Data Flow Diagrams

Structured Analysis
- Models data elements
  - Attributes
  - Relationships
- Models processes that transform data

Modeling Tools
- Data object diagrams
- ERD diagrams
- Data flow diagram
- Process narrative

A data flow diagram describes information flow among a set of processes and actors.

A process narrative describes how a single process transforms input data to output data.

DFDs and Progressive Refinement

Each DFD reveals progressively more detail than the DFD that preceded it

Level 1 DFD:

Refinement continues until each bubble can be (easily) implemented as a program module
Process Narrative

A process specification describes all of the flow processes in the final (most detailed) DFD.

The process password transform performs password validation at the control panel for the SafeHome security function. Process password receives a four-digit password from the interact with user function. The password is first compared to the master password stored within the system …

A process specification can be represented as a collection of process narratives.

Some Guidelines

- Level 0 DFD should contain only a single bubble
- All arrows and bubbles should be meaningfully labeled
- Refinement begins by isolating next level processes, data objects, and data stores
- Refine only one bubble at a time
- Data flow continuity must be maintained between levels

Exercise: Data Flow Diagrams

Create a level 0 data flow diagram for a basic automated teller machine (ATM).

You can ignore administrative scenarios.

What would the level 1 data flow diagram look like?
A Different View

Object-Oriented Analysis
- Models analysis classes
  - Data
  - Processes
- Models class collaborations

Class diagrams

Modeling Tools
- Class diagrams
- Packages
- CRC cards
- Sequence diagrams

Object-oriented analysis results in an analysis model that describes a system of collaborating objects

Every object encapsulates a set of data elements, and exports a set of operations for working with those elements

Classes and Objects
- A class consists of a set of attributes and methods. A class diagram is used to show the static structure of a class

  ```
  class AnsweringMachine {
  greeting()
  messages()
  +setGreeting()
  +getMessage()
  +recordMessage()
  }
  ```

- Each class is like a template that defines how instances of the class – objects – should be created

Accessibility
- Attributes and Methods can be declared at three levels of accessibility
  - Public (+)
    - Visible everywhere
  - Private (-)
    - Visible only from inside the declaring class
  - Protected (#)
    - Visible only from within the declaring class and any of its subclasses
- We will all now swear never to declare public attributes

Inheritance
- Inheritance allows a subclass to inherit the attributes and methods of a superclass

  ```
  class AnsweringMachineWithTime extends AnsweringMachine {
  time()
  +setTime()
  }
  ```

- A subclass begins with everything provided by its superclass (or base class)
  - Can add additional attributes and methods
  - Can override existing public/protected methods
Inheritance Hierarchy

- An inheritance hierarchy (or **class hierarchy**) refers to all of the classes in a system, and the inheritance relations among them.
- Depending on the type of system being developed, the class hierarchy can be complex!

[http://java.sun.com/j2se/1.3/docs/api/](http://java.sun.com/j2se/1.3/docs/api/)


---

**Associations**

- Just like data objects, classes can be related to one another. These relationships are called **“associations”**

---

**Other Relationships**

- **Composition**
  - Defines a whole/part relationship
  - Multiplicity may be specified in either case

- **Aggregation**
  - Defines a containment relationship

---

**Identifying Analysis Classes**

- We begin with a grammatical parse
  - **Nouns**
    - Candidate classes
      - External entities, events, places, structures, things, etc.
    - Candidate attributes
      - Entities that naturally belong to a class
  - **Verbs**
    - Candidate methods
      - Operations that manipulate data
      - Operations that perform a computation
      - Operations that inquire about state
      - Operations that check for a control event

---
**Building the OOA Model**

- Use the initial set of analysis classes as the basis for building the OOA model
- Identify class responsibilities
  - What are its primary jobs?
- Identify class relationships and collaborations
  - What other classes does it need to interact with?
  - How will they interact?
- Identify class commonalities and variabilities
  - Which attributes/methods do they have in common?
  - Which attributes/methods vary?

This will be an iterative process!

**Distribution, Cohesion, and Encapsulation**

- Some additional model desiderata
  - Responsibility distribution
    - Intelligently should be distributed across classes
    - Avoid monolithic classes
  - Class cohesion
    - Classes should be single minded
    - Avoid classes with unrelated members
  - Encapsulation
    - The class with the data should do the work
    - Avoid passing state to other classes

**Some Rules of Thumb**

- Focus on the problem
  - Identify “what” classes/attributes
  - Ignore “how” classes/attributes
- Focus on core classes
  - Eliminate procedural classes
  - Avoid data classes
  - Avoid classes with only a single attribute
- Use (don’t overuse) inheritance intelligently
  - Move commonalities up the hierarchy
  - Move variabilities down the hierarchy
  - Avoid deep hierarchies

**Categorization**

- **Categorization** is one of the most important tools in our intellectual toolbox
- How do we manage (intellectually) a large number of related items?
  - We group the items into categories of items that share common characteristics or satisfy common goals
- In object-oriented analysis, we categorize classes using *analysis packages*
Analysis Packages

- An *analysis package* is a collection of related analysis classes grouped under a representative name.

Java, C#, and VB .NET use a similar notion for categorizing implementation classes.