Array

**Java array** is an object which contains elements of a similar data type. In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Advantages

* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently. i.e, Multiple elements of Array can be sorted at the same time.
* **Random access:** We can get any data located at an index position. i.e, Using the index, we can access any element in O(1) time.
* It is helpful to store any type of data with a fixed size.

Disadvantages

* The size of the array should be known in advance.
* **Size Limit:** The array is a static data structure with a fixed size so, the size of the array cannot be modified further and hence no modification can be done during runtime.
* If the size of the declared array is more than the required size then, it can lead to memory wastage.

**Applications of Array Data Structure:**

* Arrays are used to implement data structures like a stack, queue, etc.
* Arrays are used in the implementation of various graph and tree data structures.
* Arrays are used for matrices and other mathematical implementations.
* Arrays are used in lookup tables in computers.
* Arrays can be used for CPU scheduling.
* Arrays are commonly used to store large amounts of data.
* Arrays can be used to dynamically allocate memory for storing data.
* Arrays are used in computer graphics to store and manipulate images, pixel data, and other visual elements.

## **Anonymous Array in Java**

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

1. **public** **static** **void** main(String args[]){
2. printArray(**new** **int**[]{10,22,44,66});//passing anonymous array to method
3. }}

## **Cloning an Array in Java**

Since, Java array implements the Cloneable interface, we can create the clone of the Java array. If we create the clone of a single-dimensional array, it creates the deep copy of the Java array. It means, it will copy the actual value. But, if we create the clone of a multidimensional array, it creates the shallow copy of the Java array which means it copies the references.

### What will happen if you do not initialize an Array?

The array will take default values depending upon the data type.

### Can you declare an array without assigning the size of an array?

No, we cannot declare an array without assigning size. If we declare an array without size, it will throw compile time error.

### Where is an Array stored in JVM memory?

An Array is an object in java. So, Array is stored in heap memory in Java Virtual Machine.

**Difference between Array and Object.**

* An object represents a thing with characteristics (called a property), whereas an array creates a list of data and stores it in a single variable.

**Can a Negative number be passed in Array size?**

No, a negative number cannot be passed as array size. If you pass a negative number in Array size then you will get the NegativeArraySizeException at run time.

**When will we get ArrayStoreException?**

* ArrayStoreException is a runtime exception.
* **For example**, you will get this exception at run time if you declare a String Array and then try to insert integer elements in the array.

### When will we get ArrayIndexOutOfBounds Exception?

ArrayIndexOutOfBoundsis a runtime exception that occurs when the program tries to access the invalid index of an array such as an Index higher than the size of the array or a negative index

**How can you get the index of an array element?**

* You can find the index of an element through a linear or binary search. A linear search is a function in which you loop through each and every element of an array until it finds the match of the desired element. When it finds the matching element, it returns the index. Therefore time complexity of the linear search is O(n). Linear search can be applied to sorted as well as an unsorted array.
* If the array is sorted, you can use a binary search that repeatedly splits the array in half until the median of the interval matches the desired element and returns the index. Therefore time complexity of the binary search is O(log n)

Linked list

A linear data structure used to store the elements in contiguous locations is called a Linked List in Java. It has addresses and pointers that are used to link the elements, and each element in the linked list consists of two parts, namely the data part and the address part.

Multithreading

**Multithreading in**[**Java**](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously. It is a Java feature where one can subdivide the specific program into two or more threads to make the execution of the program fast and easy.

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)

Multiprocessing and multithreading, both are used to achieve multitasking.

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

Java Multithreading is mostly used in games, animation, etc.

Advantages:

* Multi threaded application utilizes cpu idle time.
* If an exception occurs in a single thread, it will not affect other threads as threads are independent.

Thread ?

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Java allows 2 ways to create thread

1 Extending thread class

**MultithreadingDemo** class overrides the run() method available in the Thread class as we extended Thread class. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Example:

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

{

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

{

**try** {

            // Displaying the thread that is running

            System.out.println(

                "Thread " + Thread.currentThread().getId()

                + " is running");

        }

**catch** (Exception e) {

            // Throwing an exception

            System.out.println("Exception is caught");

        }

}

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

{

**int** n = 8; // Number of threads

**for** (**int** i = 0; i < n; i++) {

            MultithreadingDemo object

                = **new** MultithreadingDemo();

            object.start();

        }

}

}

Output:

Thread 15 is running

Thread 14 is running

Thread 16 is running

Thread 12 is running

Thread 11 is running

Thread 13 is running

Thread 18 is running

Thread 17 is running

2 By Implementing Runnable interface

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

{

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

{

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

}

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

{

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

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

}

}

| **Sr. No.** | **Key** | **Thread** | **Runnable** |
| --- | --- | --- | --- |
| 1 | Basic | Thread is a class. It is used to create a thread | Runnable is a functional interface which is used to create a thread |
| 2 | Methods | It has multiple methods including start() and run()  We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. | It has only abstract method run() |
| 3 |  | Each thread creates a unique object and gets associated with it | Multiple threads share the same objects. |
| 4 | Memory | More memory required | Less memory required |
| 5 | Limitation | If we extend the Thread class, our class cannot extend any other class because Java doesn’t support multiple inheritance. | If a class is implementing the runnable interface then your class can extend another class. |

**Problem in scenario Multiple threads share the same objects( by runnable interface)**

If thread (A) is executing a synchronized method and thread (B) wants to execute another synchronized method of the same object, it will be blocked until thread (A) is finished. But if thread (B) has access to different objects of the same class, none of them will be blocked.

Problems may occur when two or more threads are accessing the same data concurrently, for example, one thread stores data into the shared object and the other threads reads data, there can be synchronization problem if the first thread has not finished storing the data before the second one goes to read it.

Lifecycle of a Thread

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
6. New

When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn’t started to execute.

2 Active

**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 the running state.  
  A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.
* **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.

**3 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.

For example, a thread (let's say its name is A) may want to print some data from the printer. However, at the same time, the other thread (let's say its name is B) is using the printer to print some data. Therefore, thread A has to wait for thread B to use the printer. Thus, thread A is in the blocked state. A thread in the blocked state is unable to perform any execution and thus never consume any cycle of the Central Processing Unit (CPU). Hence, we can say that thread A remains idle until the thread scheduler reactivates thread A, which is in the waiting or blocked 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 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. A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier.

**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.

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.



Thread properties

1 Name 2 id 3 priority

Name and id are generated byv jvm using constructor of thread class

We cannot modify id of thread but can be read by using getId() method from thread. Thread also has name and it can be modified by using setName. Name is used by programmer to identify thread.

Ex:

Class Alpha extends Thread{

Public void run(){

for(int i=0;i<10;i++){

sopln(getName()+” ”+i);

}

Class Threadtest{

Psvm(){

Alpha a1=new Alpha();

Alpha a2=new Alpha();

//a1.setName(“Suresh”); a2.setName(“Naresh);

A1.start();

A2.start();

}

Output:

Thread-0:1

Thread-1:1

Thread-0:2

Thread-1:2

………….

……

Thread priority

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler schedules the threads according to their priority

## How does Java Support Multithreading?

Java has great support for multithreaded applications. Java supports multithreading through **Thread** class. Java Thread allows us to create a lightweight process that executes some tasks. We can create multiple threads in our program and start them. Java runtime will take care of creating machine-level instructions and work with OS to execute them in parallel.