

CUSTOMER JOURNEY: **herramienta para trackear las emociones del usuario y mejorar así sus experiencias.**

1. Descubrimiento - Awareness
El **consumidor descubre el producto**, las características del servicio o producto no son importantes porque no se intenta incitar al usuario a comprar, simplemente **se le informa su existencia.**
2. Consideración - Consideration
Cuando el consumidor quiere realizar una compra, y considera diferentes opciones. Acá es donde **se le informa a los consumidores de las características del producto, así como los puntos fuertes.**
3. Compra - Purchase
El usuario ya ha tomado la decisión de compra y decide llevarla a cabo. Es importante tener un canal online que no realentice o frene el proceso.
4. Retención - Retention
Primera fase post-venta donde **se busca mantener la satisfacción del cliente**, para que vuelva a comprar y se fidelice.
5. Recomendación - Advocacy
Tras una experiencia satisfactoria, es posible que **los clientes nos ayuden a mejorar la imagen de la empresa.** Las redes sociales, valoraciones y el boca a boca.

CONSTRUCCIÓN:

1. Se dibuja un gráfico donde
 - Eje X muestra las fases por la que pasa el cliente a lo largo del tiempo
 - Eje Y muestra cómo se siente la experiencia, desde negativas, positivas y puntos críticos.
2. Se definen las interacciones en las que la empresa toma parte. Visibles o invisibles desde la perspectiva del cliente.

DISEÑO: es el proceso de definir la arquitectura, componentes, interfaces, otras características del sistema y el **resultado** de ese proceso.

SISTEMA: **entidad lógica** con conjunto de **responsabilidades y objetivos.** HW y/o SW.

SUBSISTEMA: un **sistema que es parte de un sistema mayor.** Tiene una interfaz definida.

COMPONENTE: **pieza de SW o HW que tiene definido un rol claro.** Puede ser **aislado y reemplazable.** Muchos están diseñados para ser **reutilizables** y otros realizan funciones con un **propósito especial.**

MÓDULO: **componente definido a nivel de lenguaje de programación.** Package / Namespace.

PROCESO DE DISEÑO DEL SOFTWARE

1. Diseño arquitectónico
2. Especificación abstracta: se especifican los **subsistemas** donde se realiza un **servicio importante.** Contiene **objetos altamente acoplados**, relativamente **independientes de otros** subsistemas, y generalmente se **descompone en módulos o subsistemas** más pequeños.
3. Diseño de la interfaz: permite que los subsistemas se utilicen como **caja negra.**

4. Diseño de los componentes: se descompone cada **subsistema en componentes** y se diseñan sus interfaces.

PRINCIPIOS DEL DISEÑO

- Dividir y Conquistar
- Incrementar la Cohesión: mantener en subsistemas o módulos las cosas que están relacionadas y afuera las restantes. Un módulo cohesivo lleva a cabo una sola tarea dentro. Permite diseños robustos y altamente cohesionados por tener sus elementos fuertemente relacionados.
- Reducir Acoplamiento: cuando existe interdependencia entre un módulo y otro. Mantener el nivel de abstracción tan alto como sea posible, asegurando que el diseño oculte detalles, reduciendo la complejidad. Un bajo acoplamiento indica un sistema bien dividido y puede conseguirse mediante la eliminación o reducción de relaciones innecesarias. Permite conseguir que cada componente sea tan independiente como sea posible.
 - Fuertemente acoplados: cuando los módulos utilizan variables compartidas o intercambian información de control
 - Débilmente acoplados: garantizar que los detalles de la representación de datos están dentro de un componente, Interfaz con otros componentes mediante lista de parámetros. Información compartida limitada a aquellos componentes que necesitan acceder a la información evitando compartir información de forma global.
- Incrementar la reusabilidad: diseñar los aspectos del sistema de forma que pueda ser reusado en otros contextos.
- Diseño para ser flexible: activamente anticipar los cambios que el diseño pueda tener en el futuro.
- Anticipe la obsolescencia: planificar la tecnología para que el software continúe ejecutándose o sea fácilmente cambiabile.

UML (lenguaje unificado de modelado) es un lenguaje de modelado para especificar o describir método o procesos, especialmente utilizado en POO. Es un estándar relativamente abierto, controlado por el Object Management Group (OMG). UML ha unificado muchos lenguajes de modelado gráfico.

TIPOS DE DIAGRAMAS

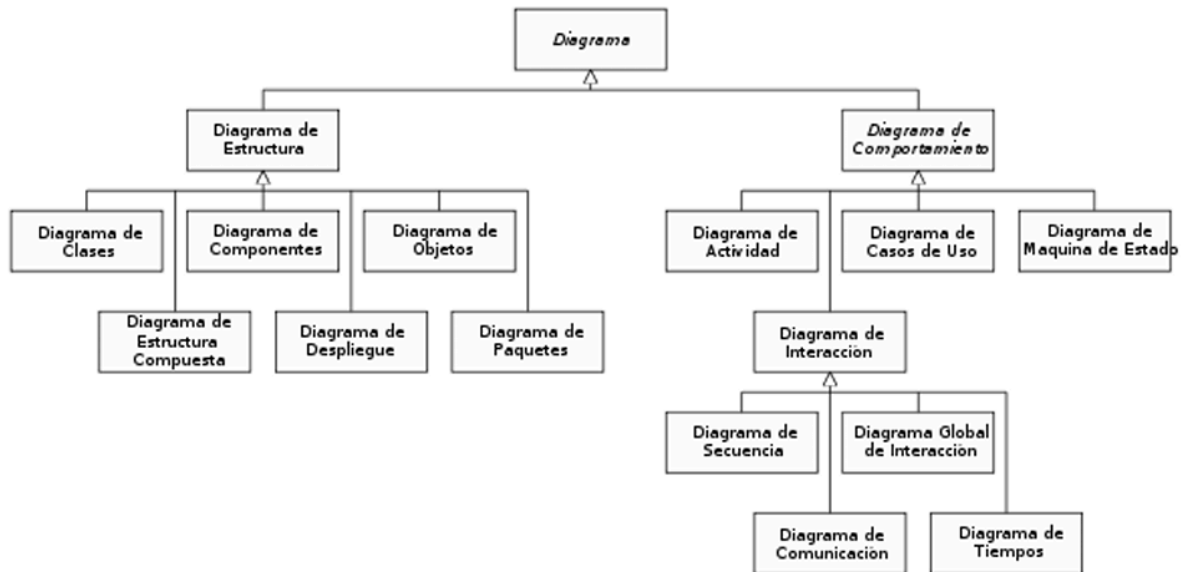


Diagrama de Estructura

- DClases

Describen los tipos de objetos en un sistema, así como las relaciones **estáticas** entre ellos. Incluyen **propiedades, operaciones y restricciones**. Es mejor evitar la creación de diagramas grandes y dividirlos en otros más pequeños. Usar colores para agrupar módulos comunes.

- DPaquetes

Permiten agrupar por algún concepto con mayor nivel de abstracción. Permiten **visualizar los módulos** y sus dependencias en la aplicación. Los diagramas de clases sirven para representar la estructura más básica de un sistema, a medida que empiezan a crecer se usan los diagramas de paquetes **para visualizar relaciones y dependencias**.

- DComponentes

Agrupación de módulos por algún criterio. Sirve para documentar las relaciones entre el sistema a través de las interfaces.

- DDeploy

Utilizado para ver cómo se sitúan los componentes lógicos en los distintos nodos físicos. Suele ser utilizado junto al diagrama de componentes, incluso diagrama de paquetes, juntos dan una visión general de cómo estará desplegado el sistema. Son nodos conectados por rutas de comunicación
tipos de nodo:

- dispositivo: HW, computadora o pieza de HW
- entorno de ejecución: SW, SO o contenedor

Los nodos contienen artefactos: manifestaciones físicas del software: generalmente archivos.

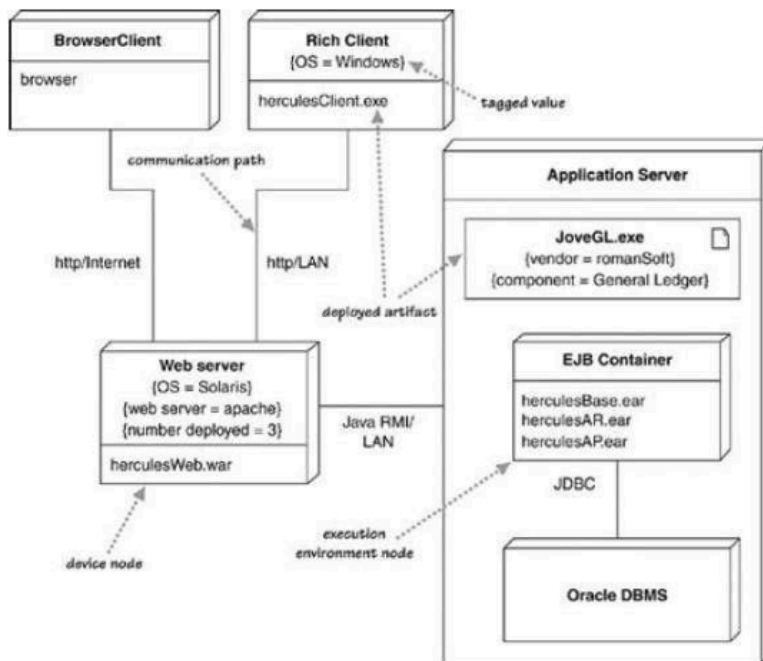


Diagrama de Comportamiento

- DActividad

Muestran una secuencia de acciones. Modelan el **workflow o comportamiento de un sistema** que va desde un punto inicial hasta un punto final. NO tiene en cuenta los mensajes, muestran **qué sucede, pero no quién hace qué**. Tiene una influencia en **comprender** el negocio o funcionalidades más que su implementación.

#Si se desea mostrar quién hace qué, se puede dividir un diagrama de actividad en particiones.

- DCasosUso

Representan los requisitos funcionales y cómo los actores se comunican con el sistema. Relaciones entre requisitos funcionales y actores. Hay que entender al actor como un “perfil” pudiendo existir varios usuarios que actúan como el mismo actor.

- <<extend>> → el usuario puede hacerlo.
Ej: seleccionar destino <<extend>> seleccionar tipo uber
ingresar usuario <<extend>> ver historial o <<extend>> actualizar perfil
- <<include>> → se hace automáticamente o es totalmente necesario hacerlo.
Ej: seleccionar destino <<include>> cálculo de tarifa

- Diagrama de interacción: DSecuencia

Representan el intercambio de mensajes entre las distintas partes del sistema. Define el comportamiento **dinámico** del sistema. Se suele construir junto al diagrama de clases. Muestran la **interacción**, mostrando cada participante.

Tiene 2 dimensiones

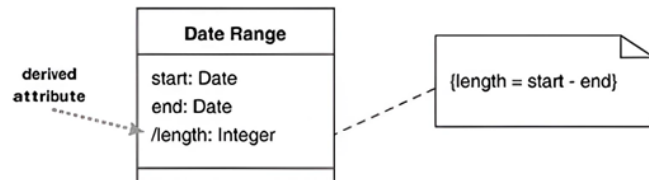
- horizontal: representa objetos que participan en la secuencia
- vertical: representa la línea de tiempo sobre la que los elementos actúan.

#Las flechas son los mensajes

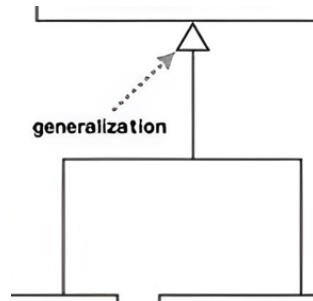
DIAGRAMA DE CLASES

Tipo de atributo:

- público (+)
- privado (-)
- restringido (#) → en herencia, tiene acceso el objeto y su descendiente
- derivado (/) → atributos que se calculan en base a otros atributos



Generalización / Herencia

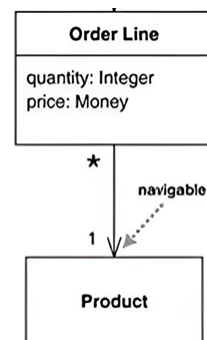


Asociación: conecta dos clases en cualquier conexión lógica

- bidireccional: Ambas clases están conscientes una de la otra y de la relación que tienen entre sí.



- unidireccional: Una clase está consciente de la otra e interactúa con ella.

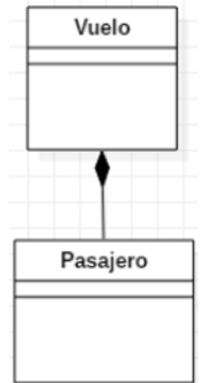


- **clase de asociación:** durante el proceso de diseño, pueden surgir comportamientos u otros atributos que no tienen un responsable claro en la asociación.

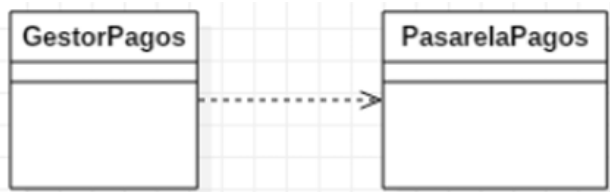
Agregación (es una asociación): En la agregación, las clases secundarias existen independientemente de la principal. Si se elimina la clase principal, la clase secundaria continúa existiendo.



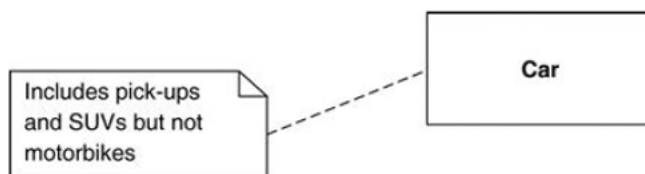
Composición (es una asociación): La composición es lo opuesto a la agregación, las clases no existen de forma independiente. Cuando se elimina la clase principal, también la secundaria.



Dependencia: Una relación de dependencia se utiliza entre dos clases o entre una clase y una interfaz, e indica que una clase requiere de otra para proporcionar alguno de sus servicios. (una clase usa las cosas del otro)



Comentarios / Notas



#Restricciones, se escriben como comentarios entre {}.

Enumeradores: set fijo de valores que no tienen comportamiento.

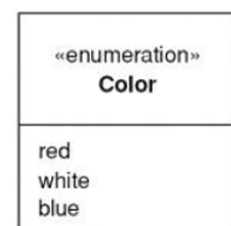
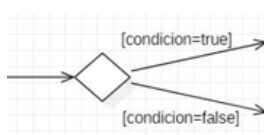


DIAGRAMA DE ACTIVIDAD

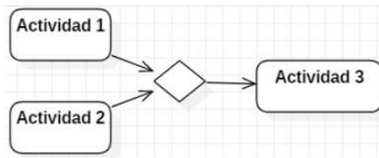


Notación de un flujo de actividad



simboliza el **orden de ejecución**

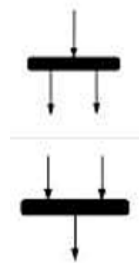


condiciones (if)



nodo de fusión: reciben 2 o más flujos y emiten 1. Los flujos no tienen por qué ser en paralelo. Común en estructuras condicionales donde diferentes ramas de decisiones convergen en una única actividad posterior.

comienza con  y termina con .



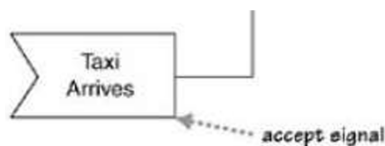
bifurcación: actividades en paralelo

unión: fin de flujos de actividades en paralelo

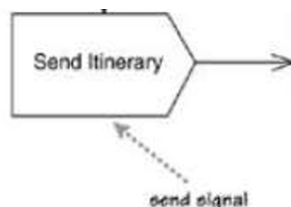
Las acciones también pueden responder a señales.



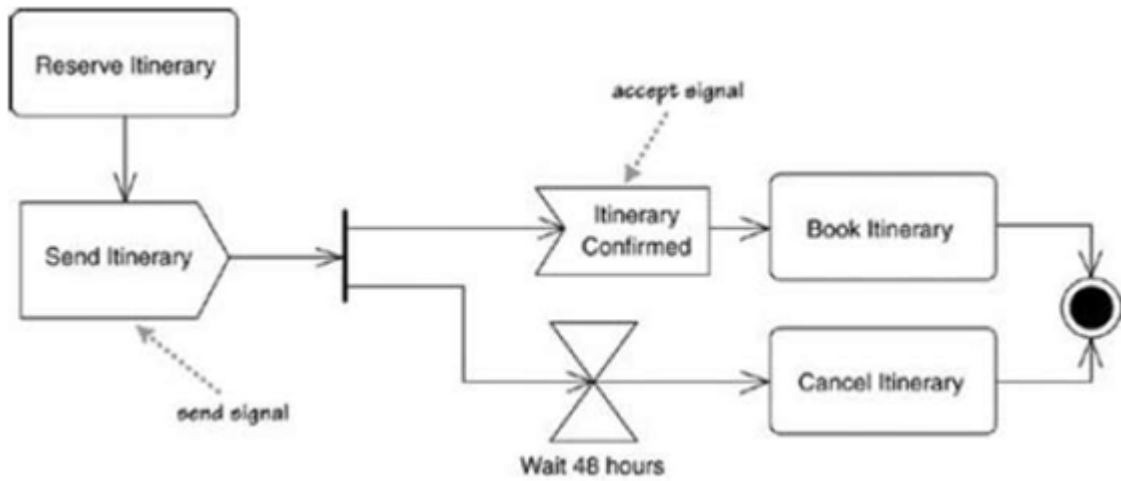
tiempo



recepción: La actividad está constantemente escuchando.



envío: cuando se envía un mensaje y luego hay que esperar una respuesta antes de poder continuar
Ejemplo:



PONER EN EL RESUMEN LA JUSTIFICACIÓN DE POR QUÉ ELEGIR CADA DIAGRAMA.