

# Programación orientada a objetos

En este capítulo vamos a entrar en la teoría de objetos. Describiremos cuáles son las principales características de este paradigma, y veremos qué son los objetos, las clases y los métodos.

Analizaremos cómo se piensan los sistemas en función de este modelo y, hacia el final del capítulo, haremos una pequeña introducción en UML.

<b>El paradigma de programación</b>	<b>16</b>
Propiedades y comportamiento de los objetos	16
<b>Propuesta de un modelo de diseño</b>	<b>20</b>
Diagramas de clases	22
Diagramas de casos de uso	23
Diagramas de interacción o comportamiento	23
Diagramas de implementación	25
<b>Resumen</b>	<b>25</b>
<b>Actividades</b>	<b>26</b>

# EL PARADIGMA DE PROGRAMACIÓN

Cuando usamos una computadora, lo que estamos buscando es simplemente una solución a un problema, y para que resuelva dicho problema debemos decirle cómo hacerlo, ya que, en rigor de la verdad, las computadoras no pueden más que sumar **bits** y mover **bytes** de un lugar a otro. Nuestra tarea como programadores es, entonces, indicarle a la computadora qué es lo que queremos que haga; para ello debemos utilizar un lenguaje particular.

El lenguaje que entiende una computadora se denomina **binario**, pero como éste es difícil de leer y escribir para nosotros, debemos usar un lenguaje intermedio que luego será traducido a binario. El lenguaje, entonces, es una herramienta que nos servirá para indicarle a la computadora qué pasos debe seguir para resolver el problema en cuestión. Ahora bien, el modo en que especifiquemos la solución dependerá del paradigma de programación que usemos.

Dicho paradigma no es más que un modelo que representa un enfoque particular para la construcción de sistemas. No hay uno mejor que otro, sino que cada uno tiene ventajas y desventajas. Por otro lado, hay situaciones en que un paradigma resulta más adecuado que otro. El paradigma que vamos a desarrollar en el presente libro es el de **Programación Orientada a Objetos** (POO).

## Propiedades y comportamiento de los objetos

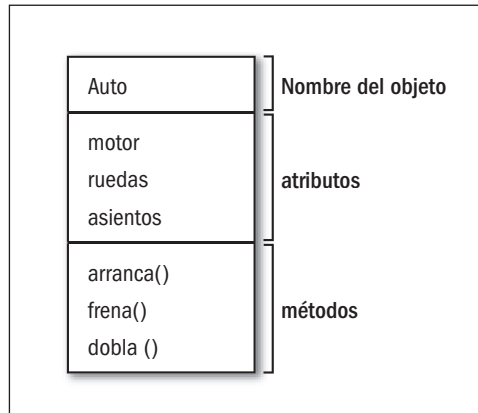
Lo primero que se nos ocurre preguntar es: ¿qué es un objeto? La respuesta es muy simple, bastará con mirar a nuestro alrededor para ver que estamos rodeados de **objetos**. Y extendiendo sólo un poco más la idea, notamos que a estos objetos les asignamos sustantivos para poder nombrarlos y diferenciarlos unos de otros.

Por ejemplo, si vamos a un aeropuerto, veremos aviones, automóviles, taxis, etc. Todos éstos son objetos; sin embargo, si observamos más detalladamente, notaremos que estos objetos tienen **propiedades** o **atributos** en común: poseen motor, ruedas y asientos, pero también se diferencian; por ejemplo, el avión puede volar y el auto no. Es decir, vemos que los objetos tienen un **comportamiento** propio.

De esta forma podemos definir a un objeto como una **entidad** compleja provista de **propiedades** (datos, atributos) y **comportamiento** (funcionalidad, métodos). Tomando en cuenta estas características, es conveniente afirmar que también representan objetos reales del mundo que nos rodea y palpamos cotidianamente.

Cada objeto expone una **interfaz** a otros objetos que especifica cómo éstos pueden interactuar con él, cómo pueden comunicarse con él. Esta interfaz está dada por un conjunto de métodos; así es como la interfaz del automóvil estará formada por

los **métodos** “arranca”, “frena”, “dobla”, etc., a través de los cuales podemos interactuar con el objeto (**Figura 1**).



**Figura 1.** Representación del objeto **Auto** con sus atributos y métodos.

Por otro lado, cabe aclarar que el comportamiento es exclusivo del objeto; si bien algunos objetos a simple vista son iguales, internamente pueden ser muy distintos. Por ejemplo, si aprendemos a manejar en un automóvil con motor diesel, podremos sin ningún problema manejar uno que funcione con nafta. Es decir, los dos objetos se nos presentan de igual forma, pero es indiscutible que los motores son muy distintos. Esto se conoce como **encapsulamiento**: los objetos presentan la misma interfaz pero ocultan información de su funcionamiento.

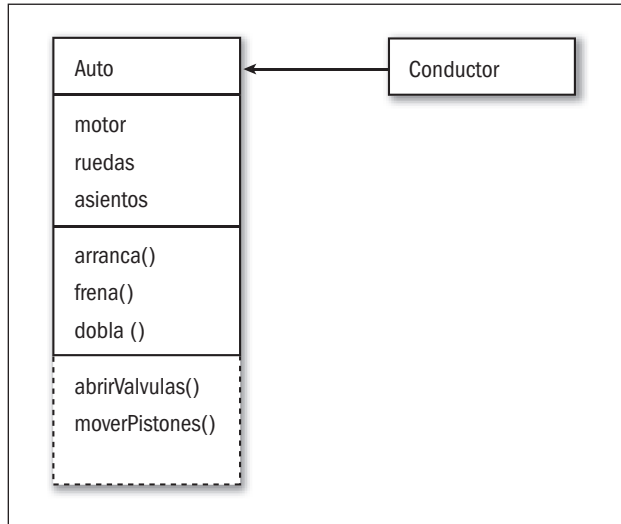
Gracias al encapsulamiento se puede cambiar un objeto por otro que presente la misma interfaz, y todo debería funcionar igual.

En segundo lugar vale mencionar que el encapsulamiento protege también al objeto de usos indebidos e inapropiados. Volvamos al ejemplo anterior para detallar un poco más esta situación. El automóvil utilizará otros mecanismos para llevar a cabo su interfaz, como, por ejemplo, abrir válvulas, mover pistones, etc., que no les permitirá usar a otros objetos (**Figura 2**).



## DEPENDERÁ DE NOSOTROS...

Ahora que conoceremos los conceptos básicos del paradigma, vale agregar que el éxito o el fracaso de nuestro programa dependerá de la capacidad que tengamos para reconocer los objetos que trabajarán en conjunto para resolver el problema deseado.



**Figura 2.** El objeto **Conductor** no puede usar los métodos *abrirVálvulas()* y *moverPistones()*.

Por lo general, cuando definimos un objeto lo hacemos en función de otros objetos conocidos. Si alguien nos habla de un objeto en particular cuyo nombre no conoce, instantáneamente empezará diciendo “es como...” para describirlo y luego deberá agregar las características particulares del objeto que desea que identifiquemos. En el ejemplo del automóvil, podría decir “es como un taxi, pero más chico”, o “es como una moto, pero tiene cuatro ruedas”... Es decir que define al automóvil a través de objetos de similares características. Sin darnos cuenta hacemos **clasificaciones**; ahora, generalizando, nos damos cuenta de que todos son transportes. En el paradigma de objetos esto se conoce como **herencia**, y sirve para no tener que definir **comportamientos** de forma repetitiva.

Desarrollemos el ejemplo; en nuestro caso tendremos un objeto **transporte**, que tendrá propiedades como **cantidad de pasajeros**, **cantidad de puertas**, etc., y métodos como **anda**, **frena** y **dobla**. De esta manera definiríamos un **automóvil** como un **transporte**, agregando las particularidades del **automóvil** que no estén definidas en **transporte**.

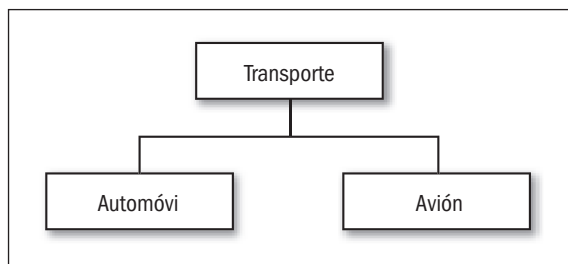
Ahora que sabemos agrupar los objetos en **clases**, podríamos agregar más objetos a nuestro modelo. Para seguir con el ejemplo, pensemos en el objeto **avión**: sin lugar a dudas, es un transporte; es decir que las clases simplemente son conjuntos de objetos que comparten propiedades y comportamientos.

Al agregar el objeto **avión** definido como un **transporte**, **heredará** de éste las propiedades (**cantidad de pasajeros**, **cantidad de puertas**) y los métodos (**anda**, **frena** y **dobla**). Si explotamos aún más el ejemplo, notamos que no es lo mismo

hacer andar un automóvil que un avión, de forma tal que necesitamos agregar el método **anda** en **automóvil**, para que ruede, y en **avión** para que vuele. Esto se denomina **polimorfismo**, y nos permite tener muchas formas de comportamiento; o sea que la referencia al método **anda** producirá el comportamiento correcto según el objeto al que se lo esté ordenando.

Para ordenarle a un objeto que haga algo, debemos mandarle un **mensaje**. A través de los mensajes establecemos la comunicación entre los objetos de forma tal que les ordenamos ejecutar un método con algunos parámetros. En la **Figura 2** veíamos que el conductor le ordenaba al auto que arrancara; esta orden se la daba a través de un mensaje.

Revisando el ejemplo del transporte, el automóvil y el avión, notamos que el gráfico evidencia una jerarquía: a la cabeza está el transporte, y de éste cuelgan el avión y el automóvil. En este paradigma, esto se denomina **jerarquía de herencia** (**Figura 3**). Notamos, también, que dicha jerarquía siempre tiene forma de árbol. Entonces encontramos una nueva agrupación de nivel superior. En resumen, los objetos se agrupan en clases, y las clases, en árboles, siempre y cuando reflejen un comportamiento común.



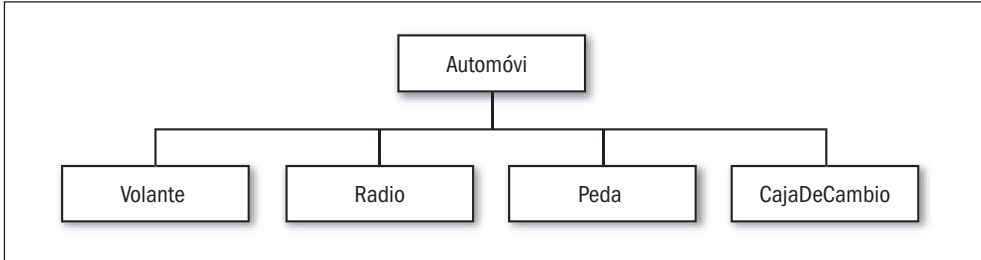
**Figura 3.** Árbol de jerarquía de herencias. En este caso, los objetos **Avión** y **Automóvil** heredan del objeto **Transporte**.

El automóvil, aparte de ser un objeto y además de ser un transporte, también está **compuesto** por más objetos con comportamientos distintos. Es decir, el automóvil está compuesto por un volante, una radio, pedales, una caja de cambios, etc.

## III COMPORTAMIENTO DE UNA CLASE

Una clase no está limitada a comportarse sólo como lo define su padre, sino que, además, puede definir un comportamiento propio, es decir que se pueden agregar nuevos métodos en ella. Veremos más sobre este aspecto a lo largo de este libro.

Pero, a su vez, la radio tiene otros componentes, como display, botones, etc. Al igual que en otros ámbitos, aquí nos encontramos ante una jerarquía de elementos. Ésta se denomina **jerarquía de composición**, y sirve para representar que uno o varios objetos están dentro de otro que los contiene (**Figura 4**).



**Figura 4.** Árbol de jerarquía de composición. En este caso, el **Automóvil** está compuesto por los objetos **Volante**, **Radio**, **Pedal** y **CajaDeCambio**.

Para finalizar, si tenemos un solo objeto **automóvil**, no significa que nuestro programa podrá tener un solo automóvil. Para entender esto hay que agregar el concepto de **instancia**, que nos permite crear la cantidad de automóviles que deseemos. Por ejemplo, si nuestro objeto **automóvil** tiene una propiedad **color**, podemos tener instancias de **automóvil** con la propiedad **color** en **rojo**, **azul** o **verde**. Para lograr esto, cada objeto deberá tener un **método**, que nos permita crear una **instancia** de éste.

## PROPUESTA DE UN MODELO DE DISEÑO

El supuesto problema para resolver es el juego del *TA-TE-TI*, donde dos participantes tienen tres fichas cada uno y las irán ubicando en un tablero de 3x3 por turnos, de acuerdo con su conveniencia. Para resolver este problema, lo primero que vamos a hacer es determinar cuáles son los objetos.

A simple vista tenemos **Jugador**, **Tablero** y **Ficha**. Cada una de las seis fichas será una instancia, pero tenemos un solo objeto que las representará a todas; por eso es por lo que los nombres de los objetos suelen ir en singular. El tablero tiene casilleros donde irán ubicadas las fichas. Entonces, agregamos a nuestra lista el objeto **Casilla**.

Por otra parte, el jugador podrá ser un humano o la computadora. Entonces, agreguemos también estos objetos a nuestra lista:

- Jugador
- Tablero
- Ficha
- Casilla
- Computadora
- Humano

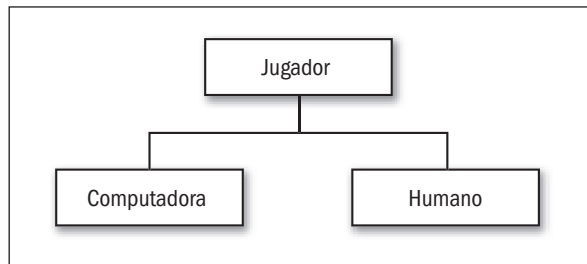
Una vez que estemos seguros de haber determinado todos nuestros objetos, debemos agregarle a cada uno atributos y métodos, para definir su comportamiento.

Comenzando por **Tablero**, el atributo más importante que encontraremos son las casillas; para simplificar el ejemplo, agregaremos nueve (más adelante veremos cómo se puede optimizar esta solución). Entre los métodos tenemos **ponerFicha()**, **sacarFicha()** y, para verificar si hay un ganador, **buscarTaTeTi()**.

Siguiendo por la **Casilla**, vemos dos atributos: uno es **color**, ya que las casillas de un tablero pueden ser de dos colores distintos, imitando al tablero de las damas; el otro es el objeto **ficha**, al cual le pondremos el nombre **iFicha**, para determinar que es una instancia del objeto **ficha**.

Finalmente, tenemos el objeto **Jugador**, que tiene tres instancias del objeto **ficha**. Por otra parte, están el objeto **humano**, que tiene un atributo **nombre**, y el objeto **Computadora**, al que agregaremos un atributo **inteligencia** para que nuestro *TA-TE-TI* contemple varios niveles de juego.

Una vez determinados los atributos y los métodos de cada objeto, procedemos a dibujar la jerarquía de herencias. De esta manera, vemos que tanto el **Humano** como la **Computadora** son jugadores, y ambos juegan con fichas. En otras palabras, tendremos **Jugador**, y debajo de éste, **Humano** y **Computadora**, como lo indica la **Figura 5**.



**Figura 5.** Árbol de jerarquía de herencia.

Ahora procedemos a graficar la jerarquía de composición. Una de las características es su simplicidad ya que, sin percibirlo, fuimos armándola cuando agregábamos los atributos a los objetos (**Figura 6**).



**Figura 6.** Árboles de jerarquía de composición.

Éste es un modelo simplificado de representación; una vez terminado, ya estamos listos para escribir el código correspondiente.

En sistemas, la idea general es que no hace falta modelar porque se pierde tiempo, pero pensemos en lo que pasaría si un arquitecto no hiciera un plano antes de construir una casa. Por otra parte, cuando modelamos tendemos a creer que nuestra representación de la realidad es la más clara que existe, sin tener en cuenta que para otros puede ser difícil o imposible de comprender. Para evitar esta situación existe diversidad de herramientas que ayudan al modelado de la programación orientada a objetos a través de estándares. Una de las más utilizadas es la de **UML**.

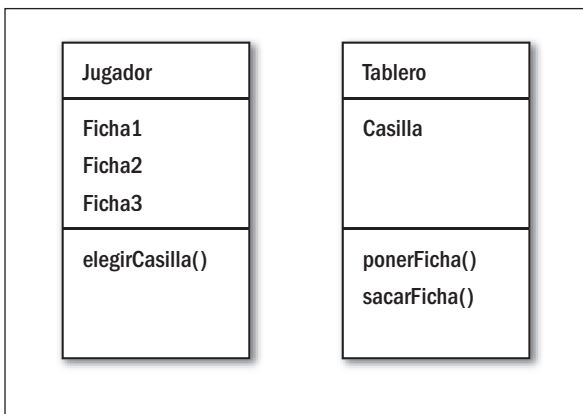
Esta herramienta provee varias representaciones; quizá esporádicamente necesitaremos usar algunas de ellas. En contraposición, otras son mucho más comunes; tanto, que ya hemos utilizado varias de ellas. UML usa como herramientas los diagramas gráficos para representar el sistema. Éstos son:

- Diagrama de clases.
- Diagrama de comportamiento.
- Diagrama de casos de uso.
- Diagramas de implementación.

A continuación haremos una explicación breve de lo que representa cada diagrama. Al efecto de ejemplificar modelaremos en cada caso nuestro problema del *TA-TE-TI*.

## Diagramas de clases

Los diagramas de clases representan un conjunto de elementos del modelo que son estáticos, como las clases y los tipos, sus contenidos y las relaciones que se establecen entre ellos (**Figura 7**).

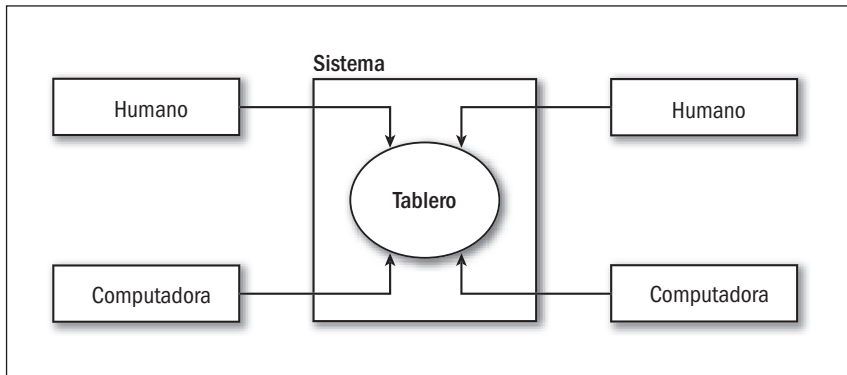


**Figura 7.** Diagrama de clases del *TA-TE-TI*.

## Diagramas de casos de uso

Los diagramas de casos de uso sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.

A los usuarios y/u otros sistemas en este modelo se los denomina actores. Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona una respuesta a eventos que se producen en él (**Figura 8**).



**Figura 8.** Diagrama de clases de uso. Vemos cómo pueden jugar, por cada lado, un **Humano** y una **Computadora**. Ésta es parte del sistema, por eso está integrada al cuadrado que lo representa.

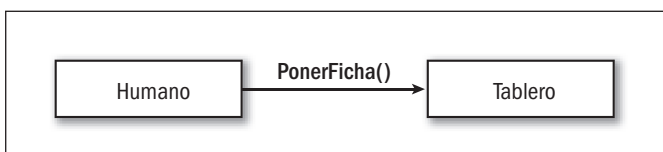
## Diagramas de interacción o comportamiento

Muestran las interacciones entre objetos ocurridas en un escenario (parte) del sistema. Hay varios tipos:

- Diagrama de secuencia.
- Diagrama de estado.
- Diagrama de actividades.
- Diagrama de colaboración.

### Diagrama de secuencia

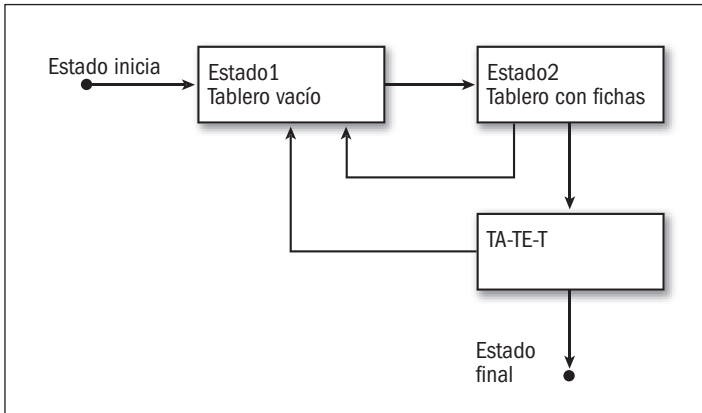
El diagrama de secuencia muestra la interacción entre los objetos que tiene lugar a través del intercambio de mensajes (**Figura 9**).



**Figura 9.** El objeto **Humano** le pasa el mensaje **PonerFicha()** al objeto **Tablero**.

### Diagrama de estados

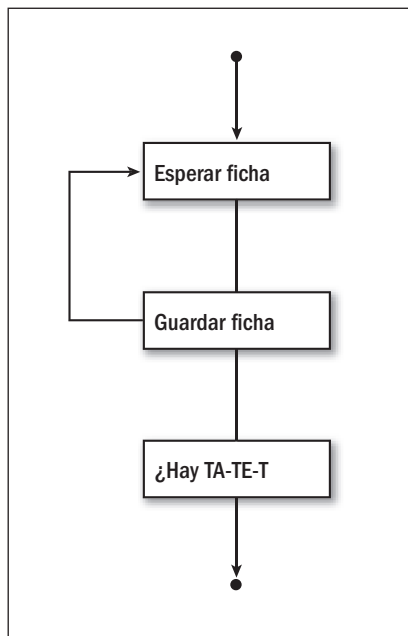
Este diagrama muestra los estados posibles que puede tomar nuestro sistema de acuerdo con los estímulos que recibe (**Figura 10**).



**Figura 10.** Algunos estados que puede tomar nuestro sistema.

### Diagrama de actividades

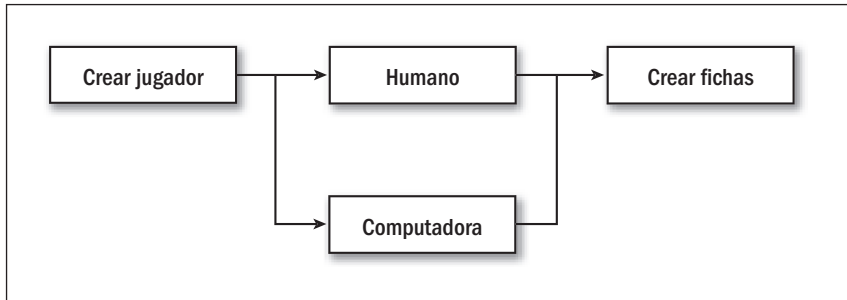
La función del diagrama de actividades es de suma importancia, ya que muestra la actividad de cada clase en particular (**Figura 11**).



**Figura 11.** Diagrama de actividades de la clase **Tablero**.

## Diagrama de colaboración

Este diagrama muestra cómo colaboran los objetos entre sí para lograr el objetivo común (**Figura 12**).



**Figura 12.** Cuando empieza una nueva partida, el sistema le pide a **Jugador** que se cree un jugador; luego esta clase crea un **Humano** o una **Computadora**, y ésta, a su vez, crea las fichas para jugar.

## Diagramas de implementación

Por último, los diagramas de implementación muestran los aspectos físicos del sistema. Incluyen la estructura del código fuente y la implementación.

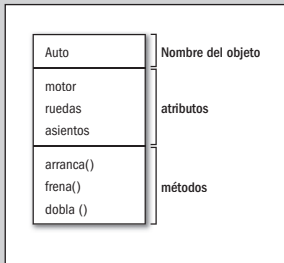
## RESUMEN

En este capítulo vimos las características principales del paradigma orientado a objetos, intentando dar un pantallazo de la teoría que después desarrollaremos a lo largo del libro, cuando ya estemos escribiendo código. Para cerrar este capítulo, conocimos las diferentes formas de modelar un sistema a través de los diagramas.



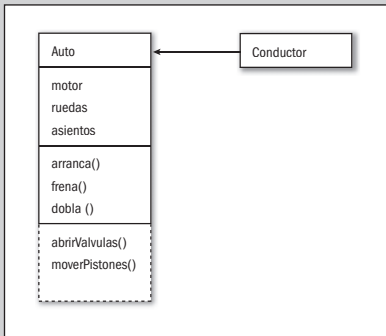
## TEST DE AUTOEVALUACIÓN

1 ¿Qué es un objeto?



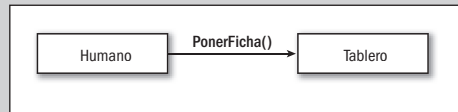
2 ¿Qué diferencia existe entre clase, objeto e instancia?

3 ¿Qué es un método?

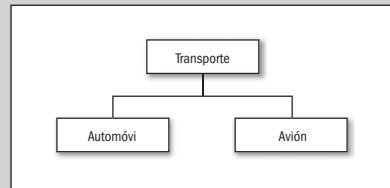


4 ¿Qué es UML y para qué se usa?

5 ¿Cómo se comunican los objetos?



6 ¿Qué es herencia?



7 ¿Qué es polimorfismo?

8 ¿Qué diferencias y semejanzas existen entre la jerarquía de herencia y la jerarquía de composición?