# Multithreading in Java

**Multithreading in** [**Java**](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously.

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

However, we use multithreading rather than multiprocessing because threads use a shared memory area. They don't allocate a separate memory area so saves memory, and context-switching between the threads takes less time than the process.

### Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

## Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

* Process-based Multitasking (Multiprocessing)
* Thread-based Multitasking (Multithreading)

### 1) Process-based Multitasking (Multiprocessing)

* Each process has an address in memory. In other words, each process allocates a separate memory area.
* The process is heavyweight.
* Cost of communication between the processes is high.

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

* Threads share the same address space.
* A thread is lightweight.
* Cost of communication between the threads is low.

**At Least one process is required for each thread.**

## Java Thread Methods

| **S** | **Modifier&Type** | **Method** | **Description** |
| --- | --- | --- | --- |
| 1) | void | [start()](https://www.javatpoint.com/java-thread-start-method) | It is used to start the execution of the thread. |
| 2) | void | [run()](https://www.javatpoint.com/java-thread-run-method) | It is used to perform action for a thread. |
| 3) | static void | [sleep()](https://www.javatpoint.com/java-thread-sleep-method) | It sleeps a thread for the specified amount of time. |
| 4) | static Thread | [currentThread()](https://www.javatpoint.com/java-thread-currentthread-method) | It returns a reference to the currently executing thread object. |
| 5) | void | [join()](https://www.javatpoint.com/java-thread-join-method) | It waits for a thread to die. |
| 6) | int | [getPriority()](https://www.javatpoint.com/java-thread-getpriority-method) | It returns the priority of the thread. |
| 7) | void | [setPriority()](https://www.javatpoint.com/java-thread-setpriority-method) | It changes the priority of the thread. |
| 8) | String | [getName()](https://www.javatpoint.com/java-thread-getname-method) | It returns the name of the thread. |
| 9) | void | [setName()](https://www.javatpoint.com/java-thread-setname-method) | It changes the name of the thread. |
| 10) | long | [getId()](https://www.javatpoint.com/java-thread-getid-method) | It returns the id of the thread. |
| 11) | boolean | [isAlive()](https://www.javatpoint.com/java-thread-isalive-method) | It tests if the thread is alive. |
| 12) | static void | [yield()](https://www.javatpoint.com/java-thread-yield-method) | It causes the currently executing thread object to pause and allow other threads to execute temporarily. |
| 13) | void | [suspend()](https://www.javatpoint.com/java-thread-suspend-method) | It is used to suspend the thread. |
| 14) | void | [resume()](https://www.javatpoint.com/java-thread-resume-method) | It is used to resume the suspended thread. |
| 15) | void | [stop()](https://www.javatpoint.com/java-thread-stop-method) | It is used to stop the thread. |
| 16) | void | [destroy()](https://www.javatpoint.com/java-thread-destroy-method) | It is used to destroy the thread group and all of its subgroups. |
| 17) | boolean | [isDaemon()](https://www.javatpoint.com/java-thread-isdaemon-method) | It tests if the thread is a daemon thread. |
| 18) | void | [setDaemon()](https://www.javatpoint.com/java-thread-setdaemon-method) | It marks the thread as a daemon or user thread. |
| 19) | void | [interrupt()](https://www.javatpoint.com/java-thread-interrupt-method) | It interrupts the thread. |
| 20) | boolean | [isinterrupted()](https://www.javatpoint.com/java-thread-isinterrupted-method) | It tests whether the thread has been interrupted. |
| 21) | static boolean | [interrupted()](https://www.javatpoint.com/java-thread-interrupted-method) | It tests whether the current thread has been interrupted. |
| 22) | static int | [activeCount()](https://www.javatpoint.com/java-thread-activecount-method) | It returns the number of active threads in the current thread's thread group. |
| 23) | void | [checkAccess()](https://www.javatpoint.com/java-thread-checkaccess-method) | It determines if the currently running thread has permission to modify the thread. |
| 24) | static boolean | [holdLock()](https://www.javatpoint.com/java-thread-holdlock-method) | It returns true if and only if the current thread holds the monitor lock on the specified object. |
| 25) | static void | [dumpStack()](https://www.javatpoint.com/java-thread-dumpstack-method) | It is used to print a stack trace of the current thread to the standard error stream. |
| 26) | StackTraceElement[] | [getStackTrace()](https://www.javatpoint.com/java-thread-getstacktrace-method) | It returns an array of stack trace elements representing the stack dump of the thread. |
| 27) | static int | [enumerate()](https://www.javatpoint.com/java-thread-enumerate-method) | It is used to copy every active thread's thread group and its subgroup into the specified array. |
| 28) | Thread.State | [getState()](https://www.javatpoint.com/java-thread-getstate-method) | It is used to return the state of the thread. |
| 29) | ThreadGroup | [getThreadGroup()](https://www.javatpoint.com/java-thread-getthreadgroup-method) | It is used to return the thread group to which this thread belongs |
| 30) | String | [toString()](https://www.javatpoint.com/java-thread-tostring-method) | It is used to return a string representation of this thread, including the thread's name, priority, and thread group. |
| 31) | void | [notify()](https://www.javatpoint.com/java-thread-notify-method) | It is used to give the notification for only one thread which is waiting for a particular object. |
| 32) | void | [notifyAll()](https://www.javatpoint.com/java-thread-notifyall-method) | It is used to give the notification to all waiting threads of a particular object. |
| 33) | void | [setContextClassLoader()](https://www.javatpoint.com/java-thread-setcontextclassloader-method) | It sets the context ClassLoader for the Thread. |
| 34) | ClassLoader | [getContextClassLoader()](https://www.javatpoint.com/java-thread-getcontextclassloader-method) | It returns the context ClassLoader for the thread. |
| 35) | static Thread.UncaughtExceptionHandler | [getDefaultUncaughtExceptionHandler()](https://www.javatpoint.com/java-thread-getdefaultuncaughtexceptionhandler-method) | It returns the default handler invoked when a thread abruptly terminates due to an uncaught exception. |
| 36) | static void | [setDefaultUncaughtExceptionHandler()](https://www.javatpoint.com/java-thread-setdefaultuncaughtexceptionhandler-method) | It sets the default handler invoked when a thread abruptly terminates due to an uncaught exception. |

# Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

## Explanation of Different Thread States

**New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

**Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

* **Runnable:** A thread that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread to the running state.
* **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

**Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

**Timed Waiting:** Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such a scenario, a timed waiting state is given to thread B. Thus, thread lies in the waiting state for a specific span of time, and not forever.

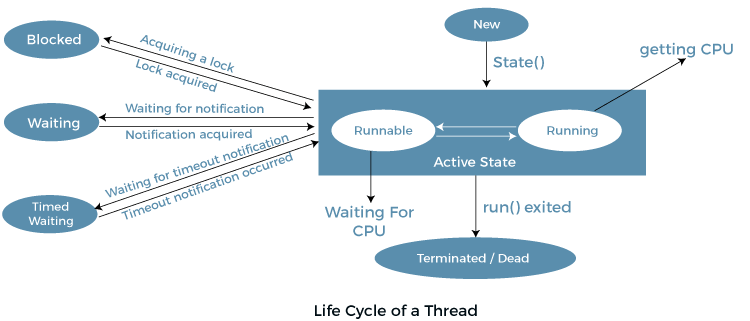
**Terminated:** A thread reaches the termination state because of the following reasons:

* When a thread has finished its job, then it exists or terminates normally.
* **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

AD

A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.

The following diagram shows the different states involved in the life cycle of a thread.



There are two ways to create a thread:

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

If you are not extending the Thread class, your class object would not be treated as a thread object. So you need to explicitly create the Thread class object. We are passing the object of your class that implements Runnable so that your class run() method may execute.

There are two factors for scheduling a thread i.e. **Priority** and **Time of arrival**.

**Priority:** Priority of each thread lies between 1 to 10. If a thread has a higher priority, it means that thread has got a better chance of getting picked up by the thread scheduler.

**Time of Arrival:** Suppose two threads of the same priority enter the runnable state, then priority cannot be the factor to pick a thread from these two threads. In such a case, **the arrival time** of the thread is considered by the thread scheduler. A thread that arrived first gets the preference over the other threads.

### Important Points to Remember About the Sleep() Method

Whenever the Thread.sleep() methods execute, it always halts the execution of the current thread.

Whenever another thread does interrupt while the current thread is already in sleep mode, then the InterruptedException is thrown.

# What if we call the Java run() method directly instead of the start() method?

* Each thread starts in a separate call stack.
* Invoking the run() method from the main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

# Daemon Thread in Java

**Daemon thread in Java** is a service provider thread that provides services to the user thread. Its life depends on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc

## Points to remember for Daemon Thread in Java

* It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.

### Why does JVM terminate the daemon thread if there is no user thread?

The sole purpose of the daemon thread is that it provides services to the user thread for background supporting tasks. If there is no user thread, why should the JVM keep running this thread? That is why JVM terminates the daemon thread if there is no user thread.

#### Note: If you want to make a user thread as Daemon, it must not be started otherwise it will throw IllegalThreadStateException.

# 

# ThreadGroup in Java

Java provides a convenient way **to group multiple threads in a single object.** In such a way, we can suspend, resume or interrupt a group of threads by a single method call.

# Java Shutdown Hook

A special construct that facilitates the developers to add some code that has to be run when the Java Virtual Machine (JVM) is shutting down is known as the **Java shutdown hook**. The Java shutdown hook comes in very handy in the cases where one needs to perform some special cleanup work when the JVM is shutting down.

# How to perform a single task by multiple threads in Java?

If you have to perform a single task by many threads, have only one run() method.

# Java Garbage Collection

It is a way to destroy unused objects.

### Advantage of Garbage Collection

* It makes java **memory efficient** because the garbage collector removes the unreferenced objects from heap memory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

## How can an object be unreferenced?

There are many ways:

* By nulling the reference
* By assigning a reference to another
* By anonymous object etc.

## finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class.

# Java Runtime class

**Java Runtime** class is used *to interact with the Java runtime environment*. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc. There is only one instance of java.lang.Runtime class available for one java application.

The **Runtime.getRuntime()** method returns the singleton instance of Runtime class.

## Important methods of Java Runtime class

| **N** | **Method** | **Description** |
| --- | --- | --- |
| 1 | public static Runtime getRuntime() | returns the instance of Runtime class. |
| 2 | public void exit(int status) | terminates the current virtual machine. |
| 3 | public void addShutdownHook(Thread hook) | registers new hook thread. |
| 4 | public Process exec(String command)throws IOException | executes given command in a separate process. |
| 5 | public int availableProcessors() | returns no. of available processors. |
| 6 | public long freeMemory() | returns amount of free memory in JVM. |
| 7 | public long totalMemory() | returns amount of total memory in JVM. |

# Synchronization in Java

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is a better option where we want to allow only one thread to access the shared resource.

### Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problems.

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

# Static Synchronization

If you make any static method as synchronized, the lock will be on the class, not on the object.

# Deadlock in Java

Deadlock can occur in a situation **when a thread is waiting for an object lock that is acquired by another thread and the second thread is waiting for an object lock that is acquired by the first thread.** Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

### wait() method

The wait() method causes the current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

By implementing Runnable interface is a better way to create a thread in java because when we create a thread by extending Thread class, all Thread class methods are inherited while we can perform the task with the one method (run method) only. It results in overhead inheritance.

| **wait()** | **sleep()** |
| --- | --- |
| The wait() method releases the lock. | The sleep() method doesn't release the lock. |
| It is a method of Object class | It is a method of Thread class |
| It is the non-static method | It is the static method |
| It should be notified by notify() or notifyAll() methods | After the specified amount of time, sleep is completed. |

# 

# Interrupting a Thread:

If any thread is in sleeping or waiting state (i.e. sleep() or wait() is invoked), calling the interrupt() method on the thread, breaks out the sleeping or waiting state and throws InterruptedException.

If the thread is not in the sleeping or waiting state, calling the interrupt() method performs normal behavior and doesn't interrupt the thread but sets the interrupt flag to true. Let's first see the methods provided by the Thread class for thread interruption.

# Java I/O Tutorial

**Java I/O** (Input and Output) is used *to process the input* and *produce the output*.

## Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

**1) System.out:** standard output stream

**2) System.in:** standard input stream

**3) System.err:** standard error stream

### OutputStream Hierarchy

Java output stream hierarchy

### InputStream Hierarchy

Java input stream hierarchy

# Java FileOutputStream Class

Java FileOutputStream is an output stream used for writing data to a [file](https://www.javatpoint.com/java-file-class).

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use [FileWriter](https://www.javatpoint.com/java-filterwriter-class) than FileOutputStream.

# Java FileInputStream Class

Java FileInputStream class obtains input bytes from a [file](https://www.javatpoint.com/java-file-class).

It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use [FileReader](https://www.javatpoint.com/java-filereader-class) class.

# Java BufferedOutputStream Class

Java BufferedOutputStream [class](https://www.javatpoint.com/object-and-class-in-java) is used for buffering an output stream. It internally uses a buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

# Java BufferedInputStream Class

Java BufferedInputStream [class](https://www.javatpoint.com/object-and-class-in-java) is used to read information from [streams](https://www.javatpoint.com/java-8-stream). It internally uses a buffer mechanism to make the performance fast.

The important points about BufferedInputStream are:

* When the bytes from the stream are skipped or read, the internal buffer automatically refills from the contained input stream, many bytes at a time.
* When a BufferedInputStream is created, an internal buffer [array](https://www.javatpoint.com/array-in-java) is created.

# Java SequenceInputStream Class

[Java](https://www.javatpoint.com/java-tutorial) SequenceInputStream [class](https://www.javatpoint.com/object-class) is used to read data from multiple [streams](https://www.javatpoint.com/java-8-stream). It reads data sequentially (one by one).

# Java ByteArrayOutputStream Class

Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte [array](https://www.javatpoint.com/array-in-java) which can be written to multiple streams later.

The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams.

The buffer of ByteArrayOutputStream automatically grows according to data.

# Java DataOutputStream Class

Java DataOutputStream [class](https://www.javatpoint.com/object-and-class-in-java) allows an application to write primitive [Java](https://www.javatpoint.com/java-tutorial) data types to the output stream in a machine-independent way.

Java applications generally use the data output stream to write data that can later be read by a data input stream.

# Java - ObjectStreamClass

ObjectStreamClass acts as a [Serialization](https://www.javatpoint.com/serialization-in-java) descriptor for the class. This [class](https://www.javatpoint.com/object-and-class-in-java) contains the name and serialVersionUID of the class.

# Java Console Class

The Java Console class is used to get input from the console. It provides methods to read texts and passwords.

If you read the password using Console class, it will not be displayed to the user.

# Java FileWriter Class

Java FileWriter class is used to write character-oriented data to a [file](https://www.javatpoint.com/java-file-class). It is a character-oriented class which is used for file handling in [java](https://www.javatpoint.com/java-tutorial).

Unlike FileOutputStream class, you don't need to convert string into byte [array](https://www.javatpoint.com/array-in-java) because it provides a method to write string directly.

# Java BufferedWriter Class

Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits [Writer](https://www.javatpoint.com/java-writer-class) class. The buffering characters are used for providing the efficient writing of single [arrays](https://www.javatpoint.com/array-in-java), characters, and [strings](https://www.javatpoint.com/java-string).

# Java BufferedReader Class

Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by the readLine() method. It makes the performance fast. It inherits [Reader](https://www.javatpoint.com/java-reader-class) [class](https://www.javatpoint.com/object-and-class-in-java).

# Java PrintStream Class

The PrintStream class provides methods to write data to another stream. The PrintStream [class](https://www.javatpoint.com/object-and-class-in-java) automatically flushes the data so there is no need to call flush() method. Moreover, its methods don't throw IOException.

# Java PushbackInputStream Class

Java PushbackInputStream [class](https://www.javatpoint.com/object-and-clas-in-java) overrides InputStream and provides extra functionality to another input stream. It can unread a byte which is already read and push back one byte.

# Java Scanner

Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.

# Serialization and Deserialization in Java

**Serialization in Java** is a mechanism of *writing the state of an object into a byte-stream*. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called *deserialization* where a byte-stream is converted into an object.

### Advantages of Java Serialization

It is mainly used to travel object's state on the network (that is known as marshaling).

**The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.**

## Java Transient Keyword

If you don't want to serialize any data member of a class, you can mark it as transient.

The transient keyword can be used with the data members of a class in order to avoid their serialization.

# 

# 

# Java Reflection API

**Java Reflection** is a *process of examining or modifying the run time behavior of a class at run time*.

### java.lang.Class class

The java.lang.Class class performs mainly two tasks:

* provides methods to get the metadata of a class at run time.
* provides methods to examine and change the run time behavior of a class.

### How to get the object of Class class?

There are 3 ways to get the instance of Class class. They are as follows:

* forName() method of Class class
* getClass() method of Object class
* the .class syntax

### 1) forName() method of Class class

* is used to load the class dynamically.
* returns the instance of Class class.
* It should be used if you know the fully qualified name of the class.This cannot be used for primitive types.

### 2) getClass() method of Object class

It returns the instance of Class class. It should be used if you know the type. Moreover, it can be used with primitives.

### 3) The .class syntax

If a type is available, but there is no instance, then it is possible to obtain a Class by appending ".class" to the name of the type. It can be used for primitive data types also.

### Pros and Cons of Reflection

**Pros:** Inspection of interfaces, classes, methods, and fields during runtime is possible using reflection, even without using their names during the compile time. It is also possible to call methods, instantiate a clear or to set the value of fields using reflection.

**Cons:** Using reflection, one can break the principles of encapsulation. It is possible to access the private methods and fields of a class using reflection. Thus, reflection may leak important data to the outside world, which is dangerous.

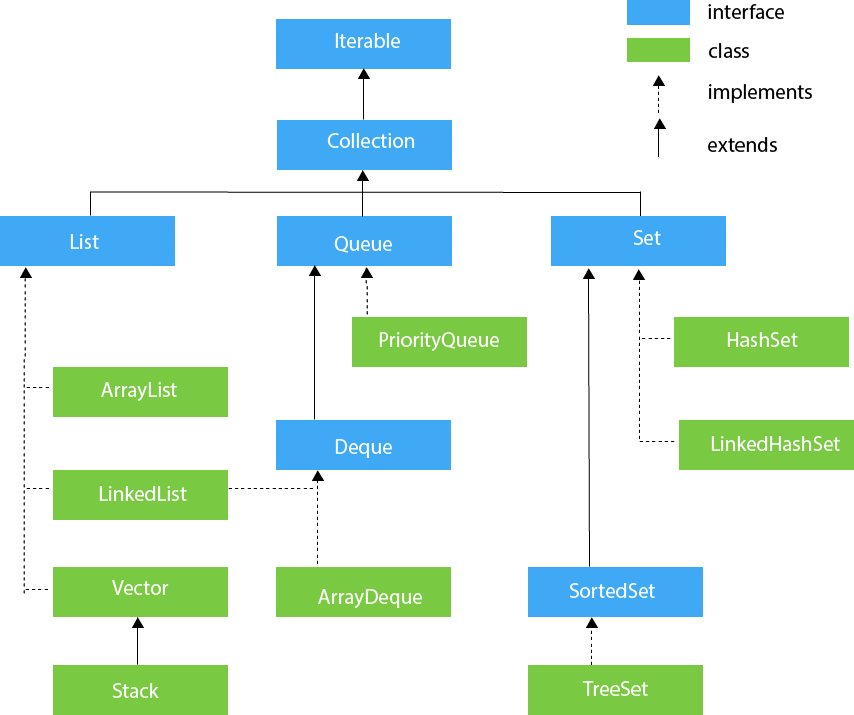
# Understanding javap tool

The **javap command** disassembles a class file. The javap command displays information about the fields, constructors and methods present in a class file.

# Collection in Java

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

A Collection represents a single unit of objects, i.e., a group.



### Iterator interface

| The Iterator interface provides the facility of iterating the elements in a forward direction only. |
| --- |

## ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed.The ArrayList maintains the insertion order internally.

The important points about the Java ArrayList class are:

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non [synchronized](https://www.javatpoint.com/synchronization-in-java).
* Java ArrayList allows random access because the array works on an index basis.
* In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
* We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases.

### Size and Capacity of an ArrayList

The size represents the total number of elements present in the array.

Capacity represents the total number of elements the array list can contain.

## LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

The important points about Java LinkedList are:

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.
* In Java LinkedList class, manipulation is fast because no shifting needs to occur.
* Java LinkedList class can be used as a list, stack or queue.

# Diff Between ArrayList and LinkedList

| **ArrayList** | **LinkedList** |
| --- | --- |
| ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |

## Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not part of the Collection framework.

## 

## Stack

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of the Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

## PriorityQueue

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

## 

## Deque Interface

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both sides. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

## ArrayDeque

The ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

The important points about ArrayDeque class are:

* Unlike Queue, we can add or remove elements from both sides.
* Null elements are not allowed in the ArrayDeque.
* ArrayDeque is not thread safe, in the absence of external synchronization.
* ArrayDeque has no capacity restrictions.
* ArrayDeque is faster than LinkedList and Stack.

## Set Interface

It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

## HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

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.
* HashSet is the best approach for search operations.
* The initial default capacity of HashSet is 16, and the load factor is 0.75.

# Java TreeSet class

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.**

**it is required to implement a Comparable interface. It is because the TreeSet maintains the sorting order, and for doing the sorting the comparison of different objects**

The important points about the Java TreeSet class are:

* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quite fast.
* Java TreeSet class doesn't allow null elements.
* Java TreeSet class is non synchronized.
* Java TreeSet class maintains ascending order.
* The TreeSet can only allow those generic types that are comparable. For example The Comparable interface is being implemented by the StringBuffer class.

## LinkedHashSet

The LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

## SortedSet Interface

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides additional methods that inhibit the natural ordering of the elements.

# 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.

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap.

A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

# Java HashMap

Java **HashMap** class implements the Map interface which allows us *to store key and value pairs*, where keys should be unique.

**If you try to insert the duplicate key, it will replace the value 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.

### Points to remember

* Java HashMap contains values based on the key.
* Java HashMap contains only unique keys.
* Java HashMap may have one null key and multiple null values.
* Java HashMap is non synchronized.
* Java HashMap maintains no order.
* the initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

### Difference between HashSet and HashMap

HashSet contains only values whereas HashMap contains an entry(key and value).

### Difference between ConcurrentHashMap and HashMap

* HashMap is non-Synchronized in nature i.e. HashMap is not Thread-safe whereas ConcurrentHashMap is Thread-safe in nature.
* HashMap performance is relatively high because it is non-synchronized in nature and any number of threads can perform simultaneously. But ConcurrentHashMap performance is low sometimes because sometimes Threads are required to wait on ConcurrentHashMap.
* While one thread is Iterating the HashMap object, if other thread try to add/modify the contents of Object then we will get Run-time exception saying **ConcurrentModificationException**.Whereas In ConcurrentHashMap we won't get any exception while performing any modification at the time of Iteration.

## What is Hashing

It is the process of converting an object into an integer value. The integer value helps in indexing and faster searches.

# 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.

### What is the difference between HashMap and TreeMap?

| **HashMap** | **TreeMap** |
| --- | --- |
| 1) HashMap can contain one null key. | TreeMap cannot contain any null key. |
| 2) HashMap maintains no order. | TreeMap maintains ascending order. |

# 

# Java Hashtable class

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

### Points to remember

* A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
* Java Hashtable class contains unique elements.
* Java Hashtable class doesn't allow null key or value.
* Java Hashtable class is synchronized.
* The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

# Difference between HashMap and Hashtable

| **HashMap** | **Hashtable** |
| --- | --- |
| 1. HashMap is **non synchronized**. It is not-thread safe and can't be shared between many threads without proper synchronization code. | Hashtable is **synchronized**. It is thread-safe and can be shared with many threads. |
| 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. |
| 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. |
| 4) HashMap is **fast**. | Hashtable is **slow**. |
| We can make the HashMap as synchronized by calling this code. Map m = Collections.synchronizedMap(hashMap); | Hashtable is internally synchronized and can't be unsynchronized. |
| 6) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. |
| 7) Iterators in HashMap are fail-fast. | Enumerator in Hashtable is **not fail-fast**. |
| 8) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. |

# Java Collections class

Java collection class is used exclusively with static methods that operate on or return collections. It inherits the Object class.

**Collections** class provides static methods for sorting the elements of a collection. If collection elements are of a Set type, we can use TreeSet. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

# Java Comparable interface

Java Comparable interface is used to order the objects of the user-defined class. This interface is found in the java.lang package and contains only one method named compareTo(Object). **It provides a single sorting sequence only, i.e., you can sort the elements on the basis of a single data member only**. For example, it may be rollno, name, age or anything else.

# Java Comparator interface

**Java Comparator interface** is used to order the objects of a user-defined class.

This interface is found in the java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).

It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example, rollno, name, age or anything else.

# Properties class in Java

The **properties** object contains key and value pairs both as a string. The java.util.Properties class is the subclass of Hashtable.

It can be used to get property value based on the property key. The Properties class provides methods to get data from the properties file and store data into the properties file. Moreover, it can be used to get the properties of a system.

### An Advantage of the properties file

**Recompilation is not required if the information is changed from a properties file:** If any information is changed from the properties file, you don't need to recompile the java class. It is used to store information which is to be changed frequently.

# Difference between ArrayList and Vector

| **ArrayList** | **Vector** |
| --- | --- |
| 1) ArrayList is **not synchronized**. | Vector is **synchronized**. |
| ArrayList **increments 50%** of current array size if the number of elements exceeds its capacity. | Vector **increments 100%** means double the array size if the total number of elements exceeds its capacity. |
| 3) ArrayList is **not a legacy** class. It is introduced in JDK 1.2. | Vector is a **legacy** class. |
| 4) ArrayList is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized, i.e., in a multithreading environment, it holds the other threads in runnable or non-runnable state until the current thread releases the lock of the object. |

# Java JDBC Tutorial

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

* JDBC-ODBC Bridge Driver,
* Native Driver,
* Network Protocol Driver, and
* Thin Driver

# Java Collectors

Collectors is a final class that extends Object class. It provides reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.

Like as: Collectors.toSet();