Providing a *run* method
First way: extending `class Thread`

```
public class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
}

public class TwoThreadsDemo {
    public static void main (String[] args) {
        new SimpleThread("Jamaica").start();
        new SimpleThread("Fiji").start();
    }
}
```

Providing a *run* method
Second way: implementing `interface` `Runnable`

```
public class Clock extends Applet implements Runnable {
    private Thread clockThread = null;
    public void start() {
        if (clockThread == null) {
            clockThread = new Thread(this, "Clock");
            clockThread.start();
        }
    }
    public void run() {
        Thread myThread = Thread.currentThread();
        while (clockThread == myThread) {
            repaint();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e){
            // the VM doesn't want us to sleep anymore,
            // so get back to work
            }
        }
    }
    public void paint(Graphics g) {
        // get the time and convert it to a date
        Calendar cal = Calendar.getInstance();
        Date date = cal.getTime();
        // format it and display it
        DateFormat dateFormatter =
DateFormat.getTimeInstance();
        g.drawString(dateFormatter.format(date), 5, 10);
    }
    // overrides Applet's stop method, not Thread's
    public void stop() {
        clockThread = null;
    }
}
```