C++ OOP Workshop: Using Static Attributes to Count Instances

El Amine Bechorfa

Part 3: Counting Instances with Static Attributes

In this section, we will explore the use of **static attributes** to count the number of instances of a class. We will use constructors and destructors to modify a shared counter, helping us understand how object lifecycles work in C++.

—

Concept Recap

- **Static Attributes**: Belong to the class itself, not to any individual object.
- **Constructors**: Initialize objects and can modify shared attributes like counters.
- **Destructors**: Clean up resources when objects are destroyed and can adjust shared counters.

—

Problem Statement

We want to keep track of how many `Creature` objects are currently alive in a game. We will use a static counter in the `Creature` class to count how many objects have been created and destroyed.

—

Questions Before Coding

- **What is a static attribute, and why is it useful here?**
- **How can constructors and destructors be used to modify a static counter?**
- **What happens if we forget to decrement the counter in the destructor?**

—

Solution: Implementing Static Attributes

Complete Code:

```cpp
#include <iostream>
using namespace std;

class Creature {
private:
    int pv;     // Health Points
    int pa;     // Attack Points
    int niveau; // Level

public:
    static int cpt; // Static counter for the number of creatures

    // Default Constructor
    Creature() : pv(5), pa(1), niveau(1) {
        cpt++;
        cout << "Creature created (default constructor). Count: " << cpt << endl;
    }

    // Parameterized Constructor
    Creature(int pv, int pa, int niveau) : pv(pv), pa(pa), niveau(niveau) {
        cpt++;
        cout << "Creature created (parameterized constructor). Count: " << cpt << endl;
    }

    // Copy Constructor
    Creature(const Creature &c) : pv(c.pv), pa(c.pa), niveau(c.niveau) {
        cpt++;
        cout << "Creature copied. Count: " << cpt << endl;
    }

    // Destructor
    ~Creature() {
        cpt--;
        cout << "Creature destroyed. Count: " << cpt << endl;
    }

    // Display method
    void affiche() const {
        cout << "Creature - PV: " << pv << " PA: " << pa << " Niveau: " << niveau << endl;
    }

    // Static method to get the current count
    static int getCount() {
        return cpt;
    }
};

// Initialize the static counter
int Creature::cpt = 0;

void foo() {
    Creature c3;
    cout << "Inside foo - Count: " << Creature::getCount()
        << endl;
}

void test1() {
    cout << "Test 1:\n";
    Creature c1;
    Creature c2(10, 2, 2);
    cout << "Count after creating c1 and c2: " << Creature::
        getCount() << endl;
    foo();
    cout << "Count after foo: " << Creature::getCount() <<
        endl;
}

void test2() {
    cout << "Test 2:\n";
    Creature c1;
    Creature c2(10, 2, 2);
    Creature c3(c2); // Copy constructor
    cout << "Count after copying c2 to c3: " << Creature::
        getCount() << endl;
}

int main() {
    test1();
    test2();
    return 0;
}
```

—

Detailed Explanation

- **Static Attribute** (`cpt`): Tracks the number of `Creature` instances.
- **Constructors**:

  - Increment the counter when a new `Creature` is created.
  - Different constructors (default, parameterized, copy) all increment the counter.

- **Destructor**:
  - Decrements the counter when an object is destroyed.
  - Ensures the count reflects the correct number of live objects.

—

Expected Output

```
Test 1:
Creature created (default constructor). Count: 1
Creature created (parameterized constructor). Count: 2
Count after creating c1 and c2: 2
Creature created (default constructor). Count: 3
Inside foo - Count: 3
Creature destroyed. Count: 2
Count after foo: 2
Creature destroyed. Count: 1
Creature destroyed. Count: 0

Test 2:
Creature created (default constructor). Count: 1
Creature created (parameterized constructor). Count: 2
Creature copied. Count: 3
Count after copying c2 to c3: 3
Creature destroyed. Count: 2
Creature destroyed. Count: 1
Creature destroyed. Count: 0
```

—

Key Takeaways

- Static attributes allow sharing data across all instances of a class.
- Constructors and destructors can modify shared data, like counters.
- Proper management of static attributes ensures accurate tracking of object lifecycles.

—

Questions for Reflection

- How would you modify the code to count only specific types of creatures?
- What would happen if we didn't decrement the counter in the destructor?
- Can you think of a real-world scenario where static attributes are useful?