# Übung 8: Semaphore in Java (eigene Implementierung)

Ziel der Übung:

Diese Übung dient dazu, eine eigene Implementierung einer Semaphore-Klasse in der Programmiersprache Java kennenzulernen.

Anschließend wird das Monitoring-Tool *JConsole* vorgestellt, mit dessen Hilfe u.a. der Speicherverbrauch von Java-Anwendungen analysiert werden kann.

Aufgabenstellung:

Es wird eine eigene Semaphor-Klasse implementiert (seit Java 1.5 ist auch bereits eine Standard-Semaphor-Implementierung verfügbar[1]) und mit einer eigenen Testklasse getestet.

Importieren Sie zunächst die zur Übung mitgelieferten Source-Files über die Eclipse-Import-Funktion (siehe *Übung 05, Seite 1*).

Die Klasse *SemaphorTestClass.java* legt eine Instanz der Klasse *MySemaphor* an und erzeugt anschließend Threads, die einen über die Semaphore geschützten kritischen Abschnitt (in dem nicht viel gemacht wird) betreten und wieder verlassen. Die Aktivitäten der Threads werden auf die Konsole protokolliert.

Folgende Aufgaben sind zu erledigen:

1. Versuchen Sie zunächst die Java-Klassen zu verstehen und überprüfen Sie, ob sie richtig übersetzt werden. Es müssen in Eclipse drei .class-Dateien erzeugt werden.

2. Bringen Sie das Programm zum Ablauf. Hierzu müssen Sie den Eclipse-Menüpunkt „Run“ benutzen.

   Prüfen Sie die Ausgaben des Programms auf die Konsole und vergleichen Sie die Ausgaben mit den im Anhang dieses Dokuments aufgelisteten Ausgaben! Was fällt Ihnen auf?

3. Erhöhen Sie die Anzahl der nebenläufigen Threads (Variable *maxThreads*) auf einen Wert von 200. Starten Sie anschließend das neu übersetzte Programm und betrachten Sie parallel dazu den Windows-Taskmanager.

   Suchen Sie im Taskmanager den Java-Prozess, in dem das Programm abläuft (Name = *javaw.exe*) und stellen Sie die Anzahl der Threads während des Ablaufs fest. Hierzu müssen Sie den Taskmanager entsprechend einstellen (Spalte *Threadanzahl* ergänzen über *Ansicht->Spalten auswählen*)

   Wie viele Threads benutzt der Prozess und wie verändert sich die Anzahl der Threads zur Laufzeit?

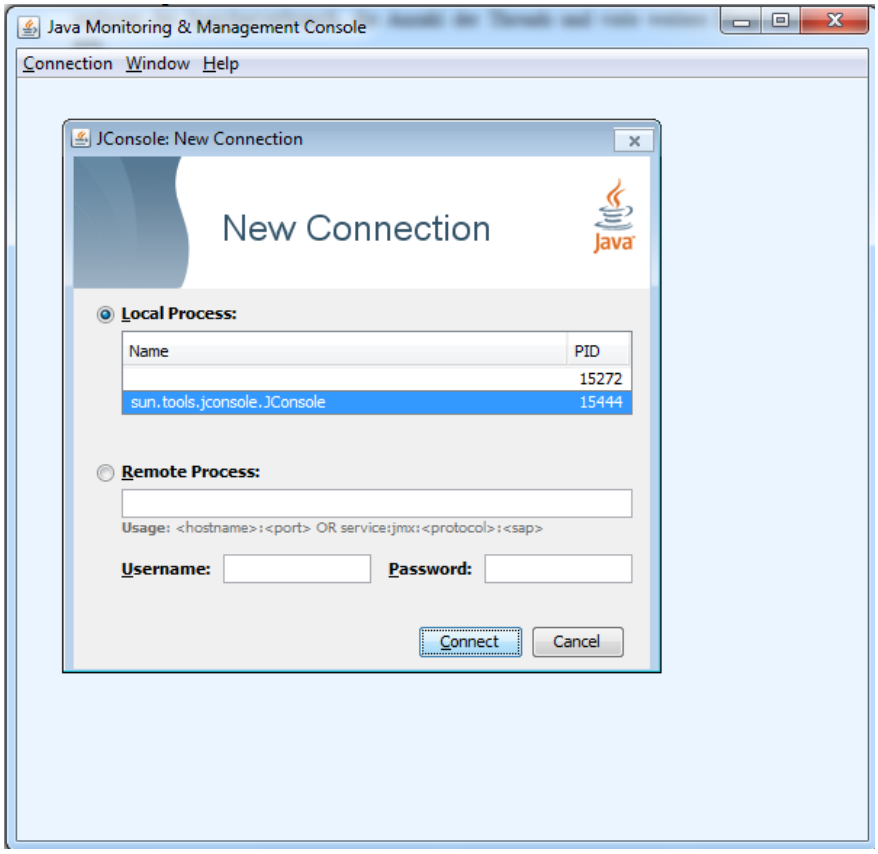
   Wiederholen Sie den Versuch mit einem Semaphorzähler von 2, 3 und 100 und variieren Sie die Anzahl der Threads bis 10.000. Betrachten Sie jeweils die Konsolausgaben!

4. Im bin-Verzeichnis der Java-Installation sind neben der Kompiler `javac.exe` auch noch viele weitere nützliche Tools vorhanden. Eines davon ist die JConsole, eine Swing-

[1] Siehe *Übung 9*

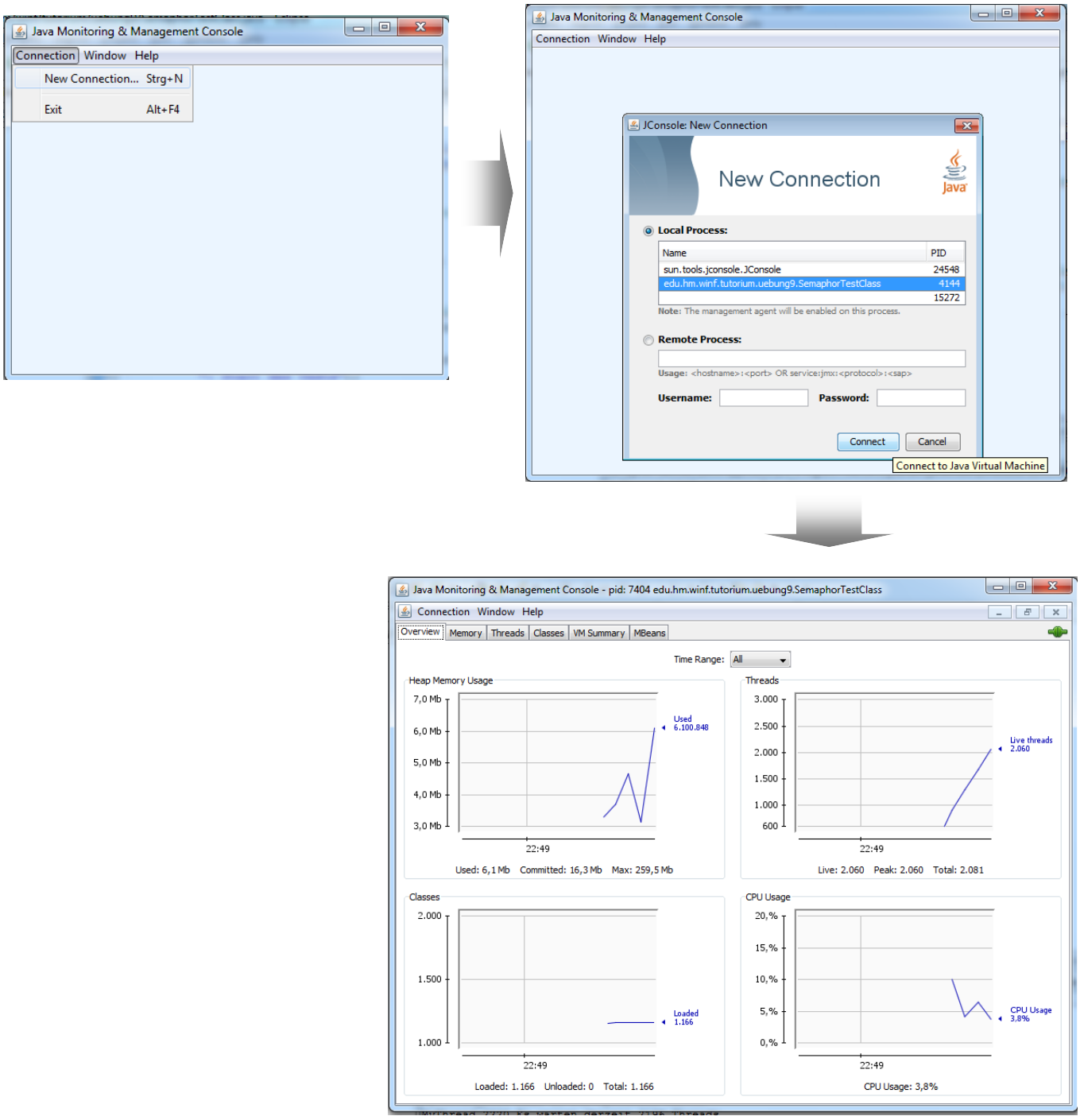
Anwendung mit deren Hilfe die JVM überwacht werden kann. Mit JConsole lassen sich unter anderem der Speicherverbrauch, die Anzahl der Threads und viele weitere nützliche Informationen anzeigen.

Wechseln Sie dazu in das bin-Verzeichnis Ihrer Java-Installation (z.B. C:\Programme\Java\jdk1.6.0_20\bin) und starten Sie das Programm jconsole.exe mit einen Doppelklick. Es müsste folgendes Fenster erscheinen:




Wiederholen Sie den Versuch aus Aufgabe 3 mit einem Semaphorzähler von 1 und 10.000 Threads. Nachdem Sie Ihr Java-Programm gestartet haben, wechseln Sie in die Anzeige der JConsole und eröffnen eine neue Verbindung, so wie es auf den Screenshots dargestellt ist.

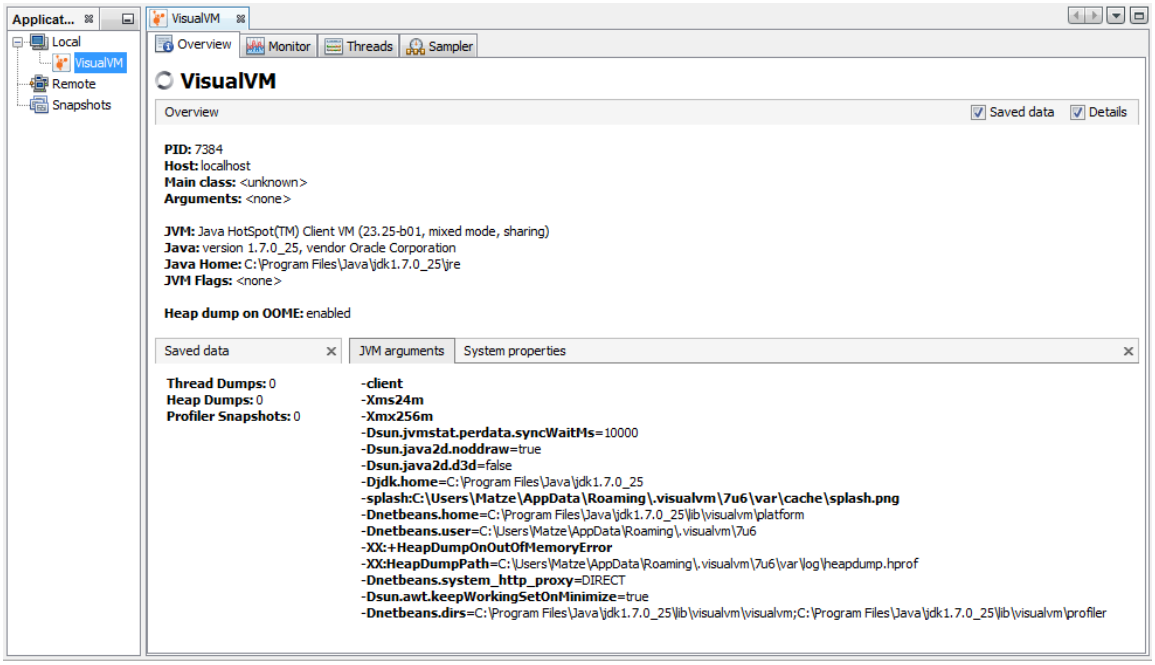
Navigieren Sie sich nun durch die Oberfläche und schauen Sie sich die dargestellten Informationen an.


Java Monitoring & Management Console
Connection Window Help
New Connection... Strg+N
Exit Alt+F4
Java Monitoring & Management Console
Connection Window Help
JConsole: New Connection
New Connection
Local Process:
Name PID
sun.tools.jconsole.JConsole 24548
edu.hm.winf.tutorium.uebung9.SemaphorTestClass 4144
15272
Note: The management agent will be enabled on this process.
Remote Process:
Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap>
Username: Password:
Connect Cancel
Connect to Java Virtual Machine
Java Monitoring & Management Console - pid: 7404 edu.hm.winf.tutorium.uebung9.SemaphorTestClass
Connection Window Help
Overview Memory Threads Classes VM Summary MBeans
Time Range: All
Heap Memory Usage
7,0 Mb
6,0 Mb
5,0 Mb
4,0 Mb
3,0 Mb
Used 6.100.848
22:49
Used: 6,1 Mb Committed: 16,3 Mb Max: 259,5 Mb
Threads
3.000
2.500
2.000
1.500
1.000
600
Live threads 2.060
22:49
Live: 2.060 Peak: 2.060 Total: 2.081
Classes
2.000
1.500
1.000
Loaded 1.166
22:49
Loaded: 1.166 Unloaded: 0 Total: 1.166
CPU Usage
20,%
15,%
10,%
5,%
0,%
CPU Usage 3,8%
22:49
CPU Usage: 3,8%

5. Ein weiteres nützliches Tool, welches sich ebenfalls im „bin“-Verzeichnis der JDK-Installation befindet, ist „Java VisualVM“. Mit diesem lassen sich nützliche Statistiken einer laufenden JVM ablesen, wie z. B. Threads, CPU-Auslastung, Garbage Collection - Informationen, etc. Manche dieser Informationen unterstützt das Tool von Haus aus, für andere stehen kostenfreie Plugins zum Download bereit. Insgesamt ist „Java VisualVM“ ein sehr nützliches Tool zur Performance-Analyse.

   Starten Sie das Programm und „doppelklicken“ Sie im linken Fensterbereich auf „VisualVM“. Nachfolgend verschaffen Sie sich einen Überblick über die Reiter „Overview“, „Monitor“ und „Threads“. Welche Informationen können diesen entnommen werden?

   Allgemein sind im linken Bereich alle laufenden JVMs zu sehen. Zu jeder können via Doppelklick Informationen abgerufen werden. Es ist dabei auch möglich, JVMs auf entfernten Servern zu betrachten. In unserem Fall sind jedoch nur die lokalen JVMs, wie z. B. Ihr Java-Programm, interessant.




   Wiederholen Sie nun erneut den Versuch aus Aufgabe 3 mit einem Semaphorzähler von 1 und 10.000 Threads. Betrachten Sie parallel dazu die Ausgaben in „Java VisualVM“. Verbinden Sie sich hierzu nach dem Start Ihres Java-Programms auf die neue lokale Applikation und betrachten Sie o.g. Reiter.

Anhang: Programmausgabe auf Konsole

Verwendet wurde eine Semaphore mit Semaphorzähler = 1 und 20 nebenläufige Threads:

```
main: Start des Tests
MyThread_1 gestartet
MyThread_2 gestartet
MyThread_3 gestartet
MyThread_4 gestartet
MyThread_5 gestartet
MyThread_1 im kritischen Abschnitt
MyThread_2 muss in P()-Operation warten
MyThread_2 Es warten derzeit 1 Threads
MyThread_3 muss in P()-Operation warten
MyThread_3 Es warten derzeit 2 Threads
MyThread_4 muss in P()-Operation warten
MyThread_4 Es warten derzeit 3 Threads
MyThread_5 muss in P()-Operation warten
MyThread_5 Es warten derzeit 4 Threads
MyThread_6 gestartet
MyThread_7 gestartet
MyThread_8 gestartet
MyThread_9 gestartet
MyThread_10 gestartet
MyThread_11 gestartet
MyThread_6 muss in P()-Operation warten
MyThread_6 Es warten derzeit 5 Threads
MyThread_7 muss in P()-Operation warten
MyThread_7 Es warten derzeit 6 Threads
MyThread_8 muss in P()-Operation warten
MyThread_8 Es warten derzeit 7 Threads
MyThread_9 muss in P()-Operation warten
MyThread_9 Es warten derzeit 8 Threads
MyThread_10 muss in P()-Operation warten
MyThread_10 Es warten derzeit 9 Threads
MyThread_11 muss in P()-Operation warten
MyThread_11 Es warten derzeit 10 Threads
MyThread_12 gestartet
MyThread_13 gestartet
MyThread_14 gestartet
MyThread_15 gestartet
MyThread_16 gestartet
MyThread_12 muss in P()-Operation warten
MyThread_12 Es warten derzeit 11 Threads
MyThread_13 muss in P()-Operation warten
MyThread_13 Es warten derzeit 12 Threads
MyThread_14 muss in P()-Operation warten
MyThread_14 Es warten derzeit 13 Threads
MyThread_15 muss in P()-Operation warten
MyThread_15 Es warten derzeit 14 Threads
MyThread_16 muss in P()-Operation warten
MyThread_16 Es warten derzeit 15 Threads
MyThread_17 gestartet
MyThread_18 gestartet
MyThread_19 gestartet
MyThread_20 gestartet
MyThread_17 muss in P()-Operation warten
MyThread_17 Es warten derzeit 16 Threads
MyThread_18 muss in P()-Operation warten
MyThread_18 Es warten derzeit 17 Threads
MyThread_19 muss in P()-Operation warten
MyThread_19 Es warten derzeit 18 Threads
MyThread_20 muss in P()-Operation warten
MyThread_20 Es warten derzeit 19 Threads
MyThread_1 verlässt kritischen Abschnitt
MyThread_1 V()-Operation ausgeführt, andere Threads dürfen wieder
```

```
MyThread_1 In V()-Operation, es warten derzeit 19 Threads
MyThread_2 in P-Operation(), zurück aus wait; es warten derzeit 19 Threads
MyThread_2 im kritischen Abschnitt
MyThread_2 verlässt kritischen Abschnitt
MyThread_2 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_2 In V()-Operation, es warten derzeit 18 Threads
MyThread_3 in P-Operation(), zurück aus wait; es warten derzeit 18 Threads
MyThread_3 im kritischen Abschnitt
MyThread_3 verlässt kritischen Abschnitt
MyThread_3 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_3 In V()-Operation, es warten derzeit 17 Threads
MyThread_4 in P-Operation(), zurück aus wait; es warten derzeit 17 Threads
MyThread_4 im kritischen Abschnitt
MyThread_4 verlässt kritischen Abschnitt
MyThread_4 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_4 In V()-Operation, es warten derzeit 16 Threads
MyThread_5 in P-Operation(), zurück aus wait; es warten derzeit 16 Threads
MyThread_5 im kritischen Abschnitt
MyThread_5 verlässt kritischen Abschnitt
MyThread_5 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_5 In V()-Operation, es warten derzeit 15 Threads
MyThread_6 in P-Operation(), zurück aus wait; es warten derzeit 15 Threads
MyThread_6 im kritischen Abschnitt
MyThread_6 verlässt kritischen Abschnitt
MyThread_6 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_6 In V()-Operation, es warten derzeit 14 Threads
MyThread_7 in P-Operation(), zurück aus wait; es warten derzeit 14 Threads
MyThread_7 im kritischen Abschnitt
MyThread_7 verlässt kritischen Abschnitt
MyThread_7 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_7 In V()-Operation, es warten derzeit 13 Threads
MyThread_8 in P-Operation(), zurück aus wait; es warten derzeit 13 Threads
MyThread_8 im kritischen Abschnitt
MyThread_8 verlässt kritischen Abschnitt
MyThread_8 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_8 In V()-Operation, es warten derzeit 12 Threads
MyThread_9 in P-Operation(), zurück aus wait; es warten derzeit 12 Threads
MyThread_9 im kritischen Abschnitt
MyThread_9 verlässt kritischen Abschnitt
MyThread_9 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_9 In V()-Operation, es warten derzeit 11 Threads
MyThread_10 in P-Operation(), zurück aus wait; es warten derzeit 11 Threads
MyThread_10 im kritischen Abschnitt
MyThread_10 verlässt kritischen Abschnitt
MyThread_10 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_10 In V()-Operation, es warten derzeit 10 Threads
MyThread_11 in P-Operation(), zurück aus wait; es warten derzeit 10 Threads
MyThread_11 im kritischen Abschnitt
MyThread_11 verlässt kritischen Abschnitt
MyThread_11 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_11 In V()-Operation, es warten derzeit 9 Threads
MyThread_12 in P-Operation(), zurück aus wait; es warten derzeit 9 Threads
MyThread_12 im kritischen Abschnitt
MyThread_12 verlässt kritischen Abschnitt
MyThread_12 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_12 In V()-Operation, es warten derzeit 8 Threads
MyThread_13 in P-Operation(), zurück aus wait; es warten derzeit 8 Threads
MyThread_13 im kritischen Abschnitt
MyThread_13 verlässt kritischen Abschnitt
MyThread_13 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_13 In V()-Operation, es warten derzeit 7 Threads
MyThread_14 in P-Operation(), zurück aus wait; es warten derzeit 7 Threads
MyThread_14 im kritischen Abschnitt
MyThread_14 verlässt kritischen Abschnitt
MyThread_14 V()-Operation ausgeführt, andere Threads dürfen wieder
```

```
MyThread_14 In V()-Operation, es warten derzeit 6 Threads
MyThread_15 in P-Operation(), zurück aus wait; es warten derzeit 6 Threads
MyThread_15 im kritischen Abschnitt
MyThread_15 verlässt kritischen Abschnitt
MyThread_15 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_15 In V()-Operation, es warten derzeit 5 Threads
MyThread_16 in P-Operation(), zurück aus wait; es warten derzeit 5 Threads
MyThread_16 im kritischen Abschnitt
MyThread_16 verlässt kritischen Abschnitt
MyThread_16 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_16 In V()-Operation, es warten derzeit 4 Threads
MyThread_17 in P-Operation(), zurück aus wait; es warten derzeit 4 Threads
MyThread_17 im kritischen Abschnitt
MyThread_17 verlässt kritischen Abschnitt
MyThread_17 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_17 In V()-Operation, es warten derzeit 3 Threads
MyThread_18 in P-Operation(), zurück aus wait; es warten derzeit 3 Threads
MyThread_18 im kritischen Abschnitt
MyThread_18 verlässt kritischen Abschnitt
MyThread_18 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_18 In V()-Operation, es warten derzeit 2 Threads
MyThread_19 in P-Operation(), zurück aus wait; es warten derzeit 2 Threads
MyThread_19 im kritischen Abschnitt
MyThread_19 verlässt kritischen Abschnitt
MyThread_19 V()-Operation ausgeführt, andere Threads dürfen wieder
MyThread_19 In V()-Operation, es warten derzeit 1 Threads
MyThread_20 in P-Operation(), zurück aus wait; es warten derzeit 1 Threads
MyThread_20 im kritischen Abschnitt
MyThread_20 verlässt kritischen Abschnitt
main: Programmende
```