TUTORÍA PROGRAMACIÓN I

FECHA: 24/08/22

TUTORA: KARLA VERÓNICA QUIRÓS DELGADO

TEMAS:

CONCEPTO DE CLASE

PRINCIPIOS BÁSICOS

PUNTEROS

REFERENCIAS

Clase:

Una clase en la programación se conoce como un “molde”, es decir es una estructura que nos ayudara con la creación de futuros objetos en un programa. Esta constituida por distintas partes como: private, public, etc. En el private de la clase se encuentran los atributos, los atributos son como las cualidades de si misma. En el public tendremos los métodos básicos de una clase y métodos de cálculo. Los métodos básicos son:

- Constructor por defecto
- Constructor con parámetros
- Destructor
- Métodos accesores (get’s)
- Métodos mutadores(set’s)
- Métodos de cálculo.

Estructura básica de una clase:

Class nombreDeLaClase{

private:

atributos

public:

constructor por defecto ()**;**

constructor parametrizado(parámetros);

destructor ();

métodos set’s();

métodos get’s();

otros métodos ();

}

Ejemplo de código de una clase básico:

```
1.  class Perro{
2.  private:
3.  string nombre;
4.  string raza;
5.  int edad;
6.  public:
7.  Perro(); // constructor por defecto
8.  Perro(string nombre, string raza, int edad);//constructor parametrizado
9.  virtual~Perro();
10. void setNombre(string nom){
11. nombre = nom;
12. }
13. void setRaza(string raz){
14. Raza = raz;
15. }
16. void setEdad(int ed){
17. edad = ed;
18. }
19. String getNombre(){
20. return nombre;
21. }
22. String getRaza(){
23. return raza;
24. }
25. Int getEdad(){
26. return edad;
27. }
28. //método para imprimir el objeto perro
29. string toString(){
30.
31. stringstream s;
32. s<<”Nombre :”<<nombre<<endl;
33. s<<”Raza   :”<<raza<<endl;
34. s<<”Edad   :”<<edad<<endl;
35.
36. return s.str();
37. }
38. };
39.
```

Principios básicos:

Principio de abierto-cerrado:

Este principio consiste en crear clases que estén “abiertas” para extender su funcionalidad, es decir agregarle métodos que la hagan más grande y eficiente para utilizar. Pero debe estar “cerrada” para que no se puedan cambiar sus atributos o métodos existentes. Ya que si se permite el cambio de sus atributos y métodos bases podría ocasionar fallos en el programa.

Principio de responsabilidad única:

Se basa en tener claro que una clase solo debe tener una única responsabilidad. Es decir, una clase debe crearse solo con atributos, métodos o funciones que vayan de acorde a su funcionalidad. Ejemplo: si tenemos la clase Persona, sus atributos básicos podrían ser: cedula, edad, nombre. Qué sentido tendría ponerle un atributo llamado “numero de placa de su carro”, aunque la persona puede tener un carro, no podríamos ponerle un atributo que debería ser de una clase “carro”. En palabras simples, este principio indica que una clase debe ser constituida de forma que solo tenga una responsabilidad.

Bajo acoplamiento:

Consiste en la capacidad de una clase para trabajar con módulos independientes, es decir los métodos de una clase deben trabajar lo máximo posible de manera que no dependan de otros métodos para funcionar, si todos los métodos dependen entre si para poder ser utilizados, entonces si hacemos un cambio en un método tendremos que cambiar todos los métodos dependientes a este, para evitar los fallos del programa.

Alta cohesión:

Hace posible que una clase solo realice la tarea para la que fue creada, haciendo que las otras funciones que le complementan sean realizadas por otras clases o funciones, es decir una clase no puede realizar funciones que involucren atributos de otras clases. Tener una alta cohesión permite entender una clase o método de forma mas clara, reutilizar clases o funciones.

Punteros:

Los punteros son variables o objetos que almacenan una dirección de memoria, estos punteros pueden inicializarse en “nullptr” o “NULL”. Lo cual es como decir que esta apuntando a la nada, cuando le asignamos valores reales del objeto al puntero le estaremos dando una dirección de memoria que apunta ese objeto y podrá acceder a la información de este.

Estructura básica de un puntero:

Tipo de dato * nombre del puntero;

Ejemplo:

dato tipo números entero:

Declaración = Int * ptr1;

Definición del puntero = int * ptr1 = 5;

dato tipo números reales: Double * ptr2;

Si desean ver más información les recomiendo:

https://docs.microsoft.com/es-es/cpp/cpp/pointers-cpp?view=msvc-170

https://www.programarya.com/Cursos/C++/Estructuras-de-Datos/Punteros

Referencias

Las referencias son similares a los punteros, se visualiza con el símbolo “&”, tienen la función de un alias es decir pueden acceder a la información de una dirección de memoria. Siempre deben inicializarse, si asignamos una referencia a un objeto original, le estaremos dando a la referencia el valor que tiene el objeto, pero si modificamos el alias (la referencia) estaremos cambiando el valor al objeto original.

Ejemplo de código:

```
1.  int main () {
2.
3.  int numero = 5; // identificador original
4.  int &copia = numero; // copia es un alias de numero
5.  copia = 10;
6.  cout<<”numero : ”<<numero<<endl; // imprime el valor del numero
7.  cout<<”copia : ”<<copia<<endl; //imprime el valor del alias(mismo del número por el momento)
8.  numero = 25; // cambia valor al número original
9.  cout<<”numero : ”<<numero<<endl; // imprime el valor del número actual
10. cout<<”copia : ”<<copia<<endl; //imprime el valor del alias(mismo del número original por el
    momento)
11. copia = 30;
12. cout<<”numero : ”<<numero<<endl; // imprime el valor del número actual
13. cout<<”copia : ”<<copia<<endl; //imprime el valor del alias
14. return 0;
15.
16. }
17.
18.
19.
```

Objetos automáticos:

Son llamados también como objetos estáticos, estos objetos existen en un rango pequeño es decir no tienen gran alcance en un programa, son creados dentro de un rango de llaves “{}”. Se eliminan automáticamente. Cuando el programa ya sale del rango de las llaves ya ese objeto no existe, solo vive en el rango que fue creado. Para acceder a sus métodos se utiliza: nombre_del_objeto, luego “.” Y este operador punto dará acceso a todos sus métodos.

```
1.    class Persona{
2.    private:
3.    string nombre;
4.    public:
5.              Persona() {
6.                        nombre = "";
7.    } // constructor por defecto
8.              Persona(string nom) {
9.                        nombre = nom;
10.   }//constructor parametrizado
11.             virtual~Persona() {
12.
13.   }
14.     void setNombre(string nom) {
15.     nombre = nom;
16.   }
17.   string toString() {
```

```
18.             stringstream s;
19.             s << "Nombre : " << nombre << endl;
20.             return s.str();
21.    }

int main(){

{

Persona p1(“juan”) ;//creación del objeto

cout<<p1.toString()<<endl;//imprimir información de p1

} //eliminación del objeto automático

return 0;

}
```

Objetos dinámicos:

Estos objetos al ser creados se les asigna un espacio de memoria mas estable en el programa. Son creados con la sentencia “new”(reserva la memoria) y se eliminan con “delete”(libera la memoria). Siempre que se creen los objetos dinámicos se deberá eliminar con el delete.

Características básicas:

- Al crearlos con new, el new devuelve un puntero (dirección de memoria) al objeto creado.
- Para acceder a los métodos de un objeto dinámico se debe poner: nombre_del_objeto, luego “->” y esto les dará acceso a sus métodos.
- No se puede acceder a métodos de objetos, luego de hacerles “delete”, porque ya no existen en el programa.

Ejemplo de codigo:

```
1.  class Persona
2.  {
3.  private:
4.    string nombre;
5.  public:
6.    Persona(){
7.        nombre = “”;
8.        }
9.    Persona(string nom){
10.   nombre = nom;
11.  }
12.   virtual~Persona(){}
13.   string toString()
14. {
15.   stringstream s;
16.   s << "Nombre ->" << nombre << endl;
17.   return s.str();
```

```
18. }
19. };
20. int main() {
21.   Persona* p1 = new Persona("Pedro");
22.   cout << p1->toString() << endl;
23.   return 0;
24. }
25.
```

Referencias:

Pascual, J. R. (2019, 27 diciembre). *Acoplamiento y Cohesión*. Disrupción Tecnológica. https://www.disrupciontecnologica.com/acoplamiento-y-cohesion/

Repaso de punteros:

Un puntero al que no se ha asignado un valor contiene datos aleatorios. Debe ser inicializado en nullptr o NULL. En caso de asignarle un objeto con valores reales se asigna según su estructura básica de creación.

Bloque de código sobre punteros y des referencia de estos:

```
1.  int* p = nullptr;// declaración del puntero, se le asignan valores aleatorios(random)
2.      int i = 5;
3.      p = &i; //asigna a p la dirección del objeto i
4.      int j = *p; // dereferencia a p, para recuperar el valor en su dirección
5.
```

This:

This no es más que un puntero que hace referencia a una clase en específico, es decir una clase es un molde podemos crear uno o mas objetos de esta. Es por eso que "this" nos permite referirnos a la clase en la que estemos trabajando, this significa (este objeto), por lo que, si ponemos la palabra "this" en c++, nos permitirá acceder a sus atributos y métodos en específico.

This almacena la dirección de memoria donde se encuentran los atributos y métodos de una clase en específico. No es obligatorio usarlo, pero si buena práctica. Cuando llamamos un método de la clase dentro de otro método indirectamente estamos haciendo uso de "this", para acceder al método.

Código de ejemplo:

```
#include<iostream>
using namespace std;
class Profesor{
    int edad;
    string ced;
    string nom;
public:
   Profesor(int,string,string);//parametrizado
};
Profesor:: Profesor (int ed,string n,string c)
{
    this->edad = ed;// "esta" edad = ed;
    this->nom = n;
    this->ced = c;
}
int main(){
    Persona per(10, "Leo", "1111");
    return 0;
}
```

Referencias:

Moisset, D. (2022). *Puntero this*. Punteros. https://www.tutorialesprogramacionya.com/cmasmasya/detalleconcepto.php?codigo=185&punto=54&inicio=45

C. (2022, August 8). *Puntero this*. Microsoft Docs. https://docs.microsoft.com/es-es/cpp/cpp/this-pointer?view=msvc-170