TECNOLOGÍA DE PROGRAMACIÓN

Modelamiento de Clases y Objetos

Dr. Diego C. Martínez

El lenguaje UML (en inglés, *Unified Modelling Language*) es un lenguaje para la especificación, visualización, construcción y documentación de las partes de un sistema de software. Consiste en una colección de las mejores prácticas de ingeniería que mostraron ser exitosas en el modelado de sistemas complejos, a tal punto que muchas empresas están actualmente incorporando UML para el desarrollo de sus productos. Fué creado en 1996, por el Object Management Group[1] con sucesivas modificaciones y agregados para permitir mayor funcionalidad, gracias al aporte y la participación de empresas como IBM, Hewlett Packard, Microsoft, Unisys y Oracle, entre otras.

UML es un lenguaje predominantemente visual, que consiste de varios diagramas, cada uno modelando una parte esencial del sistema a construir. La especificación completa de UML incluye doce diagramas, divididos en tres categorías. Cuatro diagramas son utilizados para representar la estructura estática de la aplicación en desarrollo, cinco diagramas representan diferentes aspectos del comportamiento dinámico y tres representan la forma en que se organizan los módulos de la aplicación.

Nos centraremos aquí únicamente en el *Diagrama de Clases*, el cual nos permite diseñar la estructura del programa en función de las clases que lo componen.

DIAGRAMAS DE CLASES

Clases

Las clases son dibujadas con un rectángulo, dividido en tres partes: el **nombre** de la clase, los **atributos**, y las **operaciones** correspondientes. Puede agregarse también una división en donde se especifican las **responsabilidades** de esa clase.

Nombre
Atributos
Operaciones
Responsabilidades

- **Nombre**: El nombre de la clase debe ser lo menos ambiguo posible, usualmente un sustantivo. Debe ser una palabra simple, y adoptaremos las mismas restricciones que las que se aplican para los identificadores de los lenguajes de programación como Pascal.
- **Atributos**: Los atributos describen las caracteristicas de los objetos y sus valores determinan el *estado* de los mismos. Poseen un *tipo*, que nos indica qué clase de atributo es. Si bien existen ciertos tipos primitivos, como *enteros*, *booleanos, reales y strings (cadenas)*, también se usan otras clases como tipos. La restricción más importante es que *los atributos son visibles únicamente por la clase que los contiene*. La sintaxis de declaración es la siguiente:

[1] www.omg.org

`<nombre>:<tipo>`

Recordar: *A nivel de diseño, por convención en nuestra materia, todos los atributos son considerados* ***privados****, esto es, no visibles desde otras instancias*.

- **Operaciones (servicios):** Las operaciones son utilizadas para manipular los atributos o realizar consultas. La sintaxis para describir una operación es la siguiente:

  `<nombre_operación> (<parámetros>) : <tipo_resultado>`

  A diferencia de los atributos, las operaciones pueden tener diferente *visibilidad* hacia otras clases, la cual se denota entre llaves a la izquierda de la declaración. Todas las operaciones están agrupadas de acuerdo a los estereotipos `<<comando>>`, `<<consulta>>` o `<<constructor>>` dependiendo del rol que cumplen en la clase. Algunas operaciones pueden ser mixtas, es decir, comportarse como *consulta* y *comando* al mismo tiempo. Por convención, en este caso se incluye la operación en la categoría `<<comando>>`.
  Como se mencionó anteriormente, los servicios pueden tener diferentes grados de visibilidad. En este caso usaremos una notación diferente a los estándares de UML: por medio de llaves, se indicarán las clases que pueden invocar a ese servicio. Por ejemplo

  `{Linea} Rho():real;`

  significa que sólo la clase `Linea` puede consultar el valor de *rho* en un punto bidimensional.
  Las llaves vacías `{}` indican que el servicio es absolutamente *privado*, y cuando no se escriben las llaves se indica que el servicio es totalmente *público*.

- **Responsabilidades:** Las responsabilidades son las obligaciones de una clase y son definidas por el usuario. Si bien existe un compartimiento dentro de la clase para la especificación de las responsabilidades, éstas son de carácter opcional.

Ejemplo:

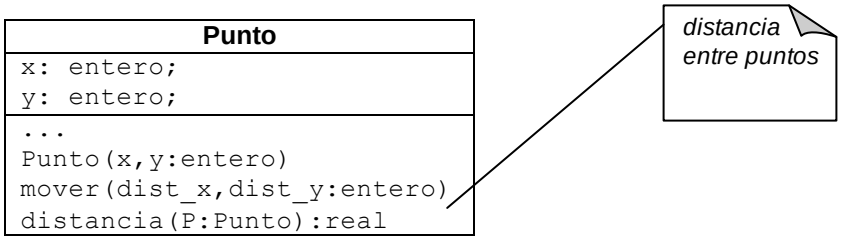
Punto
x: entero; y: entero;
<<constructor>> Punto(x,y:entero) <<comando>> mover(dist_x, dist_y:entero) <<consultas>> distancia(P:Punto):real Rho():real
Representa un punto en el plano

Recordemos que UML es una herramienta de diseño, por lo que a veces es necesario poner énfasis únicamente en algunas de las características de una clase. Dependiendo del nivel de detalle que querramos alcanzar en el modelo, se pueden obviar algunas de las divisiones del gráfico de clases. Esto permite simplificar el diagrama completo, de acuerdo al nivel de abstracción necesario. Puede utilizarse, por ejemplo:

Sólo el nombre	*Nombre y atributos*	*Nombre y operaciones*
Punto	**Punto** x: entero; y: entero;	**Punto** Punto(x,y:entero) mover(dist_x,dist_y:entero) ...

Notas

A los gráficos de clases se les puede agregar notas o comentarios sobre algún aspecto interesante de la clase. Estas notas pueden incluir observaciones sobre la clase, alguna restricción de uso, e incluso el pseudo-código de algunas operaciones. Se grafican por medio de un rectángulo con la esquina superior derecha plegada, como si fuera una hoja, y se une con una línea al elemento de la clase al cual corresponde la nota.



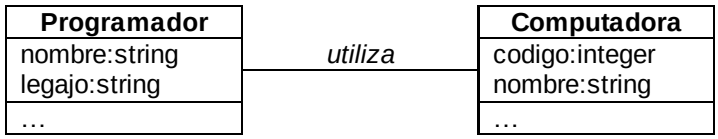
En este ejemplo se agrega un comentario sobre la operación `distancia`. Más adelante veremos cómo especificar a través de notas el código correspondiente a una operación de la clase.

Relaciones entre clases

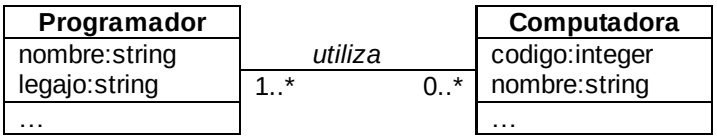
Al modelar el sistema, inevitablemente se construirán muchas clases con diferentes responsabilidades cada una. Muchas de ellas tendrán algún tipo de relación con otras clases en el sistema. Este hecho debe también reflejarse en el Diagrama, y para eso se utilizan notaciones especiales dependiendo del *tipo de relación* que las clases poseen.
Las relaciones entre clases más importantes que utilizaremos son *asociaciones*, *agregaciones*, y *generalización*. Para simplificar los gráficos, omitiremos en los siguientes diagramas de clases las declaraciones de las respectivas operaciones, pero se suponen definidas.

Asociaciones

Una asociación es una conexión entre clases. Significa que los objetos de dos clases tendrán un vínculo bidireccional en común, el cual puede interpretarse como “para cada X existe un Y”. Se representa por medio de una línea contínua entre dos clases. Por ejemplo,



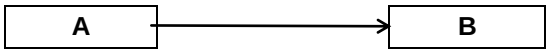
Aquí, cada programador utiliza una computadora, y cada computadora es utilizada por un sólo programador (es decir, un objeto *Programador* está asociado sólo a un objeto *Computadora*). S entiende que ambos objetos, si bien están asociados, tienen una existencia independiente. La asociación es una abstracción de la relación existente en los enlaces entre los objetos
Se utiliza *multiplicidad* cuando no necesariamente los vínculos entre objetos son de uno a uno. Se especifica en cada extremo del vínculo, a través de un rango, cuántos *objetos* pueden estar vinculados.



Este diagrama indica que cada *programador* tendrá varias *computadoras* (posiblemente ninguna), y que cada *computadora* será usada por al menos un *programador*. Algunos ejemplos de rangos son los siguientes:

0..1	0..* (o sólo *)	1..*	2	5..11
cero o uno	*cero o muchos*	*uno o muchos*	*dos*	*cinco a once*

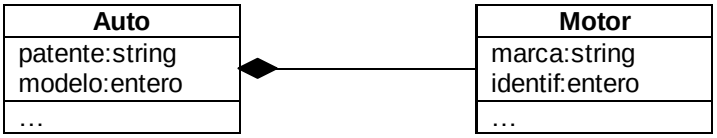
Cuando no se especifica ningún rango, se asume el valor uno (1) por defecto.
Puede agregarse una flecha a la línea, indicando la dirección del vínculo:



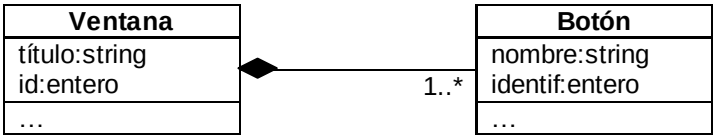
De esta manera, instancias de la clase A pueden enviar mensajes a instancias de la clase B, pero no al revés. Se dice también que objetos de tipo A tienen *conocimiento* o *acceso* a objetos de tipo B.

Agregaciones – composiciones

La *agregación* se considera, en términos generales, un caso especial de asociación. Indica que una o más clases (clases *componentes*) *forman parte* de otra clase (clase *agregada*). Se denota con un rombo en uno de los extremos del vínculo, aquél que corresponde a la clase agregada. Puede interpretarse como "posee (o contiene) un".



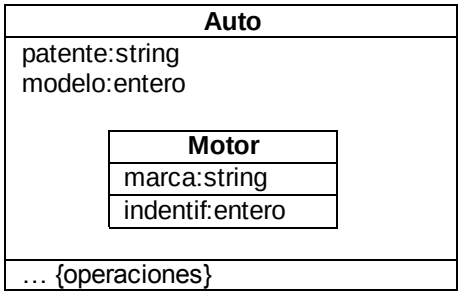
Este diagrama indica que todo *auto* posee un *motor*, de una marca particular, y con su correspondiente número de identificación. El motor forma parte estructural del auto. La clase Auto es la clase *agregada*, y la clase Motor es la clase *componente*.
En la agregación, la multiplicidad de la clase agregada debe ser cero o uno (0..1), pero en las clases que conforman las partes puede ser un rango cualquiera.



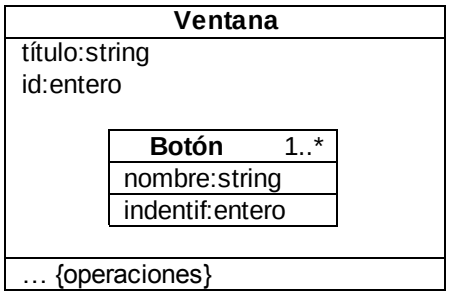
Aquí, el diagrama indica que una *ventana* de Windows puede contener uno o más *botones*. Asimismo, se indica que un botón pertenece a una ventana.

Otra posible forma de representar la *agregación* entre clases es incluir el gráfico de una clase (el agregado) dentro del gráfico de otra. Cuando las clases son muy grandes, por lo general esta notación no se utiliza. Sin embargo, cuando se quiere destacar alguna característica especial,

como los atributos, esta forma de graficar permite visualizar mejor una relación de composición. Por ejemplo,

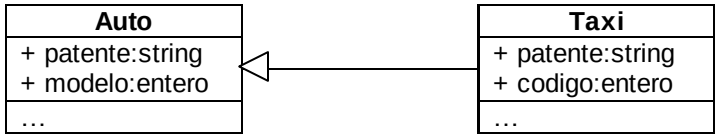


Si el agregado requiere especificar multiplicidad, escribimos el rango correspondiente en la esquina superior derecha de la clase.



Generalización

La generalización es la relación entre una clase más general y un clase más específica. La clase más especifica (denominada *subclase*) es consistente con la clase más general (denominada *superclase*), y contiene información adicional.
La generalización permite la especialización de las clases. Se denota con un triángulo en el extremo del vínculo correspondiente al elemento más general. La relación puede interpretarse informalmente como "*es un*".



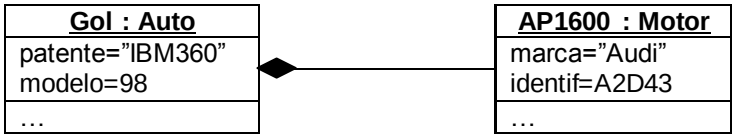
El diagrama indica que un *taxi* **es una** clase particular de *autos*, con una patente extra y un código utilizado por la empresa. Son *heredados* de la clase Auto todos los demás atributos, como la patente oficial, el modelo, el motor, etc.
Cuando una clase posee más de una subclase, pueden organizarse en forma de árbol de la siguiente manera:

DIAGRAMAS DE OBJETOS

Hasta ahora, sólo hemos mostrado clases en el modelo del sistema. Los objetos pueden modelarse también en un *diagrama de objetos*. Este diagrama utiliza la misma notación y relaciones que el *diagrama de clases*, puesto que los objetos son simplemente instancias de esas mismas clases. Mientras el diagrama de clases muestra las diferentes clases y sus relaciones, el diagrama de objetos muestra instancias específicas de estas clases, y los vínculos entre estas instancias en un momento dado.
Los objetos son mostrados como las clases, pero con el nombre del objeto subrayado y precedido del nombre de la clase que representa.
La sintaxis correspondiente es:

`<nombre del objeto>:<clase>`

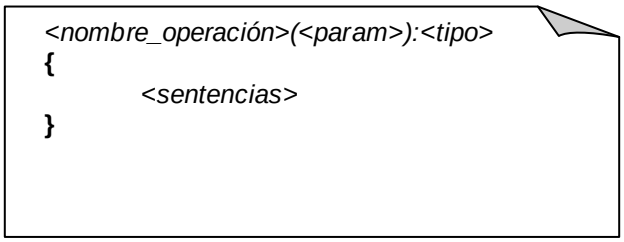
Por ejemplo, utilizando el diagrama de las clases *Auto* y *Motor*, una instancia particular de estas clases puede ser:



A veces el nombre del objeto no necesariamente debe existir, por lo que en ese caso utilizaremos sólo los dos puntos (:) y el nombre de la clase, por ejemplo **:Avion**

ESPECIFICACIÓN DE LAS OPERACIONES

En las *notas* que se agregan al diagrama de clases podemos explicitar la implementación de una o más operaciones de la clase. Para ello utilizaremos la siguiente convención:



En las notas también explicitamos los *invariantes de clase*. En este caso, la nota va unida por un vínculo al *rectángulo* que representa la clase.
En esta materia, el lenguaje que se utiliza para la especificación de las operaciones es el Lenguaje de Diseño de Algoritmos, (sentencias de asignación con ←, condicionales "*si..entonces..si no...*", ciclos "*repetir..hasta*" y "*repetir mientras..*", sentencia case, etc) con algunas operaciones especiales que se detallan a continuación.

OPERACIONES

Creación de objetos

Para la creación se utilizan los constructores de la clase

```
<nombre_del_objeto> ← !!<constructor>(<parámetros>)
```

Los signos de admiración dobles se denominan informalmente "*bang-bang*" e indican que se crea un objeto invocando el constructor mencionado.

Ejemplos

`L ← !!Linea()`
`p ← !!CrearPunto(1,3)`
`E ← !!CrearEmpleado('Homero',123,'Sector7G')`

Acceso a los servicios

```
<nombre_del_objeto>.<nombre_del_servicio>(<parametros>)
```

Notar que, aunque el servicio no posea parámetros, la invocación debe incluir los paréntesis. Cuando un objeto X invoca solicita un servicio del objeto Y, se dice también que el **objetoX envía un mensaje al objetoY**.

Ejemplos

`P.Rho()`
`Emp.Sueldo_correspondiente('A')`
`Actual.verificarCodigo(C)`

Asignación

Sintaxis:

```
Objeto1 ← Objeto2
```

Ejemplos
`Punto1 ← Punto2` `N ← Empleado.Nombre()`

La asignación se aplica tanto a referencias a objetos como a objetos de tipo **expandido** (por valor)**.** La siguiente tabla indica qué sucede para la operación X←Y en cada caso, X e Y como referencias o por valor.

X ← Y	*Y referencia*	*Y por valor*
X referencia	*X e Y referencian al mismo objeto*	*X es un nuevo objeto con los valores de Y*
X por valor	*Los valores de X se copian de los valores de Y*	*Los valores de X se copian de los valores de Y*

Todas las consultas deben devolver el resultado por medio de una sentencia de asignación al identificador reservado `Resultado`:

`Resultado` ← <expresión>

`Resultado` es considerada una referencia al objeto que se devuelve y se aplican las consideraciones de la tabla recién mostrada.

Comparación de objetos

Sintaxis:

```
Objeto1 = Objeto2
```

El operador (**=**) devuelve verdadero si la referencia de los dos objetos es **null**, o si los dos referencian al mismo objeto. Puede aplicarse también a objetos por valor (de tipo expandido). La siguiente tabla indica que ocurre en cada caso:

X = Y	*Y referencia*	*Y por valor*
X referencia	*VERDADERO si X e Y referencian al mismo objeto.*	*VERDADERO si los valores de Y son iguales a los valores del objeto que referenciado X*
X por valor	*verdadero si el objeto X es igual al referenciado por Y*	*VERDADERO si el objeto X es igual al objeto Y*

Clonación y Copia

Objeto1 ← ***clonar****(Objeto2)* {Objeto1 es la clonación del Objeto2}

*Objeto1.**copiar****(Objeto2* {Objeto1 se convierte en copia del Objeto2}

La diferencia entre la **clonación** y la **copia** es que la **clonación** involucra la creación de un objeto (en el ejemplo, *Objeto1*). La **copia** supone que los dos objetos involucrados ya han sido creados.

Objeto1 ← ***clonar_prof****(Objeto2)* {Objeto1 es la clonación de toda la estructura de objetos a la que hace referencia Objeto2}

*x.**igual****(y)* {Verdadero si x e y son referencias nulas o referencian objetos con idénticos campos}

*x.**igual_prof****(y)* {Idem anterior, pero explora la estructura formada por referencias a otros objetos}

OBSERVACIONES

Esta breve guía no es completa en cuanto al lenguaje UML. Recordemos que este lenguaje se utiliza para modelar completamente sistemas de cualquier complejidad. En particular, existen varios aspectos de los diagramas de clases que no figuran aquí y que por el momento no son de relevancia. A medida que se incluyan nuevos temas en la materia, se agregarán las

modificaciones o expansiones necesarias a los Diagramas de Clases, siempre que se considere necesario. Algunas otras características no serán mencionadas porque no serán aplicables durante el cursado de la materia.
Para los lectores motivados, en la siguiente sección figuran algunos links relacionados con UML.

ALGUNOS LINKS RELACIONADOS CON UML

- **UML en Object Management Group, la organización autora del lenguaje.**
  Contiene la especificación completa y tutoriales.
  http://www.omg.org/uml/

- **Meta-links.** *Contiene varios links a páginas relacionadas con UML. Tutoriales, ejemplos, FAQs y artículos generales.*
  http://www.cetus-links.org/oo_uml.html

- **UML Resource Center en Rational.** *Rational es una empresa de desarrollo de software ligada estrechamente a la creación de UML.*
  http://www.rational.com/uml/

- **Herramientas basadas en UML.** *Listado de productos para el desarrollo de software basados en el lenguaje UML, algunos de ellos de distribución gratuita.*
  http://www.objectsbydesign.com/tools/umltools_byPrice.html

- **Bibliografía UML.** *Listado de trabajos y publicaciones que hacen referencia a UML.*
  http://www.db.informatik.uni-bremen.de/umlbib/