

Programación Orientada a Objetos

Programación digital II
Escuela de Sistemas
Facultad de Ingeniería
Profesor: Gilberto Diaz

En 1970 Dennis Ritchie y Brian Kernigan crearon el lenguaje C.

Este lenguaje de programación fue diseñado con el objetivo de escribir Sistemas Operativos.

Debido a su simplicidad y versatilidad, pronto se utilizó para escribir distintos tipos de programas.

El problema principal del lenguaje C es que es un lenguaje orientado a procedimientos, es decir, el programador debe empezar describiendo los datos y luego escribir los procedimientos para manipular los datos.

Los programadores se dieron cuenta que ellos podían crear programas más claros y fáciles de entender si tomaban un conjunto de datos, y los agrupaban junto con las operaciones que trabajaban sobre esos datos.

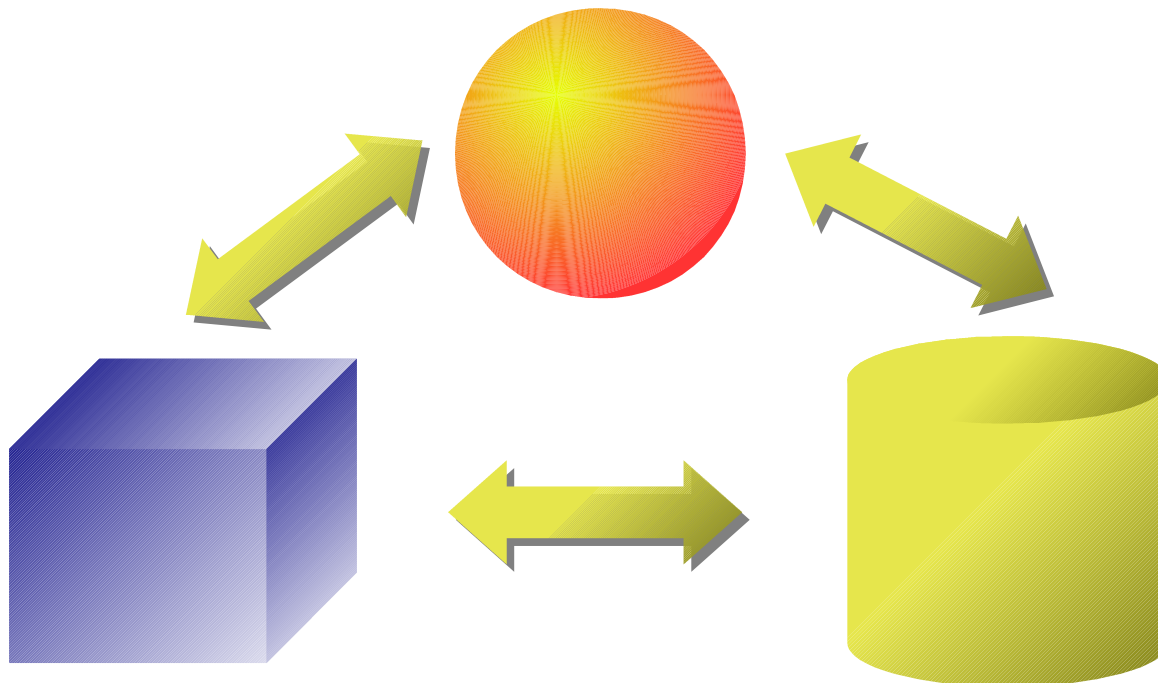
Varios lenguajes de programación orientados a objetos se han desarrollado. Sin embargo, C++ desde el principio de los 80 ha sido uno de los más populares.

C++ fue inventado por Bjarne Stroustrup

El Paradigma de Programación Orientado a Objetos es una técnica de programación que usa objetos y sus interacciones para diseñar aplicaciones y buenos programas de computadora.

La **Programación Orientado a Objetos** es una forma particular de programar que se asemeja más a la forma como expresamos las cosas en la vida real.

La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.



En programación convencional los programas se dividen en dos componentes:

- Procedimientos y
- Datos.

Las estructuras de datos utilizadas en programación son globales o se pasan como parámetros. **En esencia los datos se tratan separadamente de los procedimientos.**

En POO un programa se divide en **componentes** que contienen procedimientos y datos. Cada componente se considera un **objeto**

Un **objeto** es una unidad que contiene datos y las funciones que operan sobre esos datos.

En POO los objetos pueden ser cualquier entidad del mundo real:

- Objetos físicos
 - automóviles en una simulación de tráfico
 - Edificios modelados en computadora
 - árboles, etc
- Elementos de interfaces gráficas de usuarios
 - ventanas
 - iconos
 - menús
 - cursores

En POO los objetos pueden ser cualquier entidad del mundo real:

- Estructuras de datos
 - Arreglos
 - Números enteros
 - árboles binarios
- Tipos de datos definidos por el usuario
 - números complejos
 - hora del día

Un **Objeto** es una unidad que contiene datos y las funciones que operan sobre esos datos.

Los datos se denominan **atributos** y las funciones **métodos**.

Los datos y las funciones se **encapsulan** en una única entidad. Los datos están **ocultos** y sólo mediante las funciones miembro es posible acceder a ellos.

Los objetos son entidades que combinan estado, comportamiento e identidad:

- El **estado** está compuesto de datos, será uno o varios atributos a los que se habrán asignado unos valores concretos (datos).
- El **comportamiento** está definido por los procedimientos o métodos con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él.

Los objetos son entidades que combinan estado, comportamiento e identidad:

- La **identidad** es una propiedad de un objeto que lo diferencia del resto, dicho con otras palabras, es su identificador (concepto análogo al de identificador de una variable o una constante).

Una **Clase** es una colección de objetos similares y un objeto es una particularización de una definición de una clase.

Una **Clase** es un tipo definido por el usuario que determina las estructuras de datos y las operaciones asociadas con ese tipo.

Las clases son las definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.

Cada vez que se construye un objeto de una clase, se crea una **instancia** de esa clase.

En general, los términos **objetos** e **instancias** de una clase se pueden utilizar indistintamente.

```
class NombreDeLaClase{
```

```
    Tipo atributo1;
```

```
    Tipo atributo2;
```

```
    .
```

```
    .
```

```
    NombreDeLaClase() { }           // Constructor
```

```
    ~NombreDeLaClase() { }         // Destructor
```

```
    Tipo método1() { }
```

```
    Tipo método2() { }
```

```
};
```

Los objetos pueden ser activados mediante la recepción de mensajes.

Un **mensaje** es simplemente una petición para que un objeto se comporte de una determinada manera, ejecutando uno de sus métodos.

La técnica de enviar mensajes se conoce como pase de mensajes.

Estructuralmente un mensaje consta de tres partes:

- la identidad del objeto receptor
- El método cuya ejecución se ha solicitado
- cualquier otra información adicional que el receptor pueda necesitar para ejecutar el método requerido.

En C++, la notación utilizada es

nombreDelObjeto.Método(parámsAdicionales)

La sobrecarga de operadores es un mecanismo de abstracción que permite la modificación de la funcionalidad de los operadores incorporados en el lenguaje de programación.

C++ contempla la sobrecarga de operadores.

La sobrecarga de operadores sirve para construir sentencias sencillas que involucren los tipos de datos definidos por el usuario. Ejemplo:

Persona p1, p2;

```
if( p1 < p2 ){
```

-
-
-

Los operadores que se pueden sobrecargar son:

+	-	*	/	%	^	&
	~	!	=	<	>	+=
-=	*=	/=	%=	^=	&=	=
<<	>>	>>=	<<=	==	!=	<=
>=	&&		++	--	->*	,
->	[]	()	new	new[]	delete	delete[]

Ejemplo: Archivo VectorEnteros.h

```
class VectorEnteros {
    int tamano;
    int *valores;
public:
    VectorEnteros();
    ~VectorEnteros();
    .....
    bool operator==(VectorEnteros);
};
```

Ejemplo: Archivo VectorEnteros.cpp

```
#include "VectorEnteros.h"
bool VectorEnteros::operator==(VectorEnteros v){
    int i;
    if(tamano != v.tamano)
        return false;
    else{
        for(i=0; i<tamano; i++){
            if(this->valores[i] != v.valores[i])
                return false;
        }
        return true;
    }
}
```

Plantillas (Templates)

Los *templates* proporcionan una forma simple de representar un amplio rango de conceptos generales y una forma simple de combinarlos.

Las plantillas proporcionan un mecanismo para pasar los *tipos de datos* como *parámetros*

En C++ los tipos de datos se pueden pasar como *parámetros* en la definición de la clase

```
template <class Tipo>
class NombreClase {
    Tipo atributo1;.....

    NombreClase();
    Tipo método1();
}
```

```
template <class Tipo>
NombreClase<Tipo>::NombreClase(){
    .....
}
```

Los espacios de nombres son un mecanismo de abstracción que permite expresar agrupamientos lógicos.

Esto quiere decir, si una serie de declaraciones puede agruparse de acuerdo a un criterio lógico, entonces las podemos colocar en un mismo espacio de nombres.

Los espacios de nombres permiten organizar el código de acuerdo a las funciones que cumple cada elemento.

Los espacios de nombres permiten agrupar entidades como:

- Clases
- Objetos
- Funciones

Bajo un mismo nombre

El formato de un espacio de nombres es como sigue:

```
namespace Identificador{
```

Entidades

```
}
```

Donde **Identificador** es el nombre del espacio de nombres y *Entidades* son Clases, Objetos o funciones

Este mecanismo es especialmente útil en casos donde existen objetos globales o funciones que tengan el mismo identificador.

Ejemplo:

```
#include <iostream>
using namespace std;
namespace first
{
    int var = 5;
}
namespace second
{
    double var = 3.1416;
}
int main () {
    cout << first::var << endl;
    cout << second::var << endl;
}
```