***TP N.º 9 - Revisión de Clases - Getters y Setters - Constructor y Destructor***

Alumno: Nahuel Vilca



1. **Resolver las siguientes actividades:**
2. ¿Qué es el encapsulamiento? ¿Cuáles son los niveles de acceso a miembros de la clase?
3. ¿Qué son los getters y setters? ¿Cuál es su propósito principal? ¿Cuándo es conveniente usarlos?
4. ¿Cuál es la diferencia entre un constructor y un destructor? ¿Cuándo se ejecutan cada uno?
5. Realizar un cuadro comparativo entre las características de constructores y destructores
6. 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.
7. **El encapsulamiento** es uno de los principios más importantes de la programación orientada a objetos. Consiste en agrupar **(encapsular)** en una clase tanto los datos **(atributos)** como los métodos que los manipulan, restringiendo el acceso directo a ciertos miembros de la clase y permitiendo que solo se acceda a través de mecanismos controlados definidos por el programador. Esto se utiliza principalmente para garantizar mayor seguridad, modularidad y evitar accesos indebidos a los datos o métodos internos de la clase.

**Los niveles de acceso a los miembros de una clase son:**

* **Public:** Los miembros son accesibles desde cualquier parte del programa mediante un objeto de la clase.
* **Private:** Los miembros son accesibles únicamente por dentro de la misma. Este es el nivel de acceso por defecto.
* **Protected:** Los miembros son accesibles dentro de la clase padre que los define y en las clases hijos (osea heredaron lo que tiene la clase padre), pero no desde fuera de ellas.

1. **Los getters y setters** son métodos que permiten acceder y modificar variables privada de una clase de forma controlada. Su propósito principal es mantener el principio de encapsulamiento, evitando el acceso directo a los atributos y ofreciendo un mecanismo seguro para leer o modificar su valor.

* **Getter (Get):** es un método que devuelve el valor actual de una variable privada. Generalmente devuelve un dato del mismo tipo que la variable no suele tener parámetros de entrada.
* **Setter (Set):** Es un método que modifica el valor de una variable privada. Normalmente no devuelve ningún valor y debe tener al menos un parámetro de entrada el cual usara para modificar a la variable privada.

**Conveniencia de uso:** Se utilizan cuando queremos **proteger los atributos** de la clase de accesos o modificaciones indebidas, pero necesitamos ofrecer una manera segura de leerlos (getter) o actualizarlos (setter).

1. **La diferencia principal** entre **un** **constructor** y **un destructor** radica en cuando se ejecuta y en su propósito:

* **Constructor:**
* Es un método especial que tiene el mismo nombre de la clase.
* Se ejecuta automáticamente en el momento en que se crea un objeto
* Su objetivo es inicializar los atributos del objeto
* No devuelve valores y puede recibir parámetros para establecer valores iniciales.
* Se ejecuta una única vez al crear el objeto.
* **Destructor:**
* Es un método especial que tiene el mismo nombre de la clase precedido por el carácter ~.
* Se ejecuta automáticamente cuando el objeto es destruido o al finalizar el programa.
* Su propósito principal es liberar recursos, cerrar archivos, mostrar mensajes de salida u otras tareas de limpieza.
* No devuelve valores, no recibe parámetros y se ejecuta una sola vez por objeto

**En resumen:** el constructor se ejecuta al crear un objeto y lo inicializa, mientras que el destructor se ejecuta al destruir el objeto y se usa para liberar recursos o realizar tareas finales.

1. **Cuadro comparativo entre Constructor y Destructor:**

|  |  |  |
| --- | --- | --- |
| **Característica** | **Constructor** | **Destructor** |
| **Nombre** | Igual al de la clase. | Igual al de la clase, pero con ~ al inicio. |
| **Momento de ejecución** | Se ejecuta automáticamente al crear un objeto. | Se ejecuta automáticamente al destruir un objeto o al finalizar el programa. |
| **Número de ejecuciones** | Una sola vez al inicio de la vida del objeto. | Una sola vez al final de la vida del objeto. |
| **Objetivo principal** | Inicializar los atributos del objeto. | Liberar recursos, mostrar mensajes de salida o realizar tareas de limpieza. |
| **Retorno de datos** | No puede retornar datos. | No puede retornar datos. |
| **Parámetros** | Puede recibir parámetros para inicializar los atributos. | No recibe parámetros. |
| **Obligatoriedad** | No es obligatorio definirlo (C++ provee uno por defecto si no se crea). | No es obligatorio definirlo (C++ también crea uno por defecto si no se define). |
| **Ejemplo de uso** | Pedir datos al usuario y guardarlos en atributos al crear un objeto. | Mostrar un mensaje de despedida o cerrar un archivo antes de terminar. |

1. **Cuadro comparativo: Definir función dentro de la clase vs. fuera de la clase:**

|  |  |  |
| --- | --- | --- |
| **Característica** | **Definición dentro de la clase** | **Definición fuera de la clase (con ::)** |
| **Sintaxis** | Se declara y define en la misma parte dentro de la clase. | Se declara dentro de la clase y se define luego fuera usando NombreClase::NombreFuncion. |
| **Comportamiento** | El compilador la trata automáticamente como inline (puede expandir el código en el lugar de la llamada). | Se considera una función normal (no inline), salvo que se indique explícitamente. |
| **Ventajas** | - Código más corto y fácil de leer en clases pequeñas. - Conveniente para métodos simples (ejemplo: getters y setters). | - Mantiene la definición separada de la declaración, mejorando la organización. - Ideal para proyectos grandes con múltiples archivos. |
| **Desventajas** | - Puede hacer que la definición de la clase sea muy extensa y menos legible si los métodos son largos. - Abusar de inline puede aumentar el tamaño del ejecutable. | - Requiere más escritura (usar ::). - Al estar separado, puede costar un poco más ubicar la implementación. |
| **Cuando usar** | - En métodos muy cortos y simples (ej: getters, setters, operaciones triviales). - En clases pequeñas o de uso rápido. | - En métodos largos o complejos. - En programas grandes con múltiples archivos para mantener el código más limpio y modular. |