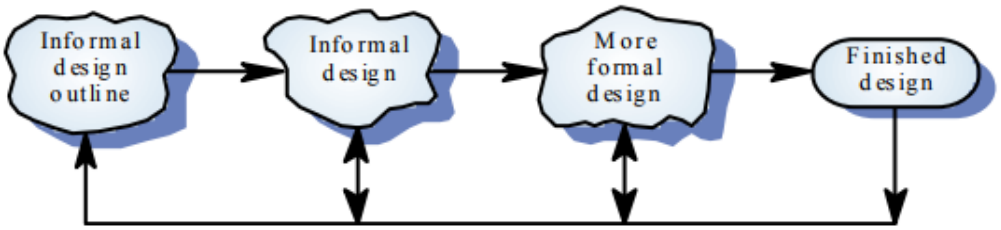
Diseño de Software es la acción de construir soluciones que satisfagan los requerimientos del cliente. Existen varias etapas en el proceso de diseño de software, a saber son:

- Entendimiento del problema
- Identificar una o más soluciones
- Describir abstracciones de la solución
- Repetir el proceso para cada abstracción identificada hasta que el diseño esté expresado en términos sencillos

Cualquier diseño debe ser modelado como una gráfica dirigida hecha de entidades con atributos los cuales participan en relaciones. El sistema debe estar descrito a distintos niveles de abstracción y el diseño ocurre en etapas que se traslapan. La primera idea que se tiene al construir una solución de un determinado problema es un modelo mental que constituye el primer intento de diseño llamado comúnmente diseño informal. Este diseño a medida que se va describiendo en papel utilizando técnicas y procedimientos esquemáticos y metódicos va adquiriendo forma hasta constituirse en un diseño formal equivalente. La siguiente figura ejemplifica este hecho:



Diseño Orientado a Objetos

Dentro del paradigma de la orientación a objetos, el diseño OO es más complejo que el diseño estructurado clásico, ya que lo que se busca es crear un diseño genérico y abierto y no cerrado y concreto. El Diseño Orientado a Objetos se define como un diseño de sistemas que utiliza objetos auto-contenidos y clases de objetos.

La naturaleza única del diseño orientado al objeto queda reflejada en su capacidad de construir sobre tres pilares conceptuales importantes del diseño de software:

Abstracción: Es un proceso mental por el que se evitan los detalles para centrarse en las cosas más genéricas de manera que se facilite su comprensión.

Modularidad: La modularidad es la manifestación más común de la división de problemas. El software se divide en componentes con nombres distintos y abordables por separado, en ocasiones llamados módulos, que se integran para satisfacer los requerimientos del problema.

Ocultamiento de información: El concepto de modularidad lleva a una pregunta fundamental:
¿Cómo descomponer una solución de software para obtener el mejor conjunto de módulos?
El principio del ocultamiento de información sugiere que los módulos se

caracterizan por decisiones de diseño que se ocultan (cada una) de las demás. En otras palabras, deben especificarse y diseñar módulos, de forma que la información (algoritmos y datos) contenida en un módulo sea inaccesible para los que no necesiten de ella.

Características principales del Diseño Orientado a Objetos

- Los objetos son abstracciones del mundo real o entidades del sistema que se administran entre ellas mismas.
- Los objetos son independientes y encapsulan el estado y la representación de información.
- La funcionalidad del sistema se expresa en términos de servicios de los objetos.
- Las áreas de datos compartidas son eliminadas.
- Los objetos se comunican mediante paso de parámetros
- Los objetos pueden estar distribuidos y pueden ejecutarse en forma secuencial o en paralelo.

Ventajas del Diseño Orientado a Objetos

- Fácil de mantener, los objetos representan entidades auto-contenidas.
- Los objetos son componentes reutilizables.
- Para algunos sistemas, puede haber un mapeo obvio entre las entidades del mundo real y los objetos del sistema.

Componentes del Diseño Orientado a Objetos

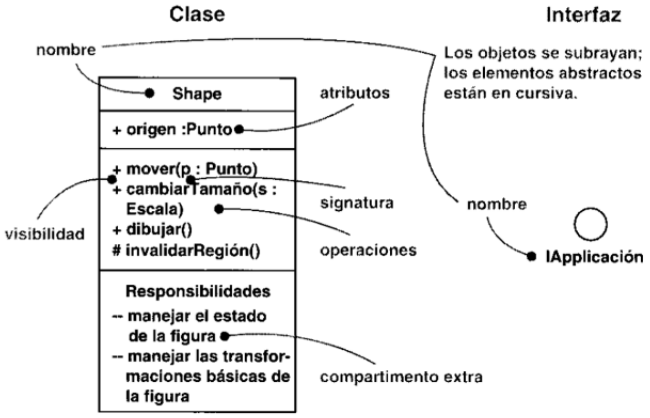
- La identificación de objetos, sus atributos y servicios.
- La organización de objetos dentro de una jerarquía.
- La construcción de descripciones dinámicas de objetos que muestran cómo se usan los servicios.
- La especificación de interfaces de objetos.

Clases

El ser humano tiende a agrupar seres o cosas (objetos) con características similares en grupos (clases). Consiste en la descripción de uno o más objetos del mundo real en base a una serie de atributos y servicios.

La agrupación en clases de los objetos permite la abstracción de un problema:

- Las definiciones comunes, tales como nombres de clases y de atributos se almacenan una vez por cada clase.
- Las operaciones se escriben una vez para cada clase → Reutilización de código.
- Las clases son la abstracción de los objetos. Pueden usarse para crear objetos.
- Las clases pueden heredar atributos y servicios de otras clases.
- Las clases son gráficamente representadas por cajas con compartimentos para:
    - Nombre de la clase
    - atributos y su visibilidad
    - métodos / operaciones y su visibilidad.
- Responsabilidades, Reglas, Historia de Modificaciones, etc. (Una Responsabilidad es un contrato u obligación de una clase)



Objetos

Para las personas, un objeto suele ser algo de lo siguiente:

Algo tangible y/o visible.
Algo que puede ser comprendido mentalmente.
Algo a lo que va dirigido el pensamiento o la acción.

Podemos utilizar como definición formal del mismo la siguiente:

Un objeto es una entidad real o abstracta con un papel bien definido en el dominio del problema.

Un objeto es una entidad que tiene un estado y un conjunto definido de operaciones que operan sobre este estado.
Los objetos son instancias de una clase y la clase de un objeto es una propiedad implícita del objeto.

- Cada objeto conoce su clase y la mayoría de los lenguajes de programación orientados al objeto pueden determinar la clase de un objeto en tiempo de ejecución.

El estado se representa mediante un conjunto de atributos del objeto.
Las operaciones asociadas con el objeto proveen servicios a otros objetos (clientes) que requieren estos servicios cuando necesitan realizar alguna actividad de cómputo.
Los objetos se crean de acuerdo a una definición de la clase.
La definición de la clase sirve como template para los objetos. Esta incluye las declaraciones de todos los atributos y servicios los cuales deben estar asociados con un objeto de esta clase.

Comunicación entre Objetos

Conceptualmente los objetos se comunican mediante el paso de mensajes.

Mensajes

El nombre del servicio requerido por algún objeto que llama.
Copias de la información requerida para ejecutar el servicio y el nombre de quien tiene esta información.

En la práctica, los mensajes se implementan a menudo mediante llamadas de procedimientos (métodos) donde:

Nombre = nombre de procedimiento.
Información = lista de parámetros.

Los beneficios prometidos por la OO son básicamente dos:

Reusabilidad: Los nuevos sistemas OO pueden ser creados utilizando otros

Sistemas Orientados a Objetos ya existentes. Es decir, es el grado en el que un programa (o partes de uno) pueden volverse a utilizar en otras aplicaciones dependiendo el empaque y el alcance de las funciones que lleva a cabo el programa.

Extensibilidad: Los nuevos Sistemas Orientados a Objetos así obtenidos son fácilmente ampliables sin tener que retocar los módulos, empleados en su construcción. El término OO debe entenderse como algo general, y que está íntimamente ligado a las etapas de Análisis y Diseño.

Diagrama de Clases

En la práctica, la creación de los Diagramas de clases normalmente se diseñan en paralelo con los Diagramas de interacción. Al comienzo del diseño se esbozan muchas clases, nombres de métodos y sus relaciones, antes de la elaboración de los diagramas de interacción. Se elaboran los Diagramas de Interacción y actualizamos los Diagramas de Clases, después extendemos los diagramas de interacción un poco más, y así sucesivamente.

Un diagrama de clase, aporta una visión estática o de estructura de un sistema, sin mostrar la naturaleza dinámica de las comunicaciones entre los objetos de las clases.

Los elementos principales de un diagrama de clase son cajas, que son los íconos utilizados para representar clases e interfaces. Cada caja se divide en partes horizontales. La parte superior contiene el nombre de la clase. La sección media menciona sus atributos. La tercera sección del diagrama de clase contiene las operaciones o métodos de la clase. Una clase abstracta o un método abstracto se indica con el uso de cursivas en el nombre del diagrama de clase.

Dependencia / Instanciación

"… Usa un ..."

Una relación débil, generalmente transitoria, que ilustra que una clase usa otra clase en algún momento. Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase).

Asociación

"… Tiene un …"

Más fuerte que la dependencia, la relación de línea sólida indica que la clase retiene una referencia a otra clase con el tiempo.

Agregación

"… posee un ..."

Más específico que asociación, esto indica que una clase es un contenedor de otras clases. Las clases contenidas se utilizan para el funcionamiento del objeto contenedor y no tienen una dependencia del ciclo de vida del mismo, por lo que cuando el contenedor se destruye, el contenido no es destruido. Esto se representa usando un diamante hueco. Agregación indica una relación "entero/parte", en la que la clase a la que apunta

la flecha se considera como una “parte” de la clase en el extremo diamante de la asociación.

Composición
“… es parte de ...”

Más específico que la agregación, esto indica una fuerte dependencia del ciclo de vida entre las clases, por lo que cuando el contenedor se destruye, también lo son los contenidos. Esto se representa utilizando un diamante relleno.

Los objetos que componen a la clase contenedora, deben existir desde el principio. (También pueden ser creados en el constructor, no sólo al momento de declarar los atributos).

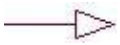
No hay momento (No debería) en que la clase contenedora pueda existir sin alguno de sus objetos componentes. Por lo que la existencia de estos objetos no debe ser abiertamente manipulada desde el exterior de la clase.

“En la composición, delegamos responsabilidades en colaboradores designados para ello”

Diferencias entre Agregación y Composición
La siguiente tabla intenta resumir algunas diferencias entre agregación y composición

	Agregación	**Composición**
Varias asociaciones comparten los componentes	Sí	No
Destrucción de los componentes al destruir el compuesto	No	Sí
Cardinalidad al nivel de compuesto	Cualquiera	0..1 ó 1
Representación	Rombo transparente	Rombo relleno

Generalización / Especialización
“… es un ...”



También conocido como **herencia**, esto indica que el subtipo es un tipo más específico del supertipo. Esto se representa usando un triángulo hueco en el lado general de la relación. Indica que una Subclase hereda los métodos y atributos especificados por una Super Clase, por ende la Subclase además de poseer sus propios métodos y atributos poseerá las características. Generalización y especialización expresan relaciones de inclusión entre conjuntos.

Multiplicidad
La multiplicidad de un extremo de una asociación significa el número de objetos de dicha clase que se asocia con la otra clase. Una multiplicidad se especifica mediante un entero no negativo o mediante un rango de enteros. Una multiplicidad especificada por “0..1” significa que existen 0 o 1 objetos en dicho extremo de la asociación. Por ejemplo, cada persona tiene un número de CUIL o no lo tiene (especialmente si no son argentinos); por

tanto, una multiplicidad de 0..1 podría usarse en una asociación entre una clase Persona y una clase CUIL en un diagrama de clase. Una multiplicidad que se especifica como “1...*” significa uno o más, y una multiplicidad especificada como “0...*” o sólo “*” significa cero o más.

1	Un elemento relacionado.
0..1	Uno o ningún elemento relacionado.
0..*	Varios elementos relacionados o ninguno.
1..*	Varios elementos relacionados pero al menos uno.
M..N	Entre M y N elementos relacionados.

Diagramas de Secuencias

En contraste con los diagramas de clase, que muestra la estructura estática de un componente de software, un diagrama de secuencia se usa para mostrar las comunicaciones dinámicas entre objetos durante la ejecución de una tarea. Este tipo de diagrama muestra el orden temporal en el que los mensajes se envían entre los objetos para lograr dicha tarea. Puede usarse un diagrama de secuencia para mostrar las interacciones en un caso de uso o en un escenario de un sistema de software.

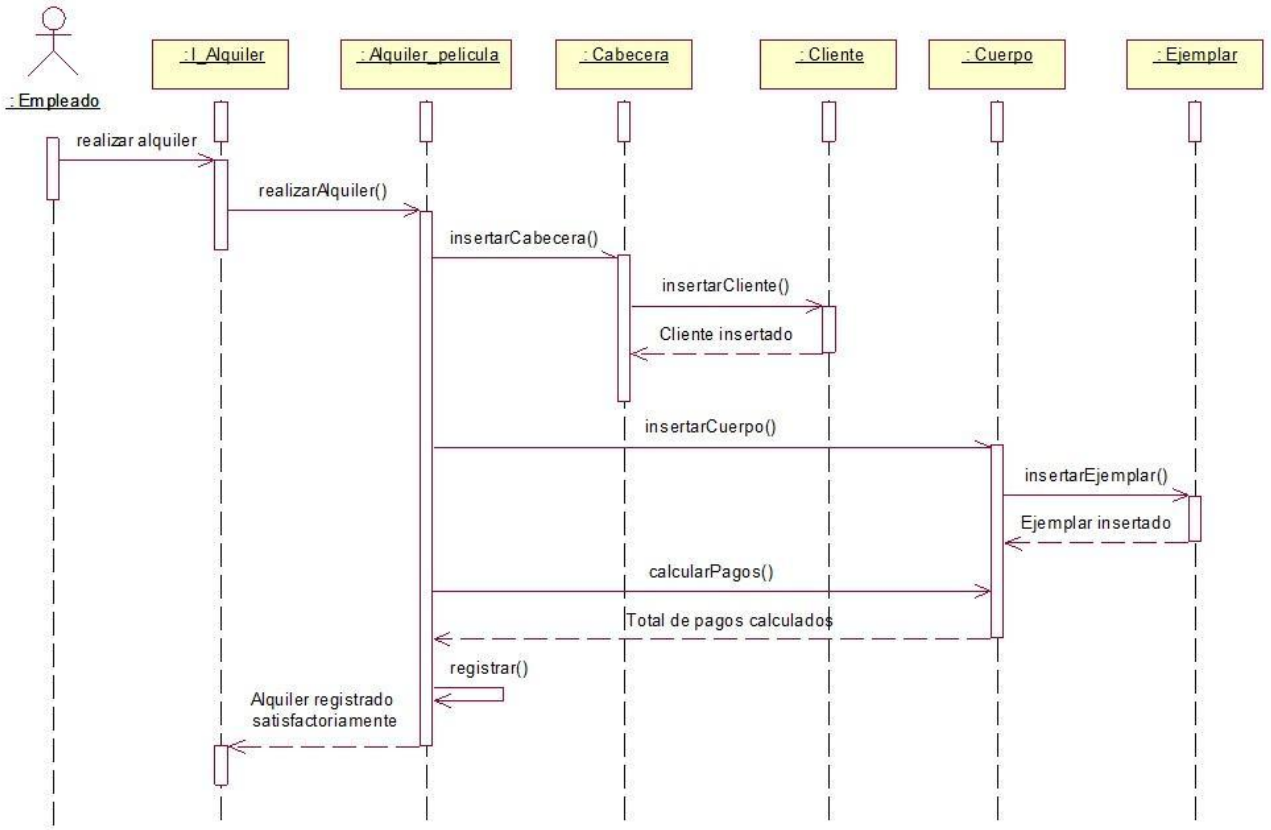
*“Los diagramas de Secuencias muestran la forma en que un **grupo de objetos** se **comunican** (interactúan) entre sí a lo **largo del tiempo**.*

El diagrama muestra los pasos involucrados, resaltando una figura en un dibujo. Por lo general, cada caja de la fila que hay en la parte superior del diagrama corresponde a un objeto, aunque es posible hacer que las cajas modelen otras cosas, como clases. Si la caja representa un objeto, entonces dentro de la caja puede establecerse de manera opcional el tipo del objeto, precedido por dos puntos.

Abajo de cada caja hay una línea punteada llamada línea de vida del objeto. El eje vertical que hay en el diagrama de secuencia corresponde al tiempo, donde el tiempo aumenta conforme se avanza hacia abajo.

Un diagrama de secuencia muestra llamadas de método usando flechas horizontales desde el llamador hasta el llamado, etiquetado con el nombre del método y que opcionalmente incluye sus parámetros, sus tipos y el tipo de retorno.

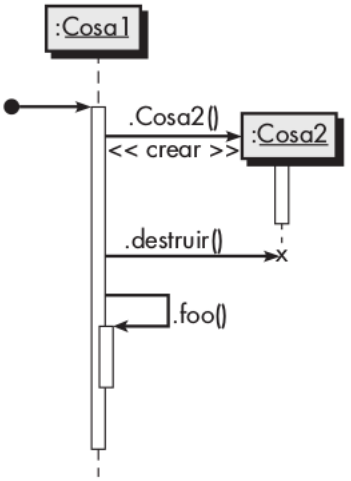
Cuando un objeto ejecuta un método, opcionalmente puede mostrar una barra blanca, llamada barra de activación, abajo de la línea de vida del objeto. El diagrama también puede mostrar opcionalmente el retorno de una llamada de método con una flecha punteada y una etiqueta opcional.



Los diagramas por lo general son muy directos y no contienen condicionales o bucles. Si se requieren estructuras de control lógico, probablemente sea mejor dibujar un diagrama de secuencia separado para cada caso, es decir, si el flujo del mensaje puede tomar dos rutas diferentes dependiendo de una condición, entonces dibuje dos diagramas de secuencia separados, uno para cada posibilidad.

Existen muchas otras características especiales que pueden incluirse en un diagrama de secuencia. Por ejemplo:

1. Se puede distinguir entre mensajes sincrónicos y asíncronos. Los primeros se muestran con puntas de flecha sólidas mientras que los asíncronos lo hacen con puntas de flecha huecas.
2. Se puede mostrar un objeto que envía él mismo un mensaje con una flecha que parte del objeto, gira hacia abajo y luego apunta de vuelta hacia el mismo objeto.
3. Se puede mostrar la creación de objeto dibujando una flecha etiquetada de manera adecuada (por ejemplo, con una etiqueta <<crear>>) hacia una caja de objeto. En este caso, la caja aparecerá en el diagrama más abajo que las cajas correspondientes a objetos que ya existen cuando comienza la acción.
4. Se puede mostrar destrucción de objeto mediante una gran X al final de la línea de vida del objeto. Otros objetos pueden destruir un objeto, en cuyo caso una flecha apunta desde el otro objeto hacia la X. Una X también es útil para indicar que un objeto ya no se usa y que, por tanto, está listo para el colector de basura.


:Cosa1
.Cosa2()
<< crear >>
:Cosa2
.destruir()
x
.foo()