DEPARTMENT OF COMMERCE(CA)

JAVA PROGRAMMING AND HTML (SEMESTER-IV)

II-MCOM(CA) Sub Code-18MCC42C

UNIT-III

One dimensional array-creating an array-Strings-Multiple Inheritance-Creating Threads-Extending thread classes-Stopping and blocking a thread-Life cycle of a thread.

One Dimensional Array

One Dimensional Array Program in Java – In this article, we will detail in on all the different methods to describe the one-dimensional array program in Java with suitable examples & sample outputs.

One Dimensional Array in java is always used with only one subscript ([]). A one-dimensional array behaves likes a list of variables. You can access the variables of an array by using an index in square brackets preceded by the name of that array. Index value should be an integer.

One-dimensional array in Java programming is an array with a bunch of values having been declared with a single index.

```
One dimensional array elements are
10
20
30
```

Declaration of One-dimensional array

Before using the array, we must declare it. Like normal variables, we must provide data type of array and name of an array. Data type means which type of elements we want to store in Array. So, we must specify the data type of array according to our needs. We also need to specify the name of an array so that we can use it later by name.

Creating an Array

You can create an array by using the new operator with the following syntax −

Syntax

arrayRefVar = new dataType[arraySize];

The above statement does two things −

It creates an array using new dataType[arraySize].

It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below −

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively, you can create arrays as follows −

dataType[] arrayRefVar = {value0, value1, ..., valuek};

The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

Strings

The most direct way to create a string is to write −

String greeting = "Hello world!";

Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

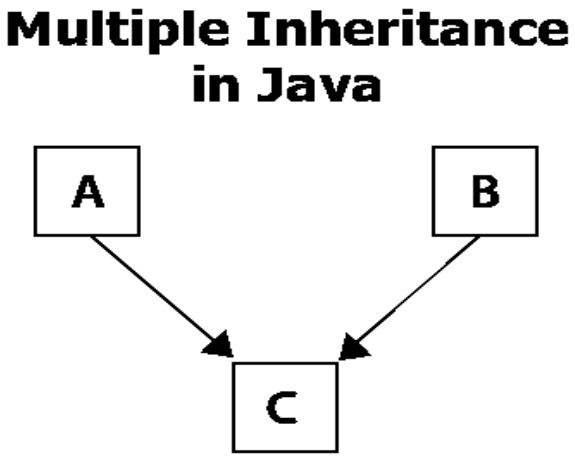
As with any other object, you can create String objects by using the new keyword and a constructor. The String class has 11 constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.

Multiple inheritance

Java does not support multiple inheritance. This means that a class cannot extend more than one class. Therefore, following is illegal

public class extends Animal, Mammal{}

However, a class can implement one or more interfaces, which has helped Java get rid of the impossibility of multiple inheritance. The extends keyword is used once, and the parent interfaces are declared in a comma-separated list. For example, if the Hockey interface extended both Sports and Event, it would be declared as −public interface Hockey extends Sports, Event



Creating threads

There are two ways to create a thread:

By extending Thread class

By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

Thread()

Thread(String name)

Thread(Runnable r)

Thread(Runnable r,String name)

Commonly used methods of Thread class:

Extending thread classes

One way to create a thread is to create a new class that extends Thread, and then to create an instance of that class. The extending class must override the run () method, which is the entry point for the new thread. It must also call start () to begin execution of the new thread.

Whenever we want to stop a thread from running state by calling stop () method of Thread class in Java. This method stops the execution of a running thread and removes it from the waiting threads pool and garbage collected. A thread will also move to the dead state automatically when it reaches the end of its method. The stop () method is deprecated in Java due to thread-safety issues.

Starting a thread:

When we create a thread by taking instance of a thread class, the thread will be in its new-born state. To move it to runnable state, we use a method named start (). When we invoke this method with a thread, java run-time schedules it to run by invoking it run () method. Now the thread will be in its running state. To start a thread, we use the following syntax:

```
MyThread t1 = new MyThread();

t1.start();
```

Stopping a thread:

Whenever we want to stop a thread from running further, we may do so by calling its stop () method. This causes a thread to stop immediately and move it to its dead state. It forces the thread to stop abruptly before its completion i.e., it causes premature death. To stop a thread, we use the following syntax:

```
t1.stop();
```

Blocking a Thread:

A thread can also be temporarily suspended or blocked from entering into the runnable and subsequently running state by using either of the following thread methods:

sleep(t) // blocked for 't' milliseconds

suspend () // blocked until resume () method is invoked

wait () // blocked until notify () is invoked

These methods cause the thread to go into the blocked (or not-runnable) state. The thread will return to the runnable state when the specified time is elapsed in the case of sleep (), the resume () method is invoked in the case of suspend (), and the notify () method is called in the case of wait ().

Life cycle of a Thread

A thread life cycle is divided into five different states, which a thread may go through in its lifetime. Each thread can be in one of the following five states. Let's understand each of these different states in the order in which they are mentioned below -:

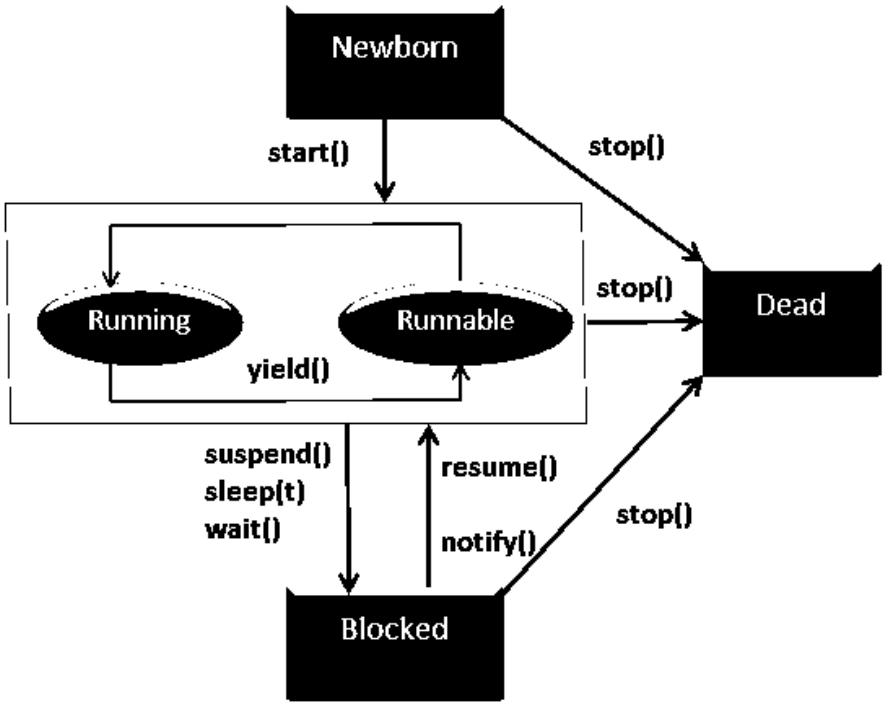


Fig: Life Cycle of Thread

New Born State:

A thread enters the new state when an object of Thread class is created but the start () method hasn't been called on it yet. In the new state, a thread is not considered alive as it's not a thread of execution. Once the start () method is called on the thread, it leaves the new

state and enters the next state but once it leaves the new state, it's impossible for it to return back to the new state in its lifetime.

Runnable State:

A thread enters the runnable state when the start () method has been called on it. It means, that a thread is eligible to run, but it's not yet running, as the thread scheduler hasn't selected it to run. At one point of time, there could be multiple thread in a runnable state, it's always the choice of thread scheduler to decide on which thread to move to the next state from the runnable state. A thread in the runnable state is considered to be alive. A thread can return to the runnable state after coming back from a sleeping, waiting/blocked or running state.

Running State:

A thread enters the running state when the thread schedular has selected it to run (out of all the threads in a thread pool). In this state, a thread starts executing the run () method and it is alive and kicking. From the running state, a thread can enter into the waiting/blocked state, runnable or the final dead state.

Blocked or Sleeping State:

A thread enters a not runnable in three situations -:

- When a thread has called the wait () method on itself and it is waiting for the other thread to notify it or wake it up
- When the method sleep () is called on a thread, asking it to sleep for a duration.
- When a thread has called the join () method on another thread, which makes the first thread wait until another thread has finished its execution.

When a thread is waiting for an Input/Output resource to be free

When a thread finds itself in any of the above mentioned three states, such events push the thread into a blocking/waiting or sleeping mode and the thread is no longer eligible to run. In any of these states, the thread is still considered to be alive. When thread gets out of waiting, blocking or sleeping state, it re-enters into the runnable state.

Dead State:

This is the last state in a thread's lifetime. A thread enters the dead state after it has successfully completed executing the run() method. At this situation, it is considered to be not alive and hence if you try to call start() method on a dead thread, it raises Illegal Thread State Exception.

REFERENCE

1. The Complete Reference Java2- Patrick Naughton and Herbert Schildt , 3rd Edition TMH Publications, 2000.
2. Programming with java2- Xavier, SciTech Publications, 2000.
3. https://javatutoring.com/java-one-dimensional-array/
4. https://www.javatpoint.com/life cycle of thread

PREPARED BY Dr.N.SHANMUGAVADIVU