Conceptos esenciales del paradigma de la Orientación a Objetos

José Alejandro Gómez Castro

Manual de Programación Orientada a Objetos

Programacion Orientada a Objetos
Profesor: Antonio Luna
- 2016 -

Tabla de contenidos

Objetos y Clases……**3**
Ámbito interno y externo……**5**
Los pilares del modelo orientado a objetos……**6**
Relaciones entre Objetos……**7**
Fundamentos de UML……**8**
UML Objetos | Diagrama de clase y diagrama de secuencia……8
UML Objetos | Relaciones entre objetos……9
Conceptos esenciales del paradigma……**10**

Objetos y Clases

El modelo de orientación a objetos, nos permite convertir una clase en un nuevo tipo de dato (un objeto).
Los tipos de dato, sirven para diferenciar qué valores se pueden almacenar en las variables relacionadas y que operaciones que se pueden realizar sobre ellas.

Los tipos de dato, sirven para diferenciar qué valores se pueden almacenar en las variables y que operaciones que se pueden realizar sobre ellas (**servicios**).

Objeto es la estructura esencial para el modelado de orientación a objetos.
Objeto = Estado + Comportamiento + Identidad.

Estado
El estado de un objeto lo conforman los atributos que lo caracterizan. Puede ser modificado cuando el objeto interactúa, (por medio de mensajes) con otros objetos. El estado puede eventualmente determinar el comportamiento del objeto en cuestión.

Los valores de los atributos definen (implementan) el estado del objeto.

La mal asignación de valores y permisos podría generar un **estado inválido** al objeto, por eso es recomendable utilizar la delegación de responsabilidades adecuadamente para realizar validaciones de los valores que se quiere modificar.

Un ejemplo de estado inválido es: Un objeto Círculo con radio 0.

Comportamiento
Estado y comportamiento están relacionados.

El comportamiento agrupa las competencias de un objeto y describe las acciones y reacciones de ese objeto.

El estado puede ser modificado cuando el objeto interactúa (por medio de mensajes) con otros objetos.

El estado del objeto puede eventualmente determinar el comportamiento del objeto en cuestión.

Identidad
Es una propiedad del objeto. La existencia de un objeto en el espacio en que se encuentra, permite diferenciar aquellos elementos particulares que me permite distinguirlos.

Enlace
Es fundamental para poder establecer el mecanismo de comunicación, permitiendo al cliente comunicarse con el servidor. Además, denota la asociación específica por la cual un objeto (cliente) utiliza los servicios de otro objeto. El paso de mensajes entre dos objetos es típicamente unidireccional, aunque puede ser bidireccional.

Mensajes
Va sobre el enlace, al ser enviado ejecuta la orden que se requiere, el mensaje se interpreta de acuerdo a la especificación de acciones que tiene el servidor.

Los métodos se ejecutan como resultado de los mensajes que envía el cliente.

Los métodos especifican los mensajes que puede recibir un objeto y los cambios que producen en el objeto en cuestión, es decir, su comportamiento.

Interfaz
Es donde el servidor expone sus servicios, la lista de métodos o acciones que puede realizar un objeto conforma su interfaz. Las métodos se especifican en la clase del objeto.

Sobrecarga
Son métodos que tienen el mismo nombre, pero se diferencian por sus parámetros, en función de sus tipos. Permite implementar de diferentes maneras una misma funcionalidad.

Constructores
Son un **método** que, **(en el caso del lenguaje Java)**, **tiene el mismo nombre de la clase**, **su objetivo es inicializar en estados válidos los atributos del objeto**.

Para llamar a un método del mismo nombre de la clase, es decir, a un constructor de la misma clase, se utiliza **this().**

NOTA:

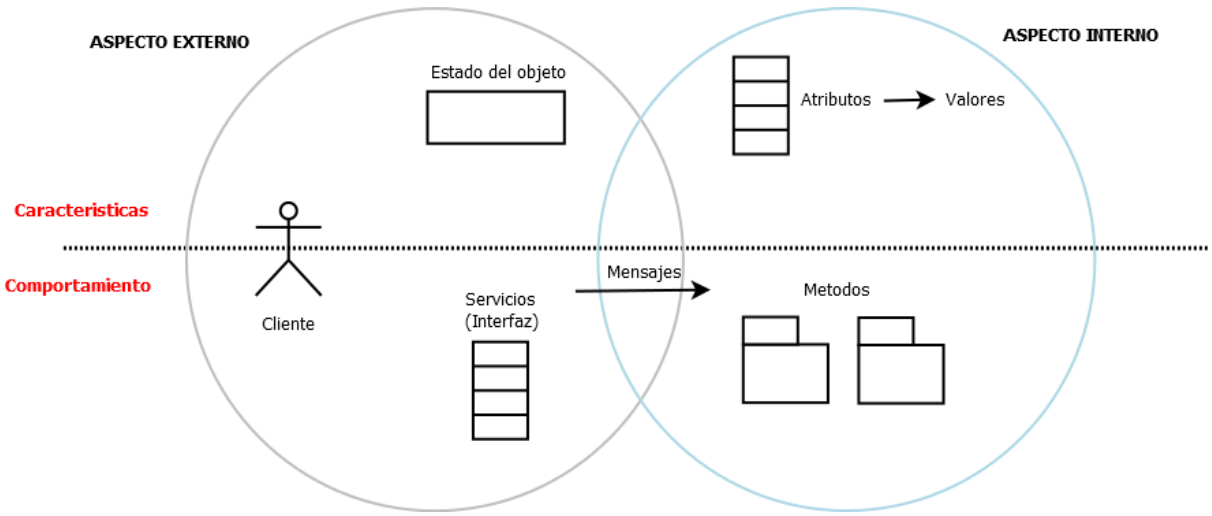
> Los tipo de dato **String,** tienen la particularidad dentro de su especificación, de ser **inmutables**, es decir, no se puede cambiar su contenido, cuando se le da un nuevo valor, automáticamente apunta a otra dirección y pierde su valor anterior, **(Recolector de Java)**.
>
> Para identificar si dos direcciones son iguales se utiliza **(x == y)**.
>
> Para identificar si dos contenidos son iguales se utiliza **(x.equals(y))**.

Ámbito interno y externo

Cuando se **declara** un objeto, **se abre un espacio en memoria**, pero almacena el valor null, debido a que no se ha creado ningún objeto. Al momento de **crear** el objeto, **el espacio en memoria apunta a una dirección válida**, donde guarda sus **características**, es decir, el **estado** del objeto viéndolo desde una perspectiva en el **ámbito externo,** o **atributos** desde el **ámbito interno**.

El **ámbito interno** se ve como un objeto software compuesto de un **conjunto de atributos y métodos relacionados**.

Cada objeto tiene **su propio espacio**.



Interactuamos con el objeto por medio de una **variable de referencia**, que funciona como **intercomunicador** entre el **cliente** y el **servidor**.

Los **atributos** de un objeto deben ser **privados**, para **evitar que el cliente pueda modificar los valores,** o bien protegerlos de un **estado inválido**.

Objeto cliente: requiere algo del servidor.
Objeto servidor: da resultado a la necesidad del cliente.

Los pilares del modelo orientado a objetos.

Abstracción

Consiste en describir o especificar los aspectos esenciales de una idea referente a un concepto, obviando sus características y comportamiento accidentales.

Sirve para separar el comportamiento esencial de un objeto de su implementación.

Encapsulamiento

Se denomina encapsulamiento al ocultamiento del estado.
Permite simplificar al dar a conocer al cliente, lo menos del servidor y viceversa.

La abstracción se centra en el comportamiento observable de un objeto, mientras que el encapsulamiento se centra en la implementación que da lugar a ese comportamiento.

Modularidad

El concepto de modularidad ofrece mecanismos para agrupar abstracciones relacionadas lógicamente.

Jerarquía

Consiste en una clasificación u organización de las abstracciones, frecuentemente un conjunto de abstracciones forman una jerarquía.

Relaciones entre objetos

Para que una relación de objeto de manifieste debe existir un cliente y un servidor.

Asociativas.
Los enlaces existen más allá de la interacción del objeto.

De uso:
Los enlaces se pierden después de la interacción.
Cuando existe una dependencia.

Relación Jerárquica

Es una relación todo parte.
Se divide en relación jerárquica de agregación y composición.
La diferencia es el nivel de vínculo.

Agregación – no vincular.

Los clientes del todo podrían eventualmente tener acceso a las partes del todo.
Es una relación jerárquica que es más liviana que la composición.
Es posible que al eliminar al todo queden las partes.

- Vehículo podría cambiar el motor que usa.

Composición – vincular.

Los clientes del todo no pueden tener acceso directo a las partes del todo.
Es una relación jerárquica muy fuerte.
Cuando los objetos están estrictamente vinculados y no hay forma de llegar a la parte sin pasar por el todo.
Al eliminar el todo, por fuerza se eliminan las partes.

- Si el vehículo se elimina automáticamente se elimina el motor.

Relación No Jerárquica

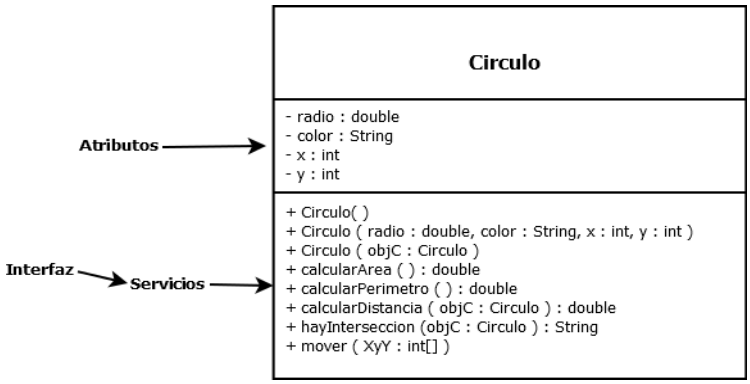
Cuando hay diferencia de importancia.
Cuando cada uno puede existir sin dependencia del otro.

- Registro de empleados
- Registro de computadoras

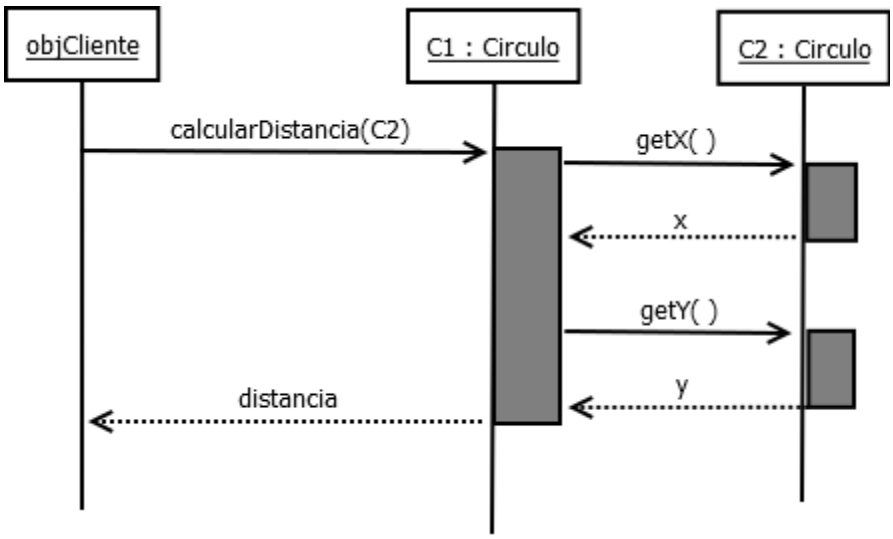
Cuando un empleado se va de la empresa, el equipo asignado al mismo, sigue existiendo. Porque es parte del equipo de la empresa y no depende de un operador para existir.

UML Objetos | Diagrama de clase y diagrama de secuencia.

El diagrama de clase permite ver la especificación del objeto o su ámbito.



El diagrama de secuencia permite ver la interacción entre objetos



UML Objetos | Relaciones entre objetos.

La flechita

En qué dirección van los mensajes.

Multiplicidad

Una instancia de un tipo, con cuantas instancias de otro tipo se relaciona.

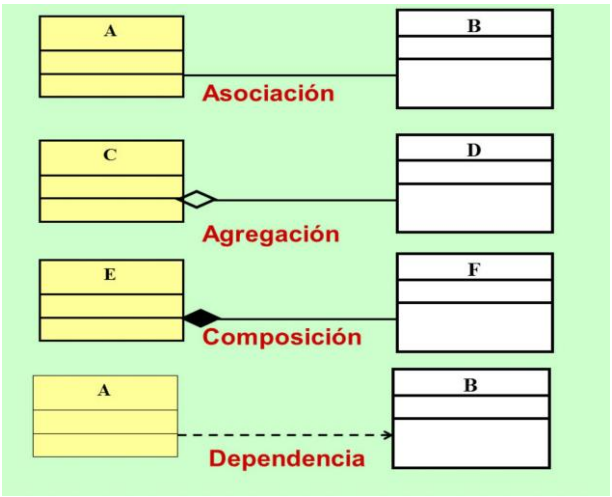
Navegabilidad

Si nos interesa restringir quien es cliente y quien servidor**.**

Rol

Nombrecito que aparece al lado de la instancia,
Una instancia de computadora se asigna a una instancia de empleado que juega el rol de responsable.
El nombre del rol corresponde al nombre de la variable que debe tener el servidor en el cliente.

El rombo siempre debe ir en el lado del todo y no de la parte.
Dependencia es una línea discontinua entre el cliente y el servidor.



Las relaciones son bidireccionales, a excepción de las relaciones de dependencia.

NOTA:
Cuando el nombre de la relación entre objetos se presenta ambiguo, es conveniente utilizar una flecha para especificar hacia donde se lee el enlace. Siempre que el dominio del problema permita ser más específicos a la hora de describir la integración entre objetos

Conceptos esenciales del paradigma de orientación a objetos.

1. Se recomienda declarar privados los atributos de un objeto, para preservar el encapsulamiento y evitar un estado inválido. Los clientes no deben tener acceso a los atributos del objeto.

2. Los métodos de un objeto necesitan de la instancia para poder existir.

3. Cada objeto tiene una única identidad.

4. Cuando se imprime directamente un objeto, Java por default llama al método **toString()** de la clase **Object**. Automáticamente el framework hace que los objetos hereden a su vez de **Object**.

```
System.out.println(c1);
```

Nota:

Si redefinimos los métodos dentro de la clase Circulo, Java llama al método redefinido dentro del objeto en cuestión.

5. En la asignación de responsabilidades, debe distribuirse, de tal forma que, el cliente conozca lo menos posible del servidor. Y que el servidor solo conozca lo esencial del cliente para llevar a cabo la tarea.

6. Desarrollar aplicaciones de manera precisa según las abstracciones respecto de los conceptos que representan.

7. Hablar de abstracción, es hablar de niveles de detalles.

8. Un Objeto tiene un estado, exhibe un comportamiento bien definido y posee una identidad

9. Siempre que se quieran ejecutar los métodos de clase, deben hacerse utilizando el nombre de la clase. Y no el nombre de instancias. El compilador no da error pero ocasionaría un error conceptual.

10. Desarrollar aplicaciones fuertemente cohesivas y débilmente acopladas.

11. Asociación, agregación y composición.

12. Cuando se representan como una relación en código, se crea un objeto del tipo del servidor en el cliente, y si la multiplicidad es muchos, se crea una colección de objetos.

13. Asociación, es común que reciba por parámetro el cliente al servidor.

14. El compilador busca el método de la clase que está declarada la variable. El runtime (en ejecución), busca el método en la subclase que fue instanciado.

15. El uso del polimorfismo se expresa conceptualmente con **is-a.**

16. Se puede ver al establecer una interface, como un tipo de comportamiento dentro del entorno del sistema.

Faltan temas:

Clase Abstracta
Conversion implícita y explita (CAST)
Is-a & Has-a