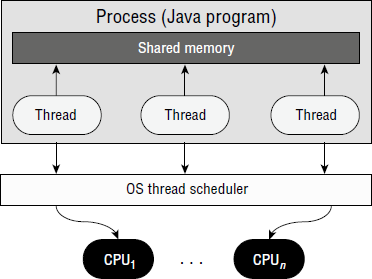
# Concurrency

**Fundamentals** A **multithreaded program** contains two or more parts that can run concurrently. Multithreading allows tasks waiting for other resources to give way to other processing requests.

A **thread** is the smallest unit of execution that can be scheduled by the operating system. A **process** is a group of associated threads executed in the same shared environment. A **single-threaded process** contains exactly one thread, whereas a **multithreaded process** supports more than one thread.

In a **shared environment**, threads in the same process share the same memory space and can communicate directly with one another.



***Figure 1****.* Process model.

The property of executing multiple threads and processes simultaneously is referred to as **concurrency**. Operating systems use a **thread scheduler** to determine which threads should be currently executing. When a thread’s allotted time is complete, but the thread has not finished processing, a context switch occurs. A **context switch** is the process of storing a thread’s current state and later restoring the state of the thread to continue execution.

When a Java application starts, its main() method is executed by the **main thread** – a special thread that is created by the Java VM to run your application. You can also create and start more threads that can execute parts of your application code in parallel with the main thread.

You can create a thread in two (2) ways:

* Extend the **Thread** class and override the **run()** method.
* Implement the **Runnable** interface.

Extending the Thread class:

**public class ThreadDemo extends Thread { public static void main(String[] args) {**

**ThreadDemo thread = new ThreadDemo(); thread.start(); System.out.println("Outside of a thread");**

**}**

**public void run(){ System.out.println("Running in a thread");**

**}**

**}**

# Output:

**Outside of a thread Running in a thread**

To start the Java thread, you will call its start() method. The run() method is what is executed by the thread after you call start(). The start() call will not wait until the run() method is done. The run() method will execute as if executed by a different CPU.

Implementing the **Runnable** interface:

**public class ThreadDemo implements Runnable { public static void main(String[] args) {**

**ThreadDemo obj = new ThreadDemo(); Thread thread = new Thread(obj); thread.start();**

**System.out.println("Outside of a thread");**

**}**

**public void run(){ System.out.println("Running in a thread");**

**}**

**}**

You can run the thread by passing an instance of the class to a Thread object's constructor and then calling the thread's start() method.

**Thread Methods** The Thread class consists of several methods that allow you to manage threads.

|  |  |
| --- | --- |
| **Method** | **Description** |
| start | Starts a thread by calling its run method |
| run | The entry point for the thread |
| setName | Sets a thread’s name |
| getName | Obtains a thread’s name |
| setPriority | Sets a thread’s priority |
| getPriority | Obtains a thread’s priority |
| isAlive | Determines if a thread is still running |
| join | Waits for a thread to terminate |
| sleep | Suspends a thread for a period of time |

A thread can interrupt or supersede another thread with a higher thread priority than the other thread. A **thread priority** is a numeric value associated with a thread that is taken into consideration by the thread scheduler when determining which threads should currently be executing.

The value of the priority level must be within the range **MIN\_PRIORITY** and **MAX\_PRIORITY**. These values are 1 and 10, respectively. To return a thread to default priority, specify **NORM\_PRIORITY**, which is 5.

The following sample program shows the use of setName(), getName(), and

setPriority().

**t1.start();**

**t2.start();**

**t3.start();**

**}**

**public void run(){ System.out.println(Thread.currentThread().getName()**

**+ " is running.");**

**}**

**}**

# Output:

**Two is running. Three is running. One is running.**

The **Thread.currentThread()** method returns a reference to the Thread instance executing currentThread() . This way, you can access the Java Thread object representing the thread executing a given block of code. A thread with higher priority does not necessarily mean that it will run faster or more often than others; it only has greater potential access to the CPU.

The following sample program shows the use of sleep(), join(), and

isAlive().

**public class ThreadDemo extends Thread {**

**public static void main(String[] args) { ThreadDemo t1 = new ThreadDemo(); ThreadDemo t2 = new ThreadDemo(); t1.start();**

**//starts 2nd thread after 2 sec or if it’s dead**

**try { t1.join(2000);**

**}**

**catch(InterruptedException e) { e.printStackTrace();**

**}**

**t2.start();**

**public class ThreadDemo extends Thread {**

**public static void main(String[] args) { ThreadDemo t1 = new ThreadDemo(); ThreadDemo t2 = new ThreadDemo(); ThreadDemo t3 = new ThreadDemo();**

**t1.setName("One");**

**t2.setName("Two"); t3.setName("Three"); t1.setPriority(4); t2.setPriority(Thread.MAX\_PRIORITY); t3.setPriority(8);**

**//waits for the threads to terminate try {**

**t1.join();**

**t2.join();**

**}**

**catch(InterruptedException e) { e.printStackTrace();**

**}**

**System.out.println("Thread-0 is alive: " + t1.isAlive());**

**System.out.println("Thread-1 is alive: " + t2.isAlive());**

**}**

**public void run(){ System.out.println(Thread.currentThread().getName()**

**+ " is running."); try {**

**Thread.sleep(3000); //suspends thread for 3 seconds**

**}**

**catch (InterruptedException e) { e.printStackTrace();**

**}**

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

**+ " has ended.");**

**}**

**}**

# Output:

**Thread-0 is running. Thread-1 is running. Thread-0 has ended. Thread-1 has ended. Thread-0 is alive: false Thread-1 is alive: false**

**References:**

Oracle Academy (n.d.). Citing sources. Retrieved from https://academy.oracle.com/ Oracle Docs (n.d.). Citing sources. Retrieved from

https://docs.oracle.com/javase/tutorial/java/javaOO/index.html

Schildt, H. (2022). *Java: The complete reference twelfth edition.* McGraw-Hill.