

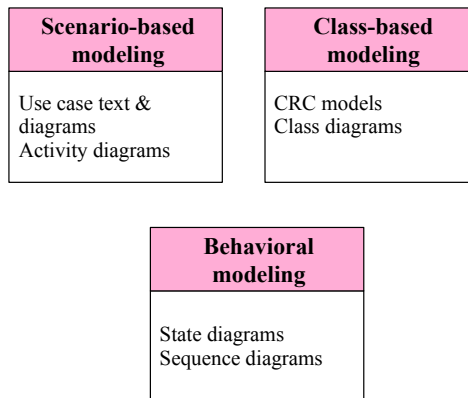
Object-Oriented Analysis and Modeling

Object-oriented analysis (OOA)

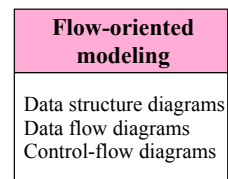
- What are the **relevant objects**? How do they **relate** to one another?
- How do we specify/model a problem so that we can create an **effective design**?
- OOA aims to model the problem domain, by developing an object-oriented (OO) system.

Elements of the Analysis Model

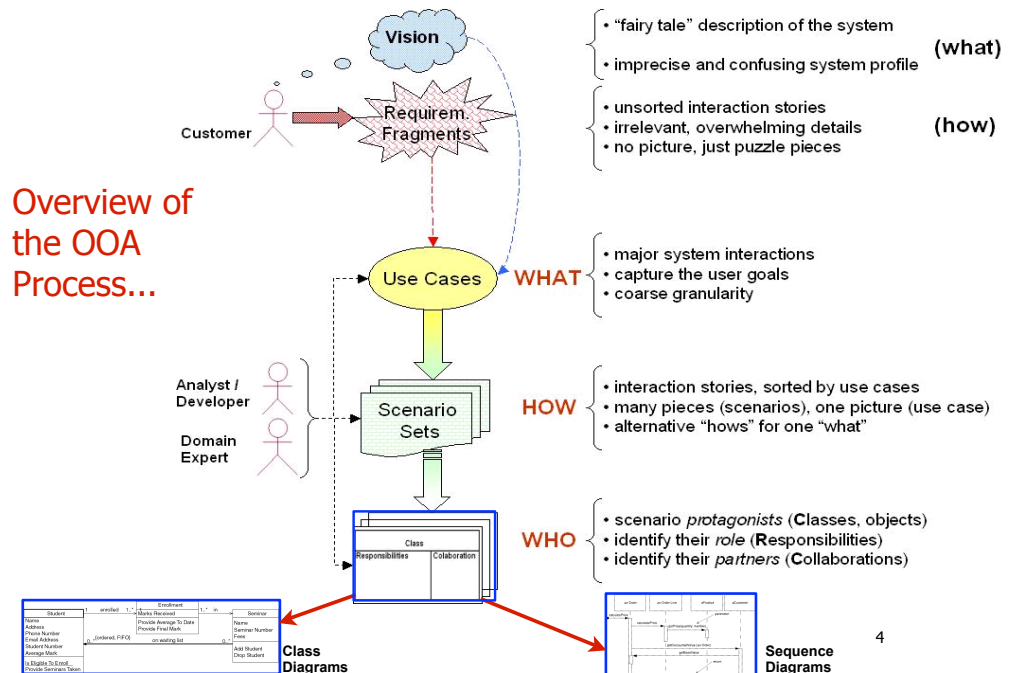
Object-oriented Analysis



Structured Analysis



Overview of the OOA Process...



Class-Responsibility-Collaborator (CRC) Modeling

- Technique to identify candidate classes and indicate their responsibilities and collaborators
 - ▶ K.Beck&W.Cunningham (1989), R.Wirfs-Brock(1990,2002)
- Uses simple index cards

[illegible]

CRC Cards Session Scenario

- Use-Case driven
 - ▶ a use-case is the token
- Goal: be able to go through the whole use-case description by using the responsibilities written on the CRC cards
 - ▶ ...and of course following the Collaborator links

Rules for Identifying Classes

1. **Retained Information**
 - information about object must be remembered for the system to function
2. **Needed Services**
 - have a set of operations that change the value of its attributes
3. **Multiple Attributes**
 - focus on “major” information
 - object with single attribute is ok during design, but during analysis is just an attribute of another object
4. **Essential Requirements objects**
 - entities that produce or consume information of the system, in any solution

Identifying Responsibilities

[illegible]

- attributes and operations of an identified class
- **Guidelines of Wirfs-Brock:**
 1. *System intelligence should be evenly distributed*
 2. *Information about one thing should be localized within a single class*
 3. *Information and its related behavior should stay in the same class*
 4. *Responsibilities should be shared among related classes*

Operations = Verbs

- **Computation**
- **Manipulation of data**
 - e.g., add, delete, modify attributes
- **Query**
 - about the state of an object
- **Monitor** an object
 - for the occurrence of a controlling event
- Has knowledge about the **state** of class and its **associations**

Collaborations

Class name:	
Class type: (e.g., device, property, role, event)	
Class characteristic: (e.g., tangible, atomic, concurrent)	
responsibilities:	collaborations:

- Class can fulfill responsibilities by:
 1. using its own operations to manipulate its own attributes
 2. collaborating with others
- Three types of generic relationships:
 3. *has-knowledge-of* (association)
 4. *is-part-of* (aggregation)
 5. *composition*

Heuristics for Object-Oriented Modeling

Problema Proliferarii Claselor [Riel96]

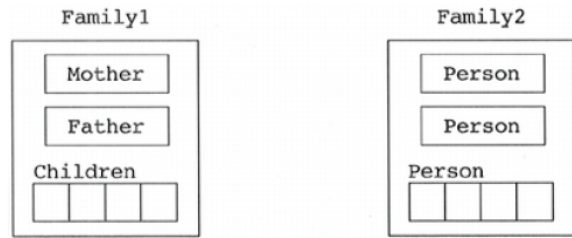
- Spaghetti Code vs. Ravioli Code
- Cum se manifesta "Codul Ravioli"
 - Vreau sa adaug o facilitate noua in sistem. Care 23 de clase din cele 4.200 de clase trebuie sa le modific?

Fiti retinuti in a modela ca si clase
entitati din afara sistemului de implementat!

- Exemplu: Clientul unui Bancomat
 - trimite un mesaj bancomatului

Clase sunt acelea care PRIMESC MESAJE (sunt apelate)
nu cele care TRANSMIT MESAJE (apeleaza)!

Roluri vs. Clase

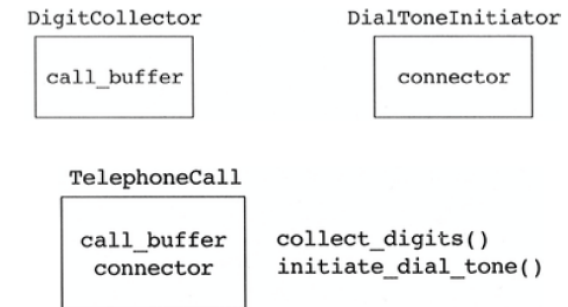


Comportamentul este cel care decide!

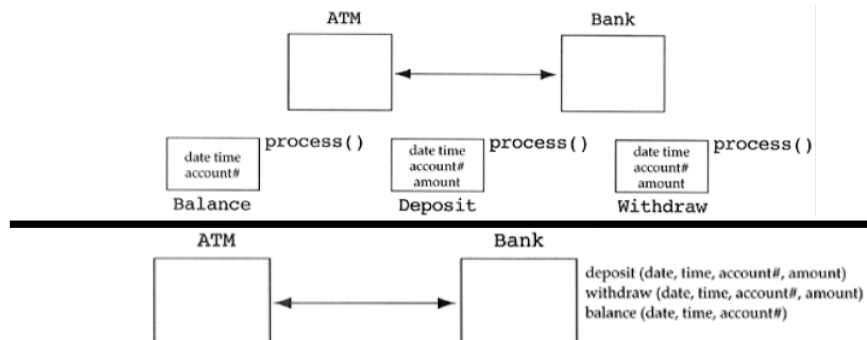
- ▶ daca comportamentul difera: **avem clase**;
- ▶ daca nu: doar roluri ale aceleiasi clase
- ▶ depinde de domeniul modelat de aplicatie
 - ◆ *naste()* vs. *schimbaScutece()*

Euristici pentru Eliminarea Claselor Inutile

Nu transformati operatiile in clase



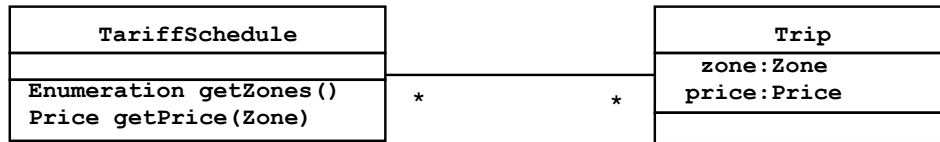
Cand Poate Fi Operatia o Clasa?



- cand cerintele sunt de asa natura incat operatiile reprezinta un "atom" adica "obiectiveaza" o anumita entitate intr-un context dat
 - ▶ ex. tiparirea tranzactiilor --> **operatii persistente**
 - ▶ tiparul Command

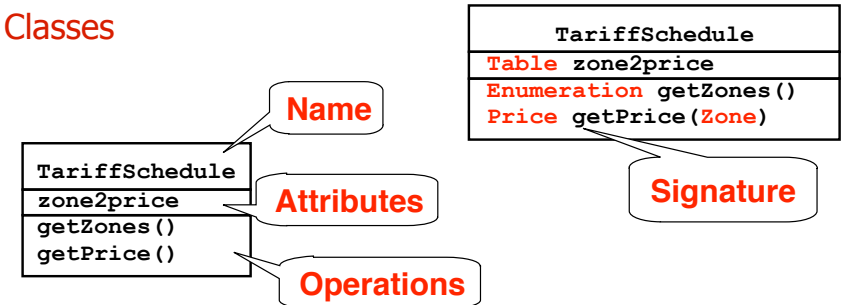
Class Diagrams

Class Diagrams



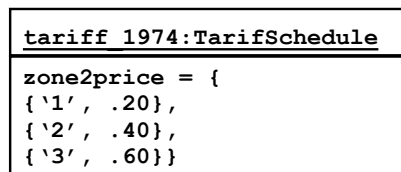
- Class diagrams represent the structure of the system.
- Class diagrams are used
 - during requirements analysis to model problem domain concepts
 - during system design to model subsystems and interfaces
 - during object design to model classes.

Classes



- A class represents a concept.
- contains state (attributes) and behavior (operations).
- Each attribute has a type.
- Each operation has a signature.
- The class name is the only mandatory information.

Instances



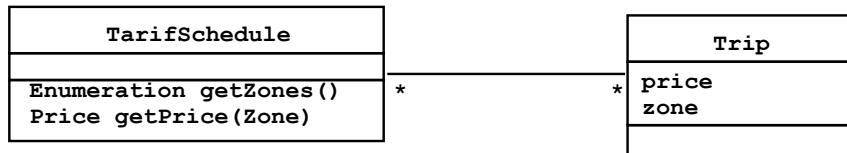
- An **instance** represents a phenomenon.
- The name of an instance is underlined and can contain the class of the instance.
- The attributes are represented with their **values**.

Instance vs. Class



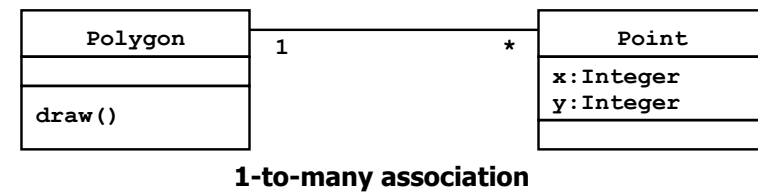
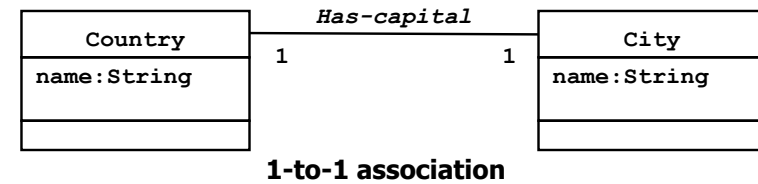
Rene Magritte, Treachery of Images - 1929

Associations



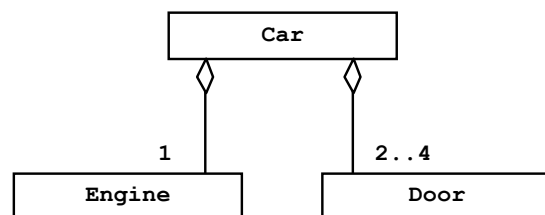
- Associations denote relationships between classes.
- The multiplicity of an association end denotes how many **objects** the source object can legitimately reference.

1-to-1 and 1-to-Many Associations



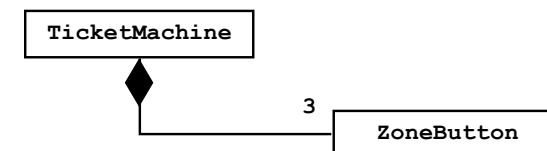
Aggregation

- An **aggregation** is a special case of association denoting a “consists of” (HAS-A) hierarchy.
- The **aggregate** is the parent class, the **components** are the children class.

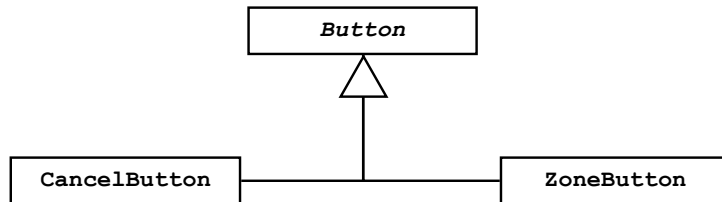


Composition

- A solid diamond denotes **composition**, a strong form of aggregation where components cannot exist without the aggregate.



Generalization



- Generalization relationships denote inheritance between classes.
- The children classes inherit the attributes and operations of the parent class.
- Generalization simplifies the model by **eliminating redundancy**.

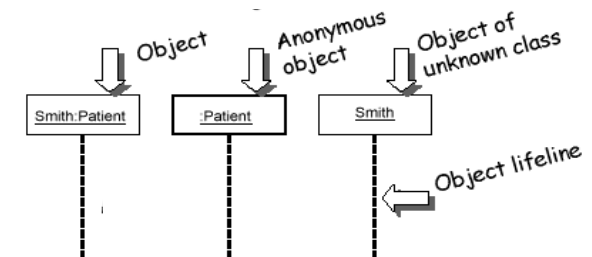
Sequence Diagrams

UML Sequence Diagrams

- Sequence Diagram**: an "interaction diagram" that models a single scenario executing in the system
 - perhaps 2nd most used UML diagram (behind class diagram)
- Participant**: an object or entity that acts in the sequence diagram
 - sequence diagram starts with an unattached "found message" arrow
- Message**: communication between participant objects
- Axes** in a sequence diagram:
 - horizontal: which object/participant is acting
 - vertical: time (down -> forward in time)

Representing Objects

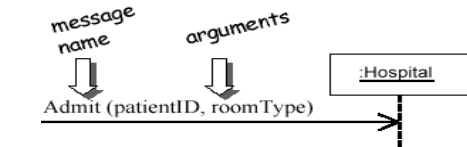
- squares with object type, optionally preceded by object name and colon
 - write object's name if it clarifies the diagram
 - object's "life line" represented by dashed vert. line



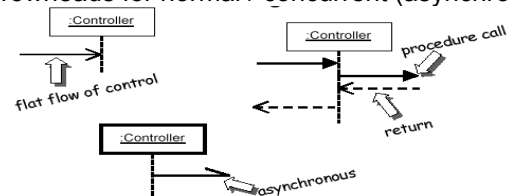
Name syntax: <objectname>:<classname>

Messages between objects

- message (method call) indicated by horizontal arrow to other object
 - write message name and arguments above arrow



- dashed arrow back indicates return
- different arrowheads for normal / concurrent (asynchronous) methods

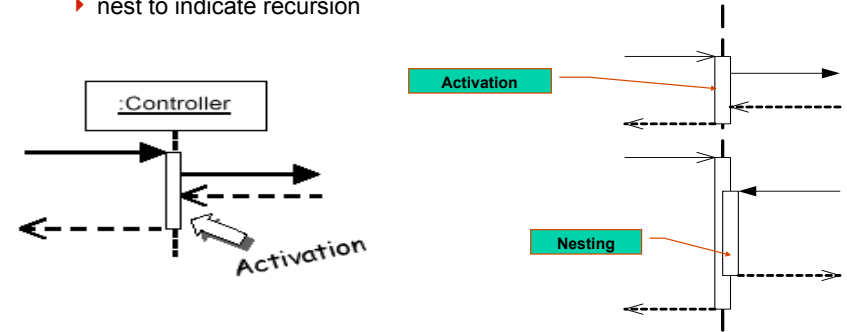


Dr. Radu Marinescu

29

Indicating method calls

- Activation**: shows when object's method is on the stack
 - either that object is running its code, or it is on the stack waiting for another object's method to finish
 - nest to indicate recursion



Dr. Radu Marinescu

30

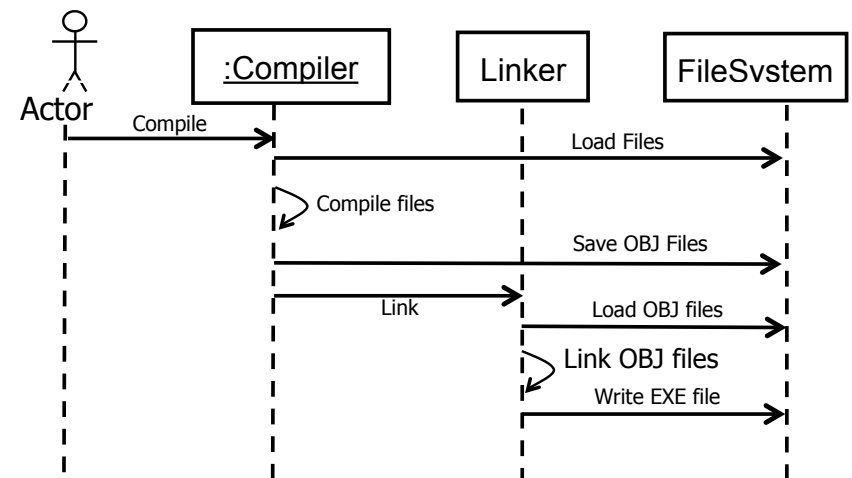
Example 1:

- Building an executable from sources
 - load source files and compile them
 - load resulting object files and link them
 - write executable file

Dr. Radu Marinescu

31

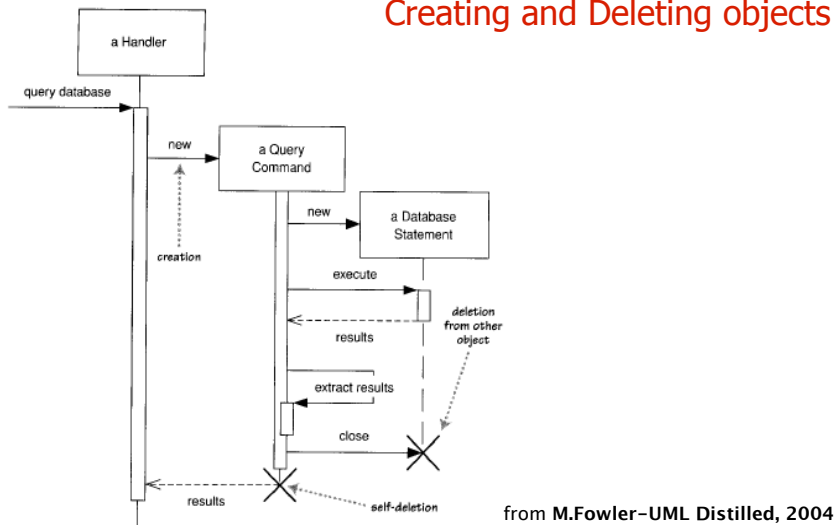
Sequence Diagram – Compilation



Dr. Radu Marinescu

32

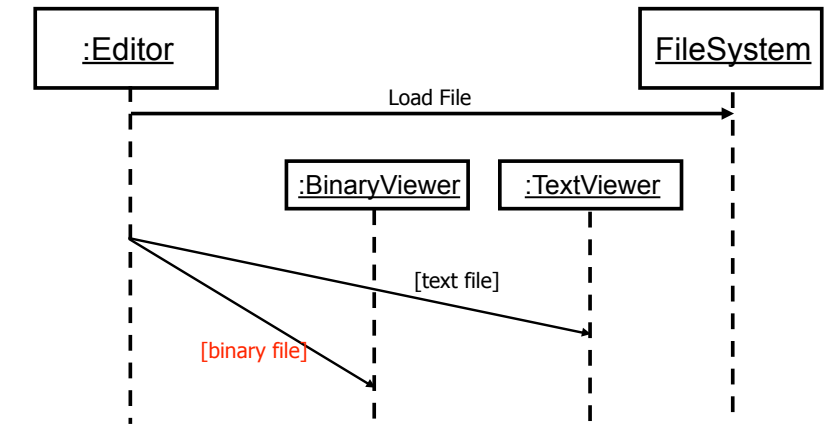
Creating and Deleting objects



Dr. Radu Marinescu

33

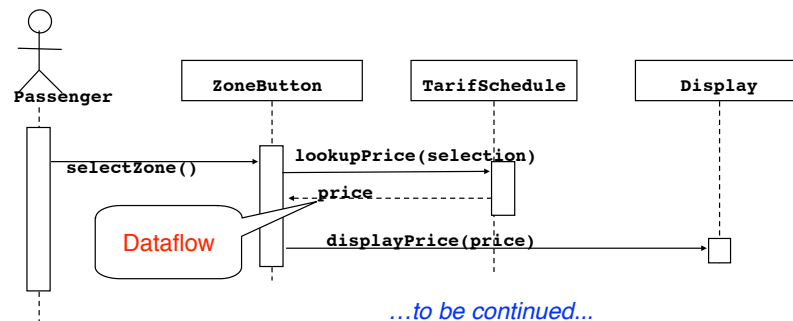
Branching Flow: flow goes to different objects
[if condition is met]



Dr. Radu Marinescu

34

Flow of messages



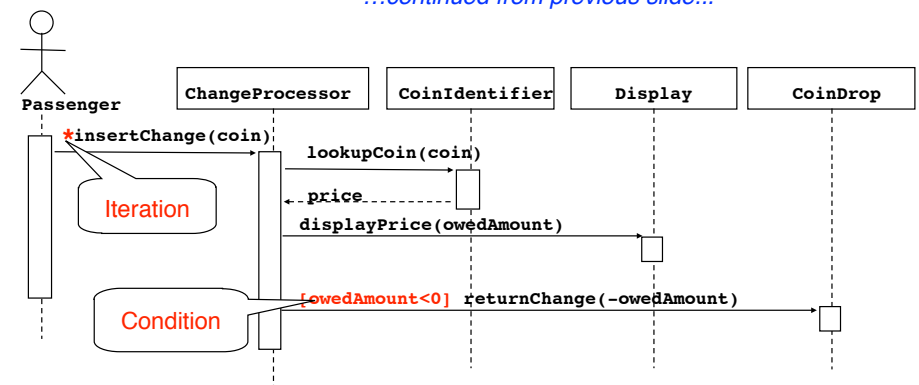
- The source of an arrow indicates the activation which sent the message
- An activation is as long as all nested activations
- Horizontal dashed arrows indicate data flow
- Vertical dashed lines indicate lifelines

Dr. Radu Marinescu

35

Iteration & condition

...continued from previous slide...



- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [] before the message name

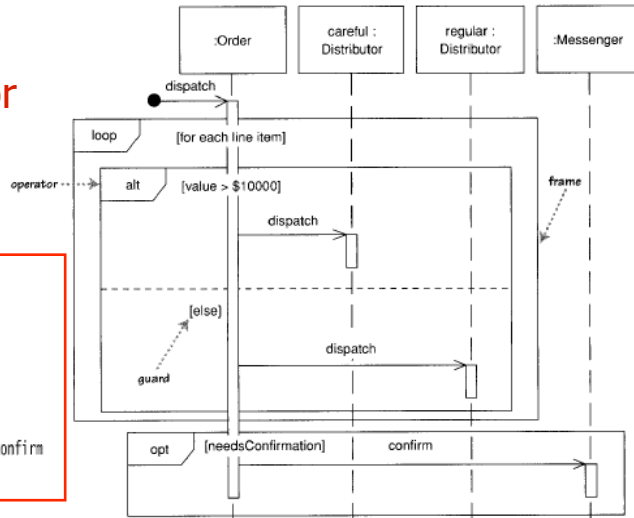
Dr. Radu Marinescu

36

Alternative Notation for Condition and Loops

```

procedure dispatch
  foreach (lineitem)
    if (product.value > $10K)
      careful.dispatch
    else
      regular.dispatch
    end if
  end for
  if (needsConfirmation) messenger.confirm
end procedure
  
```



from M.Fowler-UML Distilled, 2004

Sequence Diagram Summary

- UML sequence diagram represent behavior in terms of interactions.
- Useful to find missing objects.
- Time consuming to build but worth the investment.
- Complement the class diagrams (which represent structure).