Garbage Collection
Java & Python

by Melissa Villalovos

Early Memory Management

- In early programming languages, most developers were responsible for all memory management in their programs
- This meant before creating a list or an object, you first needed to allocate the memory for your variable
- After you were done with your variable, you then needed to deallocate it to " free" that memory for other users

This led to two problems:

- Forgetting to free your memory. If you don't free your memory when you're done using it, it can result in memory leaks. This can lead to your program using too much memory over time. For long-running applications, this can cause serious problems.
- Freeing your memory too soon. The second type of problem consists of freeing your memory while it's still in use. This can cause your program to crash if it tries to access a value in memory that doesn't exist, or it can corrupt your data. A variable that refers to memory that has been freed is called a dangling pointer.

Automatic Garbage Collection (GC)

- These problems were undesirable, so newer languages added automatic memory management
- Now that this process is automatic, programmers no longer needed to manage memory themselves
- Instead, the runtime handles this for them

Automated Garbage Collection in Java

- Java programs compile to bytecode that can be run on a Java Virtual Machine (JVM)
  - This is what controls the Garbage Collector
- JVM also decides when to perform the garbage collection
- We can request the JVM to run the garbage collector but there is no guarantee under any conditions that the JVM will comply
- JVM runs the garbage collector if it senses that memory is running low
- When Java programs request for the garbage collector, the JVM usually grants the request in short order
- However, it does not make sure that the requests accept

Implementation of GC in Java

- When a Java program runs, the automatic memory management is controlled by a thread known as **Garbage Collector**
- Java provides two methods **System.gc()** and **Runtime.gc()** that sends requests to the JVM for garbage collection
- If the Heap Memory is full, the JVM will not allow a new object to be created and will show the error **java.lang.Out OfMemoryError**
- When GC removes an object from the memory, first, the GC thread calls the **finalize()** method of that object and then removes it
- An object becomes eligible if it is not used by any program or thread or any static references or if its references is null

- We override the finalize method of the object class for cleanup processing then we utilize the System.gc() function that will request the JVM to call the Garbage Collector and hence destroys the object
- The output authenticates the working of garbage collection

```
    protected void finalize(){
        System.out.println("Object destroyed");
    }
}

public class GarbageCollectionExample {
    public static void main(String[] args) {
        Student std = new Student();
        std = null;
        System.gc();
    }
```

Overriding finalize method

Assigning null refernce to the object

Calling gc() method

garbagecollectionexample.GarbageCollectionExample

Output - GarbageCollectionExample (run)

```
run:
Object Created
Object destroyed
BUILD SUCCESSFUL (total time: 1 second)
```

Output

Automated Garbage Collection in Python

- There are a few different methods for automatic garbage collection but one of the more popular methods is reference counting
- With reference counting, the runtime keeps track of all of the references to an object
- When an object has zero references to it, it's unusable by the program code and can be deleted

Implementation of GC in CPython

- CPython is the reference implementation of the Python language and it compiles Python code into byte code before interpreting it
- Whenever you create an object in Python, the underlying C object has both a Python type (such as a list or function) and a reference count
- At a very basic level, a Python objects reference count is incremented whenever the object is referenced, and it's decremented when an object is dereferenced
- If an objects reference count is 0, the memory for the object is deallocated
- The **sys** module from the Python standard library is used to check reference counts for a particular object
- There are a few ways to increase the reference count for an object such as:
  - Assigning an object to a variable
  - Adding an object to a data structure, such as appending to a list or adding as a property on a class instance
  - Passing the object as an argument to a function

```
>>> import sys
>>> a = 'my-string'
>>> sys.getrefcount(a)
2
```

- Notice that there are two references to our variable **a**
- One is from creating the variable
- The other is when we pass the variable **a** to the **sys.getrefcount()** function

- The reference count of **a** increases when added to a data structures list or dictionary

```
>>> import sys
>>> a = 'my-string'
>>> b = [a] # Make a list with a as an element.
>>> c = { 'key': a } # Create a dictionary with a as one of the value
s.
>>> sys.getrefcount(a)
4
```

Downside to Automatic Garbage Collection

- There are a couple downsides to automatic garbage collection
  - Additional computations and memory will be needed for the program to track all of its references
  - Many programming languages with automatic garbage collection use a "stop-the-world" process where all execution stops while the garbage collector looks for and deletes objects to be collected
- However, since newer computers contain large amounts of RAM, the benefits of automatic garbage collection outweigh the downsides

Conclusion

- Garbage Collection is a memory recovery feature built into programming languages such as Java and Python
- In Java, the garbage collector destroys objects which are no longer used at a specific time
- Python uses the reference count method to handle object life time
  - So an object that has no more use will be immediately destroyed
- In-use objects, or a referenced object, means that some part of your program still maintains a pointer to that object
- Although there are drawbacks to automatic garbage collection, the advantages overcome the disadvantages, especially with today's technology

References

- Tamar Domani, Elliot K. Kolodner, Ethan Lewis, Eliot E. Salant, Katherine Barabash, Itai Lahan, Yossi Levanoni, Erez Petrank, and Igor Yanorer. 2000. Implementing an on-the-fly garbage collector for Java. In Proceedings of the 2nd international symposium on Memory management (ISMM '00). Association for Computing Machinery, New York, NY, USA, 155–166. https://doi.org/10.1145/362422.362484
- Lujing Cen, Ryan Marcus, Hongzi Mao, Justin Gottschlich, Mohammad Alizadeh, and Tim Kraska. 2020. Learned garbage collection. In Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL 2020). Association for Computing Machinery, New York, NY, USA, 38–44. https://doi.org/10.1145/3394450.3397469
- Thomas Perl. 2012. Python Garbage Collector Implementations CPython, PyPy and GaS. Seminar on Garbage Collection WS2011/12, TU Wien.
- Ulan Degenbaev, Jochen Eisinger, Kentaro Hara, Marcel Hlopko, Michael Lippautz, and Hannes Payer. 2018. Cross-component garbage collection. Proc. ACM Program. Lang. 2, OOPSLA, Article 151 (November 2018), 24 pages. https://doi.org/10.1145/3276521
- Ulan Degenbaev, Jochen Eisinger, Manfred Ernst, Ross McIlroy, and Hannes Payer. 2016. Idle-Time Garbage-Collection Scheduling: Taking advantage of idleness to reduce dropped frames and memory consumption. Queue 14, 3 (May-June 2016), 35–52. https://doi.org/10.1145/2956641.2977741