# Creating and running a thread

Each instance of class Java.lang.Thread is associated with a thread of execution. To start execution of the thread, call Thread's instance method start(), which starts execution of the new thread by calling method run(). Method run() in class Thread does nothing, and it usually overridden.

Look to the right at static method main in class T, which extends Thread. The first thread of execution of this application begins with a call on method main. The first statement of main creates an instance of class T —the second thread— and calls its method start(), which is defined in superclass Thread. Method start() starts the new thread, calls method run in that new thread, and then returns. Thus, the for-loop on i is executed in the main thread and the for-loop on k is executed in the second thread.

```
public class T extends Thread {

   public static void main(String args[]) {
     (new T()).start();
     for (int i= 0; i < 20; i= i+1) {
       System.out.println("main thread, i = " + i);
     }
   }

   public @Override void run() {
     for (int k= 0; k < 20; k= k+1) {
       System.out.println("new thread, k = " + k);
     }
   }
}
```

We urge you to copy class T and put it in a DrJava or Eclipse program. Then execute method main. You will see the output of the two threads, interspersed. Execute again and the output of the two threads will again be interspersed, but perhaps in a different order. This is because each thread is given a time slice to execute, but the time needed to execute println is can change depending on what else is going on inside your computer.

**Interface Runnable**

Actually, interface Runnable and not class T first declares method run(). Interface Runnable looks like this:

```
public interface Runnable { void run(); }
```

and class Thread, declared as follows, imlements Runnable:

```
public class Thread implement Runnable {
   public new Thread() { … }
   public new Thread(Runnable t) { … }
   public void start() { … }
   public void run() {}
   …
}
```

Class Thread has many more methods. Some of them are given in the table to the right. We don't expect you to remember all these; we just want to give you a feel for Threads and their properties.

**Some more methods in class Thread**

| | |
|---|---|
| static currentThread() | |
| getId() | |
| getName() | setName(String) |
| isAlive() | |
| getPriority() | setPriority(int) |
| interrupt() | isInterrupted() |
| isDaemon() | setDeamon(boolean) |
| sleep(long) | yield() |

**Another way to create and start threads**

The difficulty with class T is that it extends Thread, so it can't extend any other class. An alternative way to create a new thread that doesn't have this drawback is given to the right in class R, which implements Runnable.

Look at R.main. After storing a new instance of R in r, it calls a Thread constructor with argument r and calls method start in the new instance. In class T above, the constructor uses the instance of Runnable in which the constructor resides. In class R, on the other hand, it uses the instance of Runnable give by its argument, r. That's the difference. So when start() is called, method run in an instance of R will be called.

We urge you, again, to pop class R into Eclipse or DrJava and run it several times to see what is printed.

```
public class R implements Runnable {

   public static void main(String args[]) {
     R r= new R();
     new Thread(r).start();
     for (int i= 0; i < 20; i= i+1) {
       System.out.println("main thread, i = " + i);
     }
   }

   public void run() {
     for (int k= 0; k < 20; k= k+1) {
       System.out.println("r-new thread, k = " + k);
     }
   }
}
```