APRENDERAPROGRAMAR.COM

CONCEPTO O DEFINICIÓN DE HERENCIA EN JAVA Y EN PROGRAMACIÓN ORIENTADA A OBJETOS. ¿QUÉ ES? EXTENDS. EJEMPLOS. (CU00684B)

Sección: Cursos
Categoría: Curso “Aprender programación Java desde cero”

Fecha revisión: 2029

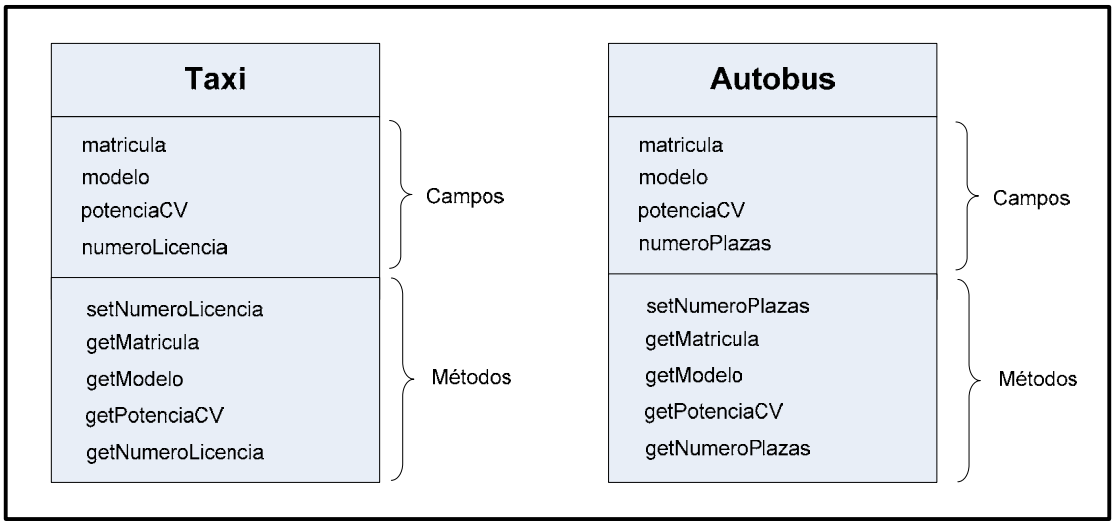
Resumen: Entrega nº84 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

¿QUÉ ES LA HERENCIA EN PROGRAMACIÓN ORIENTADA A OBJETOS?

Muchas veces distintos objetos comparten campos y métodos que hacen aproximadamente lo mismo (por ejemplo almacenar y devolver un nombre del ámbito humano con el que se designa al objeto, como el título de un álbum de música, el título de un libro, el título de una película, etc.).

Por ejemplo en un proyecto que utilice objetos Taxi y objetos Autobus podríamos encontrarnos algo así:



Para una aplicación de gestión de una empresa de transporte que tenga entre sus vehículos taxis y autobuses podríamos tener otra clase denominada FlotaCirculante donde tendríamos posibilidad de almacenar ambos tipos de objeto (por ejemplo taxis en un ArrayList y autobuses en otro ArrayList) como reflejo de los vehículos que se encuentran en circulación en una fecha dada. Esas listas conllevarían una gestión para añadir o eliminar vehículos de la flota circulante, modificar datos, etc. resultando que cada una de las listas necesitaría un tratamiento o mantenimiento.

Si nos fijamos en el planteamiento del problema, encontramos lo siguiente:

a) La definición de clases nos permite identificar **campos y métodos que son comunes** a Taxis y Autobuses. Si implementamos ambas clases tal y como lo venimos haciendo, incurriremos en duplicidad de código. Por ejemplo si el campo *matricula* es en ambas clases un tipo String, el código para gestionar este campo será idéntico en ambas clases.

b) La definición de clases nos permite identificar **campos y métodos que difieren** entre una clase y otra. Por ejemplo en la clase Taxi se gestiona información sobre un campo denominado *numeroDeLicencia* que no existe en la clase Autobus.

c) Conceptualmente **podemos imaginar una abstracción** que engloba a Taxis y Autobuses: ambos podríamos englobarlos bajo la denominación de "Vehículos". Un Taxi sería un tipo de Vehiculo y un Autobus otro tipo de Vehiculo.

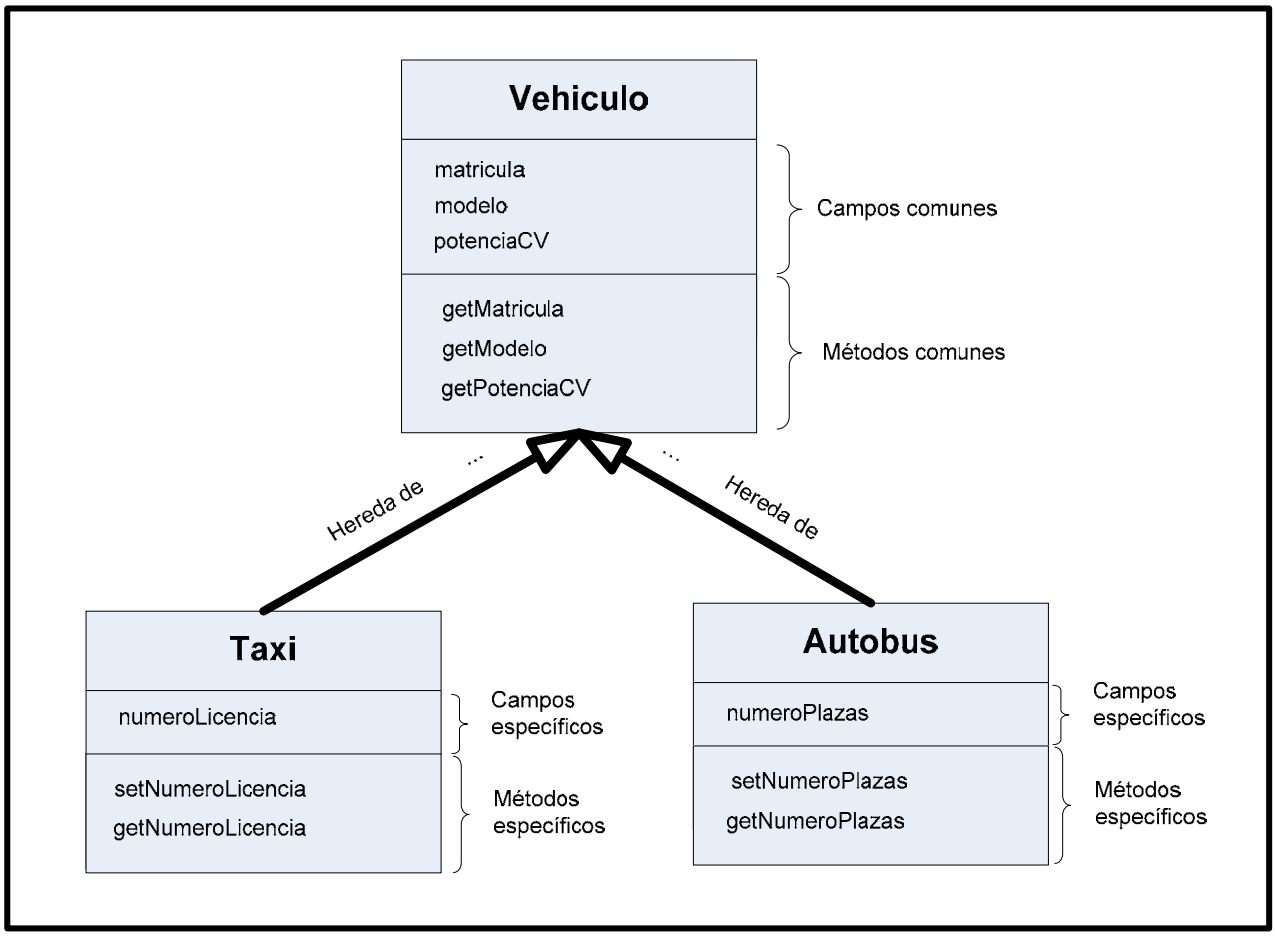
d) Si la empresa añade otros vehículos como minibuses, tranvías, etc. manteniendo la definición de clases tal y como la veníamos viendo, seguiríamos engrosando la **duplicidad de código**. Por ejemplo, un minibús también tendría matrícula, potencia… y los métodos asociados.

La duplicidad de código nos implicará problemas de mantenimiento. Por ejemplo inicialmente tenemos una potencia en caballos y posteriormente queremos definirla en kilowatios. O tenemos simplemente que modificar el código de un método que aparece en distintas clases. El tener el código duplicado nos obliga a tener que hacer dos o más modificaciones en sitios distintos. Pueden ser dos modificaciones, tres, cuatro o n modificaciones dependiendo del número de clases que se vieran afectadas, y esto a la larga genera errores al no ser el mantenimiento razonable.

En la clase FlotaCirculante también tendremos seguramente duplicidades: por un lado un ArrayList de taxis y por otro un ArrayList de autobuses, por un lado una operación de adición de taxis y otra operación de adición de autobuses, por un lado una operación para mostrar los elementos de la lista de taxis y otra para los elementos de la lista de autobuses…

¿No sería más razonable, si una propiedad o método va a ser siempre común para varios tipos de objetos, que estuviera localizada en un sitio único del que ambos tipos de objeto "bebieran"? En los lenguajes con orientación a objetos la solución a esta problemática se llama herencia. La herencia es precisamente uno de los puntos clave de este tipo de lenguajes.

La herencia nos permite definir una clase como extensión de otra: de esta manera decimos "la clase 1.1 tiene todas las características de la clase 1 y además sus características particulares". Todo lo que es común a ambas clases queda comprendido en la clase "superior", mientras lo que es específico, queda restringido a las clases "inferiores". En nuestro ejemplo definiríamos una clase denominada Vehiculo, de forma que la clase Taxi tuviera todas las propiedades de la clase Vehiculo, más algunas propiedades y métodos específicos. Lo mismo ocurriría con la clase Autobus y otras que pudieran "heredar" de Vehiculo. Podríamos seguir creando clases con herencia en un número indefinido: tantas como queramos. Si piensas en el API de Java, hay cientos de clases que heredan de clases jerárquicamente superiores como la clase Object. En un proyecto propio, podremos tener varias clases que hereden de una clase común.



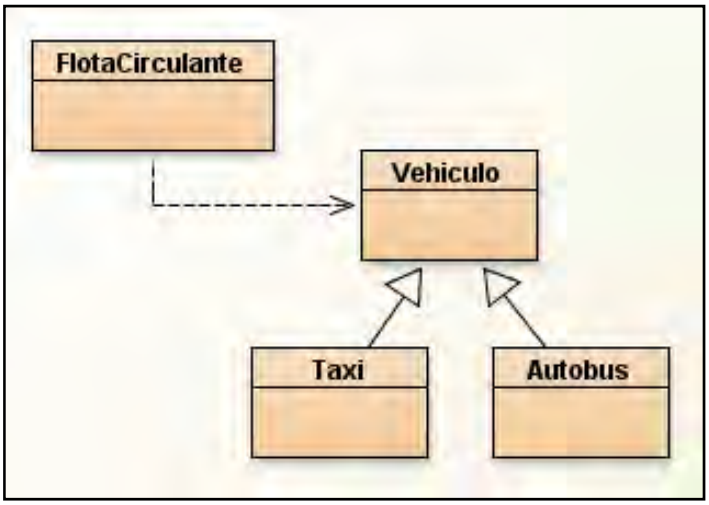
Esquema básico de herencia

Otro ejemplo: para la gestión de un centro educativo, podemos definir la clase Persona que comprende los campos y métodos comunes a todas las personas. Luego podremos definir la clase Estudiante, que es extensión de Persona, y comprende los campos y métodos de la clase superior, más algunos específicos. También podrían heredar de Persona la clase Profesor, la clase Director, la clase JefeDeEstudios, la clase PersonalAdministrativo, etc.

La primera aproximación a la herencia en Java la plantearemos para plasmar todo lo que hemos discutido en párrafos anteriores: en vez de definir cada clase por separado con operaciones o campos duplicados en cierta medida, definiremos una clase “padre” que contendrá todas las cosas que tengan en común otras clases. Luego definiremos las otras clases “hijo” como extensión de la clase padre, especificando para ellas únicamente aquello que tienen específico y distinto de la clase padre. Una característica esencial de **la herencia es que permite evitar la duplicidad de código:** aquello que es común a varias clases se escribe solo una vez (en la clase padre). La duplicidad de información, ya sea código o datos, es algo que por todos los medios hay que tratar de evitar en los sistemas informáticos por ser fuente de errores y de problemas de mantenimiento.

Si nos remitimos al esquema básico de herencia donde hemos representado las clases Vehiculo, Taxi y Autobus, la clase Vehiculo diremos que actúa como clase padre de las clases Taxi y Autobus. Vehiculo recogería todo aquello que tienen o hacen en común taxis y autobuses. Aunque aquello que tienen en común se agrupa, tanto Taxi como Autobus tienen sus campos o métodos específicos.

Que una clase derive de otra en Java se indica mediante la palabra clave **"extends"**. Por eso muchas veces se usa la expresión "esta clase es extensión de aquella otra". En los diagramas de clase la herencia se representa con una flecha de punta vacía. El aspecto en BlueJ sería algo así:



Este diagrama refleja que la clase FlotaCirculante usa a la clase Vehiculo, mientras que las clases Taxi y Autobus heredan de la clase Vehiculo (son extensión de la clase Vehiculo). Podemos decir que la clase hijo extiende (hace más extensa) a la clase padre. Una clase de la que derivan otras se denomina clase padre, clase base o superclase. Las clases que heredan se denominan clases derivadas, clases hijos o subclases. A partir de ahora los términos **superclase y subclase** son los que usaremos con más frecuencia. Una subclase podrá tener un acceso "más o menos directo" a los campos y métodos de la superclase en función de lo que defina el programador, como veremos más adelante. Para referirnos a la herencia también se usa la terminología "es un". En concreto decimos que un objeto de la subclase es un objeto de la superclase, o más bien en casos concretos, que un Taxi es un Vehiculo, o que un Estudiante es una Persona. Sin embargo, al revés esto no es cierto, es decir, un Vehiculo no es un Taxi, ni una Persona es un Estudiante.

Dos subclases que heredan de una clase no deben tener información duplicada. Esto se consideraría un mal diseño. La información común debe estar en una superclase.

EJERCICIO

Se plantea desarrollar un programa Java que permita la gestión de una empresa agroalimentaria que trabaja con tres tipos de productos: productos frescos, productos refrigerados y productos congelados. Todos los productos llevan alguna información común como fecha de caducidad y número de lote, pero a su vez cada tipo de producto lleva alguna información específica, por ejemplo los productos congelados deben llevar la temperatura de congelación recomendada. Hay tres tipos de productos congelados: congelados por aire, congelados por agua y congelados por nitrógeno.

La empresa gestiona envíos a través de diferentes medios, y un envío puede contener cierto número de productos frescos, refrigerados o congelados. Identificar las 7 clases Java principales que podemos identificar dada la forma de funcionamiento de la empresa. Crear un esquema con las relaciones de herencia y/o uso entre las distintas clases.

Puedes comprobar si tu respuesta es correcta consultando en los foros aprenderaprogramar.com.

Próxima entrega: CU00685B

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188