

# Garbage Collection :

Garbage collection in Java, with Animation and discussion of G1 GC

Press **Esc** to exit full screen

## Garbage collection

Java provides automatic memory management through a program called *Garbage collector*.

***"Remove objects that are not used anymore."***

*live object = reachable (referenced by someone else)*  
*dead object = unreachable (not referenced from anywhere)*

**Based on following hypothesis :**  
***Most objects soon become unreachable***  
***References from 'old' objects to 'young' objects only exist in small numbers.***

3:00 / 44:19

Garbage collection in Java, with Animation and discussion of G1 GC

Press **Esc** to exit full screen

## Garbage collection

**Before we start**

- Objects are allocated (eg: `new` ) in the "heap" of java memory. Static members, class definitions(metadata) etc. are stored in "method area"(Permgen/Metaspace)
- Garbage collection is carried out by a daemon thread called "Garbage collector"
- We can not force gc to happen (`System.gc()`)
- When new allocations can not happen due to a full heap you end up with a `java.lang.OutOfMemoryError` heap space and a lot of other headaches :)

5:11 / 44:19



# Garbage collection

## Involves

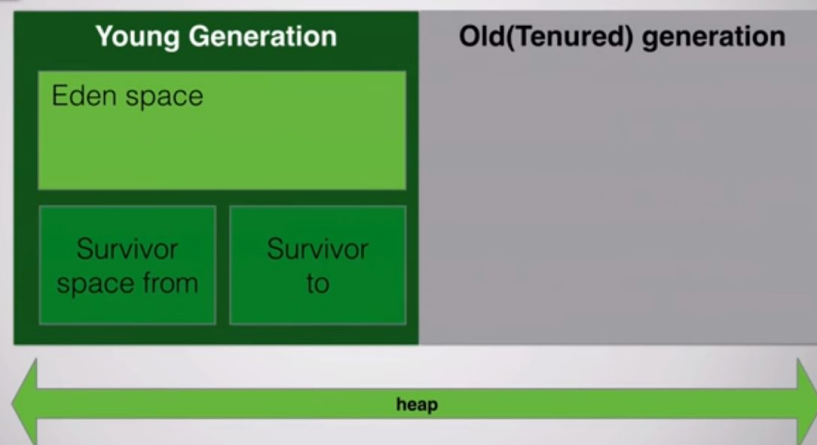
**Mark:** Starts from root node of your application(main), walks the object graph, marks objects that are reachable as live.

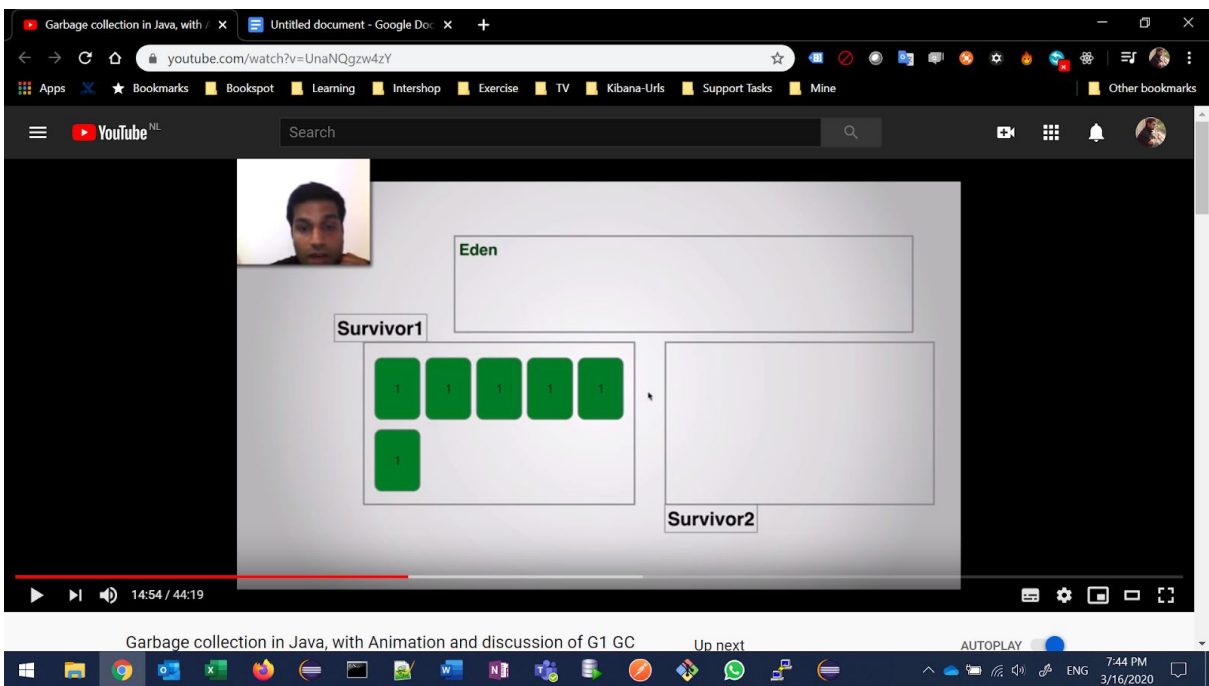
**Delete/sweep:** Delete unreachable objects

**Compacting:** Compact the memory by moving around the objects and making the allocation contiguous than fragmented.



# Generational collectors





Garbage collection in Java, with / X Untitled document - Google Doc X +

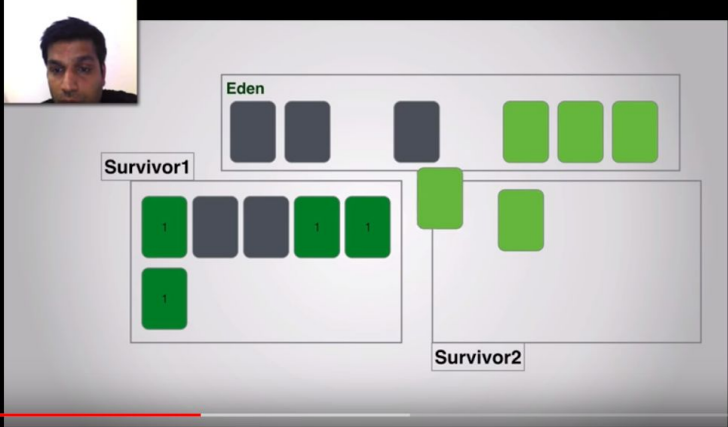
youtube.com/watch?v=UnaNQgw4zY

Apps Bookmarks Bookspot Learning Intershop Exercise TV Kibana-Urls Support Tasks Mine Other bookmarks

YouTube NL Search

15:52 / 44:19

Garbage collection in Java, with Animation and discussion of G1 GC Up next AUTOPLAY 7:45 PM 3/16/2020



The diagram illustrates the memory layout of the G1 garbage collector. It shows three main regions: Eden, Survivor1, and Survivor2. The Eden region contains three dark gray blocks and three green blocks. Survivor1 contains four green blocks, each labeled with the number '1', and one dark gray block. Survivor2 contains two green blocks. A vertical line separates Survivor1 and Survivor2.

Garbage collection in Java, with / X Untitled document - Google Doc X +

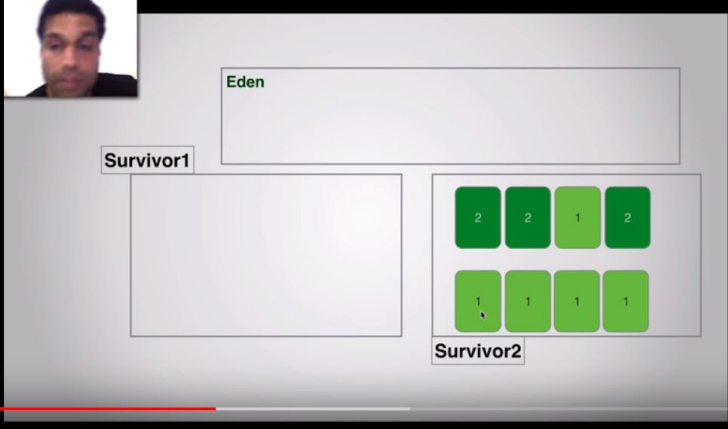
youtube.com/watch?v=UnaNQgw4zY

Apps Bookmarks Bookspot Learning Intershop Exercise TV Kibana-Urls Support Tasks Mine Other bookmarks

YouTube NL Search

16:27 / 44:19

Garbage collection in Java, with Animation and discussion of G1 GC Up next AUTOPLAY 7:45 PM 3/16/2020



The diagram illustrates the memory layout of the G1 garbage collector. It shows three main regions: Eden, Survivor1, and Survivor2. The Eden region is empty. Survivor1 is empty. Survivor2 contains eight green blocks arranged in two rows of four. The top row has blocks labeled '2', '2', '1', and '2'. The bottom row has blocks labeled '1', '1', '1', and '1'. A vertical line separates Survivor1 and Survivor2.

Garbage collection in Java, with Animation and discussion of G1 GC

Up next

AUTOPLAY

7:46 PM 3/16/2020

Garbage collection in Java, with Animation and discussion of G1 GC

Up next

AUTOPLAY

7:46 PM 3/16/2020

Garbage collection in Java, with Animation and discussion of G1 GC

Up next

AUTOPLAY

7:46 PM 3/16/2020

Garbage collection in Java, with Animation and discussion of G1 GC

Up next

AUTOPLAY

7:46 PM 3/16/2020



Garbage collection in Java, with Animation and discussion of G1 GC

Allocation

Young Generation

Promotion -XX:MaxTenuringThreshold

Old Generation

19:29 / 44:19

Garbage collection in Java, with Animation and discussion of G1 GC Up next AUTOPLAY 7:46 PM 3/16/2020

Garbage collection in Java, with Animation and discussion of G1 GC

Press **Esc** to exit full screen

# Performance

## Responsiveness/latency

How quickly an application response with a requested piece of data. Examples:

- How quickly a desktop UI responds to an event
- How fast a website returns a page
- How fast a database query is returned

For applications that focus on responsiveness, large pause times are not acceptable. The focus is on responding in short periods of time.

21:28 / 44:19

# Performance

## Throughput

Throughput focuses on maximizing the amount of work by an application in a specific period of time. Examples of how throughput might be measured include:

The number of transactions completed in a given time.

The number of jobs that a batch program can complete in an hour.

The number of database queries that can be completed in an hour.

High pause times are acceptable for applications that focus on throughput. Since high throughput applications focus on benchmarks over longer periods of time, quick response time is not a consideration.

21:34 / 44:19

# Garbage Collectors

## A serial collector

Basic garbage collector that runs in single thread, can be used for basic applications.

## A concurrent collector

A thread that performs GC along with application execution as the application runs, does not wait for the old generation to be full - Stops the world only during mark/re-mark.

## A parallel collector

Uses multiple CPUs to perform GC. Multiple threads doing mark/sweep etc. Does not kick in until heap is full/near-full. "stops-the-world" when it runs.

22:24 / 44:19



# Garbage Collectors

## Use concurrent collector(CMS) when

- There is more memory
- There is high number of CPUs
- Application demands short pauses

## Use a parallel collector when

- There is less memory
- There is lesser number of CPUs
- Application demands high throughput and can withstand pauses.

26:17 / 44:19



Press **Esc** to exit full screen

# Garbage Collectors

## G1 garbage collector (Garbage - first)

Latest entry in to GC scene ( officially available in 1.7 u4)

It straddles the young-tenured generation boundary as it divides heap in to different regions and during a GC it can collect a sub-set of regions. It dynamically selects a set of region to act as young generation in next gc cycle. Regions with most garbage(unreachable) will be collected first.

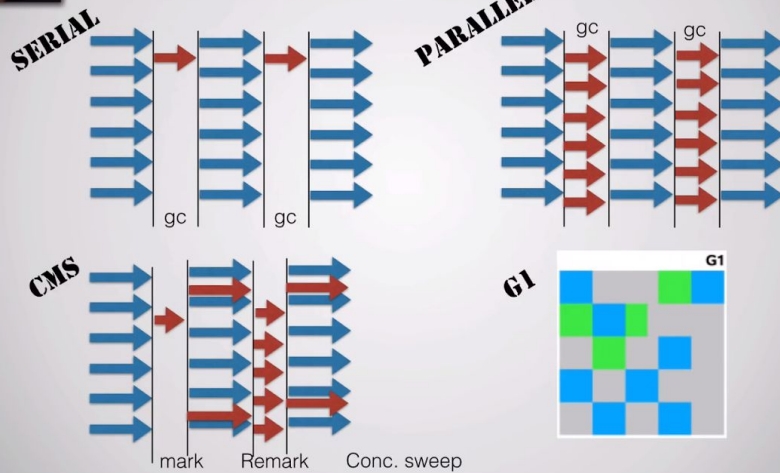
- More predictable(tunable) GC pauses
- Low pauses with fragmentation
- Parallelism and concurrency together
- Better heap utilization

27:13 / 44:19





# Garbage Collectors



## Selecting Garbage collector

| Option                         | Description  |
|--------------------------------|--|
| <b>-XX:+UseSerialGC</b>        | Single-threaded gc on young and old generation. To be used only on small heaps |
| <b>-XX:+UseParallelGC</b>      | Young generation uses parallel gc, Old generation uses single-threaded gc      |
| <b>-XX:+UseParallelOldGC</b>   | Both young and old generations have multi-threaded GC                          |
| <b>-XX:+UseParNewGC</b>        | Multi-threaded young generation garbage collector                              |
| <b>-XX:+UseConcMarkSweepGC</b> | Enables concurrent collector. Autoenables ParNewGC by default.                 |
| <b>-XX:+UseG1GC</b>            | Use G1   |

Defaults: 1.6 = Parallel, 1.7 G1



Press **Esc** to exit full screen

## Tune the Heap

- Xmsvalue
- Xmxvalue (def: 256m)
- XX:NewRatio=ratio (2 means  $\frac{1}{3} Y + \frac{2}{3} T$ )
- XX:NewSize=size
- XX:MaxNewSize=size
- XX:PermSize
- XX:MaxPermSize (def: 64m)



## GC logging

- verbose:gc
- XX:+PrintGCDetails
- Xloggc:gc.log

Very useful if gc is the suspect. Use graphical tool to analyze the logs.



## View GC

jvisualvm with visual gc plugin

