VisualVM-Workshop – Zusammenfassung

Ziel des Workshops

- Verständnis für Performance-Probleme in Java-Anwendungen entwickeln
- Erste Erfahrung mit VisualVM
- Einsatz von VisualVM zur Analyse von:
    - CPU-Hotspots
    - Speicherverbrauch (Heap)
    - Threads und Zustände
    - Deadlocks

Voraussetzungen

- Java 8–23 (z. B. Temurin/OpenJDK)
- VisualVM installiert
  → Download: https://visualvm.github.io
  → macOS: z. B. per `brew install --cask visualvm`
- Eine Java-IDE (z. B. IntelliJ IDEA oder VS-Code)

Hauptfunktionen von VisualVM

Tab	Funktion
Monitor	CPU-Auslastung, Heap-Größe, GC-Zyklen
Sampler	Stichprobenbasiertes Profiling (leichtgewichtig, geringer Overhead)
Profiler	Genaue Analyse mit Laufzeit-Overhead
Threads	Darstellung von Thread-Zuständen (Running, Sleeping, Waiting)
Heap Dump	Schnappschuss des Speichers zur Untersuchung von Speicherproblemen

---

CPU-Profiling

- Ziel: Methoden identifizieren, die viel CPU-Zeit verbrauchen
- Verwendung: Sampler → "CPU"
- Begriffe:
    - **Total Time**: Gesamtzeit inkl. aller aufgerufenen Methoden
    - **Total Time (CPU)**: Tatsächlich verbrauchte CPU-Rechenzeit
    - **Self Time**: Zeit, die ausschließlich in dieser Methode verbracht wurde

---

Speicheranalyse (Heap-Analyse)

- Beobachtung des Speicherverbrauchs über den "Monitor"-Tab
- "Sampler → Memory" zeigt Objektanzahl und Speicherbedarf pro Klasse
- Heap Dump ermöglicht:
    - Analyse lebender Objekte
    - Aufspüren von Speicherlecks
- Spalten:
    - Live Bytes = aktueller Speicherverbrauch durch Klasse
    - Live Objects = aktuelle Objektanzahl dieser Klasse

Thread-Analyse

- Der Threads-Tab zeigt:
    - Alle aktiven Threads in einer Zeitachse
    - Zustände:
        - Grün = Running
        - Orange/blau = Sleeping
        - Gelb = Waiting
- Thread Dump kann zur tiefergehenden Analyse verwendet werden
- Deadlocks sind erkennbar, wenn sich zwei Threads gegenseitig blockieren

---

Vergleich: Sampler vs. Profiler

Merkmal	**Sampler**	**Profiler**
Overhead	Gering	Hoch
Genauigkeit	Gute, Ungefähre Werte (Sampling)	Exakte Zeiten und Aufrufzahlen
Anwendung	Übersicht, schnelle Diagnose	Tiefenanalyse, exakte Messung

---

Beispielprogramme

Datei	**Beschreibung**
CpuLoadToggle.java	Interaktiv steuerbare CPU-Last
MemoryLeakDemo.java	Speicherleck durch wachsendes Array
ThreadDemo.java	Threads mit verschiedenen Zuständen / Deadlock simulation

---

Wichtige Links

- VisualVM Website: https://visualvm.github.io
- Dokumentation: https://visualvm.github.io/documentation.html
  JMX Remote Guide (Oracle):
  https://docs.oracle.com/javase/8/docs/technotes/guides/management/agent.html
- GitHub-Repository mit Code: https://github.com/liRahimpour/visualvm_workshop

---

Nützliche Befehle

bash
```
# Projekt kompilieren
javac org/workshop/CpuLoadToggle.java

# Programm starten
java org.workshop.CpuLoadToggle

# VisualVM starten in terminal (macOS/Linux)
visualvm
```

Weiterführende Ideen

- Heap Dumps vergleichen (vor/nach Aktionen)
- Instrumentiertes Profiling testen
- Eigene Projekte mit VisualVM analysieren
- JFR (Java Flight Recorder) nutzen
