## What is Java?

## Types of Java Applications

## Java Platforms / Editions

# C++ vs Java

## What happens at runtime?

### Q) Can you save a Java source file by another name than the class name?

# Difference between JDK, JRE, and JVM

### Classloader

# Java Variables

# Data Types in Java

# Operators in Java

### Loop Statements

### Jump Statements

# Java OOPs Concepts

### What is an object in Java

## What is a class in Java

### Inheritance

### Polymorphism

## Runtime Polymorphism in Java

### Upcasting

### static binding

### Dynamic binding

# Method Overloading in Java

# Method Overriding in Java

1. Difference between method overloading and method overriding in java
2. Covariant Return Type

#### Abstraction

# Interface

# Difference between abstract class and interface

### Encapsulation

### Coupling

### Association

### Aggregation

## Advantage of OOPs over Procedure-oriented programming language

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

### Method in Java

## 3 Ways to initialize object

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

## Anonymous object

# Constructors in Java

### What is the purpose of a default & parameterized constructor?

## Difference between constructor and method in Java

## Static,This,Super,Final,Strictfp keyword

# Instance initializer block

# Access Modifiers in Java

### Java Access Modifiers with Method Overriding

# Object class in Java

# Object Cloning in Java

# Wrapper,Autoboxing &Unboxing classes in Java

### How to create a string object?

# Difference between String and StringBuffer

# Exception Handling in Java

## Diff between Checked and Unchecked Exceptions

1. Java Exception Keywords

# Java Exception Propagation

# Exception Handling with Method Overriding in Java

##################################################################

## What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

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

## Types of Java Applications

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

#### 1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone applications are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

#### 2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, [Servlet](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), [Struts](https://www.javatpoint.com/struts-2-tutorial), [Spring](https://www.javatpoint.com/spring-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), etc. technologies are used for creating web applications in Java.

#### 3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, [EJB](https://www.javatpoint.com/ejb-tutorial) is used for creating enterprise applications.

#### 4) Mobile Application

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

## Java Platforms / Editions

There are 4 platforms or editions of Java:

#### 1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, [String](https://www.javatpoint.com/java-string), Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

#### 2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, [JPA](https://www.javatpoint.com/jpa-tutorial), etc.

#### 3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

#### 4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

# Features of Java

### Simple

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

### Object-oriented

Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

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

Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. [Class](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

Platform Independent

Java is platform independent because it is different from other languages like [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

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

### Secured

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

* **No explicit pointer**
* **Java Programs run inside a virtual machine sandbox**
* **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
* **Bytecode Verifier:** it ensures that code passed to the Java interpreter is in a fit state to be executed and can run without fear of breaking the Java interpreter. I
* **Security Manager:** A security manager is an object that defines a security policy for an application. This policy specifies actions that are unsafe or sensitive.

### Robust

The English meaning of Robust is strong. Java is robust because:

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

### Architecture-neutral

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

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

### Portable

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

### High-performance

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

### Distributed

### Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications.

### This feature of Java makes us able to access files by calling the methods from any machine on the internet.

### Multi-threaded

A thread **refers to a single sequential flow of activities being executed in a process**;

Each JVM server can have a maximum of 256 threads to run Java applications.

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

### Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

# C++ vs Java

There are many differences and similarities between the [C++ programming](https://www.javatpoint.com/cpp-tutorial) language and [Java](https://www.javatpoint.com/java-tutorial). A list of top differences between C++ and Java are given below:

| **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. |
| **Goto** | C++ supports the [goto](https://www.javatpoint.com/cpp-goto-statement) statement. | Java doesn't support the goto statement. |
| **Multiple inheritance** | C++ supports multiple inheritance. | Java doesn't support multiple inheritance through class. It can be achieved by using [interfaces in java](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 pointers 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 compilers 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. |
| **unsigned right shift >>>** | C++ doesn't support >>> operator. | Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works the same as the >> operator. |
| **Inheritance Tree** | C++ always creates a new inheritance tree. | Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the [inheritance](https://www.javatpoint.com/inheritance-in-java) tree in java. |
| **Hardware** | C++ is nearer to hardware. | Java is not so interactive with hardware. |
| **Object-oriented** | C++ is an object-oriented language. | Java is also an [object-oriented](https://www.javatpoint.com/java-oops-concepts) language. However, everything (except fundamental types) is an object in Java. |

**Having a semicolon at the end of class is optional in Java.**

## Valid Java main() method signature

1. **public static void main(String[] args)**
2. **public static void main(String []args)**
3. **public static void main(String args[])**
4. **public static void main(String... args)**
5. **static public void main(String[] args)**
6. **public static final void main(String[] args)**
7. **final public static void main(String[] args)**
8. **final strictfp public static void main(String[] args)**

## Invalid Java main() method signature

1. **public void main(String[] args)**
2. **static void main(String[] args)**
3. **public void static main(String[] args)**
4. **abstract public static void main(String[] args)**

## What happens at runtime?

At runtime, the following steps are performed:

**Classloader:** It is the subsystem of JVM that is used to load class files.

**Bytecode Verifier:** Checks the code fragments for illegal code that can violate access rights to objects.

**Interpreter:** Read bytecode stream then execute the instructions.

### Q) Can you save a Java source file by another name than the class name?

**Yes, if the class is not public.**

# Difference between JDK, JRE, and JVM

### JVM

JVM (Java Virtual Machine) is an abstract machine. **It is a specification that provides a runtime environment in which Java bytecode can be executed.** It can also run those programs which are written in other languages and compiled to Java bytecode.

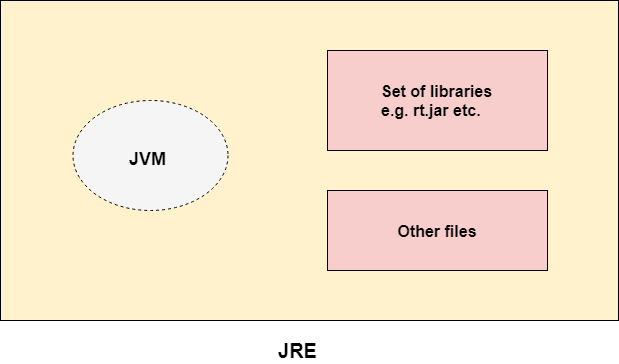
**JVM is platform dependent).**

**The JVM performs the following main tasks:**

* **Loads code**
* **Verifies code**
* **Executes code**
* **Provides runtime environment**

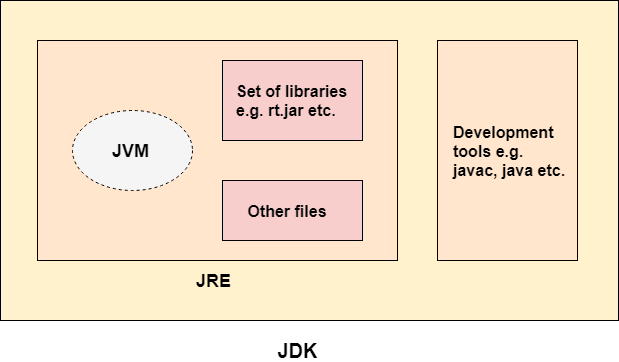
### JRE

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

****

### JDK

**JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications .**

****

### 

### 

### Classloader

### Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. **Bootstrap ClassLoader:** This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
2. **Extension ClassLoader:** This is the child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside the *$JAVA\_HOME/jre/lib/ext* directory.
3. **System/Application ClassLoader:** This is the child classloader of Extension classloader. It loads the class files from classpath. By default, classpath is set to the current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

**Just-In-Time(JIT) compiler:** It is used to improve the performance. J**IT compiles parts of the byte code that have similar functionality at the same time,** and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

# Java Variables

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) is executed. A variable is assigned with a data type.

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

#### 1) Local Variable

A variable declared inside the body of the method is called a 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 the "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.

# Data Types in Java

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

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

| **Data Type** | **Default Value** | **Default size** |
| --- | --- | --- |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

# 

# Operators in Java

**Operator** in [Java](https://www.javatpoint.com/java-tutorial) is a symbol that is used to perform operations. For example: +, -, \*, / etc.

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

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

**Java provides three types of control flow statements.**

1. Decision Making statements
   * if statements
   * switch statement
2. Loop statements
   * do while loop
   * while loop
   * for loop
   * for-each loop
3. Jump statements
   * break statement
   * continue statement

### Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order.

### Java for loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iterations is **fixed**, it is recommended to use a for loop.

### Java while loop

The [while loop](https://www.javatpoint.com/java-while-loop) is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop.**It is also known as the entry-controlled loop since the condition is checked at the start of the loop.**

### Java do-while loop

The [do-while loop](https://www.javatpoint.com/java-do-while-loop) checks the condition at the end of the loop after executing the loop statements. **When the number of iterations is not known and we have to execute the loop at least once, we can use a do-while loop.**

Java do-while loop is called an **exit control loop**.

### Jump Statements

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

### Java break statement

The [break statement](https://www.javatpoint.com/java-break) is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

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

### Java continue statement

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

Unlike the break statement, the [continue statement](https://www.javatpoint.com/java-continue) doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

## Types of Java Comments

There are three types of comments in Java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

# Java OOPs Concepts

The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language.

### What is an object in Java

An entity that has state and behavior is known as an object e.g.,

An object is *a real-world entity*.

An Object can be defined as an instance of a class.

An object has three characteristics:

* **State:** represents the data (value) of an object.
* **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
* **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

## What is a class in Java

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

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

### Inheritance

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

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

## Q) Why is multiple inheritance not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

### Polymorphism

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

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

## Runtime Polymorphism in Java

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

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

### Upcasting

If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.

## Downcasting

When Subclass type refers to the object of Parent class, it is known as downcasting.

## Java Runtime Polymorphism with Data Member

A method is overridden, not the data members, so runtime polymorphism can't be achieved by data members.

### static binding

When the type of the object is determined at compile time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding.

### Dynamic binding

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

# Method Overloading in Java

If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having the same name but different in parameters, it is known as **Method Overloading**. It increases the readability of the [program](https://www.javatpoint.com/java-programs).

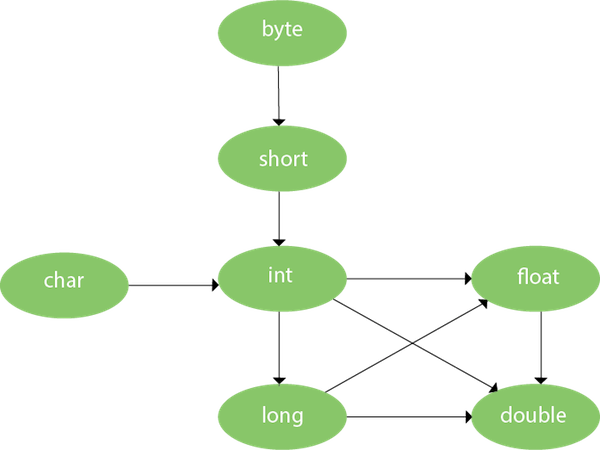
### Different ways to overload the method

There are two ways to overload the method in java

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

#### In Java, Method Overloading is not possible by changing the return type of the method only.

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

In java, method overloading is not possible by changing the return type of the method only because of ambiguity.

# Method Overriding in Java

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

* Method overriding is used for runtime polymorphism

#### In other words, **If a subclass provides the specific implementation of the method that has been declared by one of its parent classes, it is known as method overriding.**

#### Rules for Java Method Overriding

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

Difference between method overloading and method overriding in java

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

# Covariant Return Type It is possible to override a method by changing the return type if a subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type.

#### Abstraction

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

# Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).**It is used to acheive abstraction.**

### Abstract Method in Java

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

**A factory method is a method that returns the instance of the class.**

# Interface in Java

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

The interface in Java is *a mechanism to achieve* [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method bodies. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

**Since Java 8, we can have default and static methods in an interface.**

**Since Java 9, we can have private methods in an interface.**

## Why use the Java interface?

There are mainly three reasons to use interfaces. 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.

**Interface fields are public, static and final by default, and the methods are public and abstract.**

## Multiple inheritance in Java by interface

**If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.**

## Q) What is a marker or tagged interface?

An interface which has no member is known as a marker or tagged interface, for example, [Serializable](https://www.javatpoint.com/serialization-in-java), Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

# Difference between abstract class and interface

| **Abstract class** | **Interface** |
| --- | --- |
| 1) Abstract classes can **have abstract and non-abstract** methods. | Interfaces can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interfaces **can't provide the implementation of abstract classes**. |
| **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare the interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using the keyword "extends". | An **interface** can be implemented using the keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |

**Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).**

### 

### Encapsulation

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

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

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

### Advantage of Encapsulation in Java

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

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

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

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

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

### Coupling

**Coupling refers to the knowledge or information or dependency of another class.** It arises when classes are aware of each other. I**f a class has the detailed information of another class, there is strong coupling.** In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. **You can use interfaces for the weaker coupling because there is no concrete implementation.**

### Association

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

* One to One
* One to Many
* Many to One, and
* Many to Many

### Aggregation

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

Aggregation is a way to achieve Association. A**ggregation represents the relationship where one object contains other objects as a part of its state.** It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

**Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice**.

### Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

## Advantage of OOPs over Procedure-oriented programming language

1) **OOPs make development and maintenance easier**, whereas, in a procedure-oriented programming language, **it is not easy to manage if code grows as project size increases.**

2) OOPs provide data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

3) OOPs provide the ability to simulate real-world events much more effectively. We can provide the solution of real word problems if we are using the Object-Oriented Programming language.

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

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

### Method in Java

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

#### Advantage of Method

* Code Reusability
* Code Optimization

### new keyword in Java

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

## 3 Ways to initialize object

There are 3 ways to initialize objects in Java.

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

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

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

* By new keyword
* By newInstance() method
* By clone() method
* By deserialization
* By factory method etc.

## Anonymous object

Anonymous simply means nameless. **An object which has no reference is known as an anonymous object.** It can be used at the time of object creation only.

# Constructors in Java

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

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

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

We can have private,protected,public or default constructor in java.A constructor is called "Default Constructor" when it doesn't have any parameter.

### Q) What is the purpose of a default constructor?

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

### Why use the parameterized constructor?

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

## Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

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

## Java Copy Constructor

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

### Q) Does the constructor return any value?

Yes, it is the current class instance (You cannot use return type yet it returns a value).

### Static keyword

Java static property to share to all objects.

**The static variable is used to refer to the common property of all objects (which is not unique for each object),**

There are two main restrictions for the static method. They are:

1. The static method can not use non-static data members or call non-static methods directly.
2. this and super cannot be used in a static context.

## Java static block

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

# this keyword in Java

In Java, this is a **reference variable** that refers to the current object.

## Usage of Java this keyword

Here is the 6 usage of java this keyword.

1. [This can be used to refer to the current class instance variable.](https://www.javatpoint.com/this1)
2. [this can be used to invoke current class method (implicitly)](https://www.javatpoint.com/this2)
3. [this() can be used to invoke the current class constructor.](https://www.javatpoint.com/this3)
4. [this can be passed as an argument in the method call.](https://www.javatpoint.com/this4)
5. [This can be passed as an argument in the constructor call.](https://www.javatpoint.com/this5)
6. [this can be used to return the current class instance from the method.](https://www.javatpoint.com/this6)

# Super Keyword in Java

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

## Usage of Java super Keyword

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

As we know well, the default constructor is provided by the compiler automatically if there is no constructor. But, it also adds super() as the first statement.

# Instance initializer block

| **Instance Initializer block** is used to initialize the instance data member. It runs each time an object of the class is created. |
| --- |
| The initialization of the instance variable can be done directly but there can be performed extra operations while initializing the instance variable in the instance initializer block. |

## Rules for instance initializer block :

| There are mainly three rules for the instance initializer block. They are as follows: |
| --- |

1. The instance initializer block is created when an instance of the class is created.
2. The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
3. The instance initializer block comes in the order in which they appear.

# Final Keyword In Java

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

1. variable
2. method
3. class

A final variable that is not initialized at the time of declaration is known as blank final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

# Java instanceof

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

The instanceof in java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

# Java Package

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

## Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collisions.

Note:- if you import a package,subpackage will not be imported.

# Access Modifiers in Java

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

**The access modifiers in Java specify the accessibility or scope of a field, method, constructor, or class.** We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| --- | --- | --- | --- | --- |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

### Java Access Modifiers with Method Overriding

If you are overriding any method, the overridden method (i.e. declared in subclass) must not be more restrictive.

# Object class in Java

The **Object class** is the parent class of all the classes in java by default.

**The Object class is beneficial if you want to refer to any object whose type you don't know.** Notice that parent class reference variables can refer to the child class object, known as upcasting.

| public final void notify() | wakes up a single thread, waiting on this object's monitor. |
| --- | --- |
| public final void notifyAll() | wakes up all the threads, waiting on this object's monitor. |
| public final void wait(long timeout)throws InterruptedException | causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method). |

# 

# Object Cloning in Java

Object **cloning** is a way to create an exact copy of an object. The clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement a Cloneable interface, the clone() method generates a CloneNotSupportedException.

### Why use the clone() method ?

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed, which is why we use object cloning.

# Wrapper classes in Java

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

**The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.**

Java Strictfp Keyword

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable.

# Java String

In [Java](https://www.javatpoint.com/java-tutorial), string is basically an object that represents a sequence of char values.

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.

### 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

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.

### Why does Java use the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

### Why is the 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 valueOf() Method

The String class valueOf() method converts given types such as int, long, float, double, boolean, char and char array into String.

# Java StringBuffer Class

Java StringBuffer class is used to create mutable (modifiable) String objects. The StringBuffer class in Java is the same as String class except it is mutable i.e. it can be changed.

### StringBuffer capacity() Method

The capacity() method of the StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

# Java StringBuilder Class

Java StringBuilder class is used to create mutable (modifiable) String. The Java StringBuilder class is the same as the StringBuffer class except that it is non-synchronized.

# 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 a new instance. | StringBuffer is fast and consumes less memory when we concatenate the strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by the equals() method. | The StringBuffer class doesn't override the equals() method of the Object class. |
| 4) | String class is slower while performing concatenation operation. | The StringBuffer class is faster while performing concatenation operations. |
| 5) | String class uses String constant pool. | StringBuffer uses Heap memory |

# Difference between StringBuffer and StringBuilder

| **S** | **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 |

# StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a String into tokens. It is a simple way to break a String. It is a legacy Java class of Java.

#### Note: The StringTokenizer class is deprecated now. It is recommended to use the split() method of the String class or the Pattern class that belongs to the java.util.regex package.

# Java Regex

The **Java Regex** or Regular Expression is an API to *define a pattern for searching or manipulating strings*.

It is widely used to define the constraint on strings such as password and email validation.

# Exception Handling in Java

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

## What is Exception in Java

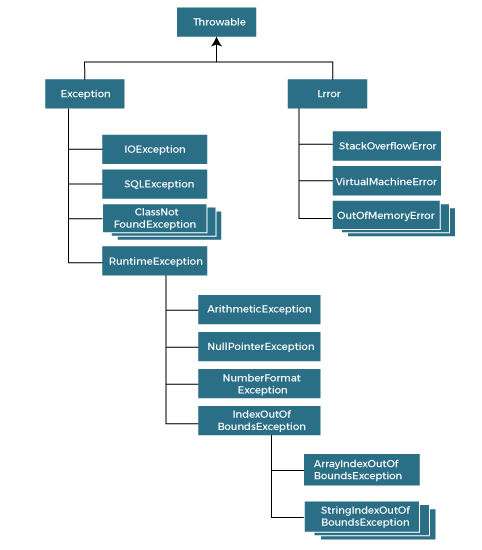
In Java, **an exception is an event that disrupts the normal flow of the program.** It is an object which is thrown at runtime.

### Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**.

## Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error. The hierarchy of Java Exception classes is given below:



### Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

## Diff between Checked and Unchecked Exceptions

### 1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable. Some examples of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

| **Key** | **Description** |
| --- | --- |
| try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by a finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signatures. |

## Java try block

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

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

## Java catch block

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

### Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

# Java finally block

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

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

#### The finally block will not be executed if the program exits (either by calling System.exit() or by causing a fatal error that causes the process to abort). There will be only one final block

## Java throw keyword

The Java throw keyword is used to throw an exception explicitly.

#### Note: Every subclass of Error and RuntimeException is an unchecked exception in Java. A checked exception is everything else under the Throwable class.

# Java Exception Propagation

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method. If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.

# Java throws keyword

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

### Which exception should be declared?

**Ans:** Checked exception only, because:

* **unchecked exception:** under our control so we can correct our code.
* **error:** beyond our control. For example, we are unable to do anything if VirtualMachineError or StackOverflowError occurs.

| **Sr** | **Basis of Differences** | **throw** | **throws** |
| --- | --- | --- | --- |
| 1. | Definition | Java throw keyword is used to 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 exceptions 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 keywords that can be thrown by the method. For example, main() throws IOException, SQLException. |

| **Sr.** | **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 an 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, the final variable becomes constant and cannot be modified.  (2) final method cannot be overridden by subclass.  (3) final class cannot be inherited. | (1) finally block runs the important code even if an 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 the block is executed as soon as the try-catch block is executed. | the finalize method is executed just before the object is destroyed. |

# Exception Handling with Method Overriding in Java

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

Some of the rules are listed below:

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

### Difference between nested class and inner class in Java

An inner class is a part of a nested class. Non-static nested classes are known as inner classes.

### Types of Nested classes

There are two types of nested classes: non-static and static nested classes. The non-static nested classes are also known as inner classes.

* Non-static nested class (inner class)
  1. Member inner class
  2. Anonymous inner class
  3. Local inner class
* Static nested class

| **Type** | **Description** |
| --- | --- |
| [Member Inner Class](https://www.javatpoint.com/member-inner-class) | A class created within class and outside methods. |
| [Anonymous Inner Class](https://www.javatpoint.com/anonymous-inner-class) | A class created for implementing an interface or extending class. The java compiler decides its name. |
| [Local Inner Class](https://www.javatpoint.com/local-inner-class) | A class was created within the method. |
| [Static Nested Class](https://www.javatpoint.com/static-nested-class) | A static class was created within the class. |
| [Nested Interface](https://www.javatpoint.com/nested-interface) | An interface created within class or interface. |