



Metodologías de Programación I

Introducción a Objetos



Métodos.

Modificadores

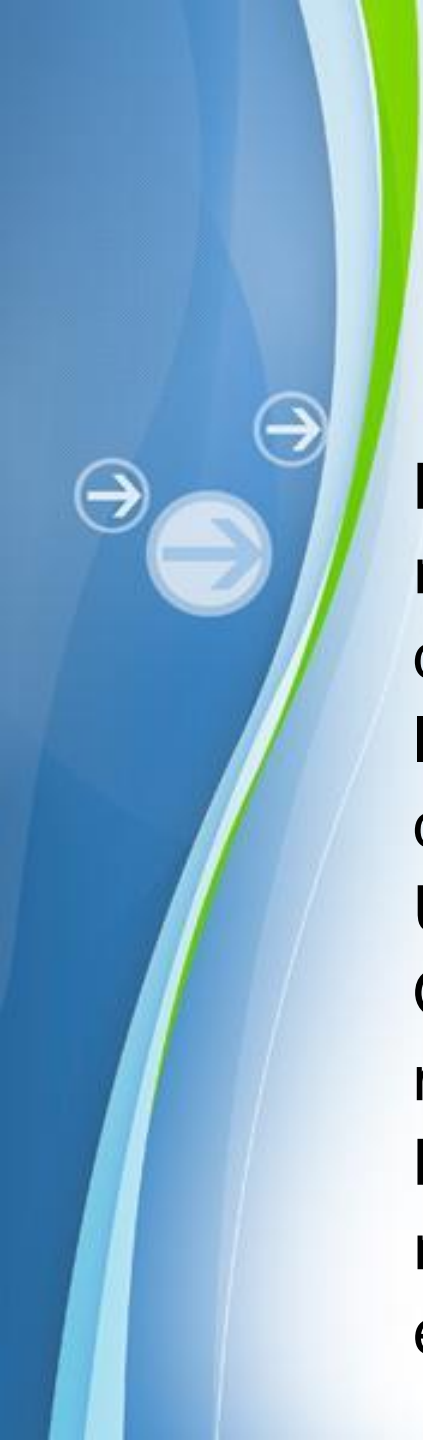
Variables

Pseudovariables

Instancias. Constructores.

Método Template y abstracto.

Métodos



Los métodos son la manera de especificar cómo responden a los mensajes los objetos de una determinada clase. Cada método especifica cómo se lleva a cabo la operación para responder a un determinado mensaje.

Un método puede acceder a la estructura interna del Objeto así como, también, enviarse mensajes a sí mismo o a otros objetos.

Los métodos describen, igualmente, cuál es la respuesta que recibe el emisor (el objeto que envía el mensaje).

Mensajes

- A través de mensajes los objetos solicitan servicios a otros objetos.
- Un mensaje tiene tres partes: identidad del receptor, método solicitado e información necesaria para el método (*parámetros*)
- Secuencia de actuación: el emisor envía el mensaje, el receptor ejecuta el método apropiado, el receptor devuelve una respuesta.

Modificadores de Accesibilidad

Los modificadores de acceso indican la visibilidad que una variable o un método tienen. Tanto los distintos tipos posibles, como la palabra reservada para denotarlos dependen, como es lógico, de cada lenguaje. De todos modos, todos los lenguajes OOP incluyen al menos los tres siguientes:

Públicos

Protegidos

Privados

Modificadores de Accesibilidad

Públicos

Son visibles dentro y fuera de la clase sin restricción alguna. La palabra reservada más común para denotarlos es "public".

Como ya hemos comentados, los datos no deben ser nunca públicos, ya que romperían el principio de encapsulación que debe seguir todo proyecto OOP.

Modificadores de Accesibilidad

Protegidos

Estos miembros de la clase (ya sean datos o métodos) son visibles desde dentro de la clase y desde cualquier otra clase heredada, es decir, clases hijas o subclases. La palabra reservada más común para denotarlos es "protected" o "friend".


Modificadores de Accesibilidad

Privados

Los miembros privados son solo accesibles desde dentro de la clase donde existen.


La palabra reservada más común para denotarlos es "private".

Ejemplos



```
Public String sNombre;  
Protected int nEdad;  
Private long nAcceso;  
Public void Imprimir();  
Protected int Calcular();  
Private string Grabar();
```


Modificadores de Contenido



Los modificadores de contenido afectan a cómo va a ser tratado el contenido de la variable. Así, una variable estática mantiene su contenido para todas las instancias de la clase que se hagan, así como para las subclases que de ella se hereden.

A estas, se les llama variables de la clase, como contraposición a las variables de instancia.


Modificadores de Contenido



Mientras que las **variables de instancia** se inicializan para cada nueva instancia que se haga de la clase, es decir existe una copia por cada instancia de la clase, de las **variables de la clase** existe una sola instancia, independientemente del número de instanciaciones que de la clase se hagan.

De este modo, todos los objetos comparten un lugar de almacenamiento común.

Modificadores de Contenido



El ejemplo más típico de variable de la clase es un contador del número de objetos existentes de la clase.

Para ello, sólo hay que incrementar el contador desde el constructor de la clase


Métodos



Método de Clase: se refiere a las porciones de código asociadas exclusivamente con una clase (se los denomina entonces métodos de clase o métodos estáticos)

Un ejemplo típico de un método de clase sería uno que mantuviera la cuenta de la cantidad de objetos creados dentro de esa clase.


Métodos



Métodos de **Instancia**: están relacionados con un objeto en particular, mientras que los métodos estáticos o de **clase** están asociados a una clase en particular.

En una implementación típica, a los métodos de instancia se les pasa una referencia oculta al objeto al que pertenecen, comúnmente denominada `this` o `self`, para que puedan acceder a los datos asociados con el mismo.


Variables de instancia



Las variables de instancia representan el estado del objeto y perduran durante toda la vida de éste.

Dos objetos diferentes, aunque pertenezcan a la misma clase, pueden tener valores diferentes en sus variables de instancia.

Variables de clase




Las variables de clase son compartidas por las instancias de una clase y sus subclases, manteniendo el mismo valor para todas las instancias.

Estas variables se declaran en la definición de la clase.

Ej: Mayoria de edad, cant limite de tarjeta de credito.

Pseudo-variables



Una pseudo-variable es un identificador que referencia a un objeto.

La diferencia con las variables normales es que no se pueden asignar y siempre aluden al mismo objeto.

Esto es, el valor de una pseudo-variable no puede modificarse con una expresión de asignación.

Pseudo-variables

“Pseudo-variables constantes”

- Nil / null. “Referencia a un objeto usado cuando hay que representar el concepto de ‘nada’ o de ‘vacío’. Las variables que no se asignaron nunca, referencian a nil”
- true. “Referencia a un objeto que representa el verdadero lógico.”
- false. “Referencia a un objeto que representa el falso lógico.”


Pseudo-variables

“Pseudo-variables no-constantes”

- Self/ this . “Referencia al receptor del mensaje.”
- super. “Referencia al receptor del mensaje, pero indica que no debe usarse la clase del receptor en la búsqueda del método a evaluar. Se usa, sobre todo, cuando se especializa un método en una subclase y se quiere invocar el método de la superclase.”

Al contrario que this, super permite hacer referencia a miembros de la clase padre (o a los ancestros anteriores, que no hayan sido ocultos por la clase padre) que se hayan redefinido en la clase hija. Si un método de una clase hija redefine un miembro –ya sea variable o método– de su clase padre, es posible hacer referencia al miembro redefinido anteponiendo super.

Constructores




Para poder utilizar un objeto, previamente tenemos que crearlo, lo hacemos mediante el constructor de la clase y puede haber más de un constructor según la sobrecarga de parámetros.

Para ello, dependiendo del lenguaje existen dos procedimientos:

a) Utilizando un método especial (normalmente con la palabra reservada "constructor"). Este método nos devuelve un objeto nuevo de esa clase. En este caso, a los métodos constructores se les suele llamar New().

Constructores



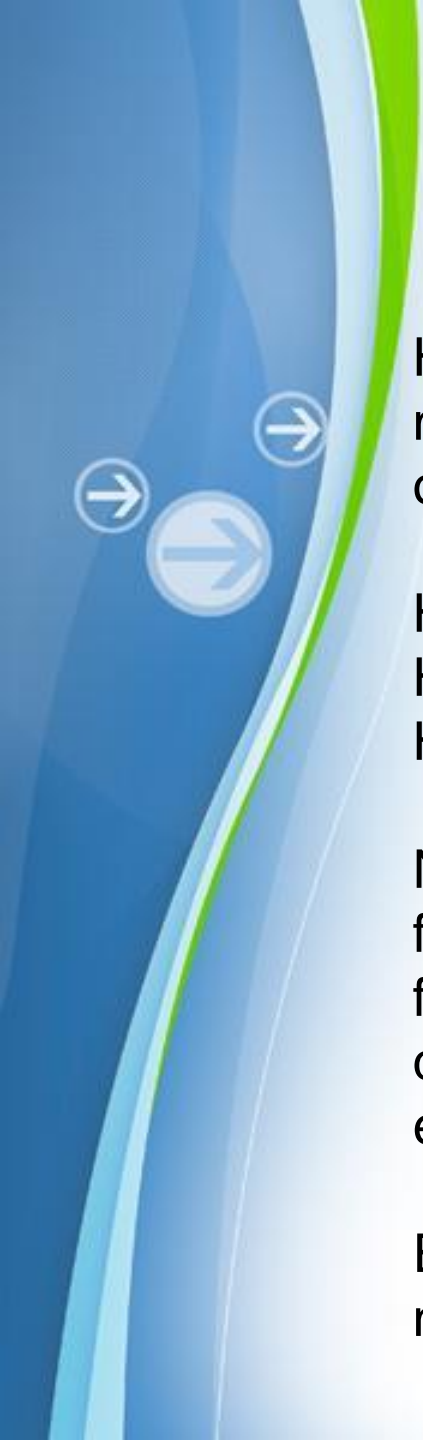
b) Utilizando un operador especial que el lenguaje proporciona y que normalmente se llama "new". Es este caso, el constructor o los constructores son notados de una forma especial: en Java, por ejemplo, se notan con el nombre de la clase y no devuelven ningún Tipo.

Así, para crear un objeto de la clase hombre, llamado Juan, escribiremos lo siguiente:

1. Hombre hmrJuan = Hombre.New();
2. Hombre hmrJuan = new Hombre();

Le estamos diciendo al método constructor que nos devuelva un nuevo objeto.

Constructores



Habitualmente, los constructores de clase se fabrican de tal modo que podamos hacer las dos cosas a la vez: crear el objeto y dar valores a sus datos:

```
Hombre hmrJuan = new Hombre( 30, 180, "Marrón" );  
Hombre hmrPepe = new Hombre( 12, 145, "Azul" );  
Hombre hmrAna = new Hombre( 24, 175, "Verde" );
```

Normalmente las clases tiene más de un constructor, de esta forma podemos crear objetos e inicializados de distintas formas. Así, podemos tener un constructor de la clase Hombre que recibe solo la edad, otro la edad y la estatura, otro la edad, la estatura y el color de ojos, etc.

El número y tipo de constructores solo depende de nuestras necesidades y del sentido común.


Constructor argumento-cero

Es aquel que no recibe ningún parámetro.

Es también importante, el concepto de "constructor por defecto".

Muchos lenguajes de OOP, permiten definir una clase sin crear un constructor para la clase. El lenguaje, entonces, utiliza el constructor por defecto (interno al lenguaje) para crear objetos de esa clase. Este método interno, normalmente se limita a reservar el espacio de memoria necesario para almacenar los datos del objeto, pero estos datos no están inicializados o no lo están correctamente, ya que el constructor por defecto no puede saber qué valores son los apropiados para los datos de la clase.

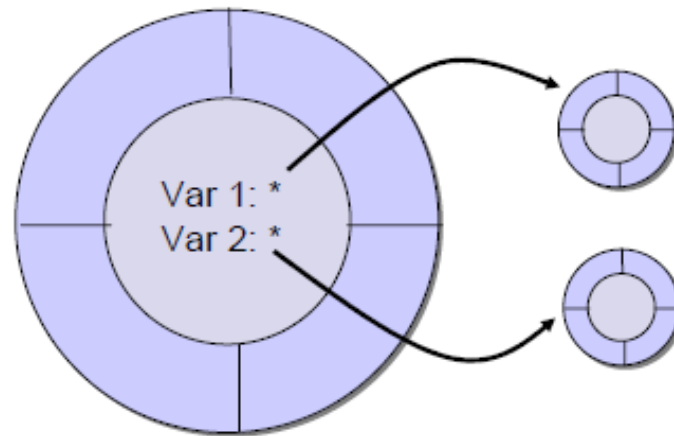
Constructor argumento-cero



En cualquier caso, la misión del constructor es construir adecuadamente el objeto, es decir, cuando el constructor haya terminado su trabajo, el objeto tiene que estar listo para ser usado.

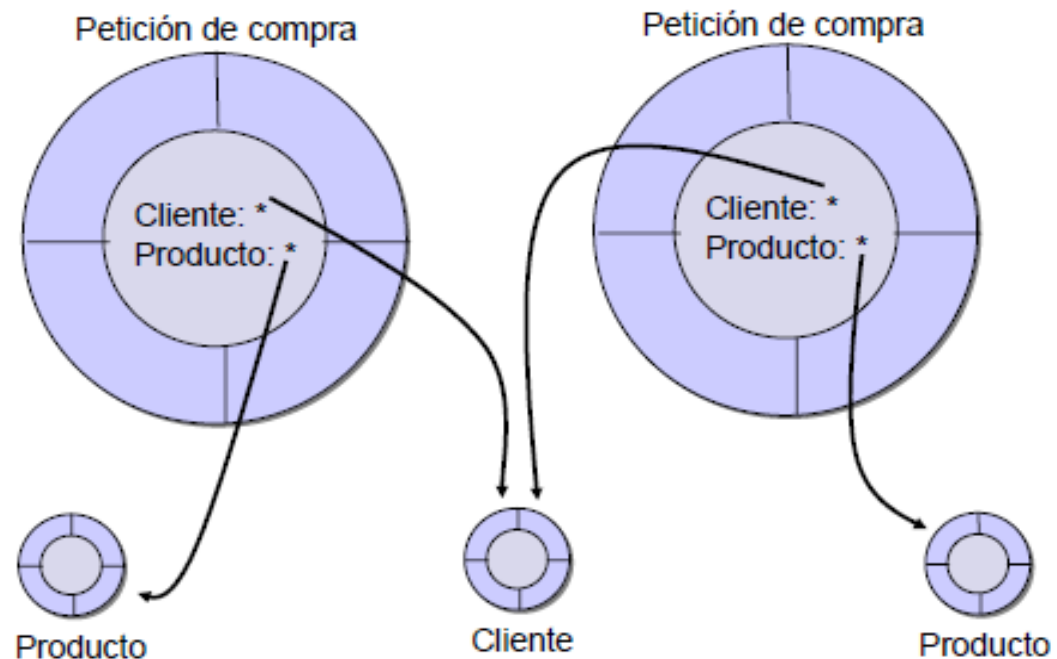
Constructor

Podemos construir objetos a partir de objetos, es decir un objeto puede contener a otro objetos. En la mayor parte de los sistemas, en realidad no los contiene, sino que los *referencia* (apunta)




- Ventajas de referenciar:
 - El objeto “contenido” puede cambiar de tamaño y composición sin que eso afecte al contenedor.
 - Los objetos “contenidos” pueden pertenecer a varios objetos a la vez.

Ejemplo de objetos compuestos




Método plantilla



Template Method (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.

Usando el Template Method, se define una estructura de herencia en la cual la superclase sirve de plantilla de los métodos en las subclases.

Método plantilla




Una de las ventajas de este método es que evita la repetición de código, por tanto la aparición de errores.

Se vuelve útil cuando es necesario realizar un algoritmo que sea común para muchas clases, pero con pequeñas variaciones entre una y otras .

Desarrollar Ejemplo de extraer de Cta Bancaria


Método abstracto



Es un método declarado en una clase para el cual esa clase no proporciona la implementación (el código).

Una clase abstracta es una clase que tiene al menos un método abstracto. Una clase que extiende a una clase abstracta debe implementar los métodos abstractos (escribir el código) o bien volverlos a declarar como abstractos, con lo que ella misma se convierte también en clase abstracta

Método abstracto



```
abstract class FiguraGeometrica {  
    ...  
    abstract void dibujar();  
    ...  
}
```

```
class Circulo extends FiguraGeometrica {  
    ...  
    void dibujar() {  
        // codigo para dibujar Circulo  
        ...  
    }  
}
```

Ejercicio

Dada la siguiente jerarquía de clases, analizar las sentencias y responder que resultado se obtiene al evaluarlas.



- a) Araña new manos : ?
- b) Mosca new orejas: ?
- c) Mosca new pies: ?
- d) Saltamontes new narices : ?
- e) Saltamontes new pies : ?
- f) Saltamontes new orejas : ?
- g) Saltamontes new sensores : ?

Clase Araña	Clase Mosca	Clase Saltamontes
ojos ^3	ojos ^6	ojos ^ self orejas
narices ^5	orejas ^super ojos	bocas ^super ojos
manos ^ self ojos	pies ^self manos	narices ^9
	antenas ^self ojos	sensores ^super antenas



Próxima Clase

Introd. a Colecciones.
Tipos de Colecciones .
Protocolo de Iteración