INTRODUCCIÓN AL BLOQUE DE ORIENTACIÓN A OBJETOS

ORIENTACIÓN A OBJETOSY UML

- La metodología O.O es una forma de ver los sistemas de información más “natural”.
- UML (Lenguaje de Modelado Unificado) permite, a través de la representación gráfica, mostrar el sistema de información desde distintos puntos de vista, ajustándose en cierta forma a la metodología O.O para presentar un sistema SÓLIDO.
- Los creadores de UML son los llamados “tres amigos”: Booch, Rumbaugh y Jacobson; y comenzaron a compartir sus ideas sobre la metodología O.O a mediados de los 90.

OBJETIVOS

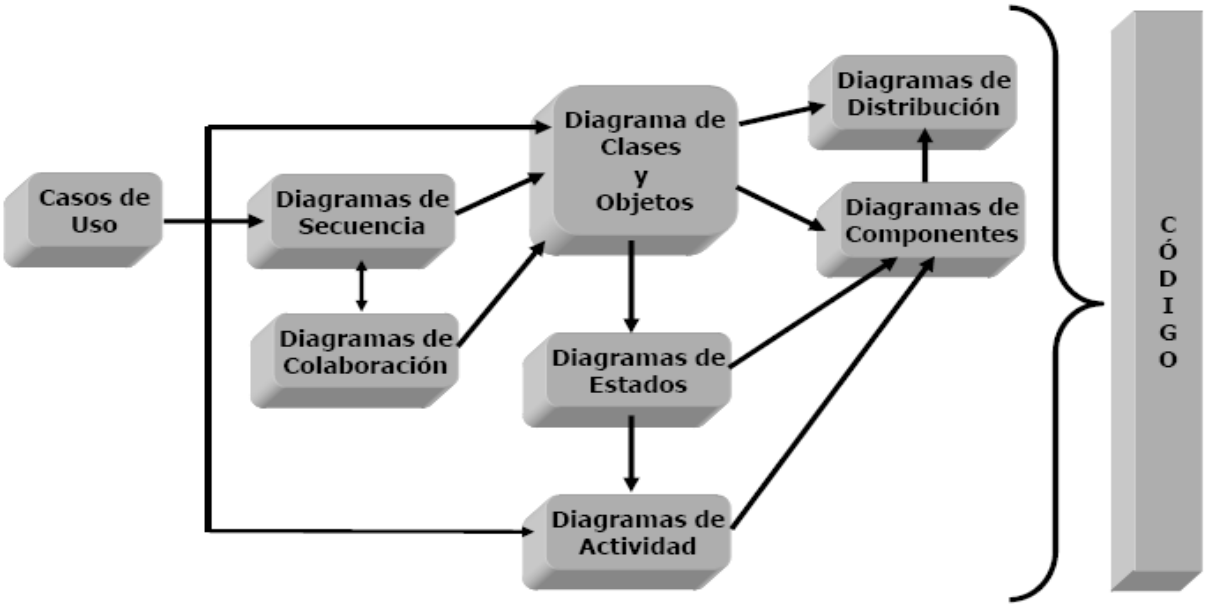
- Mejorar la ORGANIZACIÓN del trabajo
- Mejorar la COMUNICACIÓN
- Facilitar el MANTENIMIENTO
- Mejorar el TIEMPO DE ENTREGA

Para ello se crean diferentes modelos o diagramas que representan el sistema en su totalidad y desde diferentes puntos de vista. No es necesario crear diagramas de todos los tipos para representar el sistema.

DIAGRAMAS

- DIAGRAMAS DE CASOS DE USO (Análisis de requisitos)
- DIAGRAMAS ESTRUCTURALES
    - DIAGRAMAS DE CLASE
    - DIAGRAMAS DE OBJETOS
- DIAGRAMAS DE COMPORTAMIENTO
    - DIAGRAMAS DE INTERACCION
        - DIAGRAMAS DE SECUENCIA
        - DIAGRAMAS DE COLABORACION
    - DIAGRAMAS DE ESTADOS
    - DIAGRAMAS DE ACTIVIDAD
- DIAGRAMAS DE IMPLEMENTACION
    - DIAGRAMAS DE COMPONENTES
    - DIAGRAMAS DE DISTRIBUCIÓN

Relación entre Diagramas



OTROS COMPONENTES GRÁFICOS

PAQUETES: Siguiendo el objetivo de organizar el sistema gráfico, es posible unir distintos diagramas que tienen una finalidad común en un único paquete. En forma de carpeta
NOTA: Componente para agregar un comentario en un diagrama. En forma de nota
RESTRICCIÓN: Es una condición que se debe dar en la implementación de la clase. Se representa entre llaves y existe un lenguaje (OCL) concreto para especificarlas.
ESTEREOTIPOS: Si una clase, atributo, conjunto de atributos... es reutilizable, se le puede poner un nombre y especificarlo con <<nombre estereotipo>>. Por una ficha básica de datos. Un interface es un estereotipo y tambien se puede representar con un círculo en blanco.

CICLO DE VIDA DE DESARROLLO DE SOFTWARE ORIENTADO A OBJETOS

Se trata de un proceso **iterativo** e **incremental:**

El ciclo de vida iterativo se basa en cuatro fases en cascada:

- INICIO: Análisis de requisitos del sistema. (ERS).
- ELABORACIÓN: Análisis y Modelado
- CONSTRUCCIÓN: Codificación
- TRANSICIÓN: Entrega del producto

Cada fase consta de un ciclo de vida en espiral propio con los siguientes fases:

- Requisitos para esta fase
- Análisis Conceptual de lo que hay que hacer en esta fase
- Diseño de los modelos a seguir
- Implementación o realización de los diseños de forma concreta
- Pruebas: Verificar que se cumplen los requisitos de forma correcta

ANÁLISIS ORIENTADO A OBJETOS

INTRODUCCIÓN

Vivimos en un mundo de objetos. Estos objetos existen en la naturaleza, en entidades hechas por el hombre, en los negocios y en los productos que usamos. Pueden ser clasificados, descritos, organizados, combinados, manipulados y creados. Por esto no es sorprendente que se proponga una visión **orientada a objetos** para la creación de software, una abstracción que modela el mundo de forma que nos ayuda a entenderlo y gobernarlo mejor.

La primera vez que se propuso un enfoque orientado a objetos para el desarrollo de software fue a finales de los 60. Aunque no fue hasta los 90 cuando la ingeniería del software orientada a objetos se convirtió en el paradigma de elección para muchos productores de software.

Un **objeto** encapsula tanto datos como los procesos que se aplican a esos datos. Esta importante característica permite construir clases de objetos y bibliotecas de objetos y clases reutilizables. La reutilización de componentes software lleva a un desarrollo de software más rápido y a programas de mejor calidad.

Para representar los componentes de un sistema modelado según una perspectiva orientada a objetos utilizaremos **UML**, que es un lenguaje estándar para el modelado orientado a objetos.

El **Análisis Orientado a Objetos(AOO)** es “un método de análisis que examina los requisitos de un sistema desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema” [Booch 94] .

El propósito del AOO es definir todas las clases que son relevantes al problema que se va a resolver, las operaciones y atributos asociados, las relaciones y comportamientos asociados con ellos. Estas clases se encontrarán en un conjunto de modelos que describirán el software que satisfará el conjunto de requisitos definidos por el cliente.

CONCEPTOS BÁSICOS DEL ANALISIS ORIENTADO A OBJETOS (OO)

- Una clase es un tipo global de objeto. Por ejemplo la clase cliente. Un cliente concreto será un objeto o instancia de la clase cliente. De cada clase podemos crear todos los objetos diferentes que queramos.
- Un atributo es una característica de una clase y un método de una operación o función asociada a la clase
- De cada cliente tenemos datos /atributos: (nombre, teléfono, clave, NIF.).
- Con cada clase podemos utilizar métodos concretos. Con un cliente por ejemplo podemos consultar_datos(NIF), modificar_datos(NIF), comprobar_clave(NIF,clave)

Clase:CLIENTENombre
Telefono
Clave
NIFconsultar_NIF()
modificar_datos()

- Un objeto es una instancia de una clase. Al crear un nuevo objeto cliente cliente_nuevo le damos valor a los atributos: (Marcial Ruiz Escribano, 93-88721, marcialin, 766362-Z). Y tambien podemos usar los métodos de un objeto (En Java siempre hay que utilizar un punto entre el nombre del objeto y el método/atributo)
- Entre objetos además podemos utilizar "mensajes" que son funciones que implican a más de un objeto. Por ejemplo puede existir la función "cuenta.asociar_titular_cuenta" que afecte tanto a un objeto cliente como a un objeto cuenta corriente.

ABSTRACCION

- Abstracción es la capacidad de separar lo fundamental de los superficial en un clase. Eso dependerá de los objetivos y requisitos de la aplicación. De un mismo tipo de clase Coche podremos recoger distinta

información si es para una aplicación de un taller, que si es para un concesionario o para una aplicación de pago del impuesto de vehículos de tracción mecánica

ENCAPSULAMIENTO

- El encapsulamiento es la propiedad que tienen los objetos de constituir una entidad completa, que al usuario de la aplicación no le va a mostrar por separado sus atributos y métodos, si no una unidad. Por ejemplo, un cliente en una aplicación de un banco no es un conjunto de datos separados y de operaciones, si no que es un cliente sin más, con todo lo que eso conlleva a través de la aplicación, de lo que la aplicación me deja hacer con un cliente, pero su composición está oculta.

Objeto = identidad + estado(atributos) + comportamiento(métodos)

Identidad

Cada objeto posee un **oid** (object identifier) que establece la identidad del objeto y tiene las siguientes características:

- Constituye un identificador único y global para cada objeto dentro del sistema.
- Es determinado en el momento de creación del objeto.
- Es independiente del valor de los atributos del objeto.
- No cambia durante la vida del objeto.

Estado

- Son los valores de los atributos. Un atributo toma sus valores en un dominio concreto. **Por ejemplo,** color blanco, negro, plata, gris, azul, rojo, amarillo y verde.

- El estado evoluciona con el tiempo, aunque algunos atributos pueden ser constantes.

Comportamiento

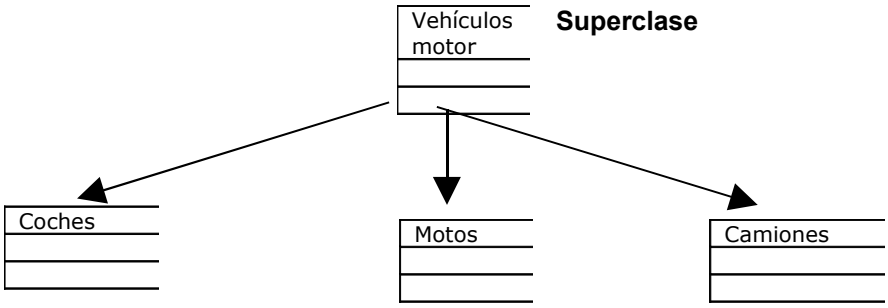
- Es la forma en que opera ese objeto. ¿qué hace y cómo lo hace?

- Un mensaje estimula cierto comportamiento en el objeto receptor del mismo (los mensajes son el medio a través del cual interactúan los objetos).

HERENCIA

- Es un mecanismo mediante el cual se puede crear una nueva clase partiendo de una existente, se dice entonces que la nueva clase hereda las características de la case existentes aunque se le puede añadir más capacidades (añadiendo datos o capacidades) o modificar las que tiene.
- Por ejemplo supongamos que tenemos la *VehiculosDeMotor.* En esta clase tenemos los siguientes atributos: *Cilindrada y Numero de Ruedas*, y el método *acelerar()*.
- Mediante el mecanismo de herencia podemos definir la clase *Coches* y la clase *Motos.*
- Estas dos clases heredan los atributos *Cilindrada* y *Numero de Ruedas* de la clase *VehículosDeMotor* pero a su vez tendrán atributos propios (como hemos dicho antes el *Numero de Puertas* es un atributo propio de la clase *Coches* que no tienen sentido en la clase *Motos*).
- Se puede decir que *Coches* extiende la clase *VehículosDeMotor,* o que *VehículosDeMotor* es una ***generalización*** de las clases *Coches* y *Motos.*

Una **superclase** es de la que se hereda y una **subclase** es una heredera de una clase.



ASOCIACIÓN ENTRE CLASES

- Un asociación es una relación entre dos clases que viene dada por los requisitos del sistema. En esta asociación, que se identifica por una flecha que las une, puede especificarse una multiplicidad o cardinalidad, es decir, cuántos objetos de esa clase intervienen en la asociación.

- Por ejemplo, en un sistema de información de una empresa de alquiler de coches; una clase coche se relacionaría con una clase persona a través de una asociación: “es conducido” con cardinalidad en la que interviene el coche concreto y una sola persona o varias dependiendo de la fecha de alquiler.

- En el próximo tema, lo veremos con más detenimiento.

AGREGACIÓN/COMPOSICIÓN

- Una clase puede estar compuesta por varias clases, no es herencia. Por ejemplo la clase Coche puede estar compuesta de la clase motor, la clase rueda... El nivel de detalle dependerá de los requisitos.

POLIMORFISMO

- **Hace referencia a la posibilidad de que dos métodos implementen distintas** acciones, aun teniendo el mismo nombre, dependiendo del objeto que lo ejecuta o de los parámetros que recibe. En el ejemplo anterior teníamos dos objetos que heredaban el método *acelerar()* de la clase *VehiculosDeMotor*.
- De hecho en clase *VehiculosDeMotor* al ser general no tiene sentido que tenga una implementación concreta de este método.
- **Sin embargo, en las clases Coches y Motos si que hay una implementación clara y distinta del método *acelerar().***

REPRESENTACIÓN DE UNA CLASE Y DE UN OBJETO

```
Alumno
Nombre: string
apellidos: string
email: string
nota: real =0
...

cambiarNota(real)
consultarNota(): real
esAprobado(): bool
...
```

Restricciones: {nota<=10 && nota >0}

```
unAlumno:Alumno
Nombre=”Pedro”
apellidos=”Rodriguez”
email=”pr@email.es”
nota=”7”

cambiarNota(real)
consultarNota(): real
esAprobado(): bool
```

EJERCICIO INICIAL

Respecto a una aplicación para controlar el sistema de una radio, conteste a las siguientes preguntas:

1. Existe una clase empleadoRadio, otra que es Locutor y otra responsableSonido. ¿qué relación crees que mantienen?
2. Cuando un responsable de sonido se registra en el sistema y aparecen los mandos del volumen, corte de canciones...., en cambio cuando el Locutor se registra sólo le aparece la lista de pistas y su duración. ¿Cómo se llama a este doble comportamiento?
3. Cuando un oyente escucha en la radio una pista, es totalmente ajeno a la forma en que se ha realizado la aplicación. ¿cómo se le llama a esta característica?
4. Especifique y represente de forma básica la clase Disco
5. Especifique y represente de forma básica la clase Pista
6. ¿Qué relación tienen las dos clases anteriores?
7. Especifique una restricción para la clase disco
8. Cree un objeto disco y representelo
9. ¿Qué tipo de relación hay entre responsableSonido y Pista?
10. Las cuñas de publicidad durante el análisis ¿cree que podrían dar lugar a varias clases?, ¿de qué depende?.

RESPUESTAS