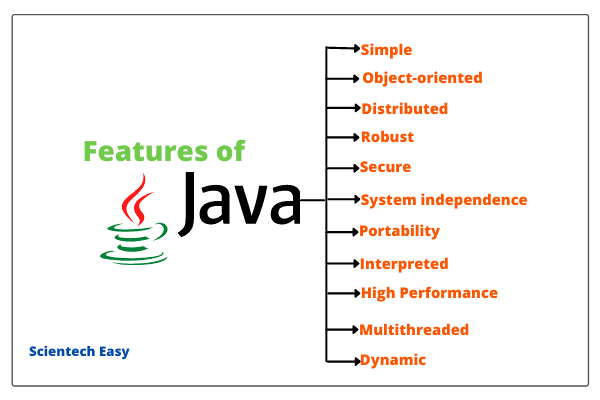
**What is Java?**

**Java** is a powerful, versatile, and simple general-purpose programming language. It is one of the most widely used high-level programming languages in the world.

Java is a full-featured general-purpose programming language that is used for developing platform-independent software (applications) running on desktop computers, mobile devices, and servers.

**Features of Java | Java Buzzwords**



### **List the features of Java Programming language.**

There are the following features in Java Programming Language.

* **Simple:** Java is easy to learn. The syntax of Java is based on C++ which makes easier to write the program in it.
* **Object-Oriented:** Java follows the object-oriented paradigm which allows us to maintain our code as the combination of different type of objects that incorporates both data and behavior.
* **Portable:** Java supports read-once-write-anywhere approach. We can execute the Java program on every machine. Java program (.java) is converted to bytecode (.class) which can be easily run on every machine.
* **Platform Independent:** Java is a platform independent programming language. It is different from other programming languages like C and C++ which needs a platform to be executed. Java comes with its platform on which its code is executed. Java doesn't depend upon the operating system to be executed.
* **Secured:** Java is secured because it doesn't use explicit pointers. Java also provides the concept of ByteCode and Exception handling which makes it more secured.
* **Robust:** Java is a strong programming language as it uses strong memory management. The concepts like Automatic garbage collection, Exception handling, etc. make it more robust.
* **Architecture Neutral:** Java is architectural neutral as it is not dependent on the architecture. In C, the size of data types may vary according to the architecture (32 bit or 64 bit) which doesn't exist in Java.
* **Interpreted:** Java uses the Just-in-time (JIT) interpreter along with the compiler for the program execution.
* **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++).
* **Multithreaded:** 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.

**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.

* **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++.

### **Q3. Why Java is platform independent?**

Java is called platform independent because of its byte codes which can run on any system irrespective of its underlying operating system.

Or

**Why java is platform independent?**

The most unique feature of java is platform independent. In any programming language soruce code is compiled in to executable code . This cannot be run across all platforms. When javac compiles a java program it generates an executable file called .class file.

class file contains byte codes. Byte codes are interpreted only by JVM’s . Since these JVM’s are made available across all platforms by Sun Microsystems, we can execute this byte code in any platform. Byte code generated in windows environment can also be executed in linux environment. This makes java platform independent.

# Q.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](https://www.javatpoint.com/c-programming-language-tutorial). | 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. |
| **Multiple inheritance** | C++ supports multiple inheritance. | Java doesn't support multiple inheritance through class. It can be achieved by using [interfaces in java](https://www.javatpoint.com/interface-in-java). |
| **Operator Overloading** | C++ supports [operator overloading](https://www.javatpoint.com/cpp-overloading). | Java doesn't support operator overloading. |
| **Pointers** | C++ supports [pointers](https://www.javatpoint.com/cpp-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](https://www.javatpoint.com/multithreading-in-java) support. |
| **Documentation comment** | C++ doesn't support documentation comments. | Java supports documentation comment (/\*\* ... \*/) to create documentation for java source code. |
| **Object-oriented** | C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible. | Java is also an [object-oriented](https://www.javatpoint.com/java-oops-concepts) 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. |

Or

1. **What is difference between c++ and Java ?**

|  |  |
| --- | --- |
| Java | C++ |
| 1) Java is platform independent | C++ is platform dependent. |
| 2) There are no pointers in java | There are pointers in C++. |
| 3) There is no operator overloading in java | C ++ has operator overloading. |
| 4) There is garbage collection in java | There is no garbage collection |
| 5) Supports multithreading | Does’nt support multithreading |
| 6) There are no templates in java | There are templates in java |
| 7) There are no global variables in java | There are global variables in c++ |

### **Q12. Why pointers are not used in Java?**

Java doesn’t use pointers because they are unsafe and increases the complexity of the program. Since, Java is known for its simplicity of code, adding the concept of pointers will be contradicting. Moreover, since JVM is responsible for implicit memory allocation, thus in order to avoid direct access to memory by the user,  pointers are discouraged in Java.

### **Q. Explain JDK, JRE and JVM?**

|  |  |  |
| --- | --- | --- |
| **JDK** | **JRE** | **JVM** |
| The full form of JDK is Java Development Kit. | The full form of JRE is Java Runtime Environment. | The full form of JVM is Java Virtual Machine. |
| JDK is a software development kit to develop applications in Java. | It is a software bundle which provides Java class libraries with necessary components to run Java code. | JVM executes Java byte code and provides  an environment for executing it. |
| JDK is platform dependent. | JRE is also platform dependent. | JVM is highly platform dependent. |
| It contains tools for developing, debugging, and monitoring java code. | It contains class libraries and other supporting files that JVM requires to execute the program. | Software development tools are not  included in JVM. |
| It is the superset of JRE | It is the subset of JDK. | JVM is a subset of JRE. |
| The JDK enables developers to create Java programs that can be executed and run by the JRE and JVM. | The JRE is the part of Java that creates the JVM. | It is the Java platform component that  executes source code. |
| JDK comes with the installer. | JRE only contain environment to execute source code. | JVM bundled in both software  JDK and JRE. |

**What is difference between length and length() method in java ?**

length() : In String class we have length() method which is used to return the number of characters in string.

## length()

* The **length()** method is a**static method**of **String class**.
* The**length()**returns the **length of a string object** i.e. the number of characters stored in an object.
* **String class** uses this method because the length of a string can be modified using the various operations on an object.
* The **String** class internally uses a **char[] array** that it does not expose to the outside world.

Ex : String str = “Hello World”;

System.out.println(str.length());

Str.length() will return 11 characters including space.

length : we have length instance variable in arrays which will return the number of values or objects in array. For example :

String days[]={” Sun”,”Mon”,”wed”,”thu”,”fri”,”sat”}; Will return 6 since the number of values in days array is 6.

## length

* An **array is an object** that holds a **fixed number of values of the same type**.
* The **length** variable in an array returns the **length of an array** i.e. a **number of elements stored in an array**.
* Once arrays are initialized, its **length cannot be changed**, so the length variable can directly be used to get the length of an array.
* The **length** **variable** is used only for an **array**.

# Bytecode in Java

**Bytecode in Java** is a highly optimized set of instructions for the Java Virtual Machine (JVM) that reads and interprets to run the java program.

A bytecode is a binary program code that can only run on JVM. In other words, it is a machine language (code) for JVM in the form of .class file but it is not machine specific because it is not a native code.

## **Difference between Bytecode vs Machine code**

The main difference between the byte code and machine code is that byte code can be run only on JVM whereas machine code is a set of instructions in machine language that can be directly run by the CPU.

**Q.Difference between compiler and interpreter**

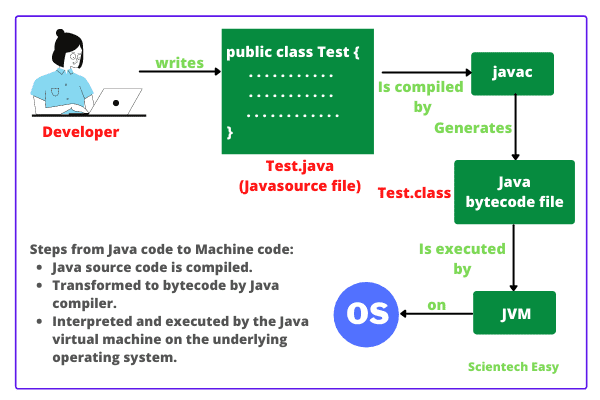
| S.No. | Compiler | Interpreter |
| --- | --- | --- |
| 1. | It is scan the entire programs first and translate it into machine  Code. | It scan the program line by line  And translate it into machine code. |
| 2. | Compiler show all errors  And running at same line | It show one error of a line |
| 3. | Error occur after scanning the whole program. | Error occur after scanning each  line |
| 4. | It converts the source code into object code. | It does not convert source code into object code instead it scans it line by line |
| 5 | It does not require source code for later execution. | It requires source code for later execution. |
| Eg. | C, C++, C# etc. | Python, Ruby, Perl, SNOBOL, MATLAB, etc. |

## **Q.How JVM works Internally?**

Java Virtual Machine performs the following operations for execution of the program. They are as follows:

a) Load the code into memory.  
b) Verifies the code.  
c) Executes the code  
d) Provides runtime environment.

When we make a program in Java, .java program code is converted into a .class file consisting of byte code instructions by the Java compiler. This Java compiler is outside of JVM.



## **What is JIT Compiler in Java?**

**JIT compiler in Java** is the part of JVM that is used to increase the speed of execution of a Java program.

In other words, it is used to improve the performance of the execution of the program. It helps to reduce the amount of time needed for the execution of the program.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Difference Between Stack and Heap Memory** The following table summarizes all the major differences between stack memory and heap space.   |  |  |  | | --- | --- | --- | | **Parameter** | **Stack Memory** | **Heap Space** | | **Application** | It stores items that have a very short life such as **methods, variables,** and **reference variables** of the objects. | It stores **objects** and Java Runtime Environment  (**JRE**) classes. | | **Ordering** | It follows the **LIFO** order. | It does not follow any order because it is a  dynamic memory allocation and does not  have any fixed pattern for allocation and  Deallocation of memory blocks. | | **Flexibility** | It is **not flexible** because we cannot alter the allocated memory. | It is **flexible** because we can alter the  allocated memory. | | **Efficiency** | It has **faster** access, allocation, and deallocation. | It has **slower** access, allocation, and deallocation. | | **Memory Size** | It is **smaller** in size. | It is **larger** in size. | | **Generation of Space** | When a thread is created, the operating system automatically allocates the stack. | To create the heap space for the application,  the language first calls the operating system at run  time. | | **Exception Throws** | JVM throws the **java.lang.StackOverFlowError** if the stack size is greater than the limit. To avoid this error, increase the stack size. | JVM throws the **java.lang.OutOfMemoryError**  if the JVM is unable to create a new native method. | | **Allocation/ Deallocation** | It is done automatically by the **compiler**. | It is done manually by the **programmer**. | | **Order of allocation** | Memory allocation is **continuous**. | Memory allocated in **random** order. | | **Thread-Safety** | It is thread-safe because each thread has its own stack. | It is not thread-safe, so properly synchronization  of code is required. | |  |  |
|  |  |  |

### **Q29. What is a classloader in Java?**

The **Java ClassLoader** is a subset of JVM (Java Virtual Machine) that is responsible for loading the class files. Whenever a Java program is executed it is first loaded by the classloader. Java provides three built-in classloaders:

1. Bootstrap ClassLoader
2. Extension ClassLoader
3. System/Application ClassLoader

## **Variable**

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed

### **Types of Variables**

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* static variable

#### **1) Local Variable**

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

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

#### **2) Instance Variable**

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

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.

1. **public** **class** A
2. {
3. **static** **int** m=100;//static variable
4. **void** method()
5. {
6. **int** n=90;//local variable
7. }
8. **public** **static** **void** main(String args[])
9. {
10. **int** data=50;//instance variable
11. }
12. }//end of class

**Q.difference between instance and local variable?**

| **Instance Variable** | **Local Variable** |
| --- | --- |
| They are defined in class but outside the body of methods. | They are defined as a type of variable declared within programming blocks or subroutines. |
| These variables are created when an object is instantiated and are accessible to all constructors, methods, or blocks in class. | These variables are created when a block, method or constructor is started and the variable will be destroyed once it exits the block, method, or constructor. |
| These variables are destroyed when the object is destroyed. | These variables are destroyed when the constructor or method is exited. |
| It can be accessed throughout the class. | Its access is limited to the method in which it is declared. |
| They are used to reserving memory for data that the class needs and that too for the lifetime of the object. | They are used to decreasing dependencies between components I.e., the complexity of code is decreased. |
| These variables are given a default value if it is not assigned by code. | These variables do not always have some value, so there must be a value assigned by code. |
| It is not compulsory to initialize instance variables before use. | It is important to initialize local variables before use. |
| It includes access modifiers such as private, public, protected, etc. | It does not include any access modifiers such as private, public, protected, etc |

## **Features of Static Keyword in Java**

1. Static keyword in Java can be applied with variables, methods, inner classes, and blocks.

2. We cannot declare a class with static keyword but the inner class can be declared as static.

3. It belongs to the class than an instance of the class.

4. One basic rule of working with static keyword is that we cannot directly call instance members within the static area because the static members are linked with the class.

## **Use of Static Keyword**

* 1. The main purpose of using static keyword is that we can access the data, method, or block of the class without any object creation.
  2. 2. It is used to make the programs more memory efficient.

## **Static Variable (Class variable)**

If we declare any instance variable with a static modifier, it is known as **static variable in java**. A static variable is also known as class variable in java. It stores the value for a variable in a common memory location.

## **Q.How to access Static variable in Java?**

There are two ways to access a static variable in Java. They are as follows:  
1. A static variable can be accessed inside any other class using the class name. The general syntax to access a static variable is as follows:

A.x; // A is the class name and "x " is a static variable declared in that class.

2. Static variables can also be accessed by the object reference but an instance variable can only be accessed by the object reference. For example, We have a class named Student. We can create the object of the Student class like this:

Student s = new Student();

  s.x;

## **Q.Can we access instance members from static area directly?**

Instance members (non-static members) can be accessed only from instance area directly. But we cannot access it from the static area directly.

In other words, we cannot call instance member from the static member because a static variable stores values into the memory before the object creation whereas an instance variable stores into the memory after the object creation.

So, when we access instance members by static members as they are not present in the memory, it will give the compile-time error.

## **Q.Can we access static variables from instance and static methods?**

Yes, static members (static variables) can be accessed from both instance and static area (i.e. instance and static methods) directly using the class name or without the class name. But outside the class, we can call only using class name.

## **Advantage of Static variable in Java**

Class variable makes program memory efficient. That is, it saves memory. All static variables are stored in PermGen space of the heap memory.

## **Q. Difference between Static variable and Instance variable**

1. A static variable is also known as class variable whereas, instance variable is also known as non-static variable.

2. Class variable can be accessed inside a static block, instance block, static method, instance method, and method of the inner class whereas, instance variable can be accessed inside the instance block, instance method, and method of the inner class.

3. Class variable is always resolved during compile time whereas, instance variable is resolved during the runtime.

4. It is not serialized in Java whereas, instance variable is serialized in Java.

# Tokens in Java | Types of Tokens

**Java Tokens |**A java program is made up of a group of classes and methods.

Definition-Tokens are the various elements in the java program that are identified by [Java compiler](https://www.scientecheasy.com/2021/03/java-compiler.html/). A token is the smallest individual element (unit) in a program that is meaningful to the compile.

## **Types of Tokens**

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

* Reserved Keywords
* Identifiers
* Literals,
* Operators
* Separators

# Keywords in Java | Reserved Keywords

**Keywords in Java** are predefined words that have specific meanings to the [compiler](https://www.scientecheasy.com/2021/03/java-compiler.html/) and that meanings cannot be changed.

a) **Data Type:** Java language has reserved eight words as keywords for data types that are as follows:

* byte
* short
* int
* long
* float
* double
* char
* boolean

b) **Access Modifiers:** There are three reserved words as keywords for access modifiers by Java language. They are as follows:

* private
* protected
* public

These keywords are called access modifier keywords.

c) **Control Statement:** There are ten reserved words as keywords for the control statement by Java language. They are as:

* else
* switch
* case
* default
* for
* while
* do
* continue
* break
* goto [reserved but not in used]
* if

d) **Object and Classes:** There are seven reserved keywords for classes and objects by Java language. They are as follows:

* class
* interface
* extends
* implements
* this
* super
* new

e) **Modifiers:** There are eight modifier keywords reserved in the Java language. These keywords are also called non-access modifier keywords. They are as follows:

* static
* abstract
* synchronized
* volatile
* native
* transient
* strictfp
* final

f) **Exceptions:** There are five exception keywords that are reserved by Java language. They are as:

* try
* catch
* finally
* throw
* throws

g) **Package:** Java language has included two reserved keywords for a package that is as follows:

* package
* import

h) **Miscellaneous keywords:**

* return
* const (\*)
* instanceof
* void
* assert
* default
* enum

# Comments in Java

# It allows the programmers to compose and express their thoughts related to the code independently.

## **Q Why should we write Comments in Java Program?**

-> We should write comments related to the code because comments help programmers who read your code understand your thoughts.

### **What is Classes and Objects in java?**

## **Objects in Java**

An **object in Java** is any real-world thing that has properties and actions. In other words, an entity that has state and behavior is known as an object. Here, state represents properties and behavior represents actions or functionality.

**Characteristics of an object in Java**

An object has three characteristics:

1. **State:** State represents properties of an object. It is represented by instance variable/attribute of an object. The properties of an object are important because the outcome of functions depends on the properties.

2. **Behaviour:** Behaviour represents functionality or actions. It is represented by methods in Java.

3. **Identity:** Identity represents the unique name of an object. It differentiates one object from the other. The unique name of an object is used to identify the object.

**Let’s take a real-world example**

-> We are taking the example “person”. A person has three characteristics: Identity (name), State (properties), and behavior (actions or functionality)

## **Classes in Java**

In Java programming, a class is basically user-defined data types that act as a template for creating objects of the identical type. It represents the common properties and actions (functions) of an object.

In other words, a class can also be defined, as “a class is a group of objects which are common to all objects of one type”.

Or

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.

**Real-time example 1:**

Let us consider two objects Samsung Galaxy S4 and iPhone. Suppose Samsung Galaxy S4 have some properties like width = “6.98 cms”, height = “13.6 cm”, OS = “Android”, brand = “Samsung”, price = “1000$” and actions are call(), sendMessage(), browser(), share().

Now, suppose iPhone has some properties such as width = “5.86 cm”, height = “12.3 cms”, OS = “iOS”, brand = “Apple”, price = “1200$” and actions are call(), sendMessage(), browse(), share().

Both objects have some different properties and actions but the type is the same “Phone”. This is the class. i.e the name of the class is “Phone”.

## **Difference between Class and Object in Java**

* 1. A class is a user-defined data type whereas an object is an instance of class data type.  
     2. A class generates objects whereas an object gives life to a class.  
     3. Classes do not occupy memory location but objects occupy memory location.  
     4. Classes cannot be manipulated due to not available in the memory location but objects can be manipulated

|  |  |  |
| --- | --- | --- |
| **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. |

### **Q16. What is an object in Java and how is it created?**

An object is a real-world entity that has a state and behavior. An object has three characteristics:

1. State
2. Behavior
3. Identity

An object is created using the ‘new’ keyword. For example:

ClassName obj = new ClassName();

## **How to create Object in Java?**

In Java, an object of a class is created using the new keyword in three steps. They are as follows.

1. Declaration of a reference variable.
2. Creation of an object.
3. Linking the object and the reference variable.

College myCollege = new College();

## **New Keyword in Java**

In Java, a new operator is a special keyword which is used to create an object of the class. It allocates the memory to store an object during runtime and returns a reference to it. This reference is the address of the object in the heap memory allocated by the new operator.

### **Q. For which purpose we create an object of class in Java?**

-> 1. Since Java is a purely object-oriented programming language. So Everything is done in the form of objects only. Therefore, objects are required in the Java programming language.

-> 2. To store data temporarily in Java application, we require to create an object. The object provides temporary storage for our data.

-> 3. In Java, By creating an object, we can call the members of one class from another class. It is useful when we need to use common code in every class again and again

-> 4. To access members of any particular class, we have to create an object of the respective class

## **Q. How many ways to create object in Java?**

1. Using the new keyword
2. Using Class.forName
3. Using Clone.
4. Using Object Deserialization.
5. Using ClassLoader.

**Q. What is object cloning in Java?**

Object cloning in Java is the process of creating an exact copy of an object. It basically means the ability to create an object with a similar state as the original object. To achieve this, Java provides a method **clone()** to make use of this functionality. This method creates a new instance of the class of the current object and then initializes all its fields with the exact same contents of corresponding fields. To object clone(), the marker interface **java.lang.Cloneable**must be implemented to avoid any runtime exceptions. One thing you must note is Object clone() is a protected method, thus you need to override it.

### **Q.What is System.out.println()?**

1. System is a class that is predefined by Sun Microsystem.  
2. out is a variable declared in System class of type PrintStream.  
3. println is a method defined in PrintStream class.

# Types of Classes in Java

### **Q15. Define a Java Class.**

A class in Java is a blueprint which includes all your data.  A class contains fields (variables) and methods to describe the behavior of an object. Let’s have a look at the syntax of a class.

|  |  |
| --- | --- |
| 1  2  3 | class Abc {  member variables // class body  methods} |

classes are considered as a blueprint for creating objects.

## **1. Concrete Class:**

A class whose object can be created and whose all methods have body is called concrete class. Java object class is a concrete class. A concrete class can have both static and non-static members.

It can extend its superclass, an abstract class, or implement an interface if it implements all their methods. It has no abstract methods.

## **2. Static Class:**

In Java, [static](https://www.scientecheasy.com/2020/06/java-static-variable.html/) is a keyword that can be applied with variables, methods, inner classes, and blocks. We cannot declare a class with static keyword but inner class can be declared as static.

When an inner class is defined with a static modifier inside the body of another class, it is known as static nested class in Java.

### **3. Abstract Class:**

An [abstract class](https://www.scientecheasy.com/2020/05/java-abstraction.html/) is a class which is declared with abstract keyword. It is just like a normal class but has two differences.

1. We cannot create an object of this class. Only objects of its non-abstract (or concrete) sub-classes can be created.

2. It can have zero or more abstract methods that are not allowed in a non-abstract class (concrete class).

An abstract class is sometimes also known as base class or superclass. It contains undefined and unimplemented abstract method bodies that are implemented and defined by derived classes.

### **4. Final Class:**

When a class is declared with [final keyword](https://www.scientecheasy.com/2020/07/final-keyword-in-java.html/), it is called final class in Java. Final class means Restricting Inheritance!. It does not allow itself to be inherited by another class.

In other words, Java classes declared as a final cannot be extended (inherited). If you do not want to be a subclass, declare it final. A lot of classes in Java API are final.

### **5. Inner class:**

A class declared inside another class is known as nested class. The class which is a member of another class can be either static or non-static.

When a member class is declared with static, it is known as static nested class. The member class which is non-static is known as [inner class in java](https://www.scientecheasy.com/2020/06/inner-class-in-java.html/).

An inner class cannot have any kind of static member. So, the members of an inner class can be:

* Instance variables
* Instance methods
* Constructors
* Initializer block
* Inner class

Based on declaration and behaviors, there are basically four types of inner classes in Java. They are as follows:

1. Normal or Regular inner class
2. Method local inner class
3. Anonymous inner class
4. Static nested class

### **6. Public Class:**

When a class is declared with a public access control modifier, it is called public class. A public class is visible and accessible from anywhere. The instance of public class can be created from any other class.

When a class is declared as public, any program in any package can use the code or some of the code. But we do not put public with class, it is non-public (default) class.

### **7. Private class:**

An outer class (top-level class) cannot be declared with a private access modifier but an inner class can be declared as private. An inner class is a member of outer class.

By mistake, if you will try to use any other modifiers with top-level class, you will get a compile-time error: “Modifier private not allowed here“.

**Q. What is singleton class?**

### **8. Singleton Class:**

A singleton class in Java is a class that allows only one instance to be created. All its variables and methods will belong to just one instance. Singleton class concept is useful when we need to create only one object of class.

A good example of a singleton is a data class that contains all the data to be used across the entire application. There is no need to create more than one object.

The main disadvantage of a singleton class is that it cannot be reusable. This is because we can create only one object of class.

### **Q.What is singleton class in Java and how can we make a class singleton?**

Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

## **Predefined Classes in Java**

### **6. Scanner Class:**

Scanner class is present in java.util package. It is used to read input from a keyboard or text file. When scanner class receives input from the keyboard then it breaks input into several parts which are called tokens. These tokens are retrieved from the scanner object using methods such as next(), nextByte(), nextInt(), etc.

### **7. Wrapper Class:**

A class whose object warps or contains primitive data type is called wrapper class. It is present in java.lang package. It is used to convert primitive data type into object form. When we will create an object to a wrapper class, it contains a field where we can store a primitive data type.

That is, we can wrap a primitive value into a wrapper class object. The list of wrapper classes defined in java.lang package is Character, Byte, Short, Integer, Long, Float, Double, and Boolean.

#### **What Is Autoboxing And Unboxing?**

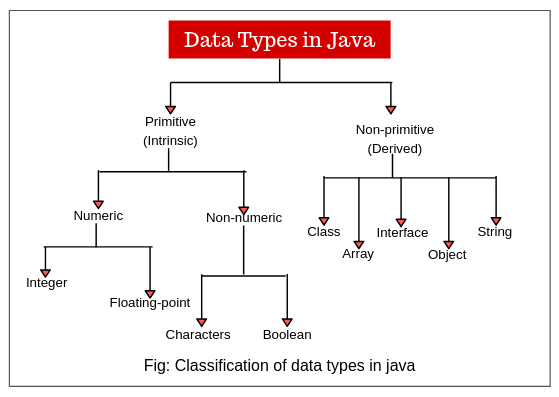
When Java automatically converts a primitive type into corresponding wrapper class object e.g. ***int to Integer,***than its called ***autoboxing because*** primitive is boxed into wrapper class while opposite case is called ***unboxing***, where an ***Integer***object is converted into primitive ***int***.

Since the whole process happens automatically without writing any code for conversion its called autoboxing and auto-unboxing.

Compiler uses ***valueOf()*** method to convert **primitive to Object**and uses ***xxxValue()*** i,e, intValue(), doubleValue() etc to get ***primitive value from Object***.

### **Q22. What is the difference between break and continue statements?**

|  |  |
| --- | --- |
| **break** | **continue** |
| 1. Can be used in switch and loop (for, while, do while) statements | 1. Can be only used with loop statements |
| 2. It causes the switch or loop statements to terminate the moment it is executed | 2. It doesn’t terminate the loop but causes the loop to jump to the next iteration |
| 3. It terminates the innermost enclosing loop or switch immediately | 3. A continue within a loop nested with a switch will cause the next loop iteration to execute |



# Packages in Java

A package is nothing but a physical folder structure (directories) that contains a group of related classes, interfaces, and sub-packages according to their functionality.

It provides a convenient way to organize your work. The Java language has various in-built packages.

For example, java.lang, java.util, java.io, and java.net. All these packages are defined as a very clear and systematic packaging mechanism for categorizing and managing.

### Core packages:

1. **Java.lang:** lang stands for language. The Java language package consists of java classes and interfaces that form the core of the Java language and the JVM. It is a fundamental package that is useful for writing and executing all Java programs.

Examples are classes, objects, String, Thread, predefined data types, etc. It is imported automatically into the Java programs.

2. **Java.io:** io stands for input and output. It provides a set of I/O streams that are used to read and write data to files. A stream represents a flow of data from one place to another place.

3. **Java util:** util stands for utility. It contains a collection of useful utility classes and related interfaces that implement data structures like LinkedList, Dictionary, HashTable, stack, vector, Calender, data utility, etc.

4. **Java.net:** net stands for network. It contains networking classes and interfaces for networking operations. The programming related to client-server can be done by using this package.

### **Q11. What is a package in Java? List down various advantages of packages.**

Packages in Java, are the collection of related classes and interfaces which are bundled together. By using packages, developers can easily modularize the code and optimize its reuse. Also, the code within the packages can be imported by other classes and reused. Below I have listed down a few of its advantages:

* Packages help in avoiding name clashes
* They provide easier access control on the code
* Packages can also contain hidden classes which are not visible to the outer classes and only used within the package
* Creates a proper hierarchical structure which makes it easier to locate the related classes

# Methods in Java

**Methods in Java** are the building block of a Java application. In Java,  a method is a set of code used to write the logic of the applications which perform some specific tasks or operations.

The main() method is the first method that is executed by [JVM (Java Virtual Machine)](https://www.scientecheasy.com/2021/03/what-is-jvm.html/) in java program.

**Q. Why do we use methods in Java?**

-> The purpose of using methods in the Java program is to write the logic of the applications. Let’s take an example program to understand the concept of using methods in java programming.

## **Types of Methods in Java**

* 1. **Predefined methods**  
     2. **User-defined methods** (Programmer-defined methods)

**Predefined methods in Java** are those methods that are already defined in the Java API (Application Programming Interface) to use in an application.

Java Programming language contains predefined classes that are organized in different predefined packages. Within these predefined classes, there are located predefined methods.

1. print()
2. sqrt()
3. max()

## User-defined methods

**User-defined methods in Java** are those methods that are defined inside a class to perform a special task or function in an application. Such methods are called user-defined methods.

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**

Let's create a user defined method that checks the number is even or odd. First, we will define the method.

1. //user defined method
2. **public** **static** **void** findEvenOdd(**int** num)
3. {
4. //method body
5. **if**(num%2==0)
6. System.out.println(num+" is even");
7. **else**
8. System.out.println(num+" is odd");
9. }

We have defined the above method named findevenodd(). It has a parameter **num** of type int. The method does not return any value that's why we have used void. The method body contains the steps to check the number is even or odd. If the number is even, it prints the number **is even**, else prints the number **is odd**.

### **How to Call or Invoke a User-defined Method**

Once we have defined a method, it should be called. The calling of a method in a program is simple. When we call or invoke a user-defined method, the program control transfer to the called method.

1. **import** java.util.Scanner;
2. **public** **class** EvenOdd
3. {
4. **public** **static** **void** main (String args[])
5. {
6. //creating Scanner class object
7. Scanner scan=**new** Scanner(System.in);
8. System.out.print("Enter the number: ");
9. //reading value from the user
10. **int** num=scan.nextInt();
11. //method calling
12. findEvenOdd(num);
13. }

In the above code snippet, as soon as the compiler reaches at line **findEvenOdd(num),** the control transfer to the method and gives the output accordingly.

* 1. **Instance Method**
  2. **Static Method**

## **Instance Method in Java**

An instance method is used to implement behaviors of each object/instance of the class. Since the instance method represents behaviors of the objects. Therefore, instance method is linked with an object.

void m1()

{

 // This area is called an instanced area/ Non-static area.

  // logic here.

}

An instance method is also known as **non-static method**. So, without [creating an object of the class](https://www.scientecheasy.com/2020/06/how-to-create-object-in-java.html/), the methods cannot exist to perform their desired behaviors or task. It is allocated in the heap memory during the object creation.

## **Static Method in Java**

### **What is the static method?**

* A static method belongs to the class rather than the object.
* There is no need to create the object to call the static methods.
* A static method can access and change the value of the static variable.

When you declare any method with a static modifier, it is called [**static method in java**](https://www.scientecheasy.com/2020/06/java-static-method.html/). A static method is linked with class.

Therefore, it is also known as a class method. It is used to implement the behavior of the class itself. Static methods load into the memory during class loading and before object creation.

static void m2()

{

// This area is called a static area.

// logic here.

}

### **Q26. Differentiate between static and non-static methods in Java.**

|  |  |
| --- | --- |
| **Static Method** | **Non-Static Method** |
| 1. *The static* keyword must be used before the method name | 1. No need to use the static keyword before the method name |
| 2. It is called using the class (className.methodName) | 2. It is can be called like any general method |
| 3. They can’t access any non-static instance variables or methods | 3. It can access any static method and any static variable without creating an instance of the class |

### **Can we override the static methods?**

No, we can't override static methods.

**What is method overloading in java ?**

A class having two or more methods with same name but with different arguments then we say that those methods are overloaded. Static polymorphism is achieved in java using method overloading.

Method overloading is used when we want the methods to perform similar tasks but with different inputs or values. When an overloaded method is invoked java first checks the method name, and the number of arguments ,type of arguments; based on this compiler executes this method.

Compiler decides which method to call at compile time. By using overloading static polymorphism or static binding can be achieved in java.

Note : Return type is not part of method signature. we may have methods with different return types but return type alone is not sufficient to call a method in java.

**What is method overriding in java ?**

If we have methods with same signature (same name, same signature, same return type) in super class and subclass then we say subclass method is overridden by superclass.

When to use overriding in java

If we want same method with different behaviour in superclass and subclass then we go for overriding. When we call overridden method with subclass reference subclass method is called hiding the superclass method.

**Difference between overriding and overloading in java?**

| S.NO | Method Overloading | Method Overriding |
| --- | --- | --- |
| 1. | Method overloading is a compile time polymorphism. | Method overriding is a run time polymorphism. |
| 2. | It help to rise the readability of the program. | While it is used to grant the specific implementation of the method which is already provided by its parent class or super class. |
| 3. | It is occur within the class. | While it is performed in two classes with inheritance relationship. |
| 4. | Method overloading may or may not require inheritance. | While method overriding always needs inheritance. |
| 5. | In this, methods must have same name and different signature. | While in this, methods must have same name and same signature. |
| 6. | In method overloading, return type can or can not be be same, but we must have to change the parameter. | While in this, return type must be same or co-variant. |

**Can we overload the main in Java?**

Yes you can [overload](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html) the main method in Java, nothing wrong with this but Java will only call your specific main method, i.e. main method with the following signature:

public static void main(String[] args) or public static void main(String args...) which is the main method as [variable argument method](http://javarevisited.blogspot.sg/2011/09/variable-argument-in-java5-varargs.html) and only supported post-Java 5 world.

**2. Can we override main in Java?**

No, you can not override the main method in Java, Why? because main is a [static method](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) and in Java static method is bonded during compile time and you can not [override](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) the static method in Java. If you declare a method with the same name and signature it's called method hiding.

**Difference between method overloading and method overriding in java ?**

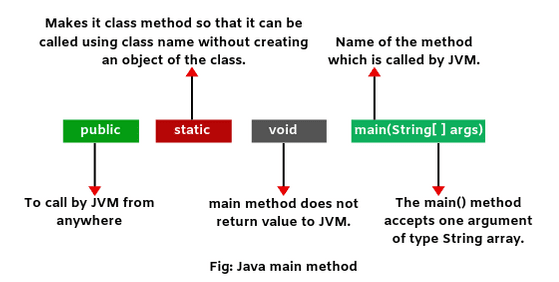
|  |  |
| --- | --- |
| **Method Overloading** | **Method Overriding** |
| 1) Method Overloading occurs with in the same class | Method Overriding occurs between two classes superclass and subclass |
| 2) Since it involves with only one class inheritance is not involved. | Since method overriding occurs between superclass and subclass inheritance is involved. |
| 3)In overloading return type need not be the same | 3) In overriding return type must be same. |
| 4) Parameters must be different when we do overloading | 4) Parameters must be same. |
| 5) Static polymorphism can be acheived using method overloading | 5) Dynamic polymorphism can be acheived using method overriding. |
| 6) In overloading one method can’t hide the another | 6) In overriding subclass method hides that of the superclass method. |

**3. Can we make the main final in Java?**

Of course, you can make the main method final in Java. JVM has no issue with that. Unlike any [final method](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html), you can not override main in Java.

# Main Method in Java | public static void main(String[] args)

A **main() method in java** is an entry point to start the execution of a program. Every Java application has at least one [class](https://www.scientecheasy.com/2020/07/java-classes-objects.html/) and at least one main method.



### **Q2. Explain public static void main(String args[]) in Java.**

**main() in Java is the entry point for any Java program. It is always written as public static void main(String[] args).**

* **public**: Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.
* **static**: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as **main**() is called by the JVM before any objects are made and only static methods can be directly invoked via the class.
* **void**: It is the return type of the method. Void defines the method which will not return any value.
* **main**: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.
* **String args[]**: It is the parameter passed to the main method.

## **Why public static void main in Java?**

Basically, the public static void main(String [ ] args) acts as an entry point to start the execution of Java application program. That’s why we use/write public static void main in java program.

## **Why do we need to declare main method as static in Java?**

The main method is declared as static. It is called by JVM when we run a class. The JVM does not know how to create an object of a class. It needs a standard way to start the execution of a program.

Therefore, the main method is declared as static so that the JVM can call it using the class name which is passed on the command line.

## **Why main method is declared as static in Java?**

If we do not declare the main method as static, it will be considered as an instance method. The code will be compiled successfully without generating any error message. But at runtime, the code will generate an [exception](https://www.scientecheasy.com/2020/08/exception-handling-in-java.html/) named: “NoSuchmethodError: main”.

## **How to call main method in Java?**

The main method is called by JVM when we run a class.

## **Can we have more than one main() method in class?**

Yes, a class can have any number of main() methods but the execution always starts from public static void main(String[ ] args) only.

## **Can we execute a program without main() method in Java?**

Yes, we can execute a program without main() method in Java in the previous version of JDK. One of the ways is a static block. But from onwards JDK 1.7 and above, is not possible.

## **Can we overload main method in java?**

Yes, we can overload the main() method but we cannot override it. We can declare any number of main() method in a class, but the method signature must be different. Let’s make a program where we will overload the main method.

# Constructor in Java

A **constructor in java** is a block of code, similar to a [method](https://www.scientecheasy.com/2020/06/java-methods.html/) that is used to initialize the state of an object in a class through a new operator.

In other words, a constructor is a special type of method that is used to initialize [instance variables](https://www.scientecheasy.com/2020/05/variables-in-java.html/) in a class.

The sole purpose of the constructor is to initialize the data fields of objects in the class.

Java constructor can perform any action but specially designed to perform initializing actions, such as initializing the instance variables.

A constructor within a class allows constructing the object of the class at runtime. It is invoked when an instance of a class is created using the new operator.

## **Characteristics of Constructor**

There are the following characteristics of constructor in java. They are as follows:

1. Constructor’s name must be the same as the name of the class in which it is declared and defined.

2. The constructor should not have any return type even void also because if there is a [return type](https://www.scientecheasy.com/2020/06/return-type-in-java.html/) then [JVM](https://www.scientecheasy.com/2021/03/what-is-jvm.html/) would consider as a method, not a constructor.

[Compiler](https://www.scientecheasy.com/2021/03/java-compiler.html/) and JVM differentiate constructor and method definitions on the basis of the return type.

Suppose you define the method and constructor with the same name as that of the class name then JVM would differentiate between them by using return type.

3. Whenever we create an object/instance of a class, the constructor will be automatically called by the JVM (Java Virtual Machine).

If we don’t define any constructor inside the class, Java compiler automatically creates a default constructor at compile-time and assigns default values for all variables declared in the class.

The default values for variables are as follows:

a. Numeric variables are set to 0.  
b. Strings are set to null.  
c. Boolean variables are set to false.

4. Java constructor may or may not contain parameters. Parameters are local variables to receive value (data) from outside into a constructor.

5. A constructor is automatically called and executed by JVM at the time of object creation. JVM first allocates the memory for variables (objects) and then executes the constructor to initialize instance variables.

6. It is called and executed only once per object. This means that when an object of a class is created, constructor is called. When we create second object then the constructor is again called during the second time.

## **Use of Constructor in Java**

The use of constructor in Java is as follows:

* The constructor is used in java programming to assign the default value of instance variables.
* Constructor is used to initializing objects of a class and allocate appropriate memory to objects. That is, it is used to initialize the instance variables of a class with a different set of values but it is not necessary to initialize.
* If you need to execute some code at the time of object creation, you can write them inside the constructor. Generally, it is used for the initialization of instance variables.
* [Use of private constructor](https://www.scientecheasy.com/2021/02/java-private-constructor.html/)

## **How to call Constructor in Java**

There are the following ways to call a constructor in java.

1. A a = new A(); // Here, A is name of class.  
2. new A(); // It is calling A() constructor.  
3. super();  
4. this();  
5. class.forName(“com.scientecheasy.A”).newInstance();

When we create an object of class by using new keyword, a constructor is automatically called by JVM. After creating the object of the class, we cannot call the constructor again.

## **Types of Constructors in Java**

Basically, there are two types of constructors in java. They are as:

1. **Default Constructor** (No-argument constructor)
2. **Parameterized Constructor** (Argument constructor)

## **Default Constructor**

A constructor that has no parameter is known as default constructor in Java. When a class does not declare a constructor, Java compiler automatically creates a constructor for that class. This constructor is called default constructor.

We cannot pass any argument to default constructor. That’s why it is known as a no-argument constructor in java. It does not do anything but it allows to create instance of class.

### **What is the purpose of a default constructor?**

The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

1. **class** Student3{
2. **int** id;
3. String name;
5. **void** display()
6. {System.out.println(id+" "+name);}
8. **public** **static** **void** main(String args[]){
9. Student3 s1=**new** Student3();
10. Student3 s2=**new** Student3();
11. s1.display();
12. s2.display();
13. }
14. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Student3)

Output:

0 null

0 null

### **Does constructor return any value?**

**Ans:** yes, The constructor implicitly returns the current instance of the class (You can't use an explicit return type with the constructor).

### **Can you make a constructor final?**

No, the constructor can't be final.

### **Can we make constructors static?**

As we know that the static context (method, block, or variable) belongs to the class, not the object. Since Constructors are invoked only when the object is created, there is no sense to make the constructors static. However, if you try to do so, the compiler will show the compiler error.

## **Parameterized Constructor**

A constructor that takes 1 or more parameters is known as parameterized constructor in java. The parameterized constructor is used to provide different values to distinct objects.

To call the parameterized constructor, we pass arguments while creating the object. Therefore, parameterized constructor is also called argument constructor.

The [argument](https://www.scientecheasy.com/2020/06/java-arguments-parameter.html/) can be of any type like integer, array, character, or object. It can take any number of arguments. Java does not provide a parameterized constructor by default.

**Note:** We cannot define two constructors with the same number of parameters and the same types.

## **Q. Difference between Constructor and Method in Java**

**Constructor vs Method**

|  |  |  |
| --- | --- | --- |
| **SN** | **Constructor** | **Method** |
| 1. | Constructor is a special type of method that is used to initialize the state of an object. | Method is used to expose the behaviour of an object. |
| 2. | It has no return type even void also. | It has both void and return type. |
| 3. | If we don’t provide any constructor in the class, Java Compiler provides a default constructor for that class. | Method is not provided by the compiler in any case. |
| 4. | Constructor name must be the same as name of the class. | Method name may or may not be the same name as the class name. |
| 5. | The purpose of a constructor is to create an object of a class. | The purpose of a method is to execute the functionality of the application. |
| 6. | They are not inherited by subclasses. | They are inherited by subclasses. |

### **Q27. What is constructor chaining in Java?**

In Java, constructor chaining is the process of calling one constructor from another with respect to the current object. Constructor chaining is possible only through legacy where a subclass constructor is responsible for invoking the superclass’ constructor first. There could be any number of classes in the constructor chain. Constructor chaining can be achieved in two ways:

1. Within the same class using this()
2. From base class using super()

# Constructor Overloading in Java

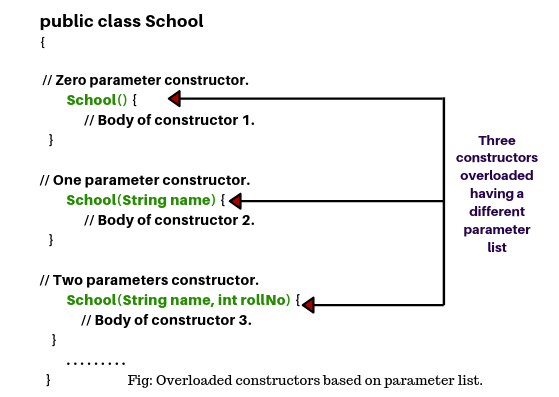
**Constructor overloading in Java** means to define multiple [constructors](https://www.scientecheasy.com/2020/06/constructor-in-java.html/) of a class but each one must have a different signature.

Constructor overloading is a technique in Java in which a class can have more than one constructor that differ in the parameter lists.

other words, defining two or more constructors with the same name but with different signatures is called constructor overloading in java. It is used to perform different tasks.

If two constructors of a class have the same signature, it represents ambiguity. In this case, Java compiler will generate an error message because Java compiler will unable to differentiate which form to execute.

Hence, overloaded constructors must have different signatures. Java compiler decides which constructor has to be called depending on the number of arguments passing with objects.



## **Use of Constructor overloading in Java**

Overloaded constructors are very common to use in Java programming based on needs because they provide many ways to create an object of a particular class. Constructor overloading allows initializing objects with different types of data.

For example, consider an object having three [instance variables](https://www.scientecheasy.com/2020/05/variables-in-java.html/) in a class. If we need to assign a specific value to the second instance variable and the default values to be assigned to the remaining variables.

### **Can we overload the constructors?**

Yes, the constructors can be overloaded by changing the number of arguments accepted by the constructor or by changing the data type of the parameters. Consider the following example.

1. **class** Test
2. {
3. **int** i;
4. **public** Test(**int** k)
5. {
6. i=k;
7. }
8. **public** Test(**int** k, **int** m)
9. {
10. System.out.println("Hi I am assigning the value max(k, m) to i");
11. **if**(k>m)
12. {
13. i=k;
14. }
15. **else**
16. {
17. i=m;
18. }
19. }
20. }
21. **public** **class** Main
22. {
23. **public** **static** **void** main (String args[])
24. {
25. Test test1 = **new** Test(10);
26. Test test2 = **new** Test(12, 15);
27. System.out.println(test1.i);
28. System.out.println(test2.i);
29. }
30. }

## **this Reference in Java**

All instance methods or constructors accept an implicit argument called “[this](https://www.scientecheasy.com/2020/07/this-keyword-in-java.html/)” which refers to the current object. A current object is an object on which the method is called and ‘this’ reference can be used inside any method or constructor to refer to the current object.

Within the body of the method or constructor, ‘this’ reference can be used like any other object reference to access instance variables, instance methods, and constructors.

### **What are the main uses of this keyword?**

There are the following uses of **this** keyword.

* **this** can be used to refer to the current class instance variable.
* **this** can be used to invoke current class method (implicitly)
* **this()** can be used to invoke the current class constructor.
* **this** can be passed as an argument in the method call.
* **this** can be passed as an argument in the constructor call.
* **this** can be used to return the current class instance from the method.

## **Advantage of Constructor Overloading in Java**

The advantages of using constructor overloading in java programming are as follows:

* Java constructor overloading helps to achieve static polymorphism.
* The main advantage of constructor overloading is to allow an instance of a class to be initialized in various ways.
* It allows to define of the multiple constructors of a class with different signatures.
* It helps to perform different tasks for different purposes.

# Copy Constructor

A constructor which is used to copy the data of one object to another object of the same class type is called copy [constructor in Java](https://www.scientecheasy.com/2020/06/constructor-in-java.html/).

It is called when a single object reference of the same class is passed as an argumen

## **Uses of Copy constructor in Java**

There are mainly two uses of copy constructor in java program that are as follows:

* A copy constructor is used to declare and initialize an object from another object.
* It can be used to copy data from one memory location to another memory location in Java.

Copy constructor in Java provides an easy and attractive mechanism to make a copy (cloning) of one object from another object of the same class type.

# Private Constructor

When a constructor is declared with a private access modifier in a class, it is called **private**[**constructor in java**](https://www.scientecheasy.com/2020/06/constructor-in-java.html/). It is visible within the same class where it has been declared.

Java private constructor is useful when we want to prevent users from creating an object of class. Only within the class, we can make a new object of class with private constructor.

## **Q. What is Use of Private Constructor in Java**

There are mainly two important uses of private constructor in java. They are as follows:

1. Private constructor is used when we want to prevent other classes from [creating an object of a class](https://www.scientecheasy.com/2020/06/how-to-create-object-in-java.html/).
2. It can be used in single tone classes where the object of the class cannot be created outside the class.

**Q.  Can a constructor be private in Java?**

**Yes.** **Class can have private constructor**. Even abstract class can have private constructor. By making constructor private, we prevent the class from being instantiated as well as subclassing of that class.

## **Q.What happens when we make Constructor private in Java?**

1. A class cannot be extended when a constructor is declared as private

2. We cannot create an instance of class when we declare constructor private. If we try, compile-time error will get.

**Q.How do you access a class with a private constructor?**

Another way of accessing a private constructor is by **creating a public static method within this class** and have its return type as its object.

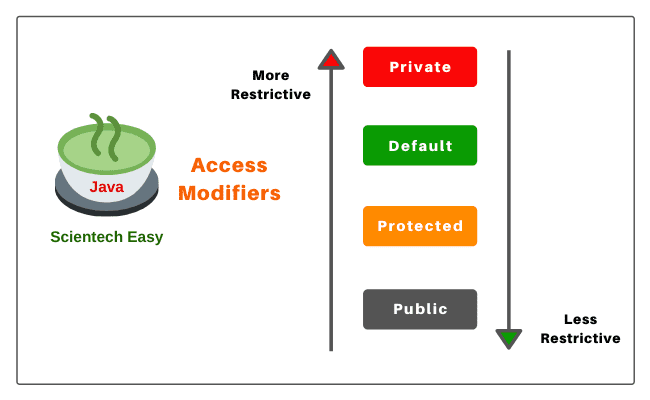
# Access Modifiers in Java

A **modifier in java** is a keyword that we add to those definitions that we need to change their meaning.

In other words, access modifiers are those modifiers that are used to restrict the visibility (accessibility) of classes, fields, methods, or constructors.

Twelve modifiers in java can be divided into two categories:

* **Access modifiers**
* **Non-access modifiers**



## **Private Access Modifier**

1. Private access modifier in java can apply to a variable, method, constructor, inner class but not the outer class that is class itself.

2. The instance variable can be private but a local variable cannot be private.

3. Private members (field, method, or constructor) of a class cannot be accessed from outside the class. They are accessible only within the class.

4. Private members of a superclass cannot be inherited to the subclass. Therefore, they are not accessible in subclasses.

5. If we make any constructor as private, we cannot create an object of that class from another class and also cannot create the subclass of that class.

6. A class cannot be private except for inner classes. Inner classes are members of the outer class. So, members of the class can be private.

7. If we declare a method as private, it behaves as a method declared as final. We cannot call the private method from outside the class.

### **Role of Private Constructor**

If we declare any constructor of a class as private then we cannot create an object of that class from outside the class. In other words, we cannot create the subclass of that class.

## **Default Access Modifier**

1. When access modifier is not specified to members of a class or a class itself, it is called default access modifier.

2. The default can apply to the instance variable, local variable, constructor, methods, inner class, or outer class.

3. Default members of a class are visible inside of the class and everywhere within classes in the same package or folder only. Therefore, they can be accessed from outside the classes in the same package but can not be accessed outside the package.

4. Default members can be inherited to the subclass within the same package only. It cannot be inherited from outside the package.

## **Protected Access Modifier**

1. Protected access modifier can be applied to instance variables, local variables, constructors, methods, inner classes but not the outer class.

2. Protected members are accessible inside the class and everywhere within classes in the same package and outside the package but through inheritance only.

3. Protected members can be inherited to the subclass.

4. If we make constructor as protected then we can create the subclass of that class within the same package but not outside the package

## **Public Access Modifier**

1. Public access modifier can apply to instance variables, constructors, inner classes, outer class, methods but not with local variables.

2. Public members of a class can be used anywhere.

3. Public members of a class can be inherited to any subclass.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifier** | **Default** | **Private** | **Protected** | **Public** |
| *Same class* | YES | YES | YES | YES |
| *Same Package subclass* | YES | NO | YES | YES |
| *Same Package non-subclass* | YES | NO | YES | YES |
| *Different package subclass* | NO | NO | YES | YES |
| *Different package non-subclass* | NO | NO | NO | YES |

**What is the difference between access specifiers and access modifiers in java?**

In C++ we have access specifiers as public,private,protected and default and access modifiers as static, final. But there is no such divison of access specifiers and access modifiers in java. In Java we have access modifiers and non access modifiers.

Access Modifiers : public, private, protected, default Non Access Modifiers : abstract, final, stricfp.

**What access modifiers can be used for class ?**

We can use only two access modifiers for class public and default.

**public:** A class with public modifier can be visible

1. In the same class
2. In the same package subclass
3. In the same package nonsubclass
4. In the different package subclass

5) In the different package non subclass.

**default :** A class with default modifier can be accesed

1. In the same class
2. In the same package subclass
3. In the same package nonsubclass
4. Not In the different package subclass

5) not In the different package non subclass.

**46) Explain what access modifiers can be used for methods?**

We can use all access modifiers public, private,protected and default for methods. public : When a method is declared as public it can be accessed

1. In the same class
2. In the same package subclass
3. In the same package nonsubclass 9) In the different package subclass

10) In the different package non subclass.

default : When a method is declared as default, we can access that method in

1. In the same class
2. In the same package subclass
3. In the same package non subclass

We cannot access default access method in 1) Different package subclass

2) Different package non subclass.

protected : When a method is declared as protected it can be accessed

1. With in the same class
2. With in the same package subclass
3. With in the same package non subclass
4. With in different package subclass

It cannot be accessed non subclass in different package.

private : When a method is declared as private it can be accessed only in that class. It cannot be accessed in

1. Same package subclass
2. Same package non subclass 3) Different package subclass

4) Different package non subclass.

**What is final access modifier in java?**

final access modifier can be used for class, method and variables. The main advantage of final access modifier is security no one can modify our classes, variables and methods. The main disadvantage of final access modifier is we cannot implement oops concepts in java. Ex : Inheritance, polymorphism. final class : A final class cannot be extended or subclassed. We are preventing inheritance by marking a class as final. But we can still access the methods of this class by composition. Ex: String class final methods: Method overriding is one of the important features in java. But there are situations where we may not want to use this feature. Then we declared method as final which will print overriding. To allow a method from being overridden we use final access modifier for methods.

final variables : If a variable is declared as final ,it behaves like a constant . We cannot modify the value of final variable. Any attempt to modify the final variable results in compilation error. The error is as follows

*“final variable cannot be assigned.”*

# Static Method in Java

When a method is declared with the keyword ‘static’, it is called **static method in java**.

Like a [static variable](https://www.scientecheasy.com/2020/06/java-static-variable.html/), static method is also tied to the class, not to an object of class. Therefore, it can be called and executed without [creating objects of the class](https://www.scientecheasy.com/2020/06/how-to-create-object-in-java.html/).

Static method can be called or accessed directly using class name. The syntax to call a static method in Java is as follows:

className.methodName(); // Here, className is the name of a class.

For example:

  Student.add(); // Student is the name of class and add is a method.

A static method is also known as **class method in java** because it belongs to a class rather than an individual instance of a class.

## **Why Instance variable is not available to Static method?**

When we declare a static method in Java, JVM first executes the static method, and then it creates objects of the class. Since objects are not available at the time of calling the static method.

Therefore, instance variables are also not available to a static method. Due to which a static method cannot access an instance variable in the class.

## **How to call static method from another class?**

We can call a static method in Java from another class directly using the class name. Let’s understand it with help of an example program.

In this example, we will create a class Student and declare static methods name, rollNo, and std with return type.

**Program source code 6:**

package staticVariable;

public class Student

{

static String name(String n)

{

return n;

}

static int rollNo(int r)

{

return r;

}

static int std(int s)

{

return s;

}

}

## **Difference between Static method and Instance method in Java**

1. A static method is also known as class method whereas, the instance method is also known as non-static method.

2. The only static variable can be accessed inside static method whereas, static and instance variables both can be accessed inside the instance method.

3. We do not need to create an object of the class for accessing static method whereas, in the case of an instance method, we need to create an object for access.

4. Class method cannot be overridden whereas, an instance method can be overridden.

5. Memory is allocated only once at the time of class loading whereas, in the case of the instance method, memory is allocated multiple times whenever the method is calling.

|  |  |
| --- | --- |
| **static or class method** | **instance method** |
| 1)A method that is declared as static is known as the static method. | A method that is not declared as static is known as the instance method. |
| 2)We don't need to create the objects to call the static methods. | The object is required to call the instance methods. |
| 3)Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly. | Static and non-static variables both can be accessed in instance methods. |
| 4)For example: public static int cube(int n){ return n\*n\*n;} | For example: public void msg(){...}. |

## **Can we execute Static block without Main method inside class?**

It is possible to execute static block without a main method inside the class up to Java 1.5 version but Java 1.6 version onwards, the main method is mandatory to execute a static block inside the class.

During the dot class file loading, first static blocks are executed and then JVM calls main method.

## **Use of Static block**

* 1. The purpose of using a static initialization block is to write that logic inside static block that is executed during the class loading.
* 2. It is mostly used for changing default value of static variables.
* 3. It is used to initialize static variables of the class.

## **Difference between Static block and Instance block in Java**

1. Static block is also known as a static initialization block whereas instance block is also known as instance initialization block or non-static block.

2. They execute before the instance block whereas instance block executes after the static blocks.

3. Only static variables can be accessed inside the static block whereas both static and non-static variables can be accessed inside the instance block.

4. Static blocks execute when the class is loaded into the memory whereas instance blocks execute only when instance of the class is created.

5. ‘this’ keyword cannot be used in the static block whereas this keyword can be used in the instance block.

### **Q21. What is final keyword in Java?**

**final**is a special keyword in Java that is used as a non-access modifier. A final variable can be used in different contexts such as:

* **final variable**

When the final keyword is used with a variable then its value can’t be changed once assigned. In case the no value has been assigned to the final variable then using only the class constructor a value can be assigned to it.

#### **final method**

When a method is declared final then it can’t be overridden by the inheriting class.

#### **final class**

When a class is declared as final in Java, it can’t be extended by any subclass class but it can extend other class.

## **When to use Final variable in Java?**

A final variable can be used where we want to remain constant the value of a variable throughout the execution of a program.

## **Use of Final class in Java**

1. The first use is to prevent inheritance, as the final classes cannot be extended.

2. The second use is to create an immutable class like the predefined String class. We cannot make a class immutable without making it final.

3. A final class is very useful when we want high security in any application because the final class cannot be extended.

### **Q24. What is the difference between this() and super() in Java?**

In Java, super() and this(), both are special keywords that are used to call the constructor.

|  |  |
| --- | --- |
| **this()** | **super()** |
| 1. this() represents the current instance of a class | 1. super() represents the current instance of a parent/base class |
| 2. Used to call the default constructor of the same class | 2. Used to call the default constructor of the parent/base class |
| 3. Used to access methods of the current class | 3. Used to access methods of the base class |
| 4.  Used for pointing the current class instance | 4. Used for pointing the superclass instance |
| 5. Must be the first line of a block | 5. Must be the first line of a block |

# Inner Class in Java

An inner class in java is a class that is declared inside of another class without [static modifier](https://www.scientecheasy.com/2020/06/instance-block-in-java.html/). It is also commonly known as a non-static nested class in Java. It can access all members (variables and methods) of its outer class.

### **In which version Inner class was introduced in Java?**

Inner class was introduced in the Java 1.1 version without any changes in the JVM used to handle the class files. It is completely implemented with the help of Java compiler.

## **Features of Inner class**

There are several important features of an inner class that is as follows:

1. An inner class can have the same name as the outer class.

2. The scope of inner class is bounded by the scope of its outer class.

3. Without existing an outer class object or instance, there will be no chance of an existing inner class object.

## **Realtime Example of Inner class**

1. Suppose there is a university. University contains several departments such as electronics, computer science, electrical, mechanical, etc.

Assume that tomorrow, the government decided to close this university due to its illegal activity happening in the university. If the university is closed, all departments of this university will be closed automatically.

That’s why we can say that departments are tied to its university. i.e. their functionalities are tied with its university. So, without an existing university object, there is no chance of existing department objects.

Since the department is always a part of the university class. Hence, we must declare the department class inside the university class.

class University {  // University is outer class.

  . . . . . . .

 class Department { // Department is inner class.

    . . . . . . . .

  }

}

## **Use of Inner class in Java**

1. The inner class is a valuable feature because it allows us to group classes and interfaces in one place that logically belongs together.

2. It helps to control the visibility of one within the other so that it can be more readable and maintainable.

3. The best use of java inner class is when its functionality is tied to its outer class. That is,  without an existing outer class object, there is no chance of existing inner class object. Understand the real-time examples.

4. . Java inner class is useful to provide security for the important code. For example, if we declare inner class as private, it is not available to other classes. This means that an object to inner class cannot be created in any other classes.

## **Difference between Nested class and Inner class in Java**

Nested class is a class that is defined inside another class. There are two types of nested class: static and non-static. Non-static nested class is the inner class.

# OOPs Concepts in Java | Realtime Example

## **Q.What Is Object-Oriented Programming in Java?**

1. oop pillar with real life example

**OOPs Concepts in Java |** Object-oriented programming (OOP) in Java is a programming methodology or paradigm (model)  to design a computer program using classes and objects.

(Procedural programming means writing code without objects).

Whereas, object-oriented programming means writing code with objects. It contains data in the form of fields or variables (often known as attributes or properties in java) and functionality code in the form of methods (often known as behavior in java).

When we write a program using these features, it is called **Object-Oriented Programming System (OOPs)**. The main goal of the OOPs concept in java programming is that everything you want to do, do through objects.

## **Features of OOPs in Java**

1. Higher priority is focused on Data rather than functions.

2. Programs are divided into the number of entities known as objects.

3. Objects communicate with each other through functions (methods).

4. Methods that operate on Data of an object are closely bound together in the data structure.

5. An object is a group of data and methods.

6. Data is hidden in the OOP and cannot be accessed by external methods. Hence, it is safe from accidental modification.

7. Data cannot move freely out of the object.

8. New data and methods can be easily added whenever needs.

9. A bottom-up approach is adopted in programming design.

## **Basic OOPs Concepts in Java with Realtime Example**

1. **Object:** A real-world entity that has state and behavior is called object in java. Here, state represents properties and behavior represents actions and functionality. For example, a person, chair, pen, table, keyboard, bike, etc.

->Let’s take a realtime example to understand the characteristics of an object. A person has three characteristics: Identity (name), State (properties), and behavior (actions or functionality).

->The state/properties of a person are black hair, black eyes, white skin, 6 feet tall, etc. The actions or behavior of the person may be “eat, sleep, walk, play, and study”.

1. **Class:** A class is basically user-defined data types that act as a template for creating objects of the identical type. It represents the common properties and actions (functions) of an object.

->For example, bus and car are objects of vehicle class.

->Object takes space in the memory but class does not take any space in the memory. Class does not exist physically but an object exists physically.

### What are the main features of OOPs?

OOPs or Object Oriented Programming mainly comprises of the below four features, and make sure you don't miss any of these:

* Inheritance
* Encapsulation
* Polymorphism
* Data Abstraction

1. **Encapsulation:** The process of binding data (variables) and corresponding functions (methods) together into a single unit (called class) is called [encapsulation in Java](https://www.scientecheasy.com/2020/07/encapsulation-in-java.html/). It is one of the striking features to achieve data security in an object-oriented program.

->Through encapsulation, data is hidden and protected from access by outside non-member methods of a class. Only member methods defined in a class will have access to the data.

->A Java class is an example of encapsulation because class binds variables and methods together.

**Realtime Example 2:**  
When you log into your email accounts such as Gmail, Yahoo Mail, or Rediff mail, there is a lot of internal processes taking place in the backend and you have no control over it.

When you enter the password for logging, they are retrieved in an encrypted form and verified, and then you are given access to your account.

You do not have control over it that how the password has been verified. Thus, it keeps our account safe from being misused.

## **Q.How to achieve or implement Encapsulation in Java?**

1. Declaring the instance variable of the class as private. so that it cannot be accessed directly by anyone from outside the class.

2. Provide the public setter and getter methods in the class to set/modify the values of the variable/fields.

## **Advantage of Encapsulation in Java**

1. The encapsulated code is more flexible and easy to change with new requirements.  
2. It prevents the other classes to access the private fields.  
3. Encapsulation allows modifying implemented code without breaking other code that has implemented the code.  
4. It keeps the data and codes safe from external inheritance. Thus, Encapsulation helps to achieve security.  
5. It improves the maintainability of the application.  
6. If you don’t define the setter method in the class then the fields can be made read-only.  
7. If you don’t define the getter method in the class then the fields can be made write-only.

## Disadvantage of Encapsulation in Java

The main disadvantage of encapsulation in Java is it increases the length of the code and slows shutdown execution.

### What would happen if we do not use Encapsulation?

If we don’t use encapsulation in a program, fields will not be private and could be accessed by anyone from outside the class.

**What is data hiding in Java?**

Ans: An outside person cannot access our internal data directly or our internal data should not go out directly. This oops feature is called data hiding in Java. After validation or authentication, the outside person can access our internal data.

**How to achieve Data hiding programmatically?**

Ans: By declaring data members (variables) as private, we can achieve or implement data hiding. If the variables are declared as private in the class, nobody can access them from outside the class.

The biggest advantage of data hiding is we can achieve security.

### **Simple Example of Encapsulation in Java**

1. //A Java class which is a fully encapsulated class.
2. //It has a private data member and getter and setter methods.
3. **package** com.javatpoint;
4. **public** **class** Student{
5. //private data member
6. **private** String name;
7. //getter method for name
8. **public** String getName(){
9. **return** name;
10. }
11. //setter method for name
12. **public** **void** setName(String name){
13. **this**.name=name
14. }
15. }
16. /A Java class to test the encapsulated class.
17. **package** com.javatpoint;
18. **class** Test{
19. **public** **static** **void** main(String[] args){
20. //creating instance of the encapsulated class
21. Student s=**new** Student();
22. //setting value in the name member
23. s.setName("vijay");
24. //getting value of the name member
25. System.out.println(s.getName());
26. }
27. }

Compile By: javac -d . Test.java

Run By: java com.javatpoint.Test

o/p ->vijay

### **What is an association?**

Association is a relationship where all object have their own lifecycle and there is no owner.

Let’s take the example of Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. These relationships can be one to one, one to many, many to one and many to many.

### **Q13. What do you mean by aggregation?**

An aggregation is a specialized form of Association where all object has their own lifecycle but there is ownership and child object cannot belong to another parent object.

Let us take an example of Department and teacher. A single teacher cannot belong to multiple departments, but if we delete the department, teacher object will not destroy.

### **Q14. What is composition in Java?**

Composition is again a specialized form of Aggregation and we can call this as a “death” relationship. It is a strong type of Aggregation. Child object does not have their lifecycle and if parent object deletes all child object will also be deleted. Let’s take again an example of a relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belongs to two different houses if we delete the house room will automatically delete.

1. **Abstraction:**

[Abstraction](https://www.scientecheasy.com/2020/05/java-abstraction.html/) is a technique by which we can hide the unnecessary data that is not needed from a user and expose only that data is of interest to the user. It hides all unwanted data so that users can work only with the required data.

->Abstraction is one of the fundamental principles of object-oriented programming that permits the user to use an object without knowing its internal details. It helps to reduce complexity by not including background details.

->A realtime example of abstraction is “sending SMS”. When you need to send SMS  from your mobile, you only type the text and send the message. But you don’t know the internal processing of the message delivery.

## 

## **Realtime Examples**

1. Let’s first take ATM machine as a real-time example. We all use an ATM machine for cash withdrawal, money transfer, retrieve min-statement, etc in our daily life.

But we don’t know internally what things are happening inside ATM machine when you insert an ATM card for performing any kind of operation.

## **How to achieve Abstraction in Java?**

There are two ways to achieve or implement abstraction in java program. They are as follows:

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

## **Abstract Class in Java**

An **abstract class in Java** is a class, which is declared with an abstract keyword. It is just like a normal class but has two differences.

1. We cannot create an object of this class. Only objects of its non-abstract (or concrete) sub-classes can be created.

2. It can have zero or more abstract methods which are not allowed in a non-abstract class (concrete class). Classloader class is a good example of an abstract class that does not have any abstract methods.

## **When to use Abstract class in Java?**

An abstract class can be used when we need to share the same method to all non-abstract subclasses with their own specific implementations.

Moreover, the common member of the abstract class can also be shared by the subclasses. Thus, abstract class is useful to make the program more flexible and understandable.

## **Abstract Method in Java**

A method that is declared with abstract modifier in an abstract class and has no implementation (means no body) is called **abstract method in java**. It does not contain any body.

1. **abstract** **class** Shape{
2. **abstract** **void** draw();
3. }
4. //In real scenario, implementation is provided by others i.e. unknown by end user
5. **class** Rectangle **extends** Shape{
6. **void** draw()
7. {
8. System.out.println("drawing rectangle");}
9. }
10. **class** Circle1 **extends** Shape{
11. **void** draw()
12. {
13. System.out.println("drawing circle");
14. }
15. }
16. //In real scenario, method is called by programmer or user
17. **class** TestAbstraction1{
18. **public** **static** **void** main(String args[]){
19. Shape s=**new** Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
20. s.draw();
21. }
22. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestAbstraction1)

drawing circle

### **Example of Abstract class that has an abstract method**

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

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. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Honda4)

running safely

## **When to use Abstract method in Java?**

1. An abstract method can be used when the same method has to perform different tasks depending on the object calling it.

2. A method can be used as abstract when you need to be overridden in its non-abstract subclasses.

# Interface in Java

An **interface in Java** is syntactically similar to a class but can have only abstract methods declaration and constants as members.

In other words, an interface is a collection of abstract methods and constants (i.e. static and final fields). It is used to achieve complete abstraction.

## **Why do we use Interface?**

* 1. In industry, architect-level people create interfaces, and then it is given to developers for writing classes by implementing interfaces provided.
* 2. Using interfaces is the best way to expose our project’s API to some other projects. In other words, we can provide interface methods to the third-party vendors for their implementation.
* For example, HDFC bank can expose methods or interfaces to various shopping carts.
* 3. Programmers use interface to customize features of software differently for different objects.
* 4. It is used to achieve full abstraction in java.
* 5. By using interfaces, we can achieve the functionality of multiple [inheritance](https://www.scientecheasy.com/2020/07/inheritance-in-java.html/).

## **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?**

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

### **Syntax:**

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

## **Java Interface Example: Drawable**

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.

*File: TestInterface1.java*

1. //Interface declaration: by first user
2. **interface** Drawable{
3. **void** draw();
4. }
5. //Implementation: by second user
6. **class** Rectangle **implements** Drawable{
7. **public** **void** draw()
8. {
9. System.out.println("drawing rectangle");}
10. }
11. **class** Circle **implements** Drawable{
12. **public** **void** draw()
13. {
14. System.out.println("drawing circle");}
15. }
16. //Using interface: by third user
17. **class** TestInterface1{
18. **public** **static** **void** main(String args[]){
19. Drawable d=**new** Circle();//In real scenario, object is provided by method e.g. getDrawable()
20. d.draw();
21. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestInterface1)

Output: drawing circle

### **In Java, Multiple Inheritance is not supported through Class but it is possible by Interface, why?I**

### In multiple inheritance, subclasses are derived from multiple superclasses.

If two superclasses have the same method name then which method is inherited into subclass is the main confusion in multiple inheritance.

That’s why Java does not support multiple inheritance in case of class.

But, it is supported through an interface because there is no confusion. This is because its implementation is provided by the implementation class.

### **Q5. What is the difference between abstract classes and interfaces?**

|  |  |
| --- | --- |
| **Abstract Class** | **Interfaces** |
| An abstract class can provide complete, default code and/or just the details that have to be overridden | An interface cannot provide any code at all, just the signature |
| In the case of an abstract class, a class may extend only one abstract class | A Class may implement several interfaces |
| An abstract class can have non-abstract methods | All methods of an Interface are abstract |
| An abstract class can have instance variables | An Interface cannot have instance variables |
| An abstract class can have any visibility: public, private, protected | An Interface visibility must be public (or) none |
| If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly | If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method |
| An abstract class can contain constructors | An Interface cannot contain constructors |
| Abstract classes are fast | Interfaces are slow as it requires extra indirection to find the corresponding method in the actual class |

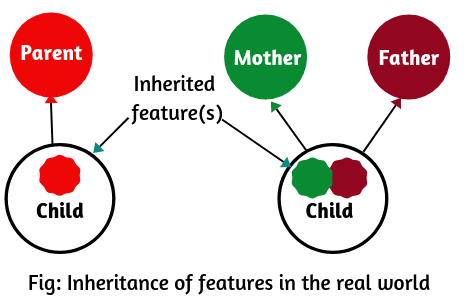
1. **Inheritance:**

Inheritance is one of the main pillars of the OOP concept. It provides a mechanism for the users to reuse the existing code within the new applications.

The technique of constructing a new class by using an existing class functionality is called [inheritance](https://www.scientecheasy.com/2020/07/inheritance-in-java.html/).

Let’s take a realtime example

1. In the real world, a child inherits the features of its parents such as beauty of mother and intelligence of father as shown in the below figure.

[](https://www.scientecheasy.com/2020/07/inheritance-in-java.html/)

Inheritance represents the IS-A relationship, also known as a parent-child relationship.

1. **What is inheritance ?**

Inheritance is one of the important feature of object oriented language. Inheriting is the process of acquiring features of others. For example a child acquires the features of their parents.

In java inheritance is the process of inheriting member of existing classes by extending their functionality. The original class is called base class, parent class or super class. The new class derived from parent is called child class, sub class, and derived class.

We use extends keyword in java to extend a class in java. All java classes extend java.lang.Object since object class is the super class for all classes in java.

When we create a new class by using inheritance ‘is-a’ relationship is formed.

## **What is Is-A relationship in Java?**

Is-A relationship in java represents Inheritance. It is implemented in Java through keywords extends (for class inheritance) and implements (for interface implementation).

All the classes extends java.lang.object by default. This is a very good example of Is-A relationship in java. This means an object is the root class of all the classes in Java. Therefore, by default, every class is the subclass of java.lang.object.

1. **What is ‘IS-A ‘ relationship in java?**

‘is a’ relationship is also known as inheritance. We can implement ‘is a’ relationship or inheritance in java using extends keyword. The advantage or inheritance or is a relationship is reusability of code instead of duplicating the code.

Ex : Motor cycle is a vehicle

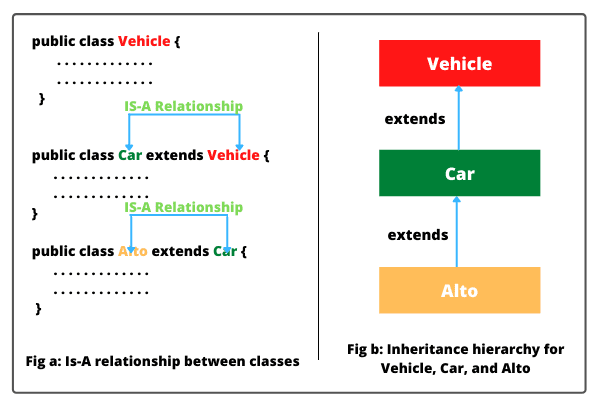
Car is a vehicle Both car and motorcycle extends vehicle.

1. **What is ‘HAS A’’ relationship in java?**

‘Has a ‘ relationship is also known as “composition or Aggregation”. As in inheritance we have ‘extends’ keyword we don’t have any keyword to implement ‘Has a’ relationship in java. The main advantage of ‘Has-A‘ relationship in java code reusability.

1. **Difference between ‘IS-A’ and ‘HAS-A’ relationship in java?**

|  |  |
| --- | --- |
| IS-A relationship | HAS- A RELATIONSHIP |
| Is a relationship also known as inheritance | Has a relationship also known as composition or aggregation. |
| For IS-A relationship we uses extends keyword | For Has a relationship we use new keyword |
| Ex : Car is a vehicle. | Ex : Car has an engine. We cannot say Car is an engine |
| The main advantage of inheritance is reusability of code | The main advantage of has a relationship is reusability of code. |



## **Why do we need/use Inheritance in Java?**

* We can reuse the code from the base class.
* Using inheritance, we can increase features of class or method by overriding.
* Inheritance is used to use the existing features of class.
* It is used to achieve runtime polymorphism i.e method overriding.
* Using inheritance, we can organize the information in a hierarchal form.

## **How is Inheritance implemented/achieved in Java?**

Inheritance in Java can be implemented or achieved by using two keywords:

1. **Extends:** extends is a keyword that is used for developing the inheritance between two classes and two interfaces. Note that a class always extends another class. An interface always extends another interface and can extend more than one interface.

2. **implements:** implements keyword is used for developing the inheritance between a class and interface. A class always implements the interface.

## **Advantage of Inheritance in Java**

* One of the main advantages is that you can minimize the length of duplicate code in an application by putting the common code in the superclass and sharing amongst several subclasses.
* Due to reducing the length of code, redundancy of the application is also reduced.
* Inheritance can also make application code more flexible to change because a class that inherits from the superclass, can be used interchangeably.

## **Important Rules of Java Inheritance**

1. We cannot assign a superclass to the subclass.  
2. We cannot extend the [final](https://www.scientecheasy.com/2020/07/final-keyword-in-java.html/) class.  
3. A class cannot extend itself.  
4. One class can extend only a single class.

5. We cannot extend a class having a private constructor but if we have [private constructor](https://www.scientecheasy.com/2021/02/java-private-constructor.html/) as well as public constructor then we can extend superclass to subclass. In this case, the only public constructor will work.

6. If we assign subclass reference to superclass reference, it is called dynamic method dispatch in java.

**Types of inheritance in java**





## **Single Inheritance Example**

When a class inherits another class, it is known as a single inheritance.

In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** TestInheritance{
8. **public** **static** **void** main(String args[]){
9. Dog d=**new** Dog();
10. d.bark();
11. d.eat();
12. }}

Output:

barking...

eating...

## **Multilevel Inheritance Example**

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** BabyDog **extends** Dog{
8. **void** weep(){System.out.println("weeping...");}
9. }
10. **class** TestInheritance2{
11. **public** **static** **void** main(String args[]){
12. BabyDog d=**new** BabyDog();
13. d.weep();
14. d.bark();
15. d.eat();
16. }}

Output:

weeping...

barking...

eating...

## **Hierarchical Inheritance Example**

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

1. **class** Animal{
2. **void** eat()
3. {
4. System.out.println("eating...");
5. }
6. }
7. **class** Dog **extends** Animal{
8. **void** bark()
9. {
10. System.out.println("barking...");
11. }
12. }
13. **class** Cat **extends** Animal{
14. **void** meow()
15. {
16. System.out.println("meowing...");
17. }
18. }
19. **class** TestInheritance3{
20. **public** **static** **void** main(String args[]){
21. Cat c=**new** Cat();
22. c.meow();
23. c.eat();
24. //c.bark();//C.T.Error
25. }}

Output:

meowing...

eating...

## **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.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

# Difference between super and this keyword in Java

|  |  |  |
| --- | --- | --- |
| **SN** | **‘this’ keyword** | **‘super’ keyword** |
| 1. | “this” is a reference variable that contains current class objects. | “super” is a reference variable that contains immediate super class objects. |
| 2. | Any member of the current class object from within an instance method or a constructor can be referred by using this keyword. | If the method overrides one of its super class’s method, the overridden method can be called through the use of super keyword. |
| 3. | ‘this’ keyword is used to call another constructor from within a constructor in the same class. | ‘super’ keyword is used to call the super class’s constructor from within a constructor of the subclass. |
| 4. | JVM never put automatically this() keyword like super() in Java. | By default JVM automatically put the super() keyword at first line inside the constructor. |

1. **Polymorphism:**

[Polymorphism](https://www.scientecheasy.com/2020/07/polymorphism-in-java.html/) is another salient feature of OOP concepts in which method can take more than one form based on the type of parameters, order of parameters, and number of parameters.

**->**It is a concept by which we can perform a single task in different ways. That is, when a single entity behaves differently in different cases, it is called polymorphism.

**->**Using polymorphism, we can achieve flexibility in our code because we can perform various operations by using methods with the same names according to requirements.

**->**The best real-life example of polymorphism is human behavior. One person can have different behavior.

**Types of Polymorphism in Java**

1. **Static polymorphism(compile-time polymorphism)/(overloading)**  
2. **Dynamic polymorphism(run-time polymorphism)(overriding)**

**Static Polymorphism in Java**

A polymorphism that exhibited during compilation is called **static polymorphism in java**. In static polymorphism, the behavior of a method is decided at compile-time.

Hence, [*Java compiler*](https://www.scientecheasy.com/2021/03/java-compiler.html/) binds method calls with method definition/body during compilation. Therefore, this type of polymorphism is also called **compile-time polymorphism in Java**.

Since binding is performed at compile-time, it is also known as **early binding**. Compile-time polymorphism can be achieved/implemented by method overloading in java.

[Method overloading](https://www.scientecheasy.com/2019/02/method-overloading-in-java.html/) is a mechanism in which a class has multiple methods having the same name but different signatures. It is one of the ways that Java implements polymorphism.

## **Dynamic Polymorphism**

A polymorphism that is exhibited at runtime is called dynamic polymorphism in java. In dynamic polymorphism, the behavior of a method is decided at runtime,

therefore, the JVM (Java Virtual Machine) binds the method call with method definition/body at runtime and invokes the relevant method during runtime when the method is called.

**7. Dynamic Binding:** [Dynamic binding](https://www.scientecheasy.com/2020/02/static-and-dynamic-binding-in-java.html/) (also known as late binding) is a way of connecting one program to another that is to be executed whenever it is called at runtime. It is associated with inheritance and polymorphism.

8. **Massage Passing:** In OOP, objects communicate with each other through member methods of the class. To establish communication between two objects, the following steps take place:

a. Creating classes that declare variables and methods.  
b. Creating objects of classes that already declared.  
c. Calling methods through suitable data to establish communication between objects.

Message passing involves three elements: name of object, name of method, and information to be sent. For example, consider the below statement.

## **Application of OOPs concepts in Java**

The concepts of OOPs provide many benefits for the programmer to design an efficient program. Due to its reusability feature, it is widely used in many areas. Some of the application areas of OOP are as follows:

a. Real-time systems  
b. Object-oriented database  
c. Graphical user interface design in the Windows operating system.  
d. Artificial intelligence and expert systems  
e. Parallel programming  
f. CAD/CAM software and in many areas.

# Exception Handling in Java

In general, an exception means a problem or an abnormal condition that stops a computer program from processing information in a normal way.

An **exception in java** is an object representing an error or an abnormal condition that occurs at runtime execution and interrupts (disrupts) the normal execution flow of the program.

In other words, unwanted and unexpected behaviour/event that interrupts the normal execution flow of the program is called exception in java. It is thrown from a method. The caller of the method can catch and handle the exception.

An exception can be identified only at runtime, not at compile time. Therefore, it is also called runtime errors that are thrown as exceptions in Java. They occur while a program is running.

For example, if we access an array using an index that is out of bounds, we will get a runtime error named ArrayIndexOutOfBoundsException.

If we enter a double value while the program expecting an integer value, we will get a runtime error called InputMismatchException.

When [JVM](https://www.scientecheasy.com/2021/03/what-is-jvm.html/) faces these kinds of errors or dividing an integer by zero in a program, it creates an exception object and throws it to inform us that an error has occurred.

If we want to continue the execution of remaining code in the program, we will have to handle exception object thrown by error condition and then display a user-friendly message for taking corrective actions. This task is known as exception handling in java

## **Realtime Example of Exception in Java**

* 1. Suppose you are watching a video on Youtube, suddenly, internet connectivity is disconnected or not working. In this case, you are not able to continue watching the video on Youtube. This interruption is nothing but an exception.

## **Why Exception occurs in Program?**

1. Opening a non-existing file in your program.  
   2. Reading a file from a disk but the file does exist there.  
   3. Writing data to a disk but the disk is full or unformatted.  
   4. When the program asks for user input and the user enters invalid data.  
   5. When a user attempts to divide an integer value by zero, an exception occurs.  
   6. When a data stream is in an invalid format, etc.

## **Exception Handling in Java**

The mechanism of handling unexpected errors in a java program is called exception handling. It is a powerful mechanism to handle runtime errors, ClassNotFoundException, FileNotFoundException, IOException, etc. so that the normal execution flow of the program can be maintained.

## **What is Exception Handler in Java?**

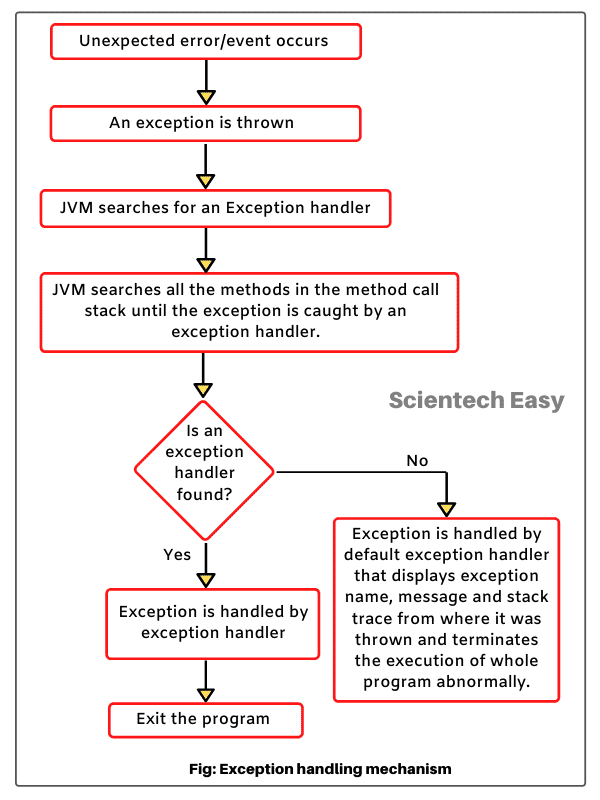
The code that catches the exception thrown by JVM is called **exception handler in Java**. It is responsible for receiving information about the exception/error.

When an exception occurs, exception handling transfers the control of execution of the program to an appropriate exception handler.

## **How does Exception handling mechanism work?**

The main purpose of using exception handling mechanism in a java program is to handle unexpected errors and maintain the normal flow of the program.

When an exceptional case occurs in a program, the exception handling mechanism performs the following tasks to manage unexpected situations in java program at runtime. The flow diagram can be seen in the below figure:

[](https://www.scientecheasy.com/2020/08/exception-handling-in-java.html/)

1. When an exception occurs inside a method in java program, the method in which exception has occurred creates an exception object (i.e., an object of exception class) internally with the help of JVM and hands it over to the java runtime system (JVM). This process is called **throwing an exception in java**.

The exception object contains information about the exception such as the name of exception, and Stack Trace/Location.

2. When a java method throws an exception, JVM searches for a method in the method call stack that can handle that exception. A method call stack is an ordered list of methods. This process continues until the exception is caught by an exception handler.

The method which handles thrown exception is called exception handler. It is used to catch the exception that is thrown by JVM. This process is called **catching an exception**.

3. If an exception handler is found, the control of execution is transferred to exception handler, and the statements specified inside exception handler are executed.

4. If JVM does not find an appropriate exception handler, exception is caught by the default exception handler provided by JVM.

Default exception handler is a part of the run-time system that displays exception information on the console such as exception name, message, and a full stack trace from where it was thrown.

The full stack track describes the sequence of steps that is responsible for throwing an error.

5. After printing exception information on the console, the default exception handler terminates the execution of the whole program abnormally.

Thus, exception handler mechanism works to manage unexpected situations in a program at runtime. The exception may be user-defined (custom exception) or predefined. Most of the exceptions are predefined and must be handled in the program to avoid runtime errors.

You can keep in mind the working of exception handling mechanism by the below points.

* Get the exception (Detects problem)
* Throw exception (Inform that an error has occurred)
* Catch the exception (Recieve error information)
* Handle exception (Take appropriate and corrective actions)

## **Advantage of Exception Handling**

1. The main advantage of exception handling technique is to maintain the normal flow of the program.

2. It provides flexibility in handling situations of errors.

3. It allows us to define a user-friendly message to handle the exception.

4. The exception handling technique helps to separate “Error-Handling code” from “Regular code.”

# Exception Hierarchy in Java

## Types of Exceptions in Java

1. **Predefined Exceptions** (Built-in-Exceptions)  
2. **Custom Exceptions**

## **Predefined Exceptions:**

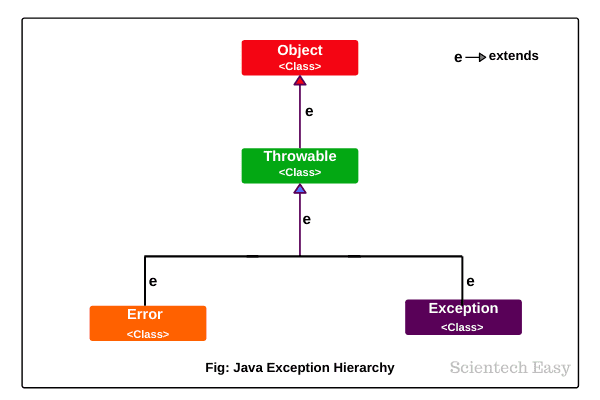
Predefined exceptions are those exceptions that are already defined by Java system. These exceptions are also called built-in-exceptions.

Java API supports exception handling by providing the number of predefined exceptions. These predefined exceptions are represented by classes in java.

When a predefined exception occurs, JVM (Java runtime system) creates an object of predefined exception class. All exceptions are derived from java.lang.Throwable class but not all exception classes are defined in the same package.

All the predefined exceptions supported by java are organized as subclasses in a hierarchy under the Throwable class.

The Throwable class is the root of exception hierarchy and is an immediate subclass of Object class. Let’s take a look at the java exception hierarchy, as shown in the below figure.

[](https://www.scientecheasy.com/2020/08/exception-hierarchy-in-java.html/)

* 1. **Throwable class:**

As shown in the above figure, Throwable class which is derived from Object class, is a top of exception hierarchy from which all exception classes are derived directly or indirectly. It is the root of all exception classes. It is present in java.lang package.

Throwable class is the superclass of all exceptions in java. This class has two subclasses: Error and Exception. Errors or exceptions occurring in java programs are objects of these classes. Using Throwable class, you can also create your own custom exceptions.

* 1. **Error:**

Error class is the subclass of Throwable class and a superclass of all the runtime error classes. It terminates the program if there is problem-related to a system or resources (JVM).

An error generally represents an unusual problem or situation from which it is difficult to recover. It does not occur by programmer mistakes. It generally occurs if the system is not working properly or resource is not allocated properly.

VirtualMachineError, StackOverFlowError, AssertionError, LinkageError, OutOfMmeoryError, etc are examples of error. We will learn more detail in further tutorials.

* 1. **Exception:**

It is represented by an Exception class that represents errors caused by the program and by external factors. Exception class is a subclass of Throwable class and a superclass of all the exception classes. All the exception classes are derived directly or indirectly from the Exception class. They generally originate from within the application.

The exception class provides two constructors:

* public Exception() (Default constructor)
* public Exception(String message) (It takes a message string as argument)

Each of the exception classes provides two constructors: one with no argument and another with a String type argument. Exception class does not provide its own method. It inherits all methods provided by Throwable class.

# What is the difference between errors and exceptions in Java?

Errors and exceptions are subclasses of the Throwable Java class. The **Error class** represents critical conditions that can not be caught and handled by the code of the program. On the other hand, the **Exception class** represents concerning conditions raised by the application itself; these can be caught and handled within the code to ensure that ​the application continues to run smoothly.

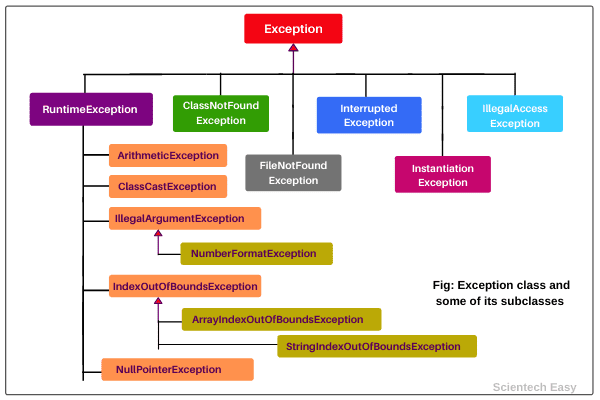
## Differences

The key differences between exceptions and errors are as follows:

| **Errors** | **Exceptions** |
| --- | --- |
| Errors are usually raised by the environment in which the application is running. For example, an error will occur due to a lack of system resources. | Exceptions are caused by the code of the application itself. |
| It is not possible to recover from an error. | The use of **try-catch** blocks can handle exceptions and recover the application from them. |
| Errors occur at run-time and are not known by the compiler; hence, they are classified as “unchecked.” | Exceptions can be “checked” or “unchecked,” meaning they may or may not be caught by the compiler. |
| “OutOfMemory” and “StackOverflow” are examples of errors. | “IndexOutOfBounds” is an example of an unchecked exception, while “ClassNotFound” is an example of a checked exception. |

## **Exception Class Hierarchy in Java**

The hierarchy of exception class in Java has been shown in the below figure that is very important for an interview purpose.

[](https://www.scientecheasy.com/2020/08/exception-hierarchy-in-java.html/)

### **Custom Exceptions:**

Custom exceptions are those exceptions that are created by users or programmers according to their own needs. The custom exceptions are also called user-defined exceptions that are created by extending the exception class.

So, Java provides the liberty to programmers to throw and handle exceptions while dealing with functional requirements of problems they are solving. We will discuss in more detail about custom exceptions in further tutorials.

Let’s see the brief description of each subclass of the Exception class.

## **RuntimeException class (Unchecked Exceptions)**

RuntimeException class is a subclass of the Exception class. It is thrown by JVM or programmatically when an arithmetic operation performed in the program is incorrect or defect/bug occurs in the program’s code.

RuntimeException and all its exception subclasses are not checked by Java compiler because they occur during runtime of a program. That’s why these exceptions are also called unchecked exceptions.

RuntimeException class consists of many other exception subclasses that are used to handle a specific type of exception. Apart from these exception subclasses of RuntimeException class shown in the above figure, there are also other subclasses of RuntimeException class which has not been shown in the hierarchy structure diagram to avoid complexity.

Let’s see a brief description of them.

* 1. **ArithmeticException:** This exception is thrown when arithmetic problems, such as a number is divided by zero, is occurred. That is, it is caused by maths error.
  2. **ClassCastException:** The ClassCastException is a runtime exception that is thrown by JVM when we attempt to invalid typecasting in the program. That is, it is thrown when we cast an object to a subclass of which an object is not an instance.

**3. IllegalArgumentException:** This runtime exception is thrown by programmatically when an illegal or appropriate argument is passed to call a method. This exception class has further two subclasses:

* NumberFormatException
* IllegalThreadStateException

**NumericFormatException:** NumberFormatException is thrown by programmatically when we try to convert a string into the numeric type and the process of illegal conversion fails. That is, it occurs due to the illegal conversion of a string to a numeric format.

**IllegalThreadStateException:** IllegalThreadStateException exception is a runtime exception that is thrown by programmatically when we attempt to perform any operation on a thread but it is incompatible with the current thread state.

**4. IndexOutOfBoundsException:** This exception class is thrown by JVM when an array or string is going out of the specified index. It has two further subclasses:

* ArrayIndexOutOfBoundsException
* StringIndexOutOfBoundsException

**ArrayIndexOutOfBoundsException:** ArrayIndexOutOfBoundsException exception is thrown when an array element is accessed out of the index.

**StringIndexOutOfBoundsException:** StringIndexOutOfBoundsException exception is thrown when a String or StringBuffer element is accessed out of the index.

* 1. **NullPointerException:** NullPointerException is a runtime exception that is thrown by JVM when we attempt to use null instead of an object. That is, it is thrown when the reference is null.
  2. **ArrayStoreException:** This exception occurs when we attempt to store any value in an array which is not of array type. For example, suppose, an array is of integer type but we are trying to store a value of an element of another type.
  3. **IllegalStateException:** The IllegalStateException exception is thrown by programmatically when the runtime environment is not in an appropriate state for calling any method.
  4. **IllegalMonitorStateException:** This exception is thrown when a thread does not have the right to monitor an object and tries to access wait(), notify(), and notifyAll() methods of the object.
  5. **NegativeArraySizeException:** The NegativeArraySizeException exception is thrown when an array is created with a negative size.

## **List of Checked Exceptions in Java**

Now, we have listed checked exceptions in a brief description.

**1. ClassNotFoundException:** The ClassNotFoundException is a kind of checked exception that is thrown when we attempt to use a class that does not exist.

Checked exceptions are those exceptions that are checked by the Java compiler itself.

**2. FileNotFoundException:** The FileNotFoundException is a checked exception that is thrown when we attempt to access a non-existing file.

**3. InterruptedException:** InterruptedException is a checked exception that is thrown when a thread is in sleeping or waiting state and another thread attempt to interrupt it.

**4. InstantiationException:** This exception is also a checked exception that is thrown when we try to create an object of abstract class or interface. That is, InstantiationException exception occurs when an abstract class or interface is instantiated.

**5. IllegalAccessException:** The IllegalAccessException is a checked exception and it is thrown when a method is called in another method or class but the calling method or class does not have permission to access that method.

**6. CloneNotSupportedException:** This checked exception is thrown when we try to clone an object without implementing the cloneable interface.

**7. NoSuchFieldException:** This is a checked exception that is thrown when an unknown variable is used in a program.

**8. NoSuchMethodException:** This checked exception is thrown when the undefined method is used in a program.

Hope that this tutorial has covered almost all the basic points related to the exception hierarchy in java. I hope that you will have understood the basic points of Throwable class and its subclasses: Exception and Error.

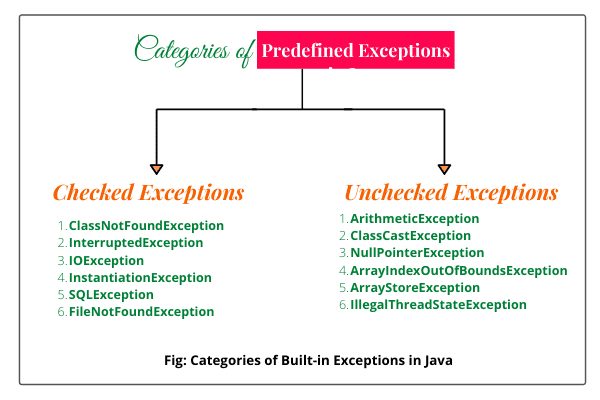
# Checked and Unchecked Exceptions in Java

**Checked and Unchecked Exceptions in Java |** In the previous tutorial, we have familiarized that there are two [types of exceptions](https://www.scientecheasy.com/2020/08/exception-hierarchy-in-java.html/) in java: First is predefined exceptions, and second user-defined exceptions.

The predefined exceptions are those exceptions that are already defined by the java system.

All the predefined exceptions are further divided into two groups:

1. **Checked Exceptions**  
2. **Unchecked Exceptions**



**difference between checked and unchecked exception**

|  |  |
| --- | --- |
| **Checked Exception** | **Unchecked Exception** |
| Checked exceptions occur at compile time. | Unchecked exceptions occur at runtime. |
| The compiler checks a checked exception. | The compiler does not check these types of exceptions. |
| These types of exceptions can be handled at the time of compilation. | These types of exceptions cannot be a catch or handle  at the time of compilation, because they get generated  by the mistakes in the program. |
| They are the sub-class of the exception class. | They are runtime exceptions and hence are not a part  of the Exception class. |
| Here, the JVM needs the exception to catch and handle. | Here, the JVM does not require the exception to  catch and handle. |
| Examples of Checked exceptions:   * File Not Found Exception * No Such Field Exception * Interrupted Exception * No Such Method Exception * Class Not Found Exception | Examples of Unchecked Exceptions:   * No Such Element Exception * Undeclared Throwable Exception * Empty Stack Exception * Arithmetic Exception * Null Pointer Exception * Array Index Out of Bounds Exception * Security Exception |

**The following table briefly describes these keywords.**

| **Keyword** | | **Description** |
| --- | --- | --- |
| **Try** | | We specify the block of code that might give rise to the exception in a special block with a “Try” keyword. |
| **Catch** | When the exception is raised it needs to be caught by the program.  This is done using a “catch” keyword. So a catch block follows the try block that raises an exception.  The keyword catch should always be used with a try. | |
| **Finally** | | Sometimes we have an important code in our program that needs to be executed irrespective  of whether or not the exception is thrown. This code is placed in a special block starting with  the “Finally” keyword. The Finally block follows the Try-catch block. |
| **Throw** | | The keyword “throw” is used to throw the exception explicitly. |
| **Throws** | | The keyword “Throws” does not throw an exception but is used to declare exceptions.  This keyword is used to indicate that an exception might occur in the program or method. |

# 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. |

# Java String

In [Java](https://www.javatpoint.com/java-tutorial), string is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java) of characters works same as Java string. For example:

1. **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=**new** String(ch);

is same as:

1. 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.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* [interfaces](https://www.javatpoint.com/interface-in-java).



## **CharSequence Interface**

The CharSequence interface is used to represent the sequence of characters. String, [StringBuffer](https://www.javatpoint.com/StringBuffer-class) and [StringBuilder](https://www.javatpoint.com/StringBuilder-class) classes implement it. It means, we can create strings in Java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

We will discuss immutable string later. Let's first understand what String in Java is and how to create the String object.

### **What is String in Java?**

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

### **How to create a string object?**

There are two ways to create String object:

1. By string literal
2. By new keyword

### **1) String Literal**

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

### **2) By new keyword**

1. String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

### **Java String Example**

**StringExample.java**

1. **public** **class** StringExample{
2. **public** **static** **void** main(String args[]){
3. String s1="java";//creating string by Java string literal
4. **char** ch[]={'s','t','r','i','n','g','s'};
5. String s2=**new** String(ch);//converting char array to string
6. String s3=**new** String("example");//creating Java string by new keyword
7. System.out.println(s1);
8. System.out.println(s2);
9. System.out.println(s3);
10. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=StringExample)

**Output:**

java

strings

example

# Immutable String in Java

A String is an unavoidable type of variable while writing any application program. String references are used to store various attributes like username, password, etc. In Java, **String objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once String object is created its data or state can't be changed but a new String object is created.

1. **class** Testimmutablestring{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s.concat(" Tendulkar");//concat() method appends the string at the end
5. System.out.println(s);//will print Sachin because strings are immutable objects
6. }
7. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Testimmutablestring)

**Output:**

Sachin

### **Why String objects are immutable in Java?**

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.

Following are some features of String which makes String objects immutable.

**1. ClassLoader:**

A ClassLoader in Java uses a String object as an argument. Consider, if the String object is modifiable, the value might be changed and the class that is supposed to be loaded might be different.

To avoid this kind of misinterpretation, String is immutable.

**2. Thread Safe:**

As the String object is immutable we don't have to take care of the synchronization that is required while sharing an object across multiple threads.

**3. Security:**

As we have seen in class loading, immutable String objects avoid further errors by loading the correct class. This leads to making the application program more secure. Consider an example of banking software. The username and password cannot be modified by any intruder because String objects are immutable. This can make the application program more secure.

**4. Heap Space:**

The immutability of String helps to minimize the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the new object. This feature allows Java to use the heap space efficiently.

### **Why String class is Final in Java?**

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

# Java String compare

We can compare String in Java on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

There are three ways to compare String in Java:

1. By Using equals() Method
2. By Using == Operator
3. By compareTo() Method

### **What is the difference between equals() and == in Java?**

Equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic.

“==” or equality operator in Java is a binary operator provided by Java programming language and used to compare primitives and objects. *public boolean equals(Object o)* is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. equals() method is used to compare the values of two objects.

# String Concatenation in Java

In Java, String concatenation forms a new String that is the combination of multiple strings. There are two ways to concatenate strings in Java:

1. By + (String concatenation) operator
2. By concat() method

## **1) String Concatenation by + (String concatenation) operator**

Java String concatenation operator (+) is used to add strings. For Example:

**TestStringConcatenation1.java**

1. **class** TestStringConcatenation1{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin"+" Tendulkar";
4. System.out.println(s);//Sachin Tendulkar
5. }
6. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestStringConcatenation1)

Sachin Tendulkar

The **Java compiler transforms** above code to this:

1. String s=(**new** StringBuilder()).append("Sachin").append(" Tendulkar).toString();

In Java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and it's append method. String concatenation operator produces a new String by appending the second operand onto the end of the first operand. The String concatenation operator can concatenate not only String but primitive values also. For Example:

**TestStringConcatenation2.java**

1. **class** TestStringConcatenation2{
2. **public** **static** **void** main(String args[]){
3. String s=50+30+"Sachin"+40+40;
4. System.out.println(s);//80Sachin4040
5. }
6. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestStringConcatenation2)

**Output:**

80Sachin4040

### **2) String Concatenation by concat() method**

The String concat() method concatenates the specified string to the end of current string. Syntax:

1. **public** String concat(String another)

Let's see the example of String concat() method.

**TestStringConcatenation3.java**

1. **class** TestStringConcatenation3{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin ";
4. String s2="Tendulkar";
5. String s3=s1.concat(s2);
6. System.out.println(s3);//Sachin Tendulkar
7. }
8. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestStringConcatenation3)

**Output:**

achin Tendulkar

The above Java program, concatenates two String objects **s1** and **s2** using **concat()** method and stores the result into **s3** object.

There are some other possible ways to concatenate Strings in Java,

### **1. String concatenation using StringBuilder class**

StringBuilder is class provides append() method to perform concatenation operation. The append() method accepts arguments of different types like Objects, StringBuilder, int, char, CharSequence, boolean, float, double. StringBuilder is the most popular and fastet way to concatenate strings in Java. It is mutable class which means values stored in StringBuilder objects can be updated or changed.

**StrBuilder.java**

1. **public** **class** StrBuilder
2. {
3. /\* Driver Code \*/
4. **public** **static** **void** main(String args[])
5. {
6. StringBuilder s1 = **new** StringBuilder("Hello");    //String 1
7. StringBuilder s2 = **new** StringBuilder(" World");    //String 2
8. StringBuilder s = s1.append(s2);   //String 3 to store the result
9. System.out.println(s.toString());  //Displays result
10. }
11. }

**Output:**

# Q. Difference between String and StringBuffer

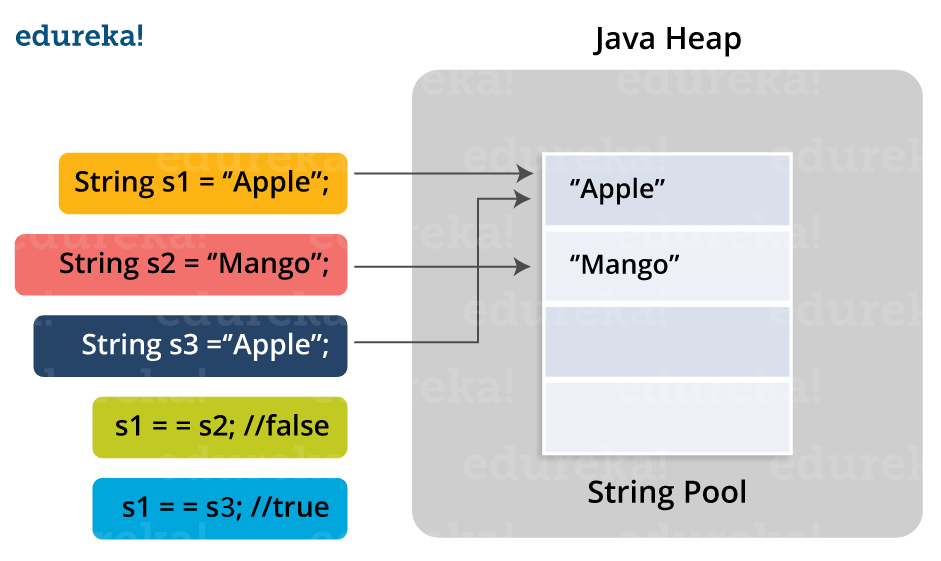
|  |  |  |
| --- | --- | --- |
| **No.** | **String** | **StringBuffer** |
| 1) | The String class is immutable. | The StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when we concatenate t strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |
| 4) | String class is slower while performing concatenation operation. | StringBuffer class is faster while performing concatenation operation. |
| 5) | String class uses String constant pool. | StringBuffer uses Heap memory |

# Q.Difference between StringBuffer and StringBuilder

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |
| 3) | StringBuffer was introduced in Java 1.0 | StringBuilder was introduced in Java 1.5 |

### **What is Java String Pool?**

Java String pool refers to a collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then the same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.

****

**Difference between String, StringBuilder, and StringBuffer.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Factor** | **String** | **StringBuilder** | **StringBuffer** |
| *Storage Area* | Constant String Pool | Heap Area | Heap Area |
| *Mutability* | Immutable | Mutable | Mutable |
| *Thread Safety* | Yes | No | Yes |
| *Performance* | Fast | More efficient | Less efficient |

### **Why Java Strings are immutable in nature?**

In Java, string objects are immutable in nature which simply means once the String object is created its state cannot be modified. Whenever you try to update the value of that object instead of updating the values of that particular object, Java creates a new string object. Java String objects are immutable as String objects are generally cached in the String pool. Since String literals are usually shared between multiple clients, action from one client might affect the rest. It enhances security, caching, synchronization, and performance of the application.

# Java Thread

A **thread in Java** simply represents a single independent path of execution of a group of statements. It is the flow of execution, from beginning to end, of a task.

When we write a group of statements in a program, these statements are executed by JVM one by one. This execution process is called thread in Java.

## **What is Process in Java?**

Thread in Java is the smallest unit of executable code in a program. It helps to divide a program into multiple parts to speed up the process. A process is a program that executes as a single thread. In other words, when an executable program is loaded into memory, it is called process.

**Points to be Noted:**

1. Every individual process has its own separate memory address space and can execute a different program.

2. Each process can have more than one thread.

3. Each process communicates through the operating system, files, and network.

When we will create a new thread in a program, it shares the same memory address space with other threads in a program whereas every individual process has its own separate memory address space.

## **Q. Why Java threads are lightweight process?**

Java Threads are also known as lightweight threads or light-weight processes. It means that they can be executed in the same memory space because all the threads in the main application program share the same address space in the memory so that they can easily communicate among themselves. Thus, they also take less space in memory and less processor time.

## **Main Thread in Java**

Every Java program has always at least one thread, even if you do not create any thread. This thread is called main thread. The main thread is also called parent thread and the rest of threads that are generated from it are called child threads of the program.

Main thread is the last thread to be executed in a program. When main thread finishes the execution, the program terminates immediately. Whenever Java program starts, main thread is created automatically.

This main thread is available in all programs. We can control the execution of the main thread by creating a Thread object and then using methods of Thread class.

To do so, we will have to create a Thread object by calling currentThread() method of class Thread. A Thread object can be created as follows:

Thread obj = Thread.currentThread();

## **Use of Thread in Java**

1. Threads are mainly used in server-side programs where we need to handle multiple clients on network or internet simultaneously.

2. Another important use of threads is in creating games and animations. For example, threads can be used to show picture in motion. In many games, threads help to perform more than one task simultaneously.

For example, in fighter game, a fighter plane may be from left to right. A machine gun in fighter plane continuously shoots enemy by releasing bullets.

Here, two tasks are happening simultaneously. One thread is moving the fighter plane while another thread releasing bullets simultaneously.

3. Generally, threads can be used to perform more than one task simultaneously.

**Explain the life cycle of thread?**

A thread can be in any of the five states :

1. **New** : When the instance of thread is created it will be in New state.

Ex : Thread t= new Thread();

In the above example t is in new state. The thread is created but not in active state to make it active we need to call start() method on it.

1. **Runnable state** : A thread can be in the runnable state in either of the following two ways : a) When the start method is invoked or

b) A thread can also be in runnable state after coming back from blocked or sleeping or waiting state.

3) **Running state** : If thread scheduler allocates cpu time, then the thread will be in running state.

4) Waited /Blocking/Sleeping state:

In this state the thread can be made temporarily inactive for a short period of time. A thread can be in the above state in any of the following ways:

1. The thread waits to acquire lock of an object.
2. The thread waits for another thread to complete. 3) The thread waits for notification of other thread.

5) **Dead State** : A thread is in dead state when thread’s run method execution is complete. It dies automatically when thread’s run method execution is completed and the thread object will be garbage collected.

**92) In how many ways we can create threads in java?**

We can create threads in java by any of the two ways :

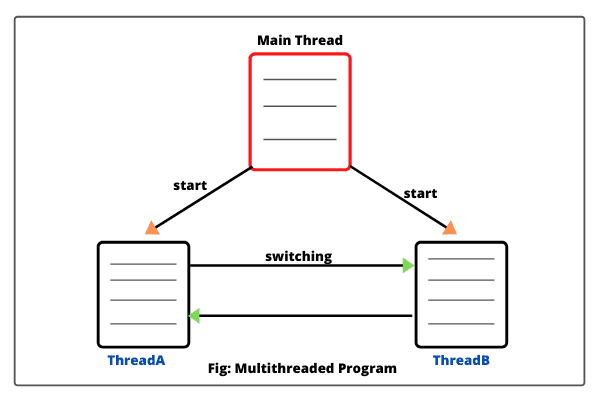
1. By ***extending Thread*** class

By ***Implementing Runnable*** interface

# Java Multithreading

Multithreading means multiple threads of execution concurrently. The process of executing multiple threads simultaneously (concurrently) is called **multithreading in Java**.

In other words, multithreading is a technique or programming concept in which a program (process) is divided into two or more subprograms (subprocesses), each of which can perform different tasks simultaneously (at the same time and in parallel manner). Each subprogram of a program is called thread in Java.

Look at the below figure where a Java program has three threads, one main and two others. The other two threads ThreadA and ThreadB are created and started from the main thread. When a program contains multiple flows of control, it is called multithreaded program.[](https://www.scientecheasy.com/2020/08/java-multithreading.html/)

Once initiated by the main thread, thread ThreadA and ThreadB run simultaneously and share the resources together. It is the same as people living in a joint family and sharing certain resources among all of them.

“When a program contains more than one thread, the CPU can switch between two threads to execute them at the same time. The switching between two threads is known as **context switch**.”

“This technique is useful for those applications which need multiple tasks to be done simultaneously. In a single processor system, multiple threads share CPU time that is known as **time-sharing**.”

## **Advantage of Multithreading**

1. In a multithreaded application program, different parts of the application are executed by different threads. The entire application does not stop even if an exception occurs in any of the threads. It does not affect other threads during the execution of the application.

2. Different threads are allotted to different processors and each thread is executed in different processors in parallel.

1. Multithreading helps to reduce computation time.  
   4. Multithreading technique improves the performance of the application.  
   5. Threads share the same memory address space. Hence, it saves memory.  
   6. Multithreaded program makes maximum utilization of CPU and keeping the idle time of CPU to minimum.  
   7. Context switching from one thread to another thread is less expensive than between processes.

### **Drawbacks of Multithreading in Java**

The drawbacks of multithreading are as follows:

1. Increased complexity.  
2. Synchronization of shared resources.  
3. In the multithreading programming concept, debugging is difficult. At times, result is unpredictable.  
4. Potential deadlocks.  
5. Programming complications may occur.

## **Multitasking in Java**

The process of executing one or more tasks concurrently or at the same time is called **multitasking**. It is the ability of an operating system to execute multiple tasks at once. The main purpose of multitasking is to use the idle time of CPU.

Multitasking can be implemented in two ways:

1. Process-based multitasking (Multiprocessing)  
2. Thread-based multitasking (Multithreading)

## **Thread-based Multitasking (Multithreading)**

A thread is a separate path of execution of code for each task within a program. In thread-based multitasking, a program uses multiple threads to perform one or more tasks at the same time by a processor.

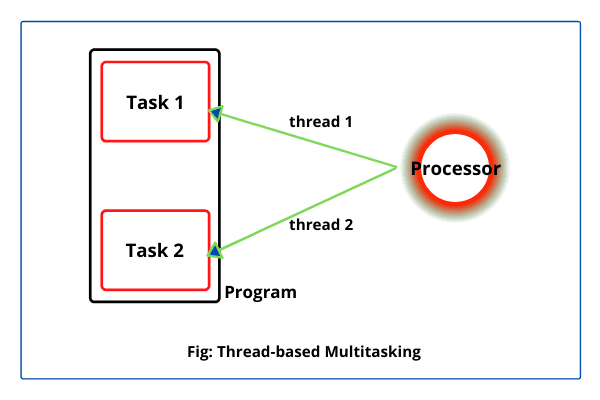
That is, thread-based multitasking feature allows you to execute several parts of the same program at once. Each thread has a different path of execution.

## **Realtime Example of Multithreading in Java**

Let’s take different examples of thread-based multithreading in Java.

1. A very good example of thread-based multithreading is a word processing program that checks the spelling of words in a document while writing the document. This is possible only if each action is performed by a separate thread.

2. Another familiar example is a browser that starts rendering a web page while it is still downloading the rest of page.

Consider a program as shown in the below figure. The program is divided into two parts. These parts may represent two separate blocks of code or two separate methods that can perform two different tasks of the same program.[](https://www.scientecheasy.com/2020/08/java-multithreading.html/)

Hence, a processor will create two separate threads to execute these two parts simultaneously. Each thread acts as an individual process that will execute a separate block of code.

Thus, a processor has two threads that will perform two different tasks at a time. This multitasking is called thread-based multiple tasking.

## **Advantage of Thread-based Multitasking over Process-Thread Multitasking**

The main advantages of thread-based multitasking as compared to process-based tasking are

1. Threads share the same memory address space.

2. Context switching from one thread to another thread is less expensive than between processes.

3. The cost of communication between threads is relatively low.

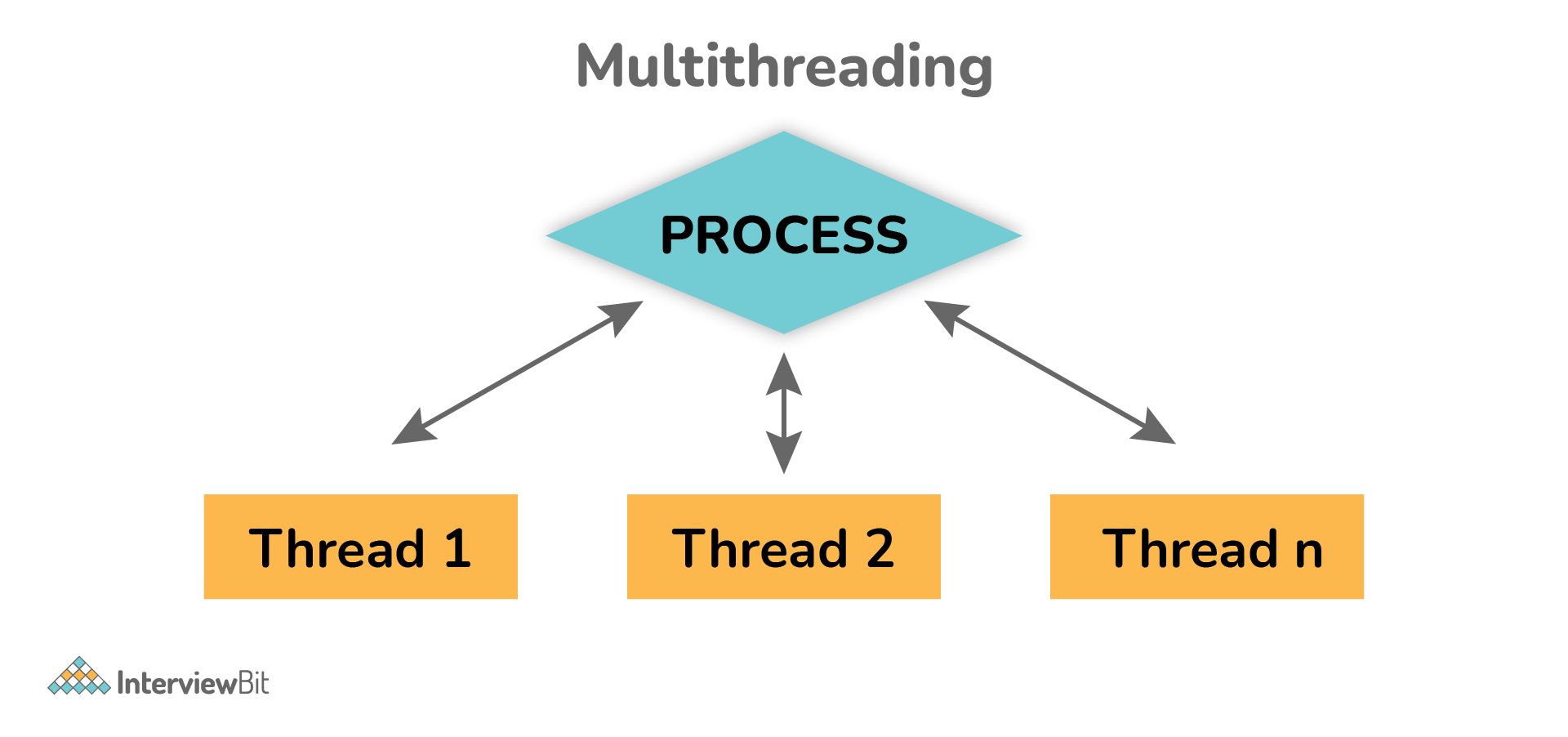
4. Threads are lightweight as compared to processes (heavyweight). They utilize the minimum resources of the system. They take less memory and less processor time.

Java supports thread-based multitasking and provides a high quality of facilities for multithreading programming.

A great way to remember the difference between process-based multitasking and thread-based multitasking is process-based multitasking works with multiple programs or processes whereas thread-based multitasking works with parts of one program.

### **What do you mean by Multithreading? Why is it important?**

Multithreading means multiple threads and is considered one of the most important [features of Java](https://www.interviewbit.com/java-interview-questions/). As the name suggests, it is the ability of a CPU to execute multiple threads independently at the same time but sharing the process resources simultaneously. Its main purpose is to provide simultaneous execution of multiple threads to utilize the CPU time as much as possible. It is a Java feature where one can subdivide the specific program into two or more threads to make the execution of the program fast and easy.



### **1. What are the benefits of using Multithreading?**

There are various benefits of multithreading as given below:

* Allow the program to run continuously even if a part of it is blocked.
* Improve performance as compared to traditional parallel programs that use multiple processes.
* Allows to write effective programs that utilize maximum CPU time
* Improves the responsiveness of complex applications or programs.
* Increase use of CPU resources and reduce costs of maintenance.
* Saves time and parallelism tasks.
* If an exception occurs in a single thread, it will not affect other threads as threads are independent.
* Less resource-intensive than executing multiple processes at the same time.

### **3. What are the two ways of implementing thread in Java?**

There are basically two ways of implementing thread in java as given below:

* Implementing **Runnable** interface in Java

Example:

**class** **MultithreadingDemo** **extends** **Thread**

{

**public** **void** **run**()

{

System.out.println("My thread is in running state.");

}

**public** **static** **void** **main**(String args[])

{

MultithreadingDemoobj=**new** MultithreadingDemo();

obj.start();

}

}

Output:

My thread is in running state.

* Extending the **Thread** class.

Example:

**class** **MultithreadingDemo** **implements** **Runnable**

{

**public** **void** **run**()

{

System.out.println("My thread is in running state.");

}

**public** **static** **void** **main**(String args[])

{

MultithreadingDemo obj=**new** MultithreadingDemo();

Threadtobj =**new** Thread(obj); tobj.start();

}

}

Output:

My thread is in running state.

### **What's the difference between thread and process?**

| **Thread** | **Process** |
| --- | --- |
| It is a subset of a subunit of a process. | It is a program in execution containing multiple threads. |
| In this, inter-thread communication is faster, less expensive, easy and efficient because threads share the same memory address of the process they belong to. | In this, inter-process communication is slower, expensive, and complex because each process has different memory space or address., |
| These are easier to create, lightweight, and have less overhead. | These are difficult to create, heavyweight, and have more overhead. |
| It requires less time for creation, termination, and context switching. | It requires more time for creation, termination, and context switching. |
| Processes with multiple threads use fewer resources. | Processes without threads use more resources. |
| Threads are parts of a process, so they are dependent on each other but each thread executes independently. | Processes are independent of each other. |
| There is a need for synchronization in threads to avoid unexpected scenarios or problems. | There is no need for synchronization in each process. |
| They share data and information with each other. | They do not share data with each other. |

**Q.User Thread vs Daemon Thread**

| **User Thread** | **Daemon Thread** |
| --- | --- |
| JVM waits for user threads to finish their tasks before termination. | JVM does not wait for daemon threads to finish their tasks before termination. |
| These threads are normally created by the user for executing tasks concurrently. | These threads are normally created by JVM. |
| They are used for critical tasks or core work of an application. | They are not used for any critical tasks but to do some supporting tasks. |
| These threads are referred to as high-priority tasks, therefore are required for running in the foreground. | These threads are referred to as low priority threads, therefore are especially required for supporting background tasks like garbage collection, releasing memory of unused objects, etc. |

### **Q. What are the wait() and sleep() methods?**

**wait()**: As the name suggests, it is a non-static method that causes the current thread to wait and go to sleep until some other threads call the notify () or notifyAll() method for the object’s monitor (lock). It simply releases the lock and is mostly used for inter-thread communication. It is defined in the object class, and should only be called from a synchronized context.

Example:

**synchronized**(monitor)

{

monitor.wait(); Here Lock Is Released by Current Thread

}

**sleep()**: As the name suggests, it is a static method that pauses or stops the execution of the current thread for some specified period. It doesn’t release the lock while waiting and is mostly used to introduce pause on execution. It is defined in thread class, and no need to call from a synchronized context.

Example:

**synchronized**(monitor)

{

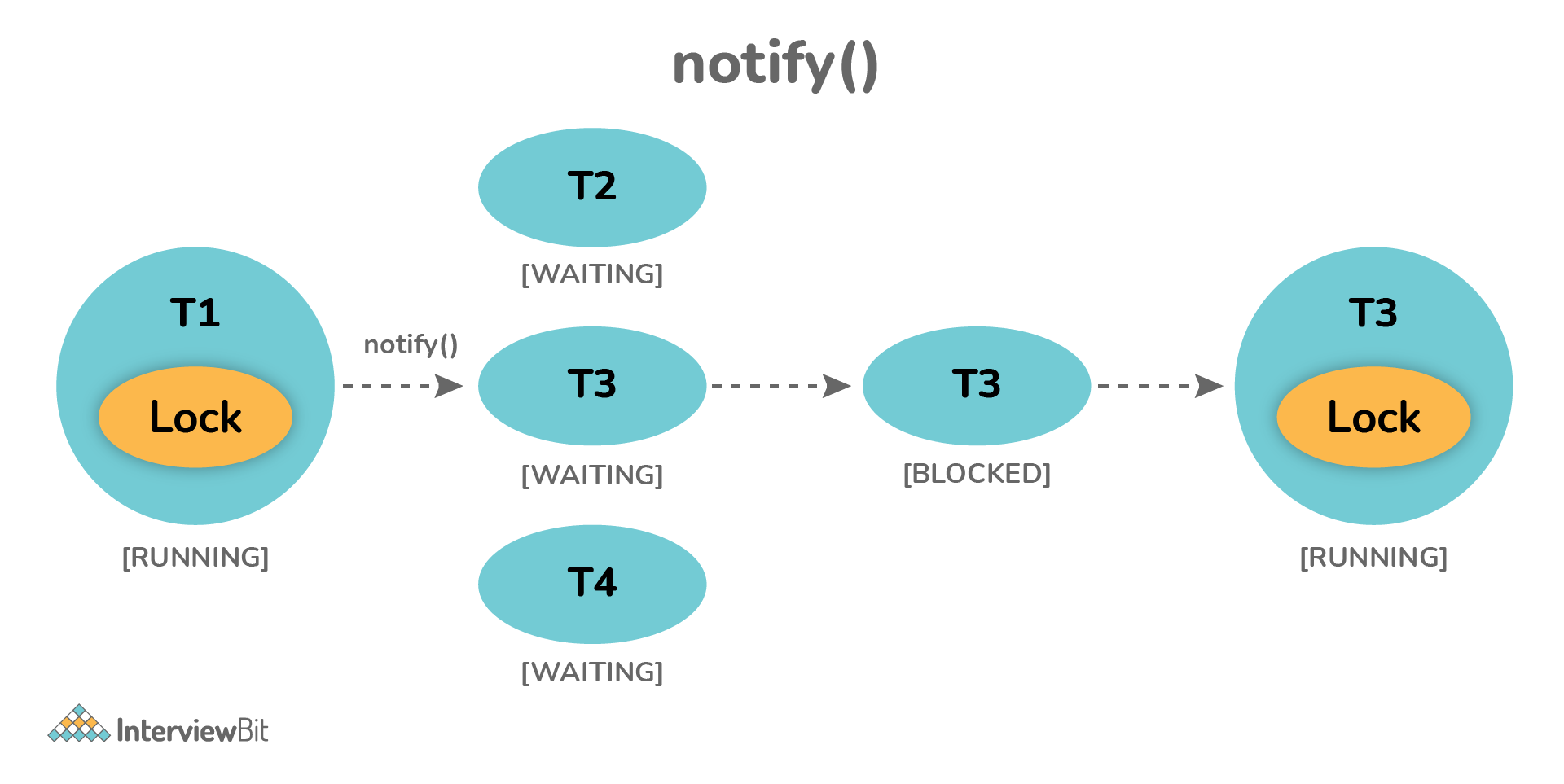
Thread.sleep(1000); Here Lock Is Held by The Current Thread

//after 1000 milliseconds, the current thread will wake up, or after we call that is interrupt() method

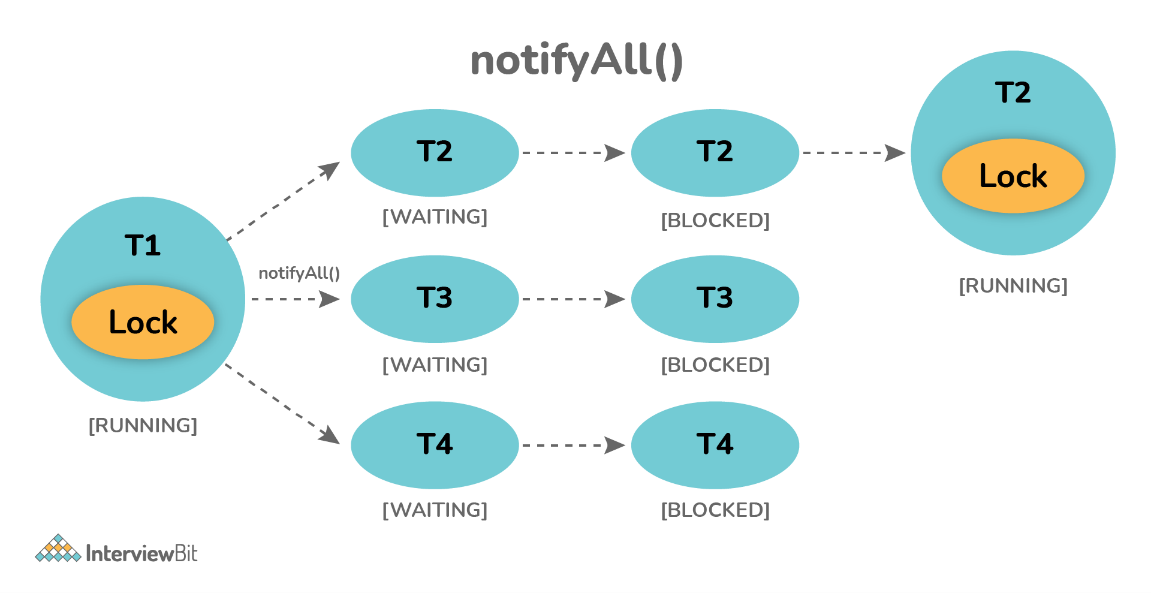
}

### 9. What’s the difference between notify() and notifyAll()?

**notify()**: It sends a notification and wakes up only a single thread instead of multiple threads that are waiting on the object’s monitor.



**notifyAll()**: It sends notifications and wakes up all threads and allows them to compete for the object's monitor instead of a single thread.



### **10. Why wait(), notify(), and notifyAll() methods are present in Object class?**

We know that every object has a monitor that allows the thread to hold a lock on the object. But the thread class doesn't contain any monitors. Thread usually waits for the object’s monitor (lock) by calling the wait() method on an object, and notify other threads that are waiting for the same lock using notify() or notifyAll() method.  Therefore, these three methods are called on objects only and allow all threads to communicate with each that are created on that object.

### **11. What is Runnable and Callable Interface? Write the difference between them.**

Both the interfaces are generally used to encapsulate tasks that are needed to be executed by another thread. But there are some differences between them as given below:   
  
**Running Interface**: This interface is basically available in Java right from the beginning. It is simply used to execute code on a concurrent thread.    
**Callable Interface**: This interface is basically a new one that was introduced as a part of the concurrency package. It addresses the limitation of runnable interfaces along with some major changes like generics, enum, static imports, variable argument method, etc. It uses generics to define the return type of object.

**public** **interface** **Runnable**

{

**public** **abstract** **void** **run**();

}

**public** **interface** **Callable**<**V**>

{

V **call**() **throws** Exception;

}

**Runnable Interface vs Callable Interface**

| **Runnable Interface** | **Callable Interface** |
| --- | --- |
| It does not return any result and therefore, cannot throw a checked exception. | It returns a result and therefore, can throw an exception. |
| It cannot be passed to invokeAll method. | It can be passed to invokeAll method. |
| It was introduced in JDK 1.0. | It was introduced in JDK 5.0, so one cannot use it before Java 5. |
| It simply belongs to Java.lang. | It simply belongs to java.util.concurrent. |
| It uses the run() method to define a task. | It uses the call() method to define a task. |
| To use this interface, one needs to override the run() method. | To use this interface, one needs to override the call() method. |

### **12. What is the start() and run() method of Thread class?**

**start()**: In simple words, the start() method is used to start or begin the execution of a newly created thread. When the start() method is called, a new thread is created and this newly created thread executes the task that is kept in the run() method. One can call the start() method only once.    
  
**run()**: In simple words, the run() method is used to start or begin the execution of the same thread. When the run() method is called, no new thread is created as in the case of the start() method. This method is executed by the current thread. One can call the run() method multiple times.

### **13. Explain thread pool?**

A Thread pool is simply a collection of pre-initialized or worker threads at the start-up that can be used to execute tasks and put back in the pool when completed. It is referred to as pool threads in which a group of fixed-size threads is created.  By reducing the number of application threads and managing their lifecycle, one can mitigate the issue of performance using a thread pool. Using threads, performance can be enhanced and better system stability can occur. To create the thread pools, java.util.concurrent.Executors class usually provides factory methods.

### **14. What’s the purpose of the join() method?**

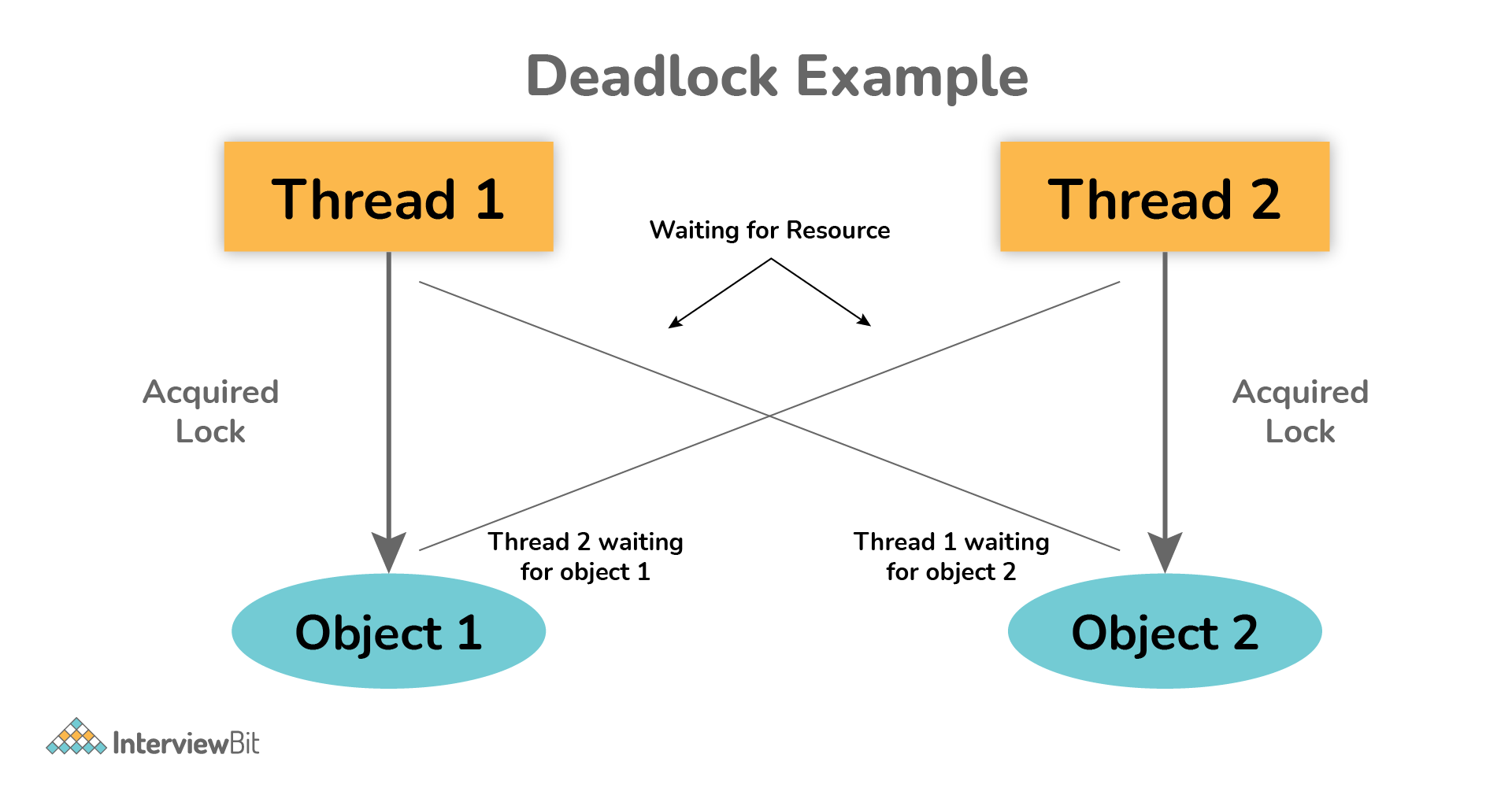
**join()** method is generally used to pause the execution of a current thread unless and until the specified thread on which join is called is dead or completed. To stop a thread from running until another thread gets ended, this method can be used. It joins the start of a thread execution to the end of another thread’s execution. It is considered the final method of a thread class.

### **15. What do you mean by garbage collection?**

Garbage collection is basically a process of managing memory automatically. It uses several GC algorithms among which the popular one includes Mark and Sweep. The process includes three phases i.e., marking, deletion, and compaction/copying. In simple words, a garbage collector finds objects that are no longer required by the program and then delete or remove these unused objects to free up the memory space.

### **16. Explain the meaning of the deadlock and when it can occur?**

Deadlock, as the name suggests, is a situation where multiple threads are blocked forever. It generally occurs when multiple threads hold locks on different resources and are waiting for other resources to complete their task.



The above diagram shows a deadlock situation where two threads are blocked forever.  Thread 1 is holding Object 1 but needs object 2 to complete processing whereas Thread 2 is holding Object 2 but needs object 1 first. In such conditions, both of them will hold lock forever and will never complete tasks.

### **17. Explain volatile variables in Java?**

A volatile variable is basically a keyword that is used to ensure and address the visibility of changes to variables in multithreaded programming. This keyword cannot be used with classes and methods, instead can be used with variables. It is simply used to achieve thread-safety. If you mark any variable as volatile, then all the threads can read its value directly from the main memory rather than CPU cache, so that each thread can get an updated value of the variable.

### **18. How do threads communicate with each other?**

Threads can communicate using three methods i.e., wait(), notify(), and notifyAll().

### **19. Can two threads execute two methods (static and non-static concurrently)?**

Yes, it is possible. If both the threads acquire locks on different objects, then they can execute concurrently without any problem.

### **20. What is the purpose of the finalize() method?**

Finalize() method is basically a method of Object class specially used to perform cleanup operations on unmanaged resources just before garbage collection. It is not at all intended to be called a normal method. After the complete execution of finalize() method, the object gets destroyed automatically.

## **Multithreading Interview Questions in Java for Experienced**

### **21. What is the synchronization process? Why use it?**

Synchronization is basically a process in java that enables a simple strategy for avoiding thread interference and memory consistency errors. This process makes sure that resource will be only used one thread at a time when one thread tries to access a shared resource. It can be achieved in three different ways as given below:

* By the synchronized method
* By synchronized block
* By static synchronization

Syntax:

synchronized (object)

{

//statement to be synchronized

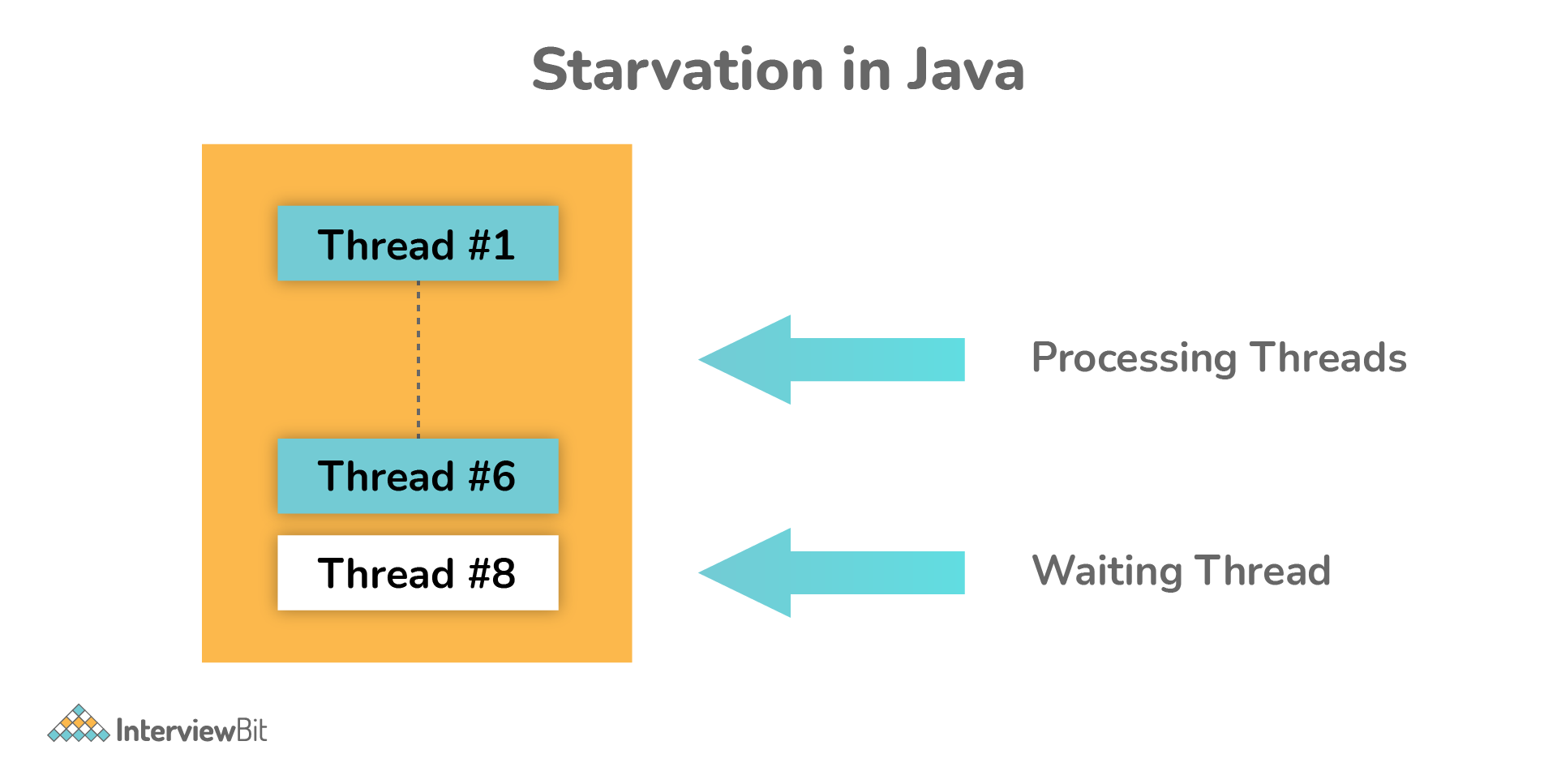
}

### **22. What is synchronized method and synchronized block? Which one should be preferred?**

**Synchronized Method**: In this method, the thread acquires a lock on the object when they enter the synchronized method and releases the lock either normally or by throwing an exception when they leave the method.  No other thread can use the whole method unless and until the current thread finishes its execution and release the lock. It can be used when one wants to lock on the entire functionality of a particular method.   
  
**Synchronized Block**: In this method, the thread acquires a lock on the object between parentheses after the synchronized keyword, and releases the lock when they leave the block. No other thread can acquire a lock on the locked object unless and until the synchronized block exists. It can be used when one wants to keep other parts of the programs accessible to other threads.  
   
Synchronized blocks should be preferred more as it boosts the performance of a particular program. It only locks a certain part of the program (critical section) rather than the entire method and therefore leads to less contention.

### **23. What is thread starvation?**

Thread starvation is basically a situation or condition where a thread won’t be able to have regular access to shared resources and therefore is unable to proceed or make progress. This is because other threads have high priority and occupy the resources for too long. This usually happens with low-priority threads that do not get CPU for its execution to carry on.



### **27. Explain context switching.**

Context switching is basically an important feature of multithreading. It is referred to as switching of CPU from one thread or process to another one. It allows multiple processes to share the same CPU. In context switching, the state of thread or process is stored so that the execution of the thread can be resumed later if required.

### **36. What is semaphore?**

Semaphore is regarded as a thread synchronization construct that is usually required to control and manage the access to the shared resource using counters. It simply sets the limit of the thread. The semaphore class is defined within the package java.util.concurrent and can be used to send signals between threads to avoid missed signals or to guard critical sections. It can also be used to implement resource pools or bounded collection

# Life Cycle of Thread in Java

A thread is a path of execution in a program that enters in any one of the following five states during its life cycle. The five states are as follows:

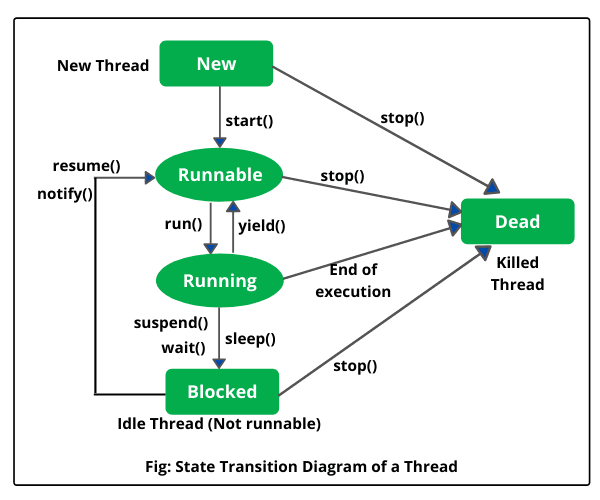
1. New

2. Runnable

3. Running

4. Blocked (Non-runnable state)

5. Dead



* 1. **New (Newborn State):** When we create a thread object using Thread class, thread is born and is known to be in Newborn state. That is, when a thread is born, it enters into new state but the start() method has not been called yet on the instance.
  2. **Runnable state:** Runnable state means a thread is ready for execution. When the start() method is called on a new thread, thread enters into a runnable state.
  3. **Running state:** Running means Processor (CPU) has allocated time slot to thread for its execution. When thread scheduler selects a thread from the runnable state for execution, it goes into running state. Look at the above figure.
  4. **Blocked state:** A thread is considered to be in the blocked state when it is suspended, sleeping, or waiting for some time in order to satisfy some condition.
  5. **Dead state:** A thread dies or moves into dead state automatically when its run() method completes the execution of statements. That is, a thread is terminated or dead when a thread comes out of run() method. A thread can also be dead when the stop() method is called.

# Difference between Comparable and Comparator

Comparable and Comparator both are interfaces and can be used to sort collection elements.

However, there are many differences between Comparable and Comparator interfaces that are given below.

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |