

## 4.4 Java Threads

Threads are the fundamental model of program execution in a Java program, and the Java language and its API provide a rich set of features for the creation and management of threads. All Java programs comprise at least a single thread of control that begins execution in the program's `main()` method.

### 4.4.1 Creating Java Threads

There are two techniques for creating threads in a Java program. One approach is to create a new class that is derived from the `Thread` class and to override its `run()` method. However, the most common technique is to define a class that implements the `Runnable` interface. The `Runnable` interface is defined as follows:

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

When a class implements `Runnable`, it must define a `run()` method. The code implementing the `run()` method is what runs as a separate thread.

Figure 4.11 shows the Java version of a multithreaded program that determines the summation of a non-negative integer. The `Summation` class implements the `Runnable` interface. Thread creation is performed by creating an object instance of the `Thread` class and passing the constructor a `Runnable` object.

Creating a `Thread` object does not specifically create the new thread; rather, it is the `start()` method that actually creates the new thread. Calling the `start()` method for the new object does two things:

1. It allocates memory and initializes a new thread in the JVM.
2. It calls the `run()` method, making the thread eligible to be run by the JVM. (Note that we never call the `run()` method directly. Rather, we call the `start()` method, and it calls the `run()` method on our behalf.)

When the summation program runs, two threads are created by the JVM. The first is the parent thread, which starts execution in the `main()` method. The second thread is created when the `start()` method on the `Thread` object is invoked. This child thread begins execution in the `run()` method of the `Summation` class. After outputting the value of the summation, this thread terminates when it exits from its `run()` method.

Sharing of data between threads occurs easily in Win32 and Pthreads, as shared data are simply declared globally. As a pure object-oriented language, Java has no such notion of global data; if two or more threads are to share data in a Java program, the sharing occurs by passing references to the shared object to the appropriate threads. In the Java program shown in Figure 4.11, the main thread and the summation thread share the object instance of the `Sum` class. This shared object is referenced through the appropriate `getSum()` and `setSum()` methods. (You might wonder why we don't use a `java.lang.Integer` object rather than designing a new `Sum` class. The reason is that the `java.lang.Integer` class is **immutable**—that is, once its integer value is set, it cannot change.)

Recall that the parent threads in the Pthreads and Win32 libraries use `pthread_join()` and `WaitForSingleObject()` (respectively) to wait for the summation threads to finish before proceeding. The `join()` method in Java provides similar functionality. Notice that `join()` can throw an `InterruptedException`, which we choose to ignore for now. We discuss handling this exception in Chapter 6.

Java actually identifies two different types of threads: (1) daemon (pronounced “demon”) and (2) non-daemon threads. The fundamental difference between the two types is the simple rule that the JVM shuts down when all non-daemon threads have exited. Otherwise, the two thread types are identical. When the JVM starts up, it creates several internal daemon threads for performing tasks such as garbage collection. A daemon thread is created by invoking the `setDaemon()` method of the `Thread` class and passing the method the value `true`. For example, we could have set the thread in the program shown in Figure 4.11 as a daemon by adding the following line after creating—but before starting—the thread:

```
thrd.setDaemon(true);
```

```

class Sum
{
    private int sum;

    public int getSum() {
        return sum;
    }

    public void setSum(int sum) {
        this.sum = sum;
    }
}

class Summation implements Runnable
{
    private int upper;
    private Sum sumValue;

    public Summation(int upper, Sum sumValue) {
        this.upper = upper;
        this.sumValue = sumValue;
    }

    public void run() {
        int sum = 0;
        for (int i = 0; i <= upper; i++)
            sum += i;
        sumValue.setSum(sum);
    }
}

public class Driver
{
    public static void main(String[] args) {
        if (args.length > 0) {
            if (Integer.parseInt(args[0]) < 0)
                System.err.println(args[0] + " must be >= 0.");
            else {
                // create the object to be shared
                Sum sumObject = new Sum();
                int upper = Integer.parseInt(args[0]);
                Thread thrd = new Thread(new Summation(upper, sumObject));
                thrd.start();
                try {
                    thrd.join();
                    System.out.println
                        ("The sum of "+upper+" is "+sumObject.getSum());
                } catch (InterruptedException ie) { }
            }
        }
        else
            System.err.println("Usage: Summation <integer value>"); }
}

```

**Figure 4.11** Java program for the summation of a non-negative integer.