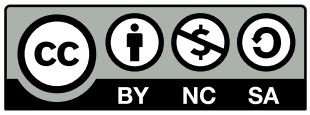
UML: Lenguaje de Modelado Unificado

Tema 2. Diagramas de clases (I)



UML: Lenguaje de Modelado Unificado

Tema 2. Diagramas de clases (I)

por: Javier Martín Juan

Revisión: `7bd97459bd82`

Última actualización: 20 de marzo de 2013

Índice

1. Diagramas de clases

Los diagramas de clases son un tipo de **diagramas estructurales**. Los diagramas estructurales describen una vista estática del sistema, a diferencia de los **diagramas de comportamiento**, que modelan el comportamiento del sistema a lo largo del tiempo.

Los diagramas de clases muestran las **clases** que componen el sistema, su estructura interna y sus **relaciones** con otras clases.

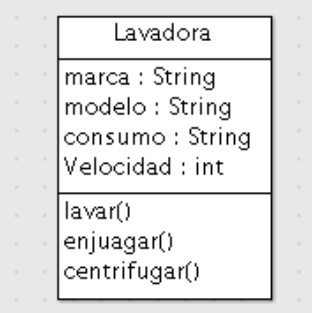
Clases y objetos

Una **clase** es una descripción de un conjunto de objetos con las mismas propiedades **(atributos)** y el mismo comportamiento **(operaciones)**.

En programación orientada a objetos, una clase es una plantilla que representa un concepto del mundo real y se utiliza para crear objetos. Por ejemplo, si pensamos en las lavadoras, podemos abstraerlas en una clase *Lavadora* junto con sus **atributos** (marca, modelo, consumo, velocidad) y sus **operaciones** (lavar, enjuagar, centrifugar).

A partir de una clase se pueden crear **objetos**, también llamados **instancias de la clase**. Los objetos son concreciones de una clase: todos los objetos de una clase comparten las mismas operaciones, pero sus atributos pueden tener valores distintos.

Las clases en UML se representan con un rectángulo dividido en 3 partes o "cajas": la caja superior contiene el nombre de la clase, la del medio los atributos, y la inferior las operaciones:



La lista de atributos

La caja central de la clase contiene la lista de atributos, uno por cada línea. La lista de atributos es opcional, así que se puede omitir si se desea dejando la caja en blanco. Los atributos tienen el siguiente formato:

```
nombre : tipo
```

En el ejemplo de la clase *Lavadora* tenemos atributos de dos clases distintas: la marca, el modelo y el consumo son de tipo *String*, mientras que la velocidad es de tipo *int*.

Lo normal es utilizar los tipos que nos proporciona el lenguaje de programación en el que vamos a escribir nuestra aplicación (*int, double, String,* etc. en el caso de Java), pero UML también permite modelar objetos del mundo real, y en ese caso no existe limitación alguna para los tipos de datos siempre y cuando tengan sentido para quien lo vaya a leer, por ejemplo podemos usar unidades como *minutos* o *euros*.

A veces es útil indicar el valor por defecto que tendrá un atributo cuando creamos la clase, por ejemplo el saldo de una cuenta corriente empezaría en 0. En UML esto se expresa con la notación:

```
nombre : tipo = valor por defecto
```

Por ejemplo:

```
saldo : Euros = 0
```

En cualquier caso, el valor por defecto es opcional y puede omitirse.

La lista de operaciones

La caja inferior de la clase contiene la lista de operaciones, que también es opcional pudiendo dejarse en blanco si se desea. Las operaciones se expresan con la siguiente notación:

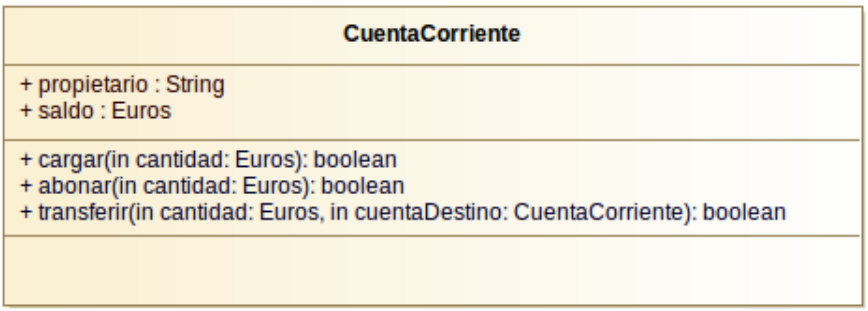
```
nombre (lista de parametros) : tipo del valor de retorno
```

La lista de parámetros se expresa del siguiente modo:

```
direccion1 nombre1 : tipo1, direccion2 nombre2 : tipo2, ...
```

La dirección es opcional y puede ser *in, out* o *inout* si el parámetro es de entrada, salida o bidirecciónal respectivamente. Nombre y tipo indican el nombre y el tipo del parámetro respectivamente.

La siguiente figura muestra una clase de ejemplo: *CuentaCorriente* con 3 operaciones: *cargar, abonar* y *transferir*. Las 3 operaciones devuelven un valor de tipo *boolean*. Las operaciones *cargar* y *abonar* toman como único parámetro de entrada la *cantidad* de tipo *Euros*. La operación *transferir* toma 2 parámetros, ambos de entrada: *cantidad* de tipo *Euros* y *cuentaDestino*, de tipo *CuentaCorriente*:



Visibilidad

El acceso a métodos y atributos de objetos de otras clases depende de la **visibilidad**. Hay tres visibilidades distintas. Ordenadas de mayor a menor, son:

- **Pública (public)**: Se representa con el signo + antepuesto al nombre de la operación o atributo. Cualquier clase puede acceder a cualquier atributo u operación declarado como público.
- **Protegida (protected)**: Se representa con el signo #. Sólo pueden ver el atributo / operación protegidos los elementos de la propia clase o las clases hijas.
- **Privada (private)**: Se representa con el signo -. Sólo pueden ver el atributo / operación privados los elementos de la propia clase.

Incluso en con la menor visibilidad (privada), un objeto siempre puede acceder a cualquier atributo u operación de la clase a la que pertenece y usarlo sin restricción.

Las definiciones de visibilidad de UML son idénticas a las de **Java**, lo que permite una traducción directa del diagrama a código y viceversa.

Sin embargo, nótese la diferencia de nomenclatura entre UML y Java: el término **operaciones** de UML equivale a **métodos** o **funciones miembro** en la mayoría de lenguajes de programación. Los **atributos** de una clase suelen llamarse **variables miembro**, aunque en Java también nos podemos referir a ellos como **atributos**, igual que en UML.

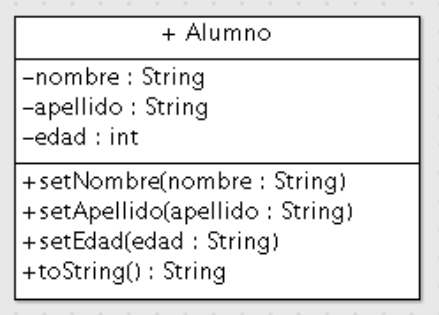
Ejemplo: Alumno

Vamos a ilustrar la visibilidad con un fragmento de código Java y su diagrama de clases equivalente. La clase *Alumno* que se muestra más abajo contiene 3 atributos: *nombre, apellido* y *edad*, de tipos *String, String* y *int* respectivamente.

Los 3 atributos son privados y tienen sus 3 métodos *setters* públicos correspondientes: *setNombre, setApellido* y *setEdad*. También hay un método público *toString* que convierte nuestro objeto *Alumno* en una cadena de texto.

```
class Alumno {
  // atributos
  private String nombre;
  private String apellido;
  private int edad;

  // metodos de acceso
  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
  public void setApellido(String apellido) {
    this.apellido = apellido;
  }
  public void setEdad(int edad) {
    this.edad = edad;
  }
  public String toString () {
    return(nombre + " tiene " + edad + " a~nos.");
  }
}
```

Veamos cuál sería la representación de la clase *Alumno* en UML:



El modificador *static*

En UML, al igual que en Java, el modificador *static* se puede aplicar a atributos y a operaciones:

- Cuando se aplica el modificador *static* a un método significa que dicho método se puede utilizar sin hacer un *new* de la clase, es decir, sin instanciarla. Un ejemplo de método estático es el punto de entrada a una aplicación Java: *main()*.

  En general, cuando un método no necesita acceder a variables del objeto y sólo depende de los parámetros que se le pasen, se puede poner como *static*.

- Cuando se aplica *static* a un atributo (constante o variable), éste se comparte entre todos los objetos de la misma clase, y además se puede utilizar sin necesidad de instanciar la clase.

A los atributos (variables o constantes) y métodos que llevan el modificador *static* se les llama **atributos estáticos** o **atributos de clase** y **métodos estáticos** o **métodos de clase**.

A los atributos y métodos que no llevan *static* (el comportamiento por defecto), se les llama **atributos de instancia** y **métodos de instancia**.

Ejemplo: atributos y métodos estáticos

Matematicas.java:

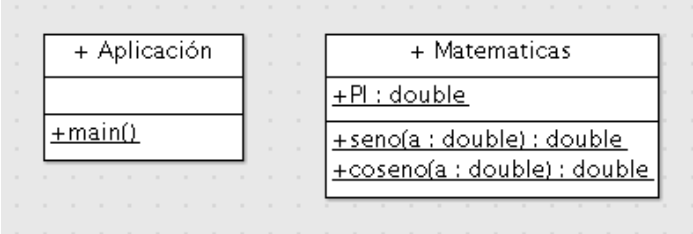
```
public class Matematicas {
  // Constante estatica
  public static final double PI = 3.1416;

  // Metodos estaticos
  public static double seno(double a) {
    ...
  }

  public static double coseno(double a) {
    ...
  }
}
```

Aplicacion.java:

```
public class Aplicacion {
  // main es siempre estatico
  public static void main(String args[]) {
    System.out.println("PI vale " + Matematicas.PI);
    System.out.println("Coseno de 30: " + Matematicas.coseno(30));
  }
}
```

En este ejemplo, los métodos *seno* y *coseno* únicamente necesitan el parámetro *a* para operar, así que se han declarado como estáticos. Si no fueran estáticos, habría que instanciar inútilmente un objeto de la clase *Matematicas* para poder llamarlos. Lo mismo pasaría con la constante *PI*.

Los métodos y atributos estáticos se indican en el diagrama de clase subrayando el elemento en cuestión. La representación en UML del código anterior sería:



Estereotipos

Los **estereotipos** son un mecanismo que permite extender el lenguaje UML añadiendo información extra a los elementos del diagrama. Por ejemplo, el estereotipo «Create» aplicado a una operación indicaría que esa operación es el constructor de la clase.

Los estereotipos se pueden aplicar a cualquier elemento del diagrama (clases, operaciones, atributos, relaciones, etc.) y se expresan con un nombre entre comillas francesas colocado encima o cerca del nombre del elemento: «*estereotipo*». Gráficamente también podemos representar el elemento estereotipado con un icono específico y distinto al original, aunque esto es opcional.

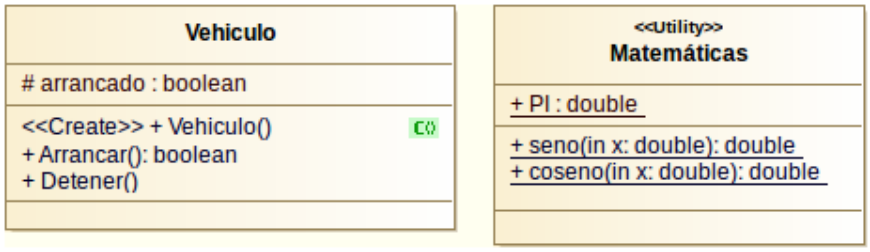
UML tiene varios estereotipos ya predefinidos, pero podemos inventarnos nuevos, por ejemplo en un diagrama de componentes de red podríamos distinguir los switches de los hubs añadiendo los respectivos estereotipos: «switch» y «hub».

Algunos de los estereotipos estándar de UML aplicables a los diagramas de clases son:

Estereotipo	Aplicado a	Significado
«Metaclass»	Clase	Las instancias de esta clase son también clases
«Utility»	Clase	La clase es una colección de métodos estáticos, no hace falta instanciarla
«Create»	Operación	La operación es un constructor
«Destroy»	Operación	La operación es un destructor

Ejemplo: estereotipos

El siguiente diagrama ilustra el uso de estereotipos en clases y operaciones:



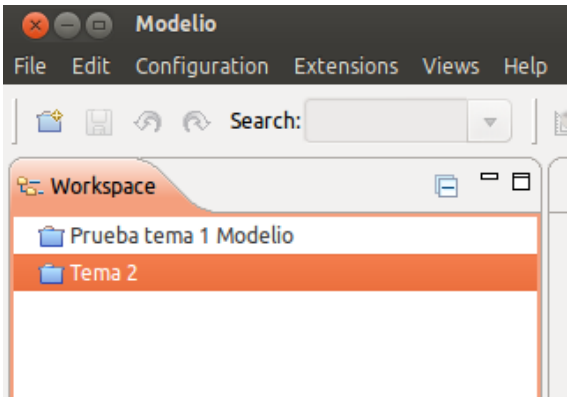
- La operación *Vehiculo* de la clase *Vehiculo* tiene el estereotipo «Create» por ser el constructor.
- La clase *Matematicas* tiene el estereotipo «Utility» porque es una clase compuesta únicamente por atributos y operaciones estáticas.

Nota: por defecto Modelio muestra los estereotipos de las operaciones como un icono y en las clases no los muestra. Para que aparezcan y el diagrama quede igual que en el ejemplo anterior, debemos hacer los siguientes cambios en la vista *Symbol*:

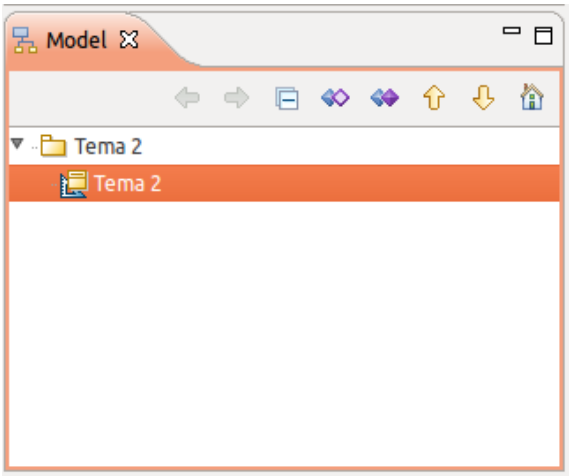
- Propiedad *Class* →*Stereotypes display mode: Text + icon*
- Propiedad *Class - Operations* →*Stereotypes display mode: Text + icon*

2. Ejercicio guiado

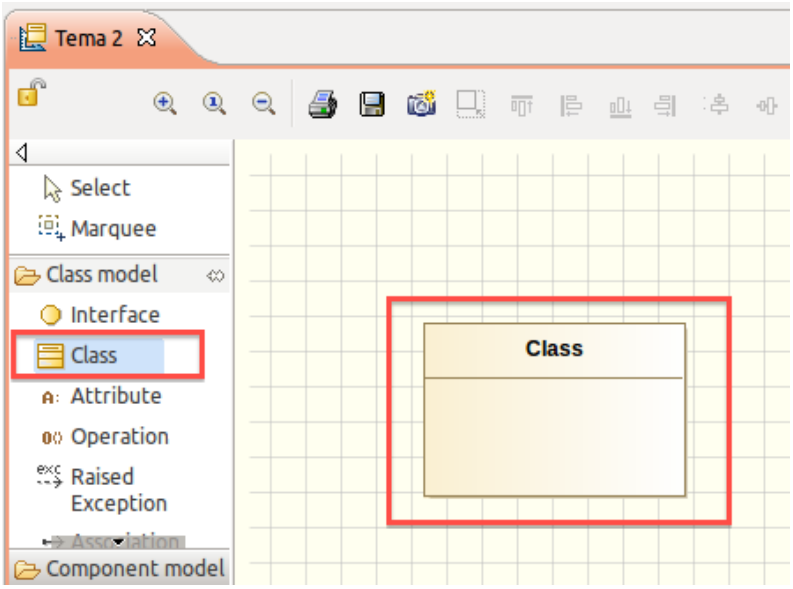
Vamos a crear el diagrama de clases visto en el ejemplo de la página 9. Lo primero es abrir Modelio. Podemos crear un proyecto nuevo o abrir uno existente dándole doble clic encima:



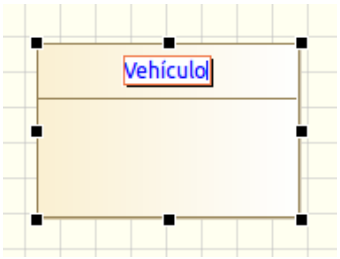
Para abrir el diagrama de clases, en la vista *Model* hacemos doble click en el diagrama:



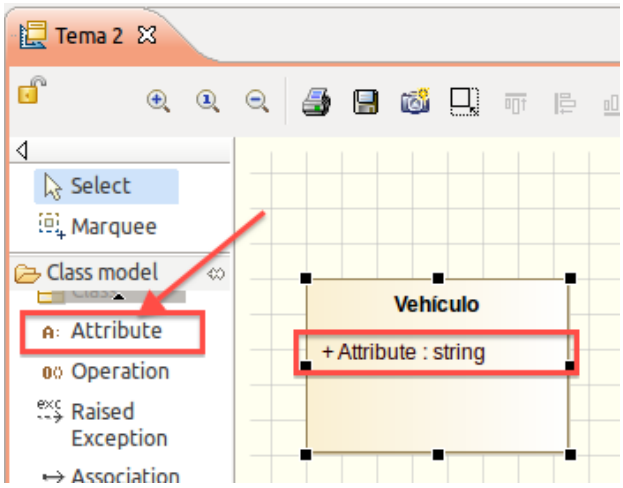
Creamos una clase seleccionando *Class* en la paleta de elementos y luego pinchando en el diagrama:



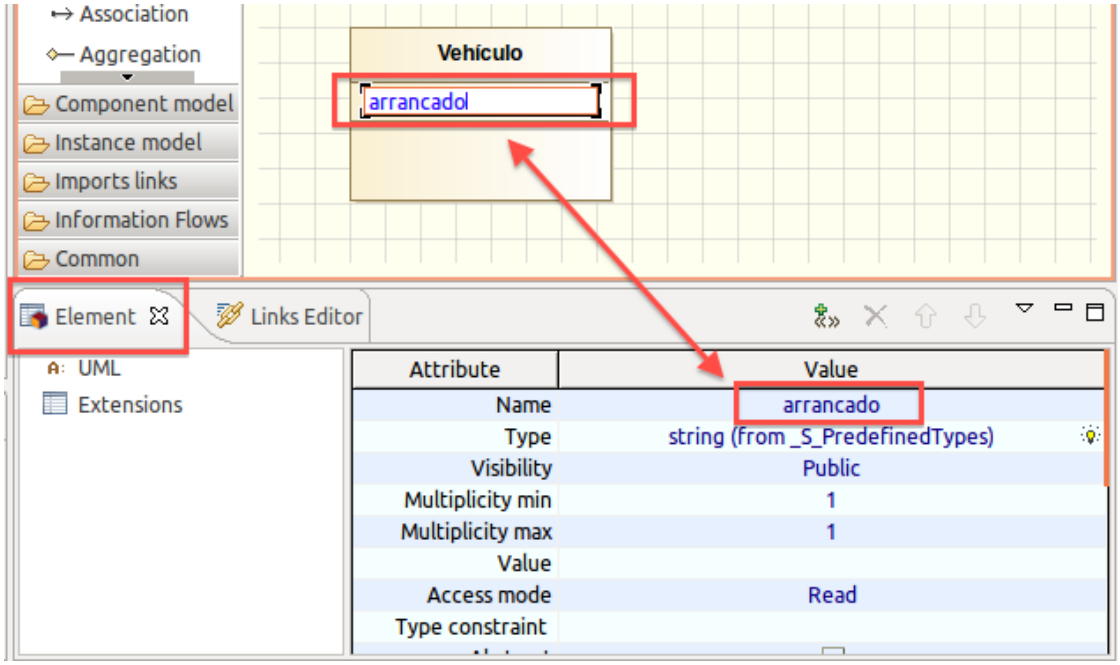
Renombramos la clase apretando en el nombre y dándole a F2:



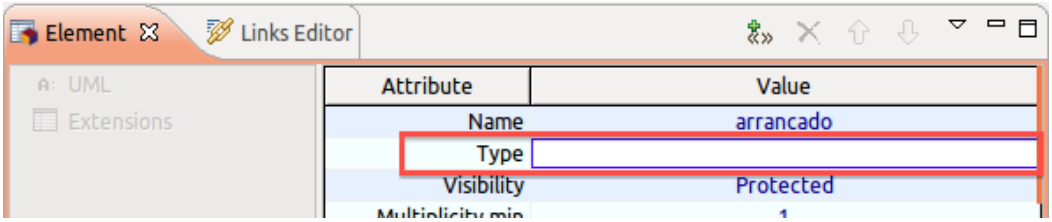
Vamos a añadir un atributo: seleccionamos *Attribute* en la paleta de elementos y pinchamos encima de la clase:



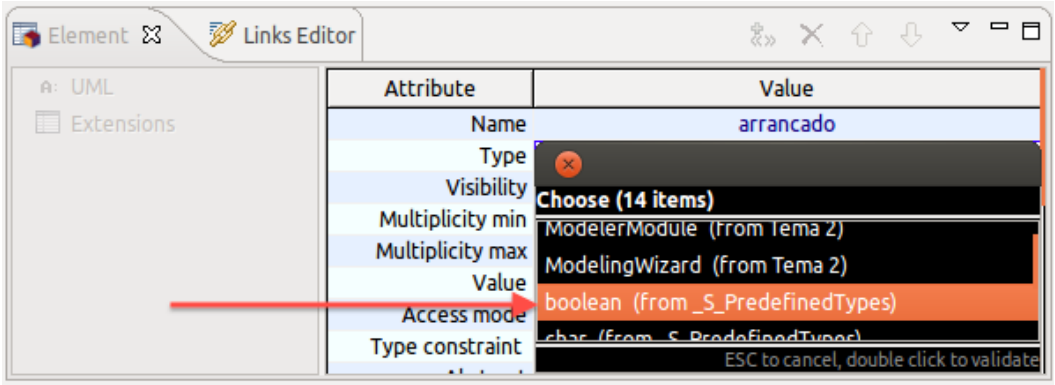
Lo renombramos (podemos seleccionar el atributo y darle a F2 o cambiarlo en la vista *Element*):



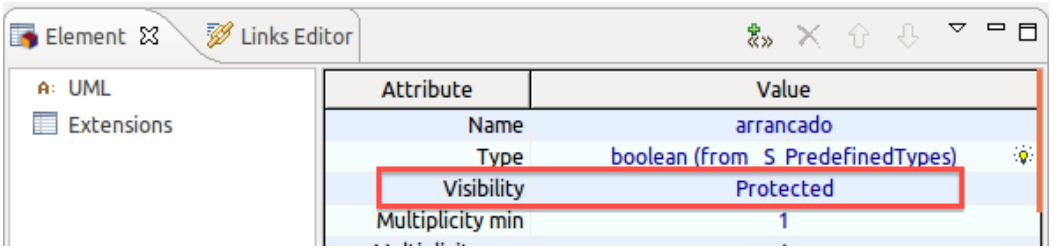
Por defecto los atributos son de tipo *string*. Lo borramos:



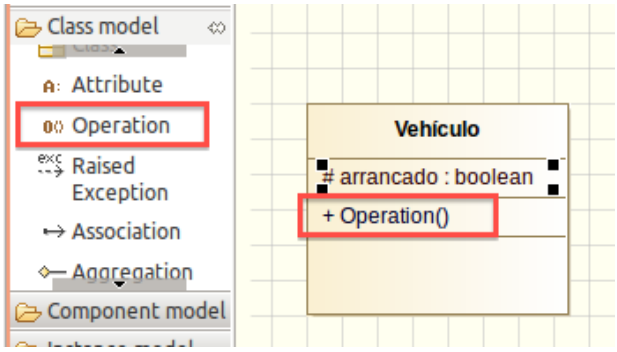
Y con CTRL + ESCAPE obtenemos la lista de tipos predefinidos. Seleccionamos *boolean*:



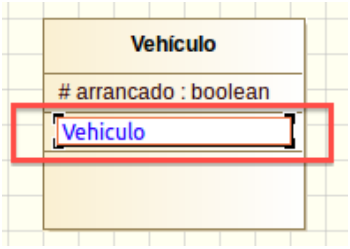
En la misma vista *Element* podemos cambiar la visibilidad del atributo por *Protected*:



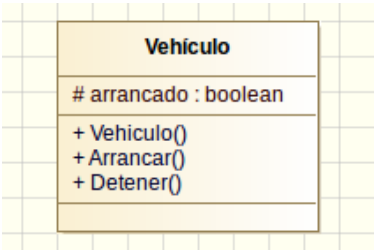
Añadimos una nueva operación:



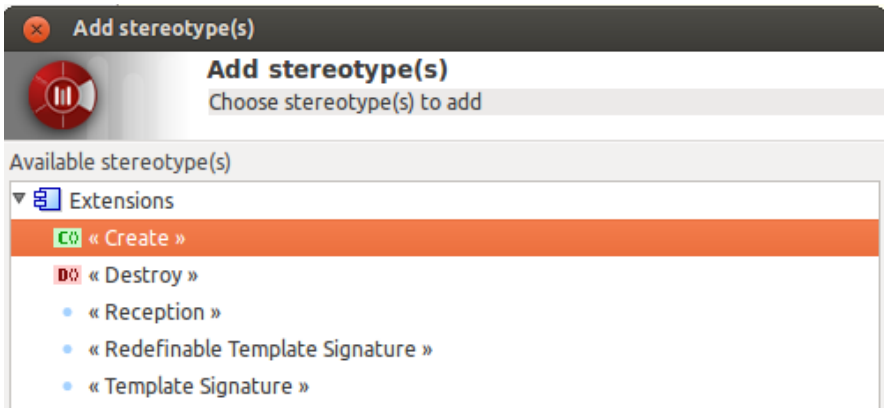
La renombramos a *Vehiculo*:



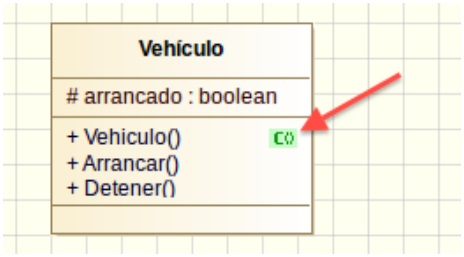
Añadimos dos operaciones más: *Arrancar* y *Detener*:



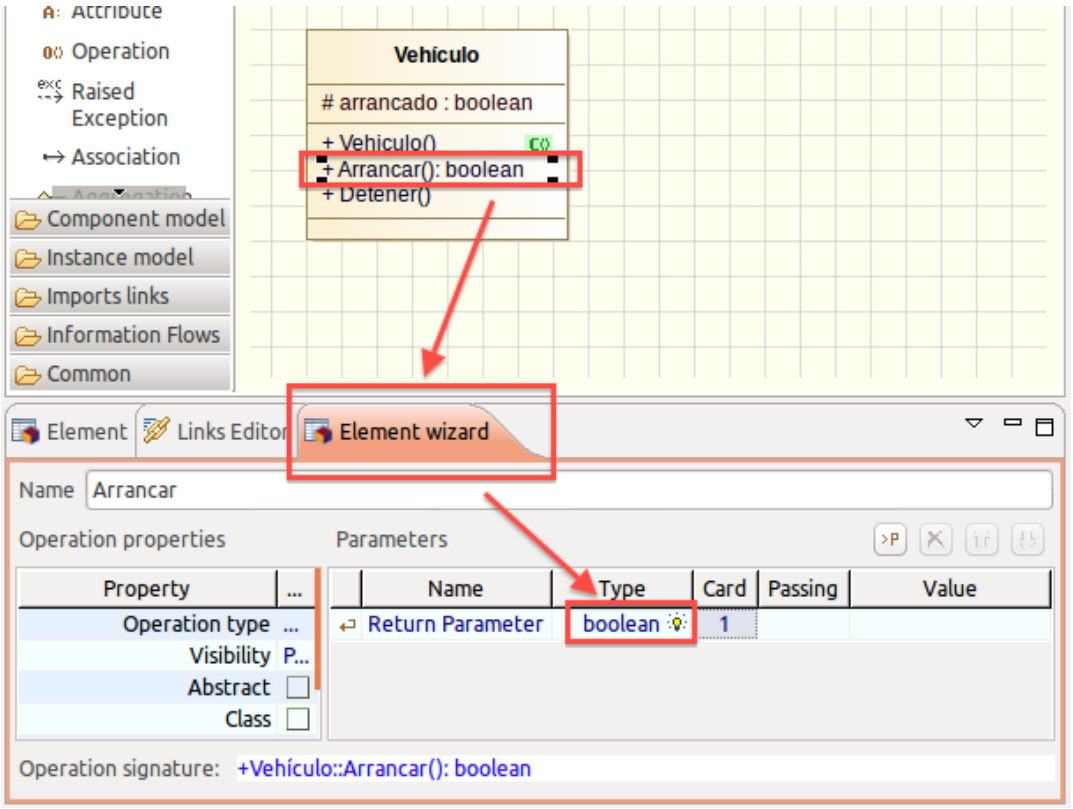
En Java, la operación *Vehiculo* sería el constructor de la clase, así que vamos a añadirle el estereotipo «Create» a dicha operación. Para ello la seleccionamos, hacemos clic derecho y después, *Add stereotype*:



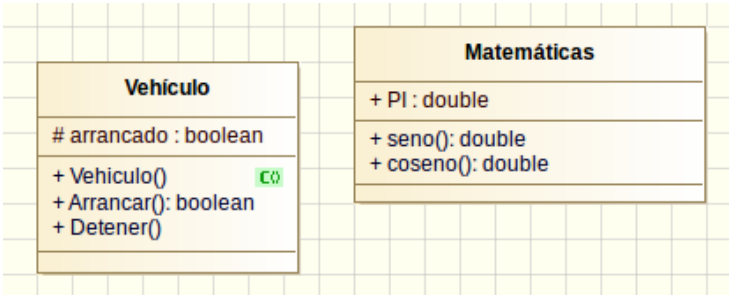
Por defecto, los estereotipos de las operaciones se muestran como un pequeño icono al lado de la operación:



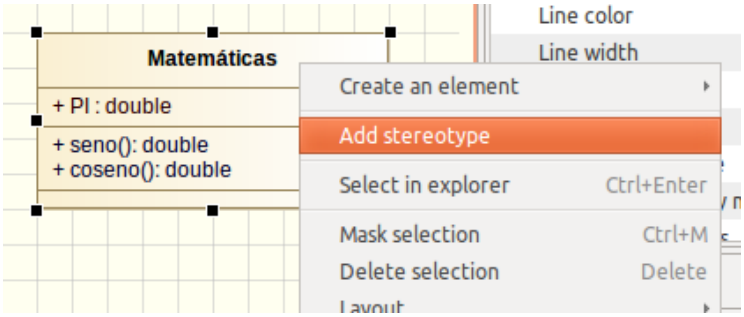
Vamos a cambiar el tipo de retorno de la operación *Arrancar*. Esto se hace desde la pestaña *Element Wizard*:



Creamos la clase *Matemáticas* con su atributo *PI* de tipo *double*, y las operaciones *seno* y *coseno*:

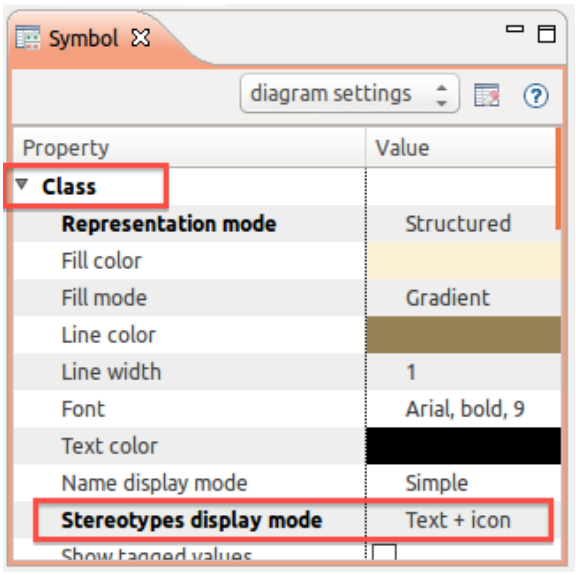


La clase *Matemáticas* sólo contiene atributos y operaciones estáticos, así que podemos añadirle el estereotipo «Utility» para recalcar que no es necesario instanciar esta clase para usarla. Para ello, clic derecho encima →*Add stereotype*:

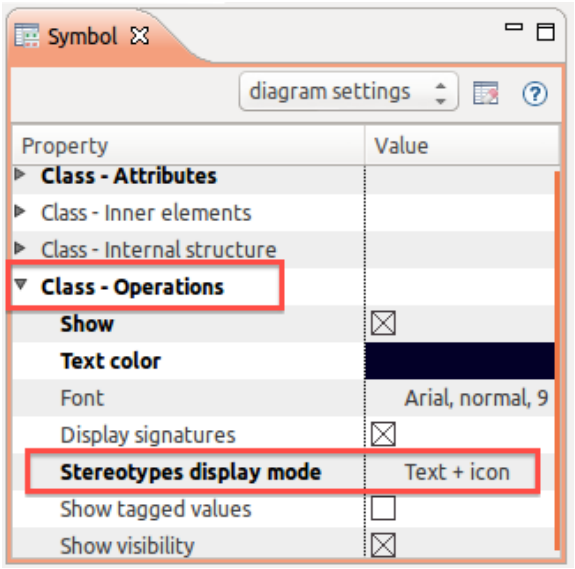


Y seleccionamos «Utility»:

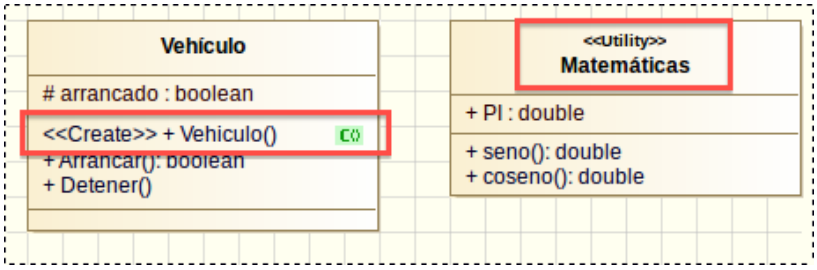
Por defecto los estereotipos de las clases no se muestran. Para que aparezcan, primero seleccionamos la clase *Matemáticas*, y luego en la vista *Symbol* debemos cambiar la propiedad *Class →Stereotypes display mode* a *Text + Icon*:



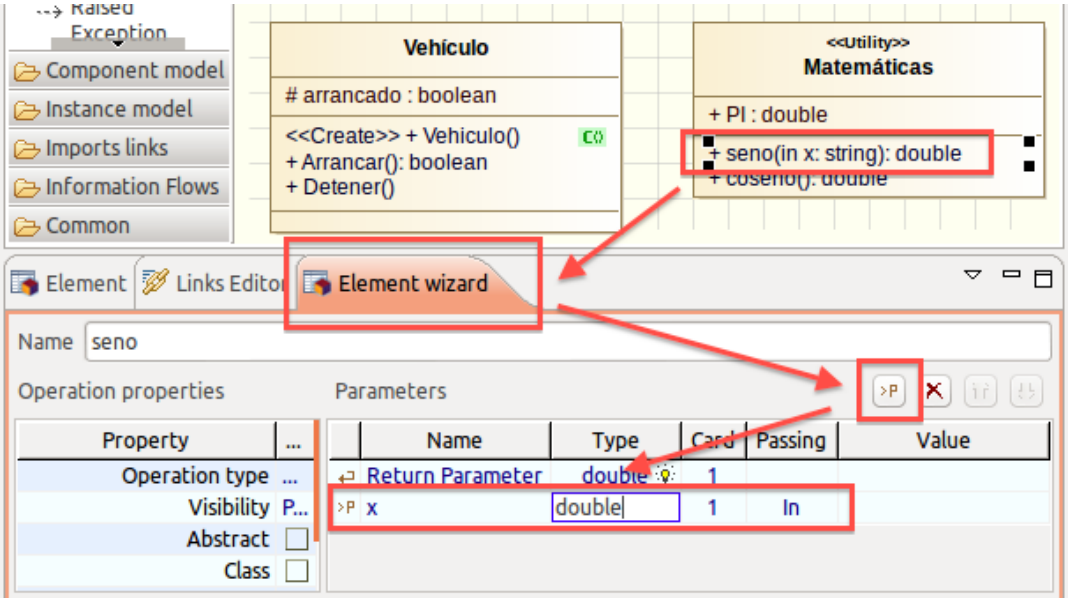
Si queremos que se muestre el estereotipo «Create» del constructor de *Vehículo* que hemos creado antes, podemos activarlo del mismo modo, seleccionando la clase *Vehículo*, pero cambiando esta vez la propiedad *Class - Operations →Stereotypes display mode*:



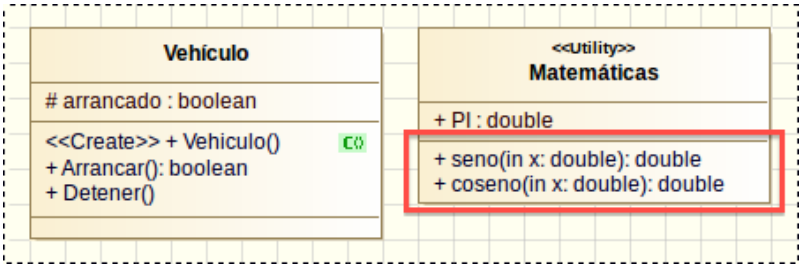
Ahora ya aparecen los estereotipos:



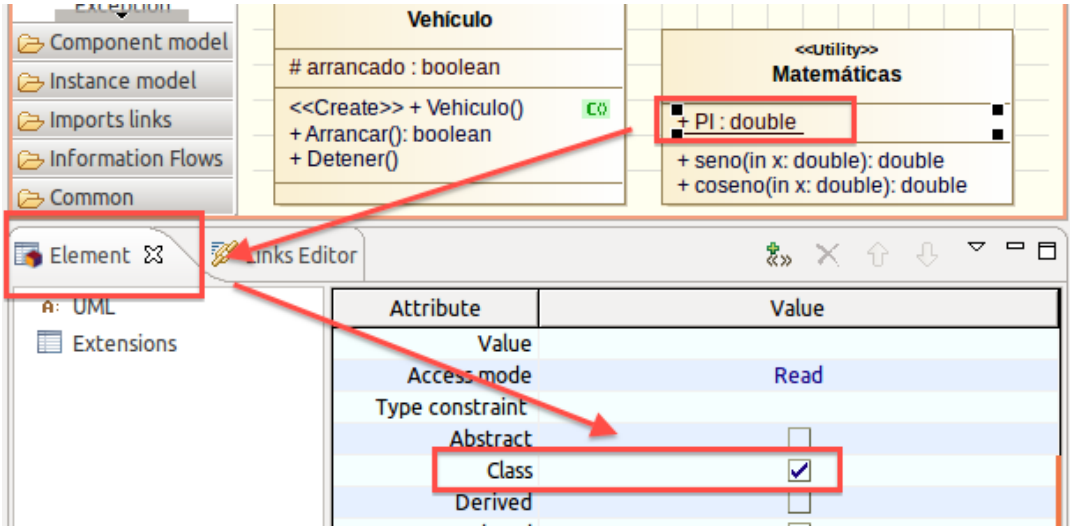
Para añadirle los parámetros a las operaciones, los seleccionamos y lo hacemos desde la vista *Element wizard*:



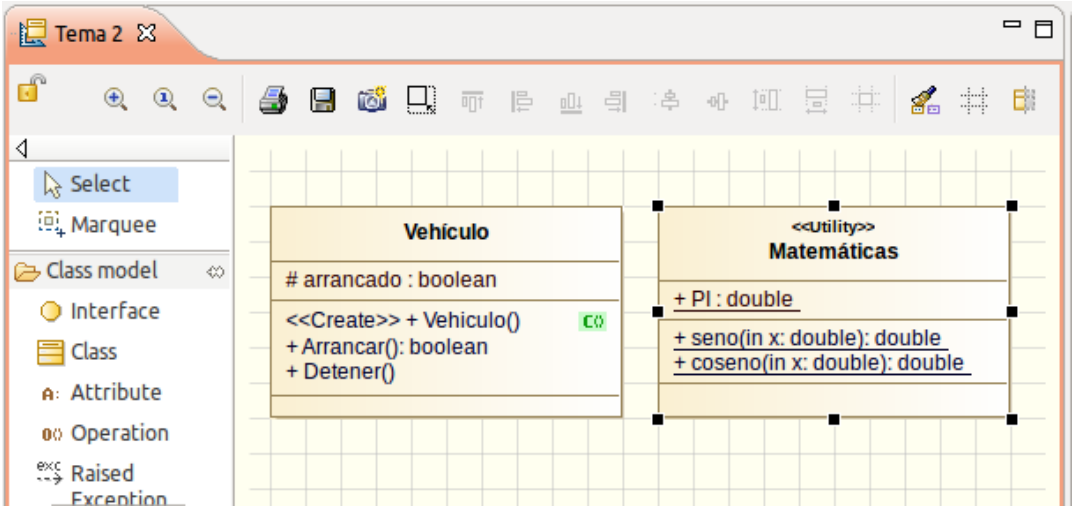
Los cambios reflejados en el diagrama:



Por último, todos los atributos y operaciones de *Matemáticas* son estáticos, lo que se indica marcando la casilla *Class* en la vista *Element* de cada elemento:



El diagrama completado:



3. Bibliografía y documentación adicional

- UML Reference Manual, 2nd Edition
  Booch, Jacobson, Rumbaugh, Ed. Addison-Wesley
- UML for Java Programmers
  Robert C. Martin, Ed. Prentice Hall
- Learning UML 2.0
  Miles, Hamilton, Ed. O'Reilly
- UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)
  Martin Fowler, Ed. Addison-Wesley
- MSDN (Microsoft): UML Use Case Diagrams: Guidelines
  http://msdn.microsoft.com/en-us/library/dd409432.aspx
- Aprendiendo UML en 24 horas
  Joseph Schmuller, Ed. Prentice Hall
- UML 2.0 - Pocket Reference
  Dan Pilone, Ed. O'Reilly
- Análisis y Diseño Estructurado y Orientado a Objetos de Sistemas Informáticos
  Amescua et al, Ed. McGraw-Hill
- Diseño Orientado a Objetos con UML
  Raúl Alarcón, Grupo Eidos