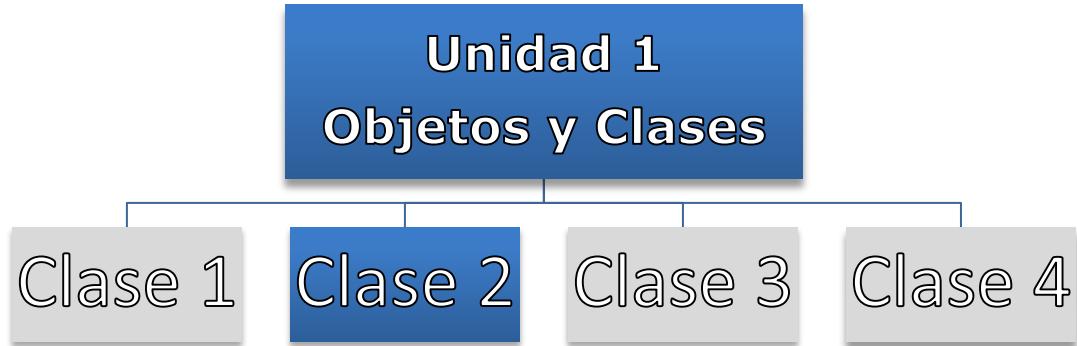


---

## PROGRAMACIÓN ORIENTADA A OBJETOS

---

---



---

Docente titular y autor de contenidos: Prof. Ing. Darío Cardacci

---

---

## Presentación

---

La clase 2 corresponde a la unidad 1 de la asignatura. En esta clase avanzaremos sobre el **modelo orientado a objetos**.

En particular analizaremos las características de las clases y los objetos.

Los siguientes **contenidos** conforman el marco teórico y práctico de esta unidad. A partir de ellos lograremos alcanzar el resultado de aprendizaje propuesto: En negrita encontrará lo que trabajaremos en la clase 2.

- El modelo orientado a objetos.
- Jerarquías "Es - Un" y "Todo - Parte".
- Concepto de Clase y Objeto.
- **Características básicas de un objeto: estado, comportamiento e identidad.**
- **Ciclo de vida de un objeto.**
- **Modelos. Modelo estático. Modelo dinámico. Modelo lógico. Modelo físico.**
- **Concepto de análisis diseño y programación orientada a objetos.**
- **Conceptos de encapsulado, abstracción, modularidad y jerarquía.**
  
- **Concurrencia y persistencia.**
- **Concepto de clase.**
- **Definición e implementación de una clase**
- **Campos y Constantes**
- Propiedades. Concepto de Getter() y Setter(). Propiedades de solo lectura. Propiedades de solo escritura. Propiedades de lectura-escritura. Propiedades con indizadores. Propiedades autoimplementadas. Propiedades de acceso diferenciado.
- Métodos. Métodos sin parámetros. Métodos con parámetros por valor. Métodos con parámetros por referencia. Valores de retorno de referencia.
- Sobrecarga de métodos.
- Constructores. Constructores predeterminados. Constructores con argumentos.
- Finalizadores.

- **Clases anidadas.**

A continuación, le presentamos un detalle de los contenidos y actividades que integran esta clase. Usted deberá ir avanzando en el estudio y profundización de los diferentes temas, realizando las lecturas requeridas y elaborando las actividades propuestas, algunas de desarrollo individual y otras para resolver en colaboración con otros estudiantes y con su profesor.

#### 1. El modelo orientado a objetos (Teoría)

##### **Lectura obligatoria**

Cardacci Dario y Booch, Grady. Orientación a Objetos. Teoría y Práctica. Buenos Aires, Argentina. Pearson Argentina, 2013. **Capítulo 2.**

#### 2. Clases y Objetos (Teoría)

##### **Lectura obligatoria**

Cardacci Dario y Booch, Grady. Orientación a Objetos. Teoría y Práctica. Buenos Aires, Argentina. Pearson Argentina, 2013. **Capítulo 3.**

#### 3. Clasificación (Teoría)

##### **Lectura obligatoria**

Cardacci Dario y Booch, Grady. Orientación a Objetos. Teoría y Práctica. Buenos Aires, Argentina. **Pearson Argentina, 2013. Capítulo 4.**

#### 4. Introducción a las clases y los objetos

##### **Lectura recomendada**

Deitel Harvey M. y Deitel Paul J. Cómo programar C#. Segunda Edición Pearson Educación. 2007. Capítulo IV.

## Links a temas de interés

- Clases: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/classes>
  - Propiedades: <https://docs.microsoft.com/es-mx/dotnet/csharp/programming-guide/classes-and-structs/properties>
  - Métodos: <https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/methods>
  - Clases y objetos: <https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/classes-and-objects>
- 

Lo invitamos a comenzar con el estudio de los contenidos de la clase 2.

## 2. El modelo orientado a objetos

Si bien el modelo orientado a objetos abarca, el “análisis orientado a objetos”, el “diseño orientado a objetos” y la “programación orientada a objetos” la incumbencia de esta asignatura se centra en esta última.

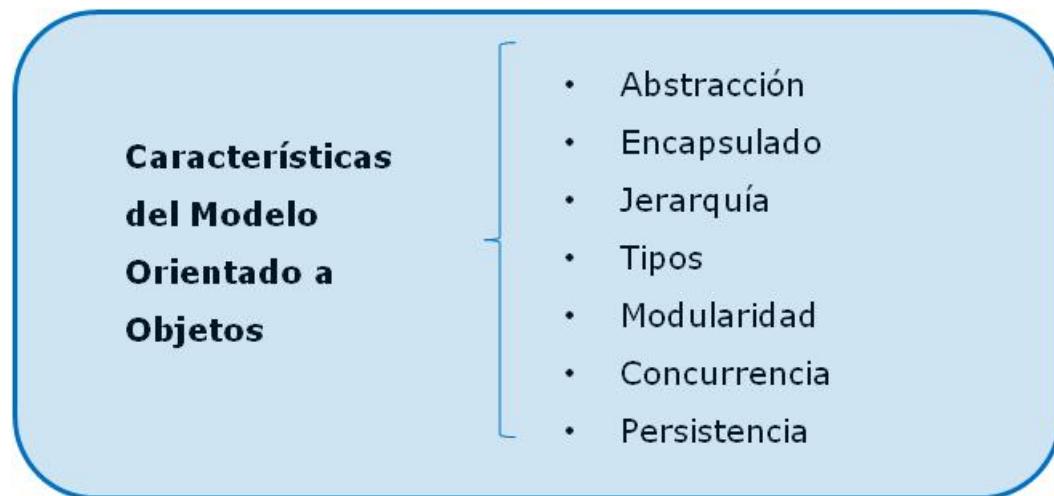
La **programación orientada a objetos** es una forma de implementación. Lo que se implementa es aquello que previamente fue analizado y diseñado. Esta implementación permite que los programas y sistemas obtenidos, sean un conjunto organizado y colaborativo de objetos.

Debido a la “jerarquía de tipos (es-un)” todo objeto se corresponderá con una clase y las clases debido a la “jerarquía estructural (todo-parte)” estarán ordenadas jerárquicamente.

En todo modelo orientado a objetos se pueden observar ciertas características. Algunas de ellas constituyen la esencia misma del modelo.

Algunos autores las dividen en *características principales* y *secundarias*, considerando que las del primer grupo caracterizan a un modelo orientado a objetos y si hubiera una o más ausentes habría que cuestionarse si el modelo es realmente orientado a objetos. En otro sentido, las secundarias le aportan al modelo, pero pueden no estar presentes.

Hemos considerado innecesaria esta división en la actualidad ya que las que se consideraban secundarias son soportadas en la mayoría de los lenguajes y tecnologías, Simplemente las enumeraremos como un conjunto.



Analicemos cada característica para poder profundizar en las particularidades del modelo.

- La **abstracción**, como ya mencionamos anteriormente es un mecanismo por el cual los seres humanos administramos la complejidad. Pero profundizaremos este concepto y le daremos el sentido que deseamos que posea dentro de la orientación a objetos.

Existen numerosas definiciones sobre abstracción, cada una con el tinte de la disciplina que desea utilizar este concepto.

Desde la óptica de la psicología y la psicopedagogía se entiende como un proceso o capacidad mental que poseen los individuos para comprender o deducir la esencia o concepto de un objeto o situación determinada.

En las artes el concepto de lo abstracto se desarrolla a partir de la oposición a la figuración, pudiéndose asociar lo figurativo como más cercano a lo definido u específico y lo abstracto a lo menos cercano.

La filosofía la comprende como una operación intelectual que consiste en separar mentalmente lo que es inseparable en la realidad. La abstracción es el precedente o el instrumento de la generalización, porque no se pueden concebir los conocimientos generales, sin eliminar lo individual, es decir, sin abstraer. Toda idea generalizada es abstracta y no concreta, porque la abstracción no es función de la imaginación, sino de la razón. A lo abstracto se opone lo concreto.

*Desde el punto de vista de la orientación a objetos, abstraer significa poder separar una porción de la realidad perteneciente al dominio del problema. Esa realidad se representa por medio de sus características más representativas y esenciales. Así se distingue de todas las demás y establece fronteras conceptuales bien definidas desde la perspectiva de cualquier observador.*

- El **encapsulamiento** es una característica del modelo orientado a objetos que oculta los detalles de implementación que este posee. El encapsulamiento establece barreras concretas que permiten distinguir el alcance de una abstracción.

Por otro lado, por contraposición de conceptos, pone de manifiesto el sentido de **interfaz**. La **interfaz** es el elemento constituyente de un objeto que representa su visión externa. Es improbable que nos podamos centrar en la idea de encapsulamiento sin pensar en la interfaz. Son las dos caras de la moneda. Ambos conceptos conforman la noción del objeto como un todo. En encapsulamiento enfatizará la noción interna de la implementación y la interfaz su visión externa, o sea, como lo perciben los demás.

A modo de comentario final sobre el encapsulamiento, no hay que confundir los términos **ocultar** con **no utilizar**.

Si bien el ocultamiento de alguna manera encapsula, ya que no tengo posibilidad de utilizar aquello que está oculto, nos seduce más la idea de **respetar el encapsulamiento**. Esto se logra independientemente a si la implementación está oculta o no para quien la utiliza. Si existe un *compromiso* de utilizar solo al objeto por medio

de lo que publica en su interfaz y así se hace, se está respetando el encapsulamiento.

Muchas veces se considera oculto a un código cuando no es accesible; reforzamos la idea que se *respeta el encapsulamiento* también cuando, a pesar de que la implementación sea visible o accesible, no se la utiliza.

No profundizaremos ahora sobre este tema, debido a que lo haremos más adelante, cuando abordemos aspectos prácticos que resaltarán las formas correctas e incorrectas de proceder. En ellas, observará claramente que pudiendo respetar el encapsulamiento si se hace un mal uso de las técnicas de programación se lo puede corromper. Se pondrá de manifiesto la incomodidad que se genera en algunos escenarios de **ocultar** el código en el sentido tradicional y las ventajas de no hacerlo, y a pesar de ello respetar el encapsulamiento por la decisión de no acceder a la implementación independientemente a si está oculta o no.

- La **Jerarquía** es un ordenamiento o clasificación de abstracciones. Esto se torna muy útil en sistemas de cierta envergadura, debido a que poseerán un número muy importante de abstracciones. Seguramente no todas estas abstracciones tengan la misma importancia en términos de utilidad para el sistema. Algunas serán sumamente importantes y otras casi triviales. Esto último amerita que exista un ordenamiento de abstracciones.

Las dos formas más populares de ordenamiento / **jerarquías** utilizadas en la programación orientada a objetos se traducen en la **herencia** y la **agregación**, temas que se desarrollan más adelante.

La herencia permitirá crear una jerarquía del tipo “**es-un**” o lo que es lo mismo de “**generalización-especialización**”, mientras que la agregación crea una jerarquía del tipo “**todo-parte**”. En términos de jerarquía, cuando nos referimos a una del tipo “**es-un**”, diremos que el elemento más **generalizado** posee una jerarquía superior al más **especializado**. Cuando nos referimos a una jerarquía del tipo “todo-parte”, diremos que el **todo** posee una jerarquía superior a las **partes**.

- Los **Tipos**, también conocidos con el nombre de **tipificación**, establecen un concepto que permite que un mismo objeto pueda adoptar distintos tipos.

Como bien menciona Grady Booch en su libro "Análisis y diseño Orientado a Objetos", "...los tipos son la puesta en vigor de la clase de los objetos..." .

Podrá comprender este concepto en profundidad cuando se aborde la práctica, por lo tanto, si se encuentra un poco confundido, no se preocupe, le aseguramos que cuando comencemos a exemplificar con código todo lo dicho en estas líneas, seguramente las dudas se desvanecerán.

A modo de ejemplo solo mencionaremos que, para poder realizar ciertas operaciones, necesitaremos que los objetos que intervienen en ellas sean del mismo **tipo** o de **tipos compatibles**. Un *objeto* podrá adoptar distintos *tipos* dependiendo de las definiciones realizadas en la *clase* que le dio origen.

- La **Modularidad** es un concepto que existe desde tiempos anteriores a la **programación orientada a objetos**. Básicamente plantea la necesidad de dividir un programa en partes con el objetivo de poder administrarlo de manera más eficiente.

Cada **módulo**, dependiendo de la tecnología que se utilice, tendrá distintas caracterizaciones. Cada parte posee una frontera nítidamente definida. Al realizar esta actividad se deben tener en cuenta al menos dos aspectos: el **acoplamiento** y la **cohesión** que poseen los módulos.

Cuando decimos **acoplamiento** nos referimos al nivel de conectividad entre módulos. Si bien dividir en módulos hace que nuestro diseño pueda ser fácilmente comprendido, documentado y organizado, exagerar en este aspecto, lleva a que la comunicación masiva genere problemas.

Se debe crear la cantidad de módulos necesarios teniendo en mente que el **acoplamiento** hay que mantenerlo bajo.

La **cohesión** se refiere al contenido interno de un módulo. Estos contenidos deben tener algún tipo de relación justificada para estar todos juntos allí, cuanto más relacionados estén, decimos que el módulo es *altamente cohesivo*. Un bajo **acoplamiento** entre módulos y una alta **cohesión** en cada uno de ellos, son atributos deseables en un sistema.

- La **concurrencia** establece que varios objetos pueden funcionar simultáneamente. Esto es muy deseable ya que se pueden aprovechar las características actuales de los sistemas de hardware en cuanto a procesamiento en paralelo y manejo de múltiples hilos de ejecución.

Cuando dos o más objetos están funcionando *concurrentemente* decimos que los mismos están **activos**. Si consideramos los procesos del sistema operativo podemos identificar dos tipos de **concurrencia**.

- La **concurrencia pesada** es aquella que en un proceso del sistema operativo se está ejecutando solo un elemento del sistema en una porción propia de memoria.
- La **concurrencia liviana** es aquella que en un proceso del sistema operativo se están ejecutando varios elementos del sistema, compartiendo una porción de memoria. Para el intercambio de datos la **concurrencia liviana** es más indicada. Si en lugar de ello priorizamos la seguridad e independencia de los procesos la más adecuada es la **concurrencia pesada**.

- La **persistencia** permite que los **estados de los objetos** perduren en el tiempo y el espacio. Este concepto está muy relacionado con el **ciclo de vida** de los objetos.

El *ciclo de vida* de un objeto queda establecido desde que el objeto es creado hasta que el objeto es matado (quitado de memoria). De

una u otra manera cuando hablamos de **persistir** la idea que subyace es la de **trascender en el tiempo**.

Ahora bien, ¿qué es lo que trasciende en el tiempo? En general, una visión establece que lo que trasciende en el tiempo es el **estado del objeto**. Es decir, desde que el objeto se crea, hasta que se mata, su **estado** existe y no dejará de existir. Pero también habría que considerar que necesitamos que algunos datos trasciendan entre distintas ejecuciones del programa.

Debido a esto y a las características físicas de las memorias de las computadoras, es que los datos que posee un objeto se deben almacenar. Generalmente se almacenan en una base de dato relacional, generando lo que conocemos como mapeo-objeto-relacional (ORM por su sigla en inglés) tema que excede a lo que estamos tratando aquí.

Simplemente mencionaré que si guardamos los datos que un **objeto** posee en un momento dado, también habrá que guardar (persistir) la estructura que le da origen a ese **objeto**, o sea, la **clase**.

Como para culminar esta introducción a la **persistencia** mencionaré que, si bien **persistir** lo asociamos a la idea de conservar los datos que manejan los objetos en el tiempo y espacio, con independencia a cuándo y quién lo creó, en una mirada más amplia deberíamos considerar también otras posibilidades. Como mínimo analizar la conservación de las clases que le dan origen a los objetos que mantendrán sus estados y además, a la sobrevivencia de esos datos al ciclo de vida del propio programa que les dio origen.

### 3. Clases y Objetos

Conocido el marco general del modelo de programación orientado a objetos, le proponemos adentrarnos en sus particularidades a través de algunas definiciones que favorecerán su comprensión. Explicaremos entonces:

- Las generalidades de los objetos

- Características de los objetos
- Relaciones entre objetos
- Generalidades de las clases
- Relaciones entre clases
- Medida de la calidad de una abstracción

---

### 3. 1. Generalidades de los objetos

---

Un **objeto** es cualquier cosa real o abstracta, visible o invisible, tangible o intangible sobre la cual se posee una comprensión intelectual. Se podría decir que si poseemos la capacidad de poder pensarla o imaginarla podemos conceptualizarla como un *objeto*.

Dentro del campo que estamos estudiando los *objetos* representarán aspectos de la realidad. La idea de modelar un sistema **orientado a objetos** es poder representar cada elemento de la realidad, que se encuentre dentro del dominio del problema, como un *objeto*. Luego cada *objeto* interactuará con el resto de la misma manera que lo hace en la realidad analizada.

Como punto de llegada y por sumatoria de las funcionalidades otorgadas a cada *objeto* y las relaciones establecidas entre ellos, obtendremos nuestro sistema.

Ahora bien, cabe aclarar que, si bien cada *objeto* representará un elemento de la realidad, lo representará por medio de sus **atributos** y **comportamientos** más relevantes.

Los **atributos** de un *objeto* son el conjunto de características, cualitativas o cuantitativas, que lo describen, mientras que el **comportamiento** está

*Un **objeto** es cualquier cosa real o abstracta, visible o invisible, tangible o intangible (...) si poseemos la capacidad de poder pensarla o imaginarla podemos conceptualizarla como un *objeto*.*

dado por el conjunto de acciones y reacciones que este *objeto* posee. Al referirnos entonces a que un *objeto* estará representado por un conjunto relevante de **atributos** y **comportamientos**, este será una versión simplificada de lo que observamos en la realidad. A esta versión simplificada la denominamos **abstracción**.

*"En definitiva podemos decir que al terminar de representar la realidad del dominio del problema que estamos analizando, contaremos con un conjunto de **abstracciones** que representan a los elementos/objetos reales y las relaciones que existen entre ellos."*

En adelante denominaremos a estas abstracciones **clases**, por lo tanto, **clase** y **abstracción** pueden ser consideradas como sinónimos o términos intercambiables.

*Clase y abstracción pueden ser considerados sinónimos o términos intercambiables*

En el gráfico siguiente se puede observar como a partir de un conjunto de autos reales (**objetos reales**) y aplicando la **capacidad de abstraer** por parte del observador (usted), se obtiene la **abstracción o clase "AUTO"** que no es más que un modelo simplificado que los representa en términos de características y comportamientos.

Autos Reales / Objetos Reales



Abstracción / Clase  
AUTO

Característica:

Peso  
Color  
Potencia  
Cantidad de Puertas  
...

Comportamiento:

Arrancar  
Frenar  
Acelerar  
...

Cómo puede observarse, una *clase* no es más que una **especificación estática** que establece cuales fueron las características y comportamientos más relevantes de lo que se desea representar. El interrogante que seguramente posee el lector en este momento es: *¿si las clases son estructuras estáticas que es lo que se ejecuta en la computadora y conforma lo que normalmente llamamos programas?*

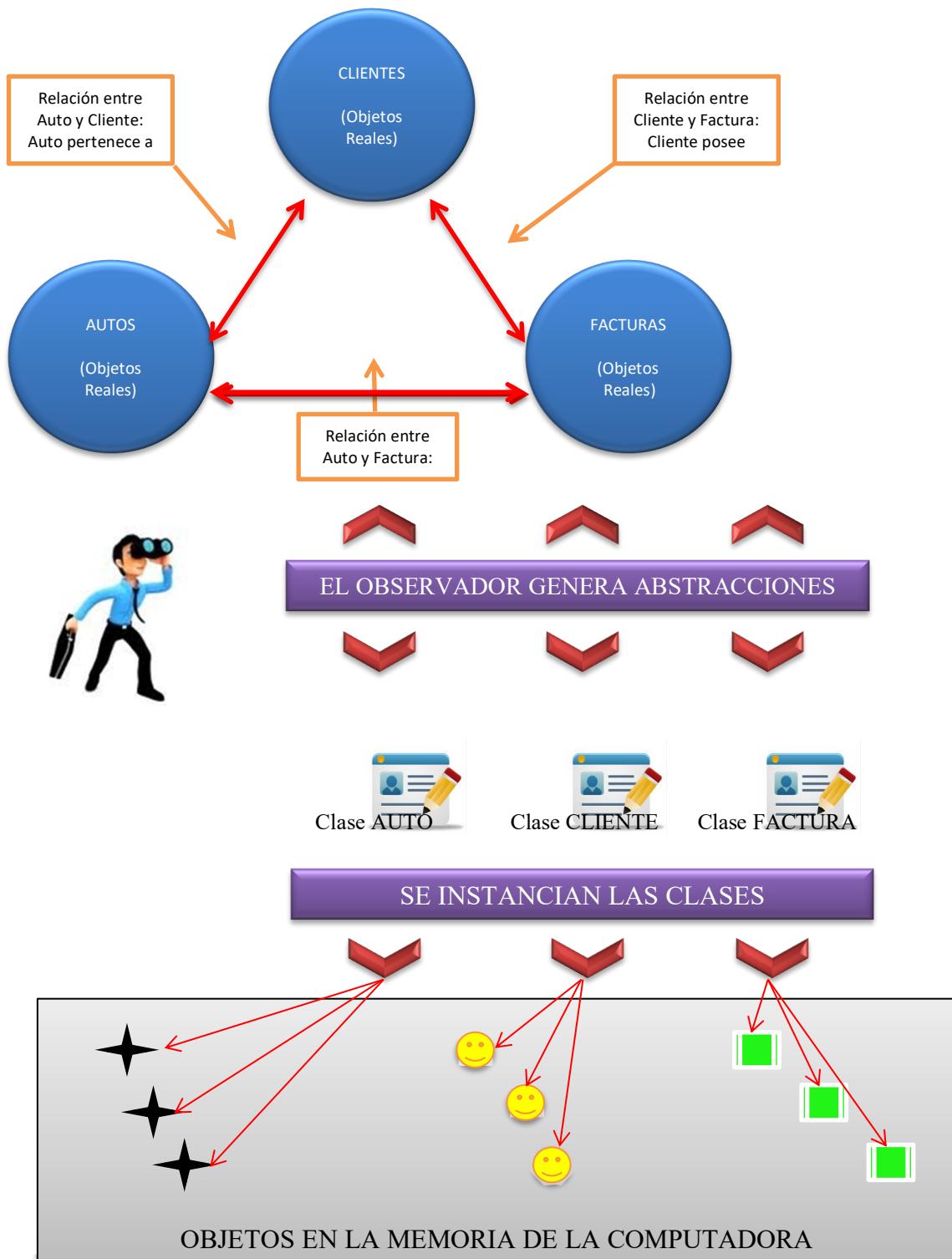
La respuesta a este interrogante es que las *clases* pueden ser sometidas a una acción denominada "**Instanciación**". Esta acción da como resultado también *objetos*, pero estos *objetos* son "**objetos virtuales**" cuya existencia se da solo en la memoria de la computadora y en tiempo de ejecución, conformando las estructuras dinámicas que son la esencia de todo programa orientado a objetos.

De esta manera se cierra el ciclo de la siguiente forma:

1. Un **observador** determina el **alcance del sistema**, es decir, establece el **alcance del dominio del problema**.
2. Ese dominio posee muchos "**objetos reales**", por medio de la **abstracción** se obtienen **clases**, cada una de estas **clases** son el modelo simplificado en términos de **características y comportamientos** de un conjunto de "**objetos reales**".
3. Luego se establece qué relaciones se dan entre estas **clases** y las establece (tema que desarrollaremos más adelante).
4. Finalmente **instancia** cada *clase* tantas veces como "**objetos virtuales**" deseé obtener, donde cada uno representará en la computadora, a un y solo un "**objeto real**" observado dentro del dominio del problema. Estos "**objetos virtuales**" se ajustan perfectamente a las características y comportamientos definidos en las clases que le han dado origen, y también se relacionan de acuerdo a las definiciones que poseen esas clases, constituyendo de esta forma el programa o sistema deseado.

Ahora que está presentado el esquema básico y clarificado en el gráfico siguiente, simplemente hablaremos de *objetos* sin distinguir entre **reales** o **virtuales**, ya que el lector lo hará por sí mismo dependiendo del ámbito donde se encuentre.

Este ámbito será observando la realidad y definiendo clases, lo cual determinará que estamos hablando de "**objetos reales**" o **instanciando clases** para obtener "**objetos virtuales**".



---

### 3.2. Características de los objetos

---

Todos los **objetos** poseen tres características:

- **Estado**
- **Comportamiento**
- **Identidad**

Analizaremos cada una:

**Estado:** "El estado de un objeto es el conjunto de características, cada una de ellas con un valor posible de su dominio, en un momento dado".

Al definir una característica también se define el dominio posible de valores (conjunto de valores) que puede adoptar. Por ejemplo, si la característica en AUTO fuera color, el posible dominio de valores sería: blanco, negro, azul, verde, etc.

Por lo tanto, dada una clase AUTO donde el conjunto de características que la definen es: marca, modelo, peso, color, potencia, año de patentamiento y precio, si tenemos un objeto AUTO producto de haber instanciado a esa clase, podríamos decir que su estado en un momento M es:

marca: ..... Toyota  
modelo: ..... Corolla  
peso: ..... 1200 kg  
color: ..... Gris Oscuro  
potencia: ..... 120 HP  
año de patentamiento: 2012  
Precio: ..... 90.000 pesos

Ese mismo objeto puede cambiar su estado (observar que cambio el precio) en otro momento M1:

marca: ..... Toyota  
modelo: ..... Corolla

peso: ..... 1200 kg  
color: ..... Gris Oscuro  
potencia: ..... 120 HP  
año de patentamiento: 2012  
Precio:..... 93.000 pesos

El estado de un objeto es como una foto instantánea que captura el conjunto de características y sus valores en un momento dado.

**Comportamiento:** "*El comportamiento de un objeto está dado por el conjunto de acciones y reacciones que posee*".

Ya hemos indicado que un objeto es siempre instancia de una clase, por lo tanto, el conjunto de acciones y reacciones que el objeto posee son aquellas que previamente se han definido en la clase que le dio origen.

Si consideramos el objeto AUTO podríamos decir que su comportamiento esta dado por: arrancar, frenar, acelerar.

**Identidad:** "*La identidad de un objeto es aquella propiedad que permite distinguirlo de todos los demás*".

---

### 3.3. Relaciones entre objetos

---

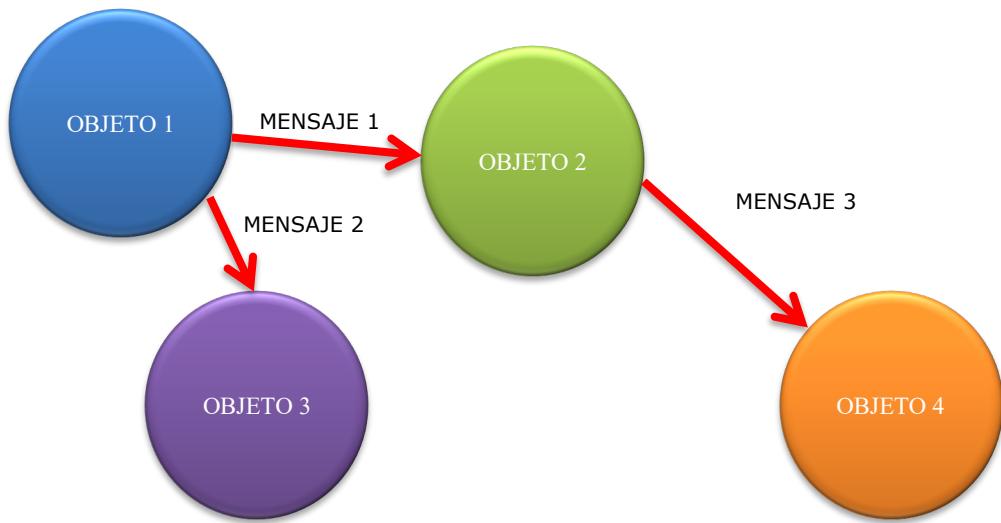
Entre los objetos se dan dos tipos de relaciones:

- **Enlace**
- **Agregación**

Estas son sus características:

**Enlace:** El enlace entre objetos es "*una conexión física o conceptual entre ellos*". El enlace entre objetos denota una relación de igual a igual.

Los objetos colaboran entre sí a través de los enlaces. Para lograr esto los objetos se envían **mensajes**. "Un mensaje es los que un objeto O1 le envía a otro objeto O2 para que O2 realice algo". La mensajería entre objetos es típicamente unidireccional, en casos muy peculiares puede ser bidireccional.



Cuando dos objetos se relacionan por medio del enlace cada uno adopta uno de los siguientes **roles**: actor, servidor o agente.

- **Actor:** "Un objeto adopta el **rol** de **actor** cuando puede enviarle mensajes a otros objetos pero los demás no pueden enviarle mensajes a él". También se lo conoce con el nombre de **objeto activo**.
- **Servidor:** "Un objeto adopta el **rol** de **servidor** cuando puede recibir mensajes de otros objetos, pero no pueden enviarle mensajes a ellos". También se lo conoce con el nombre de **objeto pasivo**.
- **Agente:** "Un objeto adopta el **rol** de **agente** cuando puede actuar como **actor** o **servidor**.

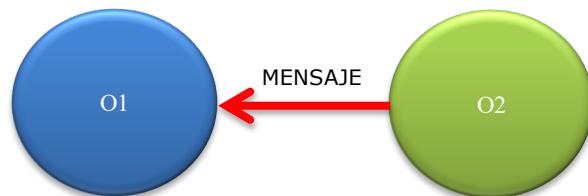
Se dice que un objeto O1 está **visible** a un objeto O2 cuando están participando de una relación de enlace y O2 le puede enviar un mensaje a O1.

Para que un objeto le pueda enviar un mensaje a otro en una relación de enlace estos deben estar **sincronizados**.

Existen tres formas de sincronización: **secuencial, vigilada y síncrona**.

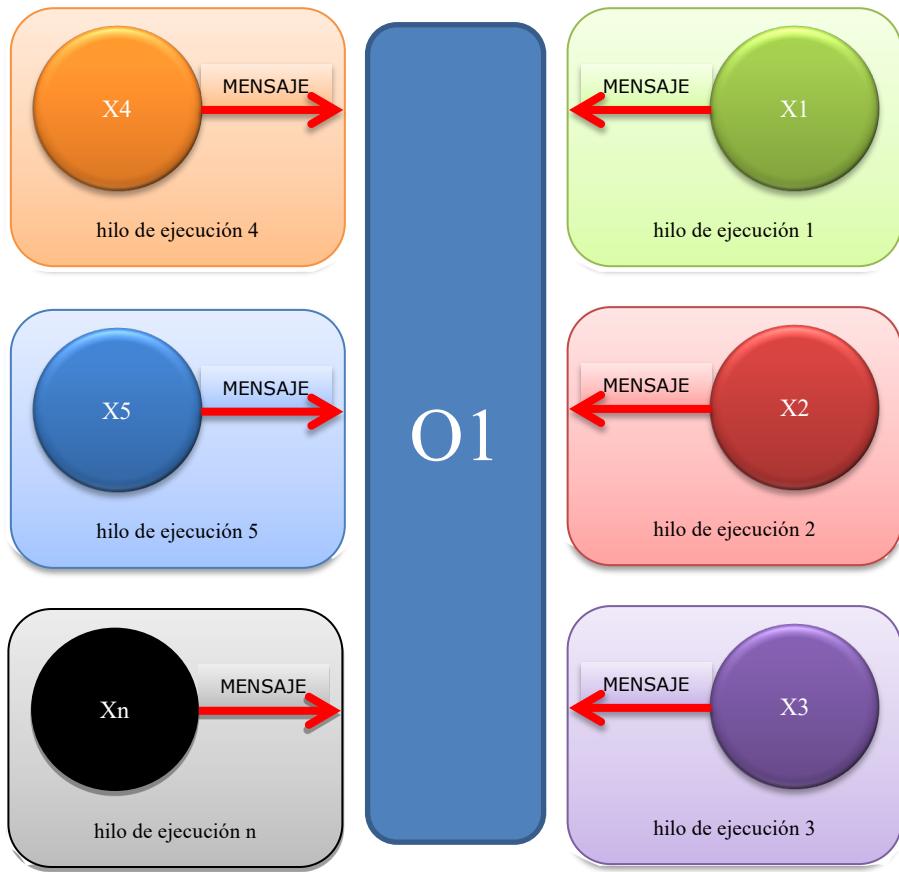
- **Secuencial:** "La semántica del **objeto pasivo** (servidor) está garantizada en presencia de un único **objeto activo** (cliente) simultáneamente".

Por ejemplo, dados dos objetos O1 y O2, siendo O1 el **objeto pasivo** (servidor) y O2 el **objeto activo** (cliente), decimos que el buen funcionamiento de O1 en la relación de enlace, queda garantizado por tener que atender a un solo objeto cliente a la vez, en este caso O2.



- **Vigilada:** "La semántica del **objeto pasivo** (servidor) está garantizada por la presencia de múltiples hilos de control, los **objetos activos** (clientes) colaboran para lograr la exclusión mutua".

Por ejemplo, dado el objeto O1 como **objeto pasivo** (servidor) y los objetos X1, X2, X3, X4, X5, ... Xn como **objetos activos** (clientes), decimos que el buen funcionamiento de O1 en la relación de enlace que mantiene con todos los clientes, queda garantizado por tener un hilo de ejecución para cada cliente X1, X2, X3, X4, X5, ... Xn. Los clientes colaboran en organizar quien posee prioridad para la atención.



- **Síncrona:** "La semántica del **objeto pasivo** (servidor) está garantizada por la presencia de múltiples hilos de control, el servidor es quien administra la exclusión mutua".

Por ejemplo, dado el objeto O1 como **objeto pasivo** (servidor) y los objetos X1, X2, X3, X4, X5, ... Xn como **objetos activos** (clientes), decimos que el buen funcionamiento de O1 en la relación de enlace que mantiene con todos los clientes, queda garantizado por tener un hilo de ejecución para cada cliente X1, X2, X3, X4, X5, ... Xn. El servidor es quien posee el conocimiento para establecer la prioridad en la atención.

**Agregación:** La **agregación** entre objetos denota una relación jerárquica del tipo "Todo-Parte".

En las relaciones de **agregación** existe un objeto que representa al "Todo" y uno o más objetos que representan a las "Partes".

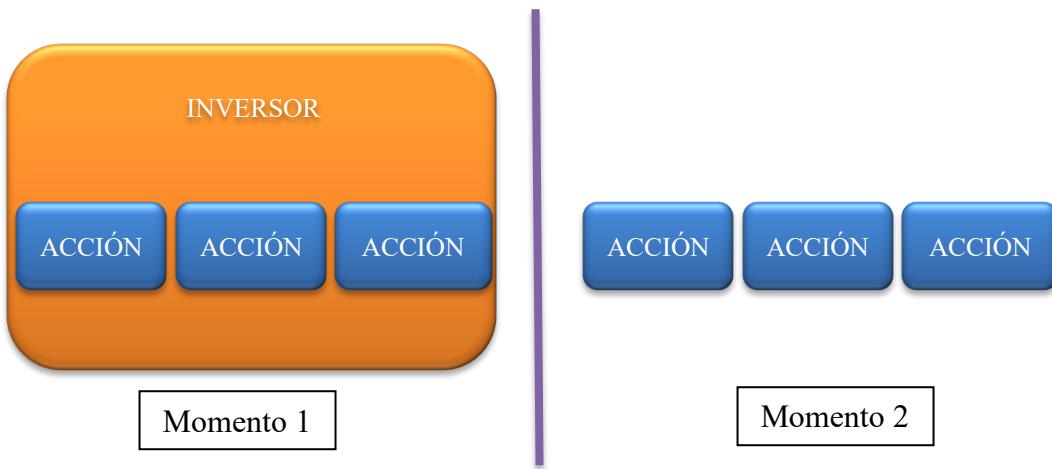


En el gráfico anterior se puede observar una **agregación**. "AVIÓN" representa al "Todo" y las "ALAS", las "TURBINAS" y el "TREN DE ATERRIZAJE" las "Partes". Esta forma peculiar de **agregación** se denomina "**agregación con contención física**".

Decimos que existe "**agregación con contención física**" cuando los ciclos de vida del "Todo" y las "Partes" están íntimamente relacionados y no tiene sentido la existencias de las "Partes" sino existe el "Todo". En la práctica decir que los ciclos de vida están íntimamente relacionados, generalmente se manifiesta de la siguiente forma: el objeto que representa al "Todo", al ser creado crea los objetos que representan las "Partes" y al ser eliminado se encarga de eliminar las "Partes" que creó.

También existe la agregación tradicional o "**agregación sin contención física**". En esta forma no encontramos dependencia en los ciclos de vida, o sea que las "Partes" tienen existencia independientemente de la existencia del "Todo". Un ejemplo significativo de ello podría ser la relación que

se establece entre un “INVERSOR” y las “ACCIONES” en las que invierte. Las acciones existen quizá desde antes que exista el inversor y seguirán existiendo con independencia de él. Un ejemplo de lo expuesto se puede observar en los gráficos siguientes.



---

### 3.4. Generalidades de las clases

---

“Una **clase** es el concepto o especificación que representa a un conjunto de objetos (reales) que comparten una estructura (conjunto de características) y comportamiento común”.

En una **clase** podemos identificar dos partes: la **interfaz** y la **implementación**.

- La **interfaz** de una **clase** denota su **visión externa** (lo que el resto del mundo ve de ella) enfatizando la abstracción y ocultando los detalles de su implementación interna.

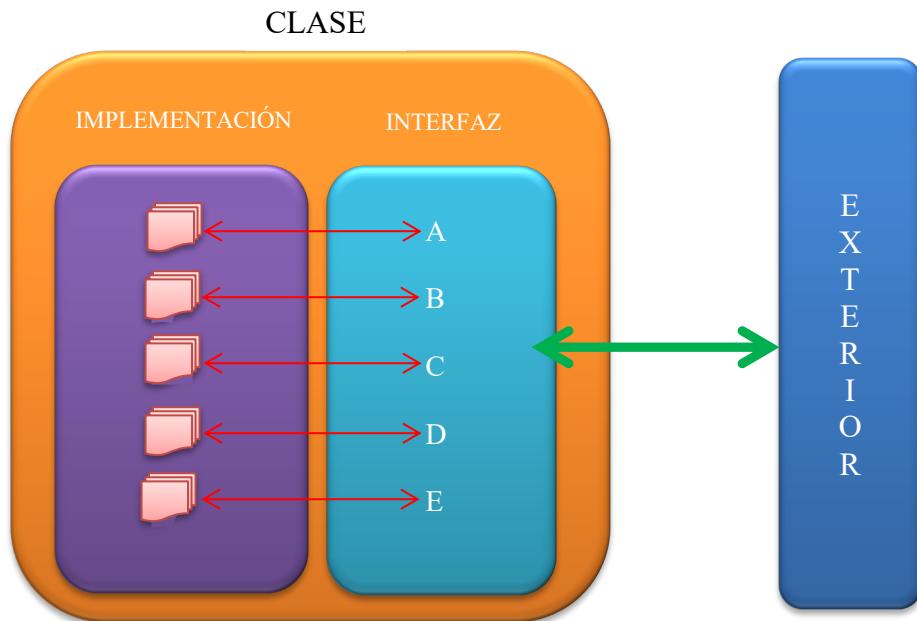
En general y teniendo en cuenta que cada lenguaje de programación establece su esquema de visibilidad para las **interfaces**, podemos distinguir tres formas soportadas por la mayoría:

Public/Pública: Todos los clientes pueden ver la **interfaz**.

Protected/Protegida: La **interfaz** es accesible por la propia clase, sus subclases y las clases amigas.

**Private/Privada:** La **interfaz** es accesible a si misma y las clases que la contienen.

- La **implementación** de una clase denota su **visión interna**. Está compuesta por la **implementación** de todas las operaciones expuestas en la **interfaz**.



### 3.5. Relaciones entre clases

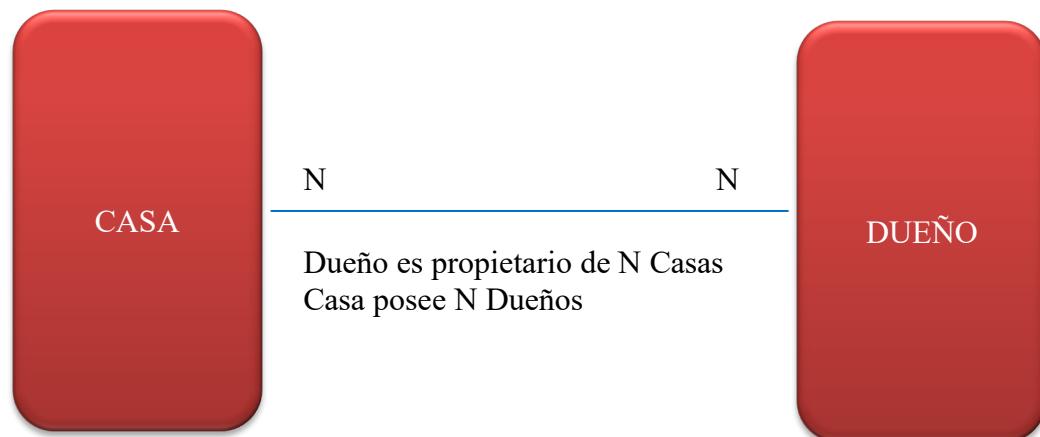
Las clases se relacionan de las siguientes maneras: **asociación, herencia, agregación, uso, instanciación** y **metaclases**.

Las dos últimas formas de relaciones no se detallarán en el presente texto debido a que carece de aplicación práctica en la tecnología que se utilizará más adelante en el presente curso.

- **Asociación:** "La **Asociación** es una relación bidireccional entre dos clases". Supongamos que tenemos las clases CASA y DUEÑO, existe una relación entre ellas dada por las semánticas "una CASA posee DUEÑOS" y "Un DUEÑO es propietario de CASAS". Esta

relación bidireccional permite conocer dada una CASA quienes son sus DUEÑOS y dado un DUEÑO que CASAS posee.

Como se puede observar claramente esta relación establece una dependencia semántica pero no la dirección de la misma. La Asociación posee **cardinalidad**. La **cardinalidad** establece cuantos elementos de un tipo se relacionan con elementos del otro tipo. De esta forma se obtienen tres tipos de cardinalidades: **uno a uno, uno a muchos y muchos a muchos**.



- **Herencia:** "La **herencia** es la relación por la cual una o más **subclases** pueden heredar (recibir) la estructura y el comportamiento de una o más **superclases**".

Existen dos tipos de **herencia**: **herencia simple** y **herencia múltiple**.

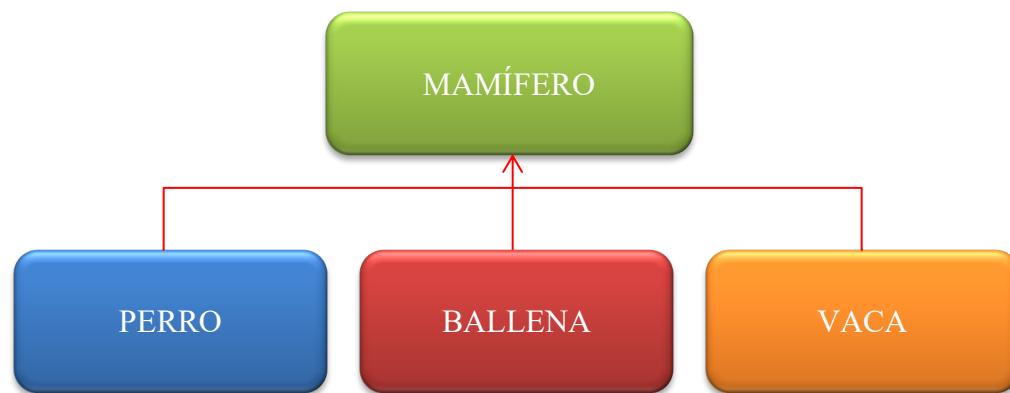
- La **herencia simple** se da cuando dos o más **subclases** heredan (reciben) la estructura y el comportamiento de una **superclases**.
- La **herencia múltiple** se da cuando dos o más **subclases** heredan (reciben) la estructura y el comportamiento de dos o más **superclases**.

La **herencia** se enmarca dentro de las **relaciones jerárquicas** del tipo "es-un". También se la conoce como relación de

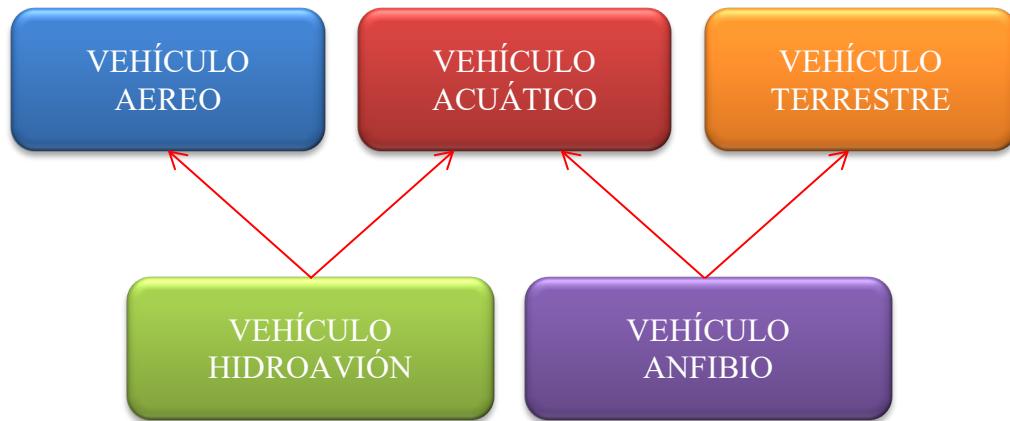
**“generalización-especialización”** donde la **superclase** es más **generalizada** que la **subclase**, o lo que es lo mismo, la **subclase** hereda todo lo de la **superclase** y luego se **especializa** agregando sus propias características y comportamientos.

Un ejemplo de **herencia simple** podría observarse si construimos una clase MAMÍFERO. Existen muchos tipos de mamíferos, perros, ballena, vaca, cada uno con características y comportamientos propios, más allá de compartir el hecho de ser mamíferos.

En este caso la superclase (generalización) es MAMÍFERO quien le heredará a las subclases (especialización) PERRO, BALLENA y VACA, quienes luego incorporarán sus características y comportamiento propios.



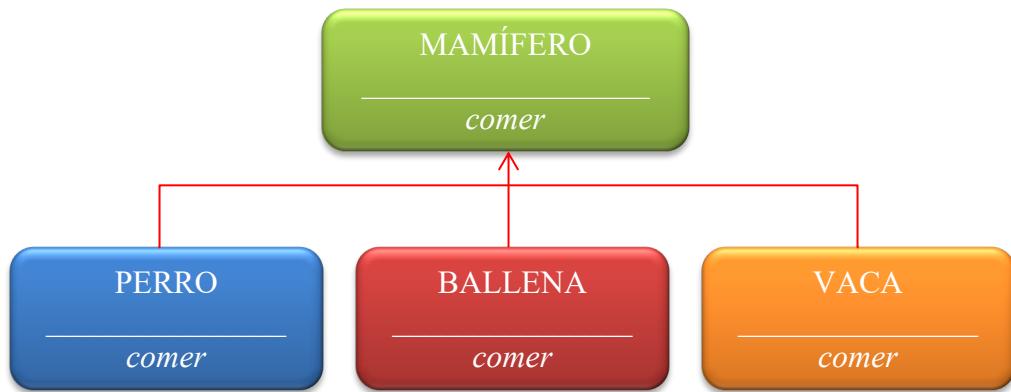
Un ejemplo de **herencia múltiple** podría observarse si construimos las superclases VEHÍCULO AEREO, VEHÍCULO TERRESTRE Y VEHÍCULO ACUÁTICO, luego hacemos que VEHÍCULO AEREO Y VEHÍCULO ACUÁTICO le hereden a la subclase HIDROAVIÓN y que VEHÍCULO TERRESTRE Y VEHÍCULO ACUÁTICO le hereden a la subclase VEHÍCULO ANFIBIO. De esta forma estaríamos en presencia de dos **herencias múltiples**.



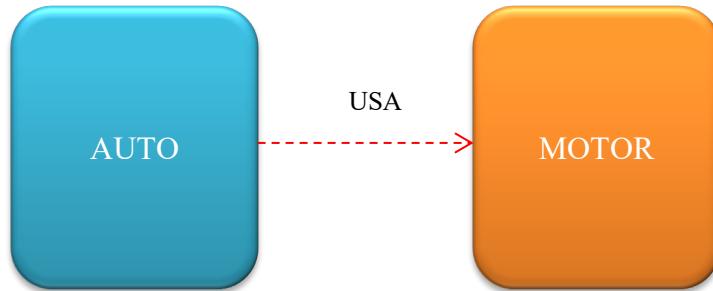
Cuando una clase hereda de otra además de recibir toda su estructura y comportamiento, también recibe su tipo debido a la relación "es-un". De esta forma podemos afirmar que un PERRO "es-un" MAMIFERO, lo que implica que cuando obtengamos un objeto producto de haber instanciado un PERRO, este podrá ser del tipo PERRO o del tipo MAMIFERO.

Se define a un método como **polimórfico**, cuando el método es heredado desde una superclase por dos o más subclases y cada subclase implementa un comportamiento distinto para él. El **polimorfismo** es un concepto que proviene de la teoría de tipos.

Considerando el ejemplo de **herencia simple** visto anteriormente, donde MAMÍFERO le hereda a PERRO, VACA Y BALLENA, suponiendo que le heredase la acción *comer*, cada sub clase debería implementar la acción *comer* (debería funcionar) de distinta manera debido a que un PERRO, una VACA y una BALLENA comen de distinta forma. Si lo expresado se da, estamos en presencia de el método polimórfico *comer*.



- **Agregación:** El concepto de **agregación** entre clases mantiene un paralelismo con el mismo concepto tratado para los objetos. Podemos afirmar que los objetos ponen de manifiesto las agregaciones que se han especificado en las clases.  
Ahora bien, ampliaremos este concepto al reconocer que se puede agregar de dos formas distintas.
  - La primera es una **agregación por valor** donde los ciclos de vida están relacionados y el todo es el encargado de crear y destruir las partes. Esta forma es muy utilizada cuando se da la **contención física**.
  - La segunda es una **agregación por referencia**, donde el todo está preparado para recibir referencias (direcciones de memoria) de donde se encuentran almacenadas las partes. Este tipo de agregación es más utilizado cuando **no existe contención física** en la relación.
- **Uso:** Se define como **relación de uso** a una asociación refinada donde se establece quien opera como cliente y quien opera como servidor. Anteriormente dijimos que una asociación era típicamente una relación bidireccional, si se transforma en una relación de uso será típicamente unidireccional.



### 3.6. Medida de la calidad de una abstracción

---

Una **abstracción** puede estar bien conceptualizada o no. Para determinar la calidad de esa **abstracción** nos basamos en cinco métricas:

- Acoplamiento
- Cohesión
- Suficiencia
- Compleción
- Ser Primitivo

Veamos cada criterio:

- **Acoplamiento:** Determina que tan interrelacionada está una abstracción respecto de otras. Es prudente mantener un **acoplamiento débil o bajo**.
- **Cohesión:** La *cohesión* mide el grado de conectividad y consistencia de los elementos colocados dentro de una abstracción. Es beneficioso que la *cohesión* sea mantenida lo más **fuerte o alta** posible.

- **Suficiencia:** Se entiende por *suficiencia* que la abstracción captura suficientes **características** y **comportamientos** para permitir una interacción significativa y eficaz.
- **Compleción:** Se entiende por *compleción* que la abstracción captura todas las **características** y **comportamientos** para permitir una interacción significativa y eficaz.

Los conceptos de **suficiencia** y **compleción** son complementarios. Mientras que la **suficiencia** plantea un criterio de mínima, la compleción plantea uno de máxima. La idea es que una abstracción debe poseer todo aquello que necesita, nada menos, pero no debe tener cosas que al momento de la definición no hacen falta.

Esto último, considerando que el modelo orientado a objetos prevé los mecanismos para incorporar paulatinamente partes estructurales y nuevos comportamientos a medida que sea demandado por el sistema.

- **Ser primitivo:** Este concepto refiere a las operaciones de la abstracción. Una operación es considerada **primitiva** "cuando su implementación es representada por la forma más simple posible de la misma, sin perder la naturaleza de lo que hace".

Por ejemplo, si pensamos en la operación *AgregarProducto* y esta agrega cinco productos simultáneamente, diremos que la misma no es primitiva, pues su representación más elemental sería agregar un producto. El problema fundamental con las acciones no primitivas es que en general le otorgan rigidez a la acción.

En nuestro ejemplo, si la acción fuera no primitiva y deseamos ingresar dos productos no podríamos, en lugar de ello si la acción fuera primitiva y cada vez que se la invoca agrega un producto, invocándola dos veces se solucionaría el problema.

#### 4. Clasificación

La **clasificación** es un método de ordenamiento para organizar las abstracciones. La clasificación de clases y objetos es un tema relacionado con

el diseño orientado a objetos, no obstante, realizaremos una pequeña introducción.

Clasificar objetos presenta el desafío de tener que ordenarlos, pudiendo confundirse objetos similares que corresponden a distintas categorías y generando resultados no deseados.

No existe una técnica estándar para clasificar e históricamente se han utilizado tres aproximaciones generales para esta tarea:

- **Categorización clásica**
- **Agrupamiento conceptual**
- **Teoría de prototipos**

Veamos cada una de estas aproximaciones:

- *La **categorización clásica** agrupa todas las entidades que poseen en común una determinada propiedad/característica o conjunto de ellas.*

Estas propiedades/características son condición suficiente para definir la categoría. Por ejemplo, los animales cuadrúpedos representan una categoría.

- *El **agrupamiento conceptual** es una variación de la categorización clásica donde es confuso detectar una característica pero es posible definir y/o describir un concepto.*

Grady Booch expone como ejemplo cuál sería la propiedad que permitiría agrupar las canciones de amor. Al intentar ubicarlas nos encontramos en un problema ya que ni el ritmo, ni la letra necesariamente, son propios y únicos en las canciones de amor. Entonces propone hacer una descripción conceptual de las mismas y clasificarlas considerando esta descripción.

- **Teoría de prototipos:** Esta forma de clasificar es propuesta cuando las entidades a tratar no cumplen con el requerimiento de poseer

características comunes (o al menos no son lo suficientemente claras), ni tampoco es fácil lograr una descripción conceptual consistente. Ante esta situación se genera un **prototipo** representativo y toda entidad que se le parece es considerada dentro de esa clasificación.

Si pensamos en clasificar la idea de juego, la misma es compleja. Habría que preguntarse ¿qué es un juego? Los juegos poseen propiedades que otras entidades que no son juegos también las tienen y conceptualmente es confuso definirlo ya que seguramente un juego de ingenio para adultos no será un juego para un niño.

En estas circunstancias, se aplica esta forma de clasificar. El **prototipo** pondrá de manifiesto además de lo que es un juego en sí, las propiedades de interacción, que permitan identificar aquellas entidades sustancialmente similares.

**Concepto de mecanismo:** Se denomina mecanismo a cualquier estructura donde se observa que los objetos colaboran entre sí para lograr un comportamiento que satisface un requerimiento a un problema. Los mecanismos representan patrones de comportamiento logrados por la colaboración de colecciones de objetos.

## 5. Introducción a las clases y los objetos

### Definición de una clase.

Una **clase** es una construcción que permite obtener y representar las representaciones que se necesitan en una solución orientada a objetos. Ellas podrán contener **campos, propiedades, métodos y eventos**.

```
public class Cliente  
{  
}  
}
```

Ej00001

### Creación de un objeto.

```
1 referencia  
private void Form1_Load(object sender, EventArgs e)  
{  
    Cliente object1 = new Cliente();  
}  
}  
2 referencias  
public class Cliente  
{  
}  
}
```

Ej0002

### Campos y Constantes.

Un **campo** es una variable de cualquier tipo que se declara directamente en una clase. Los campos son miembros de su tipo contenedor. En estas variables se pueden almacenar valores.

```
0 referencias  
public class Cliente  
{  
    private string Nombre;  
    private byte Edad;  
}
```

Ej0003

Las **constants** son valores definidos que no pueden alterarse. Se conocen en tiempo de compilación y no cambian durante la vida del programa. Las constantes se declaran con el modificador **const**.

```
public class Datos
{
    private const int X = 0;
    private const string Lenguaje = "C#";
}
```

Ej0004

## Modificadores de Acceso.

Los **modificadores de acceso** definen que visibilidad tendran las clases y los miembros que ellas poseen. Tenemos cuatro modificadores básicos que son: **public**, **private**, **internal**, **protected**. Algunos de ellos se pueden combinar. Por ejemplo **protected internal** o **private protected**. El alcance de cada uno de ellos se puede ver en la tabla siguiente. Este tema se verá más profundamente en la Unidad 2 cuando se analicen los distintos tipos de clases.

<b>public</b>	El acceso no está restringido.
<b>protected</b>	El acceso está limitado a la clase contenedora o a los tipos derivados de la clase contenedora.
<b>internal</b>	El acceso está limitado al ensamblado actual.
<b>protected internal</b>	El acceso está limitado al ensamblado actual o a los tipos derivados de la clase contenedora.
<b>private</b>	El acceso está limitado al tipo contenedor.
<b>private protected</b>	El acceso está limitado a la clase contenedora o a los tipos derivados de la clase contenedora que hay en el ensamblado actual. Disponible desde la versión C# 7.2.

Fuente: Microsoft

- NOTA: TODO EL CÓDIGO QUE SE UTILIZA EN LAS EXPLICACIONES LO PUEDE BAJAR DEL **MÓDULO RECURSOS Y BIBLIOGRAFÍA**.

# **MATERIAL DE REPASO CONCEPTUAL**

## **CONCEPTOS SOBRE CLASES**

# PROGRAMACIÓN ORIENTADA A OBJETOS



## Clases

Titular: Dario Guillermo Cardacci

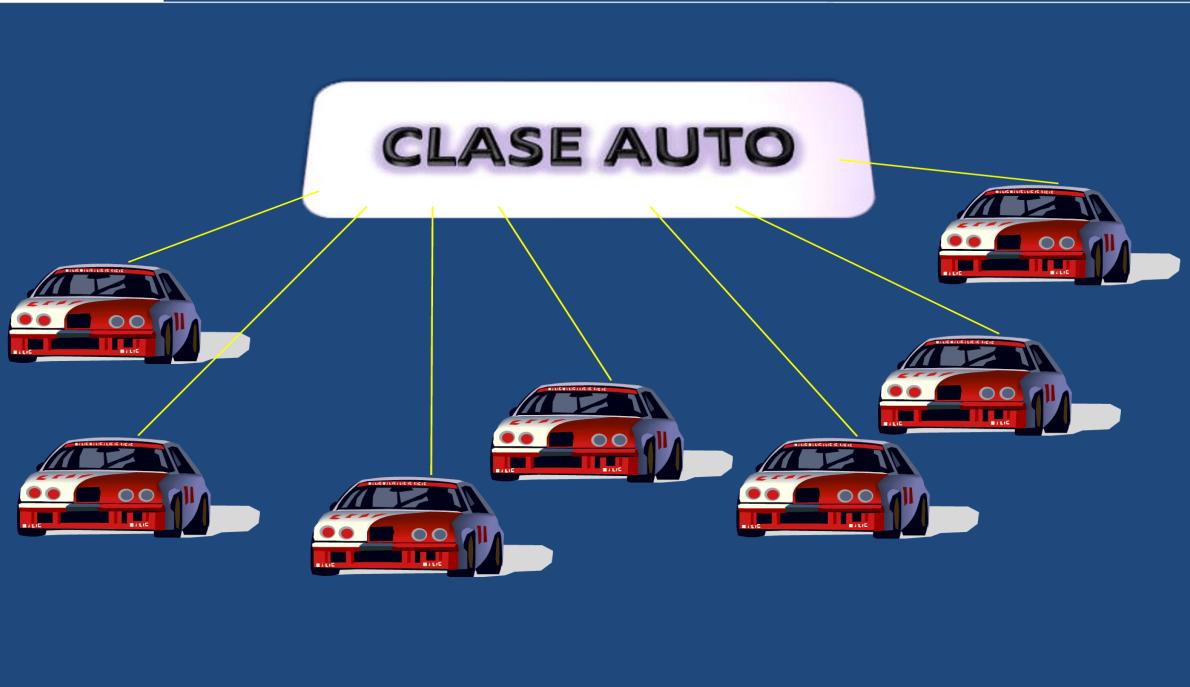


## PROGRAMACIÓN ORIENTADA A OBJETOS

Titular: Dario G. Cardacci

### ¿Qué es una clase?

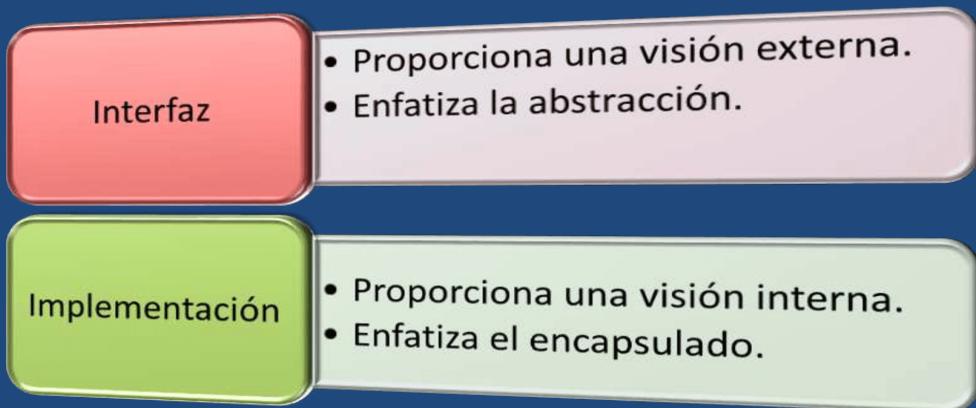
- La especificación en términos de atributos y comportamientos de un conjunto de objetos reales o abstractos que comparten una estructura y comportamiento común.
- Según Booch: “Una clase es un conjunto de objetos que comparten una estructura común y un comportamiento común”



## Una clase posee



## Una clase posee



- La implementación de una clase se compone principalmente de la implementación de las operaciones definidas en su interfaz.
- No obstante encontramos formas características en la forma que la interfaz expone sus componentes:
  - **Pública (Public):** Una declaración accesible a todos los clientes.
  - **Protegida (Protected):** Una declaración accesible a la propia clase, sus subclases y sus clases amigas (friends).
  - **Privada (Private):** Una declaración accesible solo a la propia clase y sus clases amigas (friends).

## Tipos de relaciones entre clases

- **Tipos Básicos:**

- Generalización – Especialización. (Es un).
- Todo – Parte. (Parte de).
- Asociación.

- **Tipos Derivados:**

- Herencia.
- Agregación.
- Uso.
- Instanciación.
- Metaclase.

## Generalización – Especialización. (Es un).

- Denota una jerarquía.
- Cuanto más generalizamos más abstracto es el concepto definido.
- Cuanto más especializamos menos abstracto es el concepto definido.



## Generalización – Especialización. (Es un).



## Todo – Parte (Parte de)

TODO

PARTES



## Asociación

- Cuando se da una relación de asociación se establece una relación que denota una dependencia semántica entre las clases que de otro modo serían independientes.

RELACIÓN SEMÁNTICA ENTRE LAS CLASES

VELAS

SILLAS

MESAS

## Asociación

- La asociación solo denota una relación semántica y en principio no establece la dirección de esta dependencia.
- Si no se establece lo contrario la relación se asume bidireccional.
- La cardinalidad establece la multiplicidad con que se relacionan las clases en la asociación:

## TIPOS DE CARDINALIDAD

Uno a uno

Uno a muchos

Muchos a muchos

## Herencia

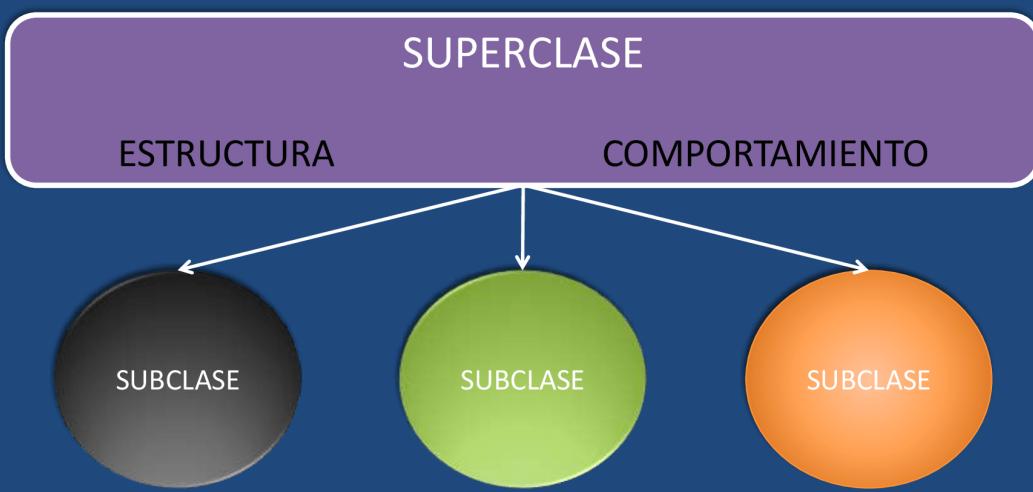
- Es un tipo de relación jerárquica donde una o varias subclases pueden recibir (heredar) la estructura y el comportamiento de otra clase normalmente denominada Superclase.

### TIPOS DE HERENCIA

- SIMPLE
- MÚLTIPLE

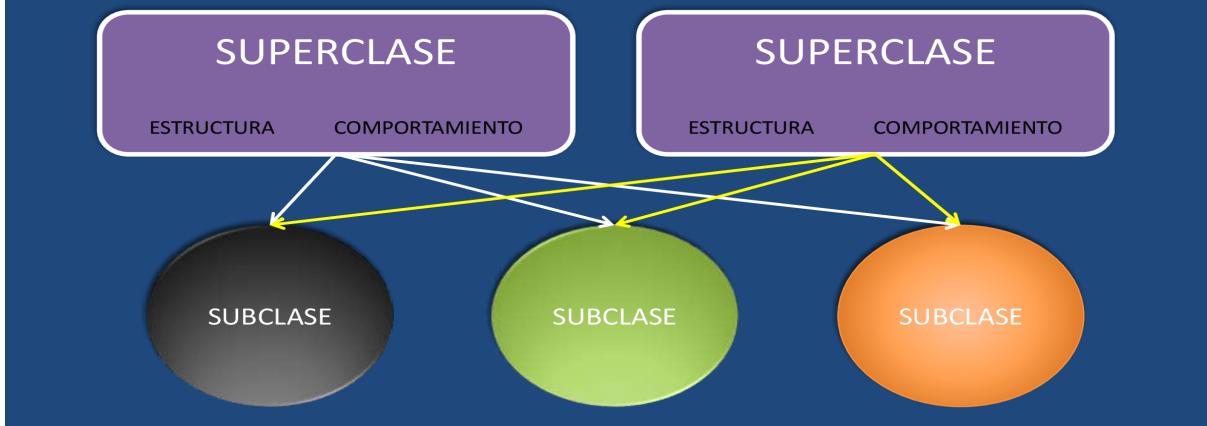
## Herencia Simple

- Se produce herencia simple cuando una o más subclases reciben la estructura y el comportamiento de una sola superclase.



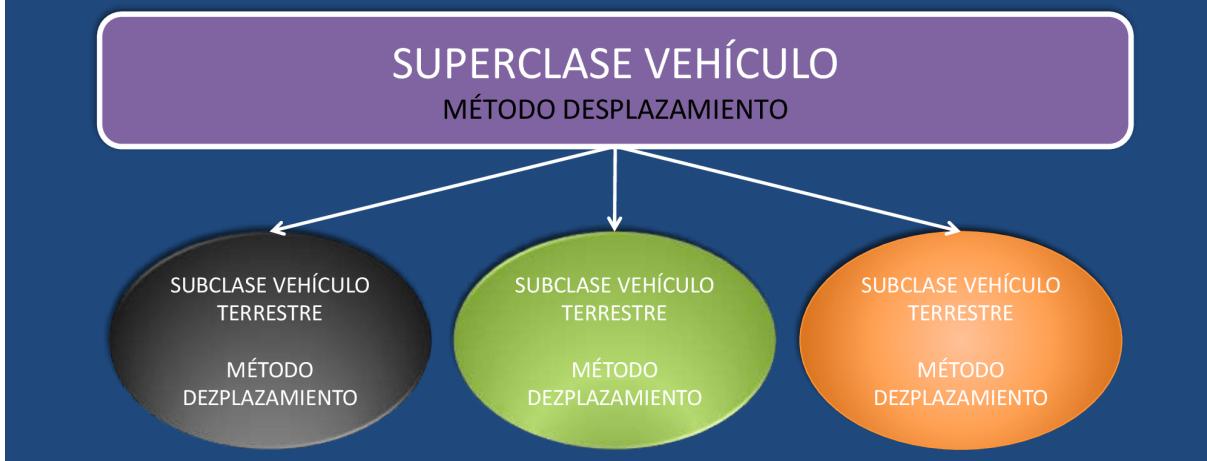
## Herencia Múltiple

- Se produce herencia múltiple cuando una o más subclases reciben la estructura y el comportamiento de más de una superclase.



## Herencia Polimorfismo

- Se produce polimorfismo en un comportamiento (método) cuando dos o más subclases lo heredan y cada una de ellas lo implementan de distinta forma.



## Agregación

- La relaciones de agregación entre clases poseen un paralelismo con las relaciones de agregación entre objetos.
- Es una relación del tipo todo-parte.
- Posee la capacidad de ir desde el “todo” a sus “partes”.
- Es una relación jerárquica.

## Agregación



## Agregación

- En las relaciones de agregación existe una particularidad o tipo especial de relación denominada “Agregación con Contención Física”.
- La **contención física** implica que el todo contiene a las partes físicamente. Por ejemplo podemos pensar en una clase “Cafetera” que contiene físicamente a la clase “Calentador”.
- El caso contrario a la **contención física** podría graficarse suponiendo a las clases “Accionista” y “Acción”, donde suponiendo que no existe Accionista sin Acción, “Accionista” representa al todo, y “Acción” a las partes. A pesar de la relación de agregación, los ciclos de vida de los objeto de estas clases pueden ser completamente distintos.

## Agregación

- En la práctica la **contención Física** puede manifestarse de dos formas:
- **Contención física por valor**, implica que los ciclos de vida del objeto que agrega con sus agregados está íntimamente relacionado. El objeto que agrega crea a los agregados y él mismo se encarga de matarlos.
- **Contención física por referencia**, implica que los ciclos de vida del objeto que agrega con sus agregados no está necesariamente relacionado.

## Uso

- Una relación de “**Uso**” puede ser percibida como una relación de “Asociación” refinada donde se establece que abstracción es el cliente y que abstracción es el servidor que provee los servicios.
- En la práctica una relación de “**Uso**” se manifiesta debido a que se declara en la implementación de alguna operación un objeto local de la clase usada.

## Metaclases

- Una metaclase son clases cuyas instancias son clases.
- Esto es soportado por lenguajes como Smalltalk y CLOS.
- Se suelen utilizar con el propósito de mantener variables de clases (compartidas por todas las instancias de la clase).

## Medida de la calidad de una Abstracción

- Acoplamiento.
- Cohesión.
- Suficiencia.
- Compleción.
- Ser Primitivo.

## Medida de la calidad de una Abstracción

### Acoplamiento

“Medida de la fuerza de la asociación establecida por una conexión entre una clase y otra o un objeto y otro”

## Medida de la calidad de una Abstracción

### Cohesión

“Medida de grado de conectividad interna entre los elementos de una clase u objeto”

## Medida de la calidad de una Abstracción

### Suficiencia

“Establece si la clase captura las características importantes de la abstracción como para permitir una interacción significativa y eficiente”

## Medida de la calidad de una Abstracción

### Compleción

“Establece si la interfaz de la clase captura las características significativas de la abstracción”

## Medida de la calidad de una Abstracción

### Ser Primitivo

“Se refiere a las operaciones primitivas. Una operación es inequívocamente primitiva si se puede implantar sólo accediendo a la representación interna”

## Aspectos a considerar para situar un comportamiento en una clase

- Reutilización.      ¿Sería este comportamiento más útil en más de un contexto?
- Complejidad.      ¿Qué grado de dificultad plantea implementar este tipo de comportamiento?
- Aplicabilidad.      ¿Qué relevancia tiene este comportamiento para el tipo en que podría ubicarse?
- Conocimiento de la Implementación.      ¿Depende la implementación del comportamiento de los detalles internos de un cierto tipo?

## Clasificación

- Categorización Clásica.

Todas las entidades poseen una determinada propiedad o conjunto de propiedades en común.

- Agrupamiento Conceptual.

Se establece una descripción conceptual de los elementos a clasificar.

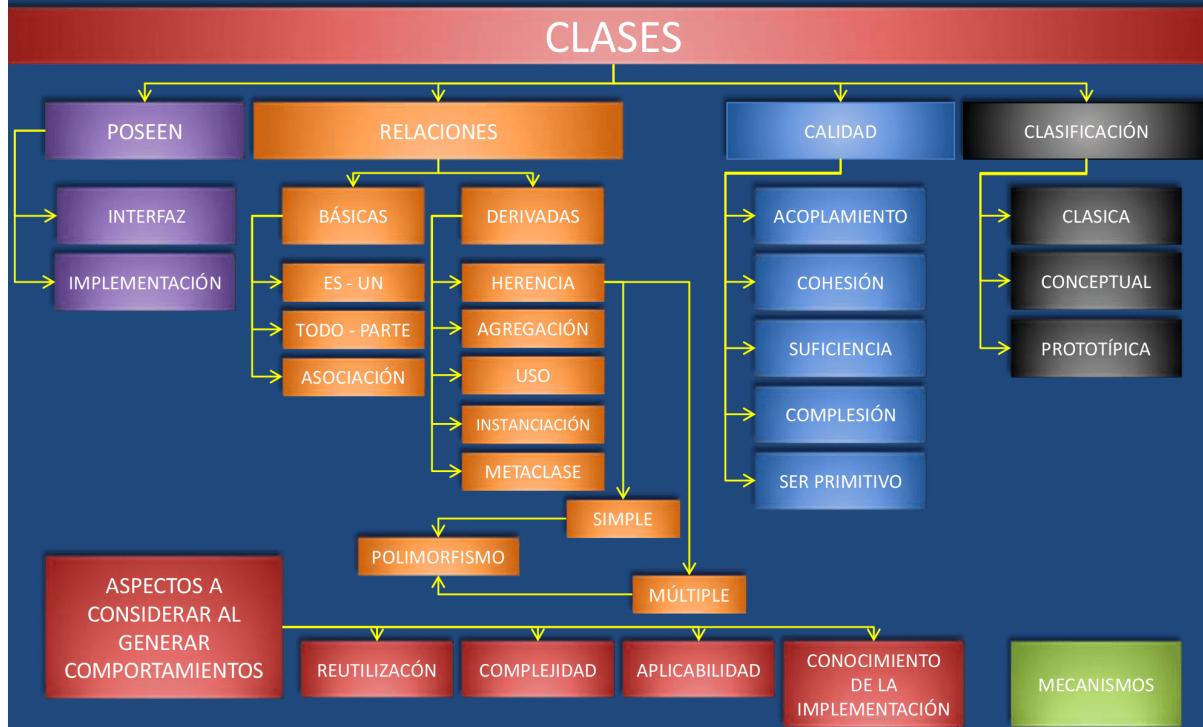
- Teoría de Prototipos.

Se establece un elemento prototípico y se considera un objeto como un miembro de esta clase si y solo si se parece a este prototipo de forma significativa.

# Mecanismo

- Se denomina “Mecanismo” a los medios por los cuales los objetos colaboran para proporcionar algún mecanismo de nivel superior.

# Resumen



**FIN**

## **CONCEPTOS SOBRE OBJETOS**

## PROGRAMACIÓN ORIENTADA A OBJETOS



# Objetos

Titular: Dario Guillermo Cardacci



## PROGRAMACIÓN ORIENTADA A OBJETOS

Titular: Dario G. Cardacci

### ¿Qué es un objeto?

- Una cosa tangible y/o visible
  - Algo que puede comprenderse intelectualmente.
  - Algo hacia lo que se dirige un pensamiento o acción.
- “Un objeto tiene estado, comportamiento e identidad; la estructura y comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables”

## ¿Qué posee un objeto?

- Estado.
- Comportamiento.
- Identidad.

## Estado

- El estado de un objeto abarca todas las propiedades (normalmente estáticas) del mismo más los valores actuales (normalmente dinámicos) de cada una de esas propiedades.

## Comportamiento

- El comportamiento es como actúa y reacciona un objeto, en términos de sus cambios de estado y paso de mensajes.

## Identidad

- La identidad es aquella propiedad de un objeto que lo distingue de todos los demás objetos.

## Tipos de relaciones

- Enlaces.
- Agregación.

## Tipos de relaciones

### Enlace

- Es una relación física o conceptual entre objetos.
- Relación de igual a igual.

## Tipos de relaciones

### Enlace - Roles

- Actor
- Servidor
- Agente

## Tipos de relaciones

### Enlace – Roles

#### Actor

- Un objeto que puede operar sobre otros objetos pero nunca se opera sobre él por parte de otros objetos; en algunos contextos, los términos objeto activo y actor son equivalentes.

## Tipos de relaciones

Enlace – Roles

Servidor

- Un objeto que nunca opera sobre otros objetos; solo otros objetos operan sobre él.

## Tipos de relaciones

Enlace – Roles

Agente

- Un objeto que puede operar sobre otros objetos y además otros objetos pueden operar sobre él; un agente se crea normalmente para realizar algún trabajo en nombre de un actor u otro agente.

## Tipos de relaciones

### Enlace - Sincronización

- Secuencial
- Vigilado
- Síncrono

## Tipos de relaciones

### Enlace – Sincronización

#### Secuencial

- La semántica del objeto pasivo está garantizada sólo en presencia de un único objeto activo simultáneamente.

## Tipos de relaciones

### Enlace – Sincronización Vigilado

- La semántica del objeto pasivo está garantizada sólo en presencia de múltiples hilos de control, pero los clientes activos deben colaborar para lograr la exclusión mutua.

## Tipos de relaciones

### Enlace – Sincronización Síncrono

- La semántica del objeto pasivo está garantizada en presencia de múltiples hilos de control, y el servidor garantiza la exclusión mutua.

## Tipos de relaciones

### Agregación

- Es una relación del tipo todo-parte.
- Posee la capacidad de ir desde el “todo” a sus “partes”.
- Es una relación jerárquica.

## Resumen





FIN

---

## **Actividades asincrónicas**

### Guía de preguntas de repaso conceptual

31. ¿Qué se debe definir para cualquier operación asociada a un objeto?
32. ¿Qué es una precondición?
33. ¿Qué es una postcondición?
34. ¿A qué se denomina excepción?
35. ¿A qué se denomina mensaje?
36. ¿El encapsulado es un concepto complementario a la abstracción? Justifique.
37. ¿Cómo se denomina al elemento de un objeto que captura su vista externa?
38. ¿Cómo se denomina al elemento de un objeto que captura su vista interna la cual incluye los mecanismos que consiguen el comportamiento deseado?
39. ¿El concepto de “ocultar los detalles de implementación” lo asociaría a “esconder los detalles de implementación” o a “evitar el uso inapropiado de los detalles de implementación”? Justifique.
40. ¿Cuáles son los dos aspectos que hacen importante considerar a la modularidad?
41. ¿Para qué se utiliza la jerarquía?
42. ¿Cómo denominamos a la caracterización precisa de propiedades estructurales o de comportamiento que comparten una serie de entidades?
43. ¿Las clases implementan tipos?
44. ¿Los tipos implementan clases?
45. ¿Cómo denominamos a los lenguajes que hacen una comprobación de tipos estricta?
46. ¿Cómo denominamos a los lenguajes que no hacen una comprobación de tipos estricta?
47. ¿A qué se denomina ligadura estática (temprana)?
48. ¿A qué se denomina ligadura dinámica (tardía)?

49. ¿Es lo mismo la comprobación estricta de tipos y la ligadura dinámica?
50. ¿Cómo se denomina la característica que permite a diferentes objetos actuar al mismo tiempo?
51. ¿A qué se denomina concurrencia pesada?
52. ¿A qué se denomina concurrencia ligera o liviana?
53. ¿La concurrencia es la propiedad que distingue un objeto activo de uno que no lo está?
54. ¿Cómo se denomina la característica en orientación a objetos que permite conservar el estado de un objeto en el tiempo y el espacio?
55. ¿Qué cosas se persisten?
56. Defina qué es un objeto desde la perspectiva de la cognición humana.
57. ¿Un objeto es real o abstracto? Justifique.
58. ¿Los objetos poseen límites físicos precisos o imprecisos?
59. ¿Cuáles son las tres cosas que debe tener un objeto?
60. ¿Cuál es la palabra que se puede utilizar como sinónimo de objeto?
61. ¿Cuál es la palabra que se puede utilizar como sinónimo de instancia?
62. ¿Cómo definiría el estado de un objeto?
63. ¿A qué definimos propiedad de un objeto?
64. ¿Qué es lo que distingue a “un objeto” de los “valores simples”?
65. ¿Cómo definiría el comportamiento de un objeto?
66. ¿El comportamiento de un objeto se ve afectado por el estado del mismo o bien que el comportamiento del objeto es función de su estado?
67. ¿Algunos comportamientos pueden alterar el estado de un objeto?
68. Se puede afirmar que el estado de un objeto termina siendo los resultados acumulados de su comportamiento.
69. ¿A qué definiría como operación (método/función miembro)?

70. ¿Cuáles son las tres operaciones más comunes?
71. ¿Cuáles son las dos operaciones habituales que se utilizan para crear y destruir instancias de clases?
72. ¿Qué tipo de operación es el modificador?
73. ¿Qué tipo de operación es el selector?
74. ¿Qué tipo de operación es el iterador?
75. ¿Qué tipo de operación es el constructor?
76. ¿Qué tipo de operación es el destructor?
77. ¿Cómo denominamos operaciones fuera de las clases en aquellos programas orientados a objetos que permiten colocarlas (ej. C++)?
78. ¿Todos los métodos son operaciones?
79. ¿Todas las operaciones son métodos?
80. Dado el protocolo de un objeto (todos sus métodos y subprogramas libres asociados al objeto si el lenguaje lo permite) es conveniente dividirlo en grupos lógicos más pequeños de comportamiento? ¿Por qué?
81. ¿Cómo denominamos a los grupos lógicos más pequeños de comportamiento del protocolo total de un objeto?
82. ¿Cuáles son las dos responsabilidades más importantes que posee un objeto?
83. ¿Es relevante el orden en que se invocan las operaciones de un objeto?
84. ¿Por qué decimos que los objetos se pueden considerar como máquinas?
85. ¿Qué es la identidad de un objeto?
86. Dadas dos variables X e Y del mismo tipo ¿qué significa que ambas son iguales?
87. Dadas dos variables X e Y del mismo tipo ¿qué significa asignarle Y a X?
88. Dadas dos variables X e Y del mismo tipo ¿qué significa clonar X en Y?
89. ¿Qué significa realizar una clonación superficial?
90. ¿Qué significa realizar una clonación profunda?

91. ¿Qué es el ciclo de vida de un objeto?
92. ¿Cómo se libera el espacio ocupado por un objeto?
93. ¿Qué tipos de relaciones existen entre los objetos?
94. ¿Cómo podemos definir al enlace entre objetos?
95. ¿Cómo pueden ser los mensajes entre dos objetos en una relación de enlace?
96. ¿Qué es un mensaje unidireccional?
97. ¿Qué es un mensaje bidireccional?
98. ¿Quién inicia el paso de un mensaje entre dos objetos en una relación de enlace?
99. ¿Cuáles son los roles o papeles que puede desempeñar un objeto en una relación de enlace?
100. ¿Qué significa que un objeto actúe como “Actor”?
101. ¿Qué significa que un objeto actúe como “Servidor”?
102. ¿Qué significa que un objeto actúe como “Agente”?
103. Dados dos objetos A y B, si A le puede enviar un mensaje a B, estando ambos relacionados por enlace, decimos que B respecto de A está: .....
104. ¿Cuáles son las cuatro formas de visibilidad que puede poseer un objeto servidor respecto de un objeto cliente?
105. En una relación de enlace de dos objetos, cuando uno le pasa un mensaje al otro, además de adoptar roles ambos deben estar:.....
106. ¿Cuáles son las posibles formas de sincronización?
107. ¿Qué significa que dados dos objetos A y B estos están secuencialmente sincronizados?
108. ¿Qué significa que la forma de sincronizarse de un conjunto de objetos es vigilada?
109. ¿Qué significa que la forma de sincronizarse de un conjunto de objetos es síncrona?
110. ¿El enlace es una relación de igual a igual o jerárquica?

111. ¿La agregación es una relación de igual a igual o jerárquica?
112. ¿Qué tipo de jerarquía denota la agregación?
113. ¿Qué otros nombres recibe el “todo” en una relación de agregación?
114. ¿En una relación de agregación las “partes” forman parte del estado del “todo”?
115. ¿Qué tipos de agregación existen?
116. ¿Qué caracteriza a la agregación con contención física?
117. ¿Qué es una clase?
118. ¿La interfaz de la clase proporciona su visión interna?
119. ¿La implementación de la clase proporciona su visión externa?
120. ¿En cuántas partes la podemos dividir una interfaz en términos de la accesibilidad o visibilidad que posee?
121. ¿Qué tipos básicos de relaciones existen entre las clases?
122. ¿Qué relaciones entre clases se desprenden de las tres relaciones básicas?
123. ¿La asociación denota una dependencia semántica y la dirección de esta asociación?
124. ¿Qué significa la cardinalidad en una relación?
125. ¿Qué cardinalidad puede existir entre clases relacionadas por asociación?
126. ¿Qué es la herencia?
127. ¿Cuántos tipos de herencia existen?
128. ¿A qué se denomina herencia simple?
129. ¿A qué se denomina herencia múltiple?
130. ¿Cómo se denomina a la clase que no se espera tener instancias de ella y solo se utilizará para heredar a otras clases?
131. ¿Cómo se denomina a la clase que se espera tener instancias de ella y puede utilizarse para heredar a otras clases o no?

132. ¿Cómo se denomina al método de una clase abstracta que no posee implementación y fue pensado para que sea implementado en las sub clases que lo heredan?
133. ¿Cómo se denomina a la clase más generalizada en una estructura de clases?
134. ¿Qué es el polimorfismo?
135. ¿Cómo se denomina cuando una clase posee métodos que comparten el nombre y se diferencian por su firma?
136. ¿Qué sentencias de código se evitan utilizar cuando se aplica correctamente el polimorfismo?
137. ¿Qué es la agregación como relación entre clases?
138. ¿Qué formas de contención física existen en la agregación?
139. ¿Qué características posee la contención física por valor?
140. ¿Qué características posee la contención física por referencia?
141. ¿Qué es una relación de uso?
142. ¿Qué es la instanciación?
143. ¿Todo objeto es una instancia de una clase?
144. ¿Qué es una metaclasa?
145. ¿Qué métricas hay que observar para determinar la calidad de una abstracción?
146. ¿Qué es el acoplamiento?
147. ¿Qué es la cohesión?
148. ¿Qué es la suficiencia?
149. ¿Qué es la compleción?
150. ¿Qué significa ser primitivo?
151. ¿Qué se debe observar al momento de decidir si una abstracción debe implementar un determinado comportamiento o no?
152. ¿Qué formas puede adoptar el paso de un mensaje?

153. ¿Qué características posee un mensaje síncrono?
154. ¿Qué características posee un mensaje de abandono inmediato?
155. ¿Qué características posee un mensaje de intervalo?
156. ¿Qué características posee un mensaje Asíncrono?
157. ¿Qué significa que una abstracción está accesible a otra?
158. ¿Qué expresa la ley de Demeter?
159. ¿Cuál es la consecuencia inmediata de aplicar la ley de Demeter?
160. ¿Cuáles son las cuatro formas fundamentales por las cuales un objeto X puede hacerse visible a un objeto Y?
161. ¿Para qué sirve clasificar a los objetos?
162. ¿Por qué es tan difícil la clasificación de objetos?
163. ¿Cómo es el rol del observador en la clasificación de objetos?
164. ¿Cuáles son las aproximaciones generales a la clasificación?
165. ¿Qué es la categorización clásica?
166. ¿Qué es el agrupamiento conceptual?
167. ¿Qué es la teoría de prototipos?
168. ¿Qué es una abstracción clave?
169. ¿Qué son los mecanismos?