

Fundamentos de Ingeniería de Software

Marcello Visconti y Hernán Astudillo
Departamento de Informática
Universidad Técnica Federico Santa María
{visconti,hernan} en inf.utfsm.cl

Diseño de una Solución

Contenido



- ✍ Aplicación de los patrones GRASP para asignar responsabilidades a las clases.
- ✍ Uso de la notación de UML en los diagramas de colaboración para describir gráficamente el diseño de la interacción de objetos.

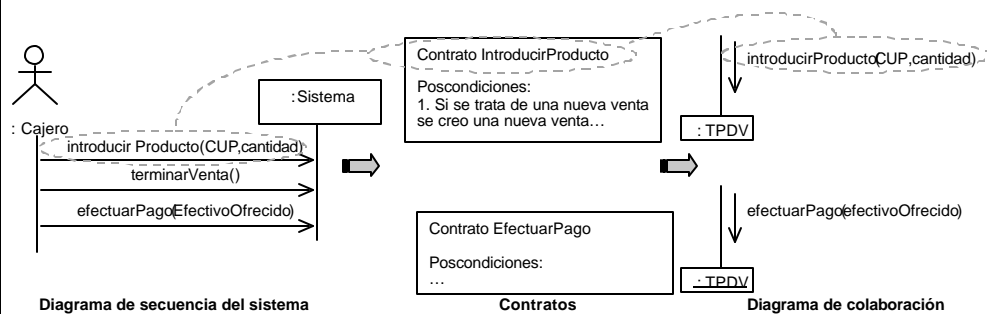
Diseño de una Solución

Introducción

- ✍ En la práctica, los diseñadores se percatan de que la preparación de los diagramas de interacción es uno de los pasos más lentos.
- ✍ La asignación de responsabilidades y la elaboración de los diagramas de interacción representan uno de los pasos más importantes en la fase de diseño.

Diseño de una Solución

Relación entre artefactos



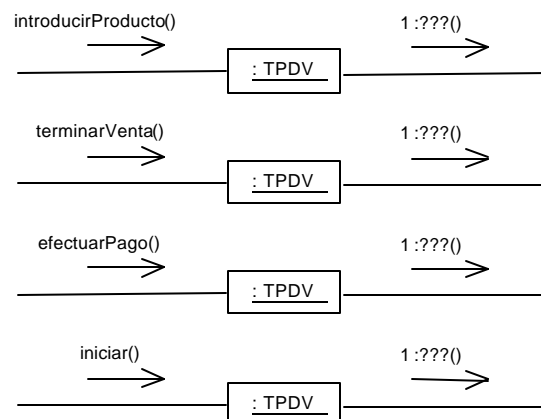
Diseño de una Solución

Diagramas de interacción y eventos del sistema

- ✍ En la iteración actual de la aplicación del punto de venta estamos considerando dos casos de uso y sus eventos sistemáticos asociados:
 - ✍ Comprar Productos: introducirProducto terminarVenta efectuarPago
 - ✍ Inicio: inicio
- ✍ Por cada evento del sistema se construye un diagrama de colaboración cuyo mensaje inicial sea el de sus eventos.
 - ✍ Habrá, pues, cuatro diagramas de interacción por lo menos: uno por cada evento del sistema.

Diseño de una Solución

Diagramas de interacción y eventos del sistema



Diseño de una Solución

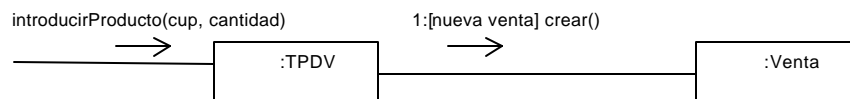
Diagramas de interacción y contratos

- ✍ En cada contrato revisamos los cambios de estado de los objetos responsables y las poscondiciones asociadas.
 - ✍ Nótese que, en caso de omitir la preparación del contrato, de todos modos deberíamos elaborar los diagramas de interacción retornando a los casos de uso y reflexionando sobre lo que debemos lograr.
- ✍ No obstante, los contratos organizan y aslan la información en un formato funcional, al mismo tiempo que estimulan la investigación en la fase de análisis y no en la de diseño.

Diseño de una Solución

Diagramas de interacción y contratos

- ✍ Por ejemplo, en esta operación parcial del sistema *introducirProducto*, en la figura incluimos un diagrama parcial de colaboración que satisface el cambio de estado de la creación de *Venta*.



Diseño de una Solución

Diagramas de interacción y contratos

- ✍ Los Diagramas deben prepararse con el propósito de cumplir las poscondiciones del contrato.
 - ✍ Poscondiciones definidas de antemano no son sino una excelente conjetura o estimación inicial de lo que se pretende alcanzar.
 - ✍ En los contratos debemos ver un mero punto de partida para establecer lo que se hará, pero sin sentirnos obligados por ellos.
- ✍ Una ventaja del desarrollo iterativo radica en que brinda un soporte espontáneo a la detección de nuevos resultados del análisis y del diseño durante las fases de solución y de construcción.

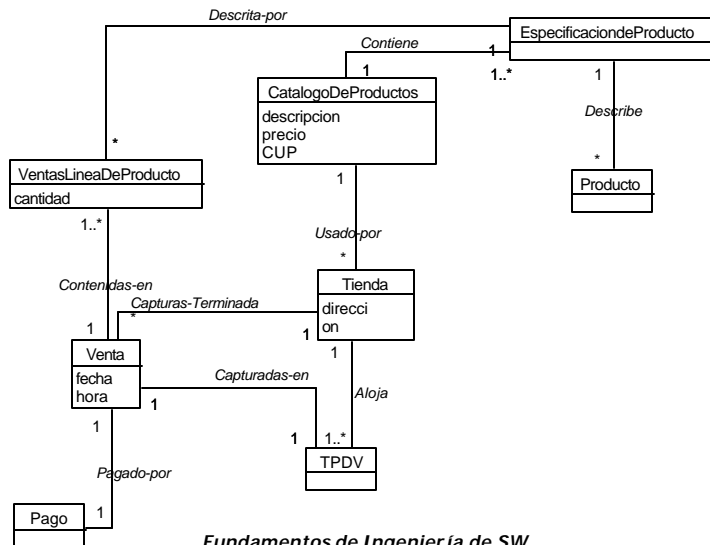
Diseño de una Solución

Diagramas de interacción y modelo conceptual

- ✍ Algunos de los objetos que interactúan a través de mensajes en los diagramas de colaboración provienen del modelo conceptual.
- ✍ En parte, la elección de la asignación acertada de las responsabilidades mediante los patrones GRASP se funda en la información contenida en el modelo conceptual.

Diseño de una Solución

Diagramas de interacción y modelo conceptual



Diseño de una Solución

Diagrama de colaboración: *introducirProducto*

Contrato

- Nombre:** *IntroducirProducto (cup:numero, cantidad:entero)*
- Responsabilidades:** Capturar (registrar) la venta de un producto y agregarla a la venta. Desplegar la descripción y el precio del producto.
- Tipo:** Sistema
- Referencias cruzadas:** Funciones del sistema: R1.1, R1.3, R1.6
Casos de uso: Comprar productos
- Notas:** Utilizar el acceso superrápido a la base de datos.
- Excepciones:** Si el CUP no es válido, indicar que se cometió un error.
- Salida:**
- Precondiciones:** El sistema conoce el CUP.
- Poscondiciones:**
- ? Si se trata de una nueva venta, se creó una *Venta* (creación de instancia).
 - ? Si se trata de una nueva venta, la nueva *Venta* fue asociada a un *TPDV* (asociación formada).
 - ? Se creó una instancia *VentasLineadeProducto* (creación de instancia).
 - ? Se asoció una instancia de *VentasLineadeProducto* a la *Venta* (asociación formada).
 - ? Se asignó una cantidad a *VentasLineadeProducto.cantidad* (modificación de atributo).
 - ? Se asoció una instancia *VentasLineadeProducto* a la instancia *EspecificaciondeProducto*, basado en la correspondencia del CUP (asociación formada).

Diseño de una Solución

Selección de la clase Controlador

- ✎ En la elección del controlador que se encargue del mensaje de las operaciones del sistema *introducirProducto*, disponemos de las siguientes opciones:
 - ✎ Representa el "sistema" global
 - ✎ *TPDV*
 - ✎ Representa la empresa u organización global
 - ✎ *Tienda*
 - ✎ Representa algo en el mundo real que está activo (por ejemplo, el papel de una persona) y que puede intervenir en la tarea
 - ✎ *Cajero*
 - ✎ Representa un manejador artificial de todas las operaciones del sistema de un caso de uso.
 - ✎ *ManejadordeComprarProductos*

Diseño de una Solución

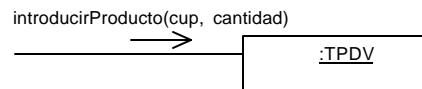
Selección de la clase Controlador

- ✎ Un controlador de fachada como *TPDV* es adecuado...
 - ✎ si el sistema ofrece pocas operaciones y si ese controlador no asume demasiadas responsabilidades (en otras palabras, si empieza a perder cohesión).
- ✎ Un controlador de papeles o de casos de uso será una decisión acertada...
 - ✎ cuando el sistema tiene demasiadas operaciones y queremos distribuir las responsabilidades para que las clases controlador no se atiborren, ni pierdan su orientación central (o sea, que sean cohesivas).
- ✎ En este caso, *TPDV* será suficiente porque el sistema tiene pocas operaciones.

Diseño de una Solución

Selección de la clase Controlador

- ⌘ Es importante darse cuenta que "TPDV" es una instancia de un objeto en el "territorio del software".
 - ⌘ No es un terminal real de punto de venta, sino una abstracción de software que representa el registro.



Diseño de una Solución

Presentación visual de la descripción y del precio

- ⌘ En virtud de un principio del diseño denominado separación de modelo - vista (*Controlador*), no compete a los objetos del dominio (como *TPDV* o *Venta*) comunicarse con la capa de interfaz para el usuario (las ventanas gráficas, por ejemplo).
- ⌘ Lo único que se requiere en lo tocante a las responsabilidades del despliegue de la información es conocer los datos, como se verá en este caso.

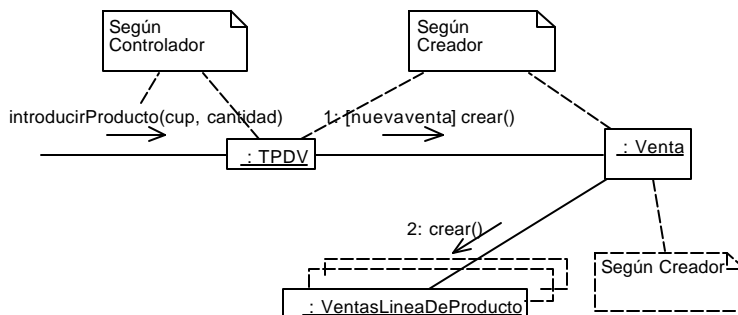
Diseño de una Solución

Realización de una nueva venta

- ✎ Poscondiciones contractuales establecen lo siguiente:
 - ✎ Si se trata de una nueva venta, se creó una *Venta* (*creación de instancia*).
 - ✎ Si se trata de una nueva venta, se asoció la nueva *Venta* a *TPDV* (*asociación formada*).
 - ✎ Además, cuando se crea la *Venta*, hay que generar una colección vacía para que registre todas las instancias futuras *VentasLineadeProducto*.

Diseño de una Solución

Realización de una nueva venta



Diseño de una Solución

Creación de una instancia *VentasLineadeProducto*

- ✎ Algunas otras poscondiciones contractuales de *introducirProducto* establecen lo siguiente:
 - ✎ Se creó una instancia *VentasLineadeProducto* (*creación de instancia*).
 - ✎ Se asoció *VentasLineadeProducto* a la *Venta* (*asociación formada*).
 - ✎ Se asignó el valor a *VentasLineadeProducto.cantidad* de cantidad (*modificación de atributo*).
 - ✎ Se asoció la instancia *VentasLineadeProducto* con una *EspecificaciondeProducto* a partir de la correspondencia en el *CUP* (*asociación formada*).

Diseño de una Solución

Obtención de instancia *EspecificaciondeProducto*

- ✎ Necesitamos asociar la instancia *VentasLineadeProducto* a *Especificaciondeproducto* que concuerde con el *CUP* que se recibe. Ello significa que se requiere recuperar una *EspecificaciondeProducto* basada en la correspondencia del *CUP*.
- ✎ Un primer paso de gran utilidad es el siguiente:
 - ✎ *Comience asignando las responsabilidades definiéndolas con claridad.*

Diseño de una Solución

Obtención de instancia *EspecificacioneProducto*

- ✎ ¿Quién debería asumir la responsabilidad de conocer una instancia *EspecificacioneProducto* basado en que coincida el valor del CUP?
 - ✎ En la generalidad de los casos, el patrón Experto de GRASP es el principio que ha de aplicarse en primer lugar.
- ✎ ¿Quién está enterado de todo lo concerniente a *EspecificacioneProducto*?
 - ✎ El análisis del modelo conceptual revela que *CatalogodeProductos* contiene lógicamente todas las especificaciones de productos y así, de acuerdo con el patrón Experto, *CatalogodeProductos* es idóneo para asumir esta responsabilidad de consulta.

Diseño de una Solución

Visibilidad ante un *CatalogodeProductos*

- ✎ ¿Quién debería enviar el mensaje especificación al *CatalogodeProductos* para solicitar una *EspecificacioneProducto*?
 - ✎ Es razonable suponer que una instancia *TPDV* y una de *CatalogodeProductos* fueron creadas durante el caso inicial de uso Inicio y que existe una conexión permanente entre el objeto *TPDV* y el objeto *CatalogodeProductos*.
- ✎ Esta suposición nos permitirá concluir que *TPDV* puede enviar el mensaje especificación a Catálogo de productos.

Diseño de una Solución

Visibilidad ante un *CatalogodeProductos*

- ✍ Ello significa la existencia de otro concepto en el diseño orientado a objetos: el de visibilidad.
- ✍ La visibilidad es la capacidad de un objeto para 'ver' o tener una referencia a otro. Para que un objeto envíe a otro un mensaje, debe ser visible al segundo.
 - ✍ Supondremos que *TPDV* tiene una conexión o referencia permanente a *CatalogodeProductos*; por tanto, es visible para esta instancia y de ahí que pueda enviarle Mensajes como *especificación*.

Diseño de una Solución

Recuperación de *EspecificaciondeProducto*

- ✍ En la versión final de una aplicación real al punto de venta, difícilmente todas las *EspecificaciondeProducto* estarán en la memoria.
 - ✍ Lo más probable es que se encuentren almacenadas en una base de datos relacional o de objetos y que sean recuperadas previa solicitud.
 - ✍ No obstante, para no hacer compleja la exposición pospondremos el estudio de los problemas concernientes a la recuperación partiendo de una base de datos.
- ✍ Supondremos que todas las *EspecificaciondeProducto* se hallan en la memoria. Más tarde se tratará el tema del acceso a la base de datos de objetos persistentes.

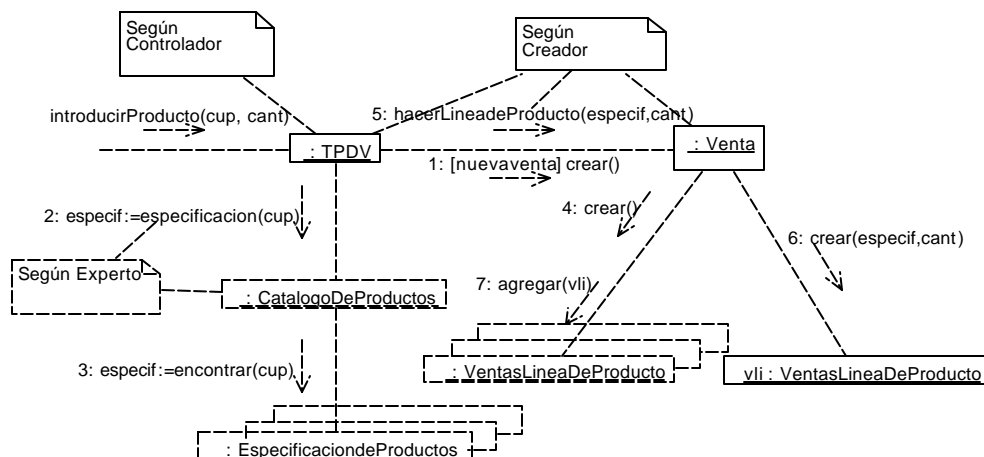
Diseño de una Solución

Diagrama de colaboración *introducirProducto*

- El diagrama de colaboración de la figura refleja las decisiones sobre la asignación de responsabilidades y sobre la manera en que los objetos deberían interactuar.
- Observe que se reflexionó detenidamente para llegar a él, con base en los patrones GRASP.
- El diseño de las interacciones de los objetos y la asignación de responsabilidades requieren una seria deliberación.

Diseño de una Solución

Diagrama de colaboración *introducirProducto*



Diseño de una Solución

Mensajes a multiobjetos

- ✎ La interpretación predeterminada de un mensaje enviado a un multiobjeto es que se envía implícitamente a todos los elementos de la colección/contenedor, pero también puede interpretarse como un mensaje dirigido al propio objeto colección.

Diseño de una Solución

Mensajes a multiobjetos

- ✎ Por ejemplo, en el Diagrama de colaboración *introducirProducto*:
 - ✎ El mensaje *encontrar* (3) enviado al multiobjeto *EspecificaciondeProducto* se dirige una vez a la estructura de datos colección representada por el multiobjeto.
 - ✎ El mensaje *crear* (4) enviado al multiobjeto *VentasLineadeProducto* tiene por objeto generar la estructura de datos colección representada por el multiobjeto; su finalidad no es crear una instancia de la clase *VentasLineadeProducto*.
 - ✎ El mensaje *agregar* (7) enviado al multiobjeto *VentasLineadeProducto* tiene por objeto incorporar un elemento a la estructura de datos colección representada por el multiobjeto.

Diseño de una Solución

Diagrama de colaboración *terminarVenta*

✍ Ejercicio:

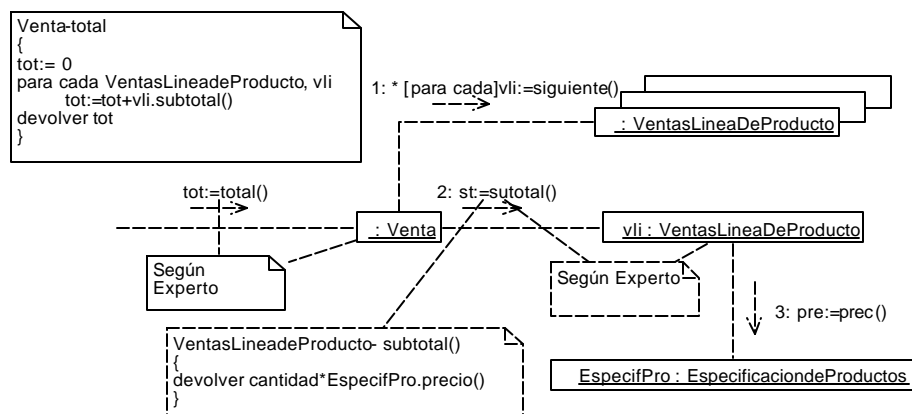
Desarrollar el Diagrama de colaboración para el caso de término de una venta.

Contrato

Nombre: terminarVenta()
Responsabilidades: Registrar que se terminó de capturar los artículos de la venta y presentar visualmente el total de la venta.
Tipo: Sistema
Referencias cruzadas: Funciones del sistema: R1.2.
Casos de uso: Comprar Productos.
Notas: Utilizar el acceso superrápido a la base de datos.
Excepciones: Si está efectuándose una venta, indicar que se cometió un error.
Salida:
Precondiciones: El sistema conoce el CUP.
Poscondiciones:
Se asignó a *Venta.estaTerminada* el valor verdadero (*modificación de atributo*)

Diseño de una Solución

Cálculo del total de la venta: Solución



Diseño de una Solución

Diagrama de colaboración: *Iniciar*

- ¿Cuándo crear el diagrama de colaboración "*Iniciar*"?
 - La mayoría de los sistemas, si no es que todos, tienen un caso de uso *Iniciar* y alguna operación inicial relacionada con el comienzo de la aplicación. Aunque es la primera en ser ejecutada, se ha considerado la conveniencia de posponer la preparación del respectivo diagrama de colaboración hasta después de considerar el resto de las operaciones del sistema.
- Prepare al final el diagrama de colaboración *Iniciar*.

Diseño de una Solución

Diagrama de colaboración: *Iniciar*

- El diagrama de colaboración de la operación *Iniciar* describe lo que sucede cuando se crea el objeto inicial del dominio del problema y, opcionalmente, lo que sucede si asume el control.
- No incluye ninguna actividad anterior ni subsecuente en la capa de objetos destinada a la interfaz gráfica para el usuario, si es que existe.

Diseño de una Solución

Diagrama de colaboración: *Iniciar*

- ✍ Por tanto la operación *Iniciar* puede interpretarse así:
 - ✍ En un diagrama de colaboración, envíe un mensaje *crear()* para producir el objeto inicial del dominio.
 - ✍ (opcional) Si el objeto inicial va a asumir el control del proceso, en un segundo diagrama de colaboración envíele un mensaje *ejecutar* (u otro equivalente).

Diseño de una Solución

Diagrama de colaboración: *Iniciar*

- ✍ ¿Cuál debería ser la clase del objeto inicial del dominio?
- ✍ Escoja como objeto inicial del dominio:
 - ? Una clase que represente todo el sistema de información lógico.
 - ? Una clase que represente íntegramente el negocio u organización.
- ✍ En la selección de estas opciones pueden influir consideraciones referentes a los patrones Alta Cohesión y Bajo Acoplamiento.
 - ? Todo el Sistema de información lógico: *TPDV*, *SistemaInformacionalmenudeo*
 - ? Negocio u organización global: *Tienda*
- ✍ En esta aplicación se escogió la *Tienda* como objeto inicial.

Diseño de una Solución

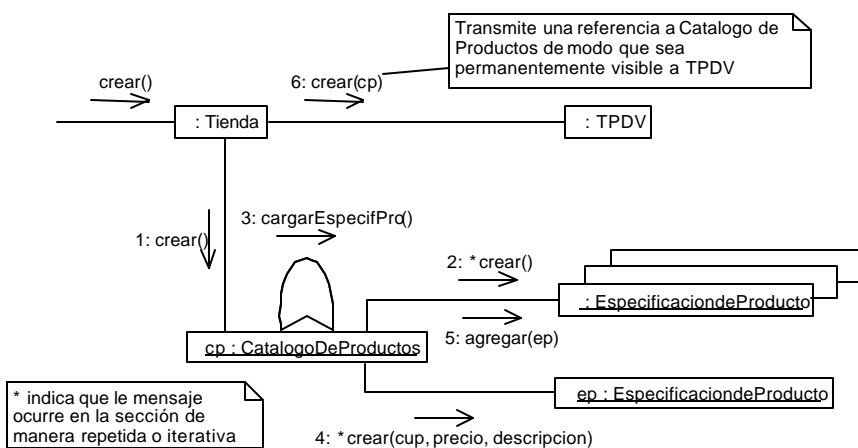
Diagrama de colaboración: *Iniciar*

Contrato

- Nombre:** IniciarO.
- Responsabilidades:** Inicializar el sistema.
- Tipo:** Sistema
- Referencias cruzadas:**
- Notas:**
- Excepciones:**
- Salida:**
- Precondiciones:**
- Poscondiciones:**
- ? Se crearon *Tienda*, *TPDV*, *CatalogodeProductos* y *EspecificaciondeProducto* (creación de instancia).
 - ? Se asoció *CatalogodeProductos* a *EspecificaciondeProducto* (asociación formada).
 - ? Se asoció *Tienda* a *CatalogodeProductos* (asociación formada).
 - ? Se asoció *Tienda* a *TPDV* (asociación formada).
 - ? Se asoció *TPDV* a *CatalogodeProductos* (asociación formada).

Diseño de una Solución

Diagrama de colaboración: *Iniciar*



Diseño de una Solución

Diagrama de colaboración: *Iniciar*

- ✍ Una discrepancia interesante entre el análisis y el diseño se ejemplifica en el hecho de que la *Tienda* sólo crea **un** objeto *TPDV*.
 - ✍ Desde el punto de vista analítico, una tienda real puede albergar muchas terminales en el punto de venta. Pero ahora estamos considerando un diseño de software, no lo que ocurre en el mundo real.
 - ✍ La *Tienda* y *TPDV* representada en un diagrama de colaboración no es una tienda real; es un objeto de software.
 - ✍ En nuestros requerimientos actuales, el objeto de SW *Tienda* no necesita crear más que una sola instancia del objeto de SW *TPDV*.

Diseño de una Solución

Resumen

- ✍ Aplicación de los patrones GRASP para asignar responsabilidades a las clases.
- ✍ Uso de la notación de UML en los diagramas de colaboración para describir gráficamente el diseño de la interacción de objetos.

Diseño de una Solución

Quiz



- ✎ ¿Cuál es la utilidad del contrato para diagramas de interacción?
- ✎ ¿Cuál es la utilidad del modelo conceptual para diagramas de interacción?
- ✎ ¿Cuando se prepara el contrato para el caso de uso Iniciar?
- ✎ ¿En qué ayudan los patrones?
- ✎ ¿Cual es el orden de aplicación de los patrones vistos?