**TP N°9**

A- Resolver las siguientes actividades:

1. ¿Qué es el encapsulamiento? ¿Cuáles son los niveles de acceso a miembros de la clase?

Encapsulamiento es uno de los principios fundamentales de la programación orientada a objetos (POO). Consiste en ocultar los detalles internos de una clase y solo exponer aquellos elementos que son necesarios para el uso externo. Esto se logra controlando el acceso a los datos y métodos de una clase, protegiendo así la integridad de los datos y mejorando la modularidad y mantenibilidad del código.

Niveles de acceso a miembros de la clase en C++:

Public: Los miembros declarados como públicos son accesibles desde cualquier parte del programa.

Protected: Los miembros declarados como protegidos solo son accesibles desde la clase misma, las clases derivadas y las funciones amigas.

Private: Los miembros declarados como privados solo son accesibles desde la clase misma y las funciones amigas.

1. ¿Qué son los getters y setters? ¿Cuál es su propósito principal? ¿Cuándo es conveniente usarlos?

Getters y setters son métodos usados para acceder y modificar los atributos privados de una clase.

Getter: Método para obtener el valor de un atributo privado.

Setter: Método para establecer o modificar el valor de un atributo privado.

Propósito principal: Su propósito principal es encapsular y controlar el acceso a los atributos de una clase y proporcionar una forma controlada de acceder y modificar los datos, permitiendo validaciones y otras lógicas adicionales.

Cuándo es conveniente usarlos:

* Cuando necesitas controlar el acceso a los atributos privados.
* Cuando se requiere realizar validaciones o ajustes al establecer o obtener los valores de los atributos.
* Para mantener la encapsulación y la integridad de los datos.

1. ¿Cuál es la diferencia entre un constructor y un destructor? ¿Cuándo se ejecutan cada uno?

Constructor: Es un método especial que se llama automáticamente cuando se crea una instancia de una clase. Se usa para inicializar los objetos de la clase

se ejecuta al crear un objeto.

Destructor: Es un método especial que se llama automáticamente cuando un objeto de una clase es destruido. Se usa para liberar los recursos que el objeto pueda haber adquirido durante su vida.

Se ejecuta al destruir un objeto, ya sea al salir del ámbito de un objeto de pila, cuando se llama a delete para un objeto de la memoria dinámica o cuando el programa

termina.

1. Realizar un cuadro comparativo entre las características de constructores y destructores

|  |  |  |
| --- | --- | --- |
| Característica | Constructor | Destructor |
| Propósito | Inicializar un objeto | Liberar recursos antes de  destruir un objeto |
| Nombre | Igual al nombre de la clase | Igual al nombre de la clase precedido por Igual al nombre de la clase  precedido por ´ ~ ´ |
| Parámetros | Puede aceptar parámetros | No acepta parámetros |
| Sobrecarga | Puede ser sobrecargado | No puede ser sobrecargado |
| Invocación | Automáticamente al crear un  objeto | Automáticamente al destruir  un objeto |
| Herencia | Invoca al constructor de la  clase base si no se especifica | Invoca al destructor de la  clase base automáticamente |

1. Investigar la diferencia entre definir una función miembro completamente dentro de la definición de la clase o simplemente incluir su declaración en la función y definirla después fuera de la clase. Elaborar un cuadro comparativo incluyendo ventajas, desventajas y cuando usar cada opción.

|  |  |  |
| --- | --- | --- |
| Aspecto | Definición dentro de  la clase | Definición fuera de la clase |
| Ventajas | * Código más conciso y directo * Puede mejorar la legibilidad si es corto | * Mejora la claridad al   separar la interfaz de la implementación   * Puede reducir el tiempo de compilación si el archivo de cabecera es incluido en   varios lugares |

|  |  |  |
| --- | --- | --- |
| Desventajas | * Puede hacer que el archivo de cabecera sea más largo y menos legible * Aumenta el tiempo de compilación si hay muchos cambios en el archivo de cabecera | - Puede hacer que el código sea menos intuitivo al  separar la declaración de la implementación |
| Uso recomendado | - Para funciones muy cortas (ej. getters y setters) | * Para funciones más largas o complejas * Para mantener archivos de cabecera limpios y legibles |

B. Resolver las actividades planteadas en el TP Nº9.

1. Plantear una clase llamada Alumno y definir como atributos su nombre y su edad. En el constructor realizar la carga de datos. Definir otros dos métodos para imprimir los datos ingresados y un mensaje si es mayor o no de edad (edad >=18).
2. Confeccionar una clase que represente un empleado. Definir como atributos su nombre y su sueldo. En el constructor cargar los atributos y luego en otro método

imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a 3000)

1. Implementar la clase Operaciones. Al constructor llegan dos enteros que deben

utilizarse en los métodos para calcular su suma, resta, multiplicación y división. imprimir dichos resultados.

1. Confeccionar una clase llamada Sumatoria que solicite la carga valores enteros por teclado. Finalizar la carga de valores al ingresar el cero. Mostrar en el destructor la

suma de los valores ingresados.

1. Crear una clase "Estudiante" con atributos "nombre", "edad" y "calificación". Añadir métodos get y set para cada atributo. Escribir un programa que cree una instancia de la clase "Estudiante", asigne valores a sus atributos utilizando los métodos set, y luego imprima los valores utilizando los métodos get.

Rta 1:

#include <iostream> #include <string>

using namespace std; class Alumno {

private:

string nombre; int edad;

public:

Alumno(const string& nom, int ed) : nombre(nom), edad(ed) {} void imprimirDatos() const {

cout << "Nombre: " << nombre << endl; cout << "Edad: " << edad << endl;

}

void esMayorDeEdad() const { if (edad >= 18) {

cout << nombre << " es mayor de edad." << endl;

} else {

cout << nombre << " no es mayor de edad." << endl;

}

}

};

int main() {

Alumno alumno1("Juan Perez", 20); alumno1.imprimirDatos(); alumno1.esMayorDeEdad();

return 0;

}

Rta 2:

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

class Empleado { private:

string nombre; double sueldo;

public:

Empleado(const string& nom, double sue) : nombre(nom), sueldo(sue) {} void imprimirDatos() const {

cout << "Nombre: " << nombre << endl; cout << "Sueldo: " << sueldo << endl;

}

void debePagarImpuestos() const { if (sueldo > 3000) {

cout << nombre << " debe pagar impuestos." << endl;

} else {

cout << nombre << " no debe pagar impuestos." << endl;

}

}

};

int main() {

Empleado empleado1("Ana Gomez", 3500); empleado1.imprimirDatos();

empleado1.debePagarImpuestos(); return 0;

}

Rta 3:

#include <iostream> using namespace std; class Operaciones { private:

int numero1;

int numero2; public:

Operaciones(int n1, int n2) : numero1(n1), numero2(n2) {} int sumar() const {

return numero1 + numero2;

}

int restar() const {

return numero1 - numero2;

}

int multiplicar() const {

return numero1 \* numero2;

}

double dividir() const { if (numero2 != 0) {

return static\_cast<double>(numero1) / numero2;

} else {

cout << "Error: Division por cero." << endl; return 0.0;

}

}

void imprimirResultados() const {

cout << "Suma: " << sumar() << endl; cout << "Resta: " << restar() << endl;

cout << "Multiplicacion: " << multiplicar() << endl; double resultadoDivision = dividir();

if (numero2 != 0) {

cout << "Division: " << resultadoDivision << endl;

}

}

};

int main() {

Operaciones op(10, 5); op.imprimirResultados(); return 0;

}

Rta 4:

#include <iostream> using namespace std;

class Sumatoria { private:

int suma; public:

Sumatoria() : suma(0) { int valor;

cout << "Ingrese valores enteros (0 para finalizar):" << endl; do {

cin >> valor;

suma += valor;

} while (valor != 0);

}

~Sumatoria() {

cout << "La suma de los valores ingresados es: " << suma << endl;

}

};

int main() {

Sumatoria sumatoria; return 0;

}

Rta 5:

#include <iostream> #include <string> using namespace std; class Estudiante {

private:

string nombre; int edad;

float calificacion; public:

void setNombre(const string& nom) { nombre = nom;

}

void setEdad(int ed) { edad = ed;

}

void setCalificacion(float cal) { calificacion = cal;

}

string getNombre() const { return nombre;

}

int getEdad() const { return edad;

}

float getCalificacion() const { return calificacion;

}

};

int main() {

Estudiante estudiante;

estudiante.setNombre("Carlos Mendoza"); estudiante.setEdad(21);

estudiante.setCalificacion(8.5f);

cout << "Nombre: " << estudiante.getNombre() << endl; cout << "Edad: " << estudiante.getEdad() << endl;

cout << "Calificacion: " << estudiante.getCalificacion() << endl;

return 0;

}

ALUMNOS:PARRONDO Nehuen, OVIEDO Santiago