PRACTICA 2: PROGRAMACION ORIENTADA A OBJETOS

Es un paradigma de programación que utiliza "objetos" para diseñar aplicaciones y programas informáticos.

CONCEPTOS BASICOS

- clase:

Una clase es una plantilla o un molde para crear objetos. Define un conjunto de atributos (datos) y métodos (funciones) que los objetos de esa clase tendrán. Por sí misma, una clase no ocupa memoria; es solo la definición.

Ej.

```
class MemoryManager
{
public:

    static MemoryManager& getInstance();
    MemoryManager(const MemoryManager&) = delete;
    void operator=(const MemoryManager&) = delete;
    void trackHeapAllocation(void* ptr, size_t size);
    void trackHeapDeallocation(void* ptr);
    void trackStackAllocation();
    void trackStackDeallocation();
    void displayUsage() const;

private:

    MemoryManager() = default;
    ~MemoryManager() = default;

    size_t m_heap_allocations = 0;
    size_t m_heap_deallocations = 0;
    size_t m_stack_allocations = 0;
```

```
    size_t m_stack_deallocations = 0;

    std::map<void*, size_t> m_heap_records;
};
```

- objeto:

Un objeto es una instancia de una clase. Es una entidad concreta que tiene un estado (sus atributos) y un comportamiento (sus métodos). Cuando se crea un objeto a partir de una clase, se reserva memoria para él.

Ej.

```
MemoryManager& MemoryManager::getInstance() {
    static MemoryManager instance;
    return instance;
}
```

- herencia:

La herencia es un mecanismo que permite a una nueva clase (llamada clase hija o subclase) heredar atributos y métodos de una clase existente (llamada clase padre o superclase). Esto promueve la reutilización de código.

Ej.

```
class DebugMemoryManager : public MemoryManager {
public:
    void logToFile();
};
```

- encapsulamiento:

El encapsulamiento consiste en agrupar datos (atributos) y los métodos que operan sobre esos datos dentro de una misma unidad (la clase). También implica restringir el acceso directo a algunos de los componentes de un objeto, lo que se conoce como ocultación de información.

Ej.

```
private:
    size_t m_heap_allocations = 0;
    std::map<void*, size_t> m_heap_records;
```

- abstracción:

La abstracción consiste en ocultar los detalles complejos de implementación y mostrar al usuario solo la funcionalidad esencial. El objeto expone una interfaz simple sin revelar cómo están implementados

sus mecanismos internos.

Ej.

```
class DebugMemoryManager : public MemoryManager {
public:
    void logToFile();
};
```

- polimorfismo:

El polimorfismo (del griego, "muchas formas") permite que objetos de diferentes clases respondan al mismo mensaje (llamada de método) de maneras específicas para cada clase. Permite tratar a objetos de distintas clases de manera uniforme.

Ej.

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

class Complex {
public:
    int real, imag;

    Complex(int r, int i) :
    real(r), imag(i) {}

    // Overloading the '+' operator
    Complex operator+(const Complex& obj) {
        return Complex(real + obj.real,
        imag + obj.imag);
    }
};

int main() {
    Complex c1(10, 5), c2(2, 4);

    Complex c3 = c1 + c2;
    cout << c3.real << " + i" << c3.imag;
    return 0;
}
```

LENGUAJE C VS PYTHON

Característica	C++	Python
Paradigma	Multiparadigma: soporta programación estructurada, POO y genérica	Multiparadigma: POO, estructurada y funcional
Definición de clases	Se usa la palabra clave `class`. Se deben declarar atributos y métodos explícitamente y definirlos luego si están fuera de la clase.	Se usa `class`. Métodos y atributos se pueden definir directamente dentro de la clase.
Tipado	Estático: es necesario declarar el tipo de cada variable y parámetro.	Dinámico: no es necesario declarar tipos, se determinan en tiempo de ejecución.
Encapsulamiento	Usa modificadores de acceso (`public`, `private`, `protected`) para controlar la visibilidad.	No hay modificadores estrictos; se usa convención: `_atributo` para protegido, `__atributo` para privado.
Constructores	Se definen con el mismo nombre que la clase y no tienen valor de retorno.	Se define con el método especial `__init__`.
Herencia	Se indica al definir la clase hija, por ejemplo: `class Hija : public Padre`.	Se coloca la clase padre entre paréntesis: `class Hija(Padre):`.
Polimorfismo	Se logra mediante funciones virtuales, sobrecarga de operadores o plantillas.	Naturalmente dinámico: cualquier objeto con un método puede responder; también con herencia.
Gestión de memoria	Manual (con `new` / `delete`) o automática con smart pointers.	Automática mediante **recolección de basura** (garbage collector).
Singleton	Se implementa manualmente usando atributos estáticos (como en `MemoryManager`).	Puede implementarse fácilmente usando módulos o metaclases.
Sintaxis	Más estricta y detallada (requiere ; y tipos explícitos).	

CONCLUSION

Ventajas de POO

- **Modularidad:** el código se divide en clases independientes, lo que facilita su comprensión y mantenimiento.
- **Encapsulamiento:** protege los datos internos de los objetos, evitando modificaciones indebidas y errores.
- **Abstracción:** oculta los detalles de implementación y muestra solo lo esencial al usuario.

- **Reutilización:** permite reutilizar código mediante herencia, evitando repetir funcionalidades.
- **Polimorfismo:** diferentes objetos pueden comportarse de distintas maneras ante la misma operación, lo que hace al sistema más flexible.
- **Mantenibilidad y escalabilidad:** facilita agregar nuevas funciones sin afectar el resto del sistema.