Curs 03

UML

Există instrumente `CASE` (Computer Aided Software Engineering) ce au la baza `UML`: Rational Rose, Magic Draw, Visio, Visual Paradigm (...). Date fiind diagrame corect specificate ştiu să genereze codul sursă în diverse limbaje de programare.
`Reverse engineering`: pe baza codului sursă dat, generează diagrame.

Diagramă de cazuri de utilizare:

- indică utilizatorii şi cum interacţionează aceştia cu sistemul.
- actori (clase de utilizatori ai sistemului) - Bucatar, Casier, Chelner
- cazuri de utilizare (scenarii particulare de utilizare a sistemului)
- se desenează ca o elipsă şi are ca nume de obicei un verb care indică o secvenţă de scenarii care se poate executa cu sistemul software.
- un actor are unul sau mai multe cazuri de utilizare.

Cazurile de utilizare pot fi descrise urmărind un şablon:

- precondiţii
- postcondiţii
- autorul
- istoric
- probleme de implementare
- puncte de extensie (cum poate fi extins cazul de utilizare prin diverse implementări)

Între cazuri de utilizare se pot stabili relaţii:

1. `<Extension>` - Dacă în secvenţa de executare a unui caz de utilizare pot apărea situaţii excepţionale, acestea vor fi reprezentate ca puncte de extensie. Sunt implementate ca apeluri de metode condiţionate de instrucţiuni de tip `IF` (sau `Switch`).
2. `<Include>` - Indică faptul că un caz de utilizare face parte din alt caz mai complex. (se pune în evidenţă stilul de modularizare / ierarhizare a cazurilor mai complexe)

Relaţii între actori: Generalizare
Spunem că actorul `A` este o generalizare a actorului `B` dacă `B` poate efectua cu sistemul tot ce executa `A` şi scenarii suplimentare.

Diagrame de clase:
O diagramă de clasă indică structura statică a sistemului. Intervine în etapa de proiectare şi implementare (`design` & `code`). Arată care sunt clasele / interfeţele sistemului şi relaţiile dintre acestea.

Simbol grafic: dreptunghi (cu trei compartimente)

- nume
- atribute (variabele)
- metode
    - atributele şi metode se declară după sintaxă

```
[specificator_acces]nume:[tip]
+: public
-: private
#: protected
```

- datele / metodele de tip static se vor sublinia

Relaţii între clase

1. Asocierea

Spunem că între clasele `A` şi `B` există relaţie de asociere dacă obiectele din clasa `A` au conştiinţa despre obiectele din clasa `B` şi / sau invers.

Desenam asociere între clasele `A` şi `B` dacă e îndeplinită cel puţin una din următoarele condiţii:

- clasa `A` conţine o referinţă către clasa `B` de tip variabilă membru.

```
class A {
  private B _refB;
}
```

- când clasa `B` apare ca parametru al unei metode din clasa `A`

```
class A {
  public void Method(B refB) { }
}
```

- clasa `B` apare ca tip returnat al unei metode din `A`

```
public B method() {
  ///...
  return refB;
}
```

- clasa `B` apare ca o excepţie într-o metodă a clasei `A`

```
public void method() {
  try{}
  catch(B refB)
}
public void method() {
  throw new B();
}
```

- clasa `B` apare ca variabilă locală într-o metodă din clasa `A`

```
class A {
  public void method() {
    B refB;
    //....
  }
}
```

Agregarea şi Compunerea

Sunt cazuri particulare de asociere prin care se indică o relaţie de tip parte-întreg.

Agregare: un obiect al clasei `A` conţine unul sau mai multe obiecte de clasa `B`.

În cazul agregării, un obiect poate să aparţină la mai mult de un întreg. Prin urmare, poate exista de sine stătător.

Compunerea este un caz particular de agregare, relaţia mai puternică în sensul că întregul depinde de parţile sale componente şi invers, o componentă nu există decât într-un întreg.
Daca dispare intregul (este distrus, isi termina durata de viata), dispar si partile componente. Acestea nu pot exista in alte obiecte!
Cand este creat intregul, sunt create si componentele!

Generalizarea (Specializarea)

O clasă `A` este generalizare a unei `B` dacă putem spune că obiectele clasei `B` sunt "un fel de" obiecte de tip `A`.

"Un fel de" - se referă la comportament, nu la atribute!! Adică, obiectele clasei `B` se comportă ca obiecte de clasa `A`!!

OBSERVAŢIE: generalizarea permite reutilizarea codului. **DAR** nu trebuie utilizată în situații nepotrivite (a nu se abuza de generalizare). Atenție, generalizarea introduce un grad ridicat de dependența între clase. Clasele `A` și `B` sunt strâns dependente.

IMPORTANT: De câte ori avem nevoie de funcționalitățile unei clase, e de preferat să utilizăm **AGREGARE** / **COMPUNERE** în detrimentul moștenirii!!

Tehnici de proiectare a claselor:

Q: Fiind date cerinţele sistemului, cum găsim clasele / interfeţele şi relaţiile dintre clase?
A: GREU.

Tehnica identificării substantivelor.

Sunt analizate cerinte descrise în limbaj natural şi sunt evidenţiate substantivele din text care au legătură cu sistemul. Obţinem o primă listă de clase candidat. Dintre acestea sunt eliminate cele foarte generale. Această tehnică oferă o primă listă de clase, fără să indice nimic referitor la relaţiile dintre clase.

Tehnica cardurilor CRC

Un card `CRC` (Class, Reponsibility, Colaboration) indică:

- responsabilităţi principale ale clasei
- relaţiile cu alte clase (asocieri, compuneri, agregări, generalizări)

Din responsabilităţi se pot deduce atributele şi metodele! (responsabilităţile devin metode în clasa respectivă). Din colaborări se deduc relaţiile cu alte clase!