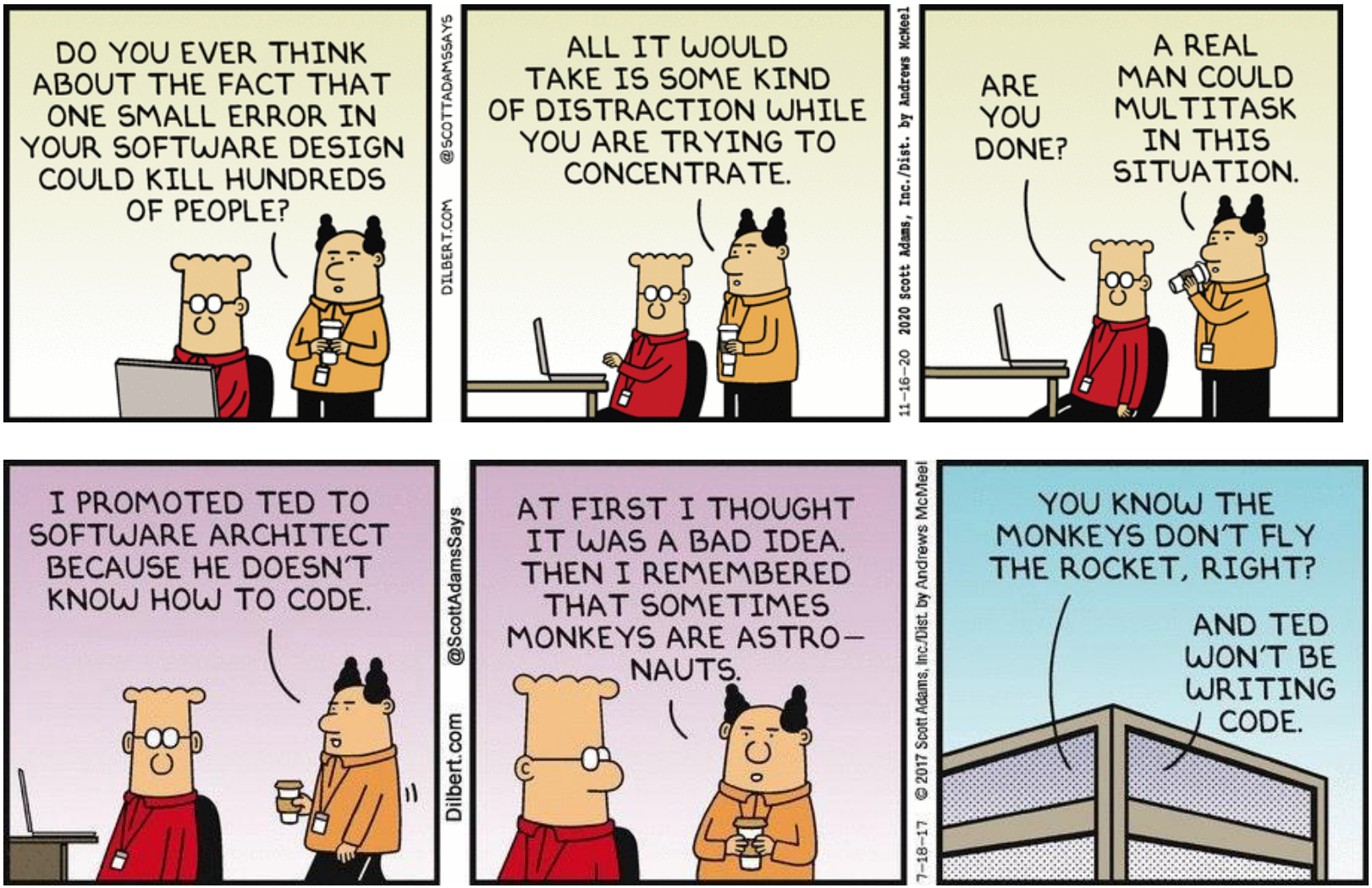
HOW TO MODEL SOFTWARE ARCHITECTURE AND DESIGN



DESIGN CONCEPTS/CHARACTERISTICS

The design should be based on the **requirements specification.**

The design should be **documented** (so that it supports implementation, verification, and maintenance.)

The design should use **abstraction** (to reduce complexity and to hide unnecessary detail.)

The design should be **modular** (to support abstraction, verification, maintenance, and division of labor.)

The design should be assessed for **quality** as it is being created, not after the fact.

Design should produce modules that exhibit independent functional characteristics.

Design should support **verification** and maintenance.

PRINCIPLES OF GOOD DESIGN

Divide and Conquer

Increase cohesion (keep related stuff together)

Decrease coupling (minimize dependencies between modules)

Increase the level of abstraction wherever possible

- When two modules interact, create abstract interfaces so that modules don't have to know specific low-level details about other modules

Design for Quality of service (Testability, Flexibility, Modifiability, etc.)

Design by Contract

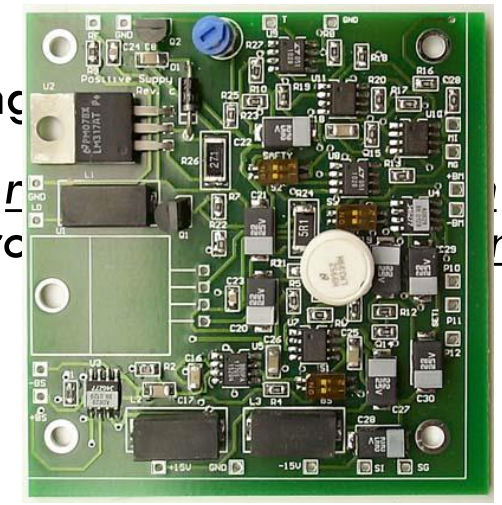
SOFTWARE MODELING USING UML

A RATIONAL DESIGN PROCESS: HOW AND WHY TO FAKE IT : DAVID PARNAS

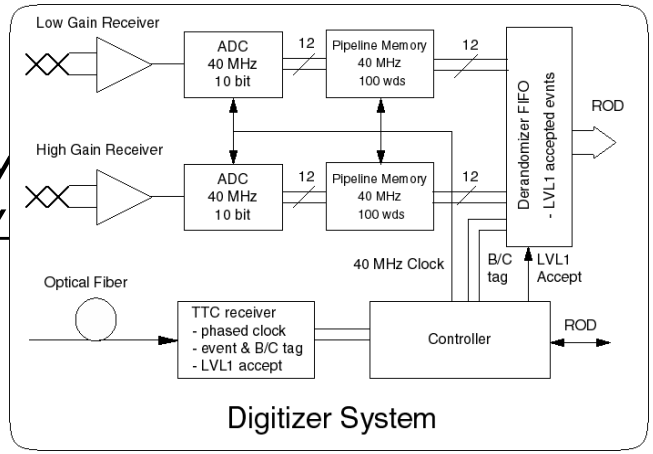
ENGINEERING MODELS

Eng

A r of some sy
pro m v



Modeled system



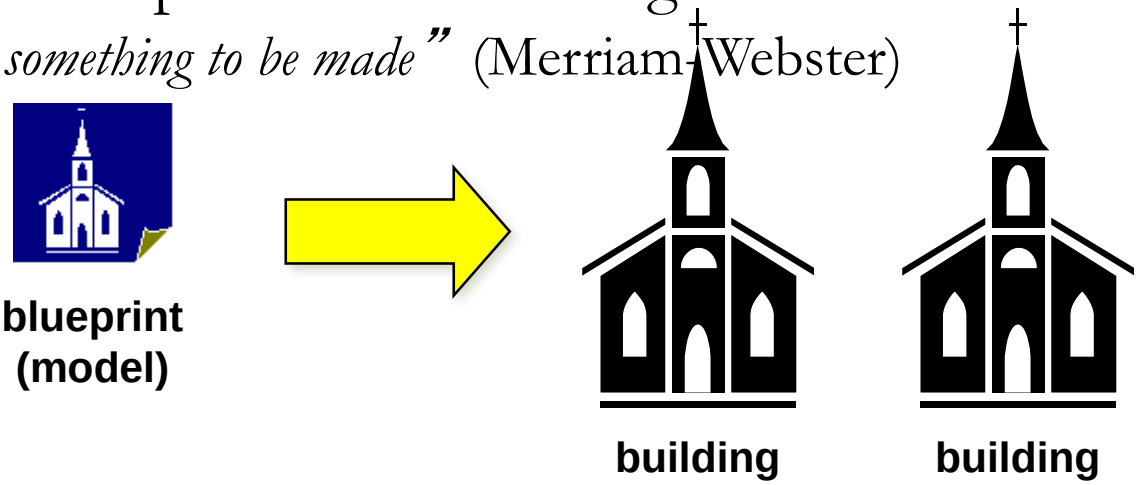
Functional Model

- We don't see everything at once
- We use a representation (notation) that is easily understood for the purpose on hand

MODELS

A *model* is a description of something

- *"a pattern for something to be made"* (Merriam-Webster)



- model ≠ thing that is modeled
  - The Map is Not The Territory

MODELS?

Why model?

What to model?

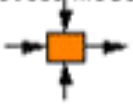
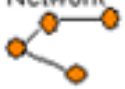
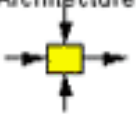
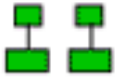
How do we model?

Modeling Maturity Level

- Level 0: No specification
- Level 1: Textual
- Level 2: Text with Diagrams
- Level 3: Models with Text
- Level 4: Precise Models

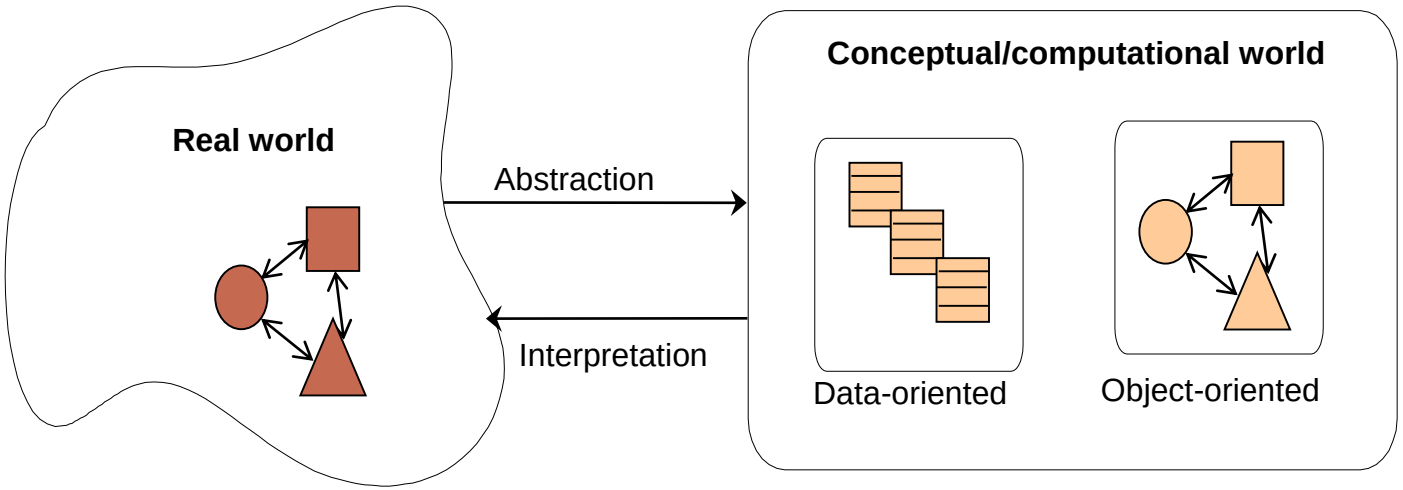
Zachman Framwork

Identification, Definition, Representation Specification, Configuration and Instantiation

abstractions / perspectives	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*
SCOPE *Planner* contextual	List of Things - *Important to the Business* 	List of Processes - *the Business Performs* 	List of Locations - *in which the Business Operates* 	List of Organizations - *Important to the Busine* 	List of Events - *Significant to the Business* 	List of Business Goals and Stra 
ENTERPRISE MODEL *Owner* conceptual	e.g., Semantic Model 	e.g., Business Process Model 	e.g., Logistics Network 	e.g., Work Flow Model 	e.g., Master Schedule 	e.g., Business Plan 
SYSTEM MODEL *Designer* logical	e.g., Logical Data Model 	e.g., Application Architecture 	e.g., Distributed System Architecture 	e.g., Human Interface Architecture 	e.g., Processing Structure 	e.g., Business Rule Model 
TECHNOLOGY CONSTRAINED MODEL *Builder* physical	e.g., Physical Data Model 	e.g., System Design 	e.g., Technical Architecture 	e.g., Presentation Architecture 	e.g., Control Structure 	e.g., Rule Design 
DETAILED REPRESEN-TATIONS *Subcontractor* out-of-context	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 	e.g. Rule Specification 
FUNCTIONING ENTERPRISE	DATA Implementation	FUNCTION Implementation	NETWORK Implementation	ORGANIZATION Implementation	SCHEDULE Implementation	STRATEGY Implementation

OBJECT-ORIENTED MODELING

Models a system as a set of objects that interact with each others | No semantic gap (or impedance mismatch)



KEY IDEAS OF O-O MODELING

Abstraction

Encapsulation

Relationship

- Association: relationship between objects
- Inheritance: mechanism to represent similarity among objects

Object-oriented

= object (class) + inheritance + message send

OBJECTS VS. CLASSES

	Interpretation in the Real World	Representation in the Model
Object	An *object* represents anything in the real world that can be distinctly identified.	An *object* has an identity, a state, and a behavior.
Class	A *class* represents a set of objects with similar characteristics and behavior. These objects are called *instances* of the class.	A *class* characterizes the structure of states and behaviors that are shared by all of its instances.

UNIFIED MODELING LANGUAGE (UML)

Notation for object-oriented modeling

Standardized by Object Management Group (OMG)

Consists of 12+ different diagrams

- Use case diagram
- Class diagram
- Statechart diagram
- Sequence diagram
- Communication diagram
- Component diagram
- …

WHY UML?

1. Sketch
2. Blueprint
3. Formal Modeling

UML DIAGRAMS (NUTSHELL)

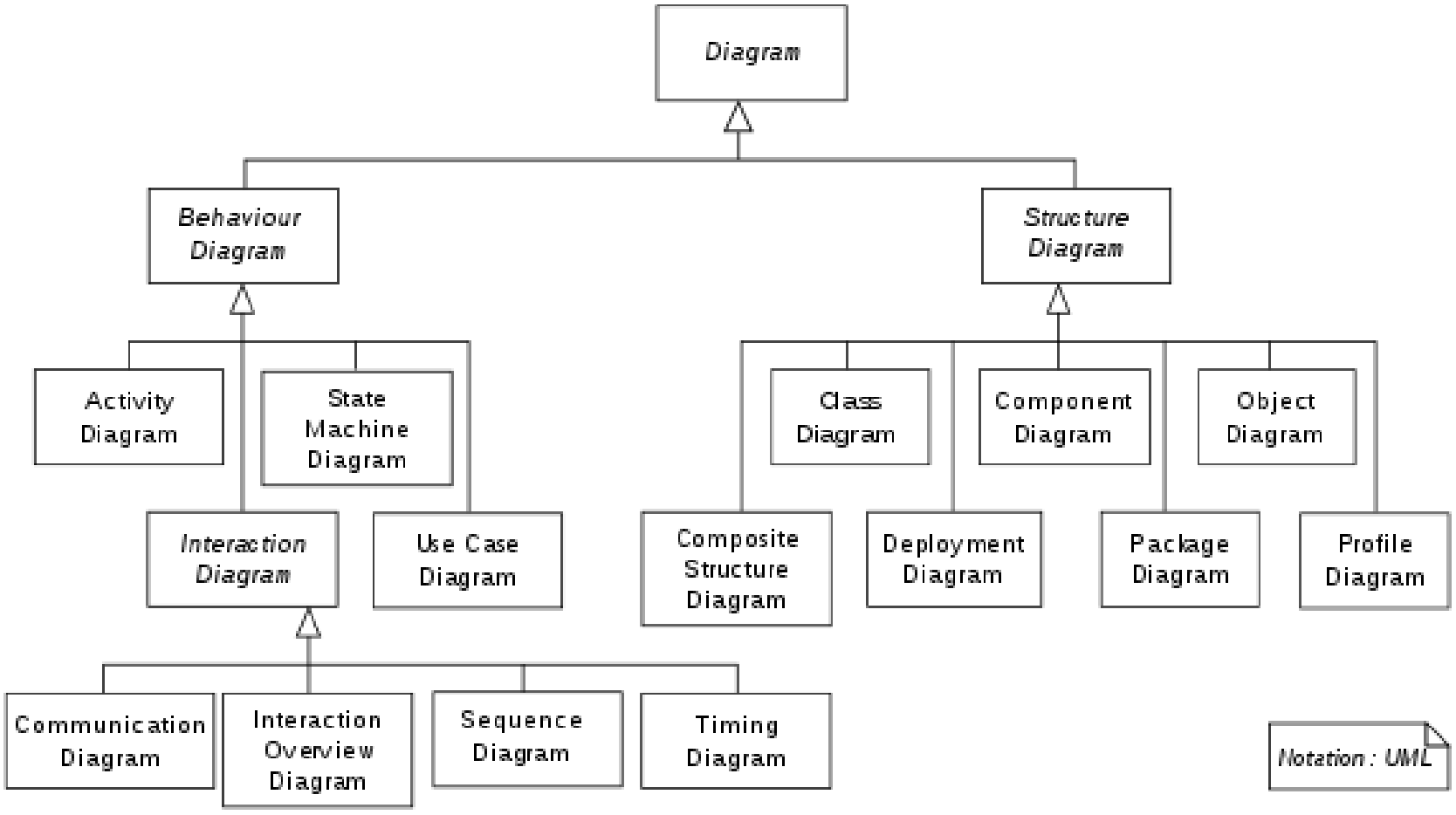


Diagram	Book Chapters	Purpose	Lineage
Activity	11	Procedural and parallel behavior	In UML 1
Class	3, 5	Class, features, and relationships	In UML 1
Communication	12	Interaction between objects; emphasis on links	UML 1 collaboration diagram
Component	14	Structure and connections of components	In UML 1
Composite structure	13	Runtime decomposition of a class	New to UML 2
Deployment	8	Deployment of artifacts to nodes	In UML 1
Interaction overview	16	Mix of sequence and activity diagram	New to UML 2
Object	6	Example configurations of instances	Unofficially in UML 1
Package	7	Compile-time hierarchic structure	Unofficially in UML 1
Sequence	4	Interaction between objects; emphasis on sequence	In UML 1
State machine	10	How events change an object over its life	In UML 1
Timing	17	Interaction between objects; emphasis on timing	New to UML 2
Use case	9	How users interact with a system	In UML 1

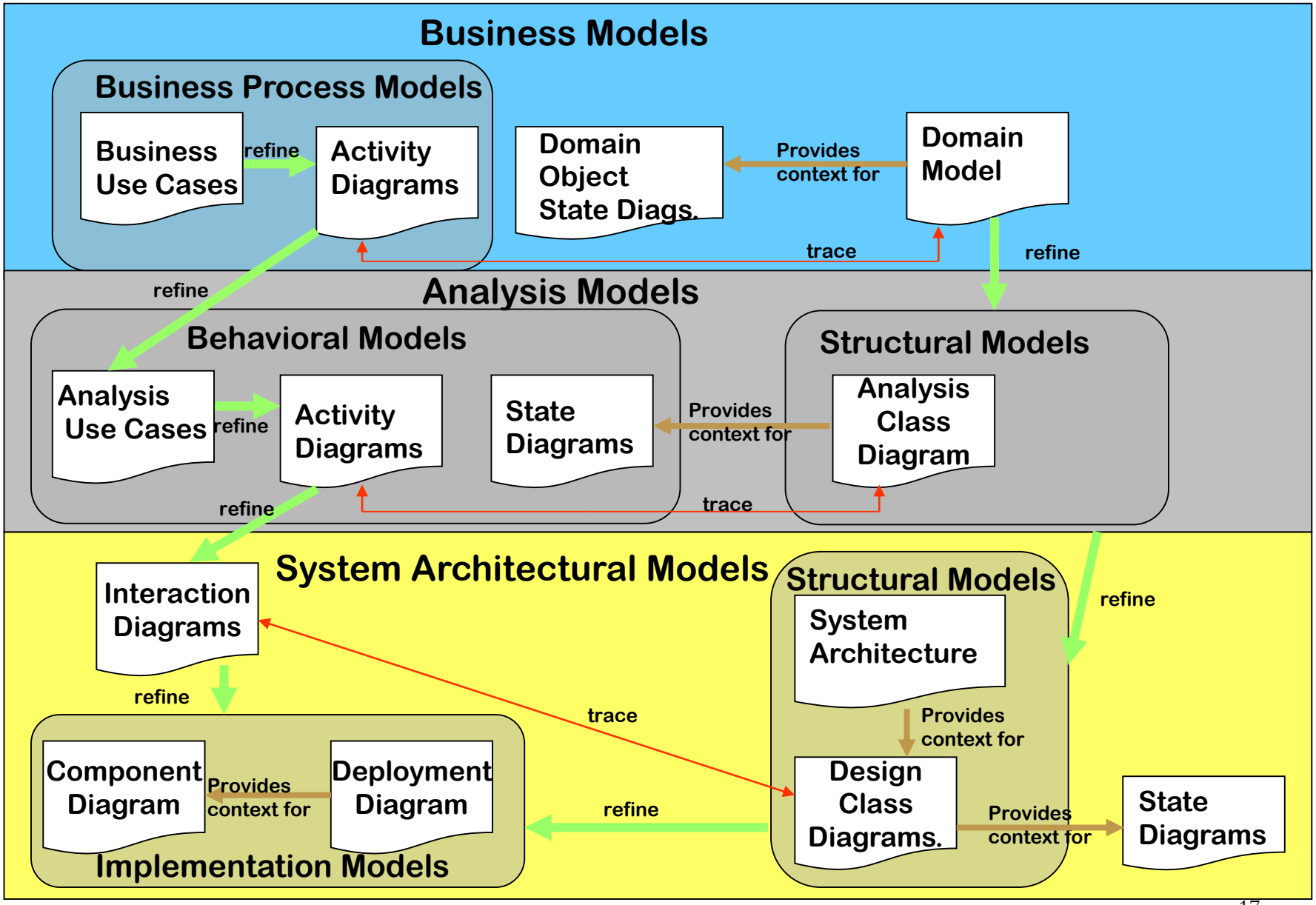
WHAT THE UML IS NOT

Not an OO method or process

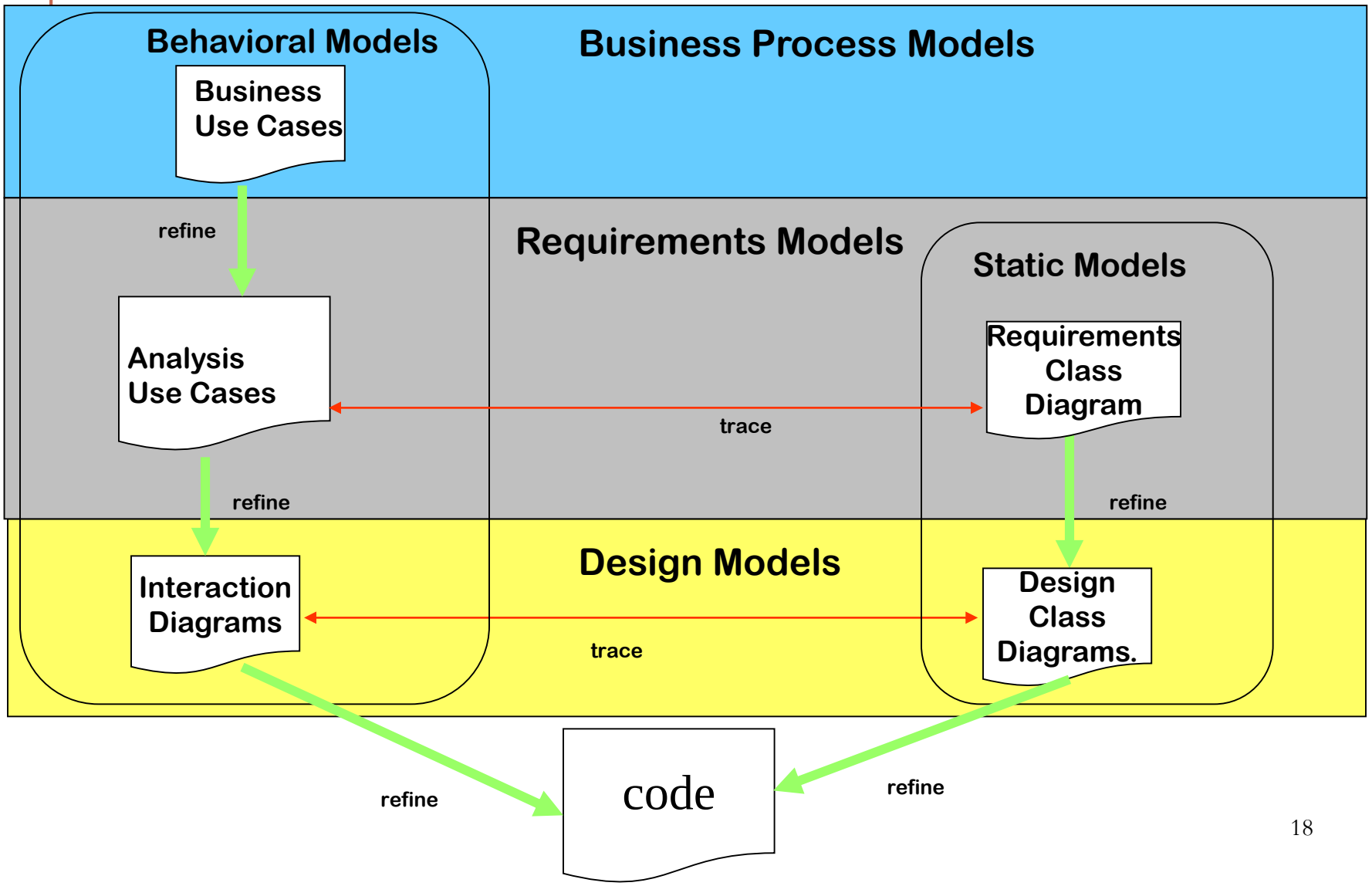
Not a visual programming language

Not a tool specification

A "Full" Process (using UML diagrams)



AN "ULTRALITE" PROCESS

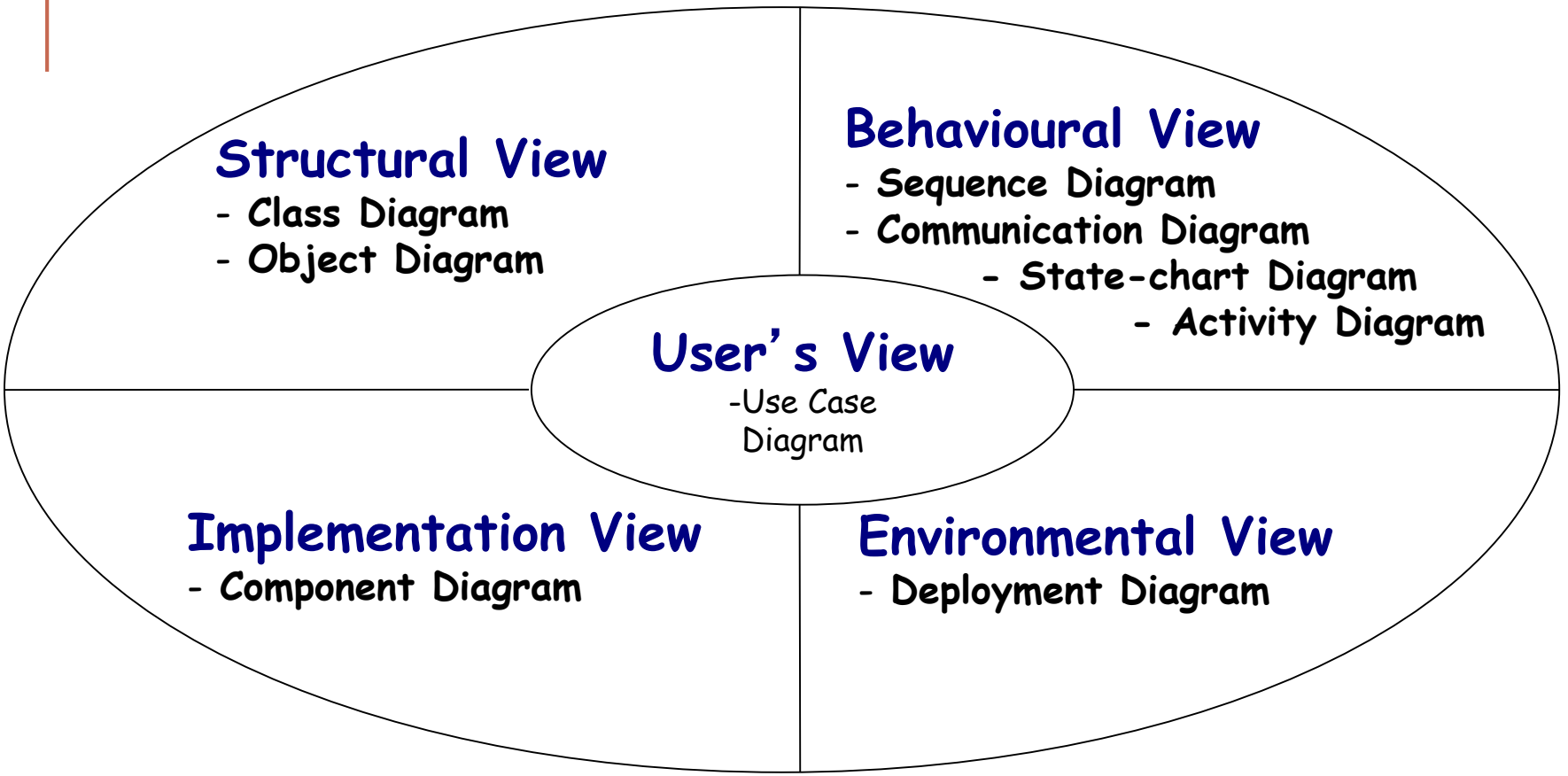


UML MODEL VIEWS

Views of a system:

- User's view
- Structural view
- Behavioral view
- Implementation view
- Environmental view

UML DIAGRAMS



Diagrams and views in UML

ARE ALL VIEWS REQUIRED FOR SPECIFYING A TYPICAL SYSTEM?

- **NO**
  - Use case diagram, class diagram and one of the interaction diagram for a simple system
  - State chart diagram required to be developed when a class state changes
  - However, when states are only one or two, state chart model becomes trivial
  - Deployment diagram in case of large number of hardware components used to develop the system

STATIC VS. DYNAMIC MODELS

Static model

- Describes static structure of a system
- Consists of a set of objects (classes) and their relationships
- Represented as class diagrams

Dynamic model

- Describes dynamic behavior of a system, such as state transitions and interactions (message sends)
- Represented as statechart diagram, sequence diagrams, and collaboration diagrams

UML CLASS DIAGRAM

Most common diagram in OO modeling

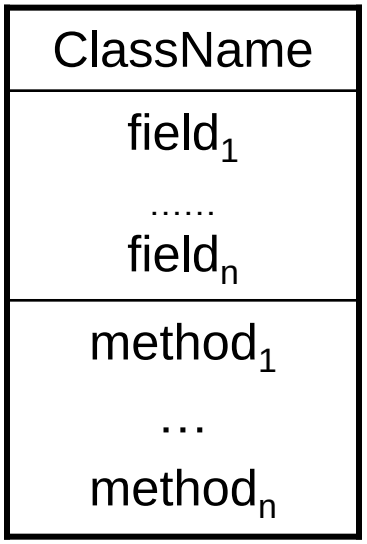
Describes the static structure of a system

Consist of:

- Nodes representing classes
- Links representing of relationships among classes
  - Inheritance
  - Association, including aggregation and composition
  - Dependency

NOTATION FOR CLASSES

The UML notation for classes is a rectangular box with as many as three compartments.

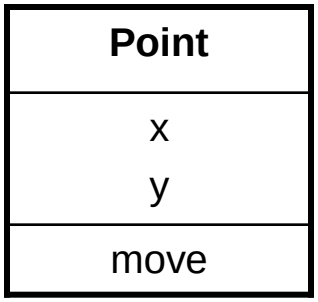


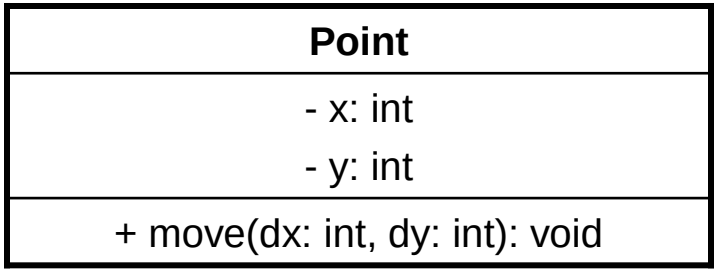
The top compartment show the class name.

The middle compartment contains the declarations of the fields, or *attributes,* of the class.

The bottom compartment contains the declarations of the methods of the class.

EXAMPLE





FIELD AND METHOD DECLARATIONS IN UML

Visibility	Notation
public	+
protected	#
package	~
private	-

Field declarations

- birthday: Date
- +duration: int = 100
- -students[1..MAX_SIZE]: Student

Method declarations

- +move(dx: int, dy: int): void
- +getSize(): int

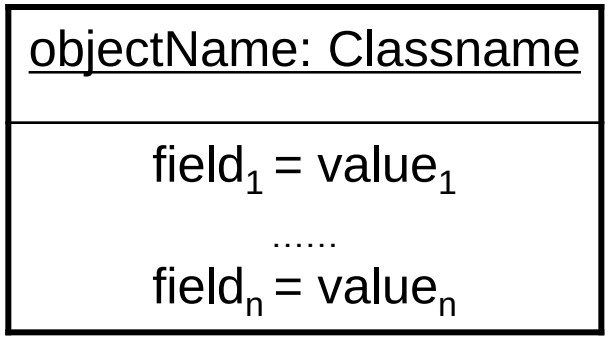
EXERCISE

Draw a UML class diagram for the following Java code.

```
class Person {
  private String name;
  private Date birthday;
  public String getName() {
    // …
  }
  public Date getBirthday() {
    // …
  }
}
```

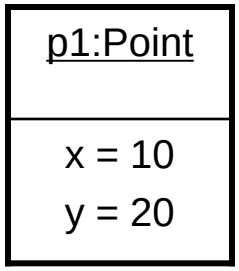
NOTATION FOR OBJECTS

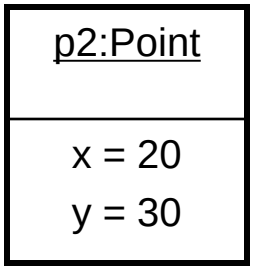
Rectangular box with one or two compartments



The top compartment shows the name of the object and its class.

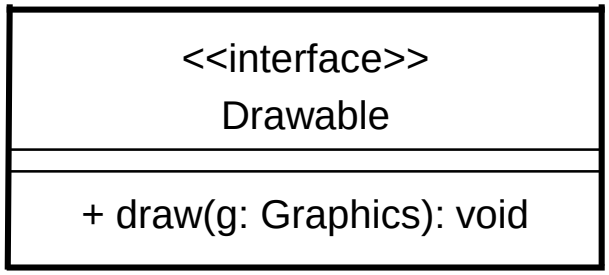
The bottom compartment contains a list of the fields and their values.

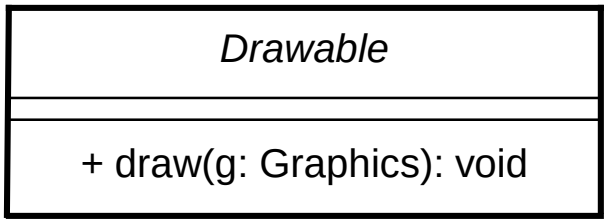




UML NOTATION FOR INTERFACES

```
interface Drawable {
   void draw(Graphics g);
}
```





INHERITANCE IN JAVA

Important relationship in OO modeling

Defines a relationship among classes and interfaces.

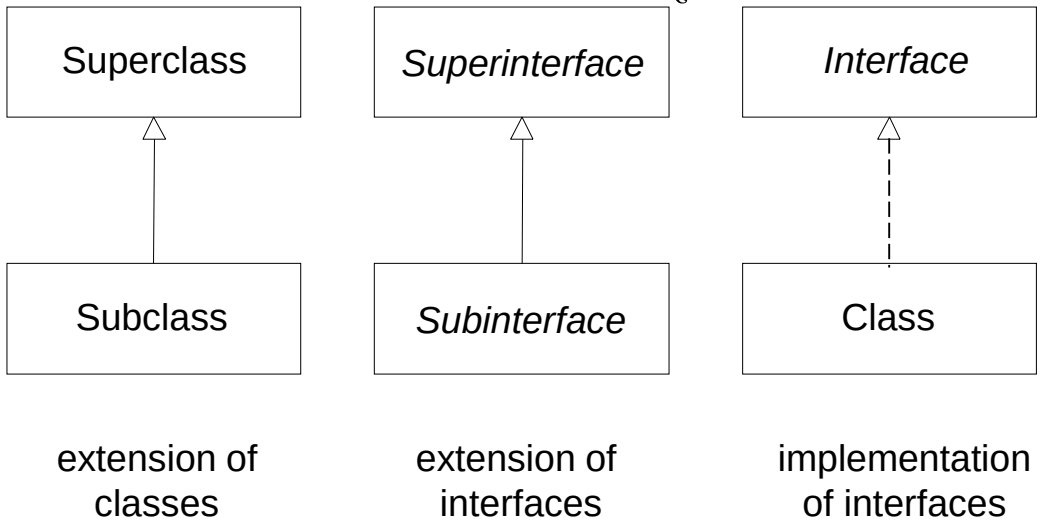
Three kinds of inheritances

- *extension* relation between two classes (*subclasses* and *superclasses*)
- *extension* relation between two interfaces (*subinterfaces* and *superinterfaces*)
- *implementation* relation between a class and an interface

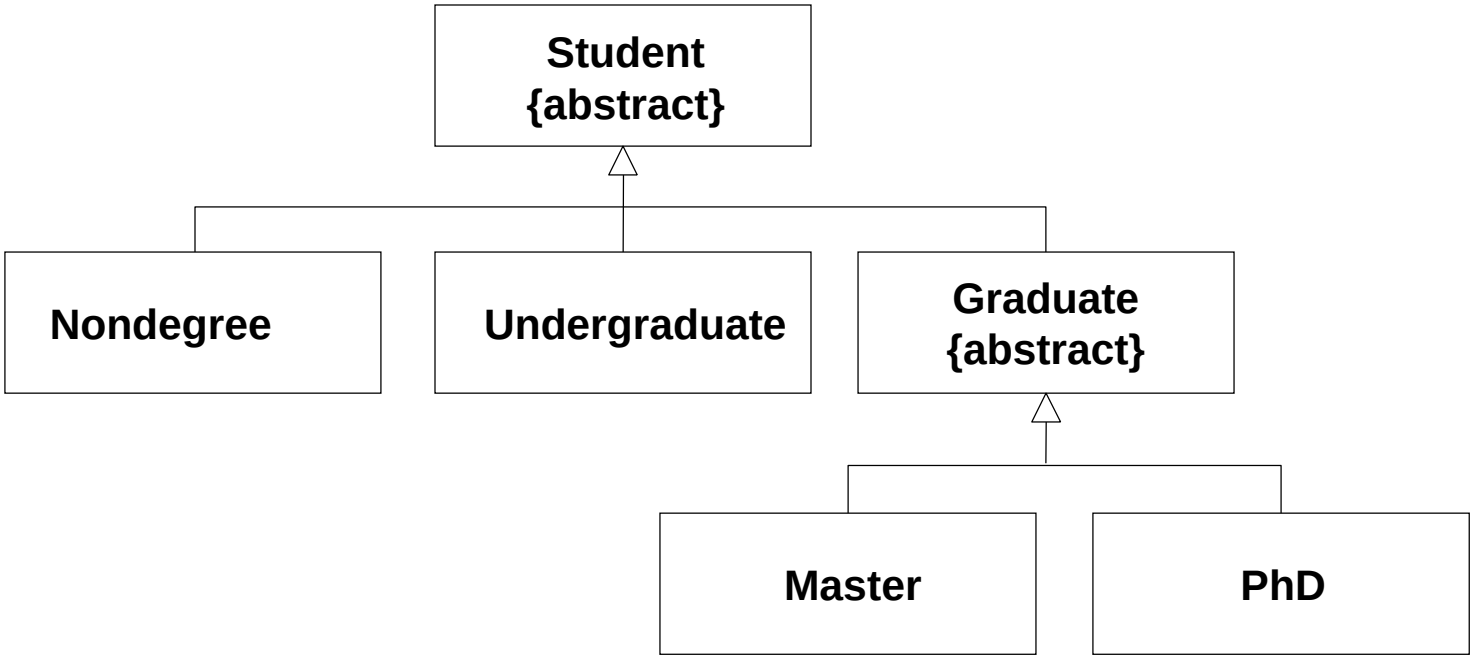
INHERITANCE IN UML

An extension relation is called *specialization* and *generalization*.

An implementation relation is called *realization*.



EXAMPLE


Student
{abstract}
Nondegree
Undergraduate
Graduate
{abstract}
Master
PhD

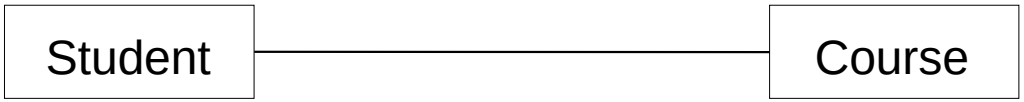
EXERCISE

Draw a UML class diagram showing possible inheritance relationships among classes Person, Employee, and Manager.

ASSOCIATION

General binary relationships between classes

Commonly represented as direct or indirect references between classes



ASSOCIATION (CONT.)

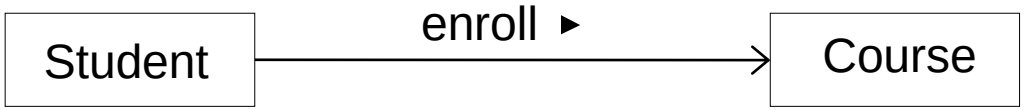
May have an optional label consisting of a name and a direction drawn as a solid arrowhead with no tail.

The direction arrow indicates the direction of association with respect to the name.

ASSOCIATION (CONT.)

An arrow may be attached to the end of path to indicate that *navigation* is supported in that direction

What if omitted?

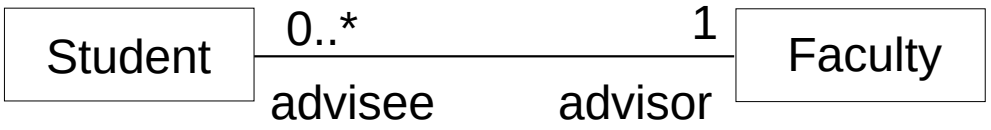


ASSOCIATION (CONT.)

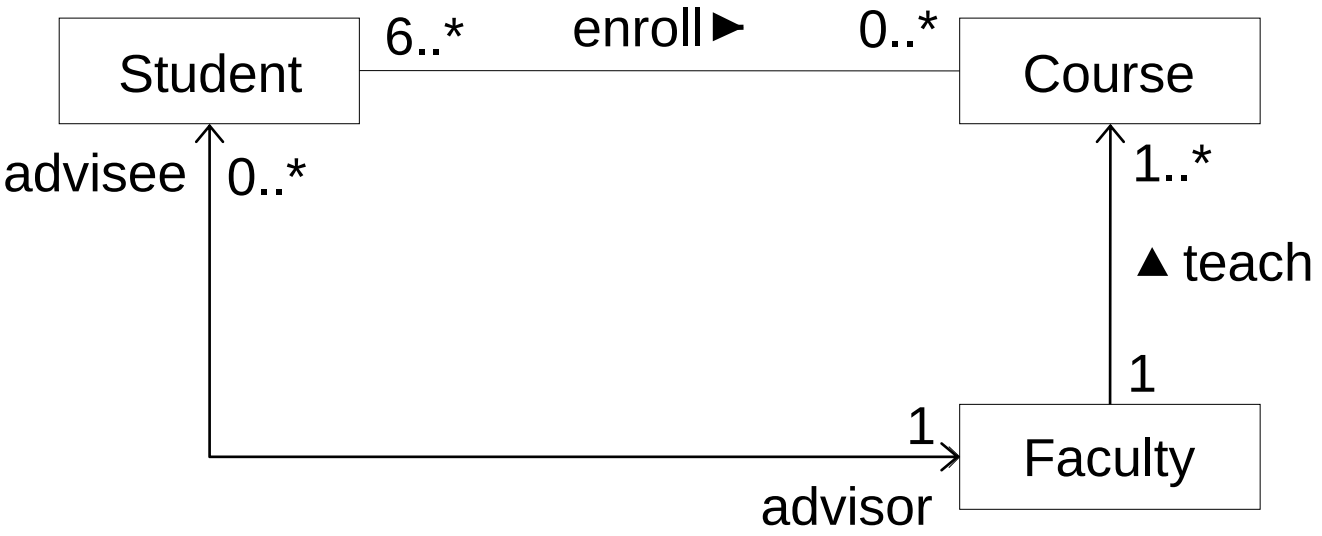
May have an optional *role name* and an optional *multiplicity specification*.

The multiplicity specifies an integer interval, e.g.,

- *l..u* closed (inclusive) range of integers
- i singleton range
- 0..* entire nonnegative integer, i.e., 0, 1, 2, …



EXAMPLE

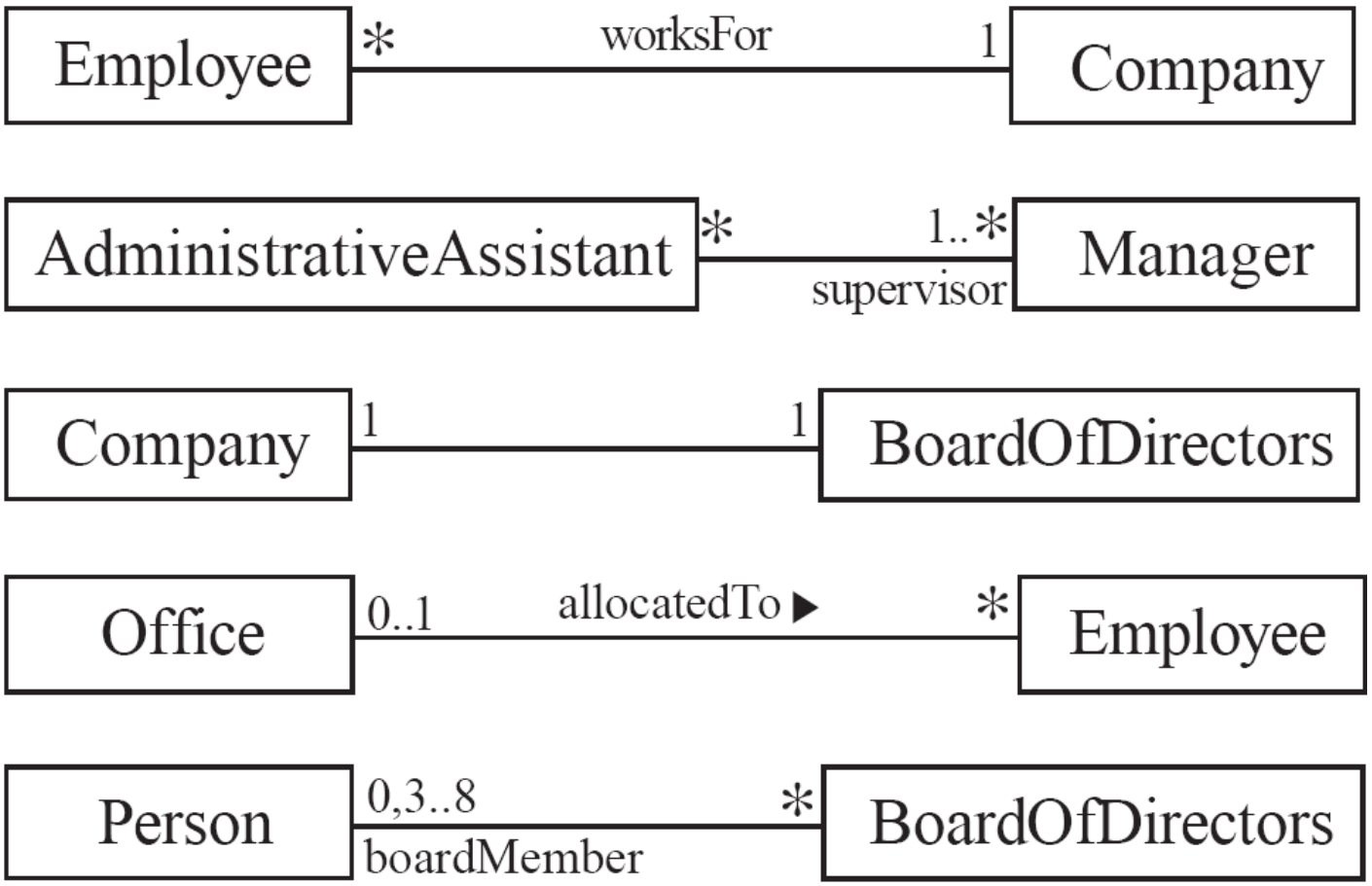

Student
6..*
enroll►
0..*
Course
advisee
0..*
1..*
▲ teach
1
1
Faculty
advisor

EXERCISE

Identify possible relationships among the following classes and draw a class diagram

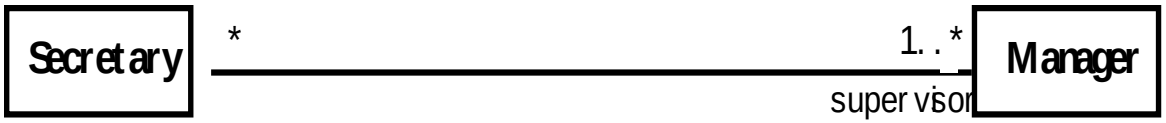
- Employee
- Manager
- Department

ASSOCIATIONS AND MULTIPLICITY



ANALYZING AND VALIDATING ASSOCIATIONS

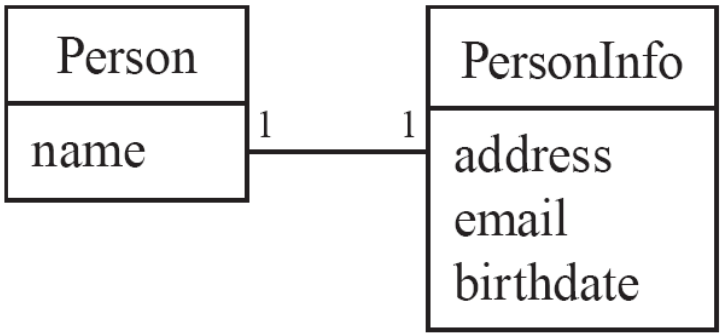
- **Many-to-many**
  - A secretary can work for many managers
  - A manager can have many secretaries
  - Secretaries can work in pools
  - Managers can have a group of secretaries
  - Some managers might have zero secretaries.
  - Is it possible for a secretary to have, perhaps temporarily, zero managers?



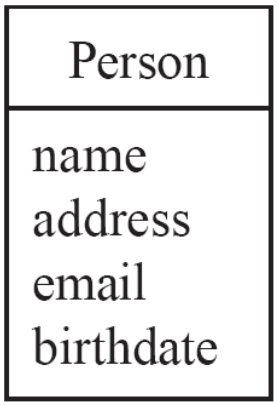
ANALYZING AND VALIDATING ASSOCIATIONS

Avoid unnecessary one-to-one associations

Avoid this



do this



DIRECTIONALITY IN ASSOCIATIONS

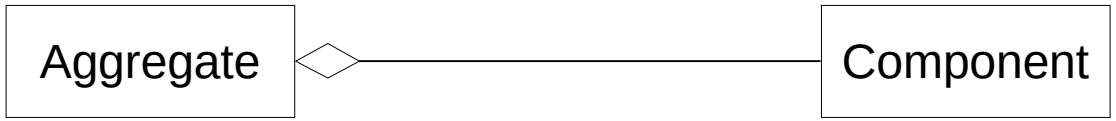
- Associations are by default are undefined, though many tools treat these as *bi-directional*.
- It is possible to limit the direction of an association by adding an arrow at one end

AGGREGATION

Special form of association representing *has-a* or *part-whole* relationship.

Distinguishes the whole (aggregate class) from its parts (component class).

No relationship in the lifetime of the aggregate and the components (can exist separately).

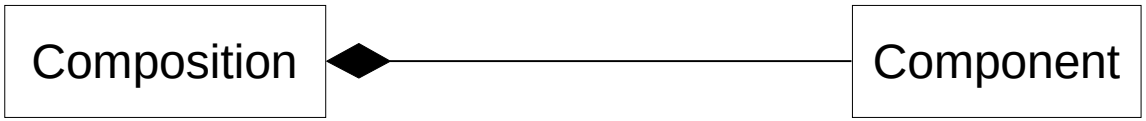


COMPOSITION

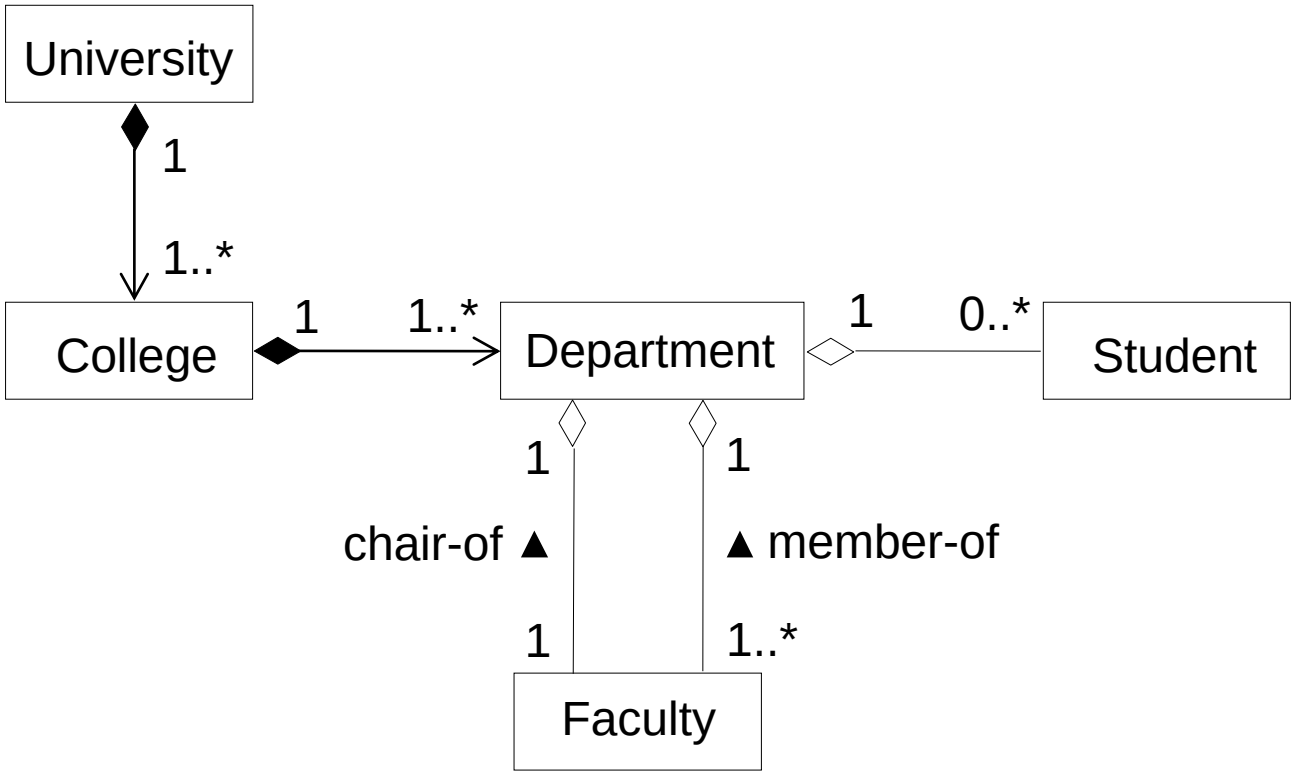
Stronger form of aggregation

Implies exclusive ownership of the component class by the aggregate class

The lifetime of the components is entirely included in the lifetime of the aggregate (a component can not exist without its aggregate).



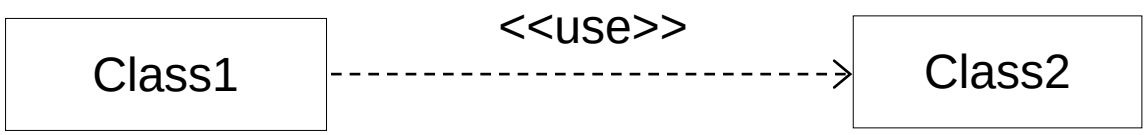
EXAMPLE


University
1
1..*
College
1
1..*
Department
1
0..*
Student
1
1
chair-of
member-of
1
1..*
Faculty

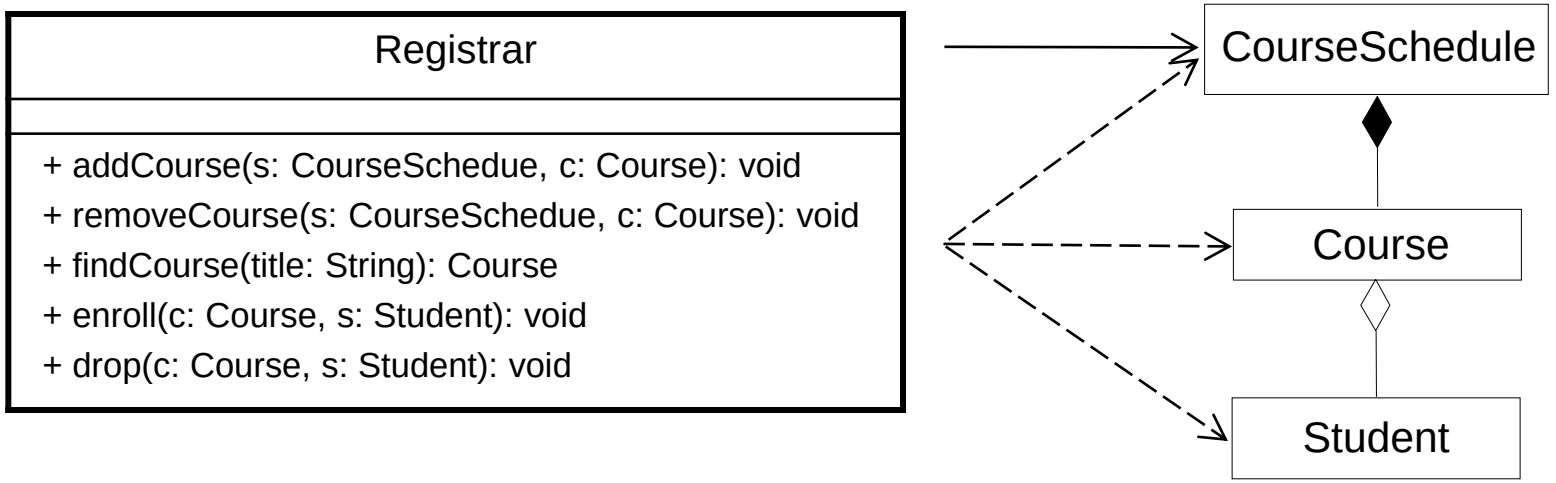
DEPENDENCY

Relationship between the entities such that the proper operation of one entity depends on the presence of the other entity, and changes in one entity would affect the other entity.

The common form of dependency is the *use* relation among classes.



EXAMPLE

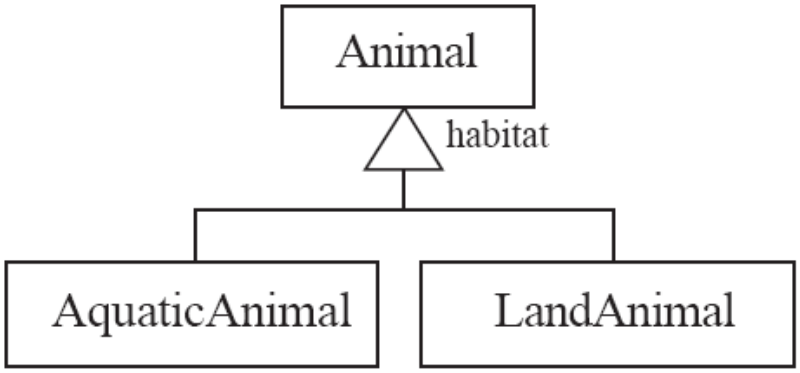


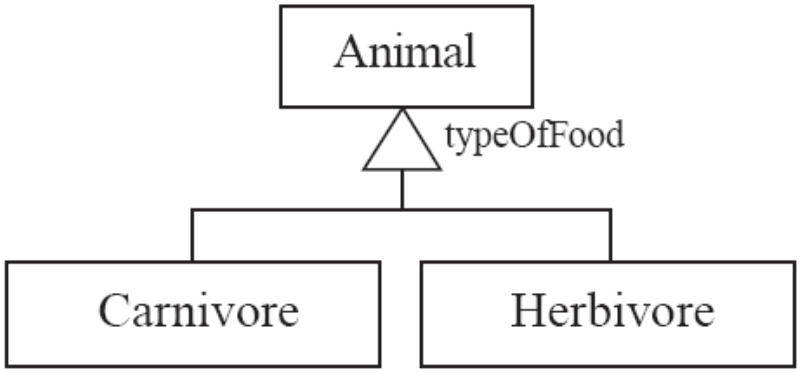
Dependencies are most often omitted from the diagram unless they convey some significant information.

GENERALIZATION

Specializing a superclass into two or more subclasses

- The *discriminator* is a label that describes the criteria used in the specialization

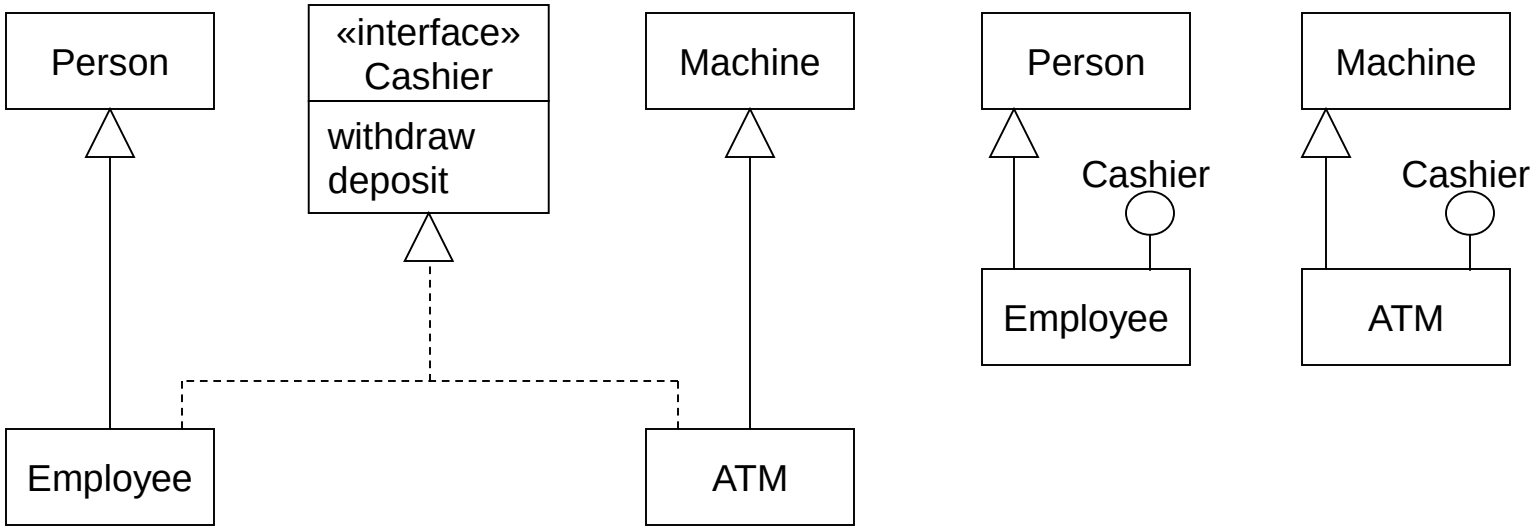




INTERFACES

An interface describes a *portion of the visible behaviour* of a set of objects.

- An *interface* is similar to a class, except it lacks instance variables and implemented methods



NOTES AND DESCRIPTIVE TEXT

- **Descriptive text and other diagrams**
  - Embed your diagrams in a larger document
  - Text can explain aspects of the system using any notation you like
  - Highlight and expand on important features, and give rationale
- **Notes:**
  - A note is a small block of text embedded *in* a UML diagram
  - It acts like a comment in a programming language

SUGGESTED SEQUENCE OF ACTIVITIES

- Identify a first set of candidate **classes** from the use cases
- Decide on specific **operations** and **attributes**
- Add relationships (**associations** and **generalizations**
- **Iterate** over the entire process until the model is satisfactory
  - Add or delete classes, responsibilities or operations associations, attributes, generalizations,
  - Identify interfaces

Don't be too disorganized. Don't be too rigid either.

IDENTIFYING CANDIDATE CLASS

List the candidate classes related to the "Process Sale" use case given below by identifying **nouns** and **noun phrases.**

Process Sale success scenario use case text:

Preconditions: Cashier is identified and authenticated on a sales terminal.

Customer arrives at POS checkout with goods and/or services to purchase.

Cashier starts a new sale.

Cashier enters item identifier.

System records sales line item and presents item description, price, and running total. Price calculated from a set of price rules.

< *Cashier repeats steps 3-4 until indicates done* >

System presents total with taxes calculated.

Cashier tells Customer the total, requests payment.

Customer pays and System handles payment.

System logs completed sale and sends sale and payment information to the external Accounting System and Inventory System.

System presents receipt.

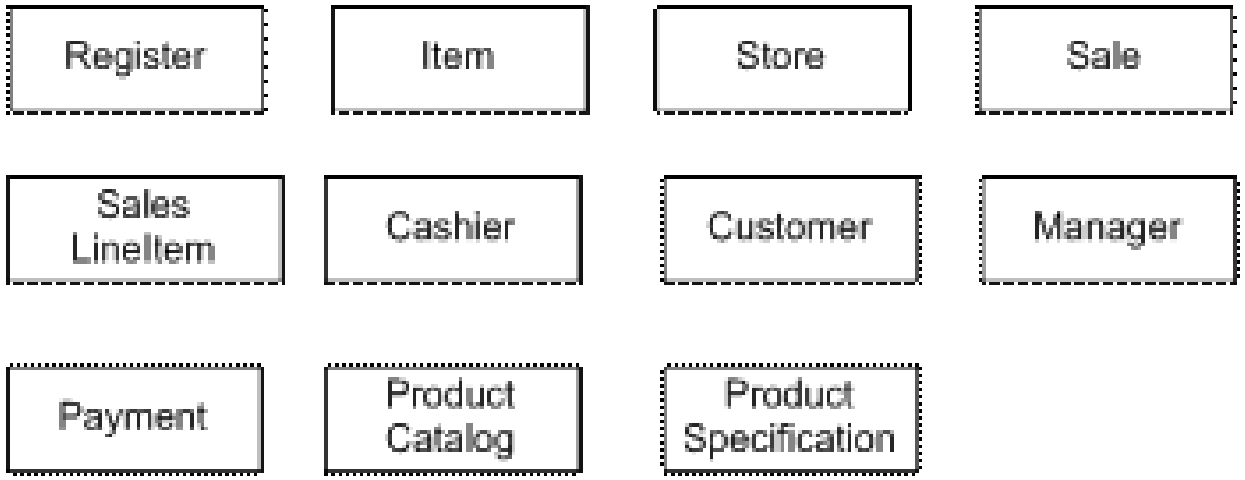
Customer leaves with receipt and goods (if any).

IDENTIFYING NOUNS AND NOUN PHRASES

Customer arrives at **POS checkout** with **goods** and/or **services** to purchase.
Cashier starts a new **sale.**
Cashier enters **item identifier.**
System records **sales line item** and presents **item description, price,** and **running total.** Price calculated from a set of price rules.
< *Cashier repeats steps 3-4 until indicates done* >
System presents total with *taxes* calculated.
Cashier tells Customer the total, requests **payment.**
Customer pays and System handles payment.
System logs completed **sale** and sends sale and payment information to the external **Accounting** System and **Inventory** System.
System presents **receipt.**
Customer leaves with receipt and goods (if any).

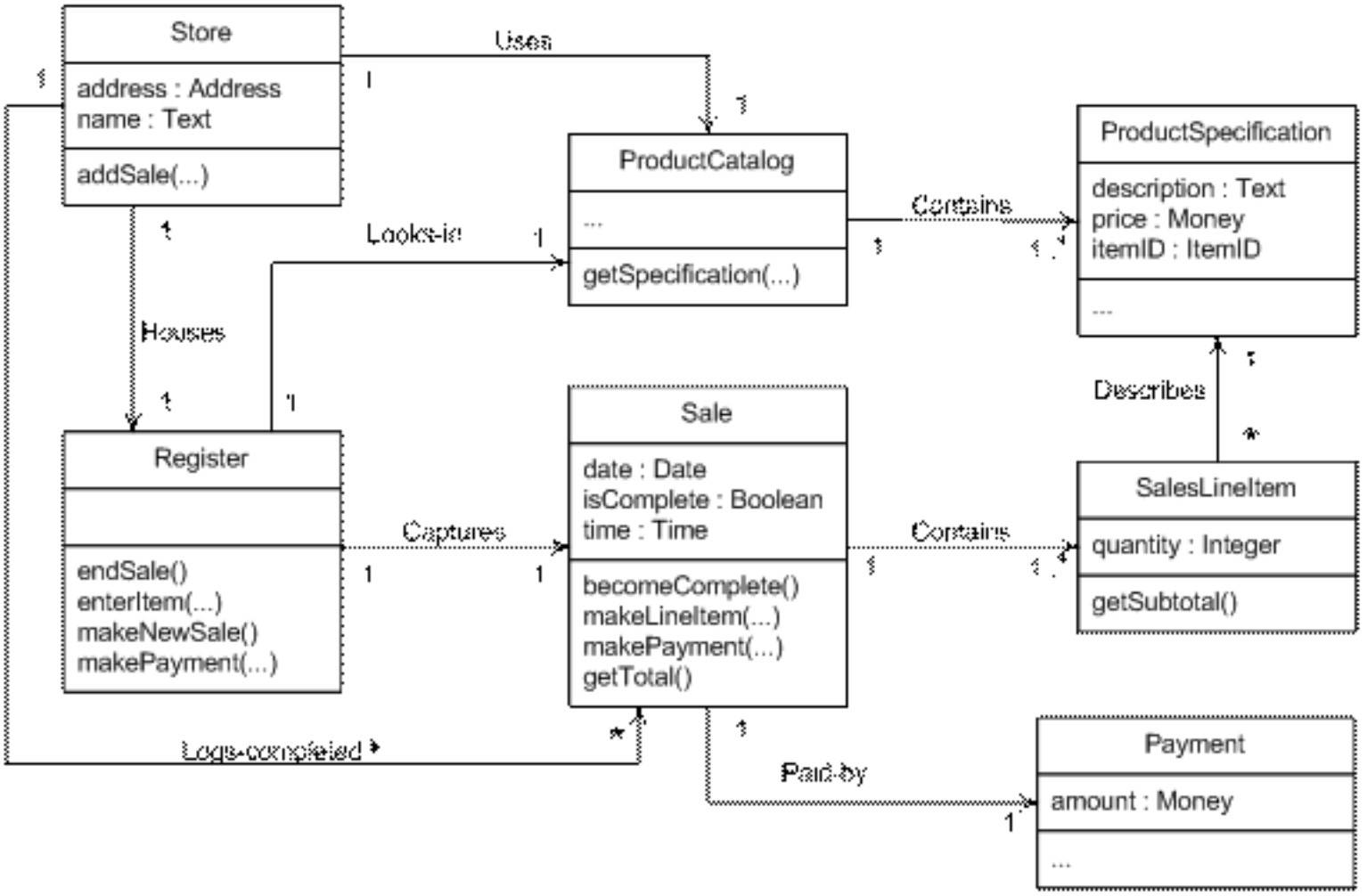
Note the differences between **class and attributes** – not all nouns automatically become classes.

CONCEPTUAL CLASSES



Moving to software classes, Cashier, Customer & Manager are not considered, Item becomes an attribute in ProductSpecification.

CLASS DIAGRAM



MAPPING CLASS DIAGRAM TO CODE

```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

public SalesLineItem(ProductDescription desc, int qty) { ... }

public Money getSubtotal() { ... }

}
```

UML INTERACTION MODELS

An interaction model shows the interactions that take place between objects in a system

An interaction "is a behavior that comprises a set of messages exchanged among a set of objects within a context to accomplish a purpose" (UML user guide)

Interaction models provide a view of system behavior

INTERACTION DIAGRAM

Models how groups of objects collaborate to realize some behaviour

Typically each interaction diagram realizes behaviour of a single use case

Two kinds: Sequence and Communication diagrams.

Two diagrams are equivalent

- Portray different perspectives

These diagrams play a very important role in the design process.

SEQUENCE DIAGRAM

Shows interaction among objects as a two-dimensional chart

Objects are shown as boxes at top

If object created during execution then shown at appropriate place

Objects existence are shown as dashed lines (lifeline)

Objects activeness, shown as a rectangle on lifeline

SEQUENCE DIAGRAM CONT...

Messages are shown as arrows

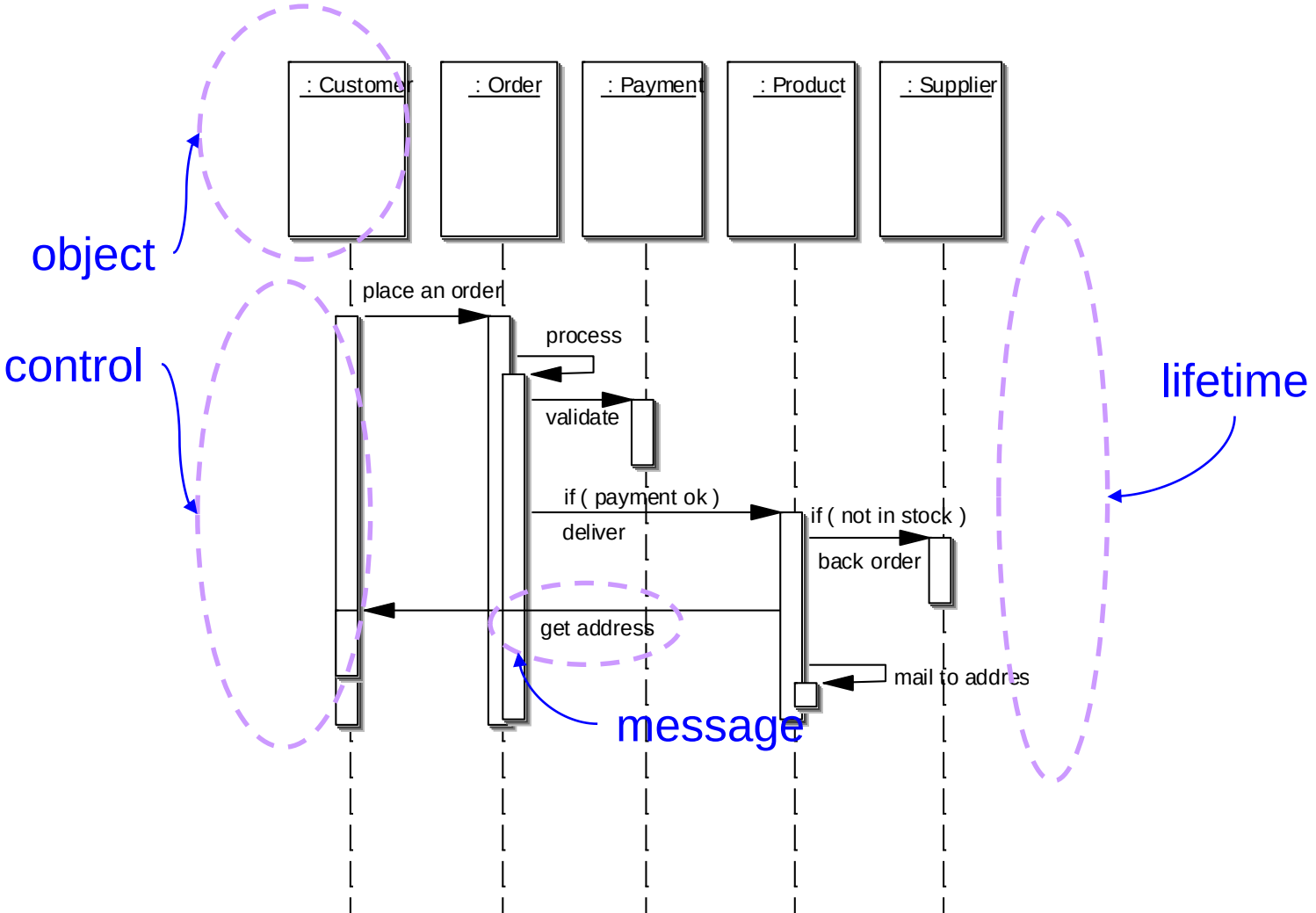
Each message labelled with corresponding message name

Each message can be labelled with some control information

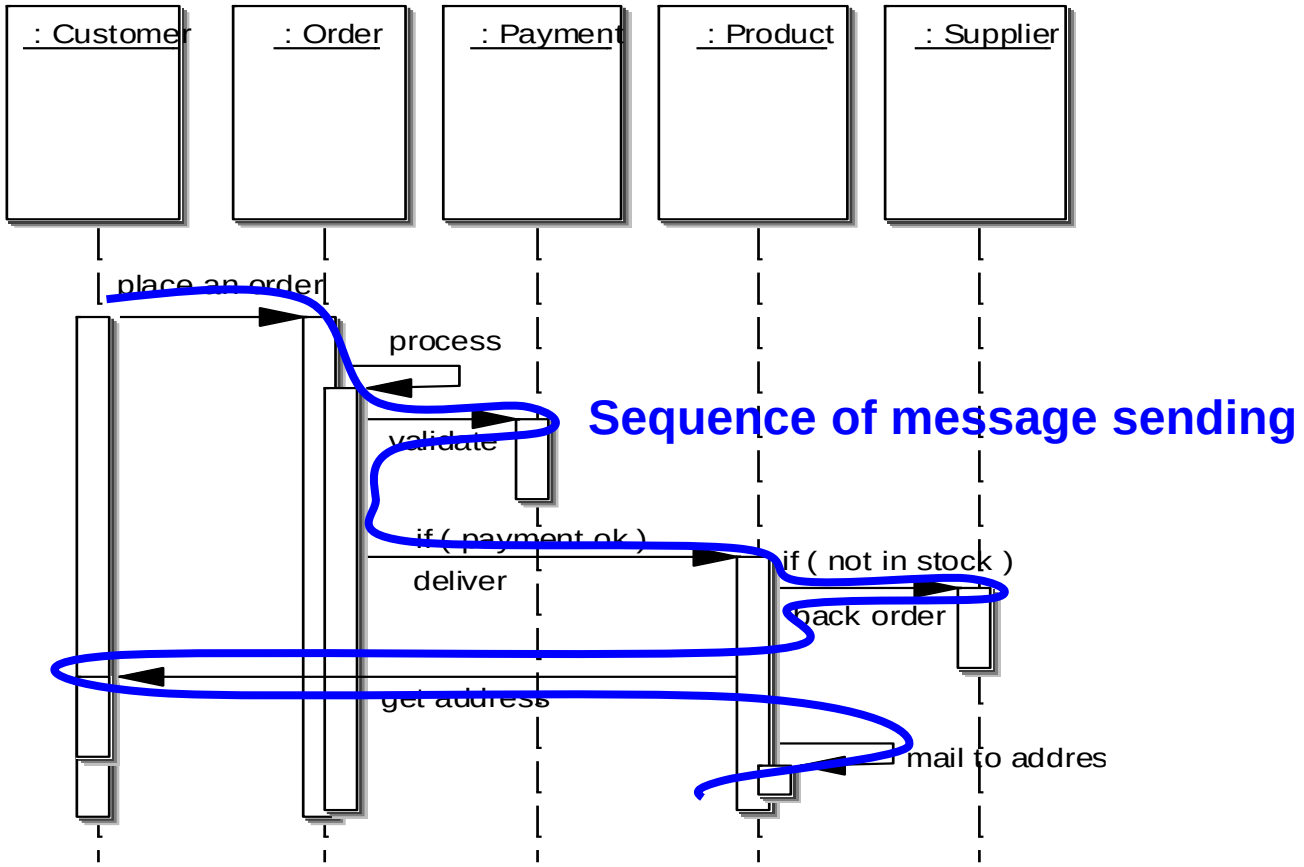
Two types of control information

- condition ([])
- iteration (*)

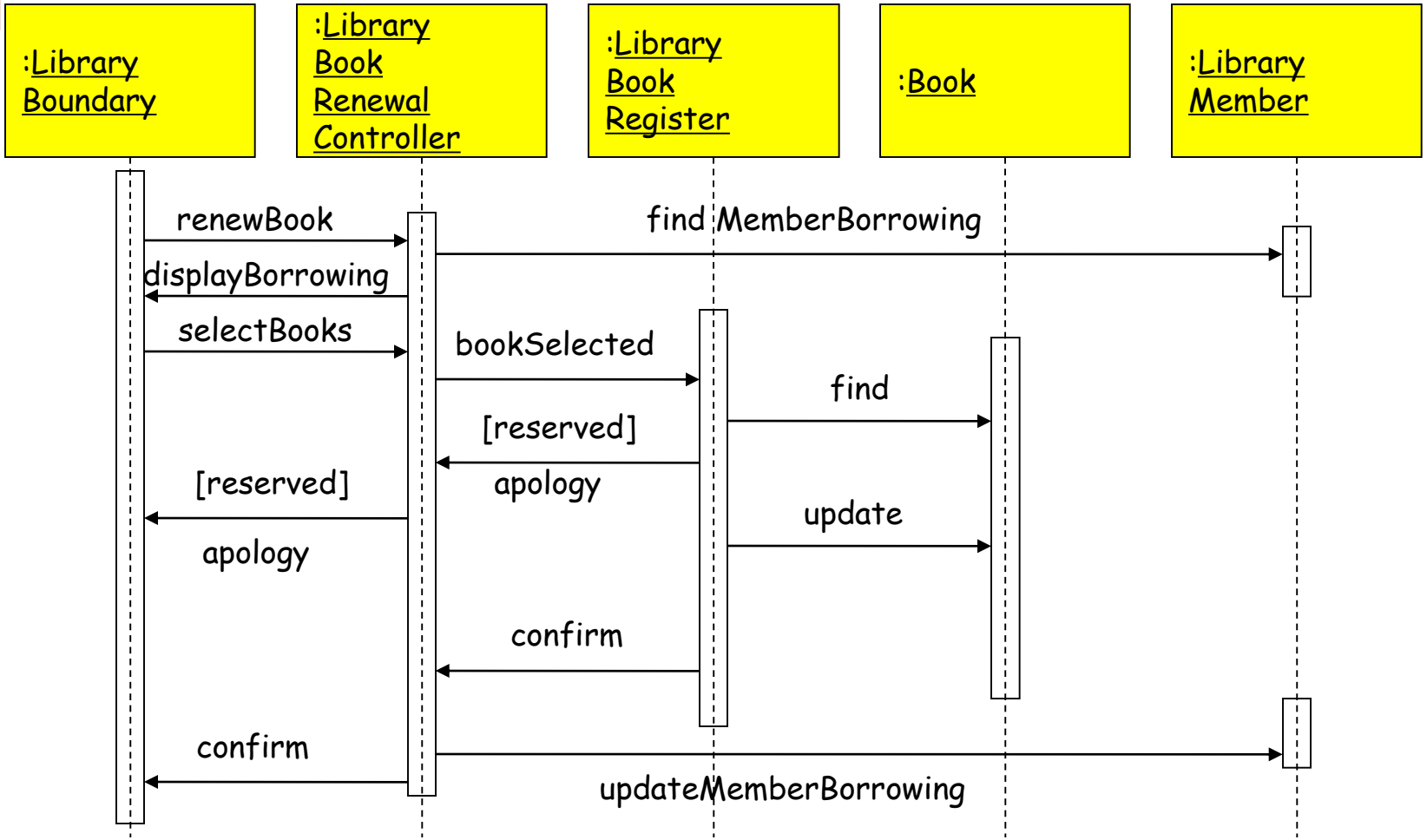
ELEMENTS OF A SEQUENCE DIAGRAM



EXAMPLE CONT...

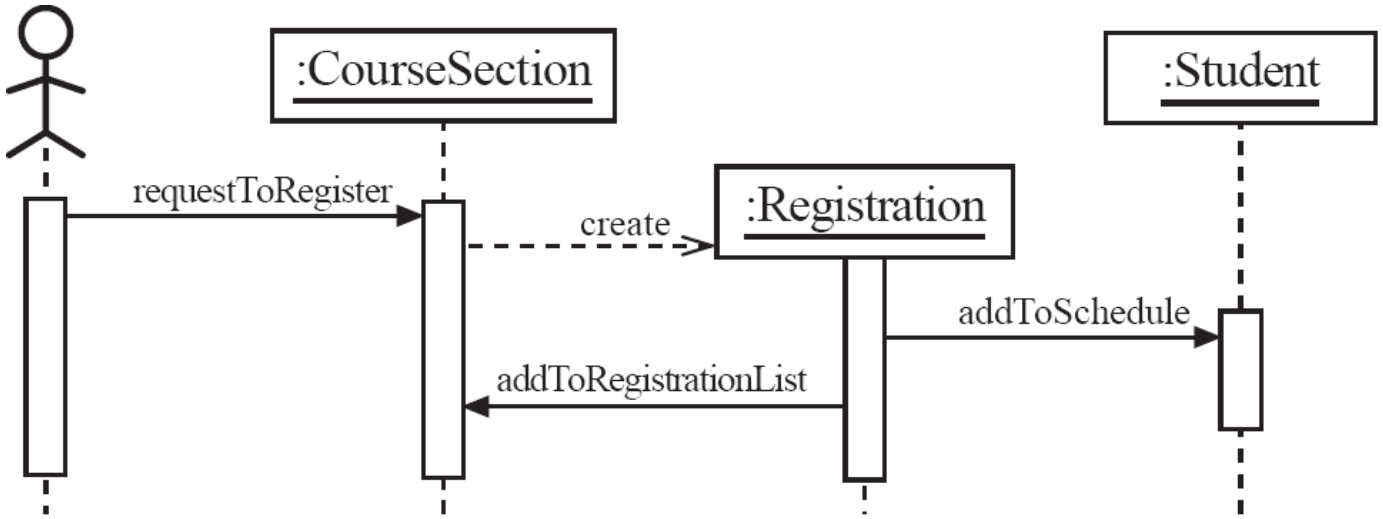


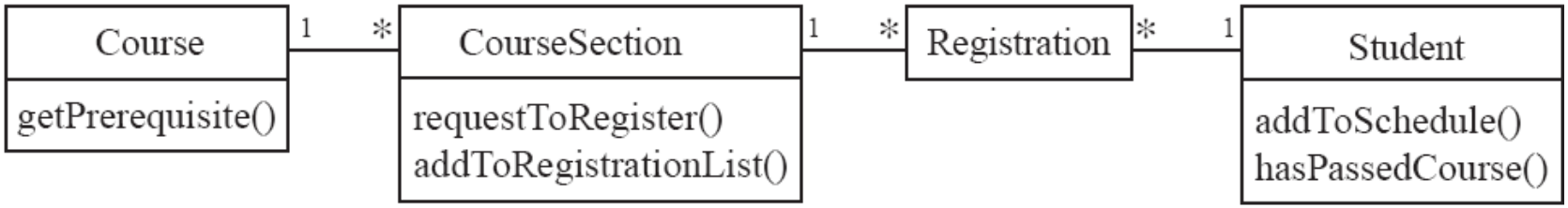
AN EXAMPLE OF A SEQUENCE DIAGRAM



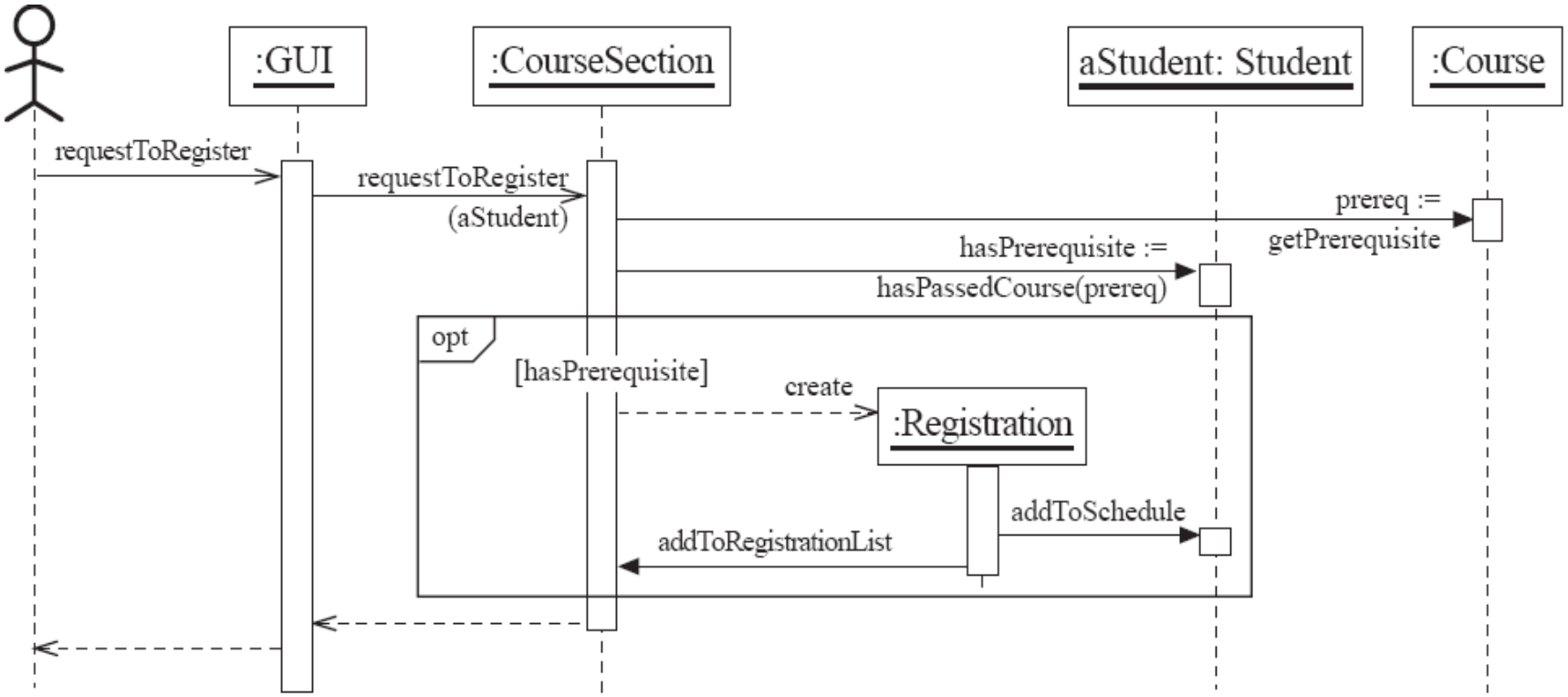
Sequence Diagram for the renew book use case

SEQUENCE DIAGRAMS – AN EXAMPLE



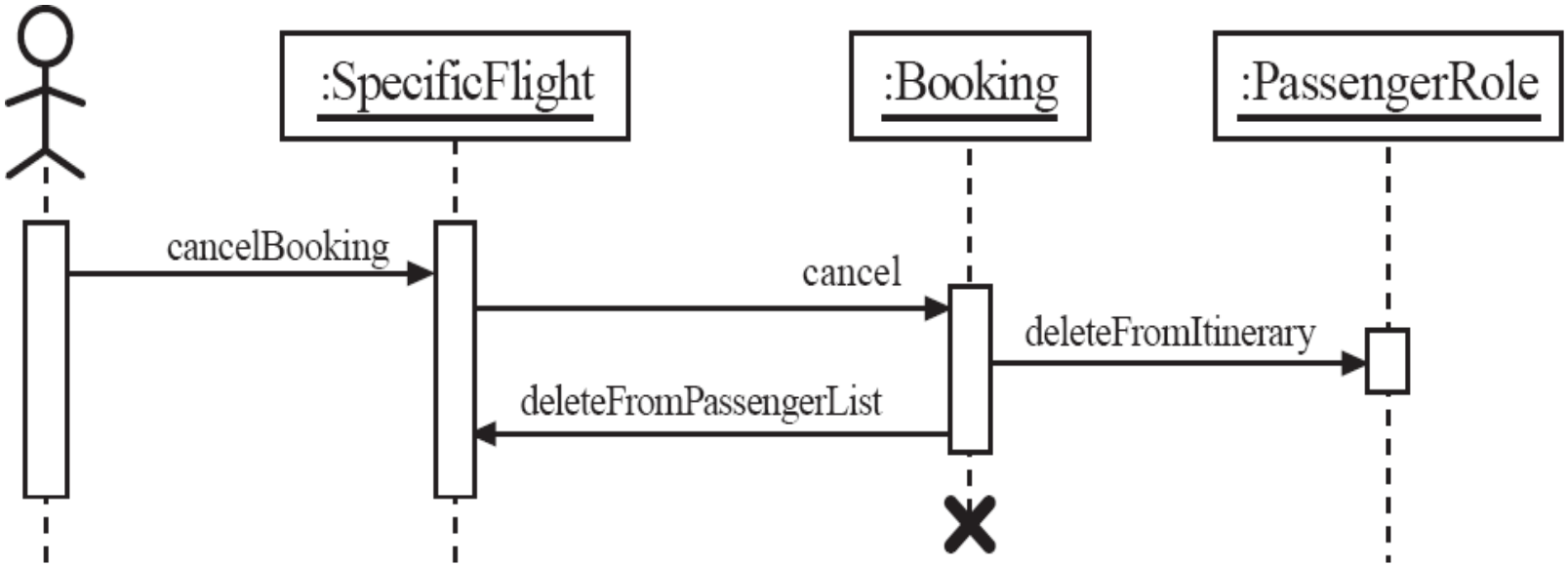


SEQUENCE DIAGRAMS – SAME EXAMPLE, MORE DETAILS



SEQUENCE DIAGRAMS – AN EXAMPLE WITH OBJECT DELETION

- If an object's life ends, this is shown with an X at the end of the lifeline



STATE CHART DIAGRAM

Based on the work of **David Harel** [1990]

Model how the state of an object changes in its lifetime

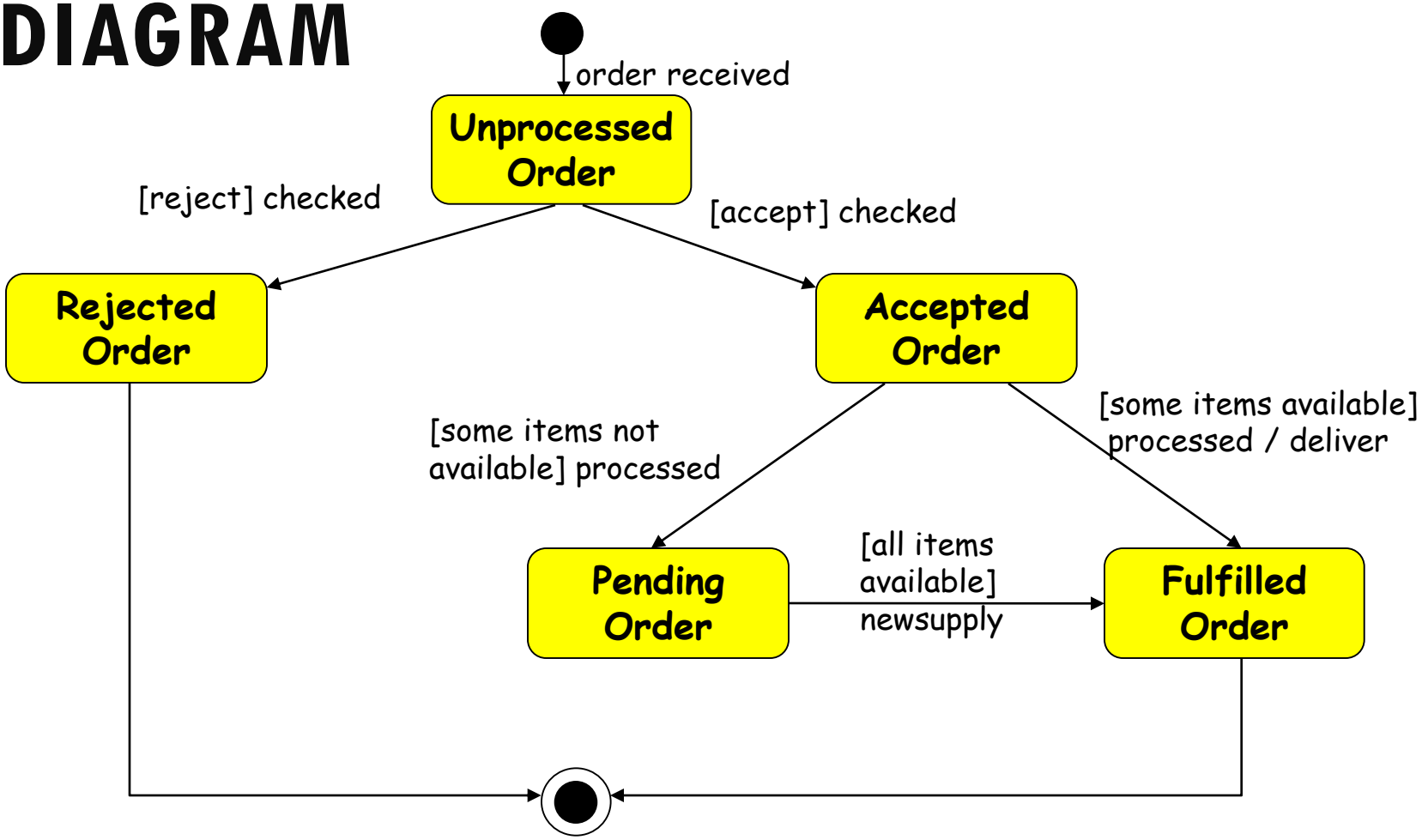
Based on finite state machine (FSM) formalism

State chart avoids the problem of state explosion of FSM.

Hierarchical model of a system:

- Represents composite **nested** states

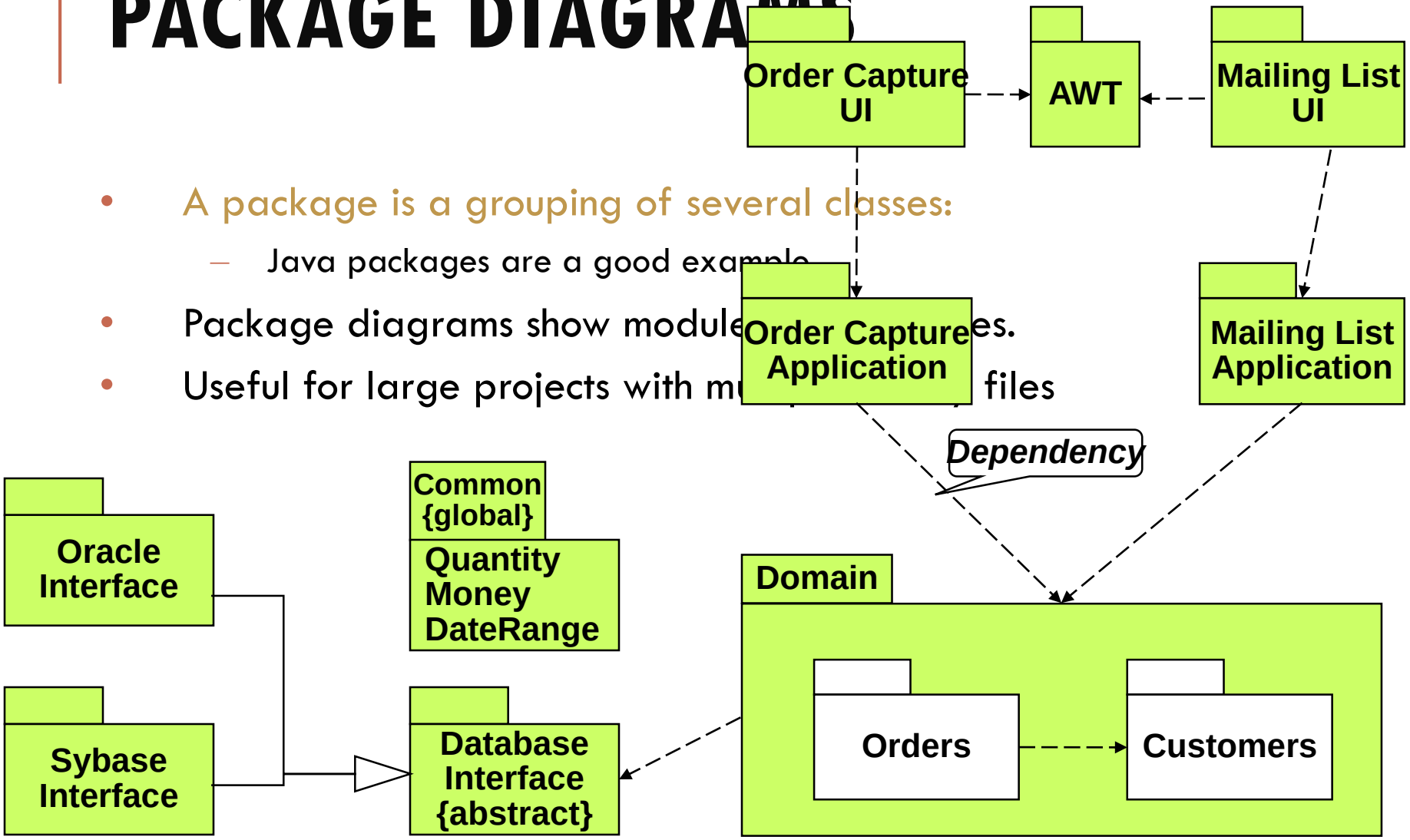
AN EXAMPLE OF A STATE CHART DIAGRAM



Example: State chart diagram for an order object

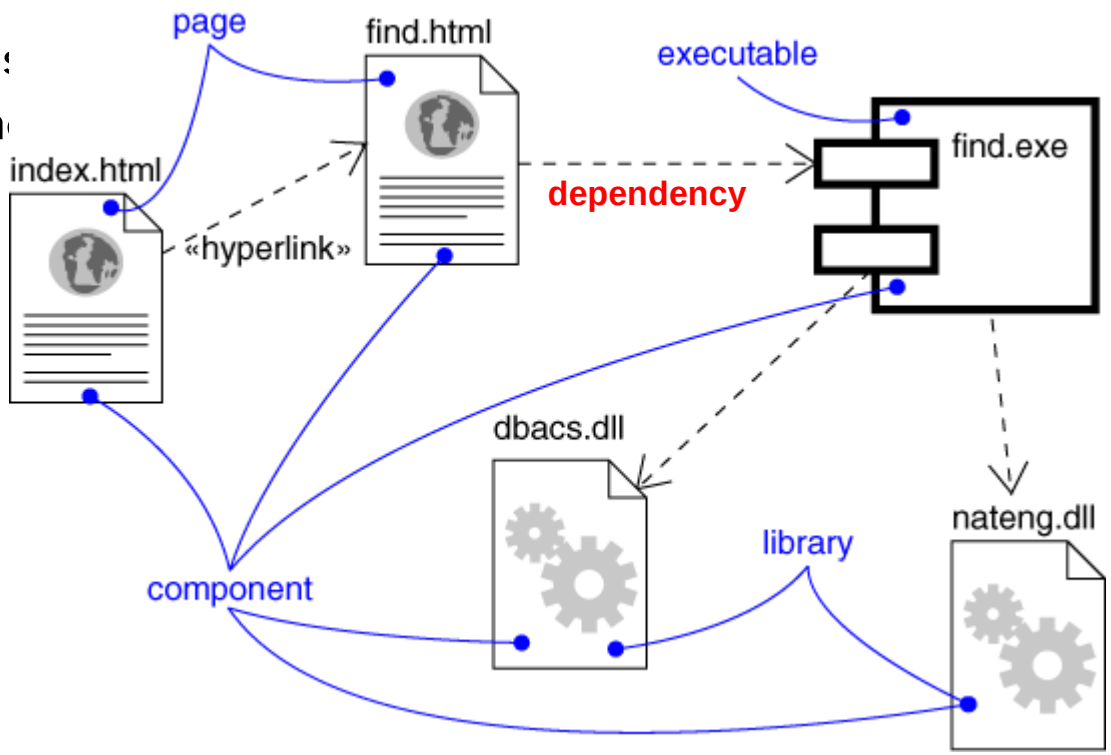
PACKAGE DIAGRAMS

- A package is a grouping of several classes:
  - Java packages are a good example
- Package diagrams show module[illegible]es.
- Useful for large projects with mu[illegible] files



COMPONENT DIAGRAM

Capture
compon



Components:

- **Executables**
- **Library**
- **Table**
- **File**
- **Document**

COMPONENT DIAGRAM

Captures the physical structure of the implementation

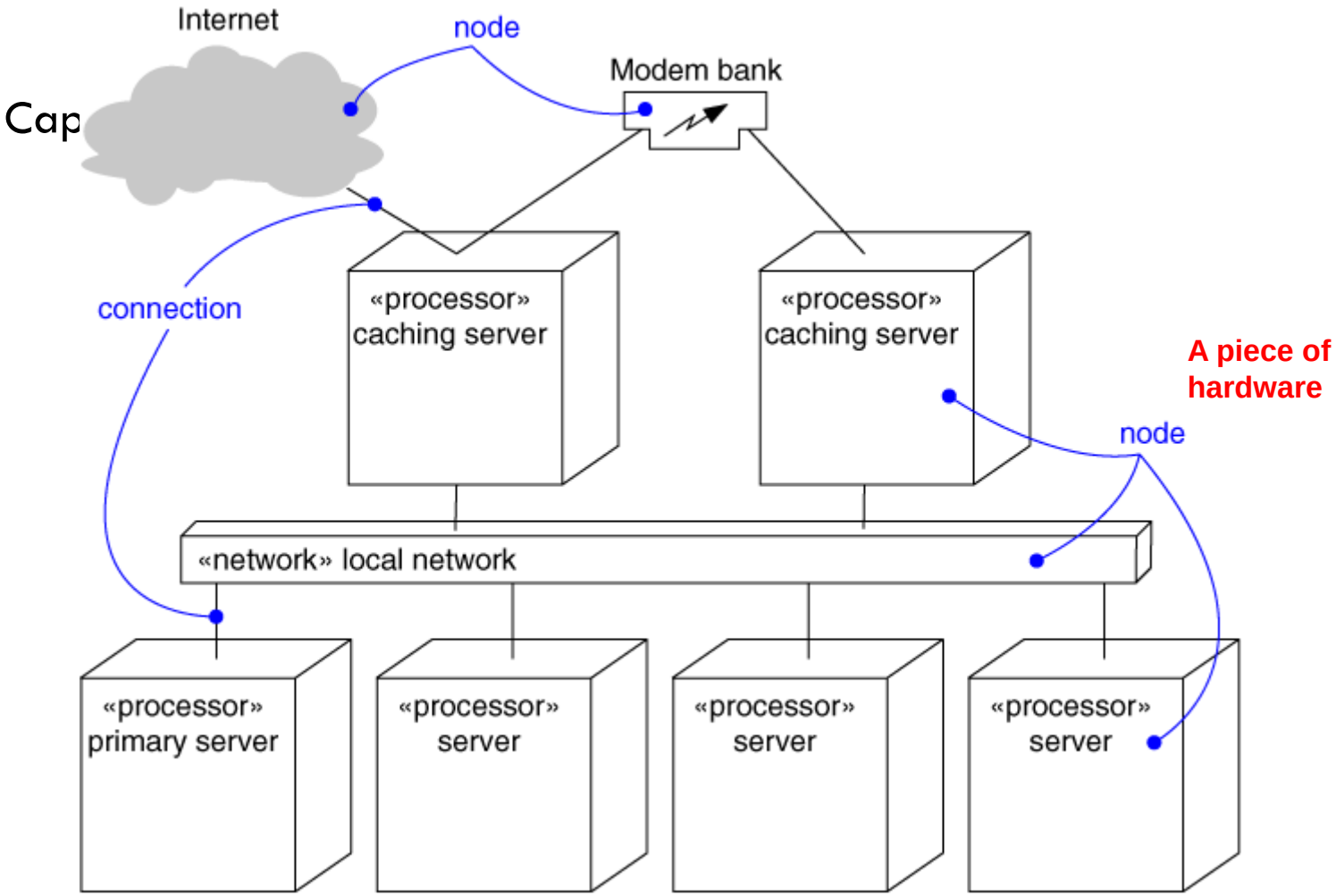
Built as part of architectural specification

Purpose

- Organize source code
- Construct an executable release
- Specify a physical database

Developed by architects and programmers

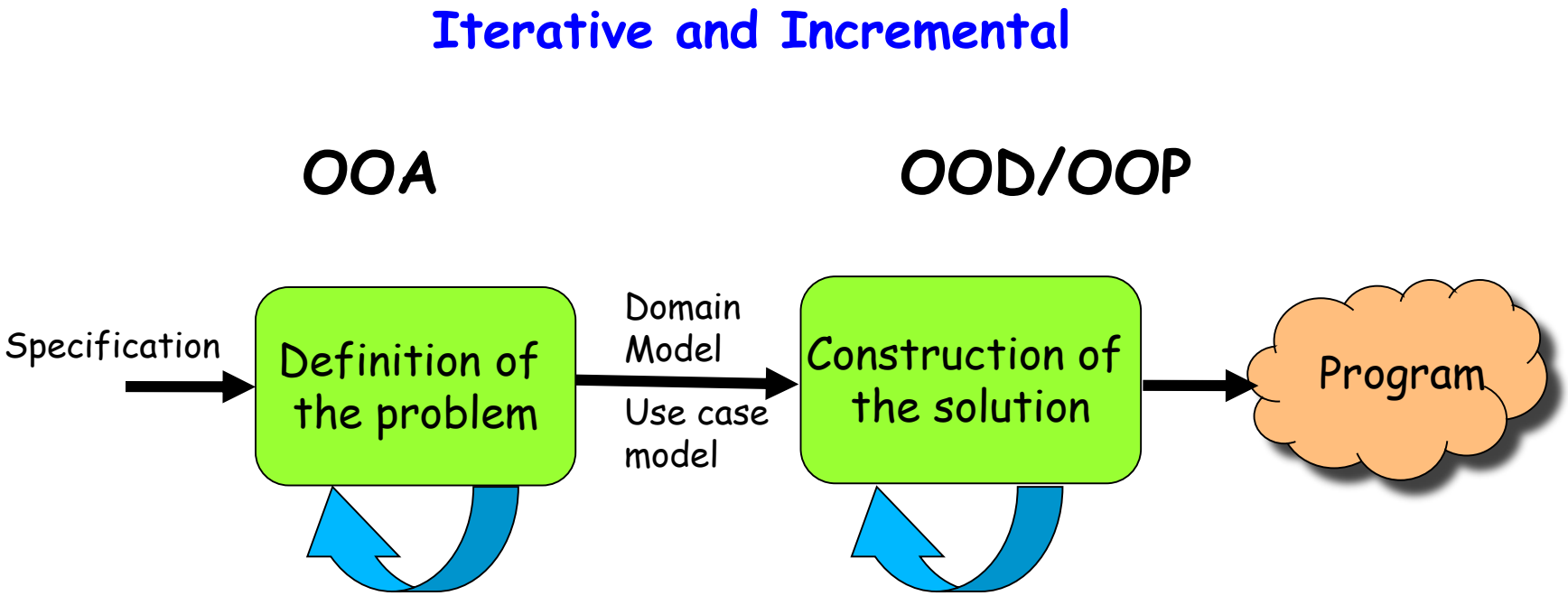
DEPLOYMENT DIAGRAM



A DESIGN PROCESS

- Developed from various methodologies.
  - However, UML has been designed to be usable with any design methodology.
- From requirements specification, initial model is developed (OOA)
  - Analysis model is iteratively refined into a design model
- Design model is implemented using OO concepts

OOAD


Iterative and Incremental
OOA
OOD/OOP
Specification
Definition of the problem
Domain Model
Use case model
Construction of the solution
Program

PACKAGE EXAMPLES