UML: Diagramas Estructurales

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2021-2022)

Créditos I

- Las siguientes imágenes e ilustraciones son libres y se han obtenido de:
  - Emojis, https://pixabay.com/images/id-2074153/
  -

https://pixabay.com/images/id-1044090/
  -

https://pixabay.com/images/id-4129246/
  -
https://pixabay.com/images/id-3480187/
  -
https://pixabay.com/images/id-36561/
  - UNIFIED MODELING LANGUAGE https://www.uml.org

Créditos II

- https://pixabay.com/images/id-3846597/

- El resto de imágenes e ilustraciones son de creación propia, al igual que los ejemplos de código

Objetivos

- Saber interpretar un diagrama de clases
  - Cada clase individualmente
  - Y las relaciones entre ellas
- Saber implementarlo
- Entender la semántica de un diagrama de clases
- Aprender diseño analizando los diagramas de clases que se os proporcionen

Contenidos

1. **Introducción**
   - UML
2. **Diagrama de clases**
3. **Relaciones entre clases**
   - Asociación
   - Dependencia
4. **Diagrama de paquetes**

Introducción

- Muchas disciplinas usan lenguajes para expresarse eliminando en parte la ambigüedad del lenguaje natural

UML

en el plano de una casa, hay una manera de dibujar las puertas, las bolas... pero no hay un número fijo de puertas que dibujar.

- UML es un **"Lenguaje Unificado de Modelado"**, un lenguaje de diseño y no una metodología
- Es independiente del lenguaje de programación con el que posteriormente se implemente el diseño
- Permite:
  - Especificar mediante modelos las características de un sistema antes de su construcción
  - Visualizar gráficamente un sistema software de forma que sea entendible por diversos desarrolladores mucho antes de tener el código
  - Documentar un sistema desarrollado para facilitar su mantenimiento, revisión y modificación sirve como documentación
- Dispone de una amplia variedad de diagramas. Propósitos:
  - Modelar la estructura de un sistema, su comportamiento dinámico, los productos resultantes de un proyecto, etc.

Diagrama de clases

atributos
métodos - nada de info extra

- Muestra las clases y sus relaciones
- Tipos de relaciones:

  (en esta lección)

  - Asociación
  - Dependencia

  (cuando veamos herencia)

  - Generalización
  - Realización

Representación de una clase

ClaseEjemplo

-deClase : long
+publico : float = 100
#protegido : float
~paquete : OtraClase [1..*]
-privado : boolean

colección (no vector)
nota: simplificamos las prácticas y usamos:
Java ArrayList
Ruby array

+metodoClase(a : int) : void
+deInstanciaPublico(a : float, b : int[]) : int
-deInstanciaPrivado()

Representación de una clase

Relaciones entre clases

- **Asociación**

  - Modela una relación estructural fuerte y duradera en el tiempo
  - Las asociaciones generan atributos de referencia

→ Error MUY común: No añadir atributos de referencia

  - Cardinalidad / multiplicidad:
    - Se representa con números (pueden definir un rango)
    - Indica cuántas instancias de la clase situada en un extremo están vinculadas a una instancia de la clase situada en el extremo opuesto
    - Si no se indica nada, por defecto su valor es 1
  - Navegabilidad:
    - Se representa con puntas de flecha
    - Indica si es posible *conocer* la/s instancia/s relacionadas con la instancia de origen
    - Si no se indican flechas, por defecto las relaciones son bidireccionales

Ejemplo de asociación

Ejemplos de asociaciones

Clases asociación

- Los vínculos entre las instancias pueden llevar información asociada
- Una asociación puede modelarse como una clase, cada enlace se convierte entonces en instancia de dicha clase

asociado a la relación
(un trabajador puede trabajar para varias empresas)
(claramente, cada empresa puede relacionarse con más de un trabajador)

Clases asociación

- Ambos diagramas son equivalentes

este diagrama da más info aunque haya que implementarlo como antes (las clases no son de la misma naturaleza)

Asociación: Propiedades de los extremos

- En los extremos de las asociaciones se pueden indicar propiedades
- Las más comunes con multiplicidad mayor a 1 son:
  - {ordered} para indicar que se trata de una secuencia ordenada
  - {unique} para indicar que los elementos no se repiten

Especificaciones de las asociaciones

Asociaciones especiales

- **Agregación**
  - Una de las clases representa el TODO y las otra las PARTES
  - La cardinalidad en el TODO puede ser cualquiera
  - Un objeto PARTE podría estar en varios TODO ...
  - ... o en ninguno

Asociaciones especiales

- **Composición**
  - Agregación fuerte donde las PARTES no tienen sentido sin el TODO
  - La cardinalidad en el TODO debe ser 1
  - Un objeto PARTE NO puede estar en varios TODO
  - Tampoco puede estar en ningún TODO

Si un banco desaparece, sus cuentas también

Relaciones entre clases

ASOCIACIÓN ≠ ~~X~~

- **Dependencia** ----------> no genera atributos de referencia

Ejemplo en los objetos TOI (dependencia bidireccional)

  - Modela una relación débil y poco duradera en el tiempo
  - Cuando desde una clase *se utilizan* instancias de otra clase
  - Ejemplos
    - ★ Un método de una clase recibe como parámetros instancias de otra clase
    - ★ Un método de una clase devuelve una instancia de otra clase
  - Si se modifica la interfaz externa de una clase podrían verse afectadas todas las que dependen de ella
  - No genera atributos

→ Error común: Añadir ~~*atributos de dependencia*~~

  - Dirección de la dependencia:
    - ★ Se representa con puntas de flecha
    - ★ Indica que una clase utiliza a la otra

Ejemplo de dependencia

Diagrama de paquetes

- Permiten expresar relaciones de dependencia entre paquetes
- *Recordar:*
  - Los paquetes son agrupaciones
  - Pueden agrupar clases y otros paquetes
    - En Java no existen los subpaquetes

Diagrama de clases → *Diseño* ←

- Ya sabemos interpretar un diagrama de clases (DC)
- También sabemos implementarlo
- Pero, ¿cómo realizamos un DC para un problema concreto?
  - Para ello tenéis que:
    - ★ Entender bien el problema, los requerimientos que plantea
    - ★ Determinar qué clases (responsabilidad, atributos y métodos) van a modelar dicho problema
    - ★ Determinar cómo se relacionan unas clases con otras

  Objetivo: Cumplir con los requerimientos planteados

  - En definitiva, hay que realizar INGENIERÍA DEL SOFTWARE

★ Todo esto lo aprenderéis en la asignatura

Fundamentos de Ingeniería del Software

Diagrama de clases → *Diseño* ←

- No obstante, una vez entendido, sí deberíais ser capaces de modificar un DC ante pequeños cambios en el problema
- Cuando implementéis un DC (por ejemplo, en prácticas)
  - No os limitéis a la parte sintáctica (flechas, cajas, símbolos, etc.)
  - No os preocupéis solamente por *traducir* el DC a código
  - Entender el DC desde el punto de vista semántico
    - Observar cómo el DC modela el problema

★ Aprender diseño analizando los DC que se os proporcionen

UML: Diagramas Estructurales

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2021-2022)