Metodologías en la Ingeniería del Software
Métodos Orientados a Objetos

Daniel Rodríguez García
Departamento de Ciencias de la Computación
Universidad de Alcalá

Contenidos

- Historia Orientación a Objetos (OO)
- Problemas métodos estructurados
- De lenguajes estructurados a la OO
- Propiedades de la OO
- Bibliografía

Historia de la Orientación a Objetos

- Partiendo de los problemas citados, el paradigma de la orientación a objetos se remonta a 1967, donde los noruegos **Ole-Johan Dahl** (1931-2002) y **Kristen Nygaard** (1926-2002) desarrollaron los conceptos básicos de la programación orientada a objetos, en un lenguaje llamado SIMULA 67.
  - Ambos recibieron conjuntamente el premio ACM Turing y la medalla IEEE John von Neumann por sus ideas para que pudiese emerger la orientación a objetos.
  - En la actualidad existe el premio Dahl-Nygaard de la asociación AITO (Association Internationale pour les Technologies Objets)
- Otros lenguajes influenciados por SIMULA 67:
  - Smalltalk, Objective C, C++, Eiffel, CLOS


Ole-Johan Dahl (1978)


Kristen Nygaard (1978)

En relación a la programación…

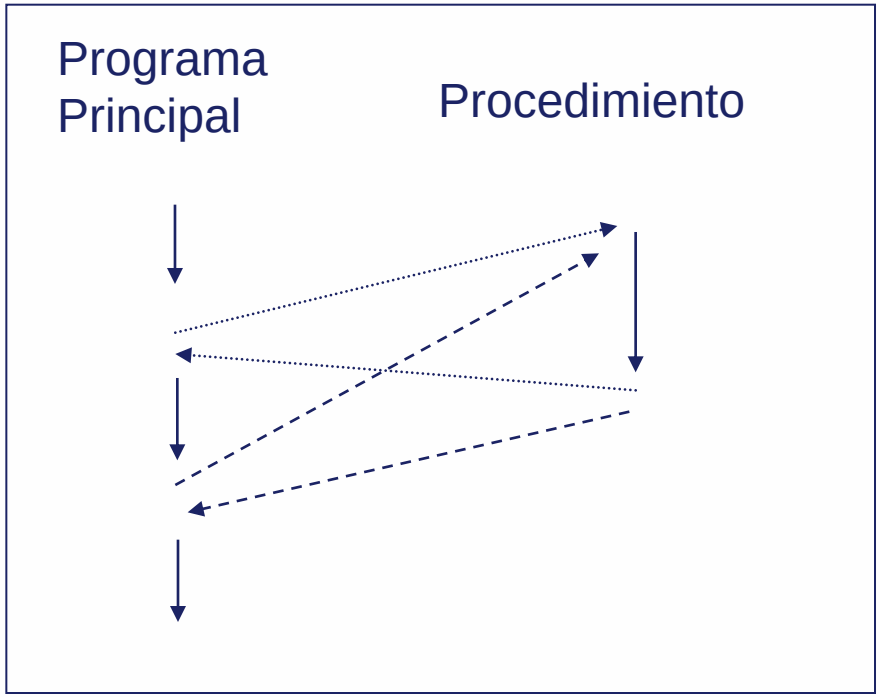
- La evolución de los lenguajes ha sido…
  - Programación no estructurada
  - Programación procedural
  - Programación modular
  - Programación orientada a objetos

Programación no estructurada

- Datos son globales y pueden ser accedidos y modificados en cualquier punto de la ejecución de un programa

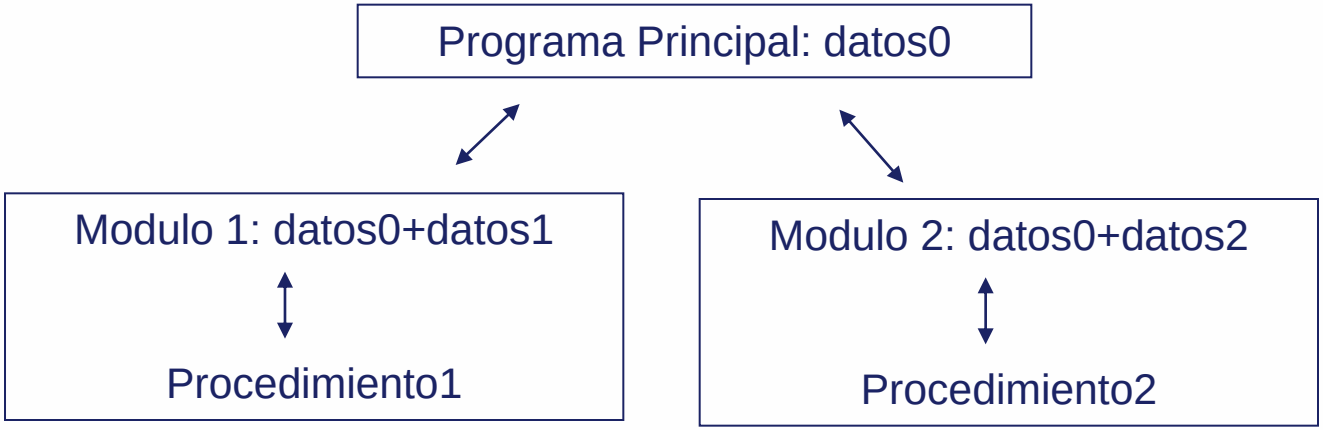
Programación procedural

- Programa estructurado con subrutinas, y las subrutinas pueden se llamadas desde cualquier punto del programa



Programación modular

- Procedimientos con funcionalidad común están agrupados
- Cada módulo tiene su propio estado interno donde la interacción ocurre entre el programa principal y módulos a través de las llamadas a procedimientos

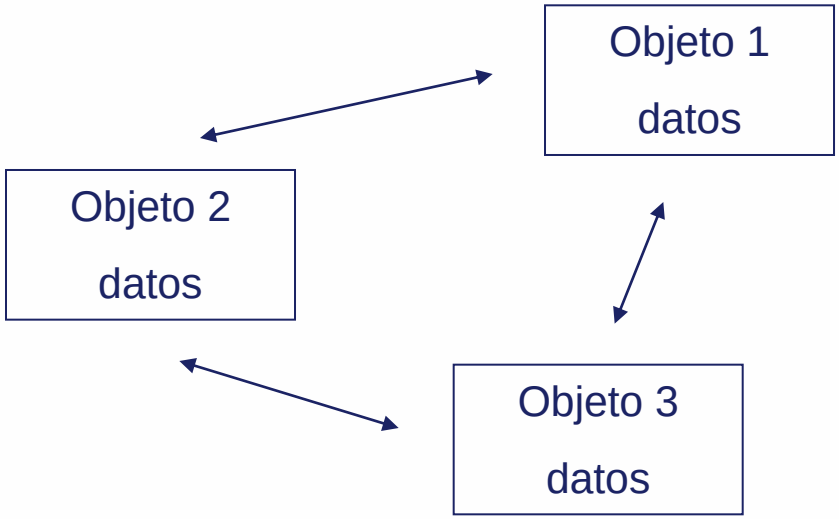


Problemas con la programación modular

- Algunos de los problemas de la programación son:
  - Problemas al modelar aplicaciones con datos no estructurados, ya que presuponen la modificación de entradas bien definidas en salidas igualmente estructuradas.
  - Difícil reutilización de código, al manipular datos muy dependientes del dominio
  - Difícil mantenimiento, por el mismo motivo que el anterior
  - Reducción de criterios como eficiencia, flexibilidad robustez, etc. con el mantenimiento

Programación orientada a objetos

- Grafo de objetos cada uno con su propio estado
  - Un objeto es cualquier entidad que pueda ser modelada.
    - Se definirá en detalle más adelante.
- Los objetos interactúan entre ellos mediante envío de mensajes



¿Por qué la orientación a objetos?

- Los conceptos de objeto son básicos para la mayoría de productos de software actuales
  - Entornos gráficos (GUI) y de interacción con el ordenador, componentes, frameworks, etc
- Las metodologías OO utilizan técnicas tradicionales de análisis del software pero:
  - La OO integra mucho mejor el análisis y el diseño. Proximidad de los conceptos de modelación respecto de las entidades.
  - Herramientas de generación de código desde entornos visuales
    - Herramientas CASE (Computer Aided Software Engineering)
    - MDA Model Driven Architecuture

Fundamentos de la OO

- Los principios sobre los que se fundamenta la O.O. los podemos resumir en:
  - Se utilizan modelos basados en conceptos del mundo real para el desarrollo de aplicaciones.
  - El software se organiza como una colección discreta de objetos a los que se asocian datos (atributos) y comportamiento (operaciones, procedimientos ó métodos)
  - La comunicación de los objetos se realiza a través del paso de mensajes. Cuando un objeto es receptor del mensaje ejecuta una acción (operación, procedimiento ó método)

Objetos

- Un objeto es “una cosa que tiene comportamiento, estado e identidad”.
  - **Comportamiento**: Cómo actúa un objeto al recibir un mensaje o estímulo externo. Existen cuatro tipos de comportamiento, también llamados operación:
    - Modificación: alteración del estado del objeto.
    - Selección: acceso al estado del objeto.
    - Construcción: creación de un objeto e iniciación de su estado.
    - Destrucción: liberación del estado de un objeto.
  - **Estado**: La suma de las propiedades o atributos que tiene el objeto, y que forman su estructura estática, junto con los posibles valores de cada uno de esos atributos que forman su estructura dinámica.
  - **Identidad**: Un atributo o propiedad característica del objeto que le distingue de todos los demás.

Objetos

- Características de los Objetos:
  - **Encapsulación**: Característica por la cual los atributos y operaciones se reúnen en una entidad única que es el objeto.
  - **Persistencia de objetos**: Capacidad de un objeto de permanecer en el tiempo o en el espacio. Sobre él se pueden realizar las operaciones:
    - Pasivación del objeto.
    - Activación del objeto.

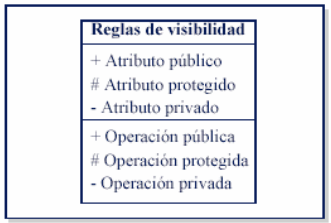
- Temas avanzados:
  - Transmisión de objetos: Consiste en el envío de objetos por un medio de comunicación cualquiera entre espacios de direccionamiento.
  - Objetos espejo (proxies): Consiste en una alternativa a la transmisión de objetos, ya que permite manipular un objeto espejo como si se estuviera manipulando un objeto remoto con el que está sincronizado.

Comunicación entre Objetos

- Comunicación entre Objetos:
  - Los objetos interactúan para realizar las funciones de la aplicación.
  - Podemos tener los siguiente tipos de mensajes:
    - Simple.
    - Síncrono.
    - Asíncrono.
    - Esperado.
    - Cronometrado.

Clases

- Una clase describe un conjunto de objetos con propiedades similares (mismos atributos y comportamientos), relaciones comunes con otros y una semántica común.
  - Las clases son abstracciones que permiten crear categorías de objetos.
  - Cada objeto de una clase es un ejemplar o una instancia de la clase. Cuando creamos objetos concretos de una clase estamos realizando un proceso de instanciación.
  - Se programan las clases, NO los objetos
  - La encapsulación de los datos y del comportamiento en una misma unidad. Define diferentes niveles de visibilidad o accesibilidad:
    - Nivel privado: visibilidad para la propia clase.
    - Nivel protegido: visibilidad para las clases derivadas.
    - Nivel público: visibilidad para todas las clases.

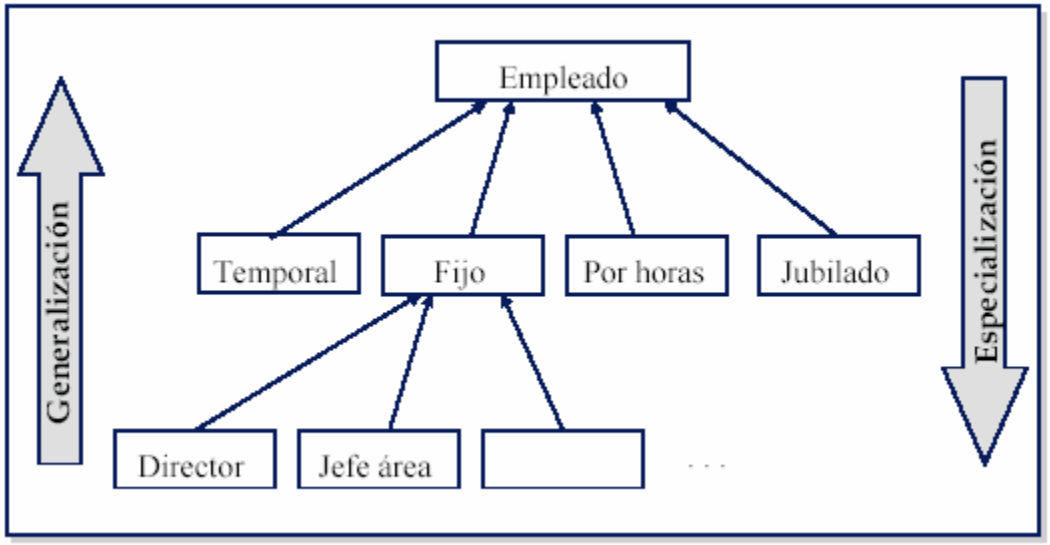


Relaciones entre Clases

- Relaciones entre clases:
  - **Asociación**: conexión semántica bidireccional entre clases. Una asociación describe un grupo de enlaces con una estructura y una semántica comunes. Se entiende por enlace una conexión física o conceptual entre instancias. Por ejemplo un enlace sería ‘Rosa trabaja en Telefónica’ y una asociación ‘Persona trabaja en Empresa’.
  - **Agregación**: conexión bidireccional asimétrica. La agregación es una forma de asociación que permite representar relaciones del tipo amo-esclavo, todo-partes o agregado-componentes. Por ejemplo ‘Departamento forma parte de Empresa’. Existe la agregación recursiva. Ejemplo ‘Persona es jefe de Persona’.
  - **Composición**: Relación semántica que describe una forma de agregación con un matiz fuerte de propiedad y de coincidencia de vida como partes de un todo. Por ejemplo ‘Panel, Cabecera y Deslizador componen Ventana’.

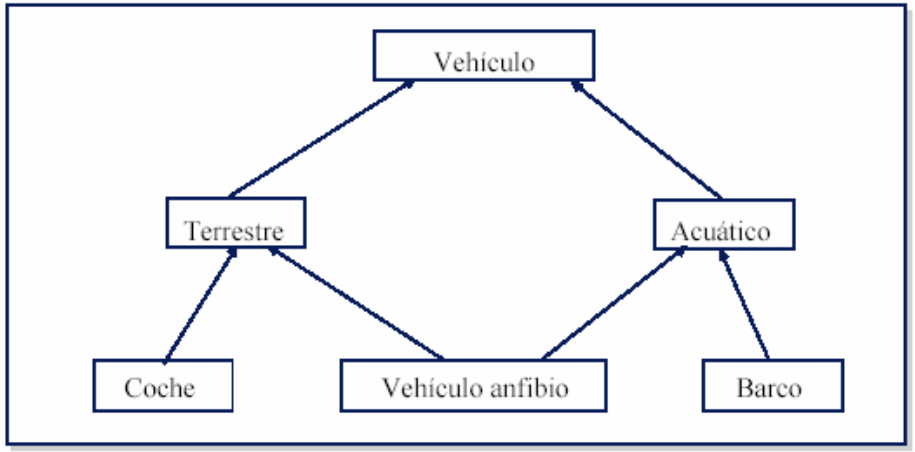
Relaciones entre Clases

- Generalización y especialización.
  - La generalización consiste en unir los elementos comunes de varias clases en una más general llamada superclase.
  - La especialización permite capturar las particularidades de un conjunto de objetos no discriminados por las clases ya identificadas en una subclase.



Relaciones entre Clases

- Herencia
  - La herencia es el mecanismo por el que las clases refinadas incorporan las características de una o más clases superiores en la estructura jerárquica. Una subclase heredará atributos y operaciones de las superclases, más las suyas propias.
  - Hay dos tipos de herencia:
    - **Herencia simple**: una clase sólo puede tener una superclase.
    - **Herencia múltiple**: una clase puede tener más de una superclase y heredar características de todos sus antecesores.



Encapsulamiento

- Encapsulamiento
  - Consiste en agrupar todos los datos y operaciones relacionadas en una misma clase. Esta propiedad facilita que aparezcan otras características de la programación orientada a objetos como la reutilización y la ocultación de información.
  - De esta manera, y debido a la ocultación de la información, los usuarios de esta clase disponen de unos métodos que permiten consultar y modificar el comportamiento de esta clase, pero no tienen acceso directo a los datos.

Polimorfismo

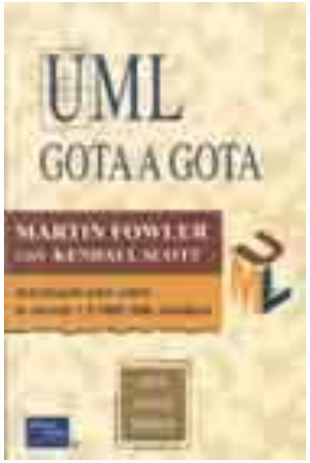
- Propiedad por la que una operación se comporta de forma diferente en la misma o diferentes clases por medio de la herencia, por tanto existe la capacidad de que un mensaje sea interpretado de maneras distintas según el objeto que lo recibe.
- El polimorfismo en los lenguajes OO se puede ver en:
  - Atributos con el mismo nombre (en distintas clases)
  - Métodos con el mismo nombre (en la misma o en distintas clases)
  - Capacidad de ciertos elementos para recibir distintos objetos mediante una sola definición. P. e.:
    - Un método declara un atributo de tipo interfaz → Puede recibir un objeto de cualquier clase que implemente ese interfaz
    - Un método declara un atributo de una determinada clase → Puede recibir un objeto de cualquiera de sus subclases

Otras propiedades…

- Genericidad
  - Es la propiedad que permite definir métodos que tienen como parámetros elementos de cualquier tipo.
- Abstracción
  - Se reduce la complejidad mediante la abstracción, es decir, la eliminación de información redundante e irrelevante.
  - Se abstrae hasta el nivel de información que es necesario.
- Tipificación estricta
  - Éste es un concepto que, aunque no es específico de la programación orientada a objetos, es necesario cumplir. Este hecho implica que si dos expresiones están relacionadas, tanto si se trata de una asignación como si se trata de cualquiera de las operaciones que se pueden hacer sobre expresiones, tienen que coincidir en tipo.
  - Si no lo hacen, el compilador genera un error en tiempo de compilación

Bibliografía

- Fowler, M., Scott Kendall, UML Gota a gota, Adison-Wesley, 2000
  - En inglés: UML Distilled, 3rd Edt, 2004



- Meyer, B., Construcción de software Orientación a objetos“, Prentice-Hall, 1999