ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

Subtemas:

- Conceptos de Análisis y Diseño Orientado a Objetos.
- Diagramas UML: Clases y Casos de Uso

Palabras clave: Modelado, ADOO, objetos, atributos, métodos, encapsulación, herencia, generalización, especialización, polimorfismo, clases, clases abstractas, UML, diagrama de casos de uso y diagrama de clases.

Introducción

- La evolución del software ha experimentado modificaciones importantes en su corta historia, que se pueden atribuir al desarrollo de metodologías del propio software y a la evolución del hardware.

- Durante la época de los años 50 y 60 las aplicaciones se desarrollaban enfocadas al proceso por lotes.

- En la época de los 60 y 70 se comienzan a construir sistemas multiusuario y de tiempo real.

- Durante la época de los 70 a los 80 se introduce el término de arquitectura cliente-servidor, surgen los primeros sistemas de desarrollo distribuido y el uso de las herramientas CASE (*Computer Aided/Assisted Software/System Engineering*).

- A medida que se acercaban los años 80, la metodología orientada a objetos comenzaba a madurar como un enfoque de desarrollo de software.

- La metodología orientada a objetos presenta características que lo hacen idóneo para el análisis, diseño y programación de sistemas.

Modelado

- Para finales de los años 90, se puede decir que el modelado es una parte fundamental dentro de las actividades del desarrollo de software. Se construyen modelos para comunicar la estructura y el comportamiento deseado del sistema.

- El software se centra en optimizar al máximo el código, es decir, escribir menos líneas y conseguir el mejor rendimiento sin que disminuya la eficacia.

- A través del modelado se consiguen 4 objetivos:
  - Ayudar a visualizar cómo es o debería ser un sistema.
  - Especificar la estructura o el comportamiento de un sistema.
  - Proporcionar plantillas que guíen la construcción de un sistema.
  - Documentar las decisiones adoptadas.

- Principios del Modelado

1. El primer principio se basa en la profunda influencia que tiene el cómo abordar el problema y cómo se da la forma de solución.

2. El segundo principio básico del modelado nos dice que todo modelo puede ser expresado a diferentes niveles de precisión. Un analista o un usuario final se centrarán en el qué; un desarrollador se centrará en el cómo.

3. El tercer principio establece que los mejores modelos están ligados a la realidad.

4. Finalmente el cuarto principio establece que un único modelo no es suficiente, de manera que cualquier sistema se aborda mejor a través de un pequeño conjunto de modelos casi independientes.

Ejemplo

- Cuando se esta construyendo un edificio se requieren distintos tipos de planos: de planta, de alzado, eléctricos, calefacción, sanitarios, etc. Y estos se estudian de manera aislada pero en conjunto nos sirven para dar vida al sistema que todos tienen en común que es “el edificio”.

Perspectiva algorítmica contra *perspectiva OO*

- ***Descomposición algorítmica.*** La visión tradicional del desarrollo de software toma una perspectiva algorítmica. En este enfoque, el bloque principal de construcción de todo el software es el procedimiento o función.

- Esta jerarquía se organiza en torno a un programa principal, desde el cual se accede a otros módulos, llamados subrutinas o subprogramas.

- Los datos tienen un papel secundario, y no son más que aquello con lo que se alimenta a los programas para que realicen su función.

- La programación estructurada se puede resumir en la siguiente expresión:

  Algoritmos + Estructura de datos = Programas

- ***Descomposición orientada a objetos.*** La visión actual del desarrollo de software toma una perspectiva orientada a objetos. En este enfoque, el principal bloque de construcción de todos los sistemas software es el objeto o clase.

- En síntesis, la programación orientada a objetos puede ser resumida de la siguiente manera:

  Objetos + Flujo de mensajes = Programas

ADOO

- El Análisis y Diseño Orientado a Objetos (ADOO) es una técnica para modelar sistemas que se encarga de describir y modelar el sistema como si fuera un conjunto de objetos relacionados que interactúan entre sí.

- Los métodos orientados a objetos centran su atención en objetos y clases:
  - Un objeto es “una cosa”.
  - Una clase es “una plantilla o definición de cosas”.

- Los métodos orientados a objetos crean modelos con los cuales:
  - Se pueden crear clases y objetos.
  - Definen su estructura, comportamiento y propósito.
  - Definen la relación entre clases.
  - Definen la relación entre objetos.

La promesa de la OO

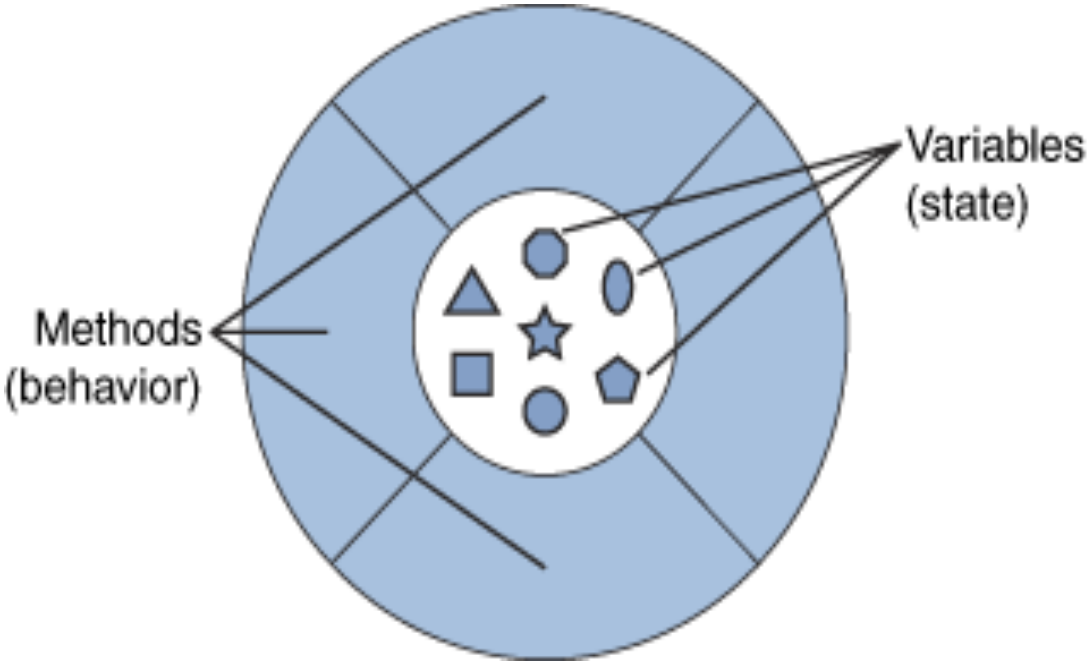
- Los requerimientos y la arquitectura pueden ser capturados en una manera repetida y razonable.
- Los objetos modelan el comportamiento del mundo real, y son fáciles de entender.
- Los componentes pueden ser reutilizados a través de interfaces bien definidas.
- El cambio de requerimientos es fácil de soportar.
- Los costos de mantenimiento son reducidos considerablemente.

Objetos

- Se define a un objeto como "*una entidad tangible que muestra alguna conducta bien definida*".

- Un objeto "*es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y hay métodos que hacen uso de dichos datos*".

- Los objetos tienen una cierta "integridad“, que es necesaria para que podamos interactuar con él sin confundirlo con otro, y para que puedan existir varios objetos de la misma clase.

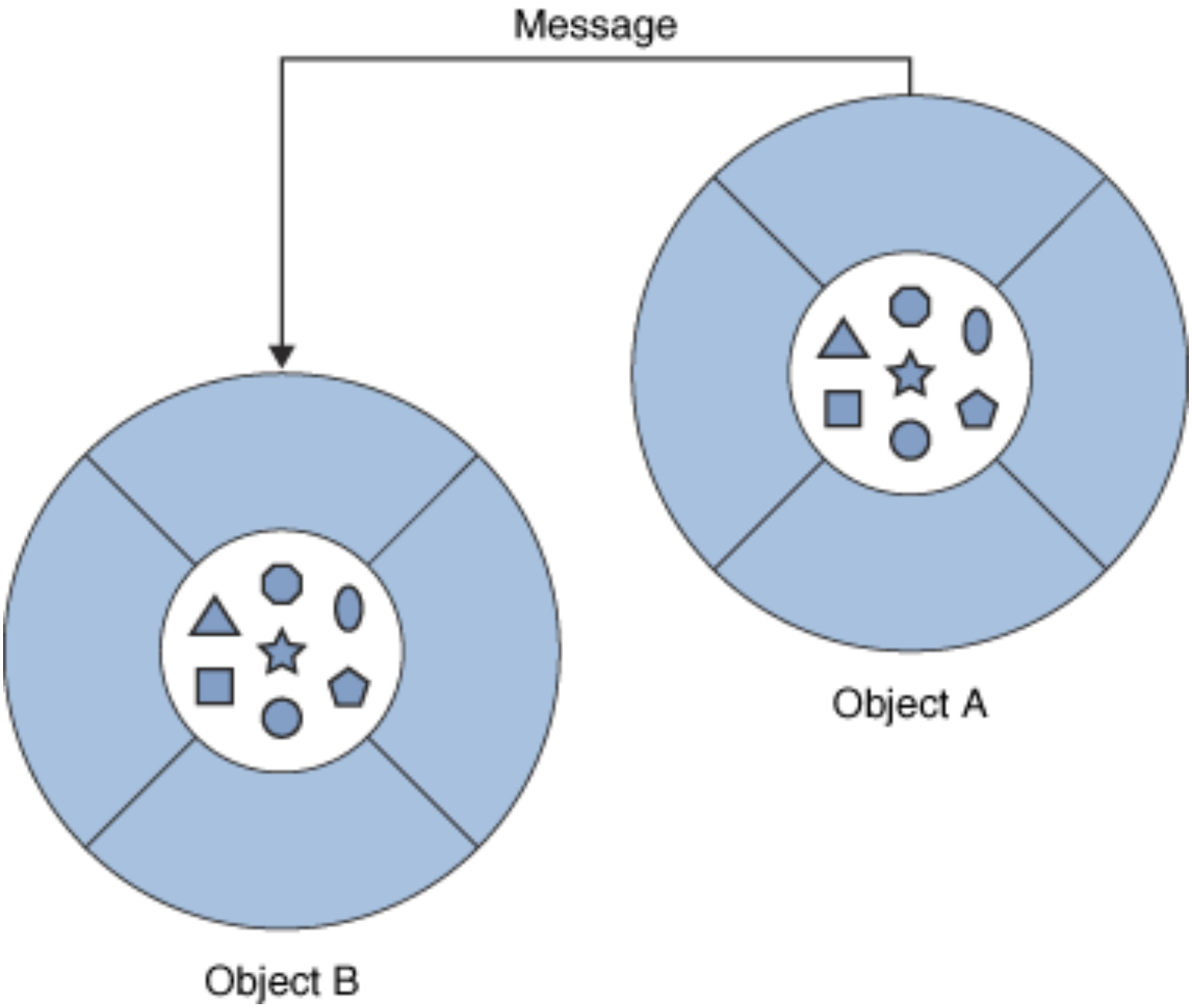


- Atributos
  - Son las características que definen a un objeto como único.

- Método
  - Es la acción de un objeto, representado generalmente por verbos. El nombre de un método debe indicar idealmente la función de un objeto.

- Mensaje
  - Los mensajes son simples llamadas a las funciones o métodos del objeto con el se quiere comunicar para decirle que haga cualquier cosa.

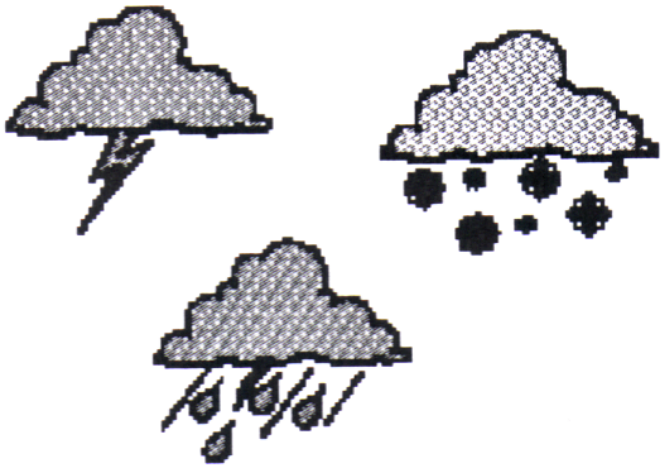
  - Un mensaje, es una acción que debe hacer algún objeto. Un objeto por sí solo no es muy útil, necesita interactuar para modelar algún problema.


Message
Object A
Object B

Ejemplos

- Atributos de las nubes: tamaño, sombra, contenido de agua.

- Operaciones de las nubes:llover, nevar, tronar.




www.esne.tv
ESNE


SOMOS IDEA
PARTIDO NACIONAL


etnies

Práctica

- 01_objetos. De acuerdo con el problema, revisar la información acerca del caso de estudio para encontrar objetos.

Abstracción

- Este concepto se refiere básicamente a describir una entidad en forma simplificada, a través de sus propiedades principales, omitiendo los detalles.
  - Ejemplo: Todo mundo sabe que un auto sirve para desplazarse sin importar cómo lo haga.
- La abstracción consiste en ignorar detalles y encontrar las principales características, de esta manera se simplifica cómo los usuarios interactúan con los objetos.

- Los tipos de abstracción son:
  - Abstracción funcional: Es lo que hace, acciones y eventos.
  - Abstracción de datos: Son las características y las propiedades.

- Finalmente los objetos que encontramos al analizar el dominio del problema y que tienen la misma estructura y comportamiento, forman parte de una clase. Así que lo que un objeto puede saber (estado) o hacer (comportamiento), está determinado por la clase a la que pertenece.

Práctica

- 02_abstracción: Encontrar todos los atributos y métodos de los objetos encontrados en el ejercicio anterior.

Encapsulación

- Esta característica se refiere a ocultar la información de los detalles de la implementación.

- El encapsulamiento es guardar o poner algo dentro de un objeto, de esta manera se separan los aspectos externos de un objeto, de los detalles internos de implementación que se ocultan en los objetos.

  - Ejemplo: Todo mundo sabe que para que un auto frene se requiere pisar el pedal de la izquierda, sin embargo sólo el mecánico conoce la implementación de los frenos.

- La encapsulación está formada por:
  - *Interfaz pública*: Son las operaciones que podemos hacer con un objeto para interactuar con él.
  - *Implementación*: Son las operaciones internas, qué se puede hacer o el propósito del objeto.
  - *Información interna*: Son los datos, atributos escondidos y que son necesarios para complementar la función.

- Los atributos y operaciones son miembros del objeto y pueden ser de tipo público o privado. Si el miembro es público, forma parte de la interfaz pública, de lo contrario forma parte de la implementación.

- En sistemas orientados a objetos todos los atributos son privados y pueden ser accedidos con operaciones públicas.

Práctica

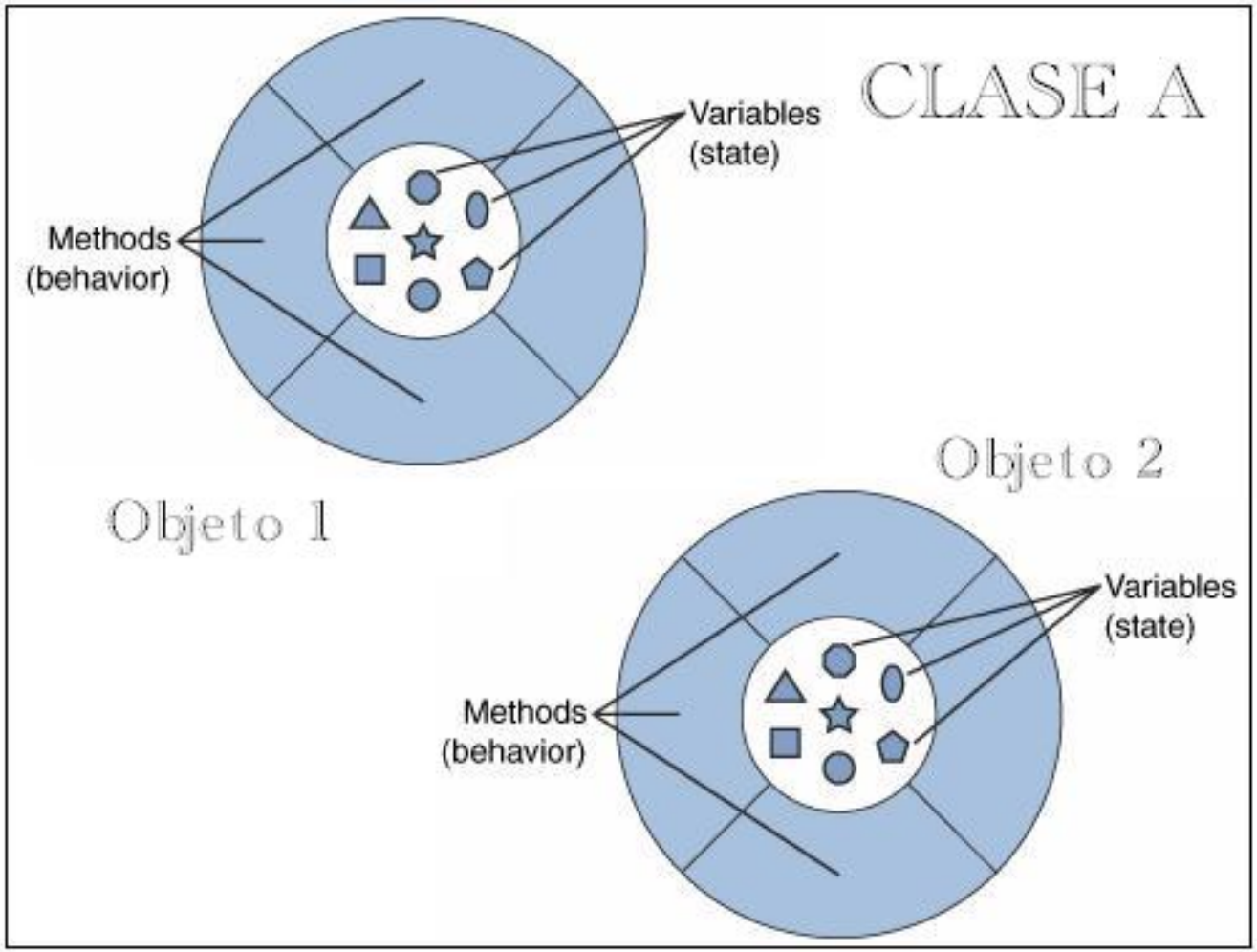
- 03_encapsulación: Agregar los métodos encapsulados que deben llevar los objetos a los se le agregó atributos y métodos en el ejercicio anterior.

Clases

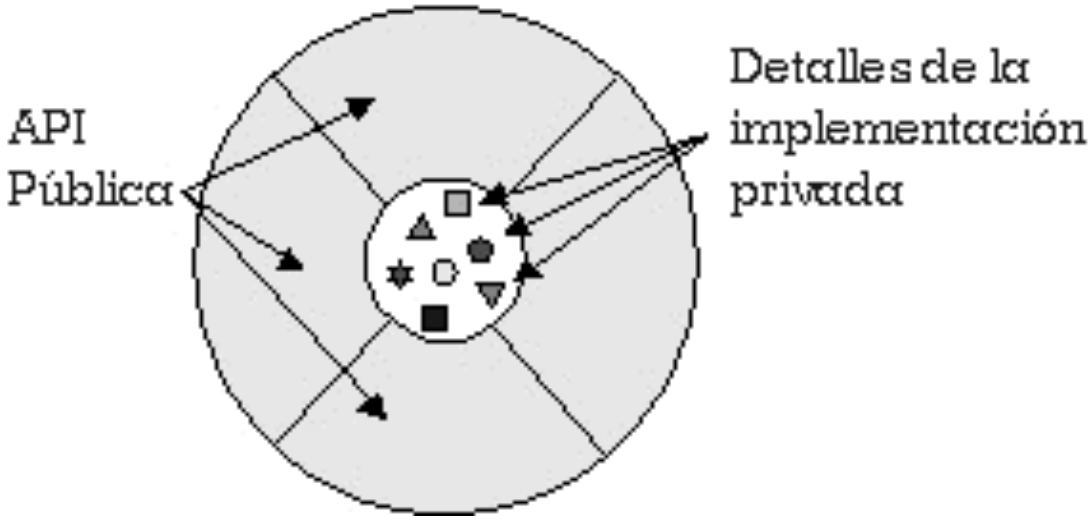
- Una *clase* es una plantilla para crear objetos con características similares. Las clases comprenden todas esas características de un conjunto particular de objetos.

- Cuando se escribe un programa en lenguaje orientado a objetos, no se definen objetos verdaderos sino se definen clases de objetos.

- Una *instancia* de una clase es otro término para un objeto real. Si la clase es la representación general de un objeto, una instancia es su representación concreta.


CLASE A
Variables
(state)
Methods
(behavior)
Objeto 1
Objeto 2
Variables
(state)
Methods
(behavior)

- En los lenguajes orientados a objetos cada clase está compuesta de dos cualidades:
  - *atributos* (estado)
  - *métodos* (comportamiento o conducta).



- Para definir el comportamiento de un objeto se crean métodos, los cuales tienen una apariencia y un comportamiento igual al de las funciones en otros lenguajes de programación.

- Los métodos no siempre afectan a un solo objeto; los objetos también se comunican entre sí mediante el uso de métodos.

- Una clase u objeto puede llamar métodos en otra clase u objeto para avisar sobre los cambios en el ambiente o para solicitarle a ese objeto que cambie su estado.

Práctica

- 04_clases: Para cada objeto identificado en los ejercicios anteriores, identificar las clases correspondientes a cada objeto.

Herencia

- Se puede crear una nueva clase u objeto heredando los atributos y métodos de una o varias clases padre (herencia simple y múltiple respectivamente).

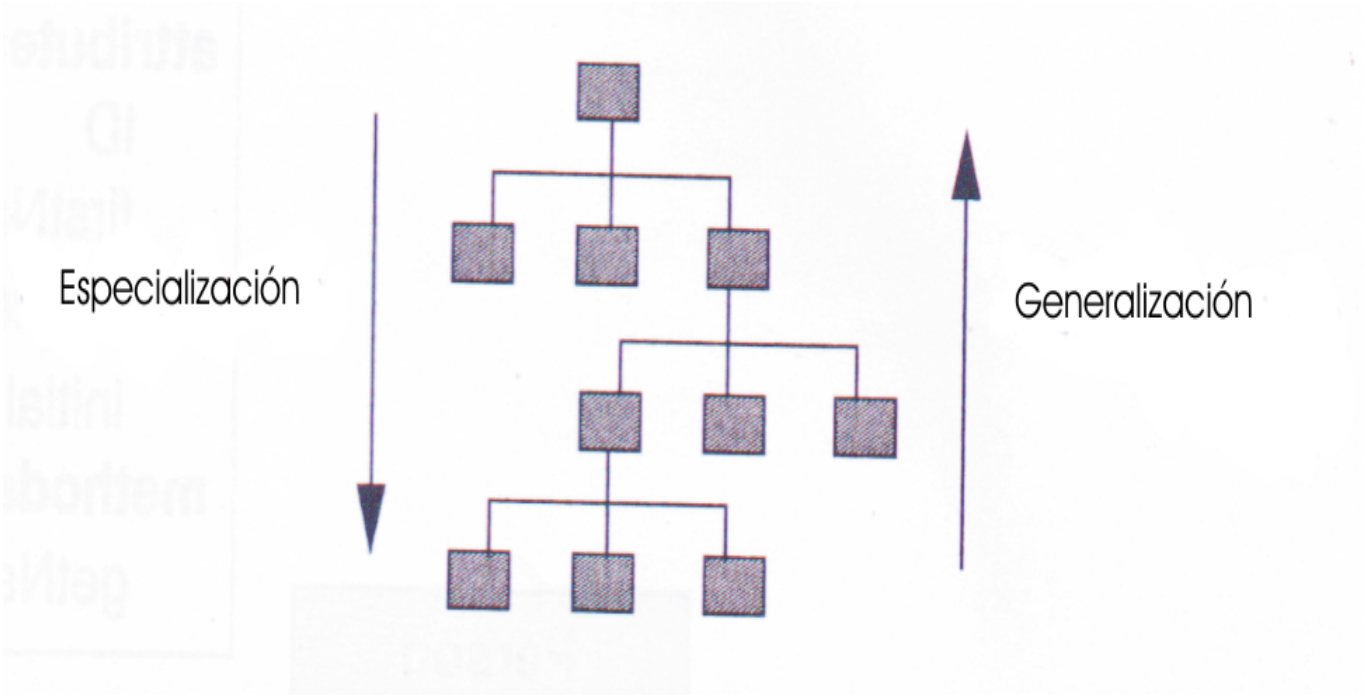
- Las nuevas clases que se van creando por herencia crean lo que se denomina como «jerarquía de clases».

- En este contexto se utiliza el término «superclase» para referir cualquiera de las clases de orden superior en una misma jerarquía.

- La herencia genera los siguientes beneficios:
  - Las subclases proveen conductas especializadas sobre la base de elementos comunes provistos por la superclase. A través del uso de herencia, los programadores pueden reutilizar el código de la superclase muchas veces.
  - Los programadores pueden implementar superclases llamadas clases abstractas que definen conductas "genéricas".

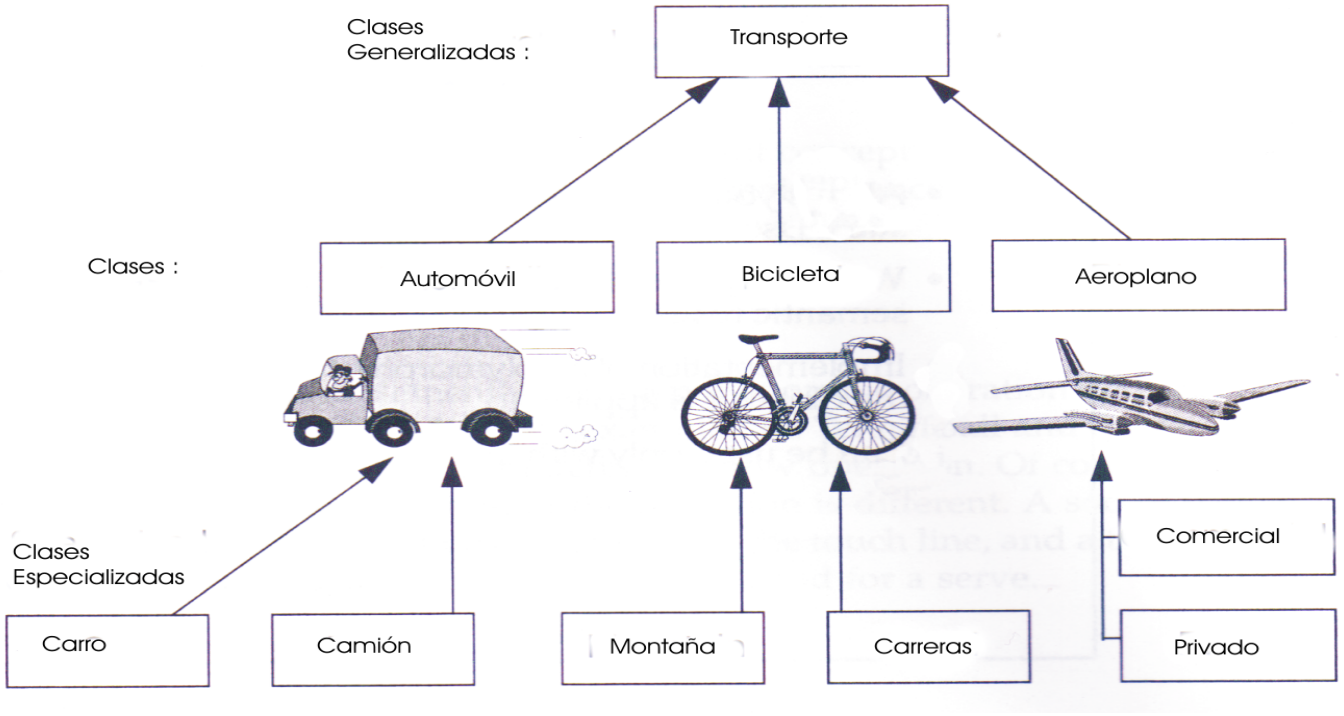
Generalización

- Se encarga de identificar y definir los atributos y operaciones comunes en una colección de objetos, de ésta manera se reduce la redundancia en el proceso de desarrollo y también ayuda a la reutilización.



Especialización

- La especialización se centra en crear nuevas clases más específicas que heredan de una clase mayor. La especialización es una herencia, la cual adiciona y modifica métodos para resolver un problema específico.



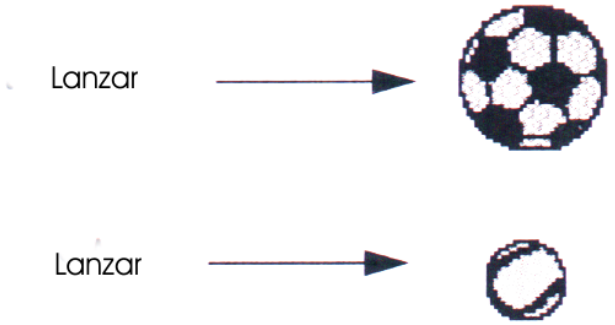
Práctica

- 05_herencia: Encontrar las superclases y subclases.

Polimorfismo

- El polimorfismo (varias formas) permite implementar una operación de herencia en una subclase (operaciones heredadas de una superclase).

- El polimorfismo significa que la operación existe en muchas clases, la operación tiene el mismo significado, pero cada clase personaliza la operación.



Práctica

- 06_polimorfismo: Encontrar los métodos polimórficos, si los hay.

Clases abstractas

- Una *clase abstracta* (*abstract*) es una clase de la que no se pueden crear objetos.

- Su utilidad es permitir que otras clases deriven de ella, proporcionándoles un marco o modelo que deben seguir y algunos métodos de utilidad general.

- Pueden tener una, todas o ninguna operación definida y se pueden tomar como plantilla para crear otras, pero no se puede crear una instancia de ellas.

Práctica

- 07_clases_abstractas: ¿qué clases pueden ser abstractas en el sistema?

UML

- UML es una técnica para la especificación sistemas en todas sus fases.

- Nació en 1994 cubriendo los aspectos principales de todos los métodos de diseño antecesores y, precisamente, los padres de UML son Grady Booch, autor del método Booch; James Rumbaugh, autor del método OMT e Ivar Jacobson, autor de los métodos OOSE y Objectory.

- La versión 1.0 de UML fue liberada en Enero de 1997 y ha sido utilizado con éxito en sistemas construidos para toda clase de industrias alrededor del mundo

- El UML es una notación gráfica y textual, rica y expresiva, que permite textos si las gráficas pudiesen hacerse difíciles de entender.

- No tiene un proceso, no se aplica como una receta de cocina.

- Cada diagrama de UML brinda una vista distinta del sistema en cuestión.

- El UML no tiene un ciclo de vida. Se utiliza en un ciclo iterativo e incremental.

Diagramas UML

- Los diagramas son las gráficas que describen el contenido de una vista.

- UML tiene nueve tipos de diagramas que son utilizados en combinación para proveer todas las vistas de un sistema.

- Diagrama de Casos de Uso
- Diagrama de Clases
- Diagrama de Objetos
- Diagramas de Comportamiento
  - Diagrama de Estados
  - Diagrama de Actividad
- Diagramas de Interacción
  - Diagrama de Secuencia
  - Diagrama de Colaboración
- Diagramas de implementación
  - Diagrama de Componentes
  - Diagrama de Despliegue

Diagramas de clase

- Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema.

- Un diagrama de clases esta compuesto por los siguientes elementos:

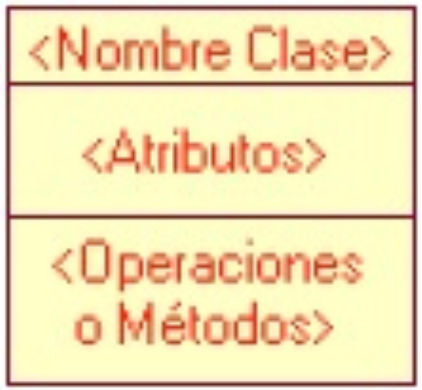
  - Clase: atributos, métodos y visibilidad.

  - Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

• Clase

- Es la unidad básica que encapsula toda la información de un objeto (un objeto es una instancia de una clase).

- A través de ella podemos modelar el entorno en estudio (una casa, un auto, una cuenta bancaria, etc.).



- **Atributos**

- Los atributos o características de una clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

  - **public** (+): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.

  - **private** (-): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden acceder).

  - **protected** (#): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de las subclases que se deriven.

- **Métodos**

- Los métodos u operaciones de una clase son la forma en cómo ésta interactúa con su entorno; éstos pueden tener las características:

  - **public** (+): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.

  - **private** (-): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden acceder).

  - **protected** (#): Indica que el método no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de métodos de las subclases que se deriven.

Relación entre clases

- Los objetos interactúan entre sí y esto lo hacen por medio del envío de mensajes.

- Se aplican el siguiente proceso:

    - Un objeto se encarga de enviar un mensaje a otro objeto.

    - El objeto receptor del mensaje, puede mandar a su vez más mensajes, reaccionando de acuerdo con el mensaje que está recibiendo y que es manejado por la interfaz pública de éste objeto.

- Un ejemplo de lo anterior es cuando una persona interactúa con la radio, la persona maneja la radio con su interfaz pública que es su sintonizador.

A través del sintonizador la persona manda mensajes a la radio para que cambie de frecuencia.

- Todos los objetos interactúan a través de diferentes relaciones como la asociación, agregación o composición.

- **Asociación**

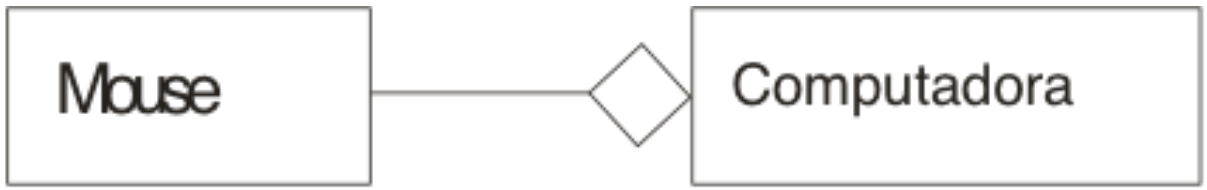
- Se produce entre dos objetos independientes que colaboran para realizar un objetivo. Un objeto se beneficia de los servicios del otro.

- Un ejemplo es cuando una persona “USA” una computadora. Esta es una relación débil.

- **Agregación**

- La agregación se produce entre dos objetos independientes, cuando uno de ellos utiliza al otro.

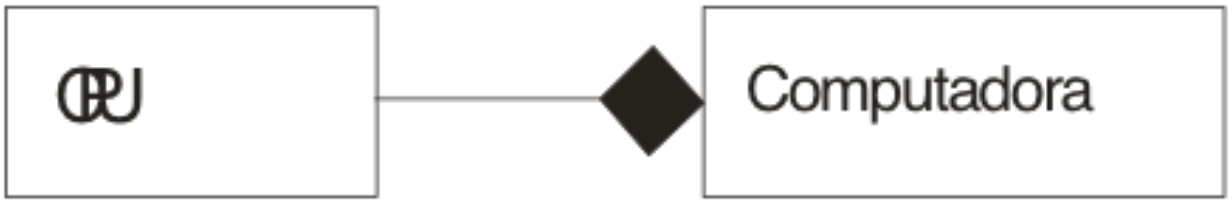
- Un ejemplo es cuando una computadora “TIENE UN” mouse. Esta es una relación regular.



- **Composición**

- La composición de dos objetos se produce cuando un esta compuesto por otro.

- Un ejemplo es cuando una computadora “ESTÁ COMPUESTA” o “SIEMPRE TIENE” un cpu. Esta es una relación fuerte.



- El tiempo de vida de los objetos depende del tipo de relación que estos presenten. Cuando dos objetos están asociados o agregados, su tiempo se traslapa.

- **Ejemplo:** Un aeroplano se compone de varias piezas, que juntas hacen que el aeroplano pueda volar, sin embargo el tiempo de vida del aeroplano termina cuando el tiempo de vida de las alas se termina, Pero a pesar de esto, los objetos que componen al aeroplano son creados en diferente tiempo y pueden morir de igual manera en otro tiempo.

Prácitca

- 08_diagrama_clases: realizar el diagrama de clases del problema propuesto.

Diagramas de Caso de Uso

- Un caso de uso representa la funcionalidad completa tal y como la percibe un actor.

- Un caso de uso en UML es definido como un conjunto de acciones que un sistema ejecuta y que permite un resultado observable por un actor en particular.

- ***Sistema***

- Un sistema es descrito como una caja; el nombre del sistema aparece arriba o dentro de la caja.

- Éste también contiene los símbolos para los casos de uso del sistema.

- ***Actores***

- Un actor es alguien o algo que interactúa con el sistema; es quien utiliza el sistema. Por la frase "interactúa con el sistema" se debe entender que el actor envía a o recibe del sistema unos mensajes o intercambia información con el sistema.
- En pocas palabras, el actor lleva a cabo los casos de uso. Un actor puede ser una persona u otro sistema que se comunica con el sistema a modelar.

- Un actor es un tipo (o sea, una clase), no es una instancia y representa a un rol. Gráficamente se representa con la figura de "stickman"

- Es posible obtener a los actores de un diagrama de casos de uso a través de las siguientes preguntas:
  - ¿Quién utilizará la funcionalidad principal del sistema (actores primarios)?
  - ¿Quién necesitará soporte del sistema para realizar sus actividades diarias?
  - ¿Quién necesitará mantener, administrar y trabajar el sistema (actores secundarios)?
  - ¿Qué dispositivos de hardware necesitará manejar el sistema?
  - ¿Con qué otros sistemas necesitará interactuar el sistema a desarrollar?
  - ¿Quién o qué tiene interés en los resultados (los valores) que el sistema producirá?

- El proceso para encontrar casos de uso inicia encontrando al actor o actores previamente definidos.
- Por cada actor identificado, hay que realizar las siguientes preguntas:
  - ¿Qué funciones del sistema requiere el actor? ¿Qué necesita hacer el actor?
  - ¿El actor necesita leer, crear, destruir, modificar o almacenar algún tipo de información en el sistema?
  - ¿El actor debe ser notificado de eventos en el sistema o viceversa? ¿Qué representan esos eventos en términos de funcionalidad?
  - ¿El trabajo diario del actor podría ser simplificado o hecho más eficientemente a través de nuevas

- Otras preguntas que nos ayudan a encontrar casos de uso pero que no involucran actores son:
  - ¿Qué entradas/salidas necesita el sistema? ¿De dónde vienen esas entradas o hacia dónde van las salidas?
  - ¿Cuáles son los mayores problemas de la implementación actual del sistema?

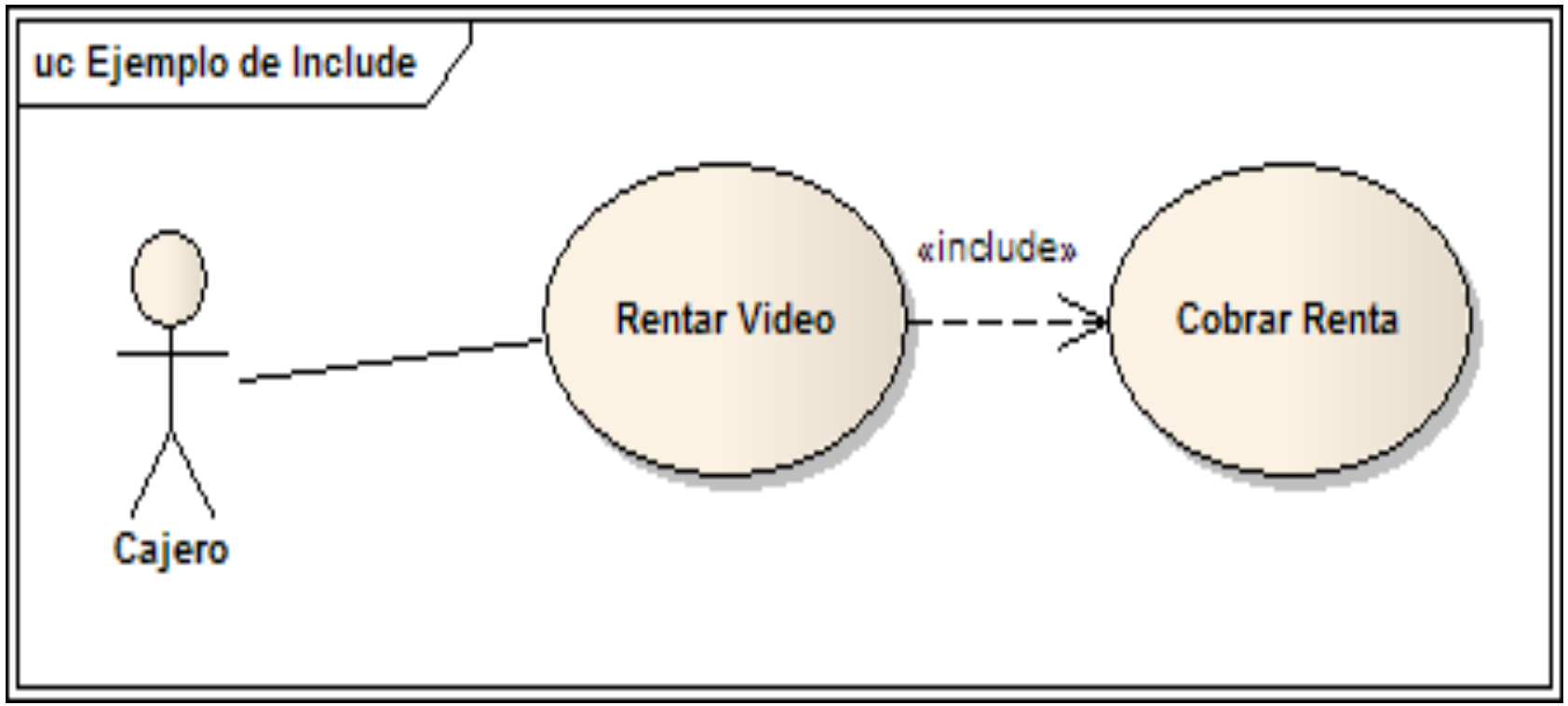
Include

- En términos muy simples, cuando relacionamos dos casos de uso con un “include”, estamos diciendo que el primero (el caso de uso base) incluye al segundo (el caso de uso incluído).

- El segundo es parte esencial del primero. Sin el segundo, el primero no podría funcionar bien; pues no podría cumplir su objetivo.

-

- El caso de uso “Cobrar Renta” está incluido en el caso de uso “Rentar Video”, o lo que es lo mismo “Rentar Video” incluye (<<include>>) “Cobrar Renta”.

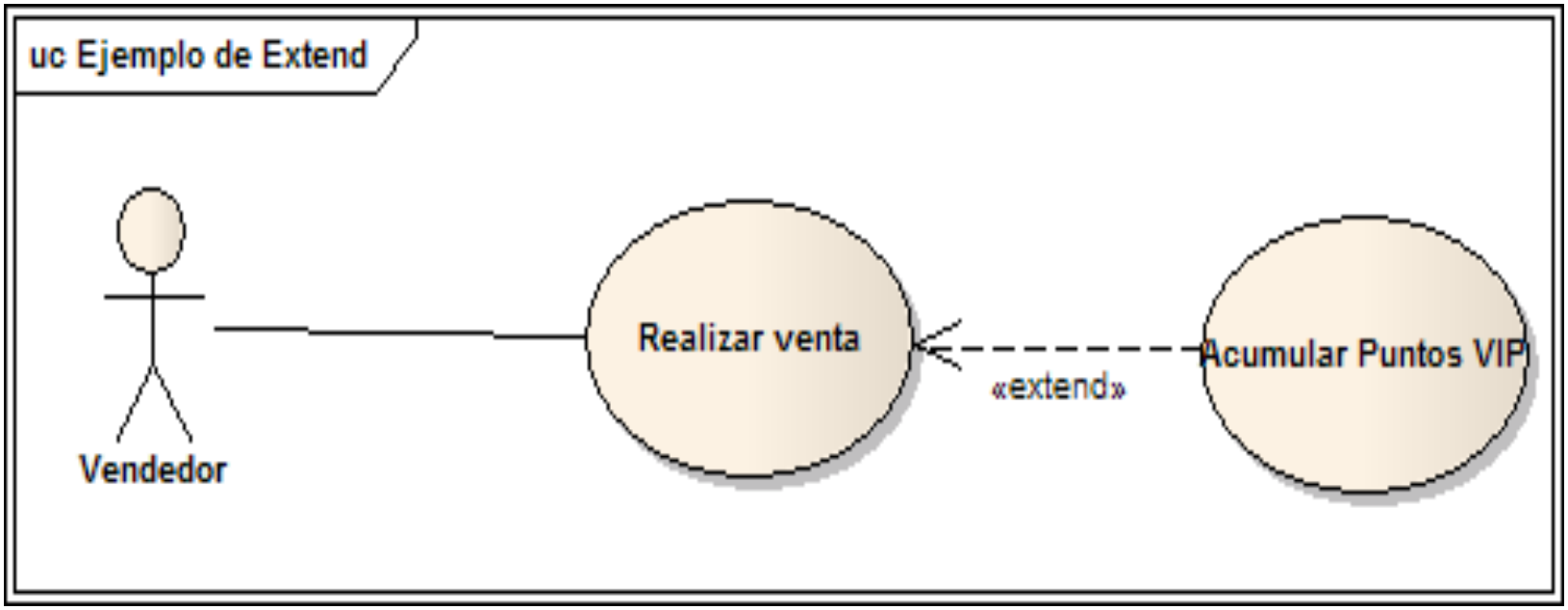


Extends

- En el caso del “extend”, hay situaciones en que el caso de uso de extensión no es indispensable que ocurra, y cuando lo hace ofrece un valor extra (extiende) al objetivo original del caso de uso base.

- Puedes “Realizar Venta” sin “Acumular Puntos de Cliente VIP”, cuando no eres un cliente VIP. Pero, si eres un cliente VIP sí acumularás puntos.

- Por lo tanto, “Acumular Puntos” es una extensión de “Realizar Venta” y sólo se ejecuta para cierto tipo de ventas, no para todas.



Malas prácticas

- Es importante comprender que el objetivo de estos tipos de relaciones NO consiste en motivar la división de los casos de uso en la mayor cantidad de pedazos.

- Debe de existir una razón importante para que decidamos dividir un caso de uso en dos que serán unidos por medio de alguna de estas relaciones.

- La razón por la que la gente suele partir sus casos de uso en infinidad de “include” y “extend” es porque quieren conocer, entender y comunicar el máximo detalle de los casos de uso en el diagrama.

- Hay quien llega a utilizar, erróneamente, estas relaciones para mostrar el orden en que se ejecutan estos casos de uso.

- Debemos de recordar que al modelar el diagrama de casos de uso no buscamos analizar el detalle, y mucho menos los flujos.

- Todo ese detalle lo podremos plasmar en otro tipo de diagramas, como los diagramas de interacción, de actividad, de estados, o simplemente un texto en una especificación.

Ejemplo

- 09_casos_de_uso: Elaborar los diagramas necesarios que representen la funcionalidad del problema propuesto.