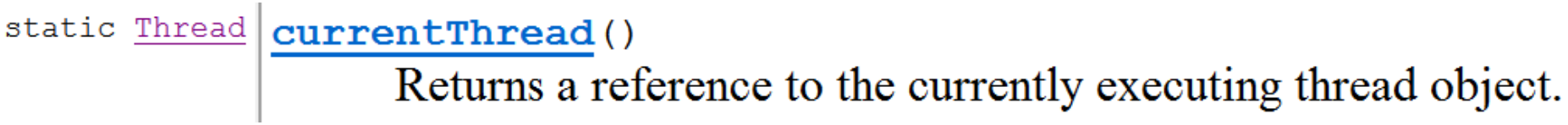
3 Implementierung nebenläufiger Prozesse

Threads (=Ausführungsstränge)
beschreiben Abläufe, die innerhalb eines Programms nebenläufig ausgeführt werden können



Implementiere eine Klasse Thread-Test, die eine Methode werIstDran() enthält, die den aktuell ausführenden Thread ausgibt.
Aufruf statischer Methoden: Thread.currentThread()

```
public class ThreadTest {
    public void werIstDran(){
        System.out.println("Thread: " + Thread.currentThread());
    }
}
```

Ausgabe: Thread: Thread[main, 5, main]

Erweitere werIstDran() um weitere Ausgaben (z.B. ID, Name, Zustand (State)). Prüfe, ob er unterbrochen ist, unterbrich ihn und prüfe nochmals. Was fällt dir auf?

```
public class ThreadTest {
    public void werIstDran(){
        System.out.println("Thread:" + Thread.currentThread());
        System.out.println("Thread-ID:" + Thread.currentThread().getId());
        System.out.println("Thread-Name:" +
                                    Thread.currentThread().getName());
        System.out.println("Thread-Priorität:" +
                                    Thread.currentThread().getPriority());
          System.out.println("Thread-State: " +
                                    Thread.currentThread().getState());
        System.out.println("is Interrupted:" +
                            Thread.currentThread().isInterrupted());
        System.out.println("Interrupt");
                            Thread.currentThread().interrupt();
        System.out.println("is Interrupted:" +
                            Thread.currentThread().isInterrupted());
    }
}
```

boolean	**isInterrupted**() Tests whether this thread has been interrupted.

Aufruf: Thread.currentThread().isInterrupted()

static boolean	**interrupted**() Tests whether the current thread has been interrupted.

Aufruf: Thread.interrupted()

Frage bei einem unterbrochenen Thread zweimal nacheinander zuerst mit Methode1, dann mit Methode2 ab, ob er wirklich unterbrochen ist. Was stellst du fest?

interrupt() setzt bei einem bestimmten Thread ein Flag, das anzeigt, dass der Thread beendet werden soll.
Dieser Status kann mit isInterrupted() abgefragt werden.
Mit interrupted() wird der Status des aktuellen Threads abgefragt, aber zusätzlich der Interrupt-Status gelöscht.

Erzeugen eines neuen Threads

1. Möglichkeit: Interface Runnable

Method Summary	
void	**run**() When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.

```
public class ZaehlRauf implements Runnable{
    public void run(){
        for (int i=0; i<1000;i++) {
            System.out.println(Thread.currentThread() + ": " + i);
        }
    }
}
```

```
public class ThreadTest{

    public void test(){
        ZaehlRauf rauf1 = new ZaehlRauf ();
        Thread rauf2 = new Thread(new ZaehlRauf ());
        rauf2.start();
        rauf1.run();
        ~~rauf2.start();~~

    }
}
```

Thread(Runnable target)

Allocates a new Thread object.

Beachte: Jeder Thread kann nur einmal gestartet werden!

2. Möglichkeit: Erweitere Klasse Thread

(Thread implements Runnable,
überschreibe also run-Methode)

```
public class ZaehlRunter extends Thread{
    public void run(){
        for (int i=1000; i>0;i--) {
            System.out.println(Thread.currentThread() + ": " + i);
        }
    }
}
```

```
public void test()){
        ZaehlRauf rauf1 = new ZaehlRauf();
        Thread rauf2 = new Thread(new ZaehlRauf());
        Thread runter1 = new ZaehlRunter();
        rauf2.start();
        runter1.start();
        rauf1.run();
    }

public class ZaehlRunter extends Thread{

    public ZaehlRunter(){
        start();
    }
...
```

Verändere die Klasse ZaehlRauf so, dass die Zählvariable i ein Attribut wird.

```
private int i=0;
public void run(){
        for (   ; i<=1000; i++) {…
```

Starte zwei Threads, die gemeinsam bis 1000 zählen.

```
 public void test(){
        ZaehlRauf rauf1 = new ZaehlRauf();
        Thread rauf2 = new Thread(rauf1, "zweiter");
        rauf2.start();
        rauf1.run();
    }
```

```
Thread[main,5,main]: 11
Thread[main,5,main]: 12
Thread[main,5,main]: 13
Thread[main,5,main]: 14
Thread[zweiter,5,main]: 14
Thread[zweiter,5,main]: 16
Thread[zweiter,5,main]: 17
Thread[zweiter,5,main]: 18
Thread[zweiter,5,main]: 19
```

```
Thread[zweiter,5,main]: 300
Thread[zweiter,5,main]: 301
Thread[main,5,main]: 98
Thread[main,5,main]: 303
Thread[main,5,main]: 304
Thread[main,5,main]: 305
Thread[main,5,main]: 306
Thread[main,5,main]: 307
Thread[main,5,main]: 308
Thread[main,5,main]: 309
Thread[zweiter,5,main]: 302
Thread[zweiter,5,main]: 311
Thread[zweiter,5,main]: 312
Thread[zweiter,5,main]: 313
Thread[zweiter,5,main]: 314
Thread[main,5,main]: 310
Thread[main,5,main]: 316
```