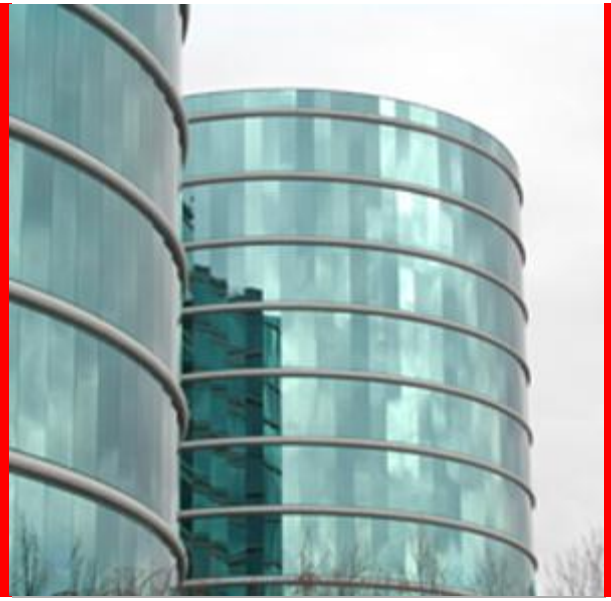

ORACLE®



ORACLE

Java Prestanda – Patterns & Anti-patterns

Nils Eliasson
Senior Member of Technical Staff, JRockit Dev Team

Prestanda

En kort repetition

- Hur snabbt man får ett jobb gjort
  - Utfört arbete
  - Tid
  - Resursutnyttjande
    - Maskiner
    - Energi
  - Skalbarhet
  - Paustider

Dåliga Vanor

Hett förra deceniet

- Återanvända JVM-parametrar
  - Olika JVMer
    - JRockit
    - Hotspot
    - m.fl.
  - Olika versioner av en JVM (Major versioner)
  - Olika program
- Klassiker
  - GC trimning

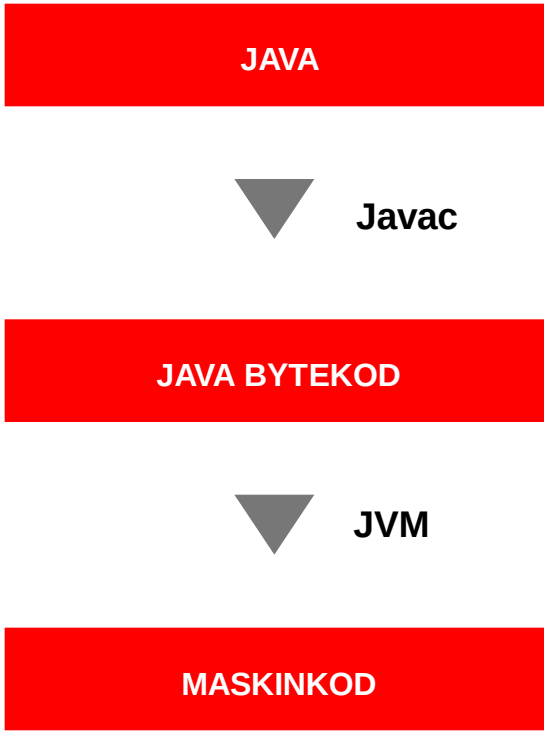
Dåliga Vanor

Gamla godingar

- Finaliserare för att städa upp tillstånd
  - Körs endast en gång
  - Kan kasta fel
- Referensobjekt
  - Lätt att göra fel ex. WeakHashMap
- Resursläckor
  - Kräver en hel GC (Old Collection)
  - Minne
  - FileHandles, Sockets

Förhastade optimeringar

Att mikrooptimera javakod




```
public static void main(String [] args) {
    System.out.println(”Hello world!”);
…

public static void main(java.lang.String[]);
  0:  getstatic      #2; //Field …
  3:  ldc            #3; //String …
  5:  invokevirtual #4; //Method …
  8:  return
```

```
sub   $0x04,%esp
mov   0x04fa9bc4,%esi
mov   0x05cd5498,%edi
mov   0x0(%esi),%ecx
call  0x05020F00
pop   %ecx
```

Förhastade optimeringar

Missuppfattningar

1. Metodanrop är dyrt
   - Manuell inlining
     - Ökar kodmassan
     - Förstör abstraktionen
2. Allokering kostar
   - Objektpoolning
     - Dödar lokalitet
     - Kan förstöra en sammanhållen objektlivscykel

- Mikrooptimering i högnivåspråk
  - Svårt för JVM att göra ogjort

TANSTAAFL

Exceptions som flödeskontroll

```
int i = 0;
int sum = 0;

try {
      while(true) {
            sum += a[i];
            i++;
      }
} catch (IndexOutOfBoundsE…);
```

```
while (i < a.len){
    sum += a[i];
    i++;
}

for (int i : a) {
    sum += a[i];
}
```

- Försvårar analysen
- Som bäst lika bra
- Använd Exceptions för att modelera abnormala tillstånd

Uppfinna hjulet

Kan själv

- Ersätta standardstrukturer med hemmasnickrade
  - Intention - Enklare implementation
  - Missar viktiga aspekter
- Att göra en collection trådsäker
  - 2 vanligaste felen
    - För många lås
    - För få lås
- Klassbiblioteken är välkända och i många fall väloptimerade

Uppfinna hjulet

Kan själv

- Skriva egna låsprimitiv
  - Explicit användande av volatile
    - Förstör optimeringsmöjligheter
    - Forcerar minnessynkronisering – Dyrt!
  - Synchronized är kraftfullt
    - Adaptiva lås
      - Lazy-lås
      - Tunna spinlås
      - Feta lås
    - Lås fusionering

Uppfinna hjulet

Kan själv

```
Class A {
    volatile int x;
    void inc(int y){
       x += y;
    }
}

a.inc(2);
…
a.inc(4);
…
a.inc(2);
```

```
Class B {
    int x;
    synchronized void inc(int y){
       x += y;
    }
}

syncronized(b) {
   b.inc(2);
}
...
syncronized(b) {
   b.inc(4);
}
...
syncronized(b) {
   b.inc(2);
}
```

Microbenchmarking

Du är vad du mäter

```
createArrays(); // Försöker dölja skapandet av dom två arrayerna

Object[] localFromArray = fromObjectArr;  // skapad t.ex. som en array av Comparables
Object[] localToArray = toObjectArr;          // skapad t.ex.som en array av Objects

loop.start();
while (!loop.done()) {
   for (int i : localFromArray) {
      localToArray[i] = localFromArray[i];
   }
}
```

Microbenchmarking

Du är vad du mäter

- Testar ArrayStoreCheckException
  - Mäter inga ArrayStoreException-checkar
  - Mäter inga IndexOutOfBounds heller för den delen
  - Skulle kanske inte ens behöva kopiera
- Felaktiga data ger felaktiga beslut

Pålitlig prestanda

Vad göra?

- Bra design och god kodhygien
- Mät på riktiga applikationer under verklig last
  - Runtime profilering
    - JRockit Mission Control
    - Visual VM
    - M.fl.
- Datalokalitet
  - Minne
  - I/O


ORACLE®

ORACLE IS THE INFORMATION COMPANY