Tutorial de UML

Introducción:

El Lenguaje de Modelamiento Unificado (UML - Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML entrega una forma de modelar cos as conceptuales como lo son procesos de negocio y funciones de sistema, además de cos as concretas como lo son escribir das es en un lenguaje determinado, esquemas de base de datos y componentes de software reusables.

Objetivos:

Entregar un material de apoyo que le permita al lector poder definir diagramas propios como también poder entender el modelamiento de diagramas ya existentes.

Audiencia:

El presente documento esta orientado a alumnos que ya poseen ciertos conceptos de OOP (o estan haciendo un curso) y a su vez conocen algún lenguaje Orientado a Objetos (ejemplo: C++ o Java), por lo tanto no es un curso en si, sino más bien un material de apoyo al estudiante.

Contenidos:

El documento contempla el estudio de tres diagramas:

- Modelamiento de Clases
- Casos de Uso
- Diagrama de Interacción

A su vez el estudio de un problema completo que los involucra (El Hotel), http://www.dcc.uchile.d/~ps.alinas/uml/ejemplo/ejemplo.html utilizando como herramienta case Rational Rose http://www.rational.com

Modelo de Clases

Introducción

Un diagrama de das es sirve para visualizar las relaciones entre las das es que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenimiento.

Un diagrama de das es esta compuesto por los siguientes elementos:

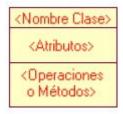
- <u>Clase</u>: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

E lementos

Clase

Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una dase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

En UML, una das e es representada por un rectángulo que pos ee tres divisiones:



Endonde:

- o **Superior**: Contiene el nombre de la Clase
- o **Intermedio**: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).
- Inferior: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

Ejemplo:

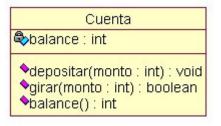
Una Cuenta Corriente que pos ee como caracterís tica:

Balance

Puede realizar las operaciones de:

- Depositar
- o Girar
- o y Balance

El diseño as ociado es:



Atributos y Métodos:

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 accsesible desde todos lados.
- * private (-, S): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden accesar).
- * **protected** (#, **): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de las subclases que se deriven (ver herencia).

Métodos:

Los métodos u operaciones de una das e son la forma en como 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 accesar).
- * protected (#, *): Indica que el método no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de métodos de las subclases que se deriven (ver herencia).

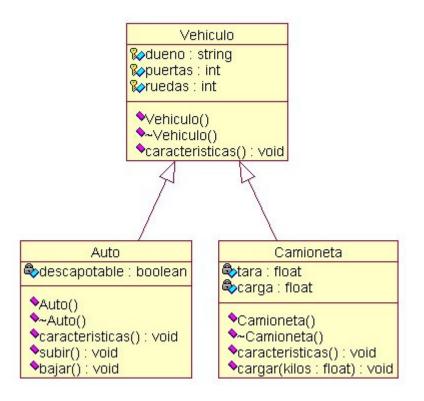
• Relaciones entre Clases:

Ahora ya definido el concepto de Clase, es necesario explicar como se pueden interrelacionar dos o más das es (cada uno con características y objetivos diferentes).

Antes es necesario explicar el concepto de cardinalidad de relaciones: En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:

- o **uno o muchos**: 1..* (1..n)
- o **0 o muchos**: 0..* (0..n)
- o **número fijo**: m (m denota el número).
- iv. Herencia (Especialización/Generalización):

Indica que una subdas e hereda los métodos y atributos es pecificados por una Super Clase, por ende la Subdas e además de pos eer sus propios métodos y atributos, pos eerá las características y atributos visibles de la Super Clase (public y protected), ejemplo:



En la figura se especifica que Auto y Camión heredan de Vehículo, es decir, Auto posee las Características de Vehículo (Precio, VelMax, etc) además posee algo particular que es Descapotable, en cambio Camión también hereda las características de Vehiculo (Precio, VelMax, etc) pero posee como particularidad propia Acoplado, Tara y Carga.

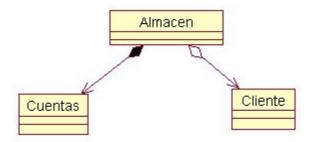
Cabe destacar que fuera de este entorno, lo único "visible" es el método Caracteristicas aplicable a instancias de Vehículo, Auto y Camión, pues tiene definición publica, en cambio atributos como Descapotable no son visibles por ser privados.

v. Agregación:

Para modelar objetos complejos, n bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de das es definidas por el des arrollador de la aplicación, tenemos dos posibilidades:

- ♣ Por Valor: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comunmente llamada Composición (el Objeto base se contruye a partir del objeto incluido, es decir, es "parte/todo").
- * Por Referencia: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comunmente llamada Agregación (el objeto base utiliza al incluido para su funcionamiento).

Un Ejemplo es el siguiente:



En donde se destaca que:

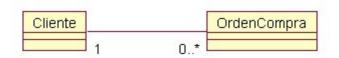
- * Un Almacen posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).
- Cuando se destruye el Objeto Almacen también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
- La composición (por Valor) se destaca por un rombo relleno.
- La agregación (por Referencia) se destaca por un rombo transparente.

La flecha en este tipo de relación indica la navegabilidad del objeto refereniado. Cuando no existe este tipo de particularidad la flecha se elimina.

vi. Asociación:

La relación entre das es conocida como Asociación, permite asociar objetos que colaboran entre si. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

Ejemplo:



Un diente puede tener asociadas muchas Ordenes de Compra, en cambio una orden de compra solo puede tener asociado un diente.

vii. Dependencia o Instanciación (uso):

Representa un tipo de relación muy particular, en la que una das e es instanciada (su instanciación es dependiente de otro objeto/das e). S e denota por una flecha punteada.

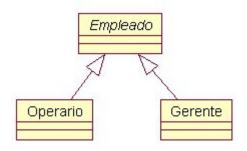
El uso más particular de este tipo de relación es para denotar la dependencia que tiene una dase de otra, como por ejemplo una aplicación grafica que instancia una ventana (la creación del Objeto Ventana esta condicionado a la instanciación proveniente desde el objeto Aplicación):



Cabe destacar que el objeto areado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo area (en este caso la Aplicación).

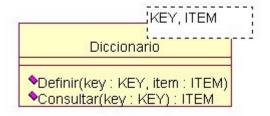
Casos Particulares:

o Clase Abstracta:



Una das e abstractas e denota con el nombre de la das e y de los métodos con letra "itálica". Esto indica que la das e definida no puede ser instanciada pues pos ee métodos abstractos (aún no han sido definidos, es decir, sin implementación). La única forma de utilizarla es definiendo subclas es, que implementan los métodos abstractos definidos.

o Clase parametrizada:



Una das e parametrizadas e denota con un subcuadro en el extremo superior de la das e, en donde se especifican los parámetros que deben ser pasados a la das e para que esta puedas er instanciada. El ejemplo más típico es el caso de un Diccionario en donde una llave o palabra tiene asociado un significado, pero en este caso las llaves y elementos pueden ser genéricos. La genericidad puede venir dada de un Template (como en el caso de C++) o bien de alguna estructura predefinida (especialización a través de das es).

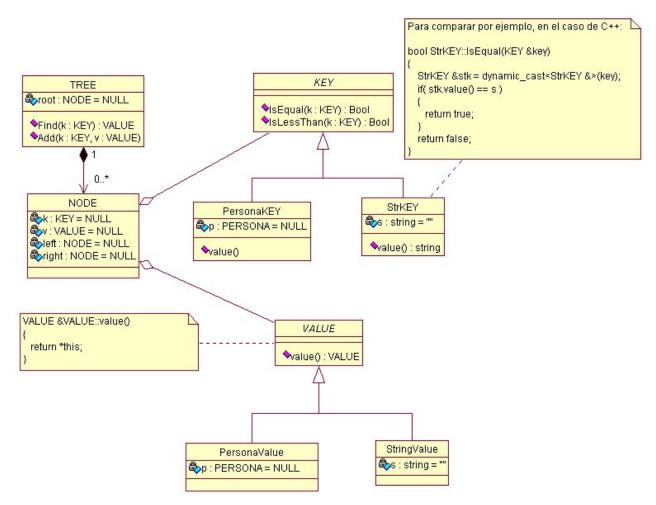
En el ejemplo no se especificaron los atributos del Diccionario, pues ellos dependerán exclusivamente de la implementación que se le quiera dar.

E jemplo:

S upongamos que tenemos tenemos un el caso del Diccionario implementado mediante un árbol binario, en donde cada nodo pos ee:

- key: Variable por la cual se realiza la búsqueda, puede ser generica.
- item: Contenido a almacenar en el diccionario asociado a "key", cuyo tipo también puede ser genérico.

Para este caso particular hemos definido un Diccionario para almacenar String y Personas, las cuales pueden funcionar como llaves o como item, solo se mostrarán las relaciones para la implementación del Diccionario:



Casos de Uso (Use Case)

Introducción

El diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactuan (operaciones o casos de uso).

Un diagrama de casos de uso consta de los siguientes elementos:

- Actor.
- Casos de Uso.
- Relaciones de Uso, Herencia y Comunicación.

E lementos

• Actor:



Una definición previa, es que un **Actor** es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema.

Como ejemplo a la definición anterior, tenemos el caso de un sistema de ventas en que el rol de Vendedor con respecto al sistema puede ser realizado por un Vendedor o bien por el Jefe de Local.

• Caso de Uso:



Es una operación/tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso.

• Relaciones:

o Asociación

Es el tipo de relación más básica que indica la invocación desde un actor o caso de uso a otra operación (caso de uso). Dicha relación se denota con una flecha simple.

Es una forma muy particular de relación entre das es, en la cual una das e depende de otra, es decir, se instancia (se area). Dicha relación se denota con una flecha punteada.

o Generalización

Este tipo de relación es uno de los más utilizados, cumple una doble función dependiendo de su estereotipo, que puede ser de **Uso** (<<use><>>) o de **Herencia** (<<extends>>).

Este tipo de relación esta orientado exclusivamente para casos de uso (y no para actores).

extends: Se recomienda utilizar cuando un caso de uso es similar a otro (características).

uses: Se recomienda utilizar auando se tiene un conjunto de características que son similares en más de un caso de uso y no se desea mantener copiada la descripción de la característica.

De lo anterior cabe mencionar que tiene el mismo paradigma en diseño y modelamiento de das es, en donde es ta la duda dásica de usar o heredar.

E jemplo:

Como ejemplo esta el caso de una Máquina Recidadora:

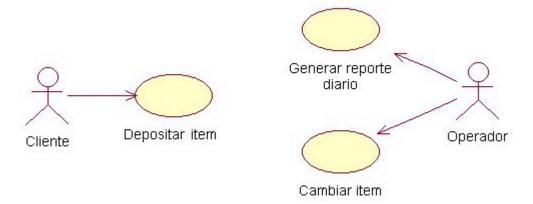
S is tema que controla una máquina de recidamiento de botellas, tarros y jabas. El sis tema debe controlar y/o aceptar:

- Registrar el número de ítemes ingresados.
- Imprimir un recibo cuando el usuario lo solicita:
 - a. Describe lo depositado
 - b. El valor de cada item
 - c. Total
- El usuario/cliente presiona el botón de comienzo
- Existe un operador que desea saber lo siguiente:
 - a. Cuantos ítemes han sido retornados en el día.
 - b. Al final de cada día el operador solicita un resumen de todo lo depositado en el día.
- El operador debe además poder cambiar:
 - a. Información asociada a ítemes.
 - b. Dar una alarma en el caso de que:
 - i. Item se atora.
 - ii. No hay más papel.

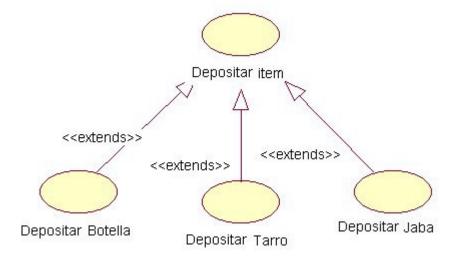
Como una primera aproximación identificamos a los actores que interactuan con el sistema:



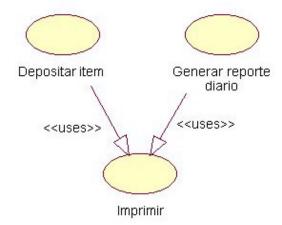
Luego, tenemos que un Cliente puede Depositar Itemes y un Operador puede cambiar la información de un Item o bien puede Imprimir un informe:



Además podemos notar que un item puede ser una Botella, un Tarro o una Jaba.



Otro as pecto es la impresión de comprobantes, que puede ser realizada después de depositar algún item por un diente o bien puede ser realizada a petición de un operador.



Entonces, el diseño completo del diagrama Use Case es:

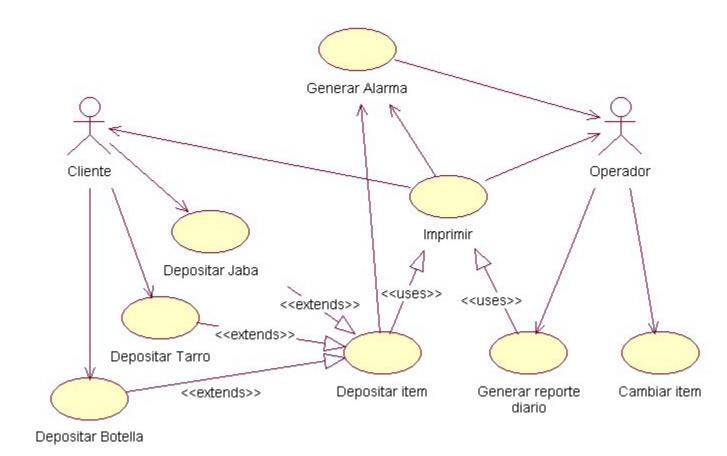


Diagrama de Interacción

Introducción

El diagrama de interacción, representa la forma en como un Cliente (Actor) u Objetos (Clases) se comunican entre si en petición a un evento. Esto implica recorrer toda la secuencia de llamadas, de donde se obtienen las responsabilidades daramente.

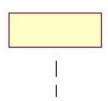
Dicho diagrama puede ser obtenido de dos partes, desde el Diagrama Estático de Clases o el de Casos de Uso (son diferentes).

Los componentes de un diágrama de interacción son:

- Un Objeto o Actor.
- Mensaje de un objeto a otro objeto.
- Mensaje de un objeto a si mismo.

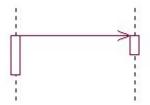
E lementos

• Objeto/Actor:



El rectángulo representa una instancia de un Objeto en particular, y la línea punteada representa las llamadas a métodos del objeto.

• Mensaje a Otro Objeto:



S e representa por una flecha entre un objeto y otro, representa la llamada de un método (operación) de un objeto en particular.

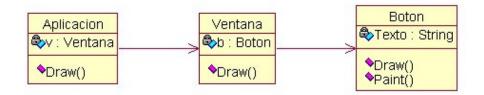
• Mensaje al Mismo Objeto:



No solo llamadas a métodos de objetos externos pueden realizarse, también es posible visualizar llamadas a métodos desde el mismo objeto en estudio.

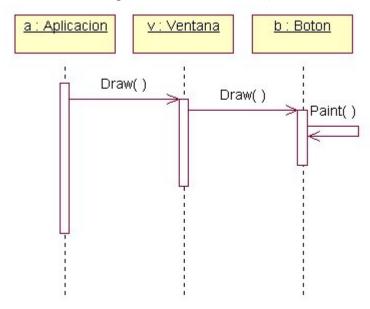
E jemplo

En el presente ejemplo, tenemos el diagrama de interacción proveniente del siguiente modelo estatico:



Aquí se representa una aplicación que posee una Ventana gráfica, y ésta a su vez posee internamente un botón.

Entonces el diagrama de interacción para dicho modelo es:



En donde se hacen notar las sucesivas llamadas a Draw() (entre objetos) y la llamada a Paint() por el objeto Botón.

Ejemplo - Tutorial UML "Hotel"

El dueño de un hotel le pide a us ted des arrollar un programa para consultar s obre las piezas disponibles y reservar piezas de su hotel.

El hotel posee tres tipos de piezas: simple, doble y matrimonial, y dos tipos de dientes: habituales y esporádicos. Una reservación almacena datos del diente, de la pieza reservada, la fecha de comienzo y el número de días que será ocupada la pieza.

El recepcionista del hotel debe poder hacer la siguientes operaciones:

- Obtener un listado de las piezas disponible de acuerdo a su tipo
- Preguntar por el precio de una pieza de acuerdo a su tipo
- Preguntar por el descuento ofrecido a los clientes habituales
- Preguntar por el precio total para un cliente dado, especificando su numero de RUT, tipo de pieza y número de noches.
- Dibujar en pantalla la foto de un pieza de acuerdo a su tipo
- Reservar una pieza especificando el número de la pieza, rut y nombre del cliente.
- Eliminar una reserva especificando el número de la pieza

El administrador puede usar el programa para:

- Cambiar el precio de una pieza de acuerdo a su tipo
- Cambiar el valor del descuento ofrecido a los clientes habituales
- Calcular las ganancias que tendrán en un mes especificado (considere que todos los meses tienen treinta días).

El hotel posee información sobre cuales dientes son habituales. Esta estructura puede manejarla con un diccionario, cuya dave sea el número de RUT y como significado tenga los datos personales del diente.

El diseño a desarrollar debe facilitar la extensibilidad de nuevos tipos de pieza o dientes y a su vez permitir agregar nuevas consultas.

El presente ejemplo contempla los siguientes diagramas:

- Casos de uso
- Diagrama de clases
- Dos diagramas de interacción

El contenido del presente ejemplo lo puedes obtener desde el siguiente archivo: hotel.zip (requerimientos: Rational Rose 98i).

CASOS DE USO

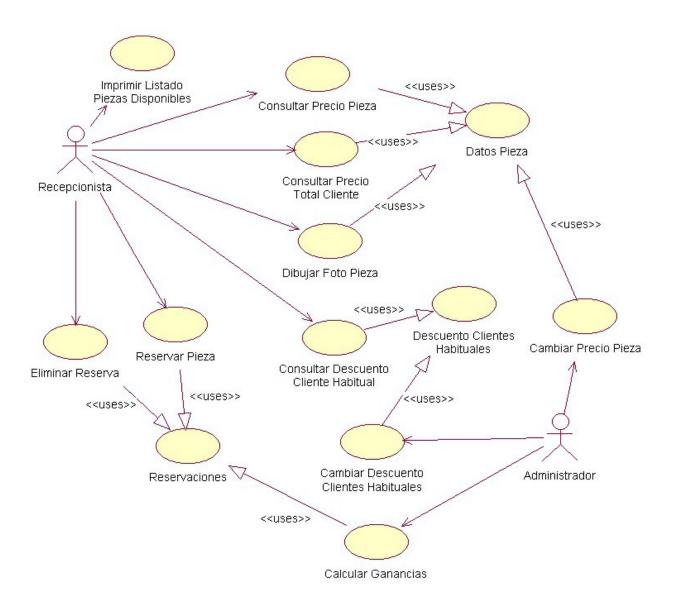
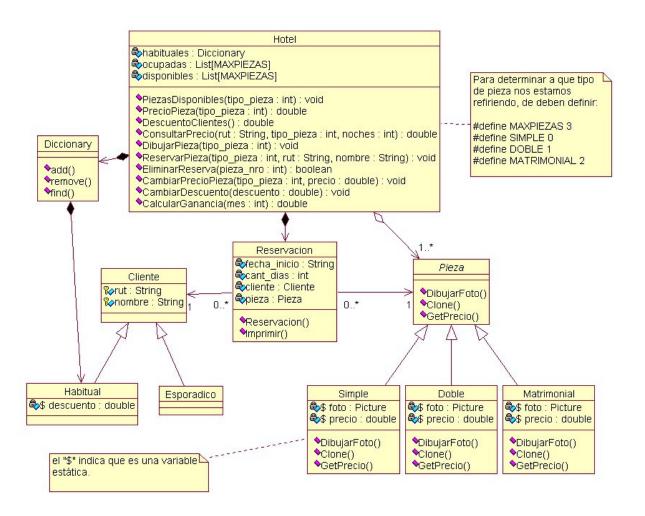


DIAGRAMA DE CLASES



```
El método Clone va orientado a poder obtener facilmente el Precio y la Foto de una pieza (para no perder generalidad).

Se crea un arreglo del siguiente tipo

Pieza *pieza = new Pieza[MAXPIEZAS];
pieza[0] = new Simple();
pieza[1] = new Doble();
pieza[2] = new Matrimonial();

Luego, por ejemplo para obtener el precio de alguna de ellas:

double Hotel::PrecioPieza(tipo_pieza : int)
{
    return pieza[tipo_pieza].Clone().GetPrecio();
}

Así, se asegura generalidad.
```

DIAGRAMA DE INTERACCION

