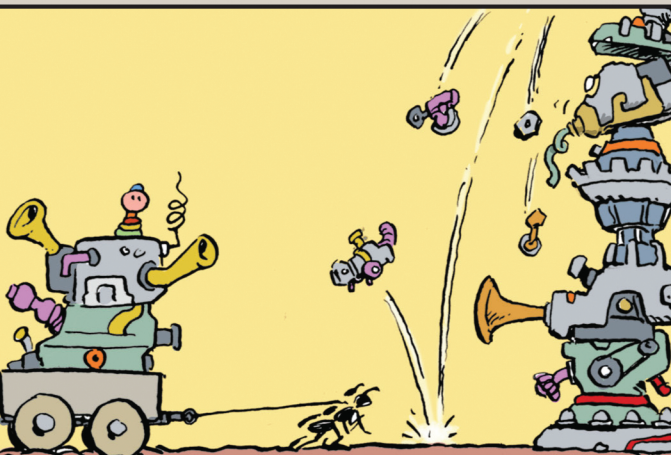


MEMORY MANAGEMENT IN .NET

WHEN YOU CREATE AN OBJECT IN .NET, IT ENTERS A MEMORY SPACE ALREADY ALLOCATED INTO FOUR HEAPS, KNOWN AS GENERATIONS 0, 1, AND 2, PLUS THE LARGE OBJECT HEAP.

BROUGHT TO YOU BY
RED GATE SOFTWARE'S
ANTS MEMORY PROFILER!

A SMALL OBJECT ($\leq 85,000$ bytes*) HEADS FOR GENERATION 0, WHERE SPACE IS MANAGED BY A BUSY LITTLE ROUTINE CALLED THE GARBAGE COLLECTOR.

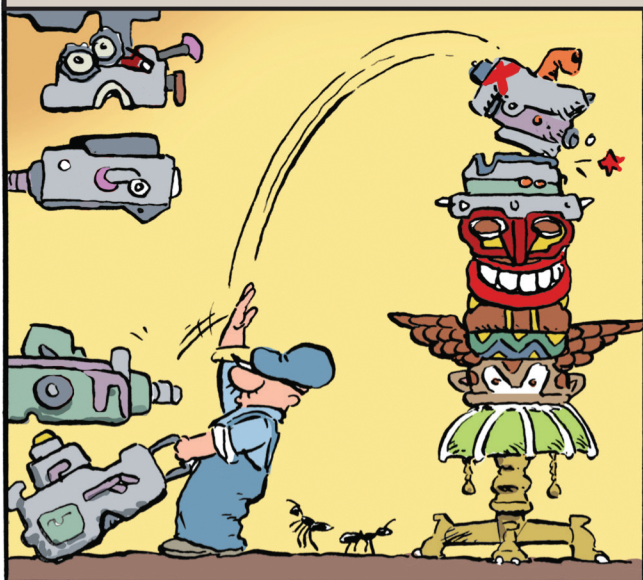


*WITH CERTAIN EXCEPTIONS. FLOAT AND DOUBLE ARRAYS $\geq 4K$ ARE ALSO CONSIDERED LARGE

THE GARBAGE COLLECTOR (GC FOR SHORT) SCANS GENERATION 0 FOR LIVE OBJECTS, MEANING THOSE STILL REFERENCED BY SOME OTHER OBJECT.



THEN GC COPIES ALL LIVE OBJECTS FROM GENERATION 0 TO GENERATION 1..



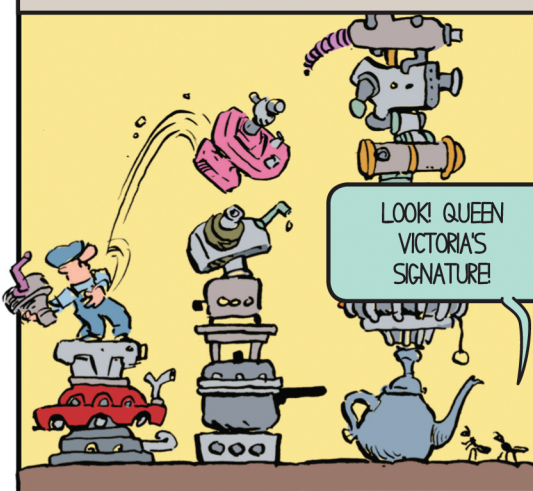
...AND SMASHES GENERATION 0 DOWN BY WRITING NEW OBJECTS OVER THE DEAD STUFF, WHICH DISAPPEARS



AS WITH GENERATION 0, SO WITH GENERATION 1: OLD SURVIVORS GO TO GENERATION 2, AND THE REST IS SQUASHED FLAT



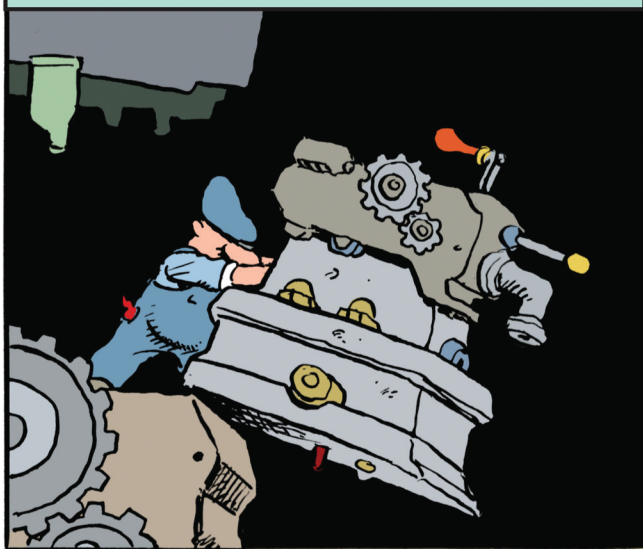
RESULT: GENERATION 2 HAS THE OLDEST OBJECTS... GENERATION 1 HOLDS THE MIDDLE-AGED MATTER... AND GENERATION 0 SEETHES WITH ACTIVITY.



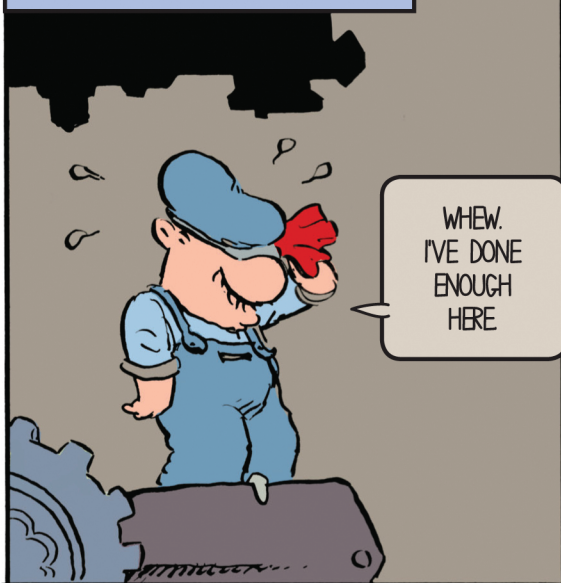
WARNING: OBJECTS MAY BE DEAD IN ALL BUT NAME: NO LONGER IN USE, BUT STILL REFERENCED SOMEWHERE IN THE PROGRAM. GC IS BLIND TO THIS STUFF, WHICH WILL PERSIST IN MEMORY FOREVER.



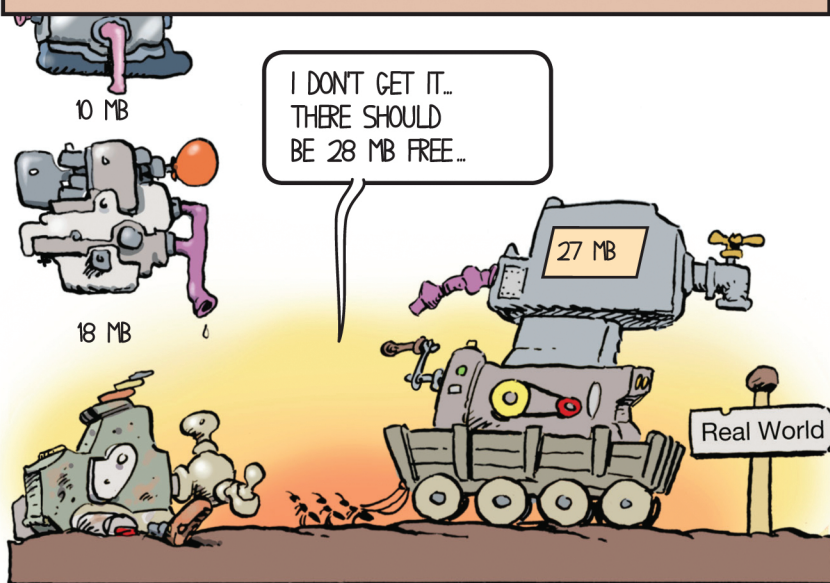
WHAT ABOUT THE LARGE OBJECT HEAP, WHERE THE MONSTERS LIVE? GC PUSHES DEAD OBJECTS OUT OF THERE, TOO.



BUT THIS HEAP IS NEVER DEFRAGMENTED.



THIS CAN LEAD TO MASSIVE LEAKS AND SURPRISING OUT-OF-MEMORY ERROR MESSAGES.



SO: EVEN THOUGH .NET RELIEVES PROGRAMMERS OF MANY ROUTINE (AND SOMETIMES OVERLOOKED) CHORES, IT ISN'T PERFECT!

HEY! WHADDA YA WANT? I'M ONLY ALMOST HUMAN

LET'S SEE... I WANT... UM... HOW ABOUT A DEFRAG LOH ROUTINE??

