



## Programación Orientada a Objetos (POO)

### Introducción a la POO

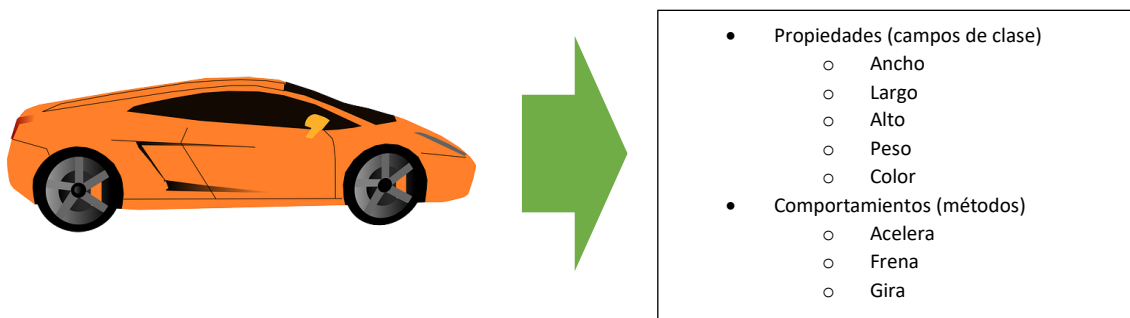
La programación orientada a objetos (en adelante la mencionaremos como POO) es el paradigma de programación utilizado en la actualidad, al haber sustituido a la programación estructurada.

Antes de la aparición de la POO, el código de los programas era extenso, confuso, muy difícil de interpretar sobre todo a personas que no crearon esos programas pero que debían corregir o añadir alguna funcionalidad en el código, con numerosos saltos en el código con instrucciones “goto” y “gosub” (código espagueti). Y esta complejidad crecía exponencialmente a medida que las aplicaciones necesitaban ser más avanzadas y complejas.

Ante este escenario, la comunidad de programadores se vio en la necesidad de replantearse los paradigmas de la programación existentes hasta el momento e inventar algo que le hiciera más fácil la vida al programador, algo que hiciera más manejables y comprensibles los códigos de las aplicaciones sobre todo a la hora de trabajar en equipo. Fue entonces cuando surgió la **POO**.

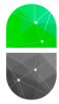
Se decidió que la mejor manera de hacer comprensible el código de programación era asemejarlo a la vida real; hacer que el código de programación tuviese unas características y comportamiento similar a los objetos que tenemos en la vida real. De esta forma, la curva de aprendizaje y el manejo del código sería mucho más sencillo para cualquier persona, especialmente para los programadores que también son personas (sic).

Los objetos de la vida real tienen **propiedades**: tamaño, ancho, alto, peso, temperatura, color etc; y tienen **comportamientos**, esto es, qué son capaces de hacer estos objetos. Se decidió trasladar todo esto al código de programación bautizándolo como **POO**.

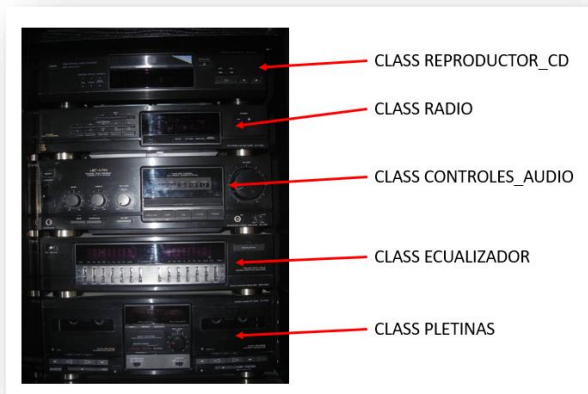


Un programa Java está formado por objetos, y los objetos tienen propiedades y son capaces de hacer determinadas operaciones. No importará tanto como está construido el objeto como su funcionalidad.

La clave para ser productivo a la hora de programar consiste en conseguir que cada objeto sea responsable de ejecutar un conjunto de tareas relacionadas. Si un objeto se basa en una tarea que no es su responsabilidad, necesitará tener acceso a otro objeto entre cuyas responsabilidades se encuentra esa tarea. A esto se le conoce con el nombre de “**modularización**”: dividir el código en unidades de código independientes que funcionan a modo de puzle formando una estructura mayor funcional.



Un objeto no debe manipular nunca directamente los datos internos de otros objetos, ni tampoco debería exponer datos para que otros objetos accedieran a ellos directamente. Toda la comunicación debe hacerse mediante la llamada de métodos. A esto se le conoce con el nombre de “**encapsulación**” (ocultar datos al exterior para impedir su manipulación). Al encapsular los datos del objeto, se maximiza la “**reutilización**” del código de programación lo que se convierte en una de las mayores ventajas de la POO.

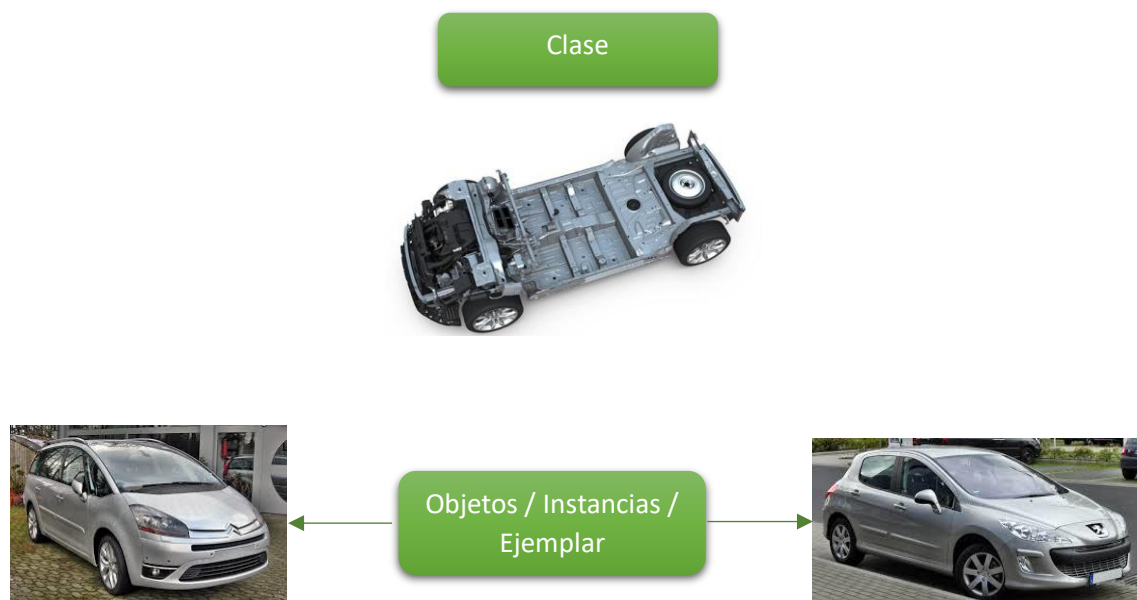


**Modularización:** características de algunos objetos de la vida real los cuales están formados por unidades independientes que funcionan de forma conjunta formando una unidad. Al mismo tiempo cada una de esas partes son independientes y no pueden acceder al interior de las demás partes (**encapsulación**). Se entiende este concepto si pensamos en un equipo HIFI modular.

### Términos de la POO

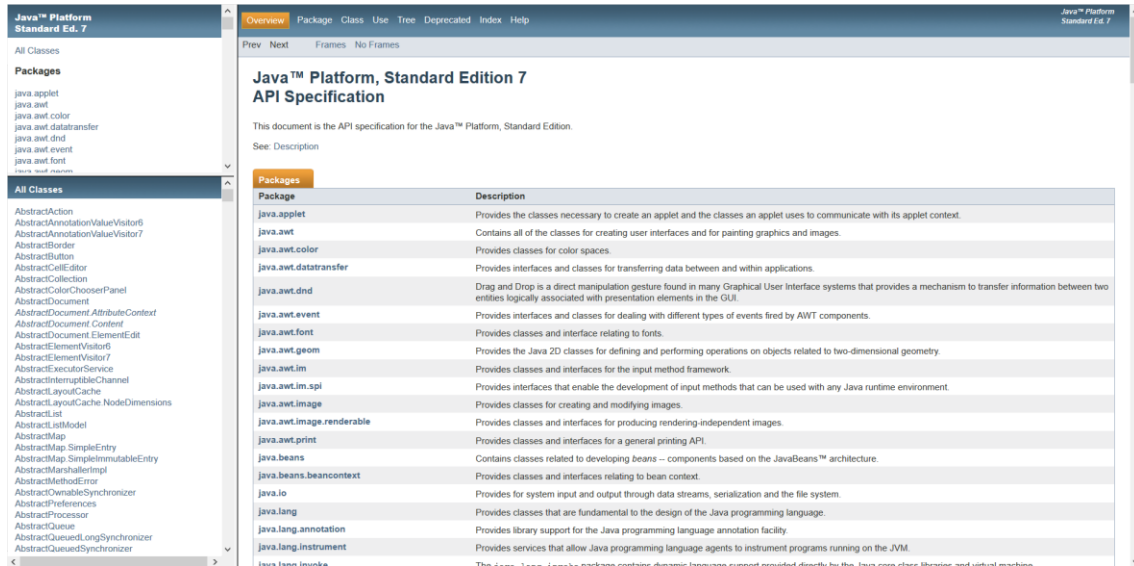
El término más importante es **clase**. Una clase es una plantilla a partir de la cual se crean realmente los **objetos**.

Cuando se construye un objeto a partir de una clase, se dice que se ha creado un **ejemplar o instancia de clase**.





Todo el código que se escribe en Java está dentro de una clase. La biblioteca estándar de Java (API) contiene varios miles de clases cada una con un propósito diferente, sin embargo, sigue siendo necesario crear nuestras propias clases para describir los objetos del problema de las aplicaciones que podamos crear y también adaptar las clases de la biblioteca estándar a nuestras necesidades.



Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing beans – components based on the JavaBeans™ architecture.
java.beans.beancontext	Provides classes and interfaces relating to bean context.
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.annotation	Provides library support for the Java programming language annotation facility.
java.lang.instrument	Provides services that allow Java programming language agents to instrument programs running on the JVM.

## API de Java

Un concepto clave a la hora de trabajar con objetos es la **encapsulación**. La **encapsulación** se podría resumir como la combinación de los datos y comportamiento del objeto en un paquete para que permanezca invisible al que maneja el objeto desde el exterior.

Un objeto contiene datos, denominados en Java **campos de ejemplar** y contiene también **métodos** que son procedimientos que operan sobre los datos.

Un objeto en concreto tendrá unos datos o campos de ejemplar concretos. A la suma de esos datos o campos se le denomina **estado del objeto o propiedades**.

Para cambiar el estado de un objeto utilizamos los **métodos** (lo que se conoce por funciones en otros lenguajes de programación). A través de estos **métodos** podemos cambiar el estado inicial de cualquier objeto. Los métodos representan el comportamiento del objeto, esto es, qué es capaz de hacer el objeto.

Es posible crear clases por **extensión**, esto es crear una **clase extendida** de otra ya existente. Es lo que en POO se denomina **herencia**.

Cuando se crea una clase que hereda de otra, esta nueva clase posee todas las propiedades y métodos de la clase extendida.