POO
PROGRAMACIÓN ORIENTADA A OBJETOS

Programación estructurada, consiste en descomponer el problema objeto de resolución en subproblemas y más subproblemas hasta llegar a acciones muy simples y fáciles de codificar. Se trata de descomponer el problema en acciones, en **verbos Pedir, Hallar, Comprobar, Calcular, Mostrar ..**

La programación orientada a objetos es otra forma de descomponer problemas. En este nuevo método de descomposición en **objetos**, vamos a fijarnos no en lo que hay que hacer en el problema, sino en cuál es el escenario real del mismo, y vamos a intentar simular ese escenario en nuestro programa.

El elemento básico no es la función, sino un ente denominado precisamente **objeto**.

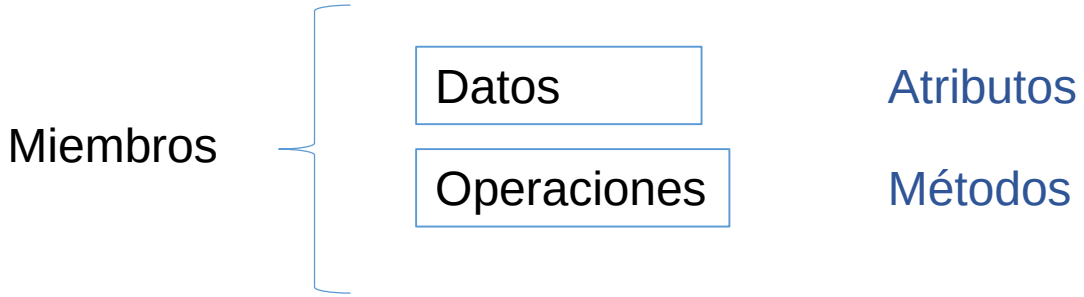
Un objeto es la representación en un programa de un concepto, y contiene toda la información necesaria para abstraerlo: datos que describen sus **propiedades o atributos** y **operaciones o métodos** que pueden realizarse sobre los mismos. Y no sólo eso, también podremos descubrir, a poco que nos fijemos, todo un conjunto de interrelaciones entre las entidades, guiadas por el intercambio de **mensajes.**

Un **Tipo Abstracto de Dato** (**TAD**) es un tipo de datos que incluye datos, funciones que manipulan dichos datos, y un método de encapsular los detalles. Se puede afirmar que es el centro de la POO.

La principal forma de crear TAD en **C++** es mediante un nuevo tipo de dato definido por el usuario, **la clase**.

CLASE = DATOS + FUNCIONES

Una **clase** es una plantilla que define las variables y los métodos que son comunes para todos los objetos de un cierto tipo.



Un **objeto** es una instancia de una clase

Diagrama de **clase coche**

Coche
Atributos: -Velocidad -Color -Potencia
Métodos: +Arrancar +Frenar +Acelerar

Instancia de clase coche. **Objeto**

Seat_Leon:Coche
Atributos: -Velocidad=190 -Color=“Rojo” -Potencia=120
Métodos: +Arrancar +Frenar +Acelerar

Inicialmente consideraremos que existen **tres momentos en la definición y utilización de una clase**:

DECLARACIÓN DE LA CLASE

Describir los datos y funciones que pertenecen a la clase.
No es necesario indicar el código de los métodos de la clase

```
class <nombre_de_la clase>
{
   private:
            <lista miembros privados (atributos y métodos)>
   public:
            <lista miembros públicos (atributos y métodos)>
};
```

DEFINICIÓN DE LA CLASE

Indicar el código de los métodos de la clase.

INSTANCIACIÓN DE LA CLASE

Proceso por el cual se crea un objeto de una clase.

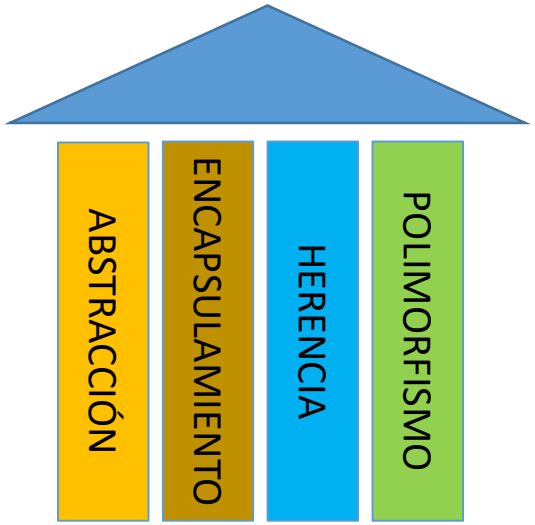
```
//clases01.cpp
#include <iostream>
#include <string>
#include <stdlib.h>
using namespace std ;
// ……..DECLARACIÓN
class coche
{
  private:
       int velocidad;
       string  color;
       float potencia;
       bool encendido;
  public:
       void apagar();
       void arrancar();
       void frenar();
       void acelerar();
       void parar();
       void mostrar();
}; //fin de la declaración
// ……..DEFINICIÓN
void coche::apagar(){
   encendido=false;
}
void coche::arrancar(){
   velocidad=0;
   encendido=true;
}
void coche::frenar(){
   velocidad -=10;
}
void coche::acelerar(){
   velocidad+=10;
}
void coche::parar(){
   velocidad=0;
}
void coche::mostrar(){
   cout<<velocidad;
   cout<<encendido;
}

int main()
{
    //……..INSTANCIACIÓN
    coche micoche;

     micoche.arrancar();
     micoche.acelerar();
     micoche.mostrar();
     micoche.frenar();
     system("pause");
     return 0;
}
```

Para instanciar un objeto de la clase coche basta escribir:

```
coche micoche;
```

Destacar que los métodos de una clase, acceden directamente a los datos y métodos de la propia clase, mientras que desde *fuera* como en el caso de la función *main*, es necesario utilizar el **operador "."** para poder ejecutarlos.

PILARES DE LA POO



ABSTRACCIÓN

Consiste en captar las características esenciales de un objeto, así como su comportamiento, al mismp tiempo que se ignoran los detalles no esenciales

En programación se refiere al énfasis en el **¿qué hace?** más que el **¿cómo lo hace?**

ENCAPSULAMIENTO

En la programación orientada a Objetos, los niveles de acceso son el medio que se utiliza para lograr el encapsulamiento que no es más que cada objeto se comporte de forma autónoma y lo que pase en su interior sea invisible para el resto de objetos. Cada objeto sólo responde a ciertos mensajes y proporciona determinadas salidas.

Los miembros privados de una clase no son accesibles para funciones y clases exteriores a dicha clase.

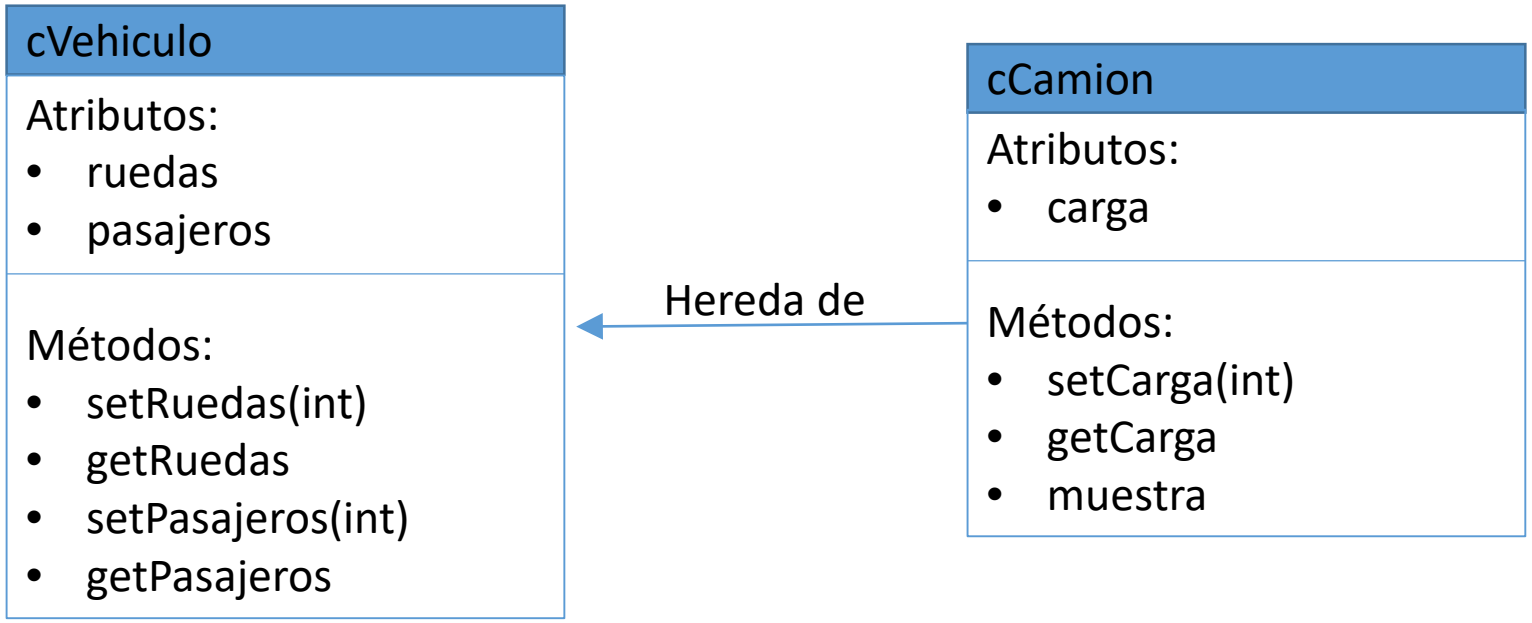
```
micoche.arrancar();
micoche.velocidad=120;  //Daría error en main() al ser atributo privado
```

HERENCIA

La herencia es un mecanismo potente de abstracción que permite compartir similitudes entre clases manteniendo al mismo tiempo sus diferencias.

Es una forma de reutilización de código, tomando clases previamente creadas y formando a partir de ellas nuevas clases, heredándoles sus atributos y métodos. Las nuevas clases pueden ser codificadas agregándoles nuevas características.

```
class   claseNueva: <acceso> claseBase
{
        //cuerpo de la subclase
}
```



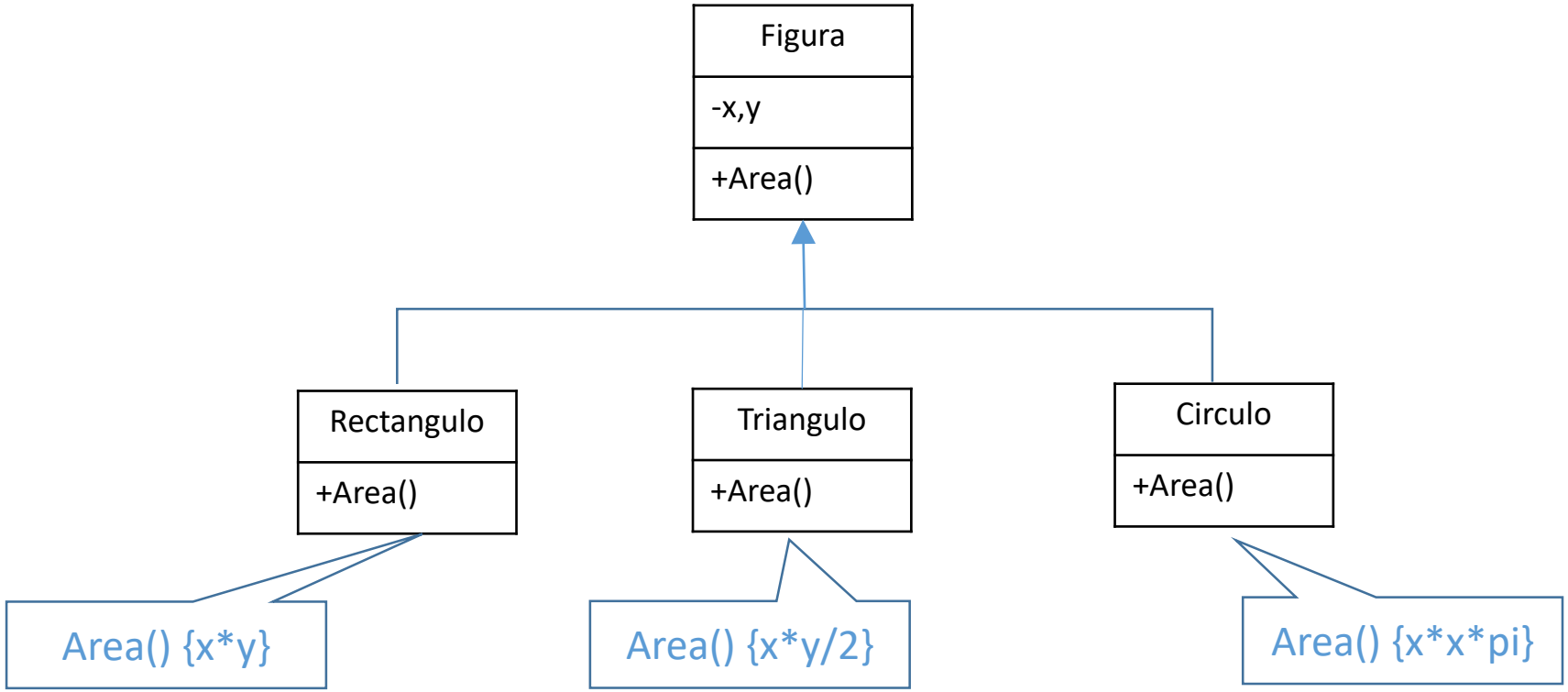
```
//ejemplo 03 de herencia
#include <iostream>
#include <stdlib.h>
using namespace std;
//Clase cVehiculo
class cVehiculo{
private:
   int    ruedas;
   int    pasajeros;
public:
   void   setRuedas(int x);
   int    getRuedas();
   void   setPasajeros(int);
   int    getPasajeros();
};
void cVehiculo::setRuedas(int num){
   ruedas=num;
}
int cVehiculo::getRuedas(){
   return    ruedas;
}
void cVehiculo::setPasajeros(int num){
   pasajeros=num;
}
int cVehiculo::getPasajeros(){
   return    pasajeros;
}
```

```
//clase cCamion con herencia  de cVehículo
class cCamion: public cVehiculo
{
private:
   int    carga;
public:
   void   setCarga(int);
   int    getCarga();
   void   muestra();
};
void cCamion::setCarga(int num){
   carga = num;
}
int cCamion::getCarga(){
   return carga;
}
void cCamion::muestra(){
   cout<<"Ruedas: "<< getRuedas()<<endl; //método heredado
   cout<<"Pasajeros:    "<<getPasajeros()<<endl; //método heredado
   cout<<"Capacidad de carga: "<<getCarga()<<endl;
}
int main(){
   cCamion ford;
   ford.setRuedas(6);    //método heredado
   ford.setPasajeros(3);    //método heredado
   ford.setCarga(3200);
   ford.muestra();
   system("pause");
}
```

POLIMORFISMO

El polimorfismo en tiempo de ejecución es logrado por una combinación de dos características: 'Herencia y funciones virtuales".

Una **función virtual** es una función que es declarada como 'virtual' en una clase base y es redefinida en una o mas clases derivadas. Además, cada clase derivada puede tener su propia versión de la función virtual. Lo que hace interesantes a las funciones virtuales es qué sucede cuando una es llamada a través de un puntero de clase base (o referencia). En esta situación, C++ determina a cual versión de la función llamar basándose en el tipo de objeto apuntado por el puntero.



```
//polimorfismo 01
#include <iostream>
#include <stdlib.h>

using namespace std;

class figura {
protected:
 double x, y;
public:
  void set_dim(double i, double j=0)
//valor por defecto j=0 caso del circulo sólo radio
   {
   x = i;
   y = j;
  }
  virtual void mostrar_area() {
   cout << "No hay calculo de area definido ";
   cout << " para esta clase.\n";
  }
};

class triangulo : public figura {

public:
   void mostrar_area() {
   cout << "Triangulo con alto ";
   cout << x << "  y base " << y;
   cout << " tiene un area de ";
   cout << x * 0.5 * y << ".\n";
  }
};

class rectangulo : public figura {

public:
  void mostrar_area() {
  cout << "Rectangulo con dimensiones ";
  cout << x << " x " << y;
  cout << " tiene un area de ";
  cout << x * y << ".\n";
  };
};

class circulo : public figura {
public:
  void mostrar_area() {
   cout << "Circulo con radio ";
   cout << x;
   cout << " tiene un area de ";
   cout << 3.14 * x * x;
  }
};

int main()
{
  figura *p;   // crear un puntero al tipo base

  triangulo t;  // crear objetos de tipos derivada
  rectangulo r;
  circulo c;

  p = &t;
  p->set_dim(10.0, 5.0);
  p->mostrar_area();

  p = &r;
  p->set_dim(10.0, 5.0);
  p->mostrar_area();

  p = &c;
  p->set_dim(9.0);
  p->mostrar_area();
  system("pause");
  return 0;

}
```