# **Java**

**What is Java?**

Java is a popular programming language.

Java is used to develop mobile apps, web apps, desktop apps, games and much more.

### **When** **to** **Use** **StringBuilder**:

* When you need to perform multiple modifications to a string.
* When you are working with large strings and performance is a concern.
* When you are constructing strings dynamically, such as in loops or recursive methods.

**Array list** and Vector

* **Dynamic Size**: Like Array List, Vector can dynamically resize itself when elements are added or removed.
* **Synchronized**: Vector is synchronized, meaning it is thread-safe and can be used safely in concurrent environments. This is the main difference between Array List and Vector.

**Array Deque**

Uses a resizable array to store elements. It is not thread-safe but provides better performance compared to linked list-based implementations.

* **Resizable Array**: Uses a dynamic array to store elements, allowing it to grow as needed.
* **Double-Ended Queue**: Supports insertion and removal of elements at both ends, making it a double-ended queue.
* **No Capacity Restrictions**: It has no fixed capacity, and it automatically resizes as elements are added or removed.
* **Not Thread-Safe**: Unlike Vector, Array Deque is not synchronized.
* **Null Elements Not Allowed**: Array Deque does not allow null elements.

**Vector**

* **Dynamic Size**: Like Array List, Vector can dynamically resize itself when elements are added or removed.
* **Synchronized**: Vector is synchronized, meaning it is thread-safe and can be used safely in concurrent environments. This is the main difference between Array List and Vector.
* **Indexed Access**: Elements in a Vector can be accessed via their index, providing fast random access (O(1) time complexity).
* **Order Maintenance**: It maintains the order of elements in which they are added.
* **Null Elements**: Allows the inclusion of null elements.

**Deque**

A Deque (pronounced "deck") is a data structure that stands for "double-ended queue." It is an extension of the queue concept that allows insertion and removal of elements at both ends (the front and the back).

* **Double-Ended Operations**: Supports insertion and removal of elements at both ends, making it suitable for use as both a stack and a queue.
* **No Fixed Size**: Deques typically have no fixed size and can grow dynamically as needed.
* **Order Maintenance**: Elements are maintained in the order they are added, but can be accessed, inserted, and removed from both ends.
* **No Null Elements**: Deques do not generally allow null elements, as null is used to indicate the absence of elements.

**Stack**

* **LIFO Order**: The last item added is the first one removed.
* **Push, Pop, Peek, Is Empty and Size**.

**Priority queue**

* **Priority-Based**: Each element in the queue has a priority. Elements with higher priority are processed before those with lower priority.
* **Dynamic Size**: The queue grows and shrinks as needed.
* **No Fixed Order**: The order of elements with the same priority is not guaranteed.
* **Efficient Retrieval**: Provides efficient retrieval of the highest priority element.
* You can also provide a custom comparator to define your own ordering.
* When iterating over the elements, the order may vary because Priority Queue doesn't provide a sorted iteration.

**Hashset**

* **Order**: No guaranteed order of elements.
* **Performance**: Offers constant time performance for basic operations (add, remove, contains, and size).
* **Internal Structure**: Uses a hash table for storage.

**LinkedHashSet**:

* **Order**: Maintains a doubly-linked list running through all its entries, thus maintaining insertion order.
* **Performance**: Slightly slower than HashSet due to the overhead of maintaining the linked list.
* **Internal Structure**: Extends HashSet with a linked list.

**TreeSet**:

* **Order**: Maintains elements in sorted order according to their natural ordering or by a specified comparator.
* **Performance**: O(log n) time complexity for basic operations.
* **Internal Structure**: Implements a Navigable Set using a Tree Map.

**SortedSet**

The SortedSet interface is a specialized Set that maintains its elements in ascending order according to their natural ordering or according to a specified comparator.

* **Sorted Order**: Elements are maintained in sorted order.
* **Range Operations**: Provides methods to return subsets of the set.
* **Additional Methods**: Includes methods like first(), last(), headSet(), tailSet(), and subSet().

**NavigableSet**

The NavigableSet interface extends SortedSet and provides additional navigation methods that allow for efficient retrieval of elements based on proximity to a given value.

* **Navigation Methods**: Additional methods for navigating the set like lower(), floor(), ceiling(), and higher().
* **Reverse Order**: Method descendingSet() returns a view of the set in reverse order.
* **Poll Methods**: pollFirst() and pollLast() methods to retrieve and remove the first or last element, respectively.

**Encapsulation:**

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users.

**What is a Thread?**

* **Definition**: A thread is a lightweight subprocess, smaller than a traditional process, that can execute independently within a program. Threads share the same memory space and resources of the process that spawned them.
* Threads can be used to perform complicated tasks in the background without interrupting the main program.
* **Concurrency**: Threads allow different parts of a program to execute concurrently. This means multiple tasks can run simultaneously, potentially speeding up execution and improving responsiveness.

**Types of Threads**:

* **User-level Threads**: Managed entirely by the application and not visible to the operating system. They are fast to create and manage but may be limited by the application's threading library.
* **Kernel-level Threads**: Managed by the operating system. They are more robust but heavier to create and manage.

### **Benefits of Using Threads**

* **Multitasking**: Threads enable a program to perform multiple tasks concurrently, making efficient use of CPU resources.
* **Responsiveness**: By using threads, a program can remain responsive to user input even when performing long-running tasks in the background.
* **Modularization**: Threads allow developers to separate different tasks or operations into separate threads, which can improve code organization and readability.

**e.printstackTrace() ?**

* **e**: This is the exception object. In the context of a catch block, e typically represents the caught exception.
* **printStackTrace()**: This method is part of the Throwable class (which Exception extends). It prints the stack trace to the standard error stream (typically the console).

### **Non-Primitive Data Types**

Non-primitive data types (reference types) in Java are more complex and can store data of multiple values or objects. Unlike primitive data types, non-primitive types are created by the programmer and include classes, interfaces, arrays, and enums. Their memory size is not fixed and depends on the data they store.

**Primitive Data Types**

Primitive data types are the most basic data types available in Java. They are predefined by the language and named by a reserved keyword. There are eight primitive data types in Java:

**Java string methods?**

* Length
* Concatenation
* Character Access
* Substring
* Index of Substring
* Comparison
* Case Conversion
* Trimming Whitespace
* Splitting Strings

**Types of math methods?**

* Absolute Value
* Maximum and Minimum
* Rounding
* Ceiling and Floor

**Loops?**

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

**For Loop?**

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop.

A for loop in Java is used to iterate over a range of values or a collection of elements. It's one of the most commonly used loops in Java, offering a concise way to iterate over arrays, collections, or any range of values.

**What is Array?**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

Multidimensional Arrays

A multidimensional array is an array of arrays.

Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.

To create a two-dimensional array, add each array within its own set of **square braces[].**

**Multi-Dimensional Array?**

A multi-dimensional array in Java is an array of arrays. It allows you to create a data structure with more than one level of nested arrays. The most common type of multi-dimensional array is the two-dimensional array, but Java supports arrays with more dimensions.

**Method Overloading?**

With**method overloading**, multiple methods can have the same name with different parameters

**Method Overriding?**

Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.

**Java Recursion?**

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

**Encapsulation?**

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

* declare class variables/attributes as private
* provide public **get** and **set** methods to access and update the value of a private variable

**Java Polymorphism**

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

**Abstract Classes and Methods**

Data abstraction is the process of hiding certain details and showing only essential information to the user.  
Abstraction can be achieved with either abstract classes or [interfaces](https://www.w3schools.com/java/java_interface.asp)

The abstract keyword is a non-access modifier, used for classes and methods:

* Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
* Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

**Enums?**

An Enum is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).

**ArrayList vs. LinkedList**

The LinkedList class is a collection which can contain many objects of the same type, just like the ArrayList.

The LinkedList class has all of the same methods as the ArrayList class because they both implement the List interface. This means that you can add items, change items, remove items and clear the list in the same way.

However, while the ArrayList class and the LinkedList class can be used in the same way, they are built very differently.

**How the ArrayList works?**

The ArrayList class has a regular array inside it. When an element is added, it is placed into the array. If the array is not big enough, a new, larger array is created to replace the old one and the old one is removed.

**How the LinkedList works?**

The LinkedList stores its items in "containers." The list has a link to the first container and each container has a link to the next container in the list. To add an element to the list, the element is placed into a new container and that container is linked to one of the other containers in the list.

**HashMap**

A HashMap however, store items in "**key**/**value**" pairs, and you can access them by an index of another type (e.g. a String).

One object is used as a key (index) to another object (value). It can store different types: String keys and Integer values, or the same type, like: String keys and String values.

**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.

**Method overloading?**

Method overloading in Java is a feature that allows a class to have more than one method with the same name, as long as their parameter lists are different. This is also known as compile-time polymorphism. Method overloading can be achieved by changing the number of arguments or by changing the type of arguments.

**Method** **overriding?**

Method overriding in Java is a feature that allows a subclass to provide a specific implementation for a method that is already defined in its superclass. This is an example of runtime polymorphism or dynamic method dispatch. The method in the subclass should have the same name, return type, and parameter list as the method in the superclass.

Non-primitive data types are called **reference types** because they refer to objects.

The main difference between **primitive** and **non-primitive** data types are:

* Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
* Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.

**What is the difference between the collection and collections in java?**

* **Collection** is a interface representing a group of objects.
* It represents a group of objects, known as elements. It is the parent interface of more specific interfaces like List, Set, and Queue.
* **Collections** Collections is a utility class providing static methods to manipulate collections.

**What is the difference between the immutable and mutable?**

**Mutable**: Can be modified after creation (e.g., ArrayList in Java).

**Immutable**: Cannot be modified after creation (e.g., String in Java).

**Difference between checked and unchecked in java?**

**Checked** exceptions are exceptions that are checked at compile time. This means that the Java compiler requires the developer to handle these exceptions, either by using a try-catch block.

Enforced by the compiler to be handled

**Examples**:

* IOException
* SQLException
* FileNotFoundException

**Unchecked** exceptions are exceptions that are not checked at compile time. They are subclasses of RuntimeException, and the compiler does not require the developer to handle.

**Examples**:

* NullPointerException
* ArrayIndexOutOfBoundsException
* ArithmeticException

**What is the difference between string and stringbuiler in java?**

**String**:

* **Immutable.** Once a String object is created, it cannot be changed.
* **Thread-safe** because it's immutable. Multiple threads can safely share a String object without needing synchronization.
* Use String when you need a constant string value that won't change.

**StringBuilder**:

* **Mutable.** The content of a StringBuilder object can be modified without creating a new object. It provides methods like append(), insert(), delete(), and replace() that modify the object's internal value.
* **More efficient** for string manipulation because it modifies the existing object without creating new ones.
* **Not thread-safe.**
* Use StringBuilder when you need to build or modify a string.

**Thread and Asynchronous?**

**Threads**: Used for parallel execution within a program. They are suitable for situations where you need fine control over concurrent execution.

**Asynchronous Programming**: Allows tasks to run in the background while the main program continues to execute. CompletableFuture and ExecutorService provide higher-level abstractions for handling asynchronous tasks more efficiently.

**how to create a thread program in java in which way we can create?**

In Java, you can create a thread program by extending the Thread class or implementing the Runnable interface.

To create a thread program, follow these steps:

1. Define a class that extends Thread or implements Runnable.

2. Override the run() method with your thread code.

3. Create an instance of your class.

4. Call the start() method to begin execution of the thread.

**What all interfaces in collection API?**

The Collection API in Java provides several interfaces that can be used to manipulate collections. Here are some of the most commonly used interfaces in the Collection API:

1. **Collection**: This is the base interface for all collections. It provides methods for basic operations like add, remove, contains, etc.

2**. List**: This interface extends Collection and provides methods for accessing and manipulating elements in a specific order.

3. **Set**: This interface extends Collection and provides methods for storing unique elements.

4. **Map**: This interface provides methods for storing key-value pairs.

5. **Queue**: This interface extends Collection and provides methods for accessing and manipulating elements in a first-in-first-out (FIFO) order.

6. **Deque:** This interface extends Queue and provides methods for accessing and manipulating elements in a double-ended queue.

7. **Iterator:** This interface provides methods for iterating over elements in a collection.

8. **ListIterator:** This interface extends Iterator and provides methods for iterating over elements in a list.

9. **Spliterator:** This interface provides methods for splitting a collection into multiple parts.

Some other interfaces in the Collection API include:

- SortedSet

- NavigableSet

- SortedMap

- NavigableMap

- BlockingQueue

- TransferQueue

- BlockingDeque

These interfaces provide additional functionality and can be used to create more specialized collections.

**What are the internal flow in collection API?**

1. ArrayList
2. LinkedList
3. HashSet
4. TreeSet
5. HashMap