Clases y Objetos

2021

Por

Yolanda Martínez Treviño

Ma. Guadalupe Roque Díaz de León

¿Qué es un **objeto**?

- Objetos físicos: pelota, laptop, celular etc.

- Objetos abstractos: cuenta bancaria, criptomoneda, etc.

Ejemplo:

- Laptop

  Objeto Físico que tiene:

  - Atributos: procesador, RAM, velProcesador, modelo, precio, disco duro, puertos: usb, hdmi, etc.
  - Comportamientos: conectarse a internet, ejecutar aplicación, almacenar archivos, instalar sw, etc.

Ejemplo:

Calculadora

Objeto Físico que tiene:

Características: peso, tamaño, precio,…

Comportamientos: realizar cálculos, guardar datos en memoria.

Ejemplo:

Cuenta bancaria:

Objeto abstracto que tiene:

Características:

Nombre, saldo, RFC, dirección, etc.

Comportamientos: Depositar, Retirar, Consultar saldo, etc.

¿Qué es un **objeto** en OOP?

- Sabemos que "algo" es un objeto si tiene nombre, se describe en base a sus atributos y es capaz de llevar a cabo acciones.

Atributos y Métodos

Las características del objeto se llaman atributos o propiedades, son variables que almacenan el estado actual del mismo.

Los comportamientos se llaman métodos y se representan con funciones.

PLATILLOS

Características (atributos):
- Porción.
- Precio.
- Cantidad.

Comportamientos (métodos):
- Servir.
- Vender.
- Reservar.

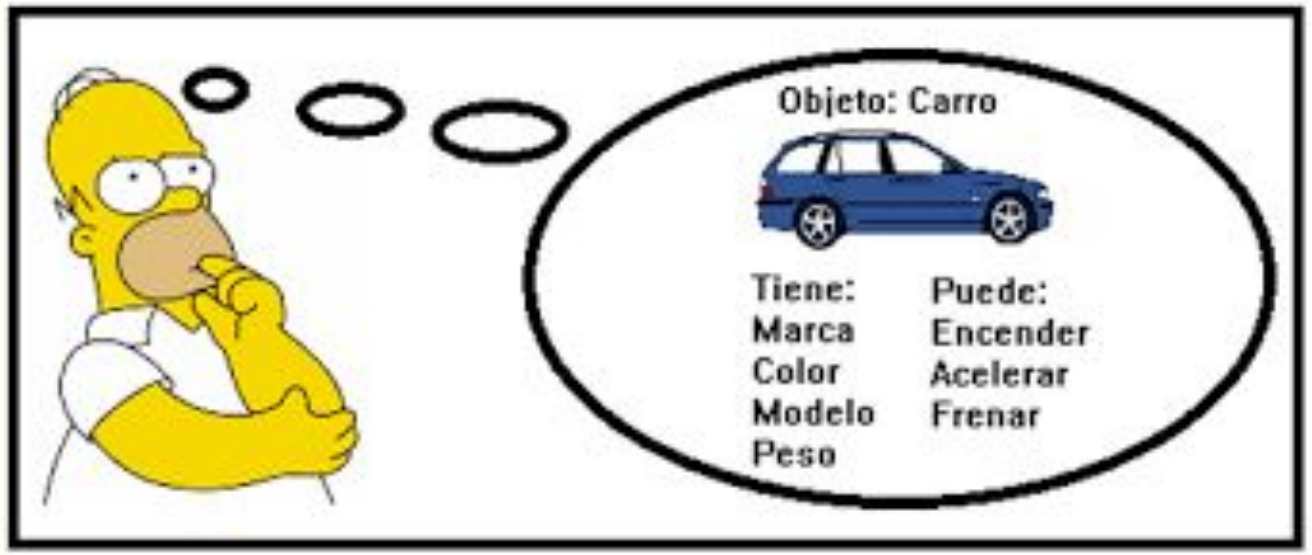
Clase: Línea_telefónica

- **Propiedades o Atributos:**
  Número de cuenta, nombre titular, Número teléfonico, Tipo de Contrato, Saldo Actual, e

- **Comportamiento o Funciones:**
  Realizar llamadas, Recibir llamadas, Consulta de Saldo, etc.

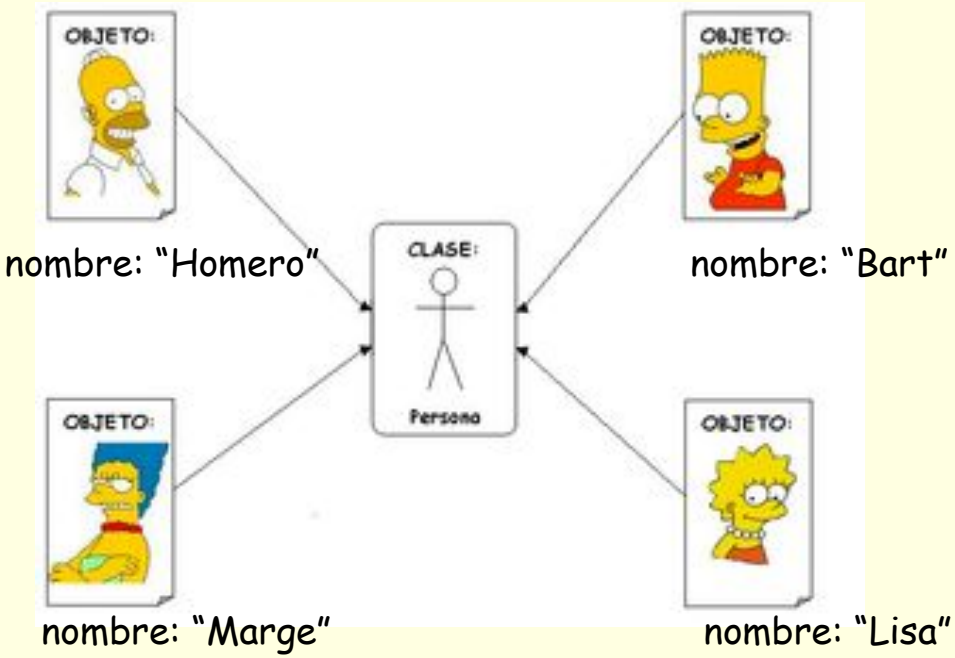
¿Qué es una **Clase** en programación?

- Las **clases** se utilizan para representar entidades o conceptos
- Cada **clase** es un modelo que define un conjunto de **variables** -el estado, y **métodos** apropiados para operar con dichos datos -el comportamiento.

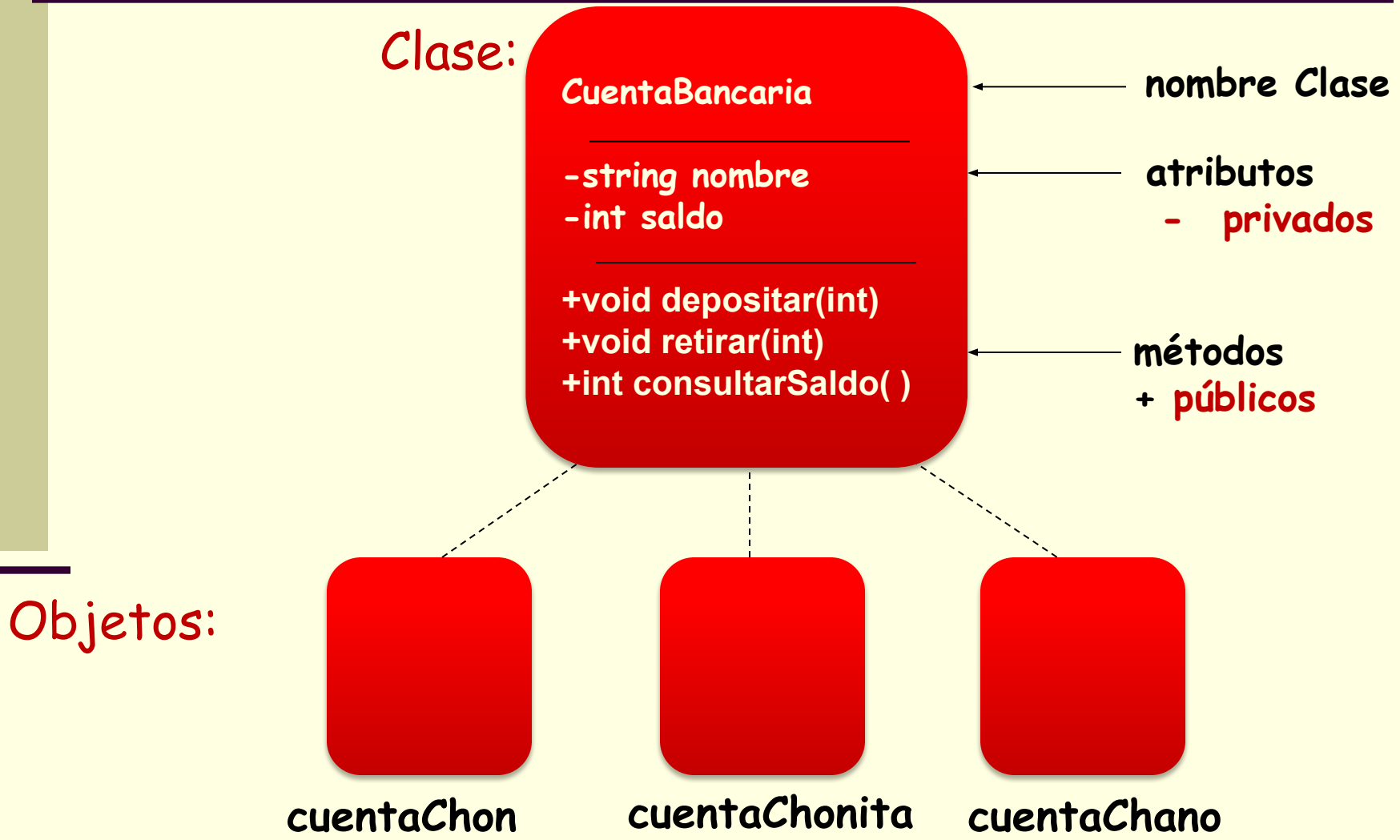


Clase - Objeto

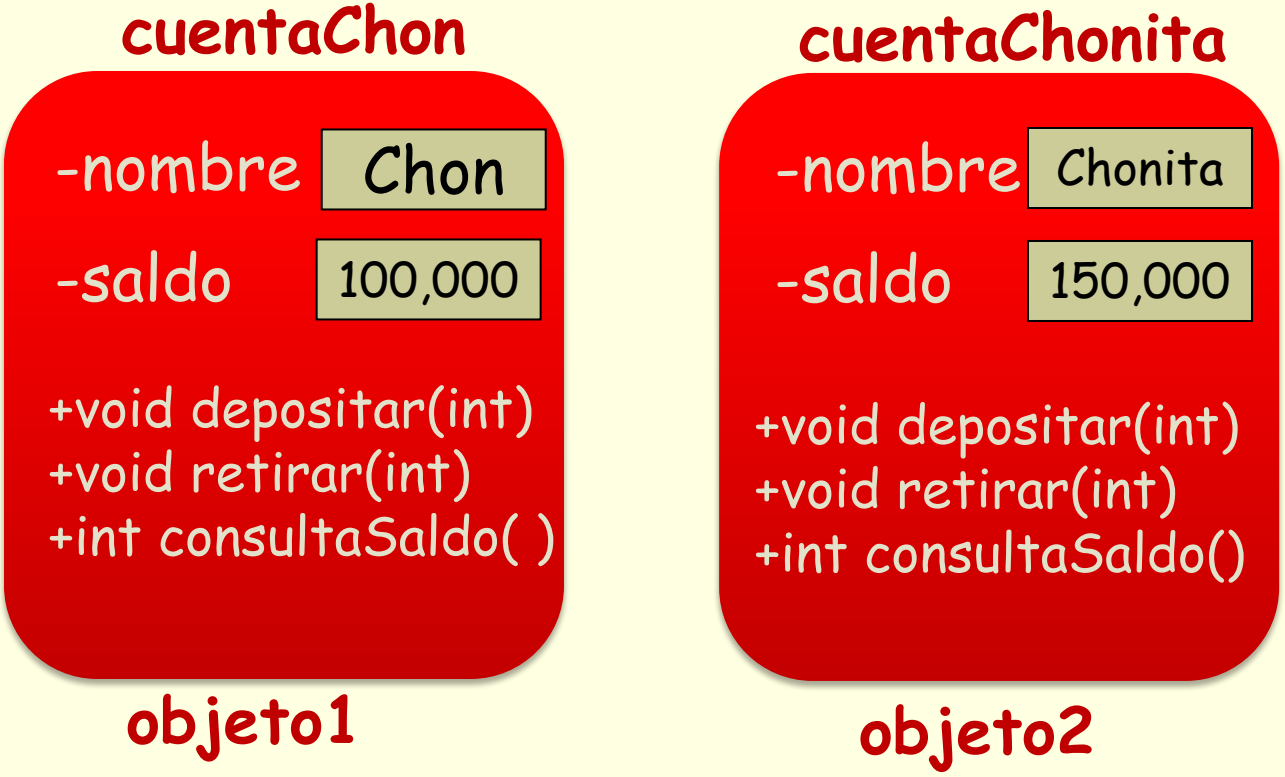
- Cada **objeto** creado a partir de la **clase** se denomina **instancia** de la **clase**.
- A un **objeto** que pertenece a una clase se le conoce como **instancia** de la clase. El objeto es la entidad concreta. Atributo nombre



Ejemplo: CuentaBancaria

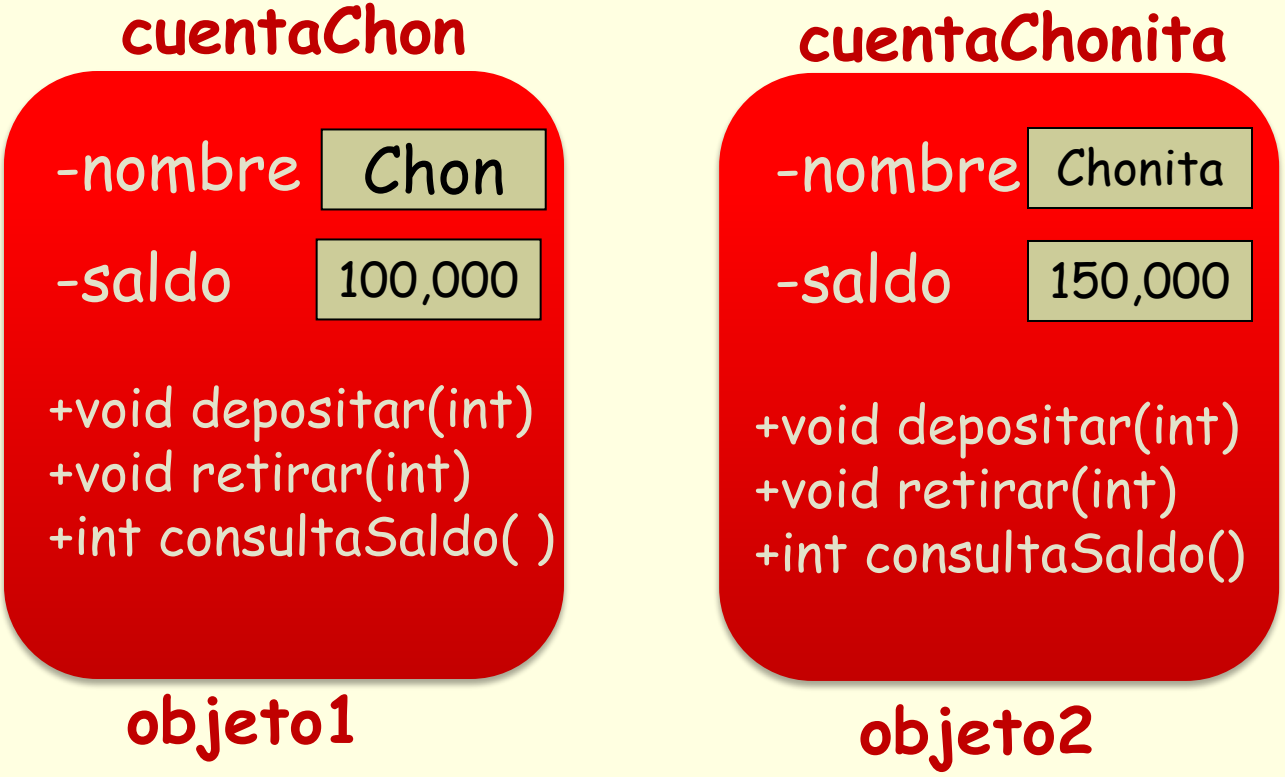


Objetos: instancias de la clase



Objetos : instancias de la clase

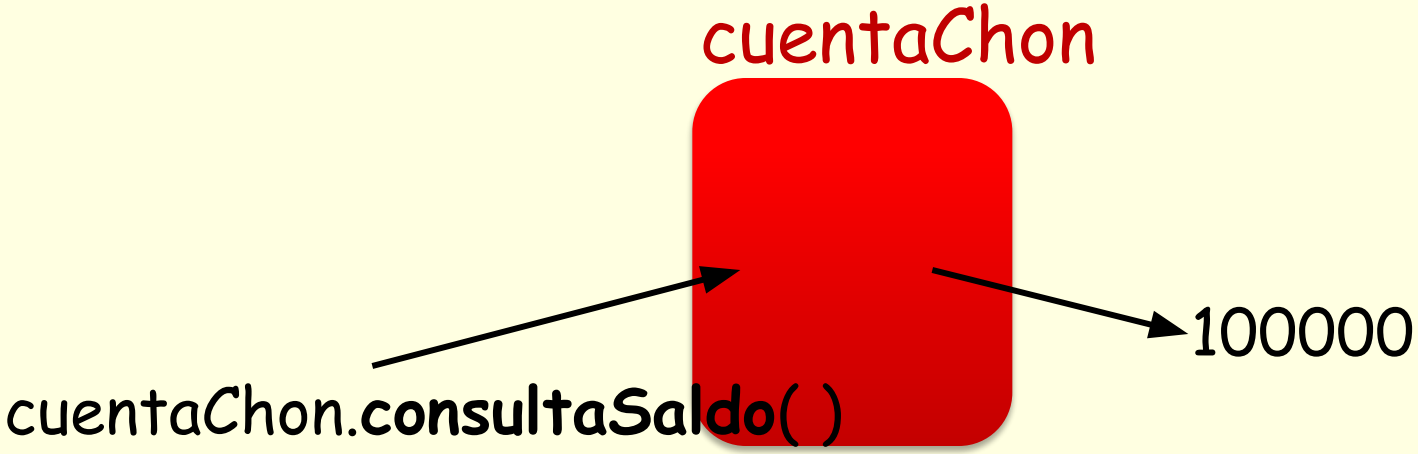
https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml



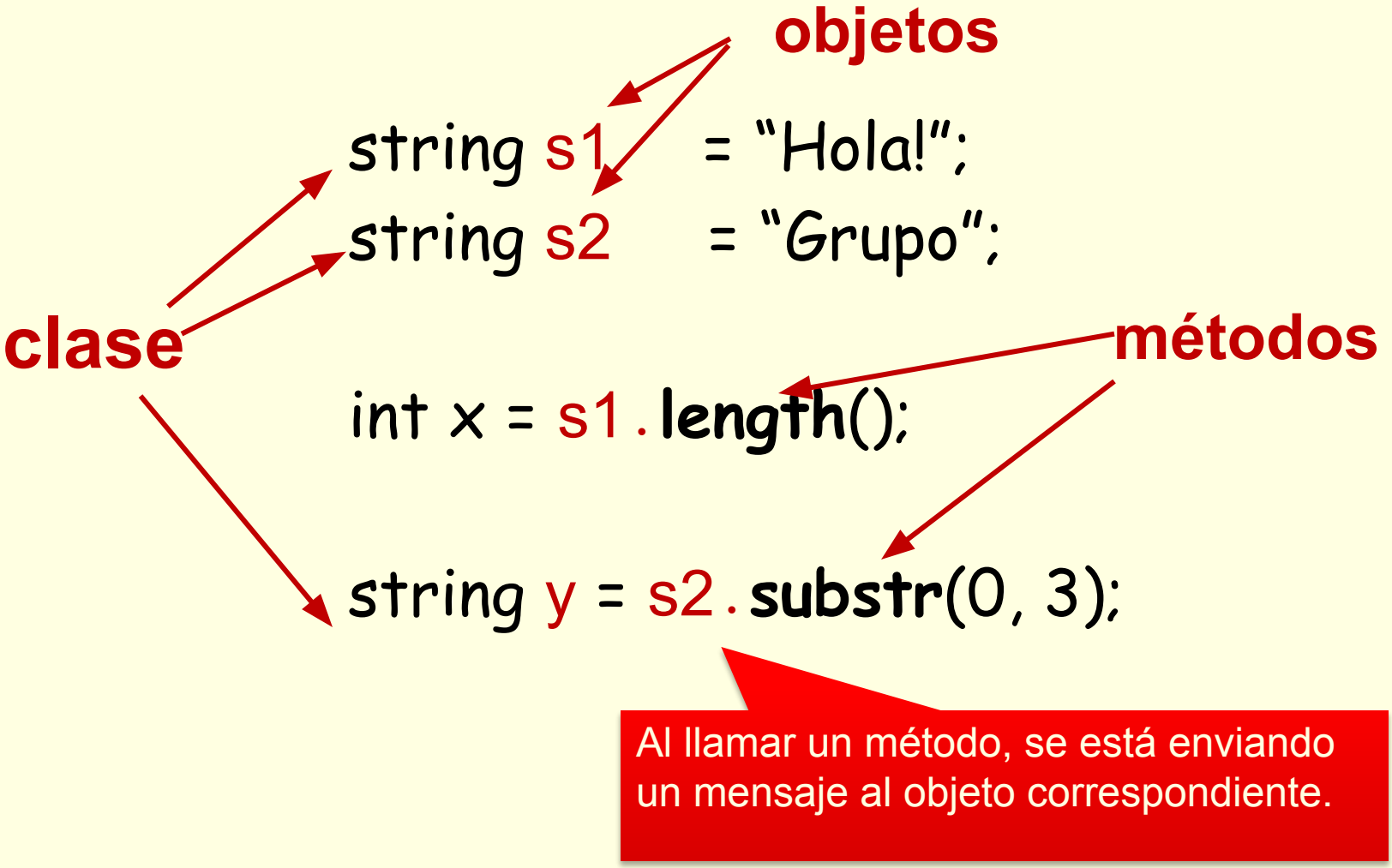
Ejecutar una acción sobre un objeto

- Un objeto ejecuta un método definido en la clase a la que pertenece usando la notación punto: objeto.metodo( )

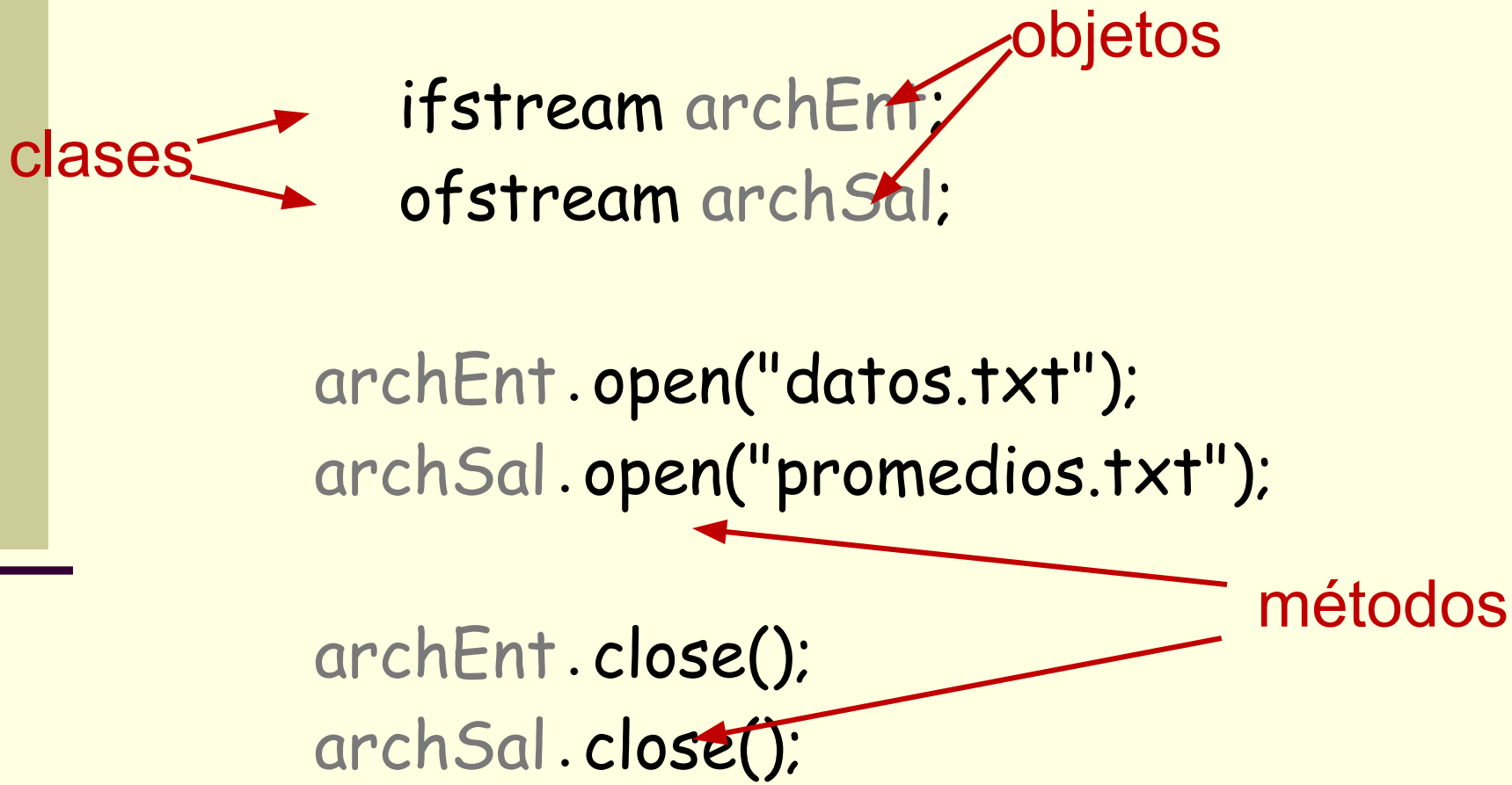
```
saldo = cuentaChon.consultaSaldo( );
```



¿Qué creen? ya hemos usado clases y objetos en C++ 😊



Ya hemos usado clases y objetos en C++




```
ifstream archEnt;
ofstream archSal;

archEnt.open("datos.txt");
archSal.open("promedios.txt");

archEnt.close();
archSal.close();
```

Resumen

- Los **atributos o propiedades** se definen como **datos miembro** de la clase, o sea como variables de la clase.
- Los **métodos** o **comportamientos** del objeto se definen como **funciones miembro** de la clase.
- Un método tiene derecho a usar los atributos -datos miembro - de la clase a la que pertenece.
- Para que un objeto ejecute un método se usa la notación . Sintaxis - objeto.**metodo**(...)

Ejemplo de la clase Rectángulo

Clase

Clase:
Rectangulo
Atributos:
-int largo,
-int ancho.
Métodos:
+double calcularArea( )
+string str( )

objeto o instancia

Objeto: rect
largo 10
ancho 20

objeto o instancia

Objeto: **rect1**
largo 5
ancho 8

Ambos objetos pueden recibir un mensaje, solicitando ejecutar el método **calcularArea**() y/o **str**()

Secciones public y private

- Los métodos y atributos de una clase pueden ser:
  - Privados: esto quiere decir que no pueden ser accesados desde afuera de la clase, esto se indica con la palabra **private**.
  - Públicos: los atributos pueden ser utilizados desde afuera de la clase, esto se indica con la palabra **public**.
- Si no se indica, por default los atributos de la clase son privados.
- Solo los métodos de la clase pueden accesar los atributos privados de la clase.

Modificadores de acceso

- Todas las clases poseen diferentes niveles de acceso en función del modificador de acceso (visibilidad).

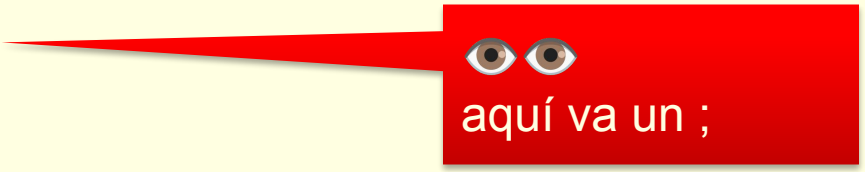
- Algunos de los niveles de acceso con símbolos correspondientes:
  - Público (+)
  - Privado (-)

Creando una clase en C++

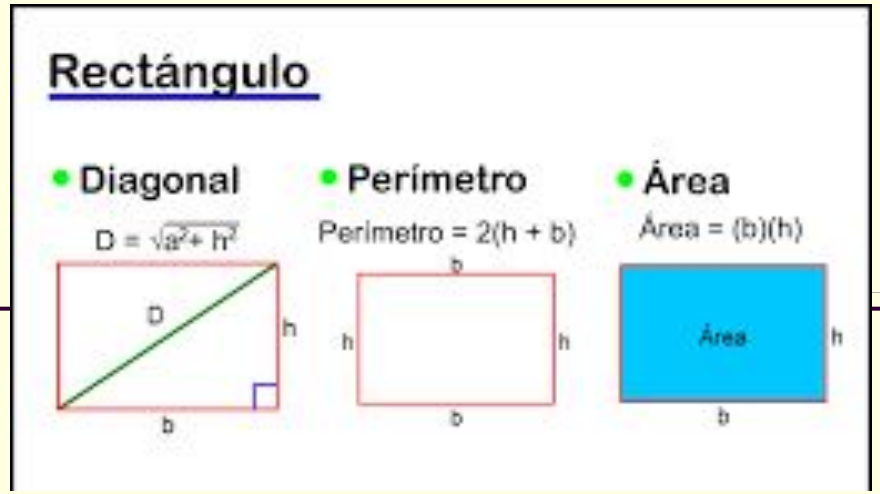
Sintaxis:

```
class NombreClaseNuevaHoy
{
    public:
    // declaración de atributos y métodos públicos

    private:
    // declaración de atributos y métodos privados
};
```



Ejemplo:




```
class Rectangulo
{
private:
        double largo, ancho;
public:
        double calcularArea( );
        void setLargo(double _largo);
        void setAncho(double _ancho);
        string str( );
};
```

Atributos de la Clase

Encabezados de los métodos de la Clase, *más adelante se define* su cuerpo:instrucciones.

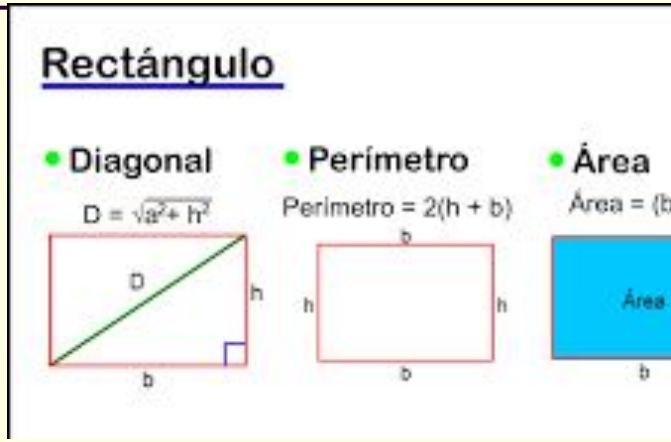
Declaración de las funciones miembro de una clase:

- Al declarar la clase, las **funciones miembro** no se declaran completas, solamente se declara su **encabezado**.
- Las funciones se añaden después, utilizando la siguiente **sintaxis**:

```
tipo NombreClase :: nombreMétodo ( lista parámetros)
{
    estatutos;
}
```

Definición de las funciones miembro de la clase Rectangulo:

```
double Rectangulo :: calcularArea( ){
      return largo * ancho;
}
void Rectangulo :: setLargo(double _largo){
      largo = _largo;
}
void Rectangulo :: setAncho(double _ancho){
      ancho = _ancho;
}
string Rectangulo :: str( ){
    return "Largo="+ to_string(largo) + "\nAncho =" + to_string(ancho);
}
```



Uso de los objetos

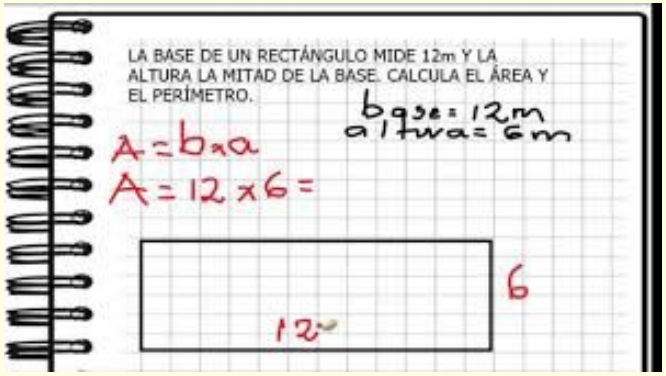
- Una instancia u objeto de la clase se declara como una variable.
- Para crear una instancia u objeto se usa igual que al declarar una variable - Sintaxis:

```
NombreClase   nombreVariable;
TipoDeDato    nombreVariable;
```

Con esta instrucción se declara una variable que es un objeto de la clase.

```
Rectangulo  rect;
```

Con esto se crea el objeto rect que es una instancia de la clase Rectángulo.



Uso de los objetos

- Pedirle a un objeto que ejecute uno de sus métodos – usa la siguiente sintaxis:
  - Método No retorna valor

objeto . **nombreMétodo** (parámetros);

  - Método retorna valor

variable = **objeto** . **nombreMétodo**(parámetro

llama a la función miembro a la que pertenece el objeto, usando la instancia – **objeto**.

Uso de clases: 🥳
#include "name.h"

- La declaración y definición de una clase se coloca en un archivo y se pone a ese archivo el nombre de la clase con extensión.h.

Ejemplo: Rectangulo.h

Programa que usa la clase

- Para usar la clase en un programa cliente se incluye la clase con un #include en la app.

Ejemplo: #include "Rectangulo.h"

🤓 delimitado por " "

Métodos de Acceso- getters

- Los métodos de acceso permiten **consultar** el valor de los atributos en la aplicación.

Sintaxis :

```
tipoAtributo NombreClase :: getAtributo() {
        return atributo;
}
```

Métodos de Acceso - clase Rectangulo

```
double Rectangulo::getAncho()
{
    return ancho;
}
double Rectangulo::getAlto()
{
    return alto;
}
```

Métodos Modificadores - **setters**

- Los métodos modificadores permiten **modificar** los atributos de un objeto.

Sintaxis:

```
void NombreClase :: setAtributo(tipoAtributo valor) {
        atributo = valor;
}
```

- El método modificador debe tener validaciones que garanticen que los datos que guarda cada atributo siempre sen valores correctos.

Métodos modificadores: clase Rectangulo

```
void Rectangulo::setAncho(double _ancho)
{
    ancho = _ancho;
}

// versión 2.0 con validación
void Rectangulo::setAncho(double _ancho)
{
    if (_ancho > 0)
        ancho = _ancho;
    else
        ancho = 1;
}
```

Ejemplo comp

Clase

Rectangulo.h

```
class Rectangulo{
public:
        double calcularArea( );
        void setLargo(double _largo);
        void setAncho(double _ancho);
        double getLargo( );
        double getAncho( );
        string str( );
private:
        double largo, ancho;
};
double Rectangulo::calcularArea( ){
    return largo * ancho;
}
void Rectangulo::setLargo(double _largo){
    largo = _largo;
}
void Rectangulo::setAncho(double _ancho){
    ancho = _ancho;
}
double Rectangulo::getLargo( ){
        return largo;
}
double Rectangulo::getAncho( ){
        return ancho;
}
string Rectangulo:: str( ){
     return "Largo="+ to_string(largo) +
              "\nAncho=" + to_string(ancho);
}
```

Ejercicio.cpp

Aplicación

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


int main( )
{    Rectangulo rect;
     double area;

     rect .setLargo(10);
     rect .setAncho(5);
     area = rect .calcularArea();
     cout << rect.str( ) <<
     "\nEl area es "<< area <<endl;
     return 0;
}
```

Aplicación – usando la clase Rectangulo

```
int main( )
{       Rectangulo rect;
        double area;

        rect. setLargo(10);
        rect. setAncho(5);
        area = rect●calcularArea( );
        cout << "El area es "<< area << endl;
        return 0;
}
```

Crea el objeto **rect** que es un Rectangulo

mensajes al objeto rect para que ejecute los métodos setLargo, setAncho y calcArea

https://www.learncpp.com/cpp-tutorial/variable-assignment-and-initialization/

CLASE RELOJ

Ing. Ma. Guadalupe Roque Díaz de León

Actividad de aprendizaje de Clases

Equipo de 2 integrantes: haciendo cada integrante su clase y app.

EJERCICIO: CREA LA CLASE RELOJD

Que contenga 2 atributos **privados**: **hora** y **minutos**, ambos de tipo **entero**. - **RelojD.h**

Agrega a la clase los siguientes métodos:

Métodos Modificadores:

- **void setHora(int)** : método que recibe un valor entero y se lo asigna al atributo hora. Este método no regresa ningún valor. – debe validar el valor: 0..23
- **void setMinutos(int):** método que recibe un valor entero y se lo asigna al atributo minu. Este método no regresa ningún valor - . – debe validar el valor: 0..59

Métodos de Acceso:

- **int getHora( ):** que no recibe parámetros y regresa como valor de retorno el valor del atributo hora.
- **int getMinutos( )** :que no recibe parámetros y regresa como valor de retorno el valor del atributo minu.

EJERCICIO: CREA LA CLASE RELOJD

- **Otros Métodos**:
- **void muestra( ):** método que despliega en la pantalla la hora en el formato usual (la hora, luego un caracter **":"** y luego los minutos); la hora y los minutos deben desplegarse con 2 dígitos.
- **void incrementaMinutos**( ): método que incrementa el atributo minutos en 1, en caso de ser 60 debe cambiar a 0 e incrementar la hora, en el caso de que la hora sea 23 la hora se debe cambiar a 0

EJERCICIO: CREA UNA APLICACIÓN PARA USAR LA CLASE RELOJ

Crea la aplicación UsaReloj.cpp que muestre el siguiente menu de opciones -

Crea un menu que tenga las siguientes opciones

1. **setHora**
2. **setMinutos**
3. **getHora**
4. **getMinutos**
5. **desplegar**
6. **Salir**

Ejercicio: Crea la clase Hora

Que contenga 2 atributos **privados**: **hora** y **minutos**, ambos de tipo **entero**.

Agrega a la clase los siguientes métodos **públicos**:

- **void setHora(int)** que recibe un valor entero y se lo asigna al atributo hora. Este método no regresa ningún valor.
- **void setMinutos(int)** que recibe un valor entero y se lo asigna al atributo minutos. Este método no regresa ningún valor.
- **int getHora( )** que no recibe parámetros y regresa como valor de retorno el valor del atributo hora.
- **int getMin( )** que no recibe parámetros y regresa como valor de retorno el valor del atributo minutos.
- **void muestra( )** que despliega la hora en el formato usual (la hora, luego un caracter ":" y luego los minutos); la hora y los minutos deben mostrarse con 2 dígitos. Ejemplo 09:09

Continua en la siguiente filmina…

Ejercicio: Crea una aplicación para usar la clase Hora

Crea la aplicación UsaHora.cpp

- Debe crear 2 objetos de clase Hora: **hrInicial** y **hrFinal**.
- Debe pedir al usuario la hora y los minutos iniciales y cambiar los valores de los atributos del objeto **hrInicial** usando los métodos set de la clase.
- Luego debe pedir al usuario la hora y los minutos finales y cambiar los valores de los atributos del objeto **hrFinal** usando los métodos set de la clase.
- Finalmente debe usar el método **str** de cada uno de los 2 objetos (**hrInicial** y **hrFinal**) y para desplegar en pantalla la hora inicial y final de un evento.

Ejercicio:

Cambia los métodos modificadores de la clase **Hora** para que incluyan las validaciones necesarias.