# Programación Orientada a Objetos con Java

## Programando Objetos

La Programación Orientada a Objetos (POO) es un **paradigma de programación**, o sea, es una forma de entender y utilizar los lenguajes de programación y diseñar software. La POO surge como una respuesta a la necesidad de métodos y técnicas de programación que permitieran describir mejor la realidad, más ágiles y más potentes. El razonamiento detrás de la POO es que resulta mucho más natural programar si pensamos en los diferentes componentes de un programa como ***objetos que interactúan entre sí***.

De esta manera, obtenemos la definición de **Programa** en la POO, que es:

|  |  |
| --- | --- |
| vaca-habla-1.png | **DEFINICIÓN : Programa**  Conjunto de **OBJETOS** que **CUMPLEN DIVERSAS FUNCIONES** e **INTERACTÚAN ENTRE SÍ**, intercambiando **INFORMACIÓN.** |

**Esquema del funcionamiento de un programa diseñado mediante POO**

**Respuesta  
del  
Programa**

**Solicitud  
del  
Usuario**

**Programa**

De esta manera, para diseñar un programa lo que se hace es **diseñar y crear los objetos** que necesitamos para que cumplan las diversas tareas que componen el programa y relacionarlos entre sí.

¿Qué es un objeto?

En la POO, un objeto puede ser **cualquier cosa que yo le pueda describir a la computadora**. Puede basarse en un objeto de la realidad, o ser imaginario, o una abstracción.

|  |  |
| --- | --- |
| vaca-habla-1.png | **DEFINICIÓN : Objeto**  **Conjunto de recursos** (variables, algoritmos, funciones, etc.) que, juntos, **proporcionan una funcionalidad determinada dentro de un programa**. Un objeto puede representar una realidad física (los datos de una persona, por ejemplo) o un concepto completamente abstracto (como la conexión a una base de datos). |

Pero todos los objetos tienen una misma estructura: ***Clase,*** ***Nombre, Atributos y Métodos.***

La ***CLASE*** se puede definir como la **“familia” a la que pertenece un objeto**. Igual que una familia determina ciertas cualidades de una persona, la clase determina las características generales del objeto. De esta manera, todos **los objetos de una misma clase tienen las mismas características generales y las mismas capacidades**.

El ***NOMBRE*** es el **identificador de un objeto dentro del programa**. Un mismo programa puede, durante su ejecución, generar **varios objetos de la misma clase** (mediante un proceso llamado ***Creación de Instancias***, que veremos más adelante) pero para poder identificarlos y usar sus funciones, cada objeto debe tener **un nombre propio único** dentro del programa.

Los ***ATRIBUTOS*** son las **características del objeto**, lo que indica “cómo es” un objeto y en general, podemos decir que son ***variables*** tal y como las vimos antes. Algo importante a tener en cuenta es que, mientras que dos objetos de la misma clase tienen **el** **mismo conjunto de atributos**, **los valores** de esos atributos **pueden ser diferentes** (de la misma manera que dos hermanos pertenecen a la misma familia y comparten los mismos rasgos, pero de forma distinta).

Los ***MÉTODOS*** son las **capacidades del objeto**, indican “qué puede hacer” el objeto. Se trata de conjuntos de algoritmos o conjuntos de algoritmos que permiten que el objeto realice una tarea concreta. Es mediante la ejecución de estos métodos que el programa lleva a cabo sus funciones.

|  |  |
| --- | --- |
| vaca-habla-2.png | **CONCLUSIÓN:**   * **Todos los componentes** del programa **son objetos** que **se generan usando clases** . * Todo objeto del programa tiene **atributos y métodos** que se definen **en su clase**. * A los atributos **se les asigna valores** (semejante al uso de variables), los métodos **se ejecutan.** * El programa está formado por **objetos creados usando ese conjunto de clases**. |

***Ejemplo 1:***

Como se explicó más arriba, la POO se basa en el concepto de objeto que manejamos en el mundo real, y para este ejemplo usaremos **una vaca**. Primero **¿cómo está formada una vaca?**, podríamos entrar en muchos detalles, pero generalicemos lo más posible:

CUERNOS

COLA

MANCHAS

****

CABEZA

BOCA

OJOS

CUERPO

PATAS

UBRE

Lo que forma a nuestra vaca, sus características, serán sus **atributos**. Y podemos agruparlos de esta manera:

|  |
| --- |
| VACA |
| * Cabeza |
| * Cuernos |
| * Ojos |
| * Boca |
| * Cuerpo |
| * Manchas |
| * Ubre |
| * Cola |
| * Patas |

Ahora bien, una vaca también es capaz de **hacer** **cosas**. Por ejemplo:

|  |  |  |
| --- | --- | --- |
| vaca-muge.png | vaca-come.png | vaca-corre.png |
| **Mugir** | **Comer** | **Correr** |

Las cosas que nuestra vaca **puede hacer** son sus habilidades o capacidades, y las llamaremos **métodos.** Al igual que hicimos con los atributos, podemos agrupar los métodos en una tabla, de esta manera:

|  |
| --- |
| VACA |
| * Mugir() |
| * Comer() |
| * Correr() |

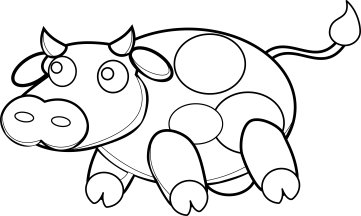
|  |  |
| --- | --- |
| vaca-habla-2.png | **IMPORTANTE:** Te habrás fijado que los métodos se escribieron con un paréntesis al final. Esto es algo importante a tener en cuenta, ya que los métodos **siempre** se deben anotar de esa manera, ya que **es la forma en que los diferenciamos** de los atributos.  Además, el paréntesis **tiene un propósito práctico** que explicaremos en detalle más adelante en este libro cuando profundicemos en el desarrollo de métodos. |

¿Cómo se crea un objeto?

Para crear un objeto **primero debe existir su clase**. No podremos crear un objeto si su clase no ha sido definida aún.

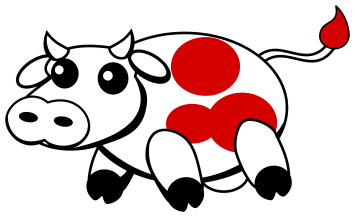
Las clases son los componentes básicos de todo proyecto de programación orientada a objetos. **Todo lo que existe y sucede dentro del programa, se define dentro de las clases, y se lleva a cabo mediante objetos que se generan a partir de esas clases.**

Podemos pensar en una clase como **en un molde**, con el cual **creamos nuevos objetos** que tendrán la “misma forma”, o sea las mismas características generales, pero que serán **objetos diferentes** durante la ejecución del programa.

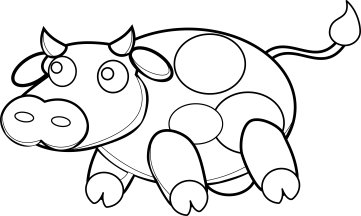
**Objeto 1**

**CLASE**



**Objeto 2**

Este proceso de creación de objetos se llama ***Creación de Instancias***o ***Instanciación*** , y cada objeto nuevo es una ***instancia*** de la clase a la que pertenece.

**Objeto  
(Instancia de Clase)**

**Genera** objetos que reciben sus características generales, pero **no es utilizable por sí misma**.

**Pertenece** a una **clase** y recibe sus características, pero puede aplicarlas de forma particular

**Clase**

**Creación de Instancias**

|  |  |
| --- | --- |
| vaca-habla-1.png | **DEFINICIÓN : Instancia**  Es la creación de un **objeto** a partir de las características definidas en una **clase.** Los objetos sólo pueden crearse mediante instancias, y las instancias solo son posibles mediante clases.  Es mediante la creación de un objeto que la clase puede usarse, ya que en muy pocos casos es posible (o recomendable) el utilizar los métodos y/o atributos de una clase de forma directa.  En general: Una **clase es inútil sin objetos que la instancien**, y **es imposible crear un objeto mediante una instancia si este no pertenece a una clase.** |

¿Cómo se crea una clase?La creación de una clase se llama ***declaración***. Al declarar una clase, se establece la descripción de los objetos que se crearán a partir de esa clase tal como lo hemos definido más arriba: mediante sus atributos y sus métodos.

Para la declaración de una clase, es importante que primero tengamos claro los atributos y los métodos que necesitaremos definir. Los diagramas que utilizamos en el ejemplo de más arriba nos servirán para eso, primero consideremos los datos que tenemos hasta el momento y vamos a plantearlos en un solo diagrama

|  |  |
| --- | --- |
| VACA | |
| Atributos | **Métodos** |
| * Cabeza | * **Mugir()** |
| * Cuernos | * **Comer()** |
| * Ojos | * **Correr()** |
| * Boca |  |
| * Cuerpo |  |
| * Manchas |  |
| * Ubre |  |
| * Cola |  |
| * Patas |  |

Para el desarrollo de clases en POO se utilizan diagramas muy similares a este mediante un lenguaje gráfico llamado **UML.** En este capítulo veremos **el primer paso** del diseño de clases en este lenguaje**: la creación de entidades**, pero para llegar a ese punto primero necesitamos **formalizar un poco** nuestro diagrama de atributos y métodos.   
En primer lugar formalicemos un poco nuestros atributos y nuestros métodos.

En este punto nos tenemos que preguntar acerca de la **visibilidad** de los atributos y métodos de la vaca.

|  |  |
| --- | --- |
| vaca-habla-1.png | **DEFINICIÓN : Visibilidad**  Es una característica de los atributos y los métodos de un objeto que indica si el atributo o método puede ser modificado o utilizado por otro objeto. |

En principio, consideraremos dos valores de visibilidad: **público** y **privado**. La accesibilidad significa cosas diferentes para un atributo y para un método:

|  |  |
| --- | --- |
| vaca-habla-1.png | **Visibilidad**  **En un Atributo:**   * **Atributo Público:** *Cualquier objeto* puede acceder y modificar los valores del atributo. * **Atributo Privado:** *Solo el objeto propietario del atributo* puede acceder y modificar los valores del mismo.   **En un Método:**   * **Método Público:** El método puede ser ejecutado por *cualquier objeto* * **Método Privado:** El método solo puede ser ejecutado *por el objeto propietario del mismo* |

Para definir cuál característica del objeto será pública y cuál será privada, tenemos que considerar **la forma en que queremos que nuestro objeto interactúe** con los demás.

En general, para **los atributos**:

|  |  |
| --- | --- |
| vaca-habla-1.png | Un **atributo se definirá como público si:**   * Contiene o va a contener datos que deben ser **modificados por otros objetos**. * Sus datos pueden o deben **modificarse durante la ejecución** del programa.   De lo contrario, **el atributo será definido como privado**. |

Y para **los métodos**:

|  |  |
| --- | --- |
| vaca-habla-1.png | Un **método se definirá como público si:**   * El objeto lo va a necesitar para intercambiar información con otros objetos. * El resultado de su ejecución es necesaria para el funcionamiento de otros objetos.   De lo contrario, **el método será definido como privado**. |

Ahora solo debemos pensar cómo serán los atributos y los métodos de nuestra vaca. Como los atributos son sus características físicas, y estas no son fácilmente modificables, diremos que son atributos privados. Si fueran públicos, es como decir que podemos cambiar a voluntad las manchas de nuestra vaca, o cuántos ojos tiene.

En cuanto a los métodos, el razonamiento es parecido. El método **comer()** será privado (solo la vaca puede decidir cuándo comer). Pero los métodos **mugir()** y **correr()** pueden depender de estímulo externo (o sea, algo o alguien puede hacer que la vaca se ponga a correr o a mugir), así que los haremos públicos.

Para indicarlo en nuestro diagrama usaremos los símbolos "**+**" y "**-**" delante de cada característica:

|  |  |
| --- | --- |
| vaca-habla-1.png | **+** Característica 🡪 Indica que la característica es **pública**  **-** Característica 🡪 Indica que la característica es **privada** |

De esta manera nuestro diagrama, por ahora, se verá de esta manera:

|  |  |
| --- | --- |
| VACA | |
| Atributos | **Métodos** |
| - Cabeza | **+ Mugir()** |
| - Cuernos | **- Comer()** |
| - Ojos | **+ Correr()** |
| - Boca |  |
| - Cuerpo |  |
| - Manchas |  |
| - Ubre |  |
| - Cola |  |
| - Patas |  |

Un último elemento de UML que veremos ahora es la forma correcta en que una clase se describe en este lenguaje. En UML, una clase se describe indicando primero el nombre, luego los atributos y finalmente los métodos, con lo que el diagrama queda de esta manera:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| vaca-habla-1.png | **Representación de una Clase en UML:**   |  | | --- | | **NombreDeLaClase** | | **Atributos** | | **Métodos()** | |

Aplicándolo en la definición de nuestra vaca:

|  |
| --- |
| **VACA** |
| **- Cabeza**  **- Cuernos**  **- Ojos**  **- Boca**  **- Cuerpo**  **- Manchas**  **- Ubre**  **- Cola**  **- Patas** |
| **+ Mugir()**  **- Comer()**  **+ Correr()** |

Así que definimos nuestra primera clase y estamos en condiciones de empezar a crear el código de programación para poder utilizar esta clase en nuestro programa, pero para ello necesitamos conocer el lenguaje que usaremos: **Java**.  
En el próximo capítulo comenzaremos con los conceptos básicos de programación en Java y empezaremos a ver cómo transformar las clases que declaremos en código en un programa.

|  |  |
| --- | --- |
| vaca-habla-2.png | **CONCLUSIÓN:**   * Un objeto necesita una clase para poder crearse. En esa clase se definen los **atributos** y los **métodos** que tendrá el objeto cuando sea creado. * La creación o definición de una clase se llama **declaración**, y la creación de un objeto a partir de una clase se llama **creación de instancias** o **instanciación**. * Todo objeto es **instancia** de alguna clase. * Los atributos y métodos de una clase tienen **visibilidad,** y pueden ser **públicos** o **privados.** |

## 2 – Introducción a la Programación con Java

En el capitulo anterior vimos algunos conceptos básicos sobre la POO. Antes de seguir profundizando en esos conceptos, veamos un poco sobre el lenguaje de programación con el que implementaremos nuestra POO en este curso: **Java.**

### ¿Qué es Java?

|  |  |
| --- | --- |
| Java logo.png | Java es un lenguaje de programación desarrollado en 1991 por James Gosling, Mike Sheridan y Patrick Naughton. Pasó por varios nombres, hasta que sus creadores lo nombraron en honor a una variedad de café (de ahí el logo de la taza). La primera versión pública fue liberada en 1995 por Sun Microsystems. Esta empresa llevó a cabo el desarrollo del lenguaje hasta 2010, cuando fue adquirida por Oracle.  Una de las principales características de Java es su **portabilidad:** un programa desarrollado en Java **no se ejecuta directamente** en el sistema operativo, sino que **es ejecutado por una máquina virtual** |
| (llamada “Máquina Virtual de Java”, **JVM,** por su sigla en inglés). De esta manera, un programa escrito en Java **puede ejecutarse en cualquier sistema operativo**, siempre y cuando la JVM esté instalada previamente en el mismo. Esta característica hace que sea un lenguaje muy usado en el mundo, manteniéndose en los primeros lugares del índice TIOBE[[1]](#footnote-1), y siendo el lenguaje de programación de aplicaciones para el sistema operativo **Android** de Google. | |

### ¿Cómo se Programa en Java?

Bueno, de eso se trata este libro, pero para empezar, concentrémonos en la sintaxis de algunas **estructuras básicas.**

|  |  |
| --- | --- |
| vaca-habla-1.png | **Estructuras Básicas:**  Cuando en este libro hablamos de estructuras básicas, nos referimos a las **estructuras de datos y de control que resultan imprescindibles para programar.**  Estos son:   * Las **variables**, y las sentencias que permitan acceder a ellas y modificarlas * La **expresión condicional** (el **SI** ó **IF**) * El **bucle** **FOR** * El **bucle** **WHILE** |

Pero sin importar el objeto o estructura que estemos definiendo, **toda sentencia debe finalizar en punto y coma (;).**La ausencia del punto y coma al finalizar una sentencia es uno de los errores más comunes (y más frustrantes) que encuentra el programador.

|  |  |
| --- | --- |
| vaca-habla-2.png | **Finalizado de sentencias:**  Toda estructura debe finalizarse con un punto y coma (;) ya que esto es lo que le indica al compilador de Java que la orden terminó y que puede evaluar la próxima línea del programa, de hecho, puedo incluso incluir saltos de renglón en medio de una sentencia sin que esto afecte el resultado final, siempre y cuando use un punto y coma al final.  De esta manera la sintaxis de una sentencia cualquiera en Java sería:  **comando modificador1 modificador2;**  O también:  **comando**  **modificador1**  **modificador2;** |
|  |  |

#### Definir Variables en Java

Para definir variables, debemos tener en cuenta que una de las principales características de Java es que es un lenguaje programación **fuertemente tipado.** Esto significa ***que todos los objetos*** que definamos **deben definirse especificando su tipo de datos,** o de lo contrario el programa no podrá ejecutarse.

De esta manera, cuando debo definir una variable en Java sigo estos pasos:

1. **Defino su tipo de datos.**
2. **Defino su nombre.**
3. **Defino su valor inicial.**

Esto nos deja la siguiente sintaxis general:

|  |  |
| --- | --- |
| vaca-habla-1.png | **Sintaxis para definir variables:**  **tipoDeDatos NombreDeVariable = valor;** |

Conviene acotar que esta forma de definir una variable en Java es lo que llamaremos **declaración directa** o **declaración con inicialización.** Más adelante veremos otras formas de declarar que pueden resultar más convenientes en algunos casos

Java es un lenguaje con muchos tipos de datos, pero en general usaremos unos pocos tipos básicos que cubrirán la mayoría de nuestras necesidades iniciales:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| vaca-habla-1.png | **Tipos de Datos básicos en Java y Ejemplos de declaración:**  Java es un lenguaje **muy estricto** con las minúsculas y las mayúsculas, y deben respetarse para evitar errores. O sea “string”, “String” y “STRING” son tres expresiones totalmente diferentes.   |  |  | | --- | --- | | **Tipo de Dato** | **Descripción** | | **int** | Para valores numéricos positivos o negativos enteros (sin decimales) entre -2.147.483.648 y 2.147.483.647 | | **int cantidad = 0;** | | | **long** | Igual que **int** pero para valores mayores que los soportados por **int** | | **long cantidad = 0;** | | | **float** | Para valores numéricos positivos o negativos con decimales | | **float resultado = 0.0;** | | | **double** | Igual que **float,** pero para valores decimales que superan los soportados por **float** | | **char** | Permite almacenar **un solo** carácter: una letra, número, espacio o símbolo | | **char letra = "a";** | | | **String** | Para cadenas de texto de uso general. | | **String texto = "texto cualquiera";** | | | **boolean** | Permite almacenar solo dos posibles valores: **true** y **false** | | **boolean pruebaL = true;** | | |

Ahora que tenemos una sintaxis general para definir variables, y los tipos de datos necesarios para definirlas, podemos plantear cómo se establecer, acceder a, y modificar los datos almacenados en una variable en Java.

Para todos los ejemplos que siguen a continuación, usaremos una variable de tipo *String* llamada “texto” y un par de variables de tipo *int* llamadas “num1” y “num2”. O sea, se supone que en alguna parte del código, antes de los ejemplos, existen las siguientes sentencias o unas muy similares:

**int num1 = 0;**

**int num2 = 0;**

**String texto = "";**

Por supuesto, esta aclaración se debe a que **es imposible establecer el valor de una variable que no existe.**

|  |  |
| --- | --- |
| vaca-habla-1.png | **DEFINICIÓN: 1 – Estableciendo el valor de una variable ya existente**  **NombreDeVariable = valor;  texto = "nuevo valor de la variable";**  Cuando la variable ya existe**, no es necesario** indicar su tipo, simplemente se utiliza el nombre que le dimos, y mediante el “ **=** ” le asignamos el nuevo valor que queremos que tenga. |

O sea que basta con **igualar la variable a un valor compatible** para que la misma almacene dicho valor.

Para el siguiente ejemplo utilizaremos una **función,** que detallaremos con mayor precisión más adelante. Esta función es **println()**, y permite mostrar en pantalla una línea de texto (que debe insertarse dentro de los paréntesis) y resulta útil para acceder al contenido de nuestra variable.

|  |  |
| --- | --- |
| vaca-habla-1.png | **DEFINICIÓN: 2.1 – Accediendo al valor de una variable ya existente (mediante una función)**  **funcion(NombreDeVariable);  System.out.println(texto);** |

Como podemos observar, simplemente incluimos el nombre de la variable en el sitio donde queremos que aparezcan los datos almacenados en ella, y Java los reemplazará de forma automática.   
Ahora bien, no siempre voy a utilizar mis variables dentro de una función, en otros casos, puedo utilizar los datos de una o más variables para establecer el valor de otra. Para explicarlo planteemos el siguiente caso: estableceremos el valor de la variable “num1” que definimos antes, usaremos a “num1” para establecer el valor de la variable “num2”, y crearemos una tercer variable de tipo *int*, llamada “total”, donde almacenaremos el valor de la suma de “num1” y “num2”. El código resultante es el siguiente

1. TIOBE, una compañía dedicada a la evaluación de la calidad del software, mantiene un índice donde mide la popularidad de los diferentes lenguajes de programación basándose en la cantidad de búsquedas realizadas por los usuarios en Internet. Puede consultarse en <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> [↑](#footnote-ref-1)