

Actividad 2 - Conexión y Tablas.

Lenguaje de Programación II.

Ingeniería en Desarrollo de Software

Tutor: Miguel Ángel Rodríguez Vega.

Alumnos: Carlos Ariel Nicolini.

Fecha: 27/112023

Índice

Introducción	3
Descripción	5
Justificación	6
Desarrollo	7
• Codigo Polimorfismo	7
Conclusión	11
Referencias.....	12

Introducción

En el ámbito de la programación orientada a objetos se entiende por polimorfismo a la capacidad de llamar a funciones distintas con un mismo nombre. Estas funciones pueden actuar sobre objetos distintos dentro de una jerarquía de clases, sin tener que especificar el tipo exacto de los objetos, es decir, es la capacidad de manejar distintas clases heredadas de una clase base de la misma forma.

Enlace dinámico con funciones virtuales

Una función virtual es una función miembro que se declara dentro de una clase base (padre) y es redefinida por una clase derivada (hija). Cuando se hace referencia a un objeto de clase derivada mediante un puntero o una referencia a la clase base, se puede llamar a una función virtual para ese objeto y ejecutar la versión de la función de la clase derivada:

- Las funciones virtuales garantizan que se llame a la función correcta para el objeto, independientemente del tipo de referencia utilizado para la llamada a la función.
- Se utilizan principalmente para lograr el polimorfismo en tiempo de ejecución.
- Las funciones se declaran con una palabra clave **virtual** en la clase base (padre).
- La resolución de la llamada a la función se realiza en tiempo de ejecución.

Reglas para funciones virtuales

- Las funciones virtuales no pueden ser estáticas. Una función virtual puede ser una función amiga de otra clase.
- Se debe acceder a las funciones virtuales utilizando el puntero o la referencia del tipo de clase base (padre) para lograr el polimorfismo en tiempo de ejecución.
- El prototipo de funciones virtuales debe ser el mismo en la base (padre), así como en la clase derivada (hija).

- Siempre se definen en la clase base (padre) y se anulan en una clase derivada. No es obligatorio que la clase derivada (hija) anule (o redefina la función virtual), en ese caso se utiliza la versión de la base.
- Una clase puede tener un destructor virtual, pero no puede tener un constructor virtual. Para construir un objeto, un constructor necesita el tipo exacto de objeto que la crea y no se puede tener un puntero a un constructor (el objeto aún no existe).

Descripción

En esta actividad realizaremos un código en Visual C++ utilizando polimorfismo. Usaremos el caso de una ferretería, y nos centraremos en tornillos, su tipo ya sea para madera o metal, su largo expresado en centímetros, la cantidad de unidades que se encuentran en el inventario de la ferretería, además de su ubicación por el número de pasillo y la estantería donde se encuentran.

Este ejercicio me resulto muy divertido e ilustrativo, en el uso del Visual Studio, el cual es una herramienta muy buena, te muestra los errores y gracias a las explicaciones del profesor con este ejercicio se me hizo muy entretenido.

Espero que el ejercicio cumpla con las expectativas, siempre un placer poder estar en las clases.

Justificación

Esta actividad se realiza con el fin de aprender los métodos del polimorfismo, anteriormente aprendimos sobre herencias.

Es necesario no solamente dominar las herencias y el polimorfismo, sino también el uso de Visual Studio ya que es la herramienta con la que vamos a estar programando scripts y ejecutándolos.

Dicho script se realizó, se probó, se adjuntaron imágenes del código, los resultados de su ejecución y se subirá al enlace de GitHub que se proporciona a continuación:

<https://github.com/CarlosNico/Lenguajes-de-Programaci-n-II>

Desarrollo

Código Polimorfismo

En esta parte del ejercicio presentamos el modelo Logico-Relacional el cual nos muestra un ordenamiento de los datos en sus respectivas tablas y sus relaciones. Los campos de las tablas están divididos en filas y columnas. Las relaciones entre tablas mejoran la gestión de los datos.

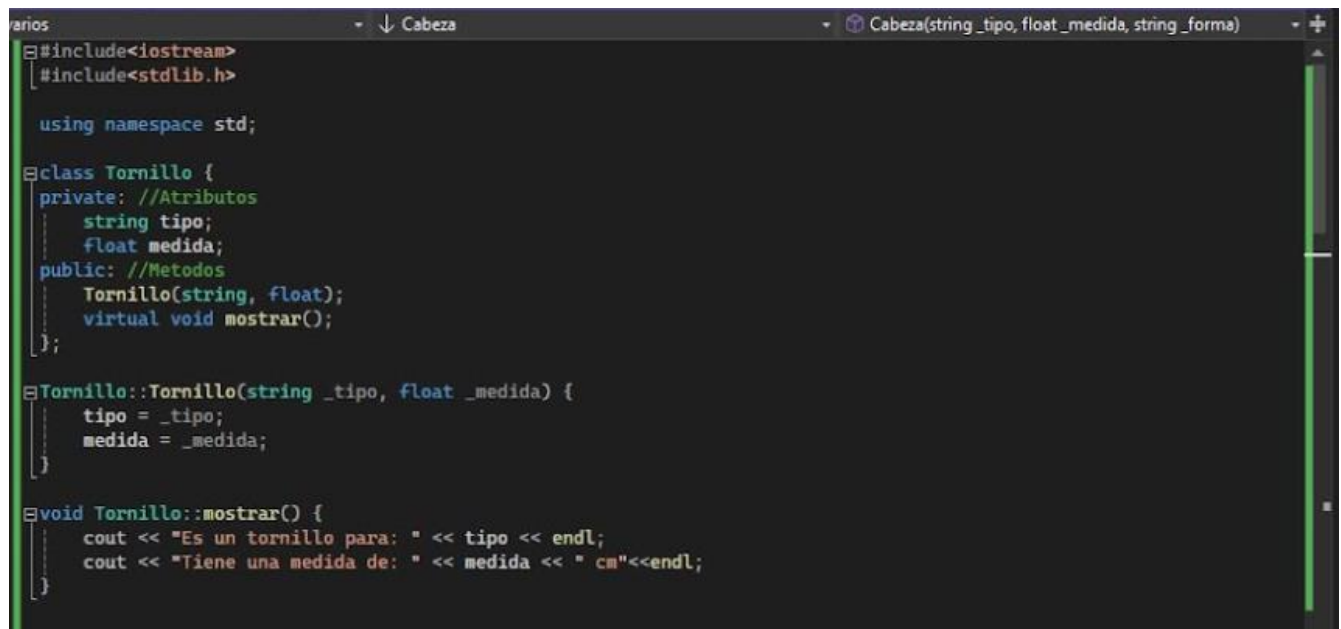
En esta parte empezamos nuestro código incluyendo las librerías `iostream` y `stdlib.h`.

Declaramos la clase padre llamada `Tornillo`, con los siguientes atributos:

- `tipo` (para tornillo de madera o de metal) de tipo `string`.
- `medida` (largo del tornillo en centímetros) de tipo `float`.

Declaramos su método de tipo `virtual void mostrar ()` (para polimorfismo).

Creamos el constructor y el método de la clase `tornillo`.



```
#include<iostream>
#include<stdlib.h>

using namespace std;

class Tornillo {
private: //Atributos
    string tipo;
    float medida;
public: //Metodos
    Tornillo(string, float);
    virtual void mostrar();
};

Tornillo::Tornillo(string _tipo, float _medida) {
    tipo = _tipo;
    medida = _medida;
}

void Tornillo::mostrar() {
    cout << "Es un tornillo para: " << tipo << endl;
    cout << "Tiene una medida de: " << medida << " cm"<<endl;
}
```

Declaramos la clase llamada Cabeza que es hija de la clase Tornillo, con los siguientes atributos:

- forma (referente a el tipo de cabeza del tornillo) de tipo string.

Declaramos su metodo de tipo void mostrar().

Creamos el constructor y el metodo de la clase hija cabeza.

```
class Cabeza : public Tornillo {
private:
    string forma;
public:
    Cabeza(string, float, string);
    void mostrar();
};

Cabeza::Cabeza(string _tipo, float _medida, string _forma) : Tornillo(_tipo, _medida) {
    forma = _forma;
}

void Cabeza::mostrar() {
    Tornillo::mostrar();
    cout << "Tipo de cabeza es: " << forma << endl;
}
```

Declaramos la clase llamada Cantidad que es hija de la clase Tornillo, con los siguientes atributos:

- unidades (referente a el numero de unidades disponibles en el inventario) de tipo int.

Declaramos su metodo de tipo void mostrar().

Creamos el constructor y el metodo de la clase hija cantidad.

```
class Cantidad : public Tornillo {
private:
    int unidades;
public:
    Cantidad(string, float, int);
    void mostrar();
};

Cantidad::Cantidad(string _tipo, float _medida, int _unidades) : Tornillo(_tipo, _medida) {
    unidades = _unidades;
}

void Cantidad::mostrar() {
    Tornillo::mostrar();
    cout << "El numero de unidades disponibles en almancen es: " << unidades << endl;
}
```


Declaramos la clase llamada Ubicacion que es hija de la clase Tornillo, con los siguientes atributos:

- pasillo(referente a el numero de paso donde se encuentra ubicado) de tipo int.
- Estante(referente a el numero de estante donde se encuentra alojado) de tipo string.

Declaramos su metodo de tipo void mostrar().

Creamos el constructor y el metodo de la clase hija Ubicacion.

```
class Ubicacion : public Tornillo {
private:
    int pasillo;
    string estante;
public:
    Ubicacion(string, float, int, string);
    void mostrar();
};

Ubicacion::Ubicacion(string _tipo, float _medida, int _pasillo, string _estante) : Tornillo(_tipo, _medida) {
    pasillo = _pasillo;
    estante = _estante;
}

void Ubicacion::mostrar() {
    Tornillo::mostrar();
    cout << "Se encuentra en el pasillo: " << pasillo << endl;
    cout << "En el estante: " << estante << endl;
}
```

Una vez que tenemos nuestras clases definidas con sus constructores y sus metodos, declaramos la funcion principal, en la cual declaramos 4 vectores (en polimorfismo es mas facil utilizar vectores). En este caso definimos que de la clase padre van a ser 4 objetos.

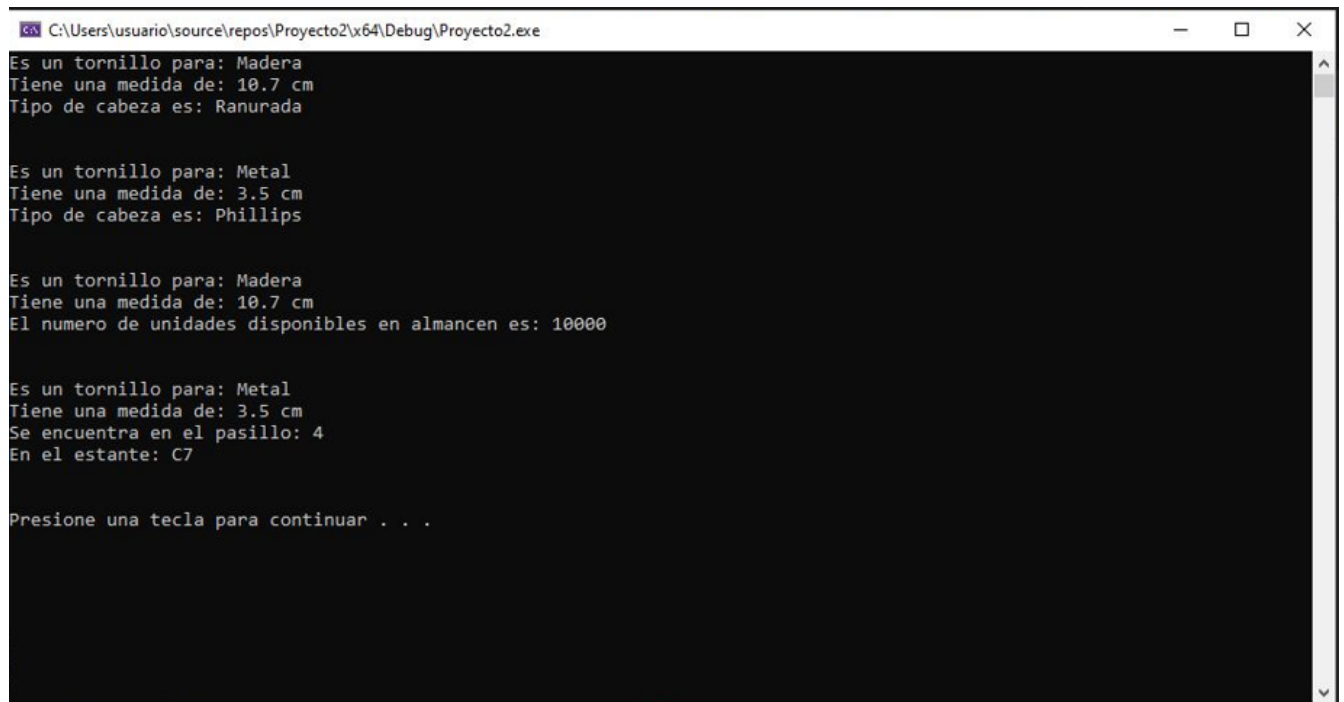
```
int main() {
    Tornillo* vector[4];

    vector[0] = new Cabeza("Madera", 10.7, "Ranurada");
    vector[1] = new Cabeza("Metal", 3.5, "Phillips");
    vector[2] = new Cantidad("Madera", 10.7, 10000);
    vector[3] = new Ubicacion("Metal", 3.5, 4, "C7");

    vector[0]->mostrar();
    cout << "\n" << endl;
    vector[1]->mostrar();
    cout << "\n" << endl;
    vector[2]->mostrar();
    cout << "\n" << endl;
    vector[3]->mostrar();
    cout << "\n" << endl;

    system("pause");
    return 0;
}
```

Y ponemos una captura del resultado de la ejecución de nuestro script.



```
C:\Users\usuario\source\repos\Proyecto2\x64\Debug\Proyecto2.exe
Es un tornillo para: Madera
Tiene una medida de: 10.7 cm
Tipo de cabeza es: Ranurada

Es un tornillo para: Metal
Tiene una medida de: 3.5 cm
Tipo de cabeza es: Phillips

Es un tornillo para: Madera
Tiene una medida de: 10.7 cm
El numero de unidades disponibles en almacén es: 10000

Es un tornillo para: Metal
Tiene una medida de: 3.5 cm
Se encuentra en el pasillo: 4
En el estante: C7

Presione una tecla para continuar . . .
```

Conclusión

El uso de polimorfismo es de gran ayuda, ya que nos permite estandarizar la llamada a métodos, también nos permite extender el funcionamiento ya establecido a través de la sobrecarga de métodos, de modo que si el comportamiento del método de una clase padre no es suficiente podemos ampliarlo en clases hijas.

Este ejercicio me gustó mucho, el uso de programación en C++ que hemos estamos teniendo en las clases y el uso del programa Visual Studio sobre el cual estoy gracias a estos ejercicios aprendiendo a usar. Espero que el trabajo presentado sea del agrado y cumpla con las expectativas del ejercicio.

Referencias

Tutor de C++. (n.d.). Ugr.Es. Retrieved November 30, 2023, from

https://ccia.ugr.es/~jfv/ed1/c++/cdrom3/TIC-CD/web/tema20/teoria_1.htm

Guzman, H. C. (n.d.). *Polimorfismo de clases*. Hektorprofe.net. Retrieved November 30,

2023, from <https://docs.hektorprofe.net/cpp/11-clases/06-polimorfismo-clases/>

Polimorfismo. (n.d.). Programacionenc.net. Retrieved November 30, 2023, from

https://www.programacionenc.net/index.php?option=com_content&view=article&id=70:polimorfismo&catid=37:programacion-cc&Itemid=55

López, E. (2022, July 7). *Ventajas y desventajas del polimorfismo - C++ avanzado 2*.

136. *Programación en C++ || POO || Polimorfismo en C++*. (2017, January 23). Youtube.

<https://www.youtube.com/watch?v=H8GkLKAKVmg>