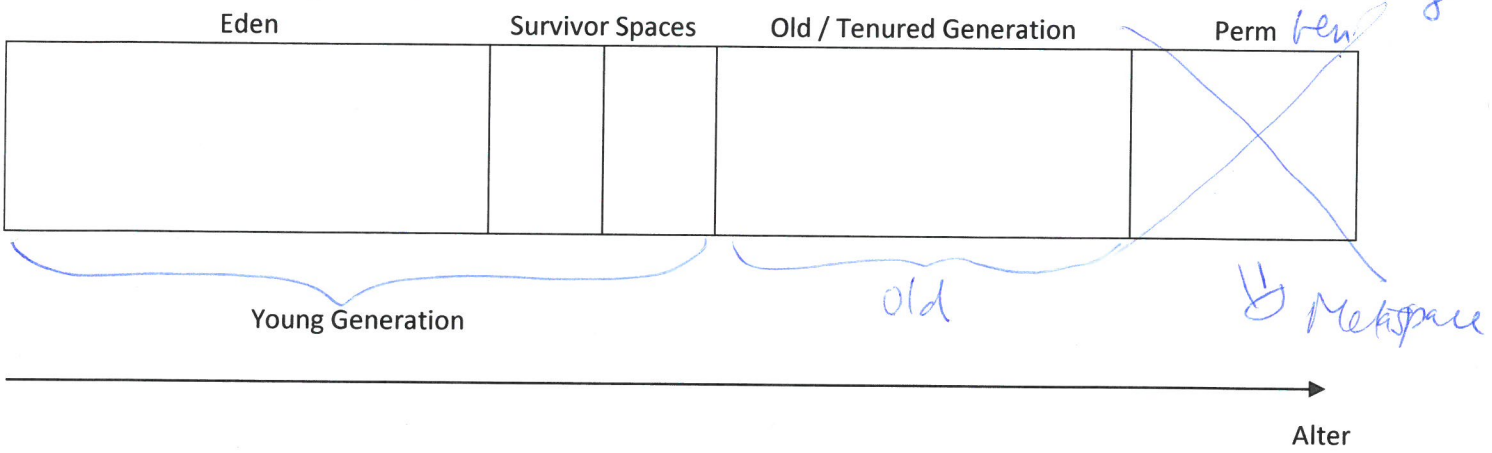
3. Generational Garbage Collection

Kernidee:

- Diese typische demographische Struktur für den Garbage Collector nutzen.
- Statisch den Heap in Bereiche aufteilen für verschiedene Altersgruppen.
- Dynamisch: in den verschiedenen Bereichen unterschiedliche GC-Algorithmen anwenden.

3.1 Statische Heapstruktur



Legende:

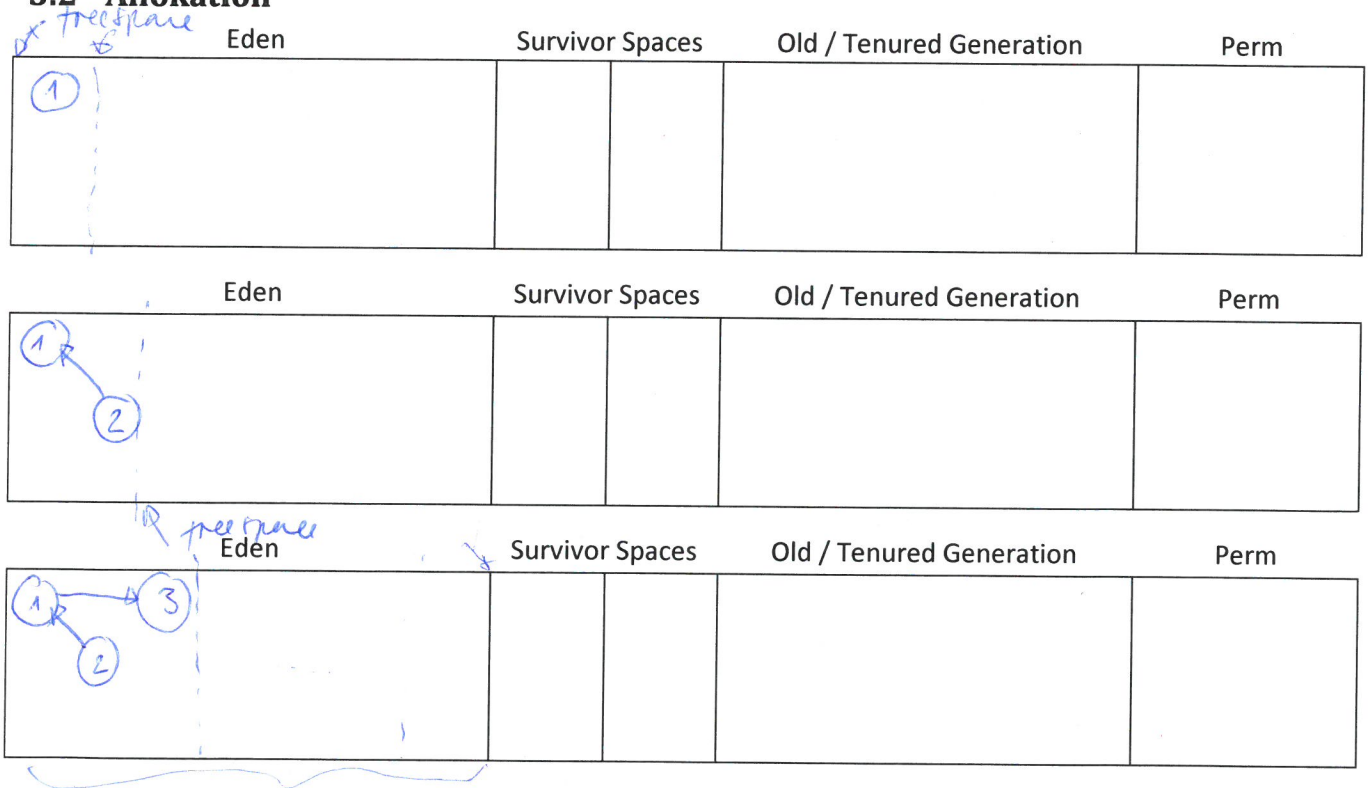
Perm: Die Permanent Generation hält alle Class-Objekte der geladenen Klassen sowie Objekte, welche die JVM für ihre Arbeit benötigt.

Eden: Nur hier werden neue Objekte allokiert.

freespace, Beginn des freien Speichers, Adresse des nächsten Objektes.

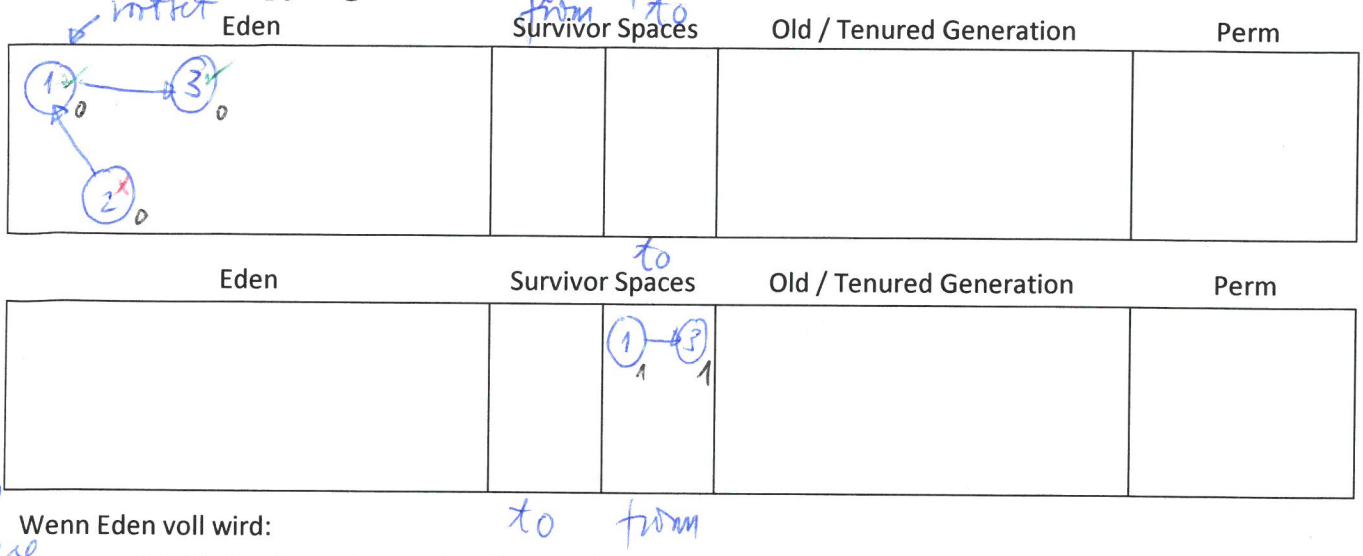
Referenz, die zum Root Set gehört.

3.2 Allokation



Neue Objekte werden immer im Eden Space angelegt. Allokation ist billig, da Speicher fortlaufend vergeben werden kann.

3.3 Mark & Copy Algorithmus



Wenn Eden voll wird:

- Alle Objekte die noch am Leben (= erreichbar) sind werden in einen Survivor Space kopiert. Referenzen müssen angepasst werden.
- Danach gilt der Eden Space wieder als freier Speicher, keine weitere Arbeit nötig!
- Ein kopierender GC wird auch als *Scavenger* bezeichnet (to scavenge = spülen, reinigen)

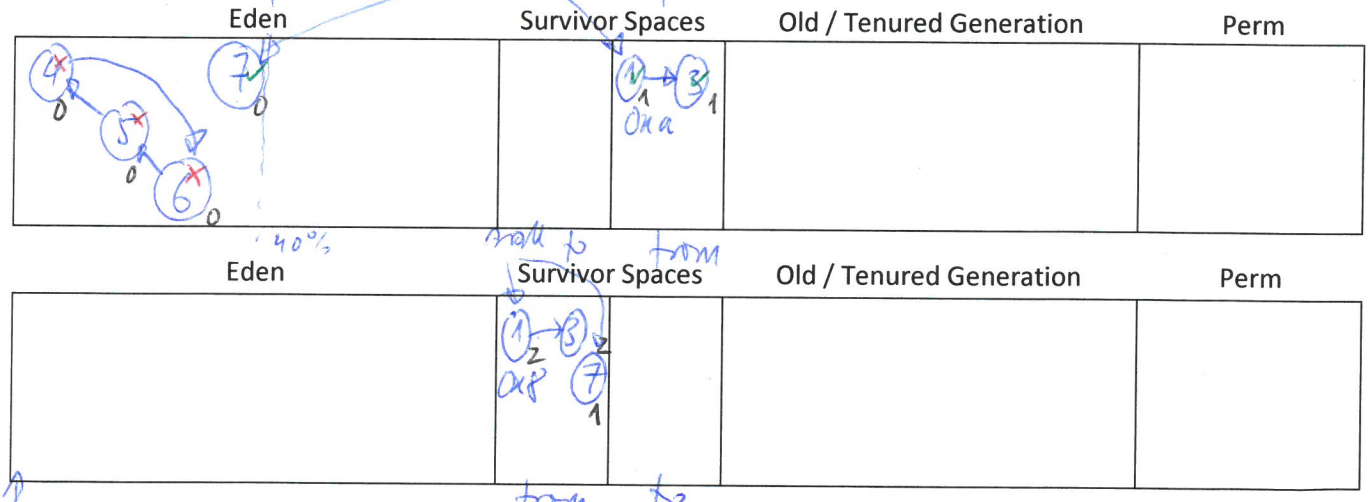
Vorteile

- Die meisten Objekte sind tot. Nur wenige Objekte müssen kopiert werden.
- Kontinuierlicher freier Speicher, schnelle Allokation, keine Fragmentierung

Nachteile

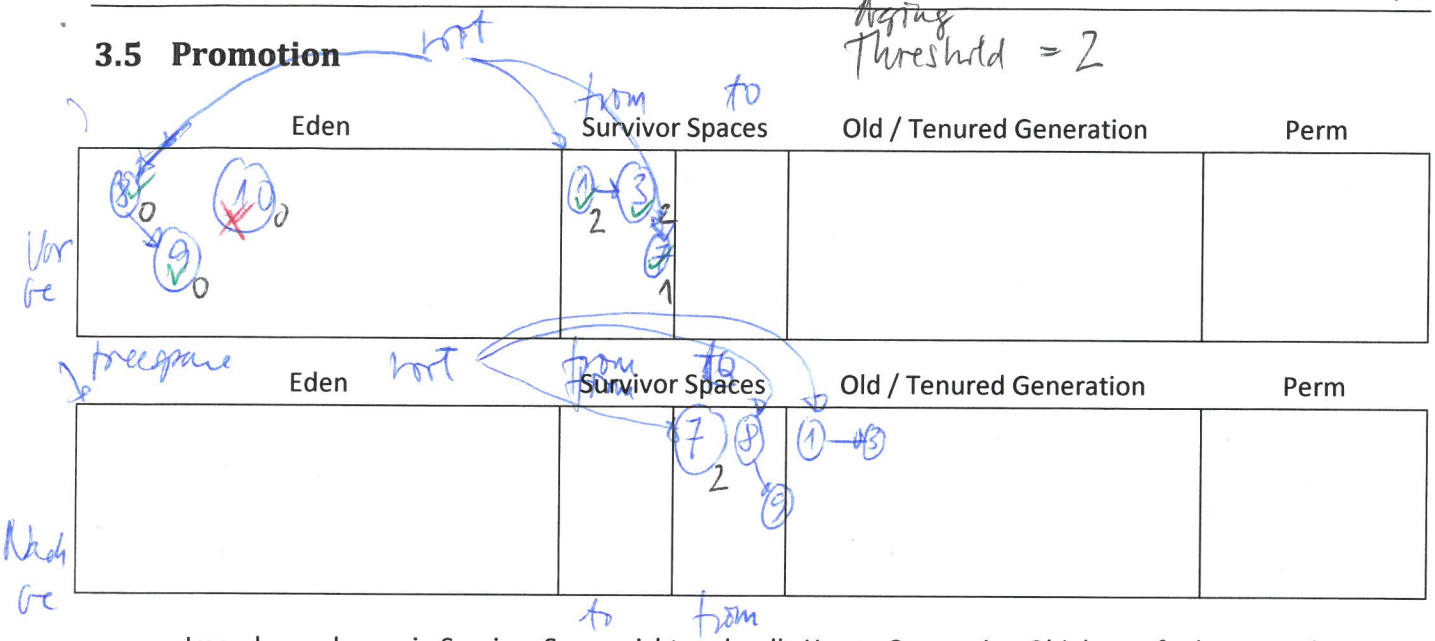
- Grösserer Memory-Footprint da immer ein Survivor Space leer (= ungenutzt) ist.

3.4 Aging



- Einer der Survivor Spaces ist immer leer. GC läuft auf Eden und im gefüllten Survivor Space.
- Danach sind alle „überlebenden" Objekte im vorher leeren Survivor Space. Eden und der alte Survivor Space gelten als freier Speicher.
- Überlebende Objekte werden einige Male zwischen den Survivor Spaces hin und her kopiert (Aging). Weil in der Old Generation viel Aufwand für die GC betrieben werden muss, will man so verhindern, dass junge und mittelalte Objekte zu schnell in die Old Generation kommen.
- Wie oft Objekte zwischen den Survivor Spaces kopiert werden, wird bestimmt durch
  - Grösse des Survivor Space (beide sind genau gleich gross)
  - Anzahl Objekte in Eden und im alten Survivor Space
  - Age Threshold (konfigurierbar)

3.5 Promotion

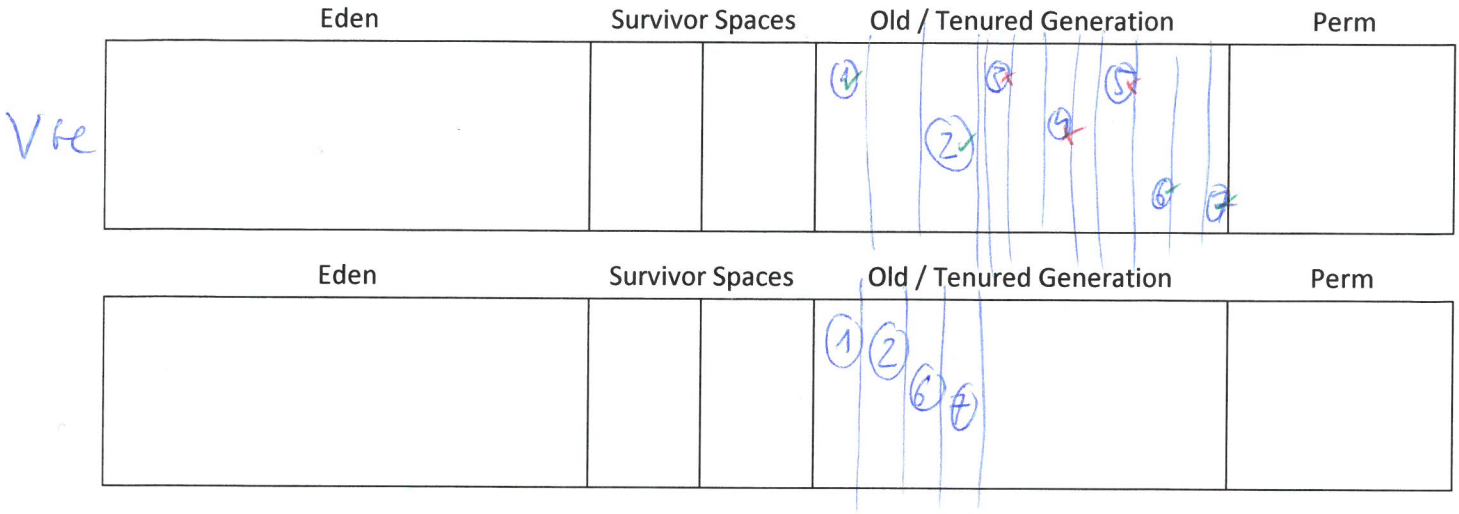


- Irgendwann kann ein Survivor Space nicht mehr alle Young Generation Objekte aufnehmen. Es kommt zur Promotion in die Old Generation.
- Die ältesten Objekte werden in die Old Generation kopiert.

3.6 Minor Garbage Collection

- Die bisher beschriebenen Schritte (3.2- 3.5) werden als *Minor Garbage Collection* bezeichnet:
  - Eden → Survivor Space
  - Survivor Space → Survivor Space
  - Survivor Space → Old Generation
- Eine *Full* oder *Major Garbage Collection* wird immer durch eine Minor Garbage Collection ausgelöst.
- Minor Collections kommen häufiger vor als eine Major Garbage Collection
  - das genaue Verhältnis hängt vom Verhalten der Applikation ab und
  - von der Grösse der verschiedenen Heap-Bereiche (Tuning-Potential!)
- Minor Collections benötigen weniger Zeit als Major Collections

3.7 Mark & Compact Algorithmus



Major Collections führen einen Mark & Compact GC auf der Old Generation durch:

- lebende Objekte werden markiert und
- innerhalb der Old Generation kopiert um eine Fragmentierung zu vermeiden.

Vorteile

- Memory Footprint moderat, kein unbenutzter Survivor Space

Nachteile

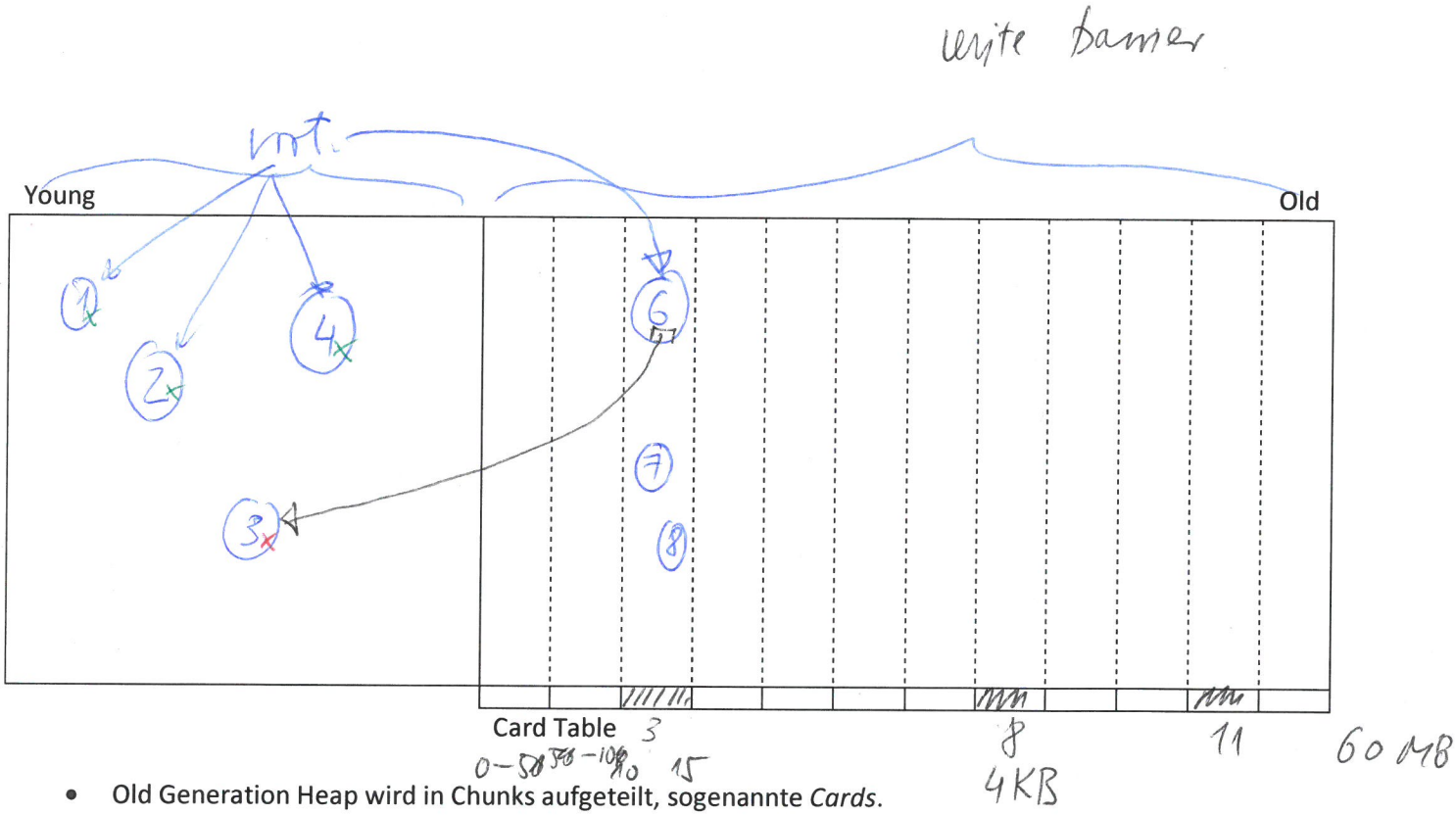
- langsam, da das Kompaktieren aufwändig ist
- Dies bedeutet relativ lange Pausen für die Applikation

Alternative**: Inkrementeller GC**

- Räumt inkrementell nur einen Teil der Old Generation auf.
- Aufwändig, da zusätzlicher Verwaltungsaufwand betrieben werden muss.
- Kürzere Pausen für die Applikation, aber merklich geringerer Durchsatz! (GC läuft häufiger)

3.8 Old-to-Young Referenzen

Ein Problem stellen Referenzen von Objekten aus der Old-Generation auf Objekte aus der Young-Generation dar. In der Mark-Phase eines Minor GC müssen auch Objekte der Old-Generation mit berücksichtigt werden:



- Old Generation Heap wird in Chunks aufgeteilt, sogenannte *Cards*.
- Wird eine Referenz in einem Objekt der Old Generation verändert, so wird die Card in welcher sich das Objekt befindet als *dirty* markiert. Die Markierung wird auch vorgenommen wenn ein Objekt aus dem Survivor Space in die Old Generation kopiert wird, und dieses Objekt noch Referenzen auf junge Objekte hält.
- Während einer Minor Collection, werden in der Mark-Phase nur die dirty Bereiche der Old Generation gescannt um Root-Referenzen zu finden.
- In der Copy-Phase einer Minor Collection ist es wichtig alle Old-to-Young-Referenzen nachzuführen. Auch hierbei hilft die Card Table, sie enthält ja alle Bereiche die eine Old-to-Young Referenz enthalten: