MODUL 7 :
Refining Requirement Model

Contents

7.1 Pendahuluan 2

7.2 Software & Specification Reuse 2

7.3 Mengidentifikasi dan Memodelkan Generalisasi, Komposisi, dan Agregasi 3

7.3.1 Aggregasi dan Komposisi 3

7.3.2 Generalisasi 4

7.4 Software Development Pattern 5

Referensi 8

7.1 Pendahuluan

Dalam hal menyempurnakan *requirement* bagaimana *object Oriented(OO)* mampu berkontribusi dalam penggunaan ulang software (Reuse). Pada pertemuan awal ketika membahas mengenai komponent OO kita mengenal istilah *Inheritance* dan *Encapsulasi*. Kedua hal ini lah berperan dalam penggunaan ulang software. *Inheritance* atau pewarisan adalah bagaimana membentuk objek baru dari objek yang memiliki sifat dengan objek yang sudah ada sebelumnya. Encapsulation adalah menyembunyikan implementasi dari client, sehingga client hanya tergantung pada interface. *Encapsulasi* akan melindungi sebuah program dari akses ataupun intervensi dari program lain yang mempengaruhinya.

7.2 Software & Specification Reuse

Pengembangan perangkat lunak telah berkonsentrasi pada menciptakan solusi baru. Baru-baru ini, penekanan telah bergeser. Banyak perangkat lunak kini dirakit dari komponen yang sudah ada. Menggunakan kembali komponen dapat menghemat uang, waktu dan usaha. Mencapai reuse masih sulit karena :

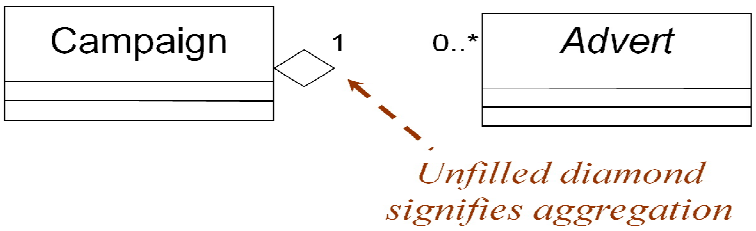
- Reuse tidak selalu tepat - tidak bisa menganggap komponen yang ada memenuhi kebutuhan baru
- Model organisasi membuat sulit untuk mengidentifikasi komponen-komponen yang sesuai
- NIH (Not-Invented-Here) sindrom
- Persyaratan dan desain yang lebih sulit dari kode untuk menggunakan kembali

Dalam hal kontribusi Reuse dengan Objek Oriented, Encapsulation membuat komponen lebih mudah untuk digunakan pada sistem yang awalnya tidak mereka rancang. Agregasi dan komposisi dapat digunakan untuk merangkum komponen. Generalisasi memungkinkan penciptaan kelas khusus yang baru bila diperlukan

7.3 Mengidentifikasi dan Memodelkan Generalisasi, Komposisi, dan Agregasi

7.3.1 Aggregasi dan Komposisi

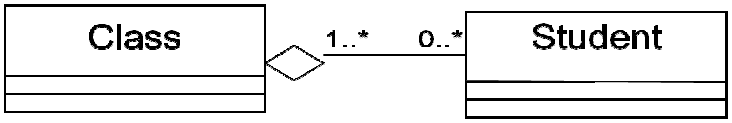
Jenis khusus dari asosiasi, baik kadang-kadang disebut whole-part. Sebuah kampanye terdiri dari beberapa iklan



Relasi Aggregation whole-part adalah yang esensial. Semantics bisa sangat tidak tepat

Composition 'stronger' adalah Setiap bagian mungkin milik satu bagian yang lebih besar pada satu waktu tertentu. Jika bagian yang besar itu rusak maka akan merusak semuanya

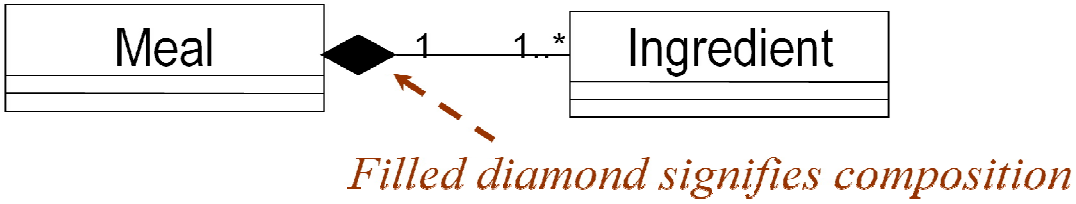
contoh:



Jelas tidak Komposisi:

- Siswa bisa dalam beberapa kelas
- Jika kelas dibatalkan, siswa masih tetap ada!
-

Contoh lainnya:



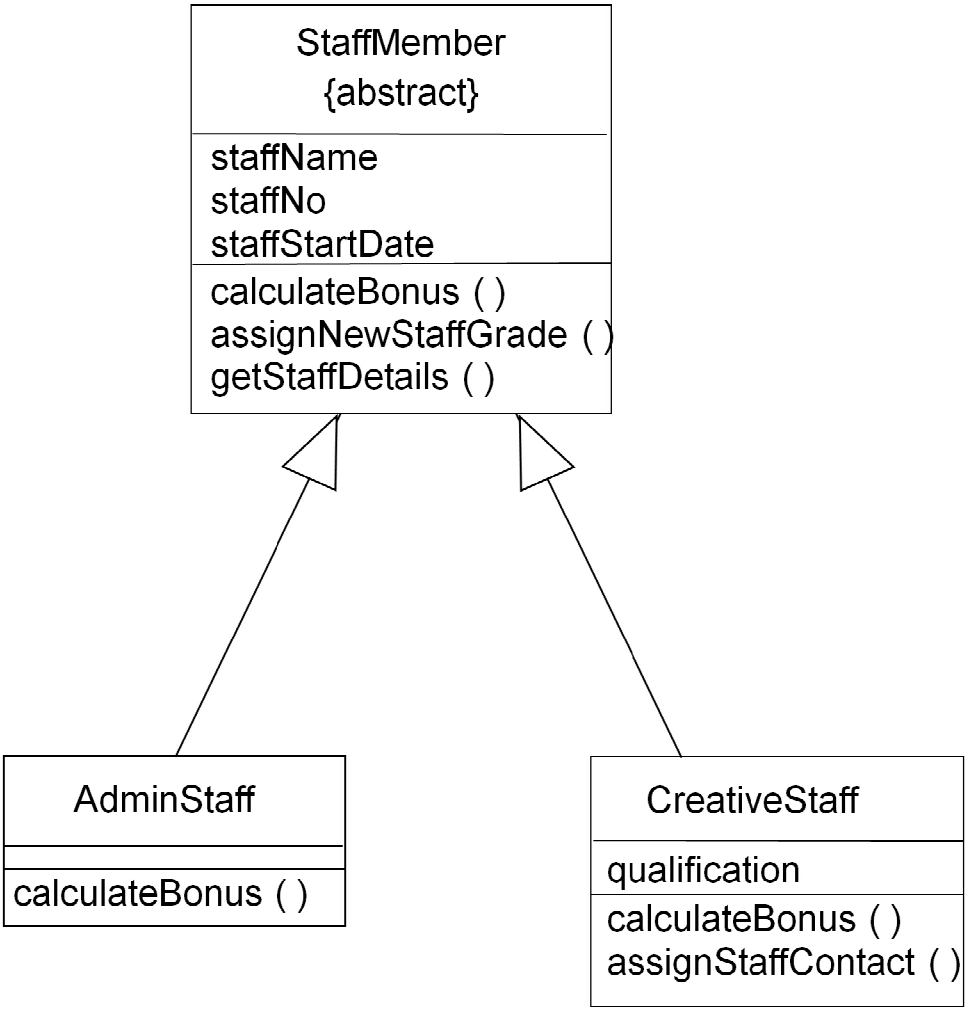
ini (Mungkin) Komposisi:

- Bahan-bahan makanan ini berada dalam satu menu makanan
- Jika makan malam ini jatuh maka kita akan kehilangan seluruh bahan makanan tersebut

7.3.2 Generalisasi

Menambah struktur generalisasi ketika :

- Dua kelas yang sama di sebagian besar rincian, tetapi berbeda dalam beberapa hal
- Mungkin berbeda:
    - Dalam perilaku (operasi atau metode)
    - Dalam data (atribut)
    - Dalam asosiasi dengan kelas lain



7.4 Software Development Pattern

" A pattern:

describes a problem which occurs over and over again in our environment, and then describes the core of a solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."

Alexander et al. (1977)

Menjelaskan masalah yang terjadi berulang-ulang di lingkungan kita, dan kemudian menjelaskan inti dari solusi untuk masalah tersebut, sedemikian rupa bahwa Anda dapat menggunakan solusi ini jutaan kali, tanpa pernah melakukannya dengan cara yang sama dua kali."

Pola yang ditemukan pada banyak titik dalam siklus hidup pengembangan sistem:

- Pola analisis adalah kelompok konsep berguna dalam persyaratan pemodelan
- Pola arsitektur menggambarkan struktur komponen utama dari sistem perangkat lunak
- Pola Desain menggambarkan struktur dan interaksi komponen perangkat lunak yang lebih kecil

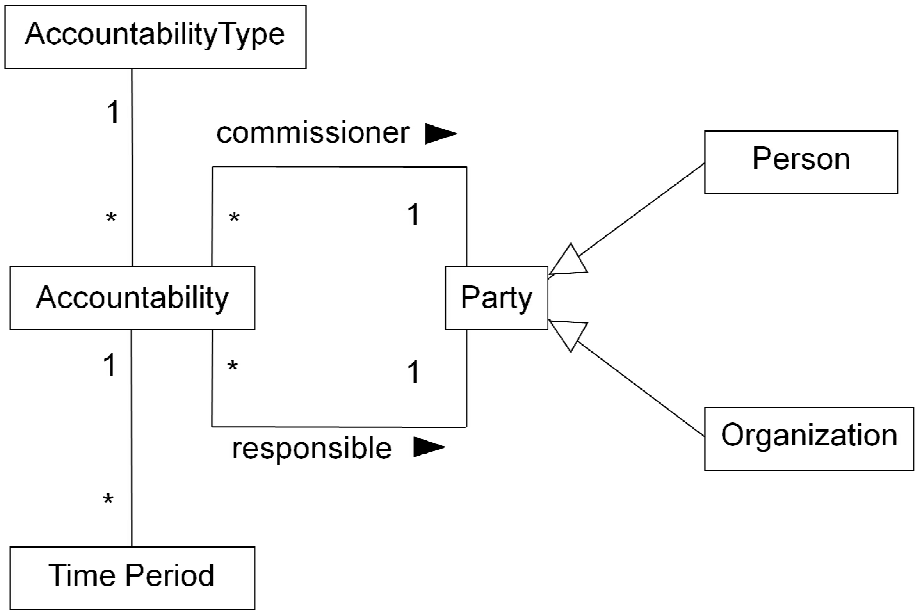
Pola telah diterapkan secara luas dalam pengembangan perangkat lunak:

- Pola Organisasi menggambarkan struktur, peran dan interaksi dalam organisasi pengembangan perangkat lunak itu sendiri

Anti-pola praktik dokumen kurang baik. Mushroom Management adalah sebuah organisasi anti-pola

Bentuk Analisa Pattern sederhana

Bentuk Analisa Pattern Akutanbilitas


AccountabilityType
1
*
commissioner
*
1
Person
Accountability
Party
1
*
1
responsible
Organization
*
Time Period

Referensi

1. Simon Bennet, Steve McRobb and Ray Farmer, *Object Oriented Systems Analysis and Design Using UML,* Edisi 3. ; McGraw Hill, 2006. (SB)