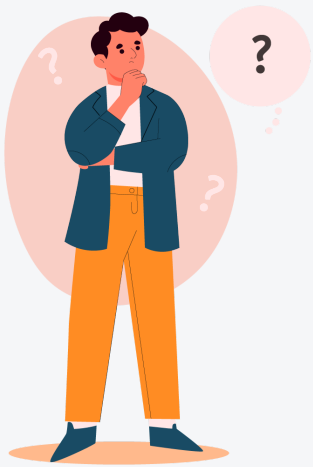
En foco: POO

Sitio: Agencia de Habilidades para el Futuro
Curso: Desarrollo de Sistemas Orientado a Objetos 1° D
Libro: En foco: POO

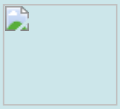
Imprimido por: Eduardo Moreno
Día: domingo, 23 de marzo de 2025, 10:00

Tabla de contenidos

Preguntas orientadoras



- ¿Por qué utilizamos un paradigma en la programación?
- ¿Qué función cumple una clase?
- ¿Qué papel juega la memoria en los objetos?
- ¿Se necesita un método? ¿Hay varios métodos?



Programación estructurada

La programación **tradicional, también denominada estructurada,** que se estudió en la materia de Técnica de Programación, consiste en trabajar con diferentes instrucciones que le indican a la computadora qué acción debe realizar.

Reflexionemos sobre la denominación "tradicional"

¿Por qué "tradicional"? Se considera tradicional a la programación que:

- es secuencial, modular y estructurada,
- organiza el código fuente de una aplicación en bibliotecas que contienen módulos (funciones, procedimientos),
- organiza estructuras de datos por separado.
- cada procedimiento es una caja negra que realiza una tarea,
- en ciertos casos accede a estructuras de datos globales y se comunican entre sí mediante el paso de parámetros.

Ahora bien, cuando se trabaja con programas simples y pequeños, esto es suficiente. Pero, cuando los programas son más complejos la cantidad de instrucciones necesarias crece y se vuelve inmanejable. En estos casos, los programas se particionan en unidades más pequeñas como las **funciones** o **procedimientos**, cada una de las cuales tiene un propósito bien definido. Las funciones suelen agruparse en módulos que se guardan en archivos distintos.

El principio siempre es el mismo: agrupar componentes que ejecutan una lista de instrucciones. La división de un programa en funciones y módulos es una de las características fundamentales de la programación estructurada y que facilita la lectura y comprensión del programa.

Sin embargo, cuando el problema a resolver es más complicado, la programación se hace difícil y excesivamente compleja. **Las dificultades provienen de las funciones que tienen acceso ilimitado a datos globales** y, además, el paradigma o enfoque procedimental proporcionan un modelo pobre del mundo real.

Otro paradigma: POO

Hasta aquí, venimos desarrollando el enfoque denominado tradicional o estructurado. Pero no es el único enfoque. En la programación tradicional la pregunta gira en torno a "¿Qué hace este programa?"; pero, ¡nosotros/as vamos a cambiar la pregunta!

Utilizaremos el enfoque **orientado a objetos** cuyo interrogante central es:
¿Qué objetos del mundo real pueden modelar (darle forma) al programa?

La POO se basa en el hecho que se debe dividir el programa no en tareas, sino en modelos de objetos físicos o conceptuales**.** Aunque esta idea parece abstracta a primera vista, se vuelve más clara cuando se consideran objetos físicos en términos de sus clases, componentes, propiedades y comportamiento, y sus objetos instanciados o creados de las clases.

En este sentido, cuando se trata de resolver un problema con orientación a objetos, dicho problema no se descompone en funciones como en programación estructurada o tradicional sino en objetos.

La POO (programación orientada a objetos) es una técnica o estilo de programación (un paradigma) que utiliza objetos como unidades fundamentales para diseñar y estructurar programas.

- Observar y pensar en términos de objetos tiene una gran ventaja: se asocian los objetos del problema a los objetos del mundo real.
- La correspondencia entre objetos de programación y objetos del mundo real es el resultado eficiente de combinar datos y funciones.
- Los objetos resultantes ofrecen una mejor solución al diseño del programa que en el caso de la programación estructurada.

A continuación, recuperemos esta idea con un ejemplo.

Entre datos, atributos y métodos. Un primer ejemplo

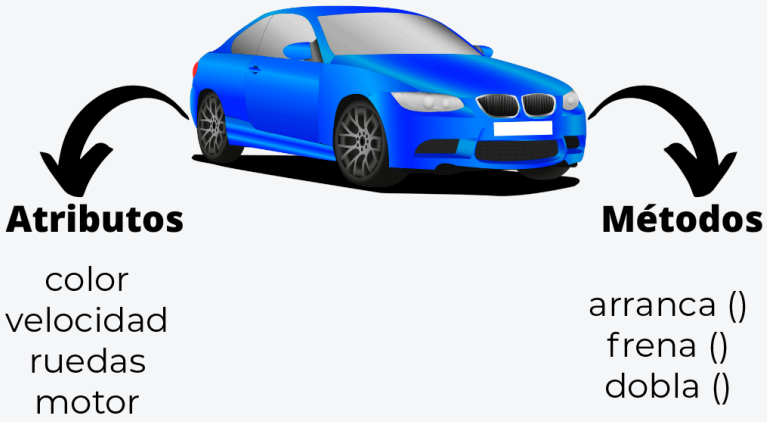
Tal como mencionamos anteriormente, la programación orientada a objetos será nuestro enfoque. En la POO, los objetos son entidades que combinan datos (atributos) y comportamiento (métodos) relacionados y se comunican entre sí a través de mensajes.

Analicemos el siguiente ejemplo:

¿Te acordás del video de introducción a la U1? Allí tomamos el **objeto auto**.

Observemos el esquema:

Ahora, listemos sus atributos y métodos:



Al dirigir nuestra mirada en términos de objetos, asociamos el auto (objeto) a un problema objeto del **mundo real**.

Ejercitando la mirada: ¿Cómo logramos construir esta mirada desde POO?

Algo ya te anticipamos en la semana anterior cuando trabajamos con el ejemplo: Sistema para un negocio e-commerce de ropa para niños y niñas (accedé desde **aquí** para leer el caso).

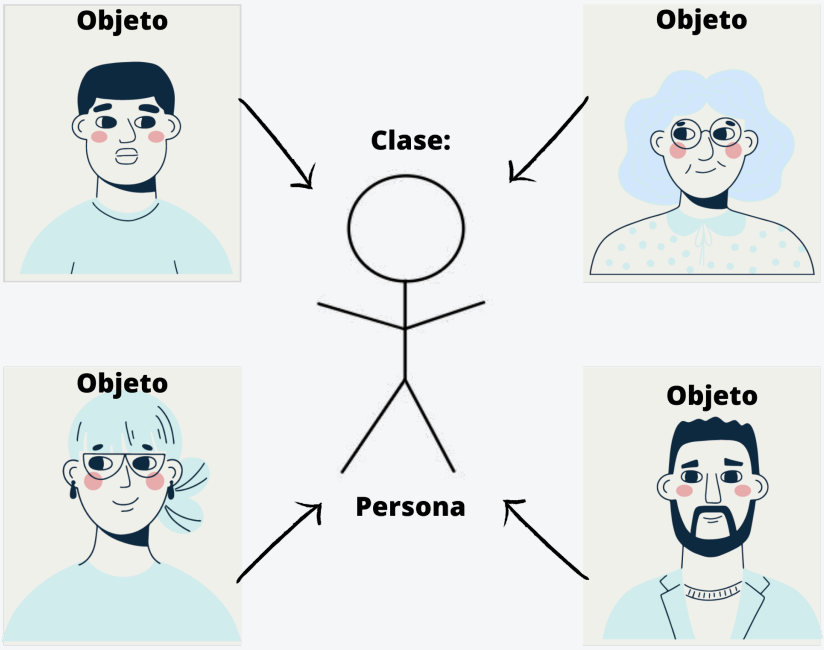
Relacionemos conceptos: Se parte de la creación de clases, que son, en realidad, tipos abstractos de datos donde se definen estos atributos y métodos. Un objeto se fabrica a partir "del molde" de la clase definida; a esto se lo conoce como "una instancia de una clase" que posee una copia de la definición de datos y métodos, considerándose estado a los datos con sus valores (atributos, propiedades o características) y comportamiento (métodos o funcionalidad) a los métodos que trabajan con estos datos.

Volviendo sobre nuestra mirada

La POO enfatiza, es decir ¡pone el ojo!, en los tipos de datos y las operaciones intrínsecas que pueden desarrollarse en aquellos mismos.

Los datos no fluyen libremente por el sistema (como sucede en el paradigma tradicional), ya que están protegidos de alguna modificación accidental; son los "mensajes" los que se pueden mover en el sistema, pues, se envía un mensaje a un objeto para que ejecute una acción y actúe sobre sus propios datos.

Veamos esta idea representada:



Como vemos en la representación, tenemos 4 objetos que fueron construidos a partir del "molde": la clase Persona.

A continuación, para comprender mejor cómo sucede esta construcción, profundicemos sobre qué son las clases.

Clases

En POO los **objetos son miembros de una clase**. En esencia, **una clase es un tipo de datos** como cualquier otro tipo de dato definido en un lenguaje de programación.

La diferencia es que la clase es un tipo de dato **que contiene atributos (datos) y métodos (funciones).**

Desarmemos la definición

Una clase es una descripción general de un conjunto de objetos similares. Por definición todos los objetos de una clase comparten los mismos atributos (aunque no los mismos valores) y las mismas operaciones (métodos).

¿Qué hace una clase?

Una clase encapsula las abstracciones de datos y operaciones necesarias para describir una entidad u objeto del mundo real.

A su vez, contiene la descripción de las características comunes de todos los objetos que pertenecen a ella:

- la especificación del comportamiento;
- la definición de la estructura interna y;
- la implementación de los métodos.

Veamos el siguiente esquema para integrar las características de una clase

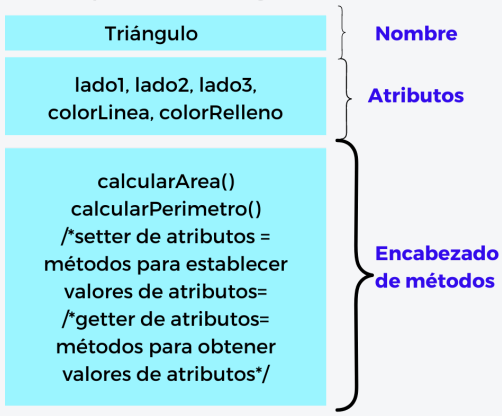
Clase

Una clase describe un conjunto de objetos comunes. Consta de:

- La declaración de las v.i. que implementan el estado del objeto.
- La codificación de los métodos que implementan su comportamiento.

Un objeto se crea a partir de una **clase**, decimos que el objeto es **instancia** de una clase.

- Representación gráfica



A continuación, pensemos estas características en nuestro lenguaje #c

Declarando una clase en C#

Podemos usar una clase para almacenar información se la debe definir y declarar antes que se pueda utilizar.

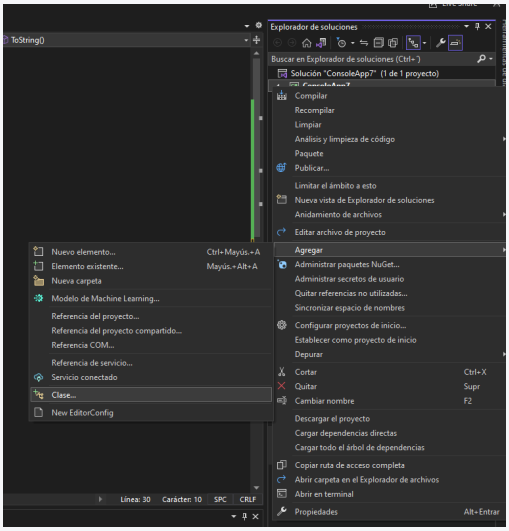
¡Seguí el paso paso. Desde el 1 al 6!

1. El formato de la declaración es:

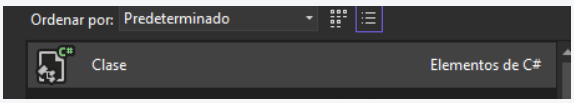
```
class <Nombre de la clase>
{
<tipo de dato miembro 1 > <nombre miembro 1 >
<tipo de dato miembro 2 > <nombre miembro 2 >
<tipo de dato miembro 3 > <nombre miembro 3 >
. . . . . .
<tipo de dato miembro n > <nombre miembro n >
};
```

2. Agregar una clase

Botón derecho en nuestro Proyecto -> agregar -> Clase….



3. Elegimos Clase c#



4. Debajo poner el nombre de la clase

- por ejemplo Persona.cs

5. Apretar el botón: Agregar

Nos queda:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApp7
{
    2 referencias
    internal class Persona
    {

    }
}
```

A continuación

6. Poner atributos a la clase Persona que estamos modelando

- para nuestro ejemplo nombre y edad:

```
2 referencias
internal class Persona
{
    private string nombre;
    private int edad;


}
```

Repasemos conceptos: a los dos atributos les pusimos el modificador de visibilidad **private**. Esto significa que ninguna otra clase ni el Main podrán conocer que atributos tiene nuestra clase y de esta manera mucho menos podrán modificarlo. ¡Esta acción es uno de los pilares de la POO y se llama **encapsulamiento!**

Hasta acá escribimos nuestra clase Persona pero no tenemos ningún objeto de esa clase.

A continuación, nos ocuparemos de eso.

Objetos

Comencemos por una definición

El objeto es el centro de la programación orientada a objetos. Un objeto es algo que se visualiza, se utiliza y juega un rol dentro del problema que se intenta resolver.

En el paradigma orientado a objetos (POO) el foco está puesto en **descubrir** e **implementar** los objetos que juegan un rol en el dominio del problema.

Ampliemos

- Un objeto es un elemento que encapsula información y comportamiento. Y es un término que representa una cosa concreta o del mundo real.

Relaciones ideas con un ejemplo:

Dependiendo del problema, nos interesará diferentes aspectos de un mismo objeto real. Por ejemplo, si se está desarrollando un **programa de administración de una concesionaria de autos**, seguramente tendremos el objeto auto y nos interesan aspectos tales como marca, modelo, motor, cantidad de puertas, color, precio, etc. En cambio, si se trabaja en un **sistema para competencias automovilísticas** tendremos también el objeto auto pero nos centraremos en aspectos tales como peso, velocidad, potencia, capacidad del tanque, tipo de combustible, etc. y si se trata de una **playa de estacionamiento**, nos bastará el número de patente y la hora de ingreso.

A los aspectos abordados en el ejemplo anterior se los denomina **atributos** y para leer su contenido desde cualquier parte del sistema que no sea de la propia clase, se deben utilizar los **métodos** incluidos en el objeto. Esto simplifica la escritura, depuración y mantenimiento del programa. Y a esto se lo denomina **encapsulamiento**.

¡Esto me suena! Estas definiciones ya te resultan familiares ¿no? Y es algo que te sucederá en la cursada, cada tema volverá en otro contexto y con otra complejidad.

¿Enviar un mensaje al objeto?

¿Sabías que cuando se realiza una llamada a un método de un objeto se dice que se le envía un mensaje al objeto?

Tal como venimos analizando, desde POO son los mensajes los que se pueden mover en el sistema, pues se envía un mensaje a un objeto para que ejecute una acción y actúe sobre sus propios datos.

Ampliemos

Un objeto no necesariamente será algo concreto o tangible. Puede ser algo abstracto y también puede describir un proceso.

Algunos ejemplos

- un partido de fútbol puede ser descrito como un objeto. Los atributos de este objeto pueden ser los jugadores, el entrenador, el tiempo transcurrido de partido, la cantidad de goles, etc.
- la cuenta de un banco será un objeto que no es tangible: Los atributos de este objeto es el o los titulares de la cuenta, un número, una descripción, tipo de cuenta, saldo. Y como métodos: ingresar dinero, extraer dinero, ver saldo disponible, etc.

Ejemplo de un objeto tangible:

Características

- **Raza**
- **Color**
- **Estatura**
- **Edad**

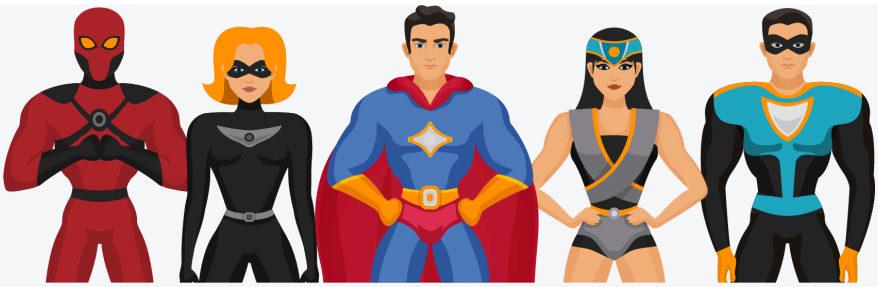
Acciones posibles

- **Comer**
- **Dormir**
- **Sentarse**
- **Ladrar**

Modelado de objetos no tangibles....incluso inexistentes en nuestras vidas.

Repasemos las ideas con un ejercicio.

A continuación, observá la imagen. Si tenemos la clase Superhéroe.
¿Te animás a encontrar al menos 5 atributos?



Ahora, veamos qué implica crear una representación de un objeto.

Instanciando la clase

¿Leíste alguna vez la palabra instanciación? **Instanciar una clase** en programación orientada a objetos (POO) **implica crear un objeto** específico basado en la plantilla o modelo definido por esa clase. En otras palabras, ¡materializarla!

> Una clase es solo una descripción o un plano de un objeto, mientras que una instancia es una representación concreta de ese objeto en memoria.

Cuando se instancia una clase, se reserva espacio en la memoria para almacenar el objeto y se inicializan sus atributos y propiedades según las definiciones establecidas en la clase. Cada instancia de la clase tiene su propio conjunto de valores para los atributos y puede ejecutar los métodos definidos en la clase.

Para instanciar la clase Persona desde el Main pondremos:

```
Persona unaPersona = new Persona("Jose", 40);
```

Analicemos cada una de las parte de la línea de código:

- **Persona** es el tipo de dato o nombre de la clase que se va a instanciar.
- **unaPersona** es la variable del tipo objeto Persona.
- **new** le solicita un lugar en memoria el sistema operativo para almacenar el objeto y se lo asigna a la variable miPersona (ver más abajo una explicación detallada).
- **Persona("Jose", 40)** llama al método constructor pasándole los valores que contendrán los atributos de ese objeto (se explica más detalladamente en el siguiente capítulo).

Ahora, repasemos los pasos que ocurren cuando se utiliza new:

1. **Reserva de memoria**: el operador new reserva espacio en la memoria para almacenar el objeto creado. Esto incluye asignar memoria para los campos, propiedades y otros miembros de la clase.

2. **Invocación del constructor**: después de reservar memoria, se invoca al constructor de la clase para inicializar el objeto. El constructor es un método especial que tiene el mismo nombre que la clase y se utiliza para realizar tareas de inicialización necesarias antes de que el objeto esté listo para su uso.

3. **Devolución de la instancia**: una vez que se ha completado la inicialización, el operador new devuelve la referencia a la instancia del objeto creado. Esta referencia se asigna a una variable, lo que permite acceder y manipular el objeto a través de esa variable.

A continuación, veámoslo en un método especial: utilizando constructores.

Un método especial: Constructores

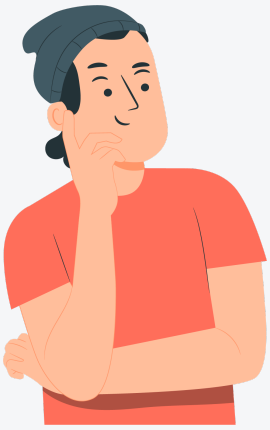
Un **constructor** en C# es un **método especial** de una clase que se llama automáticamente al crear una instancia (objeto) de esa clase utilizando la palabra clave new.

El constructor se utiliza para realizar tareas de inicialización del objeto y garantizar que esté en un estado válido y listo para su uso.

Algunas características importantes de los constructores:

1. **Nombre y firma**: un constructor **tiene el mismo nombre que la clase** y no tiene ningún tipo de datos de retorno, ni siquiera void. Esto distingue al constructor de otros métodos de la clase. Puede tener parámetros que permiten pasar argumentos durante la creación del objeto.
2. **Invocación automática**: el constructor se invoca automáticamente cuando se utiliza la palabra clave new para crear una instancia de la clase. No se llama explícitamente como un método normal.
3. **Múltiples constructores**: una clase puede tener varios constructores, cada uno con una firma de parámetros diferente. Esto se conoce como sobrecarga de constructores. Permite crear objetos de la clase con diferentes combinaciones de argumentos o sin argumentos.
4. **Inicialización de miembros**: el constructor se utiliza para asignar valores iniciales a los campos, propiedades u otros miembros de la clase. Puede realizar cualquier tarea de inicialización necesaria antes de que el objeto esté listo para su uso.

Un posible contructor:



En nuestro ejemplo de la clase Persona un posible constructor sería:

```
public Persona(string nombre, int edad)
{
    this.nombre = nombre;
    this.edad = edad;
}
```

Relacionemos ideas: observemos que es public, ya que debemos llamar al método constructor de la clase Persona desde otra clase. En nuestro caso del Main, cuando hacemos la instancia.

El nombre del método se llama exactamente igual que la clase. La palabra clave this.

En C#, la palabra clave `this` se utiliza para hacer referencia al objeto actual en el que se está ejecutando el código. Es una forma de referirse a sí mismo dentro de una clase y permite acceder a los miembros (atributos, propiedades, métodos) de la instancia actual.

Si bien hay varios usos del this, por ahora, señalaremos que estos nos sirven para distinguir cuando hay una ambigüedad de nombres entre un parámetro recibido y un atributo de la clase.

Relacionemos con un caso concreto

Si pusiésemos en nuestro constructor:

nombre = nombre

No tiene manera de poder distinguir cuál es el nombre como parámetro vs el nombre como atributo de la clase.

En nuestro ejemplo, `this.nombre` se refiere al atributo y se utiliza para distinguirlo con el parámetro nombre.

- `this` se refiere al objeto actual (instancia de la clase) y permite asignar el valor del parámetro recibido al atributo correspondiente.

A continuación, sigamos trabajando con otros métodos.

Métodos de get y set

Llegamos casi al final de nuestro módulo y necesitamos seguir ampliando nuestros métodos. En principio, recordemos que nuestra clase Persona tiene sus atributos privados con lo cual no se podrán modificar ni leer.

Y para implementar propiedades en una clase se deben utilizar los métodos get y set.

Dichos métodos proporcionan una forma de acceder y asignar valores a los atributos privados de una clase de manera controlada, lo que permite encapsular los datos y definir reglas específicas para el acceso y la modificación de esos datos.

Fundamentación de los métodos get y set

1. Método get: es utilizado para obtener (o leer) el valor de una propiedad o atributo. El método get se ejecuta cuando se accede al valor de la propiedad. Retorna el valor almacenado en el campo asociado a la propiedad.
2. Método set: es utilizado para asignar (establecer o escribir) un valor a una propiedad. El método set se ejecuta cuando se intenta asignar un valor a la propiedad. Recibe un parámetro que representa el nuevo valor que se desea asignar y actualiza el campo asociado a la propiedad con ese valor.

Revisemos lo visto con un caso concreto.

La sintaxis básica para definir una propiedad con los métodos get y set es la siguiente:

```
0 referencias
public void setNombre(string nombre)
{
    this.nombre = nombre;
}

0 referencias
public string getNombre()
{
    return nombre;
}
```

Otra forma de encapsular: Properties

Las propiedades (properties en inglés) en C# son miembros de una clase que proporcionan una forma de encapsular el acceso a los atributos privados de la clase controlando el acceso, validando los valores y realizar otras operaciones cuando se leen o se escriben.

Las propiedades combinan los conceptos de los métodos get y set en una sintaxis simplificada y más legible. Proporcionan una interfaz para leer y escribir valores como si fueran atributos públicos, pero internamente utilizan los métodos get y set para lograr el acceso controlado a los datos subyacentes.

	Ampliemos con un ejemplo

```
public string Nombre
{
    get { return nombre; }
    set { nombre = value; }
}
```

En este ejemplo, la clase Persona tiene un atributo private nombre (todo minúscula) y un atributo public Nombre (la primera letra en mayúscula) que encapsula el acceso a ese atributo.

La propiedad utiliza los métodos get y set para leer y escribir el valor de nombre. Al utilizar la propiedad, se obtiene el mismo resultado que si se estuviera accediendo directamente a un atributo público, pero internamente se ejecutan los métodos get y set para realizar las operaciones adecuadas.

Las propiedades permiten

- controlar el acceso a los atributos,
- aplicar validaciones,
- realizar cálculos adicionales y,
- mantener la coherencia interna en los datos de la clase.