Avoiding memory leaks: Memory allocated but not references to it so it does not get deallocated

1. Reference counts (active garbage collection)

Each allocated object has a count of how many references refer to it

When the count goes to zero, the memory is deallocated

Disadvantage: Cycle references, you need background updating with every reference operation, it deallocates immediately, which could be a cost but at the wrong time

1. Mark and sweep: (lazy or passive garbage collection)  
   Pause execution of the program (at a regular interval, when heap runs out)
   1. mark all heap objects as unused
   2. go to all references in the stack and mark those objects as used
   3. continue doing a DFS/BFS from the “used” objects following all references they have and mark those objects as used
   4. when DFS/BFS termination anything still unused is deallocated

BFS/DFS ?????

Java’s Garbage Collector

Heap is made up of Eden 🡪 Young 🡪 Survivors 🡪 Old

All new objects are placed in Eden. When Eden gets full, mark and sweep runs

1. Only young objects get deallocated at first
2. Those still used in Eden are moved to active survivors  
   Those still used in active survivor are moved to inactive survivor  
   Anything that has survived a long time is moved to old
3. The role of active and inactive is flipped and all of Eden and inactive survivor are cleared