OOAD

Instructor:

Agenda

- General Introduction
- Object Modeling Technique (OMT)
  - Object-Oriented Analysis
  - Object-Oriented Design
- Three models
  - Object Modeling
  - Dynamic Modeling
  - Functional Modeling
- Four phases

General Introduction
Object-Oriented Software Development

- Object-Oriented Methodology
  - development approach used to build complex systems using the concepts of object, class, polymorphism, and inheritance with a view towards reusability
  - encourages software engineers to think of the problem in terms of the application domain early and apply a consistent approach throughout the entire life-cycle
- Object-Oriented Analysis and Design
  - analysis models the “real-world” requirements, independent of the implementation environment
  - design applies object-oriented concepts to develop and communicate the architecture and details of how to meet requirements

General Introduction
OMT & UML (OMT turned into UML)

- OMT [Object Modeling Technique - Rumbaugh et al.,1991] consists of
  - building three complementary models of the system
  - adding implementation details to the models
  - implementing the models
- OMT includes a set of
  - phases [processes]
  - diagramming techniques
- OMT has four phases
  - object-oriented analysis builds a real-world model
  - system design determines overall architecture of system
  - object design decides upon data structures and algorithms
  - implementation translates design into programming language

General Introduction
OMT Stages & Models

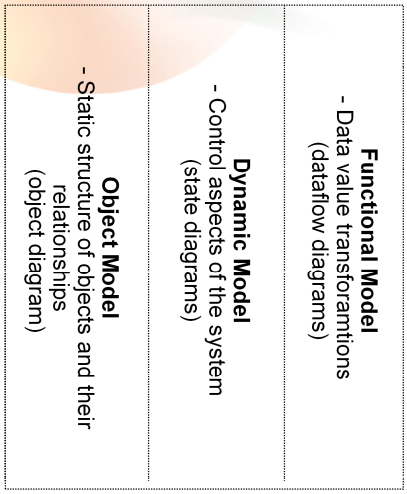
time

Analysis
- Model of real-world situation
- What ?

System Design
- Overall architecture (sub-systems)

Object Design
- Refinement of Design
- Algorithms/data structures to implement each class

Implementation
- Translation of object classes and relationships to a particular object-oriented language



System

General Introduction
Object-Oriented Analysis

- Builds a “real-world” model from requirements
  - client interviews, domain knowledge, real-world experience collected in use cases and other simple notations
- OOA models address three aspects of the system (its objects)
  - class structure and relationships
  - sequencing of interactions and events
  - data transformations and computations

General Introduction
Models of Object-Oriented Analysis

- Class Model
  - static structure
  - what objects are in the system?
  - how are they related?
- Dynamic Model
  - behavioral aspects
  - what events occur in the system
  - when do they occur and in what order?
- Functional Model
  - data transformations
  - “what” does the system do

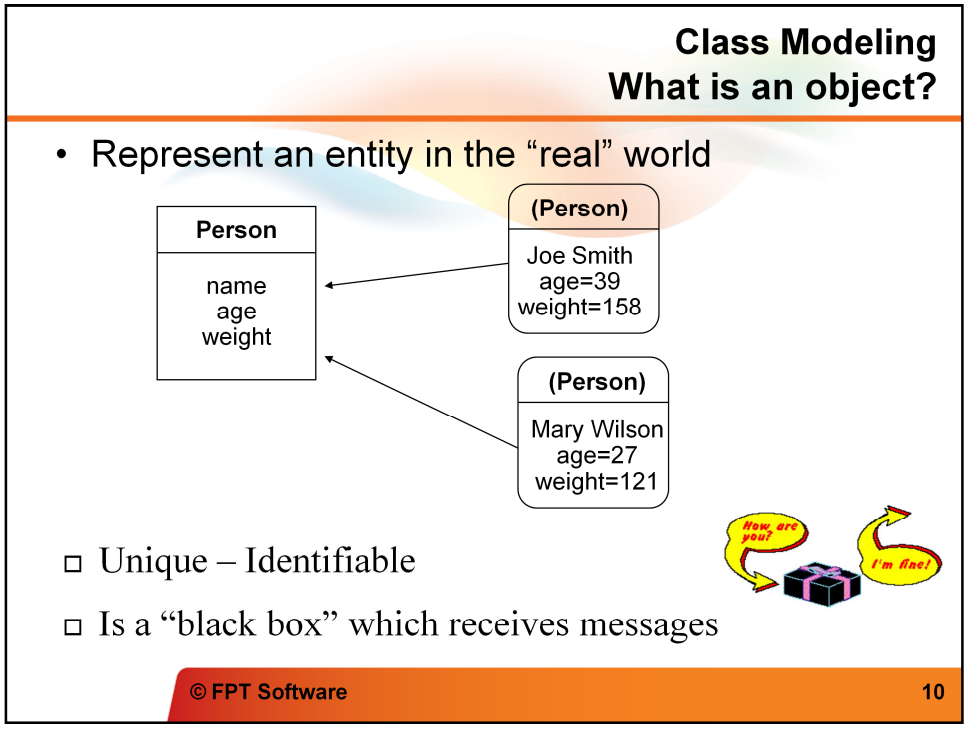
- Data-Oriented
- Action-Oriented
- Both Data and Actions

General Introduction
OO Analysis & Design Steps

- Class Modeling
- Dynamic Modeling
- Functional Modeling
- Add Operations to the Class Model
- Iterate and refine the models
  - After the first iteration, steps may occur in parallel or out of order
  - All models must be kept in synch as changes are made

Class Modeling

- Identify objects and classes
- Prepare a data dictionary
- Identify class attributes and initial set of operations
- Identify associations between objects
- Organize object classes using inheritance



Every object:

- **Contains data: The data stores information that describes the state of the object.**
- **Has a set of defined behavior. This behavior consist of all the things that the object**

"knows" how to do. These are the methods present inside the object.

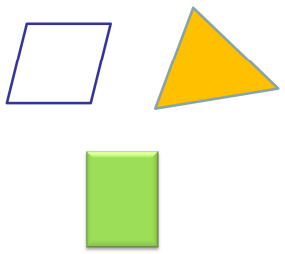
- **Has an individual identity. Each object is different from the other object even if they**

are instantiated from the same class.

A primary rule of object-oriented programming is - as the user of an object, **you would never need**

to know what is there inside the object!

Class Modeling Abstraction



Abstract info

Polygon

Attributes:

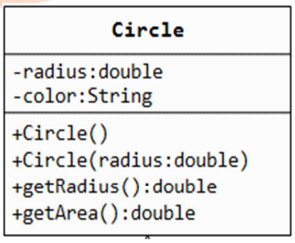
- vertices
- border color
- fill color

Operations:

- draw
- erase
- move

Class Modeling
Classes, Attributes & Operations

- Attributes define the properties of the objects
  - every instance of the class has the same attributes
  - an attribute has a data type
  - the values of the attributes may differ among instances
- Operations define the behavior of the objects
  - action performed on or by an object
  - available for all instances of the class
  - need not be unique among classes



Class	Attributes	Operations
ball	radius, weight	catch, throw
football	air pressure	pass, kick, hand-off
baseball	liveness	hit, pitch, tag

Class Modeling
Encapsulation

- **Allow to show only** the important methods as interface
- **Hide** detail information
  - Hide the data item
  - Hide the implementation
  - Access to data item only through member methods

© FPT Software 13

Capsule vs Tablet

Tablet : Thông tin/thành phần được exposed ra bên ngoài. Thuoc de bi tuong tac voi khong khi va nuoc. Khi uong co the co cam nhan ve mui vi cua thuoc

Capsule: Thành phần được gói kín bên trong. Thanh phan cua thuoc duoc bao ve tot hon. Khi uong, it kha nang cam nhan ve mui vi cua thuoc.

Encapsulation : Separate External Aspect from internal Implementation

Với encapsulation, OO can:

- hide the data item
- Chi co the access data item thông qua member method

Abstraction: Tell the external face we should present to the world

Encapsulation: Ensure implementation is not leak out

Abstraction & Encapsulation together reduce the amount of information we have to deal with.

Benefit:

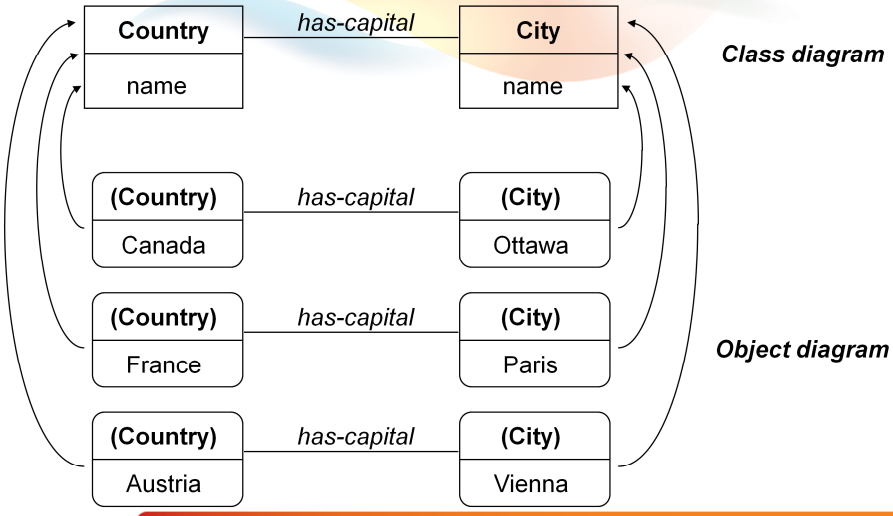
Your brains doesn't have to deal with it unless you're specifically concerned with it

When change occurs, the effects are localized

Class Modeling
Association and Links

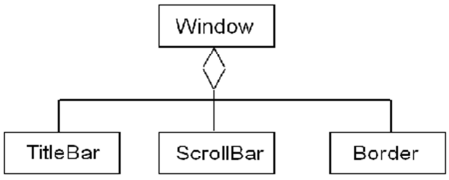
- An association is a relation among two or more classes describing a group of links, with common structure and semantics
- A link is a relationship or connection between objects and is an instance of an association

Class Modeling
Association and Links - Examples



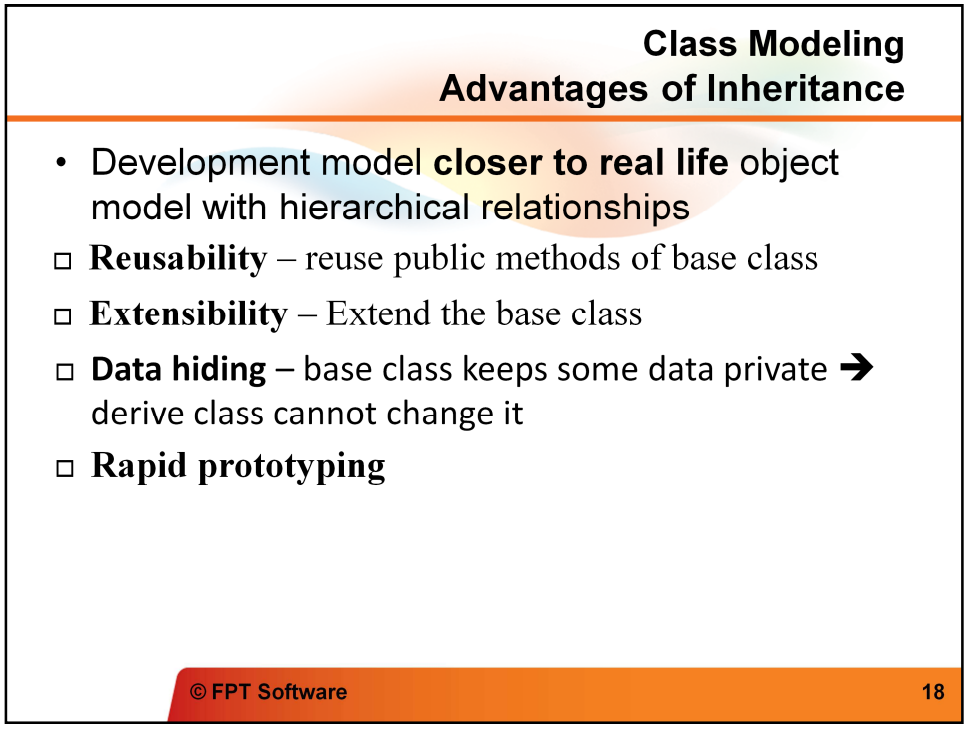
Class Modeling
Aggregation

- Aggregation is a special form of association that indicates a “part-of” relationship between a whole and its parts
- Useful when the parts do not have independent existence
  - A part is subordinate to the whole



Class Modeling
Inheritance

- Classes with a set of similar attributes and operations may be organized into a hierarchical relationship
- Common attributes and operations are factored out and assigned to a broad superclass (*generalization*)
  - generalization is the “is-a” relationship
  - superclasses are ancestors, subclasses are descendants
- A class can be iteratively refined into subclasses that *inherit* the attributes and operations of the superclass (*specialization*)



All benefit apply to design, development and maintenance.

Do you see any disadvantages to inheritance?

Class Modeling
Domain Analysis

- Nouns suggest **candidate Classes.**
- Not every noun is an Object Class.
  - Some are **attributes** of another Class.
  - Some are irrelevant, outside the scope of the application.
- Verb phrases suggest class associations
  - Some relationships are irrelevant (caution).
- Proper nouns suggest Objects of a Class type. Beware of singular nouns.

Class Modeling
Classes Identifying

- Finding classes:
  - Use domain analysis as before.
  - Derive them from the use cases (descriptions).
  - Look for data which must be stored or analysed.
  - Are there external systems?
  - Are there any devices under the control of the system?
  - Are there any organisational parts?

Class Modeling - An Example

FastData Inc. wants a subsystem to process office supply orders via the Web. The user will supply via a form their name, password, account number, and a list of supplies along with an indication of the quantities desired. The subsystem will validate the input, enter the order into a database, and generate a receipt with the order number, expected ship date, and the total cost of the order. If the validation step fails, the subsystem will generate an error message describing the cause of the failure.

We will demonstrate the UML /OMT using this example

Class modeling will be done first

This example does not demonstrate how the technique is applied to ALL problems. Be sure to distinguish between the details of the example and the details of the technique!

Cho sinh vien 5 phut doc vi du.

Class Modeling - Concise Problem Definition

- Define the problem concisely
  - Use only a single sentence

"FastData, Inc. employees may order office supplies via the Web and receive a receipt confirming the order"

- This is the first step towards identifying the classes of the subsystem

Class Modeling - Informal Strategy

- Identify the constraints governing the system
  - Use only a single paragraph

"FastData, Inc. employees may order office supplies via the Internal Web and receive a receipt confirming the order. The order must include the user name, user password, account number, and the list of supplies. A receipt must be generated containing an order number, ship date, and total cost. If the order is valid, it must be entered into an order database. If the order is invalid, an error message must be generated."

- We now have more information to be used in identifying classes for the subsystem

Class Modeling - Formalize the Strategy

• Identify the nouns of the description, which serve as the basis for identifying the subsystem's classes.

- Look for out-of-domain nouns (and throw them out!)
- Look for abstract nouns (use these for attributes)
- The remaining nouns are good candidates!

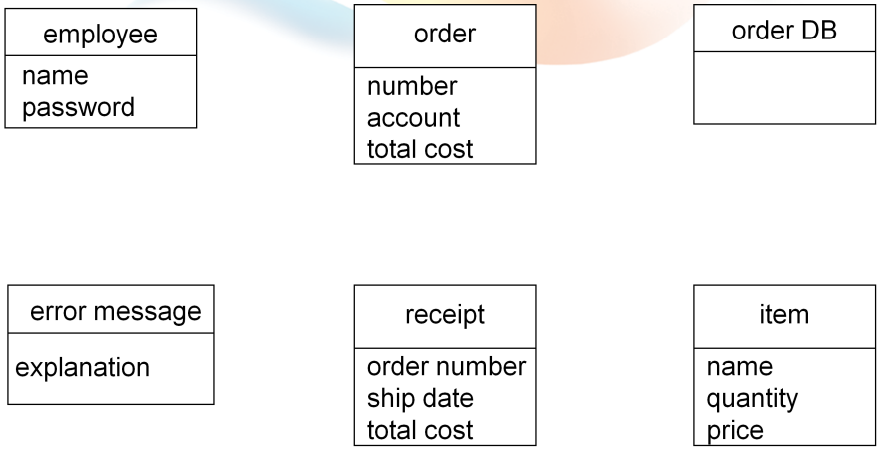
"FastData, Inc. ***employees*** may order ***office supplies*** via the ***Internal Web*** and receive a ***receipt*** confirming the ***order***. The ***order*** must include the ***user name***, ***user password***, ***account number***, and the ***list of supplies***. A ***receipt*** must be generated containing an ***order number***, ***ship date***, and ***total cost***. If the ***order*** is valid, it must be entered into an ***order database***. If the ***order*** is invalid, an ***error message*** must be generated."

Nouns

- Out-of-Domain
  - Internal Web
- Abstract
  - user name
  - user password
  - account number
  - order number
  - ship date
  - total cost
  - list of supplies
  - office supplies -> item
- Good Candidates
  - employee
  - item (was office supplies)
  - receipt
  - order
  - order database
  - error message
- Notes

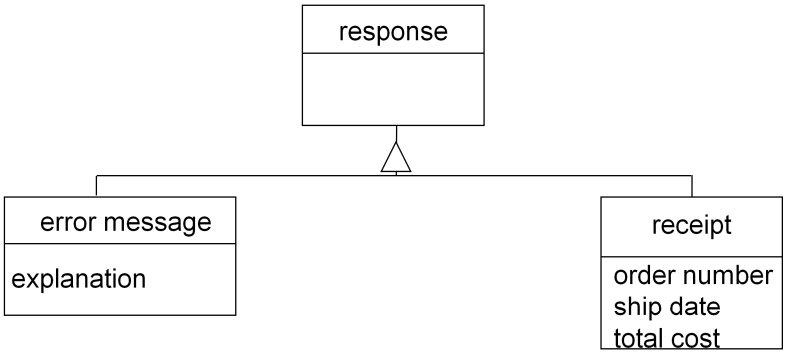
  We have decided not to worry about the Web in this design. Instead we focus on the inputs and outputs and defer the Web details until later.

Class Model

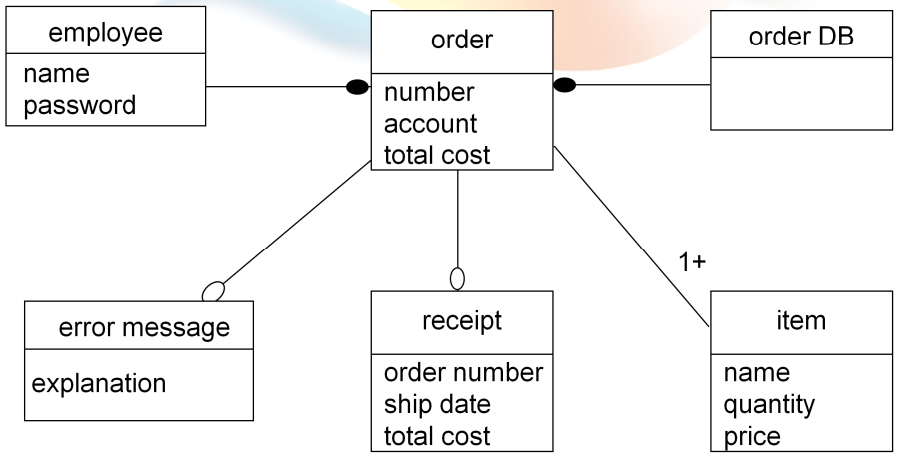

employee
name
password
order
number
account
total cost
order DB
error message
explanation
receipt
order number
ship date
total cost
item
name
quantity
price

Class Model, continued

Since both receipts and error messages will be generated as output it might make sense to have them as subclasses of a more general class. We do not know enough yet to assign it attributes however.



Class Modeling - Relationships



Dynamic Modeling

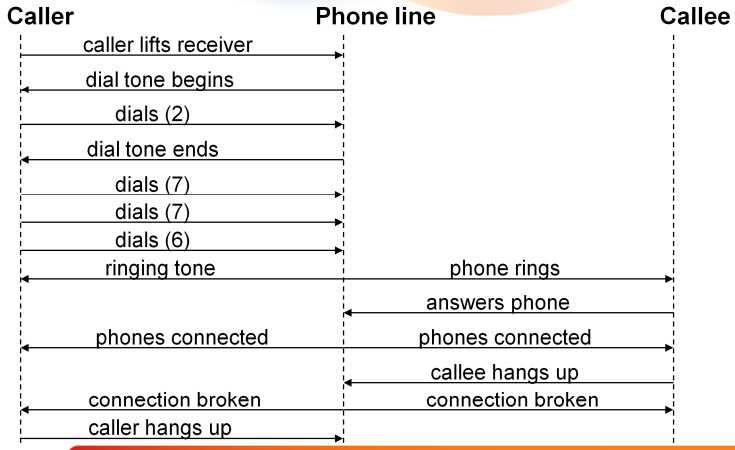
- Prepare scenarios
- Identify events between objects
- Prepare an event trace for each scenario
- Build a state diagram
- Match events between objects to verify consistency

Dynamic Modeling
An example scenario

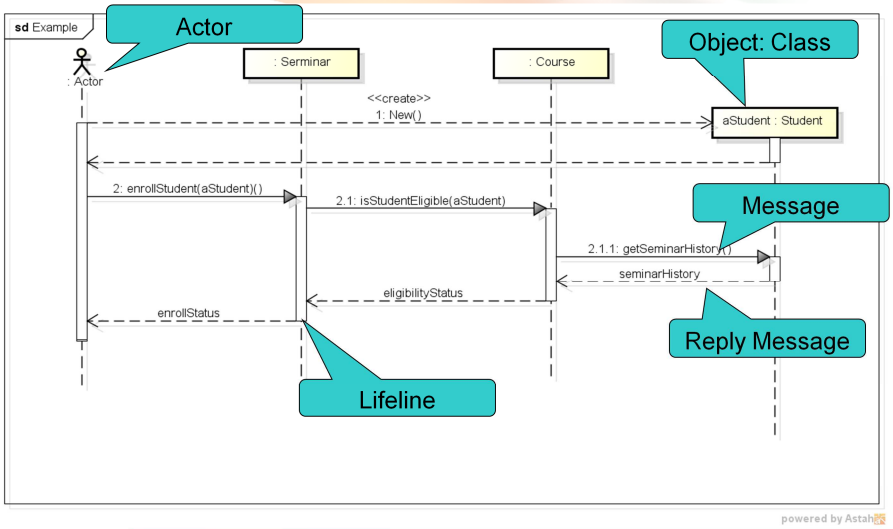
- Scenario for a phone call
  - caller lifts receiver
  - dial tone begins
  - caller dials digit (2)
  - caller dials digit (7)
  - caller dials digit (7)
  - caller dials digit (6)
  - specified phone rings
  - etc.

Dynamic Modeling
OMT Event Trace Notation

- objects are vertical lines
- events are horizontal lines
- arrows indicate sender and receiver
- time passes from top to bottom



Dynamic Modeling
Event Trace ~ Sequence Diagram

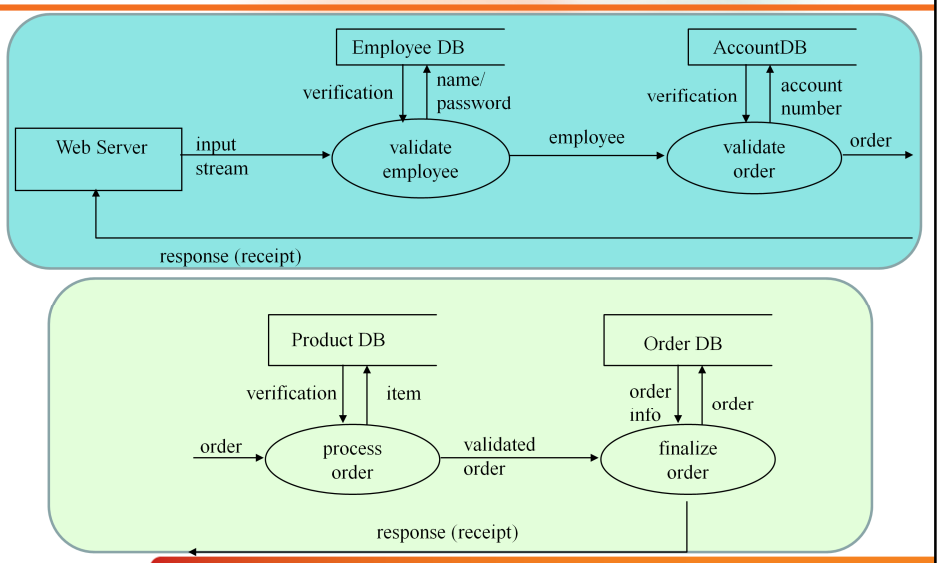


© FPT Software

Functional Modeling
Hierarchical DFD

- High-level functionality iteratively refined into smaller functional units
  - each high-level process may be expanded into a separate DFD
  - top-level processes correspond to operations on complex objects, while lower-level processes are operations on basic objects
- Nesting depth is dependent on application
  - terminates with simple functions
  - each level must be coherent
- Hierarchical DFD corresponds to the following
  - context diagram shows boundaries of system
  - mid-level DFDs show context decomposition
  - primitive DFDs are simple functions that need not be expanded

Functional Modeling
DFD Example: Office Supply



Add Operations to the Object Model

- From the Object Model:
  - Reading/writing object attributes (e.g., get_width, get_height of Rectangle)
- From Events, State Actions, and Activities in the Dynamic Model:
  - Each event sent to an object => operation (e.g., Vending machine: set_balance)
  - Actions/activities may be operations (e.g., Vending machine: do: test item and compute change)
- From Functions in the Functional Model:
  - Each function in the DFD corresponds to an operation (e.g., bank example: subtract withdrawal from Account)

OMT: Four Phases

- System design
  - determines overall architecture of system
- Object design
  - Definition all the classes in the system
- Detailed Design
  - decides upon data structures and algorithms
- Implementation
  - translates design into programming language

OMT: Four phases
1. System Design

- *Devises high-level strategy* for solving problem
  - Set trade-off priorities
- *Construct system architecture by organizing into subsystems (system structuring)*
  - Choose an approach for persistent data management (repository model)
  - Allocate components to processors and tasks (distribution model)
- *Choose the implementation of control in software system (control modeling)*
  - Identify concurrency inherent in the problem
  - Define access to global resources
- *Divide problem into implementable components (modular decomposition)*

OMT: Four phases
2. Object Design

- *Full definition* of all the classes in the system
- *Implementation alternatives* evaluated and chosen
- Combine three models to obtain class operations
- Design algorithms to implement operations
- Optimize access paths to data
- Implement control for external interactions
- Adjust class structure to increase inheritance
- Design associations and setermine object representation
- Package classes and associations into implementable modules

OMT: Four phases
3. Detailed Design 1/2

- Detailed design is the process of completely specifying an architectural design such that module implementation can proceed (independently)
- Interface specifications
  - brief description of each module
  - attributes
    - brief description and specify types
  - operations
    - brief description
    - list of parameters and parameter types
    - return type (if applicable)

OMT: Four phases
3. Detailed Design 2/2

- Algorithm and data structure specification
  - the designer can give hints as to what algorithms or data structures might be most useful for a particular module
  - also, the client may have specified a particular algorithm or data structure that must be used
  - in addition, constraints in the requirements may require one approach over another
    - for instance, implementing a data structure so that it uses the minimum amount of memory possible vs. keeping everything in memory for speed

OMT: Four phases
4. Implementation 1/2

- Most programming languages provide very similar sets of features
  - user-defined types
  - control structures
    - if...then...else...
    - while x do y
    - for i = 1 to x
    - etc
  - etc.
- This means that, in general, operations can be expressed in many different languages

OMT: Four phases
4. Implementation 2/2

- Major differences between languages usually fall into these categories
  - compiled vs. interpreted
  - procedural vs. object-oriented
  - general purpose vs. application/domain specific
    - e.g. C++ vs. FileMaker Pro's scripting language
- If a design takes advantage of, or depends on, one or more of these features then certain programming languages have to be excluded from implementation


Q&
A