Dominik Gruntz
Daniel Kröni

Threads & Tools

Im Unterricht haben Sie gesehen, wie neue Threads gestartet werden. In diesem Arbeitsblatt werden wir ein Programm mit mehreren Threads zur Laufzeit im Eclipse Debugger etwas genauer unter die Lupe nehmen. Ziel dieses Arbeitsblattes ist, dass Sie ein besseres Gefühl für Threads bekommen und diese auch debuggen können.

Aufgaben:

1. Laden Sie das 01_LE_Threads.zip Archiv auf Ihren Computer, entpacken Sie es und importieren Sie es in Ihren Eclipse Workspace (File -> Import -> Gradle/Existing Gradle Project -> Project root directory -> Finish). Falls das Projekt nicht sauber kompiliert, müssen Sie evtl. in der Gradle/Tasks View den Task "ide/eclipse" ausführen und auf dem Eclipse Projekt danach die Kontextaction "Refresh" ausführen.

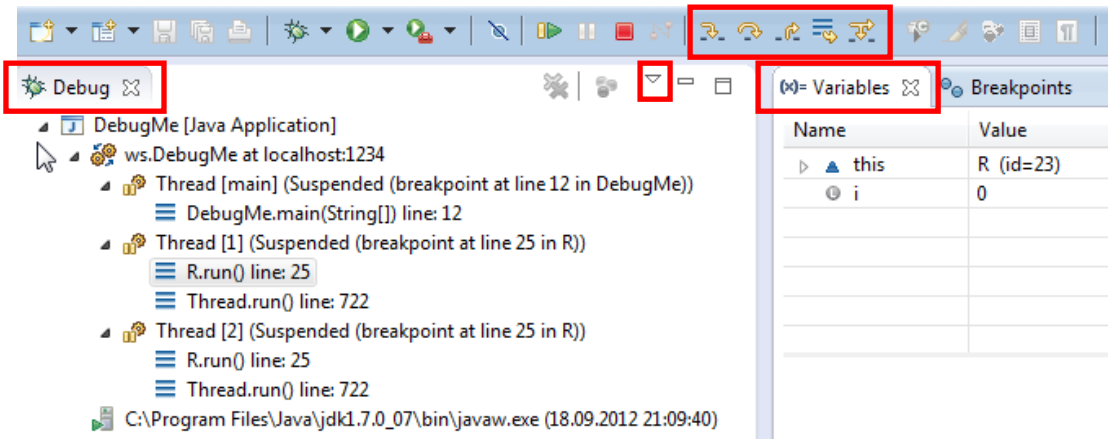
   Sie finden darin im Paket `worksheet` die folgende Klasse:

```
public class DebugMe {
    public static void main(String[] args) throws Exception {
        Thread t1 = new Thread(new R(System.out), "1");
        Thread t2 = new Thread(new R(System.err), "2");
        t1.start();
        t2.start();
        System.out.println("#CPUs: " +
                           Runtime.getRuntime().availableProcessors());
        System.out.println("main: done");
    }
}

class R implements Runnable {
    private PrintStream p;

    public R(PrintStream p) { this.p = p; }

    public void run() {
        for (int i = 0; i < 1000; i++) {
            p.println("Thread" + Thread.currentThread().getName() + ": " + i);
        }
    }
}
```

2. Führen Sie das Programm mehrmals hintereinander aus. Sehen Sie immer die gleichen Resultate? Wie viele Prozessoren meldet Ihnen die JVM Runtime?

3. Setzen Sie zwei Breakpoints:
   a. Auf der letzten Zeile der main Methode
   b. Innerhalb des For-Loops

   Starten Sie das Programm im Debug Mode: Rechts-Klick -> Debug As .. -> Java Application und wechseln Sie in die Debug Perspektive.



In der Debug View sehen Sie die drei von Ihnen erzeugten Threads (Thread 1, Thread 2 und der main-Thread) und deren Callstacks. Sie können nun jeden Thread mit den gelben Pfeil-Buttons einzeln kontrollieren. Steppen Sie den Thread 1 in die println Methode und beobachten Sie wie der Callstack wächst. In der Variables View können Sie die Variablen des aktuellen Stackframes und die des aktiven Objektes inspizieren. Primitive Werte (z.B. den int i) können Sie sogar modifizieren.
Steppen Sie nun so durch das Programm, dass folgender Output ausgegeben wird:

```
#CPUs: n
Thread2: 0
Thread1: 0
Thread2: 1
Thread1: 1
main: done
...
```

4. In der Debug View, klicken Sie auf das kleine Dreieck (oben rechts) -> Java -> Show System Threads. Nun sehen Sie alle aktiven Threads ihrer JVM. Wie viele System Threads sehen Sie?

5. Erstellen Sie ein Programm, das empirisch die maximale Anzahl Threads ermittelt welche gleichzeitig laufen können. Erzeugen und starten Sie dazu in einer Schleife Threads und geben Sie einen Zähler auf der Konsole aus. Die Threads können Sie z.B. mit der Anweisung `Thread.sleep(Long.MAX_VALUE);` beschäftigen.
   Wie viele Threads können Sie auf Ihrem System starten?

   Warnung:
   Entweder bricht ihr System nach einer gewissen Anzahl Threads mit einer Exception ab, oder aber das System wird instabil. Sichern sie daher alle offenen Dokumente bevor Sie diesen Test ausführen.

6. [Optional] Modifizieren Sie das erstellte Programm indem zwischen jedem neuen Thread den Sie erzeugen einen Thread.sleep(10) einbauen. Starten Sie dann JDK_HOME/bin/jconsole und verbinden Sie sich auf den Prozess. Vergleichen Sie den Wert den Ihr Programm ausgegeben hat mit dem Wert von JConsole den Sie unter "Threads" in der Grafik ablesen können.