**CORE JAVA**

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

**JVM**

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.

**JRE**

JRE is an acronym for Java Runtime Environment. 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.

**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.

**List of Java Keywords**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

16. **final**: Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.

17. **finally**: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.

18. **float**: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.

19. **for**: Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.

20. **if**: Java if keyword tests the condition. It executes the if block if the condition is true.

21. **implements**: Java implements keyword is used to implement an interface.

22. **import**: Java import keyword makes classes and interfaces available and accessible to the current source code.

23. **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.

24. **interface**: Java interface keyword is used to declare an interface. It can have only abstract methods.

25. l**ong**: Java long keyword is used to declare a variable that can hold a 64-bit integer.

26. **new:** Java new keyword is used to create new objects.

27. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.

28. **package:** Java package keyword is used to declare a Java package that includes the classes.

29. **private:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.

30. **protected:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.

31. **public:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.

32. **return:** Java return keyword is used to return from a method when its execution is complete.

33. **short:** Java short keyword is used to declare a variable that can hold a 16-bit integer.

34. **static:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.

35. **super**: Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.

36. **switch:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.

37. **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.

38. t**his**: Java this keyword can be used to refer the current object in a method or constructor.

39. **throw:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.

40. **throws:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.

41. **try:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.

42. **void:** Java void keyword is used to specify that a method does not have a return value.

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

**Variable:**

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location.

Ex: int data=50;//Here data is variable

**1) Local Variable**

A variable declared inside the body of the method is called local variable.

**2) Instance Variable**

A variable declared inside the class but outside the body of the method, is called an instance variable.

**3) Static variable**

A variable that is declared as static is called a static variable.

**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:

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

**Methods in Java**

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.

Syntax- public static int methodName(int a, intb){

//body

}

**Access Modifier**

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

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

**Java OOPs Concepts**

The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

**Object**

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

**Class**

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

A class can also be defined as a blueprint from which you can create an individual object

**Inheritance**

When one object acquires all the properties and behaviours of a parent object, it is known as inheritance.

## Types of Inheritance

**Single Inheritance:**When a class inherits another class, it is known as a single inheritance.

## Multilevel Inheritance: When there is a chain of inheritance, it is known as multilevel inheritance.

## Hierarchical Inheritance: When two or more classes inherits a single class, it is known as hierarchical inheritance

**Multiple Inheritance:** When one class inherits multiple classes, it is known as multiple inheritance.

### **Hybrid Inheritance:** Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance.

**Polymorphism**

If one task is performed in different ways, it is known as polymorphism.

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

# Method Overloading in Java

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

# Method Overriding in Java

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

**Abstraction**

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

There are two ways to achieve abstraction in java

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

### **Abstract class in Java**

A class which is declared as abstract is known as an **abstract class.** It can have abstract and non-abstract methods.

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

The interface in Java is a mechanism to achieve *abstraction.* There can be only abstract methods in the Java interface, not method body.

**Encapsulation**

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

# **Association**-It is a relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.

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

**Constructors in Java**

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created.

There are two rules defined for the constructor.

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

## Types of Java constructors:

## Java Default Constructor

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

### Java Parameterized Constructor

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

**Collections**

# The Java **collections** framework provides a set of interfaces and classes to implement various data structures and algorithms.

# **Java ArrayList**

Java **ArrayList** class uses a dynamic [array](https://www.javatpoint.com/array-in-java) for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of the List interface here. The ArrayList maintains the insertion order internally.

# **Java LinkedList class**

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure.

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.

# **Java HashSet**

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.

HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.

# **Java LinkedHashSet**

Java LinkedHashSet class is a Hashtable and Linked list implementation of the set interface. It inherits HashSet class and implements Set interface.

The important points about Java LinkedHashSet class are:

* Java LinkedHashSet class contains unique elements only like HashSet.
* Java LinkedHashSet class provides all optional set operation and permits null elements.
* Java LinkedHashSet class is non synchronized.
* Java LinkedHashSet class maintains insertion order.

# **Java TreeSet**

# Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quiet fast.
* Java TreeSet class doesn't allow null element.
* Java TreeSet class is non synchronized.
* Java TreeSet class maintains ascending order.

# **Java Queue Interface**

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

# **Java Map Interface**

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

# **Java HashMap**

Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.

HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key

# **Java LinkedHashMap class**

Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

### **Points to remember**

* Java LinkedHashMap contains values based on the key.
* Java LinkedHashMap contains unique elements.
* Java LinkedHashMap may have one null key and multiple null values.
* Java LinkedHashMap is non synchronized.
* Java LinkedHashMap maintains insertion order.

# **Java TreeMap class**

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

* Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* Java TreeMap contains only unique elements.
* Java TreeMap cannot have a null key but can have multiple null values.
* Java TreeMap is non synchronized.
* Java TreeMap maintains ascending order.

# **Java Vector**

**Vector** is like the dynamic array which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework.

# **Java Stack**

The **stack** is a linear data structure that is used to store the collection of objects. It is based on **Last-In-First-Out** (LIFO). [Java collection](https://www.javatpoint.com/collections-in-java) framework provides many interfaces and classes to store the collection of objects. One of them is the **Stack class** that provides different operations such as push, pop, search, etc.

## Java Iterator

An Iterator is an object that can be used to loop through collections, like [ArrayList](https://www.w3schools.com/java/java_arraylist.asp) and [HashSet](https://www.w3schools.com/java/java_hashset.asp). It is called an "iterator" because "iterating" is the technical term for looping.

# **Exception Handling**

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

### **Types of Java 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, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

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

## Java Threads

Threads allows a program to operate more efficiently by doing multiple things at the same time. Threads can be used to perform complicated tasks in the background without interrupting the main program.

There are two ways to create a thread.

It can be created by extending the Thread class and overriding its run() method.

## Running Threads

If the class extends the Thread class, the thread can be run by creating an instance of the class and call its start() method:

If the class implements the Runnable interface, the thread can be run by passing an instance of the class to a Thread object's constructor and then calling the thread's start() method.

**Arrays : array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.

**Java String Methods**

Java String length(): The Java String length() method tells the length of the string. It returns count of total number of characters present in the String.

Java String compareTo(): The Java String compareTo() method compares the given string with current string

Java String concat() **:** The Java String concat() method combines a specific string at the end of another string and ultimately returns a combined string. It is like appending another string

Java String IsEmpty() : This method checks whether the String contains anything or not. If the java String is Empty, it returns true else false.

Java String toLowerCase() : The java string toLowerCase() method converts all the characters of the String to lower case.

Java String toUpper() : The Java String toUpperCase() method converts all the characters of the String to upper case.

Java String replace(): The Java String replace() method returns a string, replacing all the old characters or CharSequence to new characters. There are 2 ways to replace methods in a Java String.

Java String equals() : The Java String equals() method compares the two given strings on the basis of content of the string i.e Java String representation. If all the characters are matched, it returns true else it will return false

Java String equalsIgnoreCase(): This method compares two string on the basis of content but it does not check the case like equals() method. In this method, if the characters match, it returns true else false.

Java String toCharArray(): This method converts the string into a character array i.e first it will calculate the length of the given Java String including spaces and then create an array of char type with the same content.