**Concurrency**

**Processes and Threads**

In concurrent programming, there are two basic units of execution: *processes* and *threads*. In the Java programming language, concurrent programming is mostly concerned with threads. However, processes are also important.

## Processes

A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space. Processes are often seen as synonymous with programs or applications

## Threads

Threads are sometimes called *lightweight processes*. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process.

Threads exist within a process — every process has at least one. Threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication.

# Defining and Starting a Thread

An application that creates an instance of Thread must provide the code that will run in that thread. There are two ways to do this:

* *Provide a Runnable object.* The [Runnable](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html) interface defines a single method, run, meant to contain the code executed in the thread. The Runnable object is passed to the Thread constructor, as in the[HelloRunnable](https://docs.oracle.com/javase/tutorial/essential/concurrency/examples/HelloRunnable.java) example:
* public class HelloRunnable implements Runnable {
* public void run() {
* System.out.println("Hello from a thread!");
* }
* public static void main(String args[]) {
* (new Thread(new HelloRunnable())).start();
* }
* }
* *Subclass Thread.* The Thread class itself implements Runnable, though its run method does nothing. An application can subclass Thread, providing its own implementation of run, as in the [HelloThread](https://docs.oracle.com/javase/tutorial/essential/concurrency/examples/HelloThread.java" \t "_blank)example:
* public class HelloThread extends Thread {
* public void run() {
* System.out.println("Hello from a thread!");
* }
* public static void main(String args[]) {
* (new HelloThread()).start();
* }
* }

Notice that both examples invoke Thread.start in order to start the new thread.

Which of these idioms should you use? The first idiom, which employs a Runnable object, is more general, because the Runnable object can subclass a class other than Thread. The second idiom is easier to use in simple applications, but is limited by the fact that your task class must be a descendant of Thread. This lesson focuses on the first approach, which separates the Runnable task from the Thread object that executes the task. Not only is this approach more flexible, but it is applicable to the high-level thread management APIs covered later.