ " "+name);}
11.
12.    public static void main(String args[]){
13.        //creating objects and passing values
14.        Student4 s1 = new Student4(111,"Karan");
15.        Student4 s2 = new Student4(222,"Aryan");
16.        //calling method to display the values of object
17.        s1.display();
18.        s2.display();
19.    }
20.}

```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor [overloading in Java](#) is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```

1. class Student5{
2.     int id;
3.     String name;
4.     int age;
5.     //creating two arg constructor
6.     Student5(int i,String n){
7.         id = i;
8.         name = n;
9.     }
10.    //creating three arg constructor
11.    Student5(int i,String n,int a){
12.        id = i;
13.        name = n;
14.        age=a;
15.    }
16.    void display(){System.out.println(id+ " "+name+" "+age);}

```

```

17.
18. public static void main(String args[]){
19.     Student5 s1 = new Student5(111,"Karan");
20.     Student5 s2 = new Student5(222,"Aryan",25);
21.     s1.display();
22.     s2.display();
23. }
24. }

```

Difference between constructor and method in Java

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Java Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

What is the purpose of Constructor class?

Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the `java.lang.reflect` package.

Java static keyword

The **static keyword** in [Java](#)

is used for memory management mainly. We can apply static keyword with [variables](#), methods, blocks and [nested classes](#). The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

It makes your program **memory efficient** (i.e., it saves memory).

2) Java static method

If you apply static keyword with any method, it is known as static method.

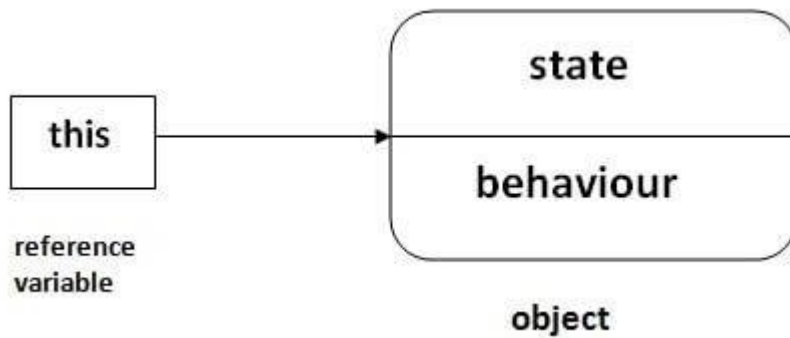
- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

this keyword in Java

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.



Usage of Java this Keyword

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

01

this can be used to refer current class instance variable.

04

this can be passed as an argument in the method call.

02

this can be used to invoke current class method (implicity)

05

this can be passed as argument in the constructor call.

03

this() can be used to invoke current class Constructor.

06

this can be used to return the current class instance from the method

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](#)

(Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](#)

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

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

Why use inheritance in java

- For [Method Overriding](#)

(so [runtime polymorphism](#)

can be achieved).

- For Code Reusability.

Terms used in Inheritance

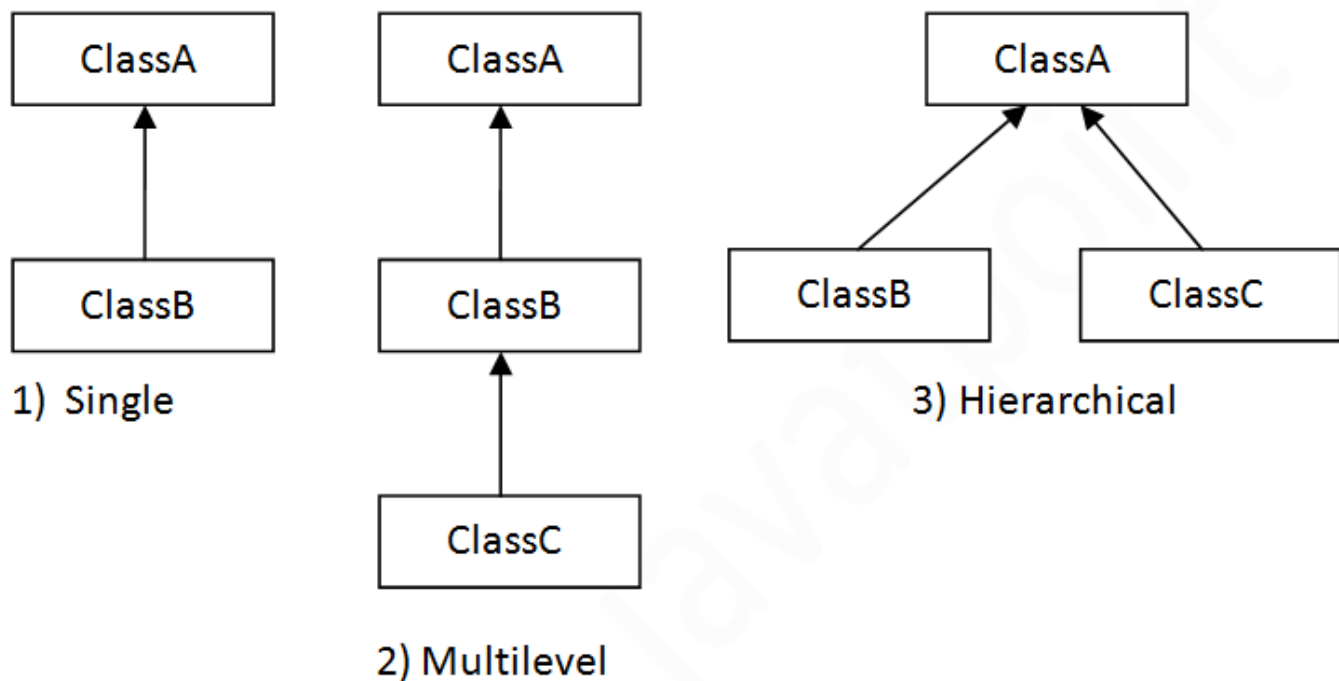
- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent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

The syntax of Java Inheritance

1. **class** Subclass-name **extends** Superclass-name
2. {
3. //methods and fields
4. }

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality. In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Types of inheritance in java



Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Aggregation in Java

If a class has an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Polymorphism

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Advantage of method overloading

Method overloading *increases the readability of the program.*

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1) Method Overloading: changing no. of arguments

2) Method Overloading: changing data type of arguments

Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Example of method overriding

```
class Vehicle{
    //defining a method
    void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
    //defining the same method as in the parent class
    void run(){System.out.println("Bike is running safely");}
```

```
public static void main(String args[]){  
    Bike2 obj = new Bike2();//creating object  
    obj.run();//calling method  
}  
}
```

Output:

```
Bike is running safely
```

Can we override static method?

No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

Why can we not override static method?

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

Can we override java main method?

No, because the main is a static method.

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class construct

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

Polymorphism in Java

Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

1. Static Binding (also known as Early Binding).

When type of the object is determined at compiled time (by the compiler), it is known as static binding. If there is any private, final or static method in a class, there is static binding.

2. Dynamic Binding (also known as Late Binding).

When type of the object is determined at run-time, it is known as dynamic binding.

Java instanceof

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in [Java](#).

. It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have [constructors](#) and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of Abstract class that has an abstract method

```
1. abstract class Bike{
2.   abstract void run();
3. }
4. class Honda4 extends Bike{
5.   void run(){System.out.println("running safely");}
6.   public static void main(String args[]){
7.     Bike obj = new Honda4();
8.     obj.run();
9.   }
10.}
```

Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve* [abstraction](#)

. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](#).

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.

- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How to declare an interface?

Syntax:

1. **interface** <interface_name>{
- 2.
3. // declare constant fields
4. // declare methods that abstract
5. // by default.
6. }

Difference between abstract class and interface

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance .	Interface supports multiple inheritance .
3) Abstract class can have final, non-final, static and non-static variables .	Interface has only static and final variables .
4) Abstract class can provide the implementation of interface .	Interface can't provide the implementation of abstract class .
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7) An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.
9) Example: <pre> public abstract class Shape{ public abstract void draw(); }</pre>	Example: <pre> public interface Drawable{ void draw(); }</pre>

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- 2) Java package provides access protection.
- 3) Java package removes naming collision.

Simple example of java package

```
1. package mypack;
2. public class Simple{
3.     public static void main(String args[]){
4.         System.out.println("Welcome to package");
5.     }
6. }
```

Subpackage in java

Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.

Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.

We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

3. **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Encapsulation in Java

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.

It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

The encapsulate class is **easy to test**. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

Object class in Java

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

Simple example of command-line argument in java

1. **class** CommandLineExample{
2. **public static void** main(String args[]){
3. System.out.println("Your first argument is: "+args[0]);
4. }
5. }

Difference between object and class

No.	Object	Class
1)	Object is an instance of a class.	Class is a blueprint or template from which objects are created.
2)	Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a group of similar objects .
3)	Object is a physical entity.	Class is a logical entity.
4)	Object is created through new keyword mainly e.g. Student s1=new Student();	Class is declared using class keyword e.g. class Student{}
5)	Object is created many times as per requirement.	Class is declared once .
6)	Object allocates memory when it is created .	Class doesn't allocated memory when it is created .
7)	There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only one way to define class in java using class keyword.

Class: Human **Object:** Man, Woman

Class: Fruit **Object:** Apple, Banana, Mango, Guava wtc.

Class: Mobile phone **Object:** iPhone, Samsung, Moto

Class: Food **Object:** Pizza, Burger, Samosa

Difference between method overloading and method overriding in java

No.	Method Overloading	Method Overriding
1)	Method overloading is used to <i>increase the readability</i> of the program.	Method overriding is used to <i>provide the specific implementation</i> of the method that is already provided by its super class.
2)	Method overloading is performed <i>within class</i> .	Method overriding occurs <i>in two classes</i> that have IS-A (inheritance) relationship.
3)	In case of method overloading, <i>parameter must be different</i> .	In case of method overriding, <i>parameter must be same</i> .
4)	Method overloading is the example of <i>compile time polymorphism</i> .	Method overriding is the example of <i>run time polymorphism</i> .
5)	In java, method overloading can't be performed by changing return type of the method only. <i>Return type can be same or different</i> in method overloading. But you must have to change the parameter.	<i>Return type must be same or covariant</i> in method overriding.

Java Method Overloading example

1. **class** OverloadingExample{
2. **static int** add(**int** a,**int** b){**return** a+b;}
3. **static int** add(**int** a,**int** b,**int** c){**return** a+b+c;}
4. }

Java Method Overriding example

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** eat(){System.out.println("eating bread...");}
6. }

Java String

In [Java](#)

, string is basically an object that represents sequence of char values. An [array](#)

of characters works same as Java string. For example:

```
String s="javatpoint";
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

Dictionary Meaning: Exception is an abnormal condition.

Java try-catch block

Java try block

Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.

Syntax of Java try-catch

1. **try**{
2. *//code that may throw an exception*
3. **catch**(Exception_class_Name ref){}

Syntax of try-finally block

1. **try**{
2. *//code that may throw an exception*
3. **finally**{}

Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

Java finally block

Java finally block is a block used to execute important code such as closing the connection, etc.

Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception. So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.

Difference between throw and throws in Java

Sr. no.	Basis of Differences	throw	throws
1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.	
3.	Syntax	The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

Difference between final, finally and finalize

Sr. no.	Key	final	finally	finalize
---------	-----	-------	---------	----------

1.	Definition	final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
2.	Applicable to	Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
3.	Functionality	(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
4.	Execution	Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependant on the exception.	finalize method is executed just before the object is destroyed.

Exception Handling with Method Overriding in Java

There are many rules if we talk about method overriding with exception handling.

Some of the rules are listed below:

- **If the superclass method does not declare an exception**
 - If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
- **If the superclass method declares an exception**
 - If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

Java Threads | How to create a thread in Java

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

Java AWT Tutorial

Java AWT (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.

Java Swing Tutorial

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Java JDBC Tutorial

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers: