Unidad 7
Clases y Objetos

Una clase es un tipo de datos definido por el usuario.

- Provee un “molde” o ”diseño" para múltiples objetos del mismo tipo o categoría.

Un objeto es una instancia de una clase.

- Cada objeto tiene una localización única en memoria, y valores únicos para sus atributos

Una clase contiene atributos (almacenan el estado del objeto) y operaciones definen sus responsabilidades o capacidades).

En la declaración de una clase, para cada dato miembro (campo o método), debe especificarse mediante los modificadores de acceso el ´ambito desde el cual puede accederse a dicho miembro. Son:

- **private:** Sólo se permite su acceso desde las funciones miembro (métodos) de la clase.
- **Public:** Se permite su acceso desde cualquier punto que pueda usar la clase. Un dato público es accesible desde cualquier objeto de la clase.
- **Protected:** Se permite su uso en los métodos de la clase y en los de las clases derivadas mediante herencia

Parte 1

```
// Empleando la definición de una clase y creación de un
objeto.
#include <iostream>
using namespace std;
class Persona{
private:
        char nombre[40];
        int edad;
public:
        void inicializar();
        void imprimir();
        void esMayorEdad();
};
void Persona::inicializar()
{
        cout<<"ingrese el nombre: ";
        cin.getline(nombre,40);
        cout<<"Ingrese edad \t";
        cin >>edad;
}
void Persona::imprimir()
{
        cout<<"Nombre:";
        cout<<nombre;
        cout<<"\n";
        cout<<"Edad:";
        cout<<edad;
        cout<<"\n";
}
void Persona::esMayorEdad()
{
```

Parte 2

```
        if (edad>=18)
        {
        cout <<"Es mayor
de edad.";
        }
        else{
        cout<<"No es
mayor de edad";
        }
        cin.get();
        cin.get();
}
 main()
{
        Persona persona1;
        persona1.inicializar();
        persona1.imprimir();
        persona1.esMayorEdad();
}
```

Ejercicio 1

Construya una clase llamada Rectángulo que tenga los siguientes atributos: largo y ancho, y los siguientes métodos: perímetro() y área() (agregar que el dato sea incluido por el usuario)

```
#include<iostream>
#include<stdlib.h>
using namespace std;
class Rectangulo{
		private: //Atributos
				float largo,ancho;
		public: //Metodos
				Rectangulo(float,float); //Constructor
				void perimetro();
				void area();
};
Rectangulo::Rectangulo(float _largo,float _ancho){
		largo = _largo;
		ancho = _ancho;
}
void Rectangulo::perimetro(){
		float _perimetro;
		_perimetro = (2*largo) + (2*ancho);
		cout<<"El perimetro es: "<<_perimetro<<endl;
}
void Rectangulo::area(){
		float _area;
		_area = largo * ancho;
		cout<<"El area es: "<<_area<<endl;
}
int main(){
		Rectangulo r1(11,7);

		r1.perimetro();
		r1.area();

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

La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

Debemos declarar una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

Decíamos que una clase es un molde que nos permite definir objetos

Constructores

Un constructor es un método que pertenece a una clase y el cual (en C++) debe tener el mismo nombre de la clase a la que pertenece. A diferencia de los otros métodos de la clase, un constructor deberá ser del tipo **void**, es decir, el mismo no regresará valor alguno. Una clase puede tener más de un método constructor. Cada clase debe tener al menos un constructor, tanto así que si el programador creador de una clase no define un método constructor para la misma, el sistema, o sea el compilador, creará de manera automática un constructor nulo.

Método

Es un conjunto de instrucciones a las que se les da un determinado nombre de tal manera que sea posible ejecutarlas en cualquier momento sin tenerlas que reescribir sino usando sólo su nombre. A estas instrucciones se les denomina **cuerpo del método**, y a su ejecución a través de su nombre se le denomina **llamada al método**. EL método describe los mecanismos que se encargan de realizar sus tareas; y oculta las tareas complejas que realiza**.**

```
#include <iostream>
using namespace std;
class tablaMultiplicar{
    private:
                void calcular(int v);
     public :
                void cargarValor();
};
void tablaMultiplicar::calcular(int v)
{
        for (int f=v; f<=v*10;f=f+v)
            {
                cout<<f;
                cout<<"-";
            }
                cout<<"\n";
}
void tablaMultiplicar::cargarValor()
{
       int valor;
         do
           {
                cout<<"Ingrese un valor (-1 para finalizar): ";
                cin>>valor;
                if (valor!=-1)
                {
                        calcular(valor);
                }
           }
        while (valor!=-1);
}
main()
{
       tablaMultiplicar tabla1;
       tabla1.cargarValor();
}
```

Unidad 8
Herencia

La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todas los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios.

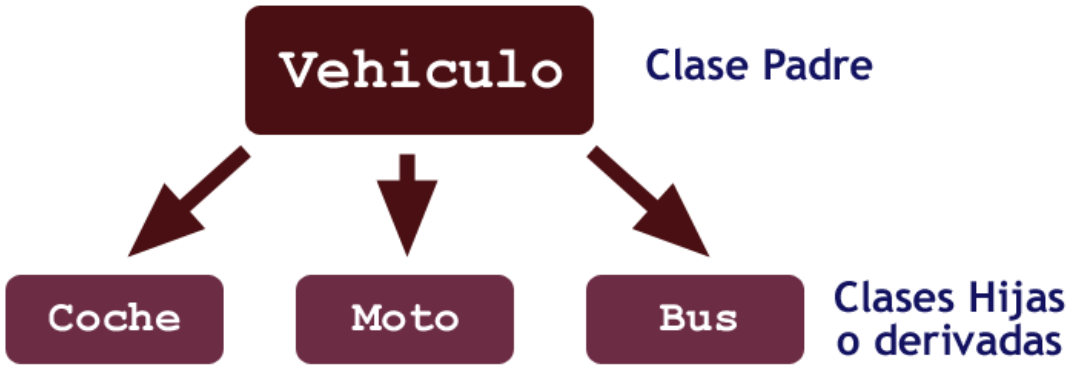
- **Clase padre**

Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la clase padre.

- **Subclase**

Clase que desciende de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase. Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

-Ejemplos teóricos de herencia



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

class Vehiculo {
public:
  void avanza() {}
};

class Coche : public Vehiculo {
public:
  void avanza(void)
    { cout << "Avanza coche." << endl; }
};

class Moto: public Vehiculo {
public:
  void avanza(void)
    { cout << "Avanza moto." << endl; }
};

int main()
{
  Moto t;
  Coche c;

  t.avanza();
  c.avanza();

  return 0;
}
```

```
#include <iostream>

using namespace std;

class Animal {
  public:
      void caminar() {
          cout << "Caminando en Animal" << endl;
      }
};

class Mamifero{
  public:
      void caminar() {
          cout << "Caminando en Mamifero" << endl;
      }
};

class Perro:public Animal, public Mamifero { };

int main(){
    Perro dog;
    dog.caminar();
    dog.caminar();
}
```

Unidad 9
Polimorfismo

En programación orientada a objetos se denomina polimorfismo a la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

Dicho de otra forma, el polimorfismo consiste en conseguir que un objeto de una clase se comporte como un objeto de cualquiera de sus subclases, dependiendo de la forma de llamar a los métodos de dicha clase o subclases. Una forma de conseguir objetos polimórficos es mediante el uso de punteros a la superclase.

En la práctica esto quiere decir que un puntero a un tipo puede contener varios tipos diferentes, no solo el creado. De esta forma podemos tener un puntero a un objeto de la clase Trabajador, pero este puntero puede estar apuntando a un objeto subclase de la anterior como podría ser Márketing, Ventas o Recepcionistas (todas ellas deberían ser subclase de Trabajador).

Clasificación

Se puede clasificar el polimorfismo en dos grandes clases:

• Polimorfismo dinámico (o polimorfismo paramétrico) es aquél en el que el código no incluye ningún tipo de especificación sobre el tipo de datos sobre el que se trabaja. Así, puede ser utilizado a todo tipo de datos compatible.

• Polimorfismo estático (o polimorfismo ad hoc) es aquél en el que los tipos a los que se aplica el polimorfismo deben ser explicitados y declarados uno por uno antes de poder ser utilizados. El polimorfismo dinámico unido a la herencia es lo que en ocasiones se conoce como programación genérica. También se clasifica en herencia por redefinición de métodos abstractos y por método sobrecargado. El segundo hace referencia al mismo método con diferentes parámetros.

Otra clasificación agrupa los polimorfismos en dos tipos: Ad-Hoc que incluye a su vez sobrecarga de operadores y coerción, Universal (inclusión o controlado por la herencia, paramétrico o genericidad).

Parte 1

```
#include <iostream>
#include <string>

using namespace std;

class Mascota
{
public:
  Mascota(string nombre, int patas);
  string hablar();

private:
  virtual string palabra();

  string nombre;
  int patas;
};

Mascota::Mascota(string nombre, int patas):nombre(nombre),patas(patas)
{
  cout << "Acaba de nacer tu mascota "<<nombre<<" con "<<patas<<"patas"<<endl;
}

string Mascota::palabra()
{
  return "BUUU";
}

string Mascota::hablar()
{
  return nombre+" dice: "+this->palabra();
}

class Perro : public  Mascota
{
public:
  Perro(string nombre);
```

Parte 2

```
  string palabra();
};

Perro::Perro(string nombre):Mascota(nombre, 4)
{
}

string Perro::palabra()
{
  return "GUAU";
}

class Gato : public  Mascota
{
public:
  Gato(string nombre);

  string palabra();
};

Gato::Gato(string nombre):Mascota(nombre, 4)
{
}

string Gato::palabra()
{
  return "MIAU";
}

int main()
{
  Mascota *m, *g;

  m=new Perro(“Toto");
  g=new Gato("Lancelot");

  cout << m->hablar() << endl;
  cout << g->hablar() << endl;
  return 0;
}
```

Dos importantes reglas

Las siguientes reglas son muy importantes a la hora de usar punteros, y deben ser perfectamente aprendidas.

1. Un nombre de variable, precedido por un signo de amperson define la **dirección de la variable**, y ocasionalmente “la apunta”. Puede leerse la línea 6 del programa como “pt1 es asignado al valor de la dirección de index”.

2. Un puntero con una estrella procediéndose se refiere al **valor de la variable** apuntada por el puntero. La línea 9 del programa puede leerse como: “al puntero pt1 le es asignado el valor 13”.

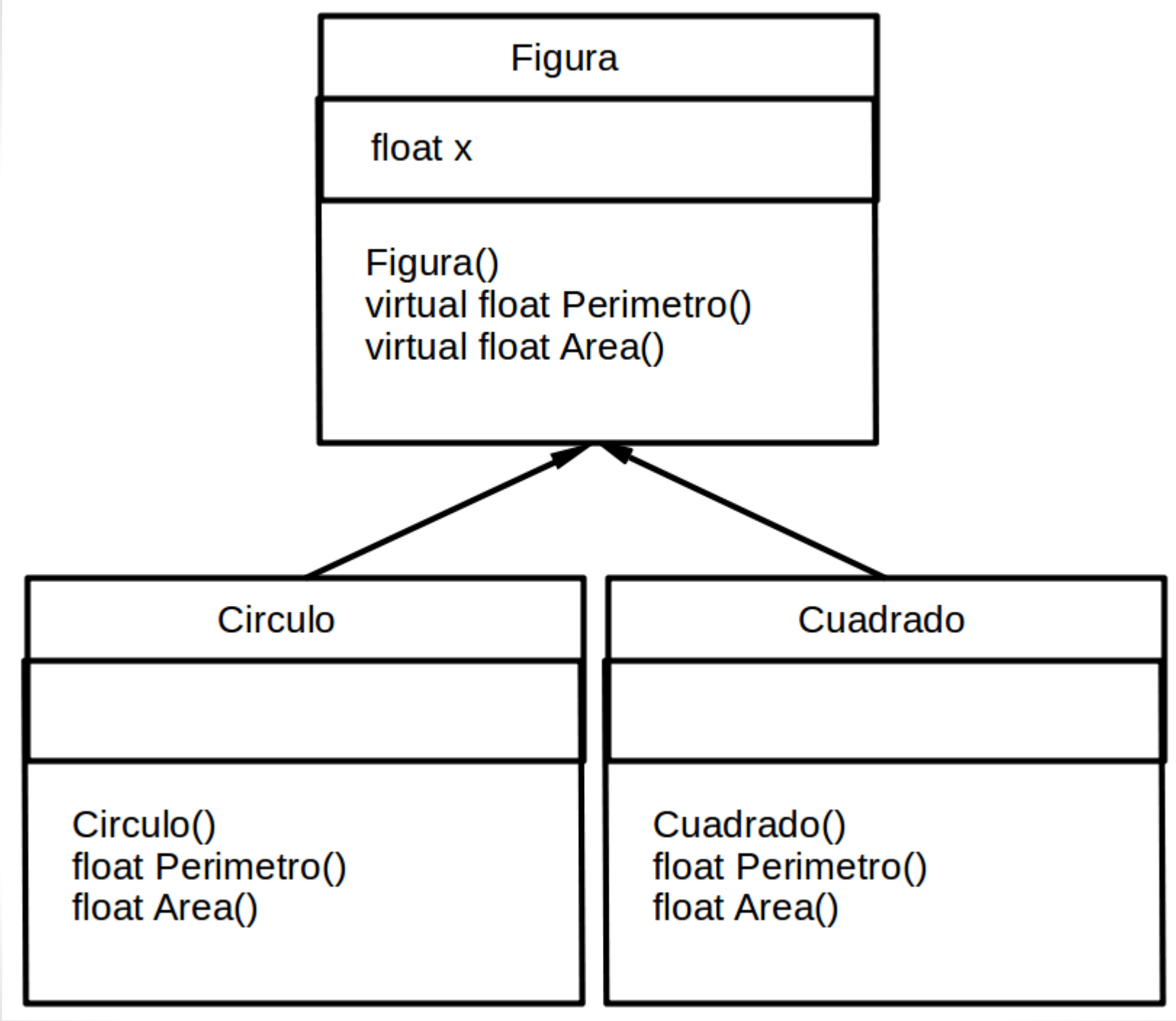


Figura
float x
Figura()
virtual float Perimetro()
virtual float Area()
Circulo
Circulo()
float Perimetro()
float Area()
Cuadrado
Cuadrado()
float Perimetro()
float Area()

```
#include <iostream>
#include <locale>
                    using namespace std;
const double PI = 3.141592;
class Figura{
 protected:
                      float x;
                     public:
                      Figura(float cx=0){
                        x = cx;
    }
 virtual float Perimetro() = 0;
 virtual float Area() = 0;
};

// Clases derivadas
class Circulo: public Figura{
  public:

  Circulo(float radio){
    x = radio;
  }
  float Perimetro(){
   return 2* PI * x;
  }
  float Area(){
   return PI * x * x;
  }
};

class Cuadrado: public Figura
{

 public:
 Cuadrado(float lado){
   x = lado;
 }
 float Perimetro(){
  return 4 * x;
 }
 float Area(){
  return x * x;
 }
};
```

```
int main(){
  setlocale(LC_ALL, "");

  float l, r;

  cout << "Entre el lado del cuadrado" << endl;
  cin >> l;

  cout << "Entre el radio del círculo" << endl;
  cin >> r;

  Cuadrado cuad1(l);
  Circulo  Circ1(r);

  cout << "El perimetro del circulo es:" << Circ1.Perimetro() << endl;
  cout << "El área del circulo es:" << Circ1.Area() << endl;

  cout << "El perimetro del cuadrado es:" << cuad1.Perimetro() << endl;
  cout << "El area del cuadrado es:" << cuad1.Area() << endl;
}
```

Unidad 10
Archivos de Entrada y Salida

Se utiliza el termino stream para indicar el flujo de datos a través.
Dentro de C++ existen librerías que permiten leer y escribir de un stream:

Entrada/salida con Streams en C++
Definición: un stream en C++ es un objeto mediante el cual un programa puede insertar o extraer datos utilizando los operadores de inserción << y de extracción >> o funciones. Los datos pueden ser tipos fundamentales (int, char, double, etc.), o cadenas de caracteres.
Librería de tipos I O y encabezados:

Encabezado	Tipo	Descripcion
iostream	istream, wistream	Lee de un stream
	ostream,wostream	Escribe a un stream
	iostream,wiostream	Lee y escribe a un stream
fstream	ifstream,wifstream	Lee de un archivo
	ofstream,wofstream	Escribe a un archivo
	fstream,wfstream	Lee y escribe a un archivo
sstream	istringstream,wistringstrea m	Lee de un string
	ostringstream,wostringstre am	Escribe a un string
	stringstreamstringstream	Escribe y lee un string

Streams predefinidos por defecto, la salida asignada a un programa es la pantalla, y en C++ está representada por el stream cout a la que se denomina “salida estándar”. Las definiciones necesarias para su uso están contenidas en el archivo de encabezamiento iostream. Caracteres, números y variables pueden insertarse en cout mediante el operador de inserción <<.

Streams especiales: archivos

C++ provee tres streams específicas para leer y escribir caracteres desde y hacia un archivo en disco:

1. ifstream para leer caracteres desde un archivo
2. ofstream para escribir caracteres en un archivo
3. fstream para leer y escribir caracteres en un archivo

Apertura de un archivo

Para utilizar un objeto ifstream, ofstream o fstream debe llevarse a cabo la vinculación del stream al archivo físico mediante una operación de apertura mediante la función open(), que toma dos argumentos: nombre del archivo y modo de apertura.

Modo	Significado
ios::in	El archivo se abre para operaciones de lectura (sólo extracción de caracteres)
ios::out	El archivo se abre para operaciones de escritura (sólo inserción de caracteres)
ios::binary	El archivo se abre en modo binario (su contenido no es texto sino bytes)
ios::app	Las operaciones de inserción tienen lugar al final del archivo, esto es, preservando su contenido inicial
ios::trunc	Si el archivo abierto existe, se descarta la totalidad de datos que contiene al momento de la apertura

Cierre de un archivo

Para garantizar que las operaciones realizadas efectivamente se reflejen en el archivo en disco, cuando se termina de trabajar sobre él debe ejecutar una operación de cierre mediante la función close():
arch close();

Ejercicio 1: Realice un programa que pida al usuario el nombre de un fichero de texto y, a continuación permita almacenar al usuario tantas frases como el usuario desee.
Ejercicio 2: Realice un programa que pida al usuario el nombre o ubicación de un fichero de texto y, a continuación de lectura a todo el fichero.
Ejercicio 3: Realice un programa que pida al usuario el nombre o ubicación de un fichero de texto y, a continuación añada texto en el hasta que el usuario lo decida.
Ejercicio 4: Hacer un programa en C++. Para guardar números telefónicos que muestre un menú con las siguientes opciones:
1. Crear (nombre, apellidos, teléfono)
2. Agregar más contactos (nombre, apellidos, teléfono)
3. Visualizar contactos existentes

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
        ofstream myfile;
        myfile.open ("salida.txt");
    myfile << "Escribiendo en archivos.\n";
    myfile.close();
    return 0;
    }
```

```
#include<iostream>
#include<stdlib.h>
#include<fstream> // libreria para la creación, escritura y lectura de archivos
using namespace std;

void lectura();

int main(){
		lectura();

		system("pause");
		return 0;
}

void lectura(){
		ifstream archivo; // de input o de lectura (ifstream
		string texto;

		//archivo.open("leerarchivo.txt", ios::in); //Abrimos archivo en modo lectura
		archivo.open("C:/Users/Mario Rocha Moreno/Documents/BSU/FIME/FIME
MODULOS/Modulo 2 C++/Semana 4/ juego.txt", ios::in); //Abrimos archivo en modo lectura de
una ruta diferente

		//Este IF nos ayuda cuando no tenemos el archivo nos indique que no se puede
abrir
				if(archivo.fail()){ //Si a ocurrido algun error
				cout<<"No se pudo abrir el archivo";
				exit(1); // para salir del programa en casi de ser asi
		}
		while (!archivo.eof()){ //mientras no sea el final del archivo
				getline(archivo,texto); //Cadena de caracteres para string
				cout<<texto<<endl;
		}
		archivo.close();
}
```