Laboratorio

Polimorfismo

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas usando polimorfismo del lenguaje de programación C++.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando polimorfismo del lenguaje de programación C++.

2. Equipos y Materiales

- Un computador.
- IDE para C++.
- Compilador para C++.

3. Marco Teórico

3.1. Introducción

Polimorfismo, por definición, es la capacidad de adoptar formas distintas. En el ámbito de la Programación Orientada a Objetos se entiende por polimorfismo 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.

3.2. Polimorfismo

La palabra polimorfismo significa tener muchas formas. En palabras simples, podemos definir el polimorfismo como la capacidad de un mensaje para mostrarse en más de una forma. Un ejemplo de polimorfismo de la vida real, una persona al mismo tiempo puede tener diferentes características. Como un hombre al mismo tiempo es un padre, un esposo, un empleado. Entonces, la misma persona posee un comportamiento diferente en diferentes situaciones. A esto se le llama polimorfismo. El polimorfismo se considera una de las características importantes de la programación orientada a objetos.

3.2.1. Polimorfismo de sobrecarga

El polimorfismo de sobrecarga ocurre cuando las funciones del mismo nombre existen, con función similar, en clases que son completamente independientes unas de otras (estas no tienen que ser clases secundarias de la clase objeto). Por ejemplo, la clase complex, puede tener otra función. Esto significa que no necesitamos preocuparnos sobre el tipo de objeto con el que estamos trabajando si todo lo que deseamos es verlo en la pantalla.

Veamos un ejemplo, el operador '+' está sobrecargado. El operador '+' es un operador de suma y puede sumar dos números (enteros o de coma flotante), pero aquí el operador está hecho para realizar la suma de dos números imaginarios o complejos.

```
// Clase Complex.h
#pragma once
class Complex
{
public:
	Complex(int,int);
	~Complex();
	Complex operator+(Complex const & obj);
	void print();
private:
	int real, imag;

   };


// Clase Complex.cpp
#include "stdafx.h"
#include "Complex.h"
#include "iostream"

using namespace std;

Complex::Complex(int r = 0, int i = 0)
{
	real = r;
```

```
        imag = i;
}
Complex::~Complex()
{
}
// Esto se llama automáticamente cuando se usa '+' con entre dos objetos
complejos
Complex Complex::operator+(Complex const &obj)
{
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
        return Complex();
}
void Complex::print()
{
        cout << real << " + i" << imag << endl;
    }
```

```
#include "stdafx.h"
#include "Complex.h"
#include <iostream>
using namespace std;

int main()
{
        Complex c1(10, 5), c2(2, 4);
        Complex c3 = c1 + c2; // Un ejemplo de llamado al "operator+"
        c3.print();
        system("pause");
}
```

3.2.2. Polimorfismo paramétrico

El polimorfismo paramétrico es la capacidad para definir varias funciones utilizando el mismo nombre, pero usando parámetros diferentes (nombre y/o tipo). El polimorfismo paramétrico selecciona automáticamente el método correcto a aplicar en función del tipo de datos pasados en el parámetro.

Por lo tanto, podemos por ejemplo, definir varios métodos homónimos de funcion(). El método void funcion(int) devolvería el valor entero de x. Por su parte, void funcion(double) devolvería el valor double de x. En cuanto a void funcion (int, int) daría por resultado la impresión de los valores de “x” y “y”.

Una signature es el nombre y tipo (estático) que se da a los argumentos de una función. Por esto, una firma de método determina qué elemento se va a llamar.

Veamos un ejemplo:

```
// Clase ClaseBase.h
#pragma once
#include<iostream>

using namespace std;

class ClaseBase
{
public:
	ClaseBase();
	~ClaseBase();
	void funcion(int x);
	void funcion(double x);
	void funcion(int x, int y);
};


// Clase ClaseBase.cpp
#include "stdafx.h"
#include "ClaseBase.h"
#include <iostream>

using namespace std;

ClaseBase::ClaseBase()
{
}
ClaseBase::~ClaseBase()
{
}
// funcion con un parametros entero
void ClaseBase::funcion(int x)
{
	cout << "Funcion 1: El valor de x es: " << x << endl;
}
// funcion con el mismo nombre con un parametros double
void ClaseBase::funcion(double x)
{
	cout << "Funcion 2: El valor de x es: " << x << endl;
}
// funcion con el mismo nombre con dos parametros enteros
void ClaseBase::funcion(int x, int y)
{
	cout << "Funcion 3: Los valores de x y y son: " << x << ", " << y <<
endl;
}
```

```
#include "stdafx.h"
#include "ClaseBase.h"

int main()
{
	ClaseBase obj1;

	// La función que se llame dependerá de los parámetros pasados.
	// La 1ra funcion es llamada
	obj1.funcion(7);

	// La 2da funcion es llamada
	obj1.funcion(9.132);

	// La 3ra funcion es llamada
	obj1.funcion(85, 64);
	system("pause");
	return 0;
}
```

3.2.3. Polimorfismo en tiempo de ejecución

Este tipo de polimorfismo se logra mediante la anulación de funciones. La anulación de funciones, por otro lado, ocurre cuando una clase derivada tiene una definición para una de las funciones miembro de la clase base. Se dice que esa función base está anulada.

```
// Clase ClaseBase.h
#pragma once
#include<iostream>

using namespace std;

class ClaseBase
{
public:
	ClaseBase();
	~ClaseBase();
	virtual void print();
	void show();
};
```

```
// Clase ClaseBase.cpp
#include "stdafx.h"
#include "ClaseBase.h"
#include <iostream>

using namespace std;
```

```
ClaseBase::ClaseBase()
{
}
ClaseBase::~ClaseBase()
{
}
void ClaseBase::print()
{
	cout << "Imprime ClaseBase" << endl;
}
void ClaseBase::show()
{
	cout << "Muestra ClaseBase" << endl;
}


// Clase ClaseDerivada.h
#pragma once
#include "ClaseBase.h"
class ClaseDerivada : public ClaseBase
{
public:
	ClaseDerivada();
	~ClaseDerivada();
	void print();
	void show();
};


// Clase ClaseDerivada.cpp
#include "stdafx.h"
#include "ClaseDerivada.h"

ClaseDerivada::ClaseDerivada()
{
}
ClaseDerivada::~ClaseDerivada()
{
}
void ClaseDerivada::print() //print() ya es una función virtual en la clase
derivada, también podríamos declararlo como virtual void print()
explícitamente
{
	cout << "Imprime ClaseDerivada" << endl;
}
void ClaseDerivada::show()
{
	cout << "Muestra ClaseDerivada" << endl;
    }


#include "stdafx.h"
#include "ClaseDerivada.h"
#include <iostream>
```

```
using namespace std;

int main()
{
	ClaseBase *bptr;
	ClaseDerivada d;
	bptr = &d;
	// función virtual, enlazada en tiempo de ejecución (polimorfismo en
tiempo de ejecución)
	bptr->print();
	// Función no virtual, enlazada en tiempo de compilación
	bptr->show();
	system("pause");
	return 0;
}
```

4. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Defina una clase Forma que tenga los siguientes miembros de datos:
   - Color
   - Coordenada del centro de la forma (objeto Punto)
   - Nombre de la forma (char *)

   Y, al menos, las siguientes funciones miembro:
   - Imprimir
   - Obtener y cambiar el color
   - Mover la forma (o sea, su centro)

   Defina una clase derivada Rectángulo que tenga los siguientes miembros como datos:
   - Lado menor.
   - Lado mayor.

   Y, al menos, las siguientes funciones miembro:
   - Imprimir. Debe imprimir qué se trata de un rectángulo mostrando su nombre, color, centro y lado. Debería usarse la función Imprimir de la clase base para realizar parte de este trabajo.
   - Calcular el área (lado menor * lado mayor).
   - Calcular el perímetro. (2 * lado menor + 2 * lado mayor).
   - Cambiar el tamaño del rectángulo. Recibe como parámetro un factor de escala. Así, por ejemplo, si el factor vale 2, el rectángulo duplicará su tamaño y si es 0,5 se reducirá a la mitad.

   Realice un programa que pruebe el funcionamiento de estas clases. Debe crear objetos y comprobar el correcto funcionamiento de las funciones miembro.

2. Defina una clase Elipse derivada de forma. Recordatorio: una elipse queda definida por su radio mayor (R) y su radio menor (r), tal que el área de una elipse es igual a π*(R*r).
3. Defina una clase Cuadrado derivada de la clase Rectángulo.
4. Defina una clase Circulo derivada de la clase Elipse.
5. Realice un programa que defina varias formas diferentes, cree un vector de punteros de la clase Forma que apunten a los objetos creados. El programa debe realizar un bucle que recorra todas las formas, las ponga todas del mismo color y las mueva a una determinada posición.
6. Analice qué ocurre en el ejercicio anterior si se intenta imprimir la información de cada forma y qué sucede si se intenta obtener en ese bucle el área de todas las formas del vector.
7. Utilice la técnica de las funciones virtuales para arreglar los comportamientos anómalos detectados en el ejercicio anterior.
8. Desarrolle un programa que, dado un conjunto de formas, calcule cuál tiene el área máxima e imprima la información de dicha forma.

5. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB08_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB08_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB08_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB08_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.