Memory Management in Amethyst (Pseudo-Code)

1. Destructor Chain

pseudo

Copy

```
class Object {
property = "some value"
nested = new OtherObject()
destructor() {
// Free unused properties
internal_free_not_consumed()
// Free nested object if not referenced elsewhere
if nested != null AND NOT is_referenced_elsewhere(nested):
free(nested)
}
}
```

2. Reference-Based Reading

pseudo

Copy

```
// Reading a value is done by reference
object t = new Object()
t.property = "some value"
global_value = t.property // global_value references t.property
print(global_value) // Output: "some value"
```

3. Copy-on-Write

pseudo

Copy

```
// Writing a value depends on the case
object t = new Object()
t.property = 3
global_value = t.property // global_value references t.property
// Modify global_value (copy-on-write)global_value = 2 + 4 // global_value now stores its own data (7)
// t.property is no longer referenced by global_value
```

4. Scope-Based Freeing

pseudo

Copy

```
global_value = 0
{
object t = new Object()
t.property = 2
global_value = t.property // global_value references t.property
} // t goes out of scope, destructor is called
// t.property is still referenced by global_value
print(global_value) // Output: 2
// Free global_value when it goes out of scope
free(global_value) // t.property is finally freed
```

Key Concepts

Destructor Chain:

When an object goes out of scope, its destructor is called.

The destructor frees unused properties and nested objects (if not referenced elsewhere).

Reference-Based Reading:

Reading a property (e.g., global_value = t.property) creates a reference to the original

data.
No duplication occurs during reading.
Copy-on-Write:Writing to a referenced value (e.g., global_value = 2 + 4) creates a new instance of the
data.
The original data (e.g., t.property) is no longer referenced.
Scope-Based Freeing:
Objects are freed when they go out of scope.
Properties that are still referenced elsewhere are not freed until all references are gone.
Example Workflow
pseudo
Copy

```
// Global variable
global_value = 0
{
// Create an object
object t = new Object()
t.property = 2
// Reference t.property
global_value = t.property // global_value references t.property
// Modify global_value (copy-on-write)
global_value = 2 + 4 // global_value now stores its own data (6)
// t.property is no longer referenced by global_value
} // t goes out of scope, destructor is called
// t.property is still referenced by global_value
print(global_value) // Output: 6
// Free global_value when it goes out of scope
free(global_value) // t.property is finally freedSummary
```

Reading: Done by reference (no duplication).
Writing: Copy-on-write (new instance if modified).
Destructor Chain: Automatically frees unused properties and nested objects.
Scope-Based Freeing: Objects are freed when they go out of scope, but properties are preserved if still referenced.
This system ensures efficient memory management while minimizing duplication and avoiding memory leaks. It’s simple, predictable, and well-suited for a lightweight scripting language like Amethyst!