Java: Visibility + Race Conditions

1 Java Speichermodell

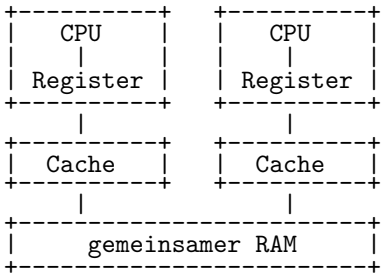
Ein eigener Stack für jeden Thread

- lokale Variablen
- Methodenparameter
- Rückgabewerte
- Verwaltungsaufrufe für Methodenaufrufe

Ein gemeinsamer Heap für alle Threads

- Objekte
- Membervariablen
- Klassenvariablen (static)

2 Hardware Speicherarchitektur



3 Probleme beim Zugriff mehrerer Threads auf gemeinsame Daten

Sichtbarkeit

Wenn ein Thread den Wert einer **gemeinsamen Variablen** verändert, kann es sein, dass ein anderer Thread die Änderung des Wertes **nicht sieht**, weil die Änderung nur im Cache stattgefunden hat.

Lösung

Variable als **volatile** deklarieren:
Der Lese- und Schreibzugriff findet immer direkt auf dem RAM statt.
⇒ **visibility** ist gewährleistet, aber nicht **atomicity**

Race Conditions

Mehrere Threads führen **den selben Quellcode** aus und greifen **schreibend** auf **gemeinsame Variablen** zu. Je nachdem, in welcher Reihenfolge die Einzelbefehle der Threads ausgeführt werden, kann nach Ausführung des Codes der Wert der Variablen unterschiedlich sein.

Lösung

Codeabschnitt als **synchronized** markieren:
Der Codeabschnitt wird immer nur von einem Thread gleichzeitig ausgeführt.
⇒ **visibility** und **atomicity** sind gewährleistet.