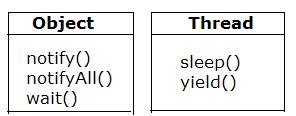
**INTRODUCTION TO JAVA THREADS**

Multithreading refers to two or more tasks executing concurrently within a single program. A thread is an independent path of execution within a program. Many threads can run concurrently within a program. Every thread in Java is created and controlled by the **java.lang.Thread class**. A Java program can have many threads, and these threads can run concurrently, either asynchronously or synchronously.

Multithreading has several advantages over Multiprocessing such as;

* Threads are lightweight compared to processes
* Threads share the same address space and therefore can share both data and code
* Context switching between threads is usually less expensive than between processes
* Cost of thread intercommunication is relatively low that that of process intercommunication
* Threads allow different tasks to be performed concurrently.

The following figure shows the methods that are members of the Object and Thread Class.



**THREAD CREATION**

There are two ways to create thread in java;

* Implement the Runnable interface (java.lang.Runnable)
* By Extending the Thread class (java.lang.Thread)

**IMPLEMENTING THE RUNNABLE INTERFACE**

**The Runnable Interface Signature**

public interface Runnable {

void run();

One way to create a thread in java is to implement the Runnable Interface and then instantiate an object of the class. We need to override the run() method into our class which is the only method that needs to be implemented. The run() method contains the logic of the thread.

**The procedure for creating threads based on the Runnable interface is as follows:**

1. A class implements the Runnable interface, providing the run() method that will be executed by the thread. An object of this class is a Runnable object.

2. An object of Thread class is created by passing a Runnable object as argument to the Thread constructor. The Thread object now has a Runnable object that implements the run() method.

3. The start() method is invoked on the Thread object created in the previous step. The start() method returns immediately after a thread has been spawned.

4. The thread ends when the run() method ends, either by normal completion or by throwing an uncaught exception.

Below is a program that illustrates instantiation and running of threads using the runnable interface instead of extending the Thread class. To start the thread you need to invoke the **start()** method on your object.

class RunnableThread implements Runnable {

Thread runner;

public RunnableThread() {

}

public RunnableThread(String threadName) {

runner = new Thread(this, threadName); // (1) Create a new thread.

System.out.println(runner.getName());

runner.start(); // (2) Start the thread.

}

public void run() {

//Display info about this particular thread

System.out.println(Thread.currentThread());

}

}

public class RunnableExample {

public static void main(String[] args) {

Thread thread1 = new Thread(new RunnableThread(), "thread1");

Thread thread2 = new Thread(new RunnableThread(), "thread2");

RunnableThread thread3 = new RunnableThread("thread3");

//Start the threads

thread1.start();

thread2.start();

try {

//delay for one second

Thread.currentThread().sleep(1000);

} catch (InterruptedException e) {

}

//Display info about the main thread

System.out.println(Thread.currentThread());

}

}

**Output**

thread3  
Thread[thread1,5,main] Thread[thread2,5,main] Thread[thread3,5,main] Thread[main,5,main]private

[**Download**](http://www.wideskills.com/sites/default/files/Zip_Files/runnable-thread.zip)Runnable Thread Program Example

This approach of creating a thread by implementing the Runnable Interface must be used whenever the class being used to instantiate the thread object is required to extend some other class.

**EXTENDING THREAD CLASS**

The procedure for creating threads based on extending the Thread is as follows:

1. A class extending the Thread class overrides the run() method from the Thread class to define the code executed by the thread.

2. This subclass may call a Thread constructor explicitly in its constructors to initialize the thread, using the super() call.

3. The start() method inherited from the Thread class is invoked on the object of the class to make the thread eligible for running.

Below is a program that illustrates instantiation and running of threads by extending the Thread class instead of implementing the Runnable interface. To start the thread you need to invoke the **start()**method on your object.

class XThread extends Thread {

XThread() {

}

XThread(String threadName) {

super(threadName); // Initialize thread.

System.out.println(this);

start();

}

public void run() {

//Display info about this particular thread

System.out.println(Thread.currentThread().getName());

}

}

public class ThreadExample {

public static void main(String[] args) {

Thread thread1 = new Thread(new XThread(), "thread1");

Thread thread2 = new Thread(new XThread(), "thread2");

// The below 2 threads are assigned default names

Thread thread3 = new XThread();

Thread thread4 = new XThread();

Thread thread5 = new XThread("thread5");

//Start the threads

thread1.start();

thread2.start();

thread3.start();

thread4.start();

try {

//The sleep() method is invoked on the main thread to cause a one second delay.

Thread.currentThread().sleep(1000);

} catch (InterruptedException e) {

}

//Display info about the main thread

System.out.println(Thread.currentThread());

}

}

**Output**

Thread[thread5,5,main] thread1  
thread5  
thread2  
Thread-3  
Thread-2  
Thread[main,5,main]

[**Download**](http://www.wideskills.com/sites/default/files/Zip_Files/java-thread-example.zip)**Java Thread Program Example**

When creating threads, there are two reasons why implementing the Runnable interface may be preferable to extending the Thread class:

* Extending the Thread class means that the subclass cannot extend any other class, whereas a class implementing the Runnable interface  
  has this option.
* A class might only be interested in being runnable, and therefore, inheriting the full overhead of the Thread class would be excessive.

An example of an anonymous class below shows how to create a thread and start it:

( new Thread() {

public void run() {

for(;;) System.out.println(“Stop the world!”);

}

}

).start();