



UNJu
Universidad
Nacional de Jujuy



**Argentina
programa
4.0**



**Argentina
programa
4.0**

Clase 2

Contenido

Constructores, Destructores y Garbage Collector	1
Constructores y destructores	1
El Garbage Collector (GC)	3
Relación entre clases y objetos	3
Fundamentos de POO	7

Constructores, Destruktores y Garbage Collector

Constructores y destructores

Como ocurre con los objetos físicos, los objetos que se utilizan tienen un ciclo de vida definido. Este ciclo comienza con la instanciación del objeto y finaliza con su destrucción y limpieza por parte del Garbage Collector (es un proceso para reclamar el espacio de memoria de un objeto, cuando ya no exista ninguna referencia al objeto).

Es un tipo específico de método, cuyo nombre es el mismo nombre que la clase. Y que es utilizado cuando se quieren crear objetos de esa clase. Es utilizado o ejecutado al crear e iniciar dicho objeto.

Puede existir más de un constructor y se conocen como constructores múltiples, con o sin parámetros. Si no existe algún constructor, se puede proporcionar uno por omisión, el cual inicializa las variables del objeto con los valores que se dan predeterminadamente.

Para ejecutar instrucciones durante cada una de estas dos operaciones, existen métodos específicos: el constructor y el destructor.

1. Un CONSTRUCTOR es un método especial que permite instanciar un objeto. Su nombre es idéntico al de la clase en la que se ha definido, y no tiene ningún tipo de retorno. Puede recibir de 0 a n parámetros.
2. Un DESTRUCTOR es, por tanto, otro método particular cuya firma está impuesta. Su nombre es el mismo que el de la clase, precedido por el carácter ~. Además, no puede recibir ningún parámetro y, como el constructor, no tiene ningún tipo de retorno.

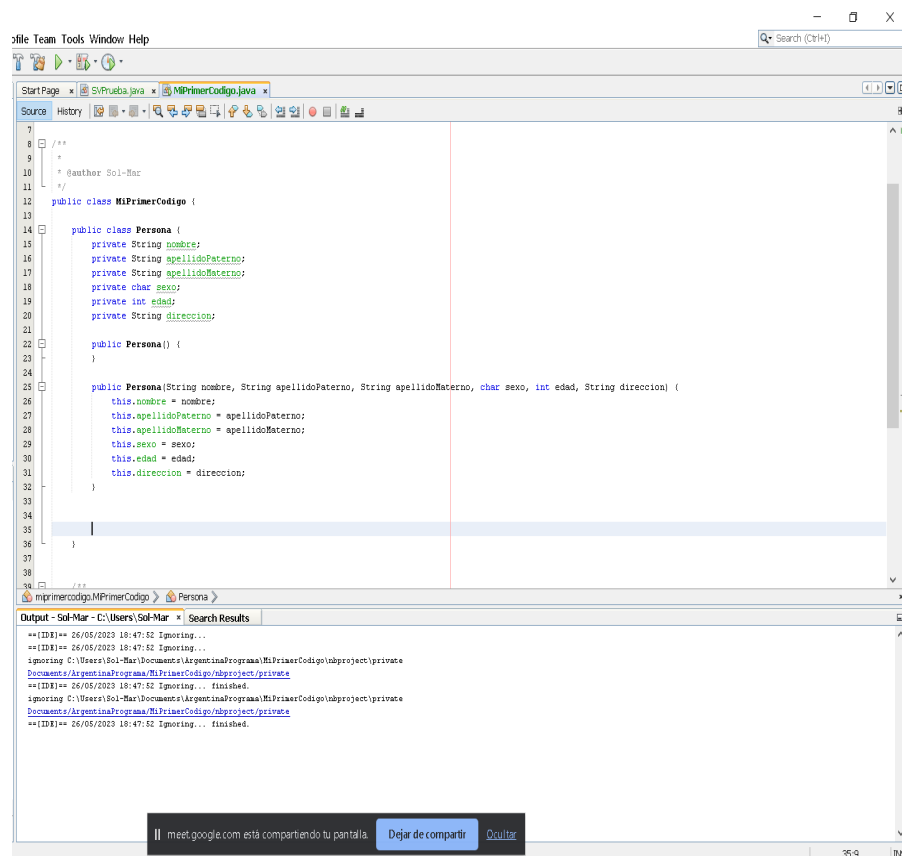
La destrucción de un objeto puede ser una operación automática. En java no existen los destructores, esto es gracias al recolector de basura de la máquina virtual de java.

Ejemplo de un constructor

<https://www.youtube.com/watch?v=4plcVlxh0lw>

Primero creamos una clase a la que llamaremos: Persona. En Netbeans creamos un proyecto y luego dentro de un paquete podremos crear la clase.

Una vez creado el paquete y la clase Persona, iniciaremos declarando sus atributos, es muy común que los atributos se declaren como privados, ya que se utilizan para las operaciones de los métodos dentro de la misma clase, así que en este caso los atributos serán privados.



```
1  /**
2   *
3   * @author Sol-Mar
4   */
5
6  public class MiPrimerCodigo {
7
8      public class Persona {
9          private String nombre;
10         private String apellidoPaterno;
11         private String apellidoMaterno;
12         private char sexo;
13         private int edad;
14         private String direccion;
15
16         public Persona() {
17             // Constructor vacío
18         }
19
20         public Persona(String nombre, String apellidoPaterno, String apellidoMaterno, char sexo, int edad, String direccion) {
21             this.nombre = nombre;
22             this.apellidoPaterno = apellidoPaterno;
23             this.apellidoMaterno = apellidoMaterno;
24             this.sexo = sexo;
25             this.edad = edad;
26             this.direccion = direccion;
27         }
28     }
29 }
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Output - Sol-Mar - C:\Users\Sol-Mar \ Search Results

```
==[IDE]== 26/05/2023 18:47:52 Ignoring...
==[IDE]== 26/05/2023 18:47:52 Ignoring...
Ignoring C:\Users\Sol-Mar\Documents\ArgentinaPrograma\MiPrimerCodigo\src\private
Documents\ArgentinaPrograma\MiPrimerCodigo\src\private
Documents\ArgentinaPrograma\MiPrimerCodigo\src\private
==[IDE]== 26/05/2023 18:47:52 Ignoring... finished.
Ignoring C:\Users\Sol-Mar\Documents\ArgentinaPrograma\MiPrimerCodigo\src\private
Documents\ArgentinaPrograma\MiPrimerCodigo\src\private
Documents\ArgentinaPrograma\MiPrimerCodigo\src\private
==[IDE]== 26/05/2023 18:47:52 Ignoring... finished.
```

Ahora creamos los diferentes constructores.

```
//Constructor por defecto
```

```
public Persona(){  
  
}
```

```
//Constructor de copia
```

```
public Persona(Persona persona){  
  
    this.nombre=persona.nombre;  
  
    this.apellidoPaterno=persona.apellidoPaterno;  
  
    this.apellidoMaterno=persona.apellidoMaterno;  
  
    this.sexo=persona.sexo;  
  
    this.edad=persona.edad;  
  
    this.direccion=persona.direccion;  
  
}
```

```
//Constructor común
```

```
public Persona(String nom, String app, String apm, char sexo, int edad, String dir){  
  
    this.nombre=nom;  
  
    this.apellidoPaterno=app;  
  
    this.apellidoMaterno=apm;
```

```
this.sexo=sexo;  
  
this.edad=edad;  
  
this.direccion=dir;  
  
}
```

Pero ¿es posible crear instancias de la clase Persona si no se ha definido explícitamente el constructor? No es obligatorio definir CONSTRUCTORES y en el caso en que no se definan (como en el ejemplo anterior de la clase Persona) se brindará uno sin parámetros y de cuerpo vacío (constructor por defecto). Dicho CONSTRUCTOR sólo se ocupa de asignarle a todos los campos sus valores por defecto.

¿Qué sucede si los parámetros del constructor se denominan igual que los campos?
¿Dentro del cuerpo del constructor a quien se haría referencia: al campo o al parámetro?

La respuesta a estos interrogantes se puede obtener a partir de los conceptos de alcance (ámbito) y tiempo de vida de las variables.

1. Tiempo de vida de las variables: Es el tiempo durante el cual se puede hacer referencia a la variable y va desde su creación hasta que dejan de existir. Para el caso de los campos de los objetos, estos existen durante toda la vida de los objetos. Pero en el caso de las variables locales (ejemplo, los parámetros), su tiempo de vida es más limitado pues dejan de existir después que finalice el bloque de instrucciones al que pertenece.

2. Alcance o ámbito: Este concepto está relacionado con la visibilidad de las variables y especifica desde qué código es posible hacer referencia a la variable.

Persona personaCopia=new Persona("Ricardo", "González","Mercado");

De éste modo se inicializan los valores de la persona con los datos pasados como parámetros.

Destructores

Un destructor es un método opuesto a un constructor, éste método en lugar de crear un objeto lo destruye liberando la memoria de nuestra computadora para que pueda ser utilizada por alguna otra variable u objeto.

En java no existen los destructores, esto es gracias al recolector de basura de la máquina virtual de java. Como su nombre lo dice, el recolector de basura recolecta todas las variables u objetos que no se estén utilizando y que no haya ninguna referencia a ellos por una clase en ejecución, liberando así automáticamente la memoria de nuestra computadora.

Aunque Java maneja de manera automática el recolector de basura, el usuario también puede decir en que momento Java pase el recolector de basura con la instrucción.

`System.gc();`

Aunque la instrucción anterior poco o casi nunca se utiliza es importante saber que se puede llamar en cualquier momento.

El Garbage Collector (GC)

Supervisa la totalidad de instancias creadas a lo largo de la ejecución de la aplicación y marca como huérfano cada objeto que no se utiliza. Cuando el sistema necesita más

memoria, el GC destruye aquellos objetos que considera que es posible destruir invocando a sus destructores, si los tienen.

Video explicativo de como funciona **Garbage Collector en NetBeans**

<https://www.youtube.com/watch?v=hq8AJUmPcWQ>

Relación entre clases y objetos

Repasemos un poco algunas definiciones...

Un objeto es una INSTANCIA de una clase. Por lo tanto, los objetos hacen uso de los Atributos (variables) y Métodos (Funciones y Procedimientos) de su correspondiente Clase.

También lo podemos pensar como una variable de tipo clase. Por ejemplo: el objeto Cesar es un objeto de tipo clase: Persona.

Como se puede observar, un objeto a través de su CLASE está compuesto por 2 partes: Atributos o estados y Métodos que definen el comportamiento de dicho objeto a partir de sus atributos. Los atributos y los métodos pueden ser o no accedidos desde afuera dependiendo de la solución a plantear.

Por lo general los atributos siempre se ocultan al exterior y algunos métodos quedan visibles al exterior para convertirse en la interfaz del objeto (Encapsulamiento).

Algorítmicamente, las clases son descripciones netamente estáticas o plantillas que describen objetos. Su rol es definir nuevos tipos conformados por atributos y operaciones.

Por el contrario, los objetos son instancias particulares de una clase. Las clases son una especie de molde de fábrica, en base al cual son construidos los objetos.

Durante la ejecución de un programa sólo existen los objetos, no las clases.

La declaración de una variable de una clase NO crea el objeto.

La creación de un objeto debe ser indicada explícitamente por el programador, de forma análoga a como inicializamos las variables con un valor dado, sólo que para los objetos se hacen a través de un método CONSTRUCTOR.

Clases, objetos, métodos y variables de instancia

Comencemos con una analogía simple, para comprender el concepto de las clases y su contenido:

Supongamos que deseas conducir un automóvil y, para hacer que aumente su velocidad, debes presionar el pedal del acelerador. ¿Qué debe ocurrir antes de que puedas hacer esto? Bueno, antes de poder conducir un automóvil, alguien tiene que diseñarlo. Por lo general, un automóvil empieza en forma de dibujos de ingeniería, similares a los planos de construcción que se utilizan para diseñar una casa. Estos dibujos de ingeniería incluyen el diseño del pedal del acelerador, para que el automóvil aumente su velocidad. El pedal “oculta” los complejos mecanismos que se encargan de que el automóvil aumente su velocidad, de igual forma que el pedal del freno “oculta” los mecanismos que disminuyen la velocidad del automóvil y por otro lado, el volante “oculta” los mecanismos que hacen que el automóvil de vuelta. Esto permite que las personas con poco o nada de conocimiento acerca de cómo funcionan los motores puedan conducir un automóvil con facilidad.

Desafortunadamente, no se pueden conducir los dibujos de ingeniería de un auto.

Antes de poder conducir un automóvil, éste debe construirse a partir de los dibujos de ingeniería que lo describen. Un automóvil completo tendrá un pedal acelerador verdadero para hacer que aumente su velocidad, pero aun así no es suficiente; el automóvil no acelerará por su propia cuenta, así que el conductor debe oprimir el pedal del acelerador.

Ahora utilicemos nuestro ejemplo del automóvil para introducir los conceptos clave de programación:

Para realizar una tarea en una aplicación se requiere un método. El método describe los mecanismos que se encargan de realizar sus tareas; y oculta al usuario las tareas complejas que realiza, de la misma forma que el pedal del acelerador de un automóvil oculta al conductor los complejos mecanismos para hacer que el automóvil vaya más rápido. Empezamos por crear una unidad de aplicación llamada clase para alojar a un método, así como los dibujos de ingeniería de un automóvil alojan el diseño del pedal del acelerador. En una clase se proporcionan uno o más métodos, los cuales están diseñados para realizar las tareas de esa clase. Por ejemplo, una clase que representa a una cuenta bancaria podría contener un método para depositar dinero en una cuenta, otro para retirar dinero de una cuenta y un tercero para solicitar el saldo actual de la cuenta.

Así como no podemos conducir un dibujo de ingeniería de un automóvil, tampoco podemos “conducir” una clase. De la misma forma que alguien tiene que construir un automóvil a partir de sus dibujos de ingeniería para poder conducirlo, también debemos construir un objeto de una clase para poder hacer que un programa realice las tareas que la clase le describe cómo realizar. Como ya mencionamos antes, esta es una de las características de un lenguaje de programación orientado a objetos.

Cuando conduces un automóvil, si oprimes el pedal del acelerador se envía un mensaje al automóvil para que realice una tarea (hacer que el automóvil vaya más rápido). De manera similar, se envían mensajes a un objeto; cada mensaje se conoce como la llamada a un método, e indica a un método del objeto que realice su tarea.

Hasta ahora, hemos utilizado la analogía del automóvil para introducir las clases, los objetos y los métodos. Además de las capacidades con las que cuenta un automóvil, también tiene muchos atributos como: su color, el número de puertas, la cantidad de gasolina en su tanque, su velocidad actual y el total de kilómetros recorridos (es decir, la lectura de su odómetro). Al igual que las capacidades del automóvil, estos atributos se representan como parte del diseño en sus diagramas de ingeniería. Cuando se conduce un automóvil, estos atributos siempre están asociados con él. Cada uno mantiene sus propios atributos. Por ejemplo, cada conductor sabe cuánta gasolina tiene en su propio tanque, pero no cuánta hay en los tanques de otros automóviles.

De manera similar, un objeto tiene atributos que lleva consigo cuando se utiliza en un programa. Éstos se especifican como parte de la clase del objeto. Por ejemplo, un objeto tipo cuenta bancaria tiene un atributo llamado saldo, el cual representa la cantidad de dinero en la cuenta. Cada objeto tipo cuenta bancaria conoce el saldo en la cuenta que representa, pero no los saldos de las otras cuentas en el banco. Los atributos se especifican mediante las variables de instancia de la clase.

Fundamentos de POO

POO se ha convertido durante las pasadas dos décadas en el paradigma de programación dominante, y en una herramienta para resolver la llamada crisis del software. Algunos motivos son: por la escalabilidad, porque proporciona un modelo de abstracción que razona con técnicas que la gente usa para resolver problemas, y por el gran desarrollo de herramientas OO (IDEs, librerías,...) en todos los dominios.

“El paradigma orientado a objetos es una metodología de desarrollo de aplicaciones en la cual éstas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representan una instancia de alguna clase, y cuyas clases son miembros de jerarquías de clases unidas mediante relaciones de herencia” (Grady Booch)

Veamos un ejemplo de la vida cotidiana: “Supongamos que Luis quiere enviar flores a María, que vive en otra ciudad”. Luis entonces irá a la floristería más cercana, que es administrada por Pedro, y le dirá a Pedro qué tipo de flores enviar a María, y la dirección de recepción.

Ahora analicemos el mecanismo que utilizó Luis para resolver el problema:

1. Encontrar un agente apropiado (Pedro)
2. Enviarle un mensaje conteniendo la petición (enviar flores a María)
3. Es la responsabilidad de Pedro cumplir esa petición
4. Para ello, es posible que Pedro de algún modo disponga de algún método (algoritmo o conjunto de operaciones) para realizar la tarea
5. Luis no necesita (ni le interesa) conocer el método particular que Pedro utilizará para la petición (esa información es oculta)

Así, la solución del problema requiere de la cooperación de varios individuos para su solución. La definición de problemas en términos de responsabilidades incrementa el nivel de abstracción y permite una mayor independencia entre objetos.

Repasemos algunos conceptos importantes:

Agentes y comunidades: Un programa OO se estructura como una comunidad de agentes que interaccionan (OBJETOS). Cada objeto juega un rol en la solución del

problema. Cada objeto proporciona un servicio o realiza una acción que es posteriormente utilizada por otros miembros de la comunidad.

Mensaje: A un objeto se le envían mensajes para que realice una determinada acción. El objeto selecciona un método apropiado para realizar dicha acción y a este proceso se le denomina Paso de mensajes.

Responsabilidades: El comportamiento de cada objeto se describe en términos de responsabilidades (hay mayor independencia entre los objetos). Otro término importante es el de Protocolo, el cual refiere al conjunto de responsabilidades de un objeto.

Objetos y clases: Un objeto es una encapsulación de un estado (valores de los datos) y comportamiento (operaciones). Otra manera de pensar en un objeto es como la instancia de una clase, además, los objetos se agrupan en categorías (CLASES). El método invocado por un objeto en respuesta a un mensaje viene determinado por la clase del objeto receptor.

En definitiva, el objeto de POO es producir aplicaciones + fáciles de cambiar (MANTENIBILIDAD)

Abstracción

El Enfoque Orientado a Objetos se basa en cuatro principios que constituyen la base de todo desarrollo orientado a objetos. Estos son:

- Abstracción
- Encapsulamiento
- Modularidad

- Herencia

Y otros elementos a destacar (aunque no fundamentales):

- Polimorfismo
- Tipificación
- Concurrencia
- Persistencia

Ya hemos mencionado algunas características de estos principios, pero ahora desarrollémoslo con más detalle.

Abstracción

¿Qué es la abstracción? Una abstracción denota las características esenciales de un objeto (datos y operaciones), que lo distingue de otras clases de objetos. Decidir el conjunto correcto de abstracciones de un determinado dominio, es el problema central del diseño orientado a objetos.

Otra forma de conceptualizar la Abstracción es pensarla como la supresión intencionada (u ocultación) de algunos detalles de un proceso o artefacto, con el fin de destacar más claramente otros aspectos, detalles o estructuras. En cada nivel de detalle cierta información se muestra y cierta información se omite. Por ejemplo: Diferentes escalas en mapas.

Mediante la abstracción creamos MODELOS de la realidad.

Abstracción es el proceso de representar entidades reales como elementos internos a un programa, La abstracción de los datos es tanto en los atributos, como en los

métodos de los objetos. Por medio de la abstracción se puede tener una visión global del tema, por ejemplo en la clase PILA (Estructura de Datos, Ultimas Entradas Primeras Salidas LIFO), se pueden definir objetos de este tipo, tomando únicamente las propiedades LIFO (del inglés Last Input First Output) de las PILAS, algunos atributos como lleno, vacío, el tamaño y las operaciones o métodos que se pueden realizar sobre estos objetos como PUSH (meter un elemento) o POP (retirar un elemento) y no es necesario que el programador conozca o sea un experto en la Estructura de Datos con la que se implementa la PILA (como por ejemplo con una organización contigua o simplemente ligada, ni de los algoritmos que realizan el PUSH y el POP).

La abstracción se centra en las características esenciales de un objeto en relación a la perspectiva del observador. Los mecanismos de abstracción son usados en el EOO para extraer y definir del medio a modelar, sus características y su comportamiento. Entre ellos podemos nombrar:

La **GENERALIZACIÓN**: es el mecanismo de abstracción mediante el cual un conjunto de clases de objetos es agrupado en una clase de nivel superior (Superclase), donde las semejanzas de las clases constituyentes (Subclases) son enfatizadas, y las diferencias entre ellas son ignoradas. En consecuencia, a través de la generalización la superclase almacena datos generales de las subclases. Las subclases almacenan sólo datos particulares.

La **ESPECIALIZACIÓN**: es lo contrario de la generalización, por ejemplo, la clase Médico es una especialización de la clase Persona, y a su vez, la clase Pediatra es una especialización de la superclase Médico.

La **AGREGACIÓN**: es el mecanismo de abstracción por el cual una clase de objeto es definida a partir de sus partes (otras clases de objetos). Mediante agregación se puede

definir por ejemplo un computador, por descomponerse en: la CPU, la memoria y los dispositivos periféricos. El contrario de agregación es la DESCOMPOSICIÓN.

La CLASIFICACIÓN: consiste en la definición de una clase a partir de un conjunto de objetos que tienen un comportamiento similar. La EJEMPLIFICACIÓN es lo contrario a la clasificación, y corresponde a la instanciación de una clase, usando el ejemplo de un objeto en particular. La clasificación es el medio por el que ordenamos, el conocimiento ubicado en las abstracciones.

OCULTACIÓN DE INFORMACIÓN: Omisión intencionada de detalles de implementación tras una interfaz simple.

Las clases y objetos deberían estar al nivel de abstracción adecuado: ni demasiado alto ni demasiado bajo

Encapsulamiento

¿Qué es el encapsulamiento? El encapsulamiento en un sistema orientado a objeto se representa en cada clase u objeto, definiendo sus atributos y métodos con diferentes tipos de visibilidad (mediante los modificadores de acceso). Como mencionamos anteriormente, si usamos UML, podemos cambiar el tipo de visibilidad, o accesibilidad, usando:

- Público (+) Atributos o Métodos que son accesibles fuera de la clase.

Pueden ser llamados por cualquier clase, aun si no está relacionada con ella.

- Privado (-) Atributos o Métodos que solo son accesibles dentro de la implementación de la clase.

- Protegido (#): Atributos o Métodos que son accesibles para la propia clase y sus clases hijas (subclases).

El encapsulamiento oculta los detalles de implementación de un objeto. Cada objeto está aislado del exterior, esta característica permite verlo como una caja negra, que contiene toda la información relacionada con ese objeto. Este aislamiento protege a los datos asociados a un objeto para que no se puedan modificar por quien no tenga derecho a acceder a ellos. Permite manejar a los objetos como unidades básicas, dejando oculta su estructura interna.

Retomemos el ejemplo anterior de una PILA: La PILA oculta una representación interna de datos que lleva el registro de los elementos que esperan actualmente en la línea, y ofrece operaciones a sus clientes (PUSH, agregar un elemento o POP, retirar un elemento). Al desarrollador no le preocupa la implementación de la PILA, simplemente dependen de que esta opere “como se indicó”. Cuando un cliente quiere agregar un elemento (PUSH), la PILA debe aceptarlo y colocarlo en algún tipo de estructura de datos interna. De manera similar, cuando el cliente desea retirar un elemento (POP), la pila debe retirar el elemento que haya estado más tiempo en la PILA.

El encapsulamiento da lugar al Principio de Ocultamiento. Esto implica que sólo los métodos de una clase deberían tener acceso directo a los atributos de esa clase, para impedir que un atributo sea modificado en forma insegura, o no controlada por la propia clase.

Diferencia entre Abstracción y encapsulamiento

1. Abstracción: Busca la solución en el diseño. Únicamente información relevante, Centrado en la ejecución.



UNJu
Universidad
Nacional de Jujuy



**Argentina
programa
4.0**

2. Encapsulamiento: Busca la solución en la implementación. Ocultación de código para protegerlo. Centrado en la ejecución.

Video explicación de Abstracción y encapsulamiento usando el IDE NetBeans:

<https://www.youtube.com/watch?v=oxGHV9qb9Xg>