Application Performance Management

Frühling 2021

Object Lifecycle

Zoltán Majó

Agenda

Finalisierung GC Tuning

- Besprechung Arbeitsblatt
- Diskussion

Object Lifecycle

Individualarbeit 2: GC Tuning

Ziele des Tunings

- Durchsatz → Übung 1 (ThroughputTuning.pdf)
- Reaktionsfähigkeit → Übung 2 (PauseTimeTuning.pdf)

Besprechung Arbeitsblatt

Diskussion

Ist GC Tuning einfach oder eher schwierig? Wieso?

Kommandozeilen – Beispiele

```
java -Xmx12g -XX:MaxPermSize=64M -XX:PermSize=32M -XX:MaxNewSize=2g
     -XX:NewSize=1g -XX:SurvivorRatio=128 -XX:+UseParNewGC
     -XX:+UseConcMarkSweepGC -XX:MaxTenuringThreshold=0
     -XX:CMSInitiatingOccupancyFraction=60 -XX:+CMSParallelRemarkEnabled
     -XX:+UseCMSInitiatingOccupancyOnly -XX:ParallelGCThreads=12
     -XX:LargePageSizeInBytes=256m …
java –Xms8g –Xmx8g –Xmn2g -XX:PermSize=64M -XX:MaxPermSize=256M
    -XX:-OmitStackTraceInFastThrow -XX:SurvivorRatio=2 -XX:-UseAdaptiveSizePolicy
    -XX:+UseConcMarkSweepGC -XX:+CMSConcurrentMTEnabled
    -XX:+CMSParallelRemarkEnabled -XX:+CMSParallelSurvivorRemarkEnabled
    -XX:CMSMaxAbortablePrecleanTime=10000 -XX:+UseCMSInitiatingOccupancyOnly
    -XX:CMSInitiatingOccupancyFraction=63 -XX:+UseParNewGC –Xnoclassgc …
```

Flags

java -XX:+PrintFlagsFinal | grep "GC\|CMS\|G1”
169

```
uintx AdaptiveSizeMajorGCDecayTimeScale         = 10                                  {product}
uintx AutoGCSelectPauseMillis                   = 5000                                {product}
 bool BindGCTaskThreadsToCPUs                  = false                               {product}
 bool CMSAbortSemantics                        = false                               {product}
uintx CMSAbortablePrecleanMinWorkPerIteration   = 100                                 {product}
 intx CMSAbortablePrecleanWaitMillis           = 100                                 {manageable}
uintx CMSBitMapYieldQuantum                     = 10485760                            {product}
uintx CMSBootstrapOccupancy                     = 50                                  {product}
...
```

Übergeordnete Flags

Die Oracle VM erlaubt dem Benutzer Ziele zu definieren

- Durchsatz: `-XX:GCTimeRatio=`
- Pausenzeiten: `-XX:MaxGCPauseMillis=`

Wie gut klappt das für unser Beispiel aus der Individualarbeit?

Frage

Können wir Applikationseingeschaften festhalten um GC zu lenken?

- Rate der Allozierungen (engl. allocation rate)
- Rate der Mutationen (engl. mutation rate)
- ...

Ohne messbare Applikationseigenschaften: Experimentieren

> If the heap grows to its maximum size and the throughput goal isn't being met, then the maximum heap size is too small for the throughput goal. Set the maximum heap size to a value that's close to the total physical memory on the platform, but doesn't cause swapping of the application. Execute the application again. If the throughput goal still isn't met, then the goal for the application time is too high for the available memory on the platform.

HotSpot Virtual Machine Garbage Collection Tuning Guide (empfohlene Bibliographie)

Praxis

Manchmal muss man mit den zur Verfügung stehenden Werkzeugen was machen...

Empfehlung(en) für Tuning

- Sinnvolle und systematische Experimente
- Genug Zeit einplanen
- ...

Limiten der GC-Algorithmen gelten

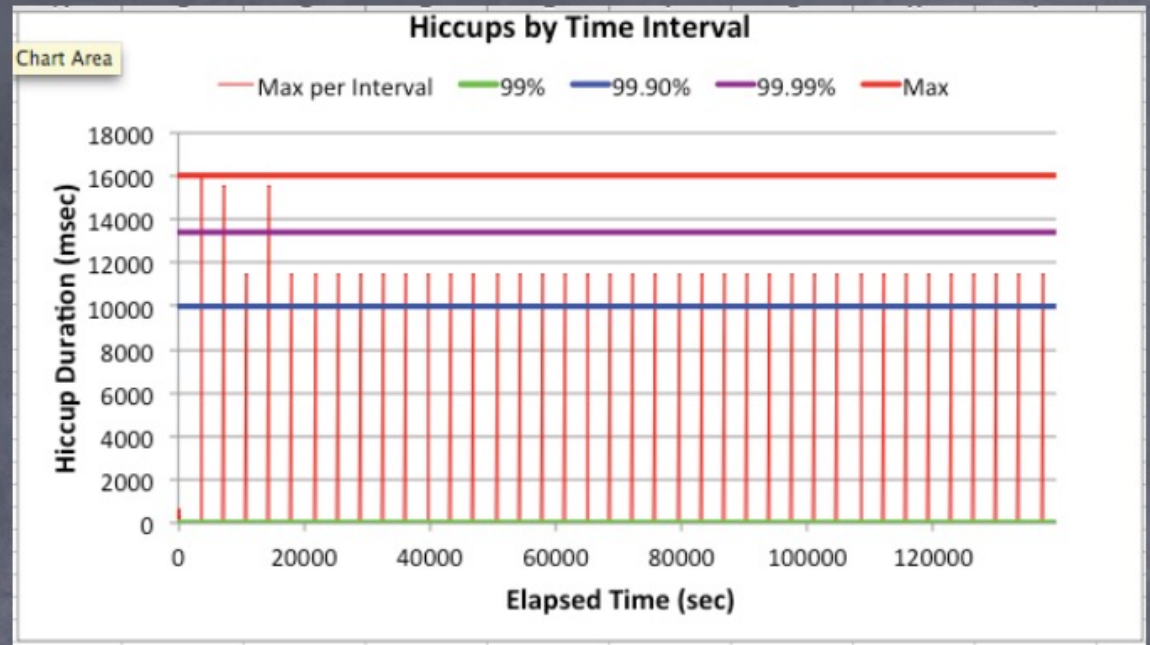
- Stop-the-world kann bei allen in GC-Implementierungen in der Oracle JVM vorkommen

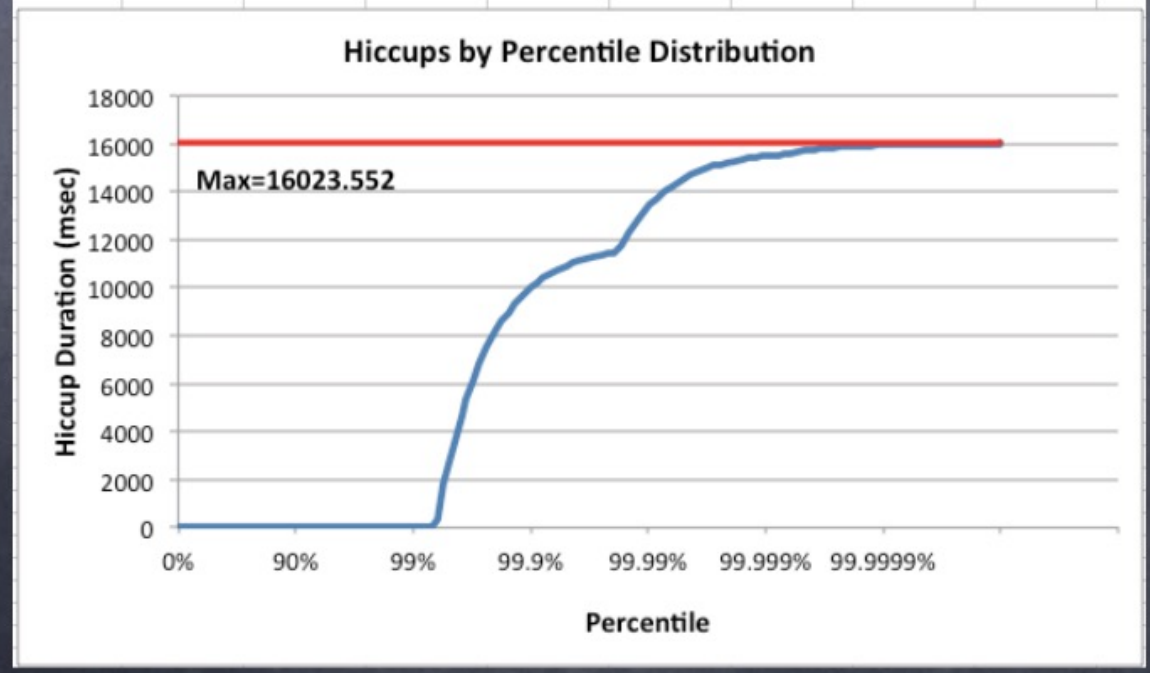
Neue GC-Implementierungen mit niedrigeren Pausenzeiten

- Azul Zing VM C4 – Continuously Concurrent Compacting Collector
- Oracle JVM Shenandoah GC – ultra-low pause time garbage collector
- Oracle JVM ZGC – scalable low latency garbage collector

Azul C4

Die folgenden 3 Folien sind von Gil Tene (Azul) übernommen worden





AZUL SYSTEMS

Another way to cope: "Creative Language"

- "Guarantee a worst case of X msec, 99% of the time"

- "Mostly" Concurrent, "Mostly" Incremental

Translation: "Will at times exhibit long monolithic stop-the-world pauses"

- "Fairly Consistent"

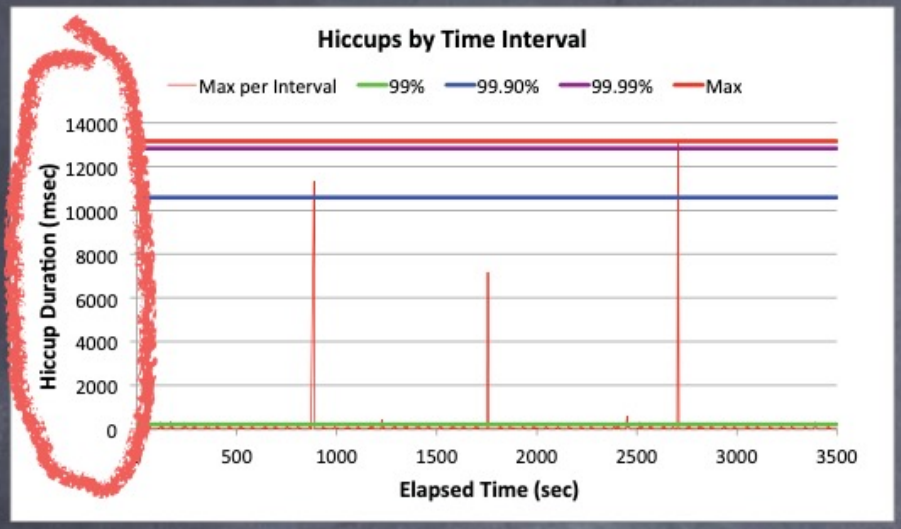
Translation: "Will sometimes show results well outside this range"

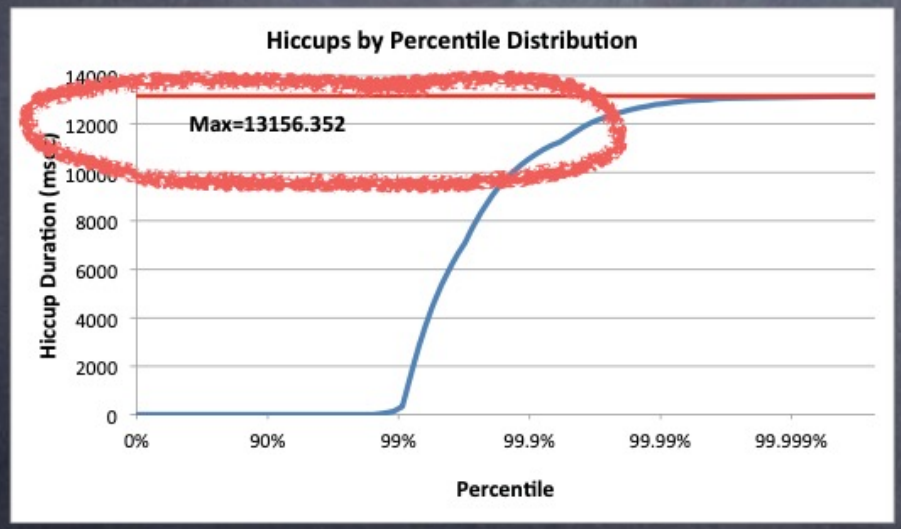
- "Typical pauses in the tens of milliseconds"

Translation: "Some pauses are much longer than tens of milliseconds"

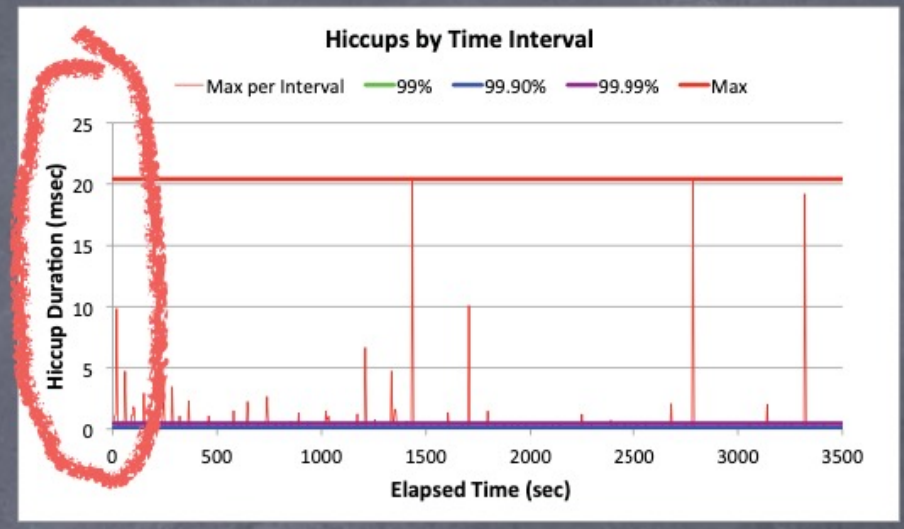
AZUL SYSTEMS

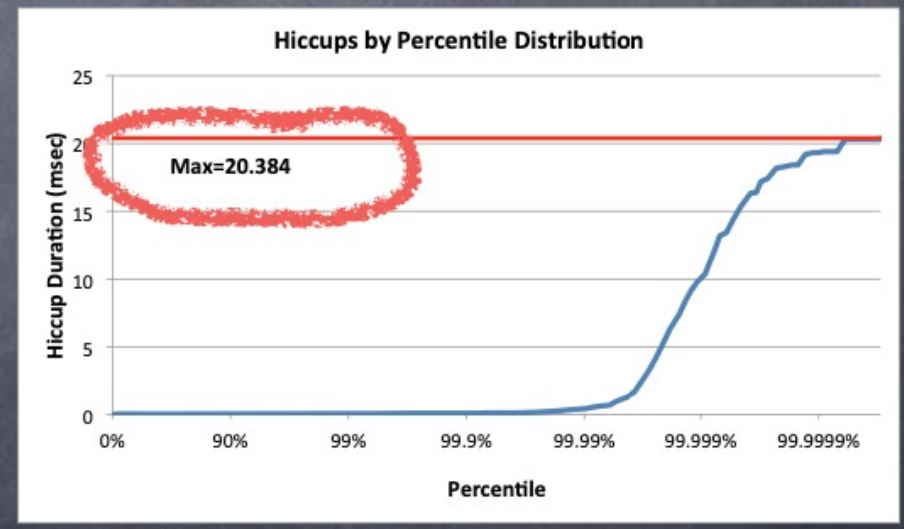
Oracle HotSpot CMS, 1GB in an 8GB heap





Zing 5, 1GB in an 8GB heap





AZUL SYSTEMS

Oracle ZGC

Empfohlene Bibliographie

- Gute Zusammenfassung: https://www.youtube.com/watch?v=88E86quLmQA (39 Minuten)
- Details: https://wiki.openjdk.java.net/display/zgc/Main

Agenda

Finalisierung GC Tuning

- Besprechung Arbeitsblatt
- Diskussion

Object Lifecycle

Kurze Wiederholung

Zentrale Frage eines GC: Welche Objekte sind Müll?

- Ein Objekt im Programm ist Müll, wenn keine Berechnung im Programm dieses Objekt wieder verwendet.

Übliche Lösung: Ein Objekt ist Müll, wenn es von den ”Roots” aus nicht mehr erreichbar ist

Frage: Ist das eine zufriedenstellende Lösung?

- Denken Sie an Memory Leaks

Nicht verwendet ≠ Nicht erreichbar

Übliche GC-Ansicht: Ein Objekt ist Müll, wenn es von den ”Roots” aus nicht mehr erreichbar ist

Bietet nur eine zurückhaltende Approximation

- Die Erreichbarkeit eines Objekts heiss nicht, dass dieses Objekt nochmals verwendet wird
- Potentielles Memory Leak

Was tun?

Eine Möglichkeit: dem Programmierer erlauben die Erreichbarkeit von Objekten zu kontrollieren

- Mittels «Weak References»
- Auch von Java unterstützt
  - `WeakReference`, `SoftReference`, `PhantomReference`

Beispiel: Strong vs. Weak Reference

Beispiel

Strong Reference

```
Object getRandomData() {
  Object a = new Integer(”rand_nr”);
  return a;
}
...
Object b = getRandomData();
// a is strongly reachable here
// (via b)
```

Weak Reference

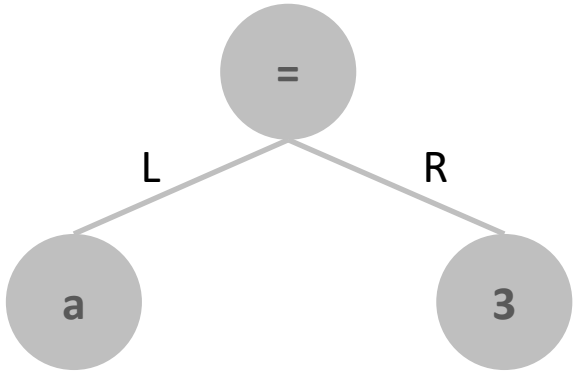
```
Object getRandomData() {
  Object a = new Integer(”rand_nr”);
  return new WeakReference(a);
}
...
Object b = getRandomData();
// a is weakly reachable here
// (via b)
```

Wie können Weak References genutzt werden?

Einfaches Beispielszenario: Compiler

Phase 1: Syntaktische Analyse

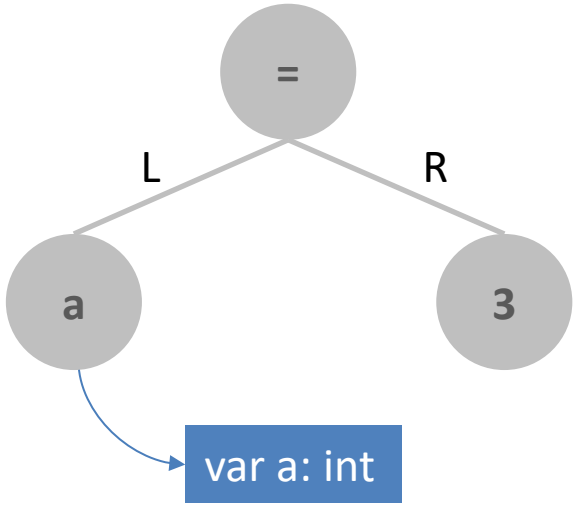
- Generiert Intermediate Representation (IR)
- Zum Beispiel Folgendes für `a = 3`



Einfaches Beispielszenario: Compiler

Phase N: Semantische Analyse

- Ergänzt IR mit Informationen



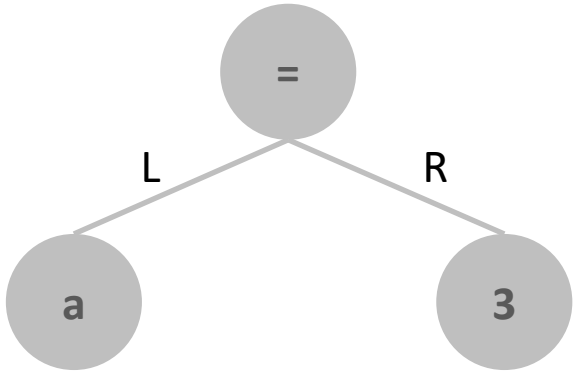
Frage: Wo kann man diese Zusatzinformationen speichern?

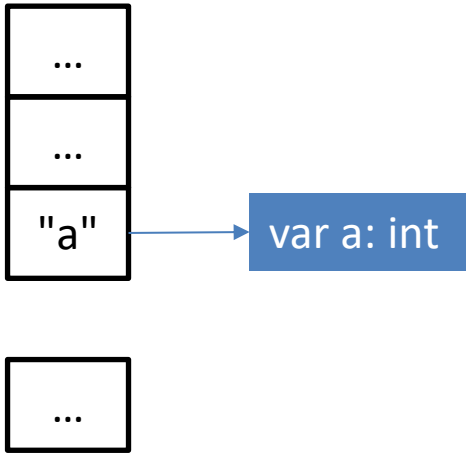
- Eine Möglichkeit: Eine Hash-Table verwenden

Einfaches Beispielszenario: Compiler

Phase N: Semantische Analyse

- Ergänzt IR mit Informationen

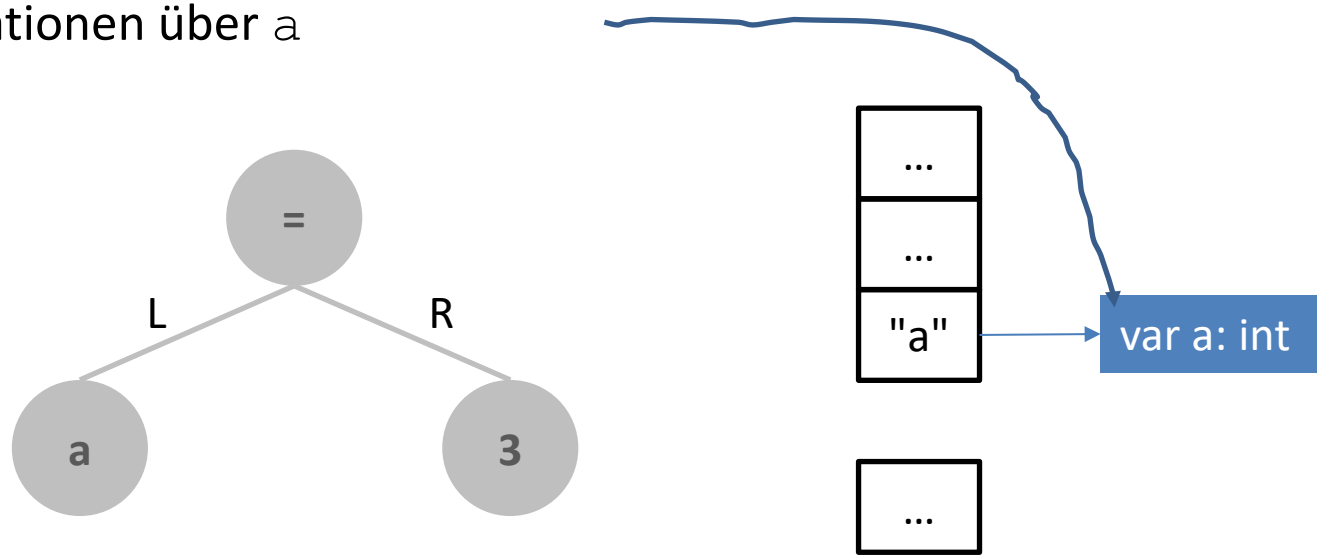




Einfaches Beispielszenario: Compiler

Phase N + 1: Optimierung 1

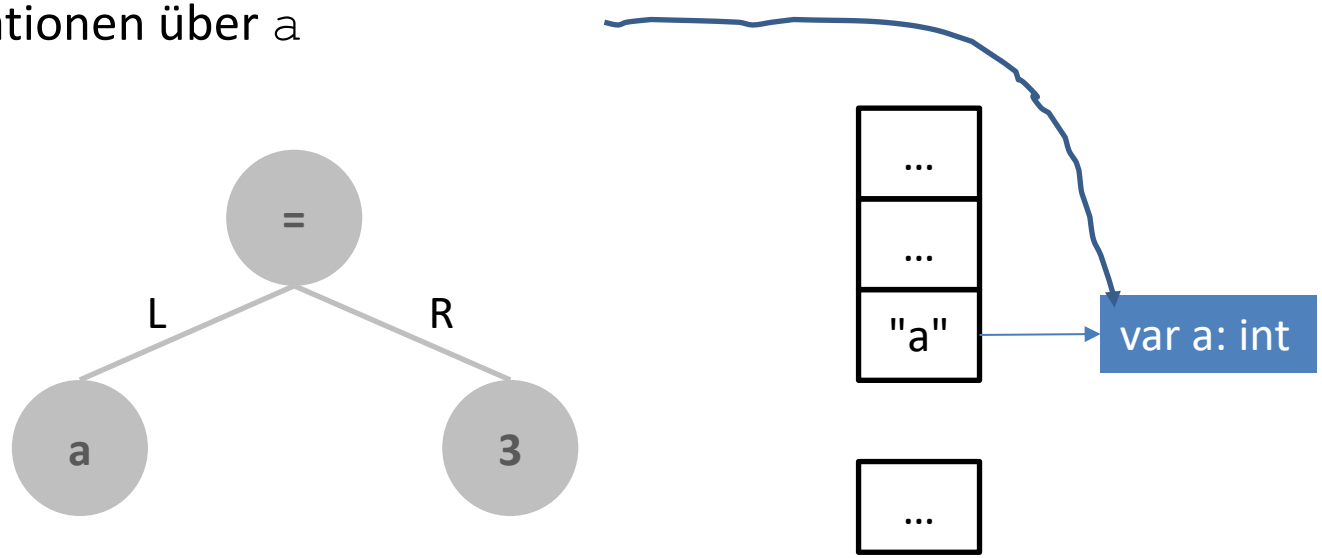
- Nutzt die Informationen über `a`



Einfaches Beispielszenario: Compiler

Phase N + 2: Codegenerierung

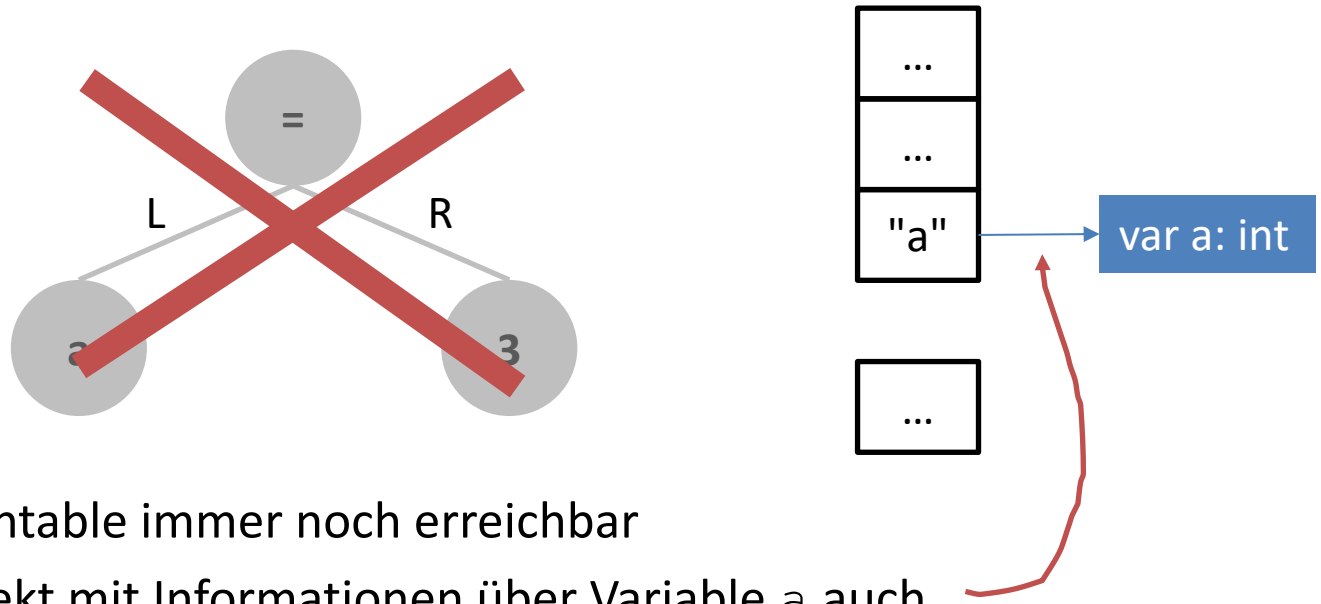
- Nutzt die Informationen über `a`



Einfaches Beispielszenario: Compiler

Kompilierung einer anderen Methode

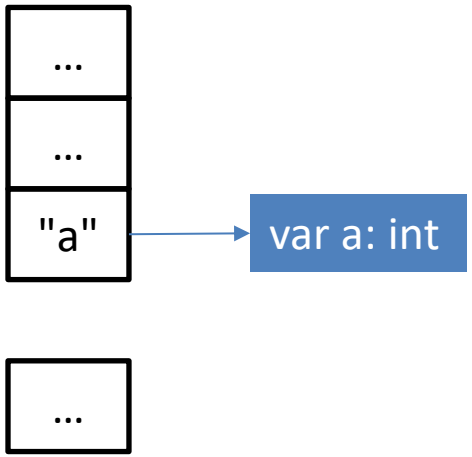
- IR für methode nicht mehr relevant: wird GCd



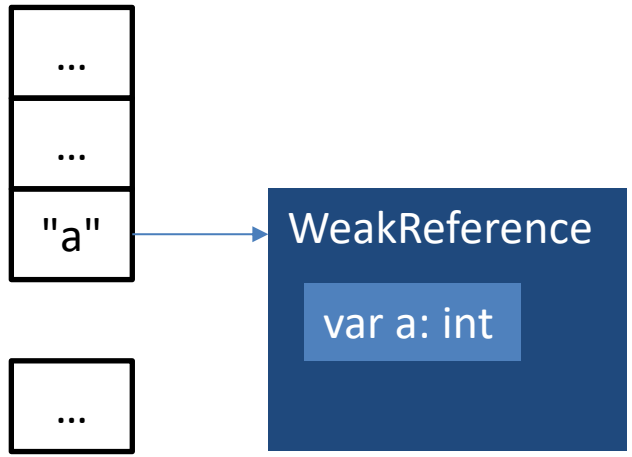
- Hashtable immer noch erreichbar
- Objekt mit Informationen über Variable `a` auch
- (Potentieller) Memory Leak

Lösung mit WeakReference

Vorher



Nachher



Sobald `var a:int` nur via WeakReference erreichbar, kann es GCd werden

- Falls Tabelle eine `java.util.WeakHashMap` ist, Objekt wird automatisch aus der Tabelle entfernt

Andere mögliche Anwendungsfälle (von Weak References)

Caching von Daten (z.B. Bilder)

Andere Möglichkeiten?

Strong References, Weak References...

...und das wäre es.

Nicht ganz.

Mögliche Referenzentypen (Java)

An object is *strongly reachable* if it can be reached by some thread without traversing any reference objects. A newly-created object is strongly reachable by the thread that created it.

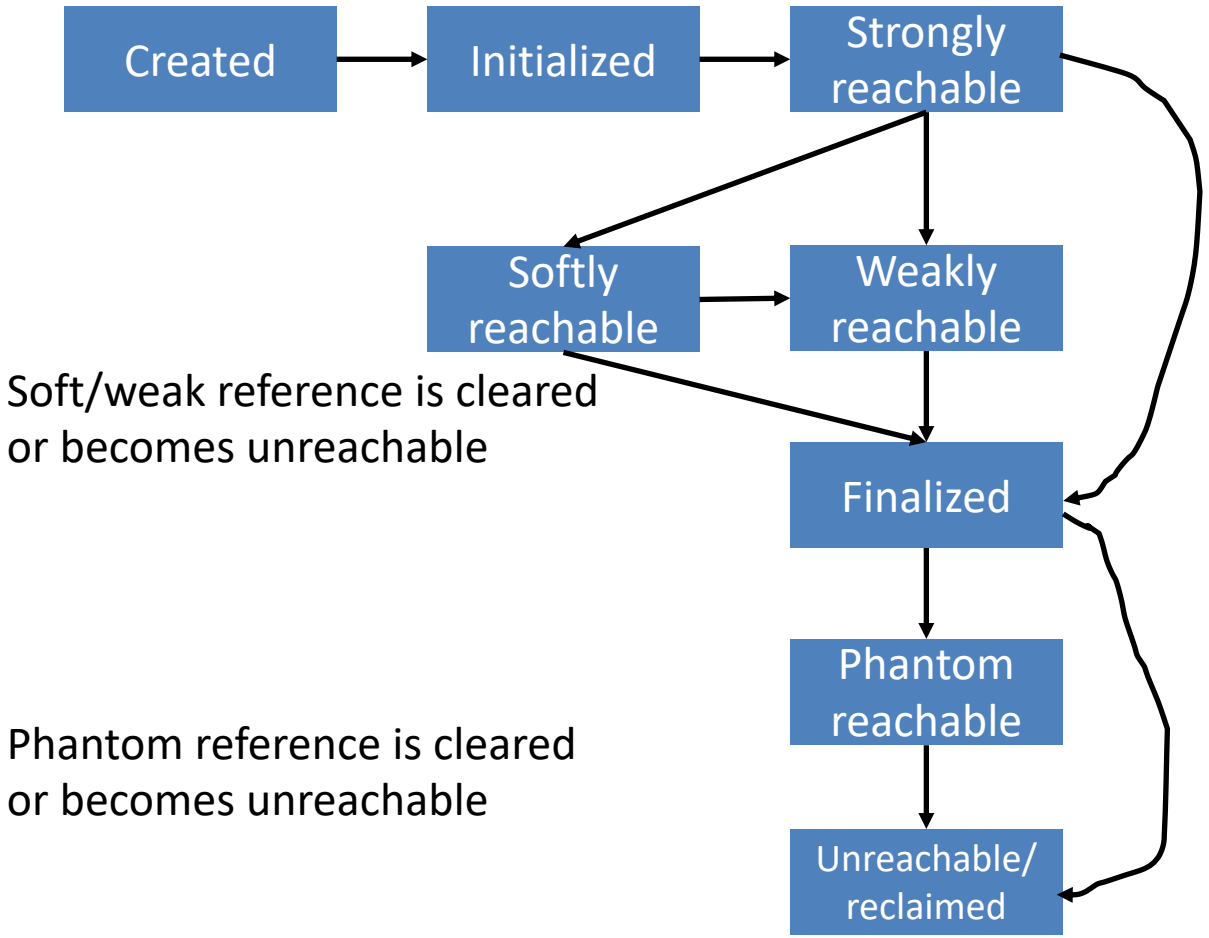
An object is *softly reachable* if it is not strongly reachable but can be reached by traversing a soft reference.

An object is *weakly reachable* if it is neither strongly nor softly reachable but can be reached by traversing a weak reference. When the weak references to a weakly-reachable object are cleared, the object becomes eligible for finalization.

An object is *phantom reachable* if it is neither strongly, softly, nor weakly reachable, it has been finalized, and some phantom reference refers to it.

Finally, an object is *unreachable*, and therefore eligible for reclamation, when it is not reachable in any of the above ways.

Object Lifecycle (eines Java-Objekts)



Komplikationen

Objekte können wieder zum Leben erweckt werden

- Engl. «resurrect»
- `this` Pointer verfügbar in der `finalize()` Methode

```
GlobalObject o;
...
void finalize() {
    o.ref = this;
}
```

Zusammenfassung GC

GC

- Automatische Speicherverwaltung kann Softwareentwicklung erleichtern
- Kein Heilmittel gegen Memory Leaks

Weak References: Teils manuelle Speicherverwaltung

- Vorteile: Erlaubt mehr Kontrolle
- Nachteile: Kompliziert und kann unerwartete Nebeneffekte haben
  - (auch schlaue Programmierer machen Fehler)

Individualarbeit

Erklärung des Verhaltens des Programmes `Phantom.java` (siehe Repo)