

# Transparency, HW1, & Garbage



With Pizza

ASSOCIATION FOR COMPUTING  
MACHINERY AT UAB PRESENT:

# LEETCODE NIGHT


ENJOY DINNER AND SOLVE LEETCODE  
PROGRAMMING PROBLEMS TOGETHER!

WEDNESDAY FEB 5  
4:30PM - 5:30PM  
UH-1008 : FIRST FLOOR UH



# Lab vs Lecture

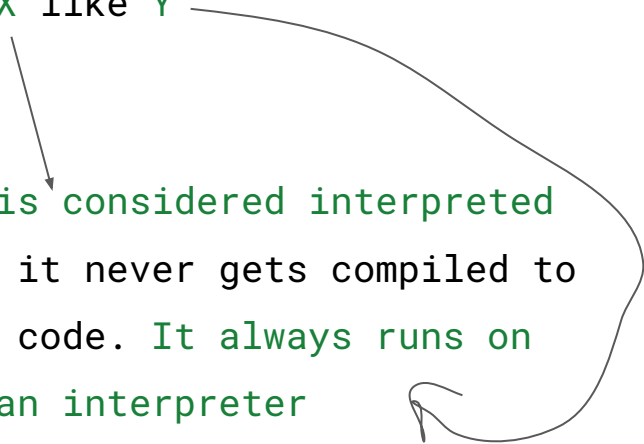
## Lecture:

- High level details
- Learn **X** is like **Y**  

- ie: **Python** is **interpreted**

We're always open to feedback

[mikegtr@uab.edu](mailto:mikegtr@uab.edu) / [akshar20@uab.edu](mailto:akshar20@uab.edu)

## Lab:

- Lower level details
- Why is **X** like **Y**
- How is **X** like **Y**  

- **Python** is **considered interpreted**  
because it never gets compiled to  
machine code. **It always runs on**  
**top of an interpreter**

# How we grade

## Labs

- Answering the questions in writing - typically with a class approved answer
- We worry less about code implementation and more about code running
- Turnaround: ~10 minutes
- No late submissions (hw too)

## Homework:

- Depth in writing
  - Did you explain everything you needed to explain?
  - Did you talk about your choices/design?
- Code implementation
  - Did you follow good practices?
  - Are your files, functions, variables named correctly and well?
  - Is your memory managed well?
- Turnaround: ~1-3 weeks

# How we grade - Continued

## Exams

- Multiple choice on canvas -  
Automatically graded
- Free response - seeing if you answered  
the question
- Exams are worth 70% of your grade

# Homework 1

- We're looking for OOP understanding
- Use multiple files
  - For C++, use headers and cpp files
  - For python, use multiple python files
- We're pretty flexible on what you implement - talk about your choices in your writing

Can be the same implementation or different

\*\*\* We have provided you with a broad overview of how the e-commerce app is structured and how it functions. Use this as a foundation to identify the key components and relationships, but feel free to expand on the details. You can decide the specific properties and functionalities for each class based on the story, as long as they align with the overall purpose of the app. Additionally, you are welcome to introduce new categories, subcategories, or product types to enhance the system's functionality or showcase your creativity. Ensure your implementation reflects the core principles of object-oriented programming and results in a cohesive, extendable design.

# Garbage Collection - What is it

- The automation of memory deallocation
  - Freeing up memory occupied by objects that are no longer reachable
- Multiple methods:
  - Reference Counting
    - Keeping track of the references pointing to an object - when zero then the object can be deallocated
  - Mark & Sweep
    - Pauses a program to **trace** reachable objects starting at the root references
      - objects not reached are marked as garbage and deallocated
  - Generational Collection
    - Young objects are collected more frequently, while older objects are collected less often.

# Garbage Collection - How - C++

- C++ bridges that gap between having manual memory management and having more powerful automatic tools like Resource Acquisition Is Initialization (RAII) and Smart Pointers

## RAII

- Scope based memory management
- When you create an object - the constructor is called immediately
- When you exit the scope the destructor is automatically invoked
- **Exception Safety:** When an exception is thrown then all destructors are called

## Smart Pointers

- Wraps raw pointers and manages their lifetime through RAII
- `std::unique_ptr` manages exclusive ownership and frees memory when out of scope
- `std::shared_ptr` reference counts - frees memory when the last owner is destroyed



# Garbage Collection - How - C

- C has no garbage collection whatsoever - you are the garbage collector
- Fully up to you to manage memory using `malloc` and `free`

# Garbage Collection - Example - C++

```
1  #include <memory>
2  #include <iostream>
3
4  struct Resource {
5      Resource() { std::cout << "Resource acquired\n"; }
6      ~Resource() { std::cout << "Resource released\n"; }
7  };
8
9  int X() {
10     std::shared_ptr<Resource> ptr1 = std::make_shared<Resource>();
11     {
12         std::shared_ptr<Resource> ptr2 = ptr1; // Reference count increases
13     } // ptr2 goes out of scope; reference count decreases
14     // Resource is released when ptr1 goes out of scope and the count drops to zero
15 }
16
17
18 int Y() {
19     std::unique_ptr<Resource> ptr = std::make_unique<Resource>();
20     // Resource is automatically released when ptr goes out of scope
21 }
```

# Reflection

Scope based memory management is great!

# Garbage Collection - How - Java

- **Heap Allocation:** All Java objects that are created using the `new` operator get allocated to the heap. The JVM has its own GC that `periodically` scans the heap to determine which objects are still in use
- **Reachability Analysis:** The GC starts at root references (local variables, static variables, JNI References) then traces the object graph to mark all objects that can be reached
- **JNI References:** Java Native Interface, The JVM's interface that interacts with native (machine) code which are typically written in C or C++. These are scope based memory managed

# Garbage Collection - How - Java

- **Generational Collection:** Modern JVM's typically involve a generational memory management approach - an approach based on the observation that most objects are short-lived. This is split into three parts:
  - **Eden Generation:**

New objects are allocated in the eden generation. Garbage collections in this area (often called minor collections) occur frequently and are optimized for speed.
  - **Survivor Generation:**

Objects that survive multiple minor collections are promoted to the old generation, where collections (major collections) happen less frequently but typically take longer.
  - **Metaspace Generation:**

This area stores metadata about classes and methods. It is managed separately and has its own GC behavior.

# Garbage Collection - How - Java

- **Other GC Implementations:** Depending on the JVM version and config you can have a couple other GC algorithms:
  - **Serial GC:**  
A single-threaded collector primarily used in small applications or single-processor machines.
  - **Parallel GC:**  
Uses multiple threads to perform garbage collection, improving throughput in multi-core systems.
  - **Concurrent Mark Sweep (CMS):**  
Aims to minimize pause times by doing most of the work concurrently with the application threads.
  - **G1 (Garbage-First) Collector:**  
Divides the heap into regions and prioritizes collecting regions with the most garbage, balancing pause times and overall throughput.
  - **ZGC and Shenandoah:**  
More recent collectors designed to provide very low pause times even for large heaps.

# Garbage Collection - When - Java

## Automatic and Non-Deterministic:

The JVM schedules garbage collection automatically based on various factors such as heap occupancy, allocation rates, and tuning parameters. Unlike RAII in C++, Java's garbage collection is non-deterministic; you cannot predict exactly when an object will be collected.

You can use `System.gc()` to “hint” at the JVM to collect memory - chances are it won't listen to you

## Stop-the-World Events:

Many GC algorithms involve “stop-the-world” pauses, during which the application's execution is suspended to perform the garbage collection. Modern collectors aim to reduce the duration and frequency of these pauses.

# Garbage Collection - Options - Java

## Tuning and Observability

- **JVM Options:**

Developers can tune the garbage collection behavior by specifying JVM options. These options can control the size of the heap, the thresholds for triggering collections, and the choice of the GC algorithm.

- **Monitoring Tools:**

Tools such as Java VisualVM, JConsole, and command-line utilities can help monitor GC activity, analyze pause times, and identify performance bottlenecks.

- **-Xms, -Xmx:** Places boundaries on the heap size to increase the predictability of garbage collection. The heap size is limited in replica servers so that even Full GCs do not trigger SIP retransmissions. **-Xms** sets the starting size to prevent pauses caused by heap expansion.
- **-XX:+UseG1GC:** Use the Garbage First (G1) Collector.
- **-XX:MaxGCPauseMillis:** Sets a target for the maximum GC pause time. This is a soft goal, and the JVM will make its best effort to achieve it.
- **-XX:ParallelGCThreads:** Sets the number of threads used during parallel phases of the garbage collectors. The default value varies with the platform on which the JVM is running.
- **-XX:ConcGCThreads:** Number of threads concurrent garbage collectors will use. The default value varies with the platform on which the JVM is running.
- **-XX:InitiatingHeapOccupancyPercent:** Percentage of the (entire) heap occupancy to start a concurrent GC cycle. GCs that trigger a concurrent GC cycle based on the occupancy of the entire heap and not just one of the generations, including G1, use this option. A value of 0 denotes 'do constant GC cycles'. The default value is 45.



# Garbage Collection - How - Python

## Reference Counting

- **Immediate Object Management:**

Every Python object has an internal counter that tracks how many references point to it. When a new reference to an object is created, its count increases; when a reference is removed, the count decreases. Once the count reaches zero, the object is immediately deallocated. This approach is deterministic, meaning memory is freed as soon as it is no longer needed.

## Cyclic Garbage Collector

- **Generational Collection:**

The cyclic GC in Python is organized into generations. Newly allocated objects start in the youngest generation.

- **Generation 0:**

The youngest objects reside here, and collections occur frequently. If an object survives several collections in Generation 0, it is promoted to Generation 1.

- **Generation 1 and Generation 2:**

Objects that have survived collections move to older generations. Collections for these generations happen less frequently, reducing overhead. Generation 2 is the oldest and is collected infrequently.

# Garbage Collection - How - Python

## Detection Algorithm:

The cyclic garbage collector uses a form of mark-and-sweep. When a collection is triggered for a generation, the collector:

1. Traverses the object graph starting from objects in that generation.
2. Identifies objects that are part of reference cycles by checking which objects are not reachable from any root reference.
3. Reclaims the memory occupied by these unreachable cyclic structures.

You can adjust threshold values to decide when to run a collection using the `gc` Python library.

# Garbage Collection - How - Python

The `Reference Counter` and `Cyclic Garbage Collector` work hand in hand.

- Reference counting handles most memory management tasks immediately and deterministically.
- The cyclic GC periodically scans the heap to find and clean them up.

# Garbage Collection - del - Python

Some important considerations to keep in mind

- The `del` operator decrements a reference count for the associated object
- Removing a name with `del` does not free up memory if other references to the object exist
- Cython's implementation of `del` is different from PyPy's implementation of `del`
  - PyPy does not include reference counting - PyPy does Tracing Garbage Collection with Generational collection - More Java style
- Since `del` only dereferences - if you have cyclic references then you may have to call `gc.collect()`

# Garbage Collection - gc - Python

Some important considerations to keep in mind

- The `gc` library in Python gives you a lot of control over garbage collection

`gc.enable()`

Enable automatic garbage collection.

`gc.disable()`

Disable automatic garbage collection.

`gc.get_objects(generation=None)`

Returns a list of all objects tracked by the collector, excluding the list returned. If *generation* is not `None`, return only the objects tracked by the collector that are in that generation.

`gc.collect(generation=2)`

With no arguments, run a full collection. The optional argument *generation* may be an integer specifying which generation to collect (from 0 to 2). A [ValueError](#) is raised if the generation number is invalid. The sum of collected objects and uncollectable objects is returned.

The free lists maintained for a number of built-in types are cleared whenever a full collection or collection of the highest generation (2) is run. Not all items in some free lists may be freed due to the particular implementation, in particular [float](#).

# Why is Java's GC better than Python's?

- **Better algorithms:** Java's GC algos are typically more complex yet more performant
  - **Concurrency and parallel collection:** Java's GC has concurrent and multi-threaded implementations
  - **Tuning options:** Java gives you more granular control on the GC setup
  - **JIT Compilation:** Java's JIT compilation allowing Java to be Machine Code allows it to take full advantage of C/C++'s speed
- 
- And more

# Why is Java's GC better than Python's?

Average Java Developer



Oracle Java Devs Bring in ~400k - 1m/yr

Average Python Developer



Python is open-source, the top maintainers probably make ~100-200k/yr

# More reading/Work Cited

Java GC :

<https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

GC Options:

[https://docs.oracle.com/cd/E55119\\_01/doc.71/e55122/cnf\\_jvmgc.htm#WSEAD420](https://docs.oracle.com/cd/E55119_01/doc.71/e55122/cnf_jvmgc.htm#WSEAD420)

Python GC:

<https://docs.python.org/3/library/gc.html>

Python GC:

<https://stackify.com/python-garbage-collection/>



# Credits

First Page Image:

<https://www.usatoday.com/picture-gallery/tech/2018/10/25/red-dead-redemption-2-tour-rockstar-games-open-world-western/1763209002/>

Rich Donald Duck:

[https://www.liveauctioneers.com/item/196278283\\_large-jozza-mixed-media-scrooge-bags-of-cash-painting](https://www.liveauctioneers.com/item/196278283_large-jozza-mixed-media-scrooge-bags-of-cash-painting)

Squidward:

<https://static1.cbrimages.com/wordpress/wp-content/uploads/2020/09/homeless-squidward.jpg>