BASICS OF JAVA
OOPS CONCEPTS
STRING HANDLING
EXCEPTION HANDLING
JAVA MULTITHREADING
Introduction to Multithreading
Thread Class
Creating a thread
Joining threads
Sleeping Thread in Java
Naming Thread in Java
Thread Priority in Java
Daemon Thread in Java
Synchronization
Interthread Communication
Thread group
ADVANCED TOPICS
COLLECTION FRAMEWORK
JAVA GUI
REFLECTION API
RMI APPLICATION
INNER CLASS
WRAPPER CLASS
FILE HANDLING
LIST
SET
MAP
QUEUE & DEQUE
JDBC
LAYOUT MANAGERS

Java Thread Class

Thread class is the main class on which Java's Multithreading system is based. Thread class, along with its companion interface Runnable will be used to create and run threads for utilizing Multithreading feature of Java.

It provides constructors and methods to support multithreading. It extends object class and implements runnable interface.

Signature of Thread class

```
public class Thread extends Object implements Runnable
```

Thread Class Priority Constants

Field	Description
MAX_PRIORITY	It represents the maximum priority that a thread can have.
MIN_PRIORITY	It represents the minimum priority that a thread can have.
NORM_PRIORITY	It represents the default priority that a thread can have.

Constructors of Thread class

1. **Thread()**
2. **Thread**(String str)
3. **Thread**(Runnable r)
4. **Thread**(Runnable r, String str)
5. **Thread**(**ThreadGroup** group, **Runnable** target)
6. **Thread**(**ThreadGroup** group, **Runnable** target, String name)
7. **Thread**(**ThreadGroup** group, **Runnable** target, String name, long stackSize)
8. **Thread**(**ThreadGroup** group, **String** name)

Thread Class Methods

Thread class also defines many methods for managing threads. Some of them are,

Method	Description
setName()	to give thread a name
getName()	return thread's name
getPriority()	return thread's priority
isAlive()	checks if thread is still running or not
join()	Wait for a thread to end
run()	Entry point for a thread
sleep()	suspend thread for a specified time
start()	start a thread by calling run() method
activeCount()	Returns an estimate of the number of active threads in the current thread's thread group and its subgroups.
checkAccess()	Determines if the currently running thread has permission to modify this thread.
currentThread()	Returns a reference to the currently executing thread object.
dumpStack()	Prints a stack trace of the current thread to the standard error stream.
getId()	Returns the identifier of this Thread.
getState()	Returns the state of this thread.
getThreadGroup()	Returns the thread group to which this thread belongs.
interrupt()	Interrupts this thread.
interrupted()	Tests whether the current thread has been interrupted.
isAlive()	Tests if this thread is alive.
isDaemon()	Tests if this thread is a daemon thread.
isInterrupted()	Tests whether this thread has been interrupted.
setDaemon(boolean on)	Marks this thread as either a daemon thread or a user thread.
setPriority(int newPriority)	Changes the priority of this thread.
yield()	A hint to the scheduler that the current thread is willing to yield its current use of a processor.

Some Important points to Remember

1. When we extend Thread class, we cannot override **setName()** and **getName()** functions, because they are declared final in Thread class.
2. While using **sleep()**, always handle the exception it throws.

```
static void sleep(long millisecond) throws InterruptedException
```

Runnable Interface

It also used to create thread and should be used if you are only planning to override the `run()` method and no other Thread methods.

Signature

```
@FunctionalInterface
public interface Runnable
```

Runnable Interface Method

It provides only single method that must be implemented by the class.

Method	Description
run()	It runs the implemented thread.

Shutdown hook

In Java, Shutdown hook is used to clean-up all the resource, it means closing all the files, sending alerts etc. We can also save the state when the JVM shuts down. Shutdown hook mostly used when any code is to be executed before any JVM shuts down. Following are some of the reasons when the JVM shut down:

- Pressing ctrl+c on the command prompt
- When the System.exit(int) method is invoked.
- When user logoff or shutdown etc

addShutdownHook(Thread hook)

The addShutdownHook(Thread hook) method is used to register the thread with the virtual machine. This method is of Runtime class.

Example:

```
class Demo1 extends Thread
{
    public void run()
    {
    System.out.println("Shutdown hook task is Now completed...");
    }
}

public class ShutdownDemo1
{
    public static void main(String[] args)throws Exception
    {

        Runtime obj=Runtime.getRuntime();
        obj.addShutdownHook(new Demo1());
        System.out.println("Now main method is sleeping... For Exit press ctrl+C");
        try
        {
```

Output:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17763.973]
(c) 2018 Microsoft Corporation. All rights reserved.

D:\Studytonight>javac ShutdownDemo1.java

D:\Studytonight>java ShutdownDemo1
Now main method is sleeping... For Exit press ctrl+C
Shutdown hook task is Now completed...
```

OutOfMemory Exception

In Java, as we know that all objects are stored in the heap. The objects are created using the new keyword. The OutOfMemoryError occurs as follow:

Exception in thread "main" java.lang.OutOfMemoryError : Java heap

This error occurs when Java Virtual Machine is not able to allocate the object because it is out of memory and no memory can be available by the garbage collector.

The meaning of OutOfMemoryError is that something wrong is in the program. Many times the problem can be out of control when the third party library caches strings.

Basic program in which OutOfMemoryError can occur

Example:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class OutOfMemoryDemo1 {
    public static void main(String[] args) {
        List<Object> a= new ArrayList<>();
        Random obj1= new Random();
        while (1<10) {
            obj.add(obj1.nextInt());
        }
    }
}
```

Output:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17763.973]
(c) 2018 Microsoft Corporation. All rights reserved.

D:\Studytonight>javac OutOfMemoryDemo1.java

D:\Studytonight>java OutOfMemoryDemo1
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Unknown Source)
        at java.util.Arrays.copyOf(Unknown Source)
        at java.util.ArrayList.grow(Unknown Source)
        at java.util.ArrayList.ensureExplicitCapacity(Unknown Source)
        at java.util.ArrayList.ensureCapacityInternal(Unknown Source)
        at java.util.ArrayList.add(Unknown Source)
        at OutOfMemoryDemo1.main(OutOfMemoryDemo1.java:13)
```

Program in which OutOfMemoryError can occur because of low memory

Example:

```
public class OutOfMemoryErrorDemo2
{
    public static void main(String[] args)
    {
        Integer[] a = new Integer[1000000*1000*500];
        System.out.println("Done");
    }
}
```

Output:

```
D:\Studytonight>java OutOfMemoryErrorDemo2.java

D:\Studytonight>java OutOfMemoryErrorDemo2
Exception in thread "main" java.lang.NegativeArraySizeException
        at OutOfMemoryErrorDemo2.main(OutOfMemoryErrorDemo2.java:5)
```

Program in which OutOfMemoryError can occur, when Garbage Collector exceed the limit

Example:

```
import java.util.*;

public class OutOfMemoryErrorDemo3
public static void main(String args[]) throws Exception
    {
        Map a = new HashMap();
        a = System.getProperties();
        Random b = new Random();
            while (1<10) {
a.put(b.nextInt(), "RandomValue");
        }  }

}
```

Output:

```
C:\Windows\System32\cmd.exe
D:\Studytonight>javac OutOfMemoryErrorDemo3.java

D:\Studytonight>java OutOfMemoryErrorDemo3
Exception in thread "main" java.lang.NegativeArraySizeException
        at OutOfMemoryErrorDemo3.main(OutOfMemoryErrorDemo3.java:5)
```

← Prev
Next →