

La progettazione orientata agli oggetti utilizza e può modificare i risultati dell'analisi per questioni implementative. Queste modifiche però devono essere ridotte al minimo per mantenere coerenza con l'analisi e con le specifiche.

Nella progettazione logica ora si studia come deve funzionare il sistema.

Siccome gli strumenti CASE non supportano bene la distinzione tra ooa e ood, si può fare in modo che l'analisi diventi a mano a mano progettazione (ma in questo modo si perde l'analisi originale) oppure si può generare l'ood a partire dall'ooa, ma mantenendolo distinto.

In questa fase è ora necessario definire il tipo di dato (int, string, boolean, ...), le strutture dati, l'implementazione delle relazioni tra le classi, le operazioni per implementare il sistema, gli algoritmi e la visibilità di classi, attributi, operazioni.

Nel caso di associazioni 0-1 o 1-1 bisogna aggiungere al cliente un attributo che riferisce all'oggetto della classe del fornitore, e il valore dell'oggetto della classe fornitore (in caso di composizione).

Nel caso di associazioni 0-N o 1-N si utilizza una classe contenitore in cui le istanze sono collezioni a oggetti della classe fornitore, inoltre viene comunque aggiunto al cliente un attributo che rappresenta per valore o per riferimento l'istanza della classe contenitore.

Un classe contenitore è una classe le cui istanze appartengono ad altre classi. Se gli oggetti contenuti sono in numero fisso e senza ordine, allora si può utilizzare un vettore; viceversa, se gli oggetti sono in numero variabile, o è chiesto che abbiano un ordine, allora occorre una classe contenitore. Queste classi contenitore hanno funzionalità minime (memorizzare, aggiungere, togliere, trovare un oggetto, enumerare gli oggetti) e possono essere classificati in base al modo in cui contengono gli oggetti, o all'omogeneità/eterogeneità con degli oggetti contenuti.

Contenimento per valore: l'oggetto contenuto è memorizzato nella struttura dati del contenitore, ed esiste proprio perchè è contenuto fisicamente in un altro oggetto. Quando un oggetto viene inserito nel contenitore, viene duplicato, e la distruzione del contenitore implica la distruzione degli oggetti contenuti.

Contenimento per riferimento: l'oggetto contenuto ha una propria esistenza e può essere contenuto in diversi contenitori, infatti quando viene inserito nel contenitore non viene copiato, ma ne viene memorizzato il riferimento. Per questo motivo, alla distruzione del contenitore, l'oggetto contenuto non viene eliminato.

Contenimento di oggetti omogenei (valore o riferimento): è sufficiente l'uso di template (classi generiche o parametriche). Il tipo degli oggetti contenuti viene lasciato generico e si pensa soprattutto agli algoritmi di gestione della collezione. Quando serve una classe contenitore di oggetti di una classe specifica, basta istanziare una classe generica specificando il tipo dell'oggetto.

Contenimento di oggetti eterogenei (riferimento): è necessario utilizzare l'ereditarietà e sfruttare la proprietà che un puntatore alla superclasse può puntare alle istanze di qualunque sottoclasse. La classe contenitore può essere generica, ma solo per la gestione dei riferimenti agli oggetti contenuti.

Implementazione delle associazioni:

Nell'ooa le associazioni di contenimento/agggregazione hanno una direzione ben precisa; nell'ood le associazioni possono essere monodirezionali (come per l'ooa) o bidirezionali (cioè da A si può accedere a B e da B si può accedere velocemente ad A).

La bidirezionalità si ottiene aggiungendo una struttura dati di B: un puntatore all'oggetto che lo contiene, o una lista di oggetti (nel caso di contenimento

multiplo). Questo porta all'utilizzo di strutture dati con puntatori incrociati: bisogna darci un occhio perché potrebbero generare inconsistenze.

Un modo alternativo per implementare un'associazione è per mezzo di un dizionario, un particolare contenitore che associa una chiave unica al suo rispettivo valore. Partendo dalla chiave deve essere possibile trovare il valore associato.

Identificazione degli oggetti:

Un oggetto può contenere un riferimento (univoco) ad un altro oggetto.

Deve quindi essere possibile identificare in modo univoco un oggetto affinché possa essere associato ad un altro oggetto.

Nel caso di strutture contenute nello spazio di indirizzamento dell'applicazione, un oggetto può essere identificato dalla sua posizione in memoria.

Nel caso di db o sistemi distribuiti, ad ogni oggetto può essere associato un identificatore persistente attribuito alla creazione dell'oggetto e inizializzato con un valore generato automaticamente dal sistema o con il valore della chiave primaria di una tabella relazionale.

Controllo della visibilità:

Attributi: devono essere dichiarati private (o protected).

Metodi: se devono essere visibili all'esterno devono essere dichiarati public, altrimenti devono essere dichiarati private o protected.

La differenza tra private e protected sta nella visibilità all'interno delle sottoclassi della classe in cui è stato dichiarato il metodo: private non permette la visualizzazione all'esterno di quella classe, mentre protected permette la visualizzazione dell'attributo o del metodo anche all'interno delle sottoclassi.

Classe radice:

Nei casi in cui classi eterogenee hanno operazioni o gruppi di attributi simili conviene realizzare un protocollo comune e definire una classe radice da cui derivano tutte le classi con gli attributi/metodi simili. In java di solito tutte le classi derivano da Object, mentre in .NET da System.Object.

Ereditarietà:

In molti casi il linguaggio di programmazione non supporta l'ereditarietà multipla (che in analisi è ammessa). E quindi necessario convertire queste strutture in altre che abbiano solo ereditarietà semplice.

1) si può scegliere la superclasse più significativa ed ereditare da questa: le altre superclassi diventano dei ruoli e sono associate col contenimento alla superclasse. In questo modo le caratteristiche delle superclassi escluse vengono memorizzate nella classe specializzata tramite contenimento e non tramite ereditarietà.

2) si può appiattare tutto in una gerarchia semplice: così una o più relazioni di ereditarietà si perdono e gli attributi/operazioni devono essere ripetuti nelle classi specializzate.