Modèles et système complexe

Cedric Dumoulin

Construire un système complexe
Un immeuble

- Pour construire un immeuble
  - On part d'une idée :
    - croquis, maquettes, descriptions
  - Un architecte fait des plans
    - plans détaillés
  - Plusieurs points de vues :
    - gros œuvre, électriciens, plombiers …

Point de vue utilisateur

- Les plans et les maquettes permettent de tester avant de construire

Construire un système complexe
Une application

- Pour construire une application
  - On recueil les besoins ou on imagine ce que fera l'application
    - scénarios concrets, croquis d'écrans Point de vue utilisateur
  - on fait des modèles :
    - diagrammes de classes, de CU, de séquences …
  - plusieurs points de vues :
    - concepteurs : haut niveau d'abstraction
    - développeurs : concepts plus proche des technologies et de l'implémentation
- Les scénarios et les modèles permettent de tester avant de développer
- Les modèles peuvent être productifs : on peut générer du code

Trois façon d'aborder la complexité

Gérer la complexité

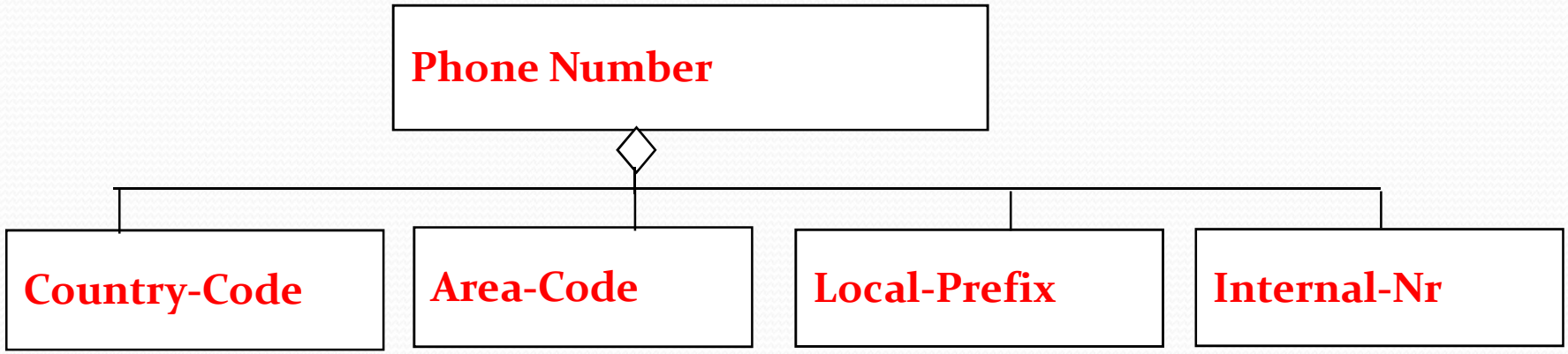
- Trois outils :
  - **Abstraction**
  - Décomposition
  - Hiérarchie

Gérer la complexité : 1) Abstraction

- Les systèmes complexe sont difficile à comprendre
  - Le phénomème 7 + 2
    - Notre mémoire à court terme ne peut pas stocker plus de 7 +- 2 pieces en même temps => limitation du cerveau
    - Mon numero de tel est: 498928918204

Abstraction

- Découpage en morceaux:
  - Grouper les objets en collections pour réduire la compléxité
  - 4 morceaux:
    - State-code, Area-code, Local-Prefix, Internal-Nr



Abstraction

- L’abstraction permet d’ignorer les détails non essentiels
- Deux définitions pour l’abstraction :
  - L'abstraction est un processus de pensée dans lequel l’idée s’éloigne des objets
    - Abstraction comme **activité**
  - L’abstraction est l'idée résultant d'un processus de pensée dans lequel l’idée s’est éloignée d'un objet
    - Abstraction en tant **qu'entité**
- Les idées peuvent être exprimées par des modèles

Models

- Un modèle est une abstraction d'un système
  - Un système qui n'existe plus
  - Un système existant
  - Un système à venir à construire.

Gérer la complexité :
2) Décomposition

- Une technique utilisée pour maîtriser la complexité
("divide and conquer")
- Deux grands types de décomposition
  - Décomposition fonctionnelle
  - Décomposition orientée objet
- Décomposition fonctionnelle
  - Le système est décomposé en modules
  - Chaque module est une fonction importante dans le domaine d'application
  - Les modules peuvent être décomposées en modules plus petits.

Decomposition (cont'd)

- Décomposition orientée objet
  - Le système est décomposé en classes ("objects")
  - Chaque classe est une entité majeur dans le domaine d'application
  - Les classes peuvent être décomposées en plus petites classes
- Décomposition orientée objet vs fonctionnel

On s'interresse à la décomposition objet

Identification des classes

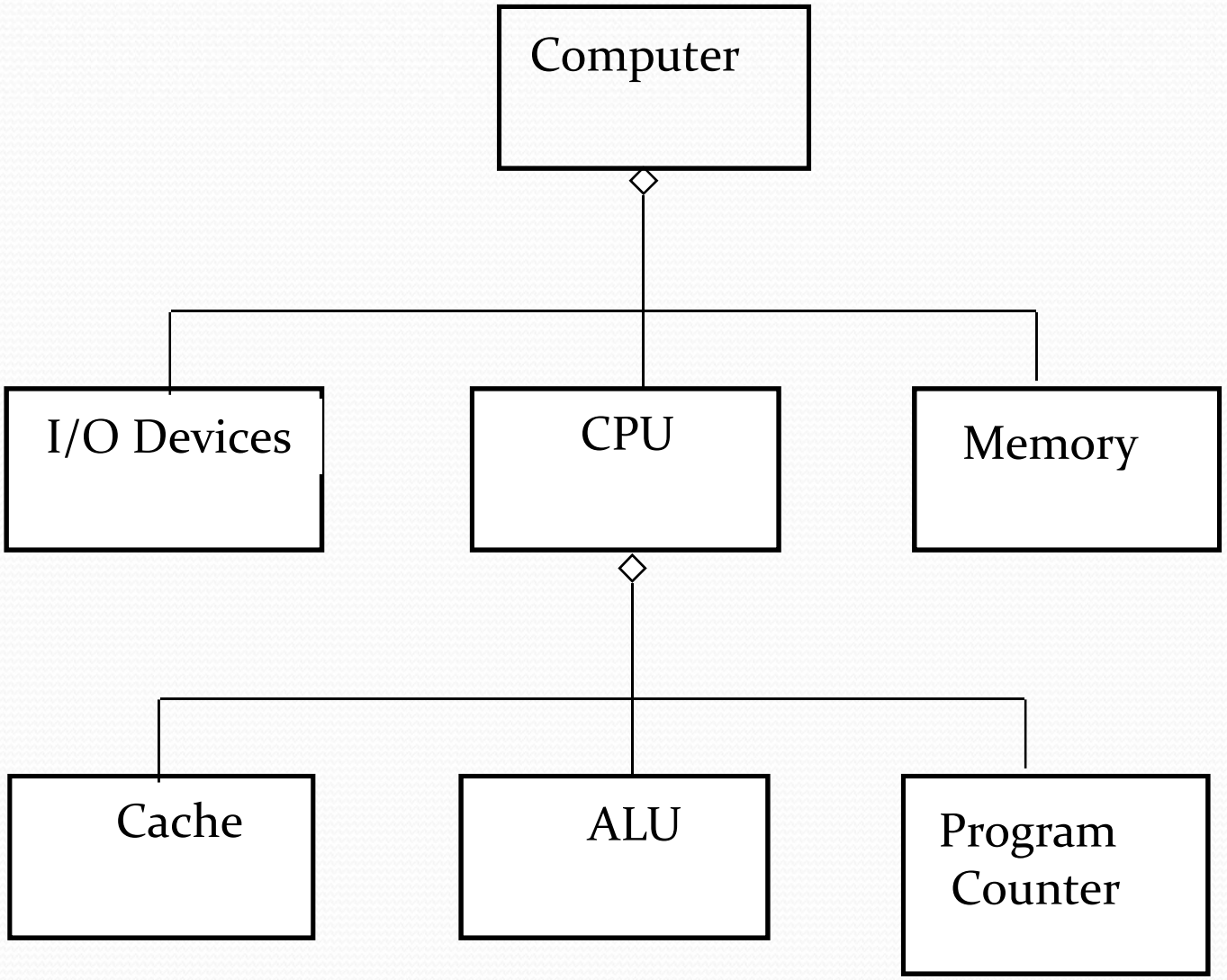
- **hypothèses de base :**
  - Nous pouvons trouver les classes pour un nouveau système logiciel : Ingéniérie à partir de zéro
  - Nous pouvons identifier les classes dans un système existant : Retro-ingénierie
  - Nous pouvons créer des interfaces basées sur les classes pour système existant : Interface Engineering.

Gérer la complexité :
3) Hierarchie

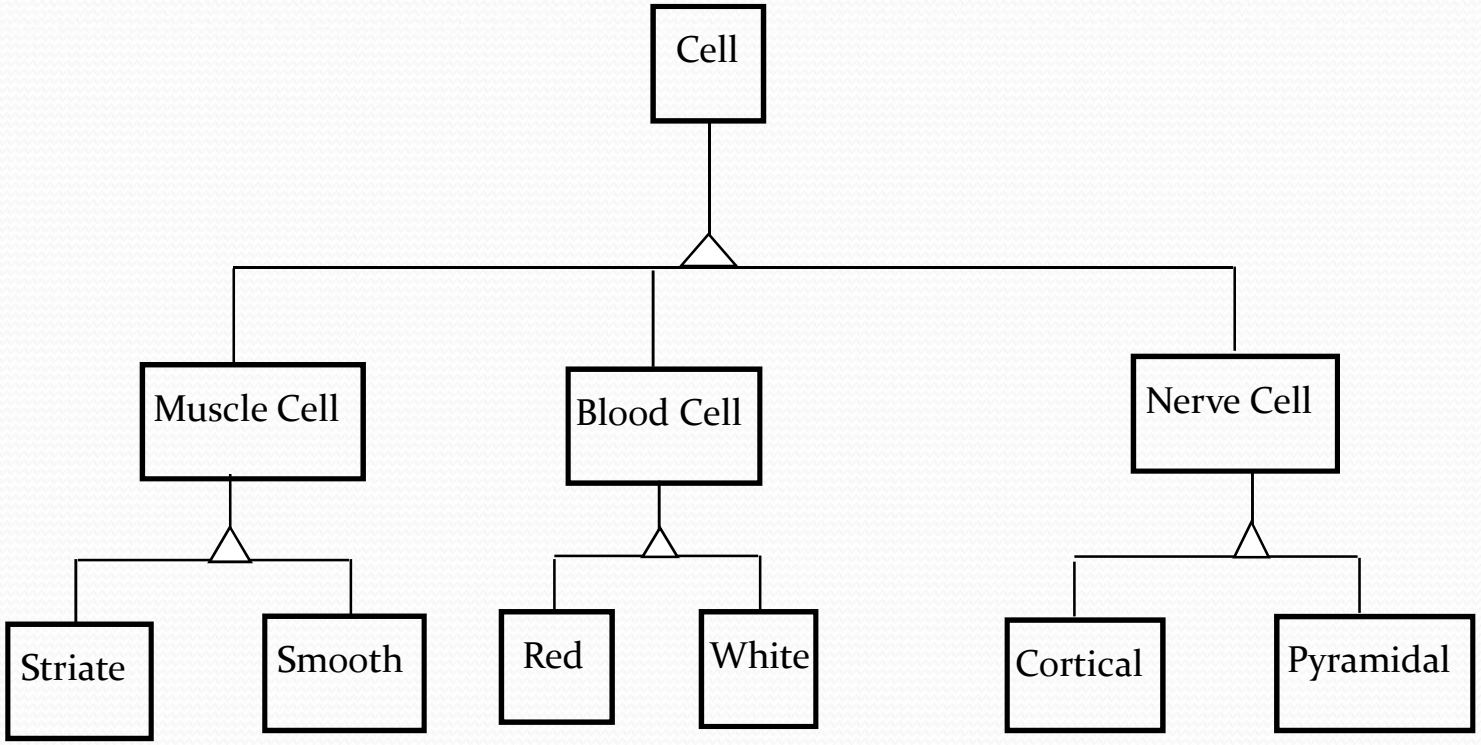
- So far we got abstractions
  - This leads us to classes and objects
  - "Chunks"

- Another way to deal with complexity is to provide relationships between these chunks
- One of the most important relationships is hierarchy
- 2 special hierarchies
  - "Part-of" hierarchy
  - "Is-kind-of" hierarchy.

Part-of Hierarchy (Aggregation)



Is-Kind-of Hierarchy (Taxonomy)



Where are we?

- Three ways to deal with complexity:
  - Abstraction, Decomposition, Hierarchy
- Object-oriented decomposition is good
  - Unfortunately, depending on the purpose of the system, different objects can be found
- How can we do it right?
  - Start with a description of the functionality of a system
  - Then proceed to a description of its structure
- ➔ Ordering of development activities
  - Software lifecycle

Quelques définitions

Système, sous-système

Vues

Systems

- A *system* is an organized set of communicating parts
  - Natural system: A system whose ultimate purpose is not known
  - Engineered system: A system which is designed and built by engineers for a specific purpose
- The parts of the system can be considered as systems again
  - In this case we call them *subsystems*

Examples of natural systems:

- Universe, earth, ocean

Examples of engineered systems:

- Airplane, watch, GPS

Examples of subsystems:

- Jet engine, battery, satellite.

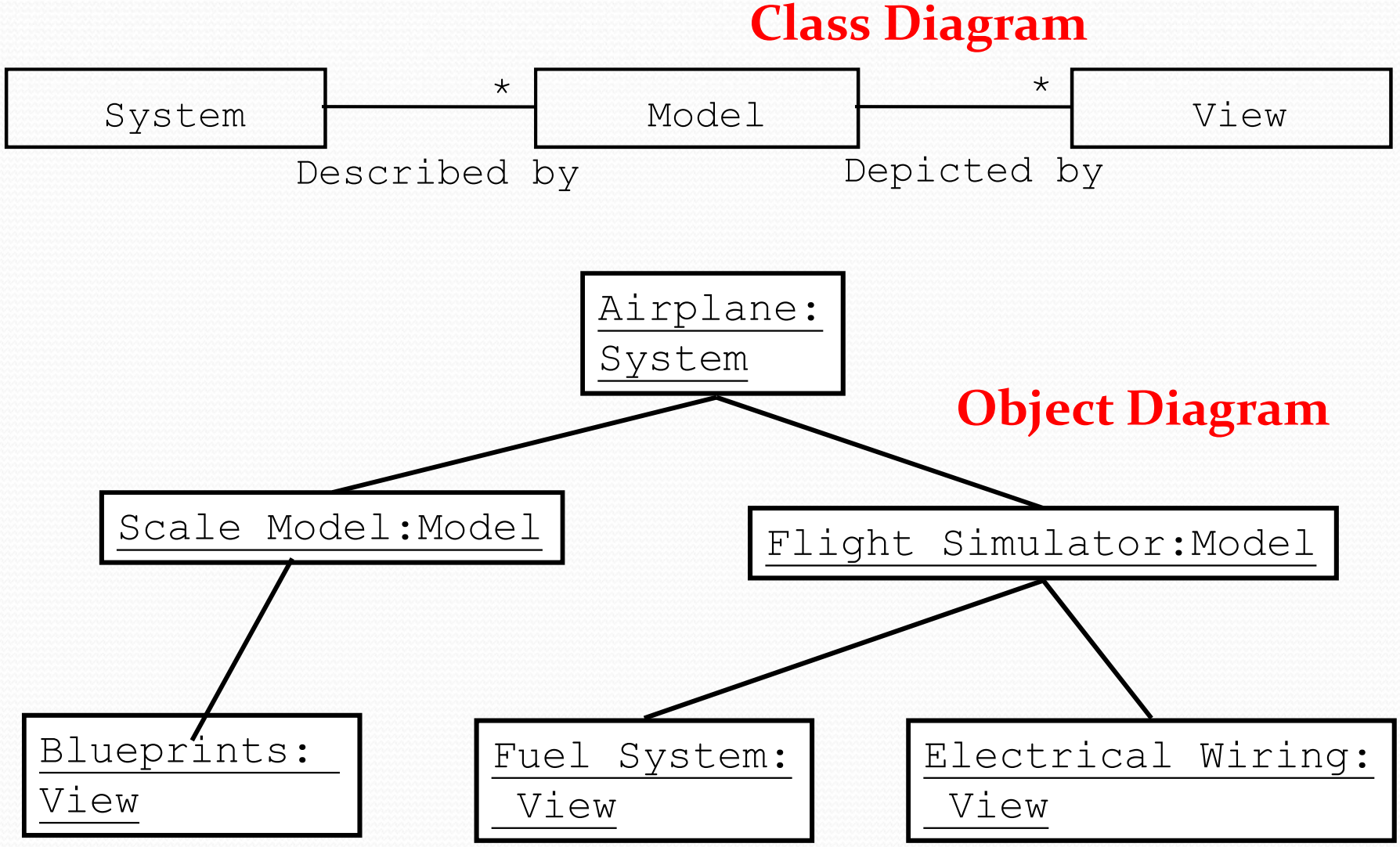
Systems, Models and Views

- A ***model*** is an abstraction describing a system or a subsystem
- A ***view*** depicts selected aspects of a model
- A ***notation*** is a set of graphical or textual rules for depicting models and views:
  - formal notations, "napkin designs"

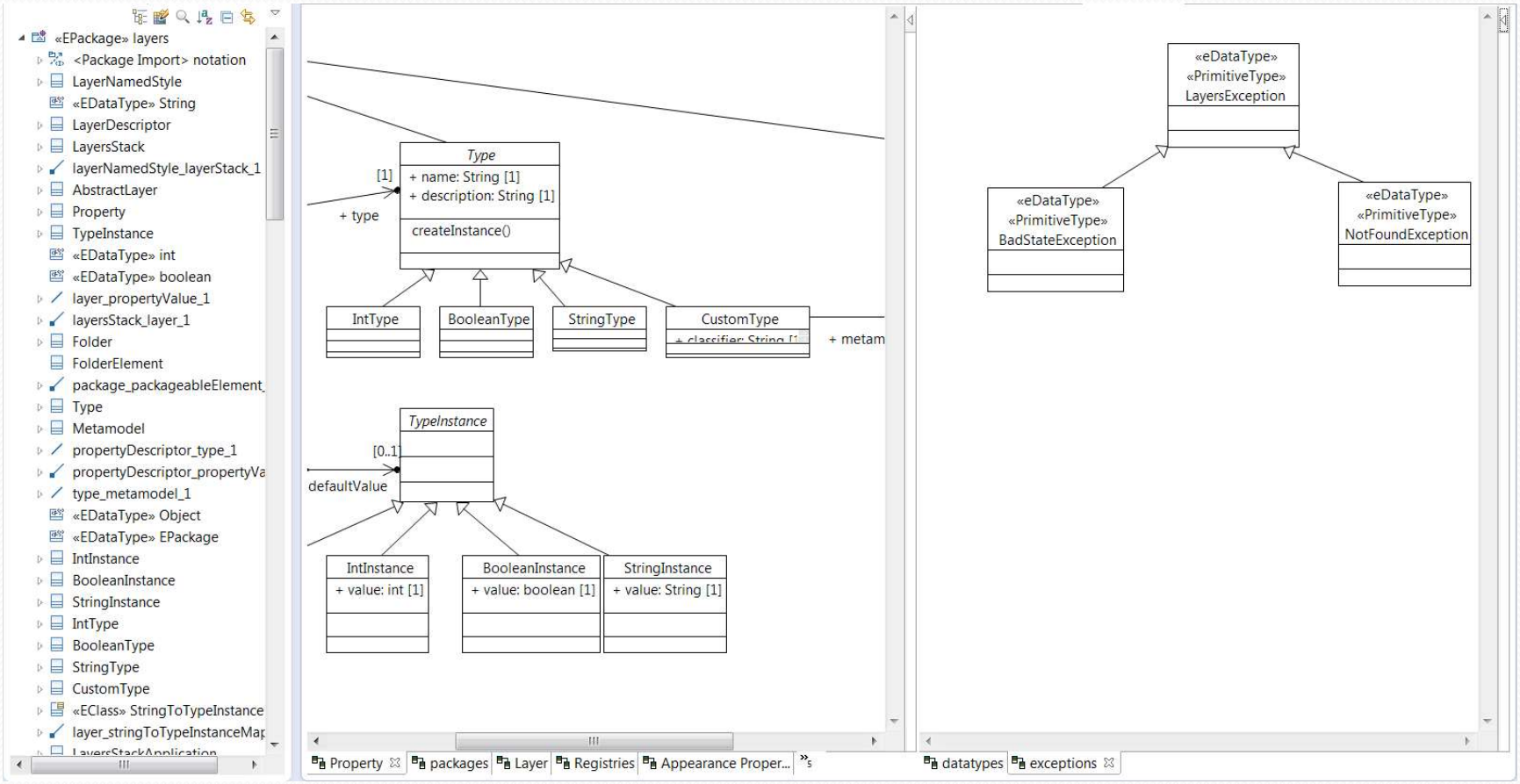
System: Airplane

Models:	**Views:**
Flight simulator	Blueprint of the airplane components
Scale model	Electrical wiring diagram, Fuel system Sound wave created by airplane

Systems, Models and Views (UML Notation)



Models and Views (Modeleur UML)



IDM

1. Construit des modèles à un haut niveau d'abstraction, et indépendant des technologies
    1. Décrit le modèle à l'aide d'une notation (ex: UML)
    2. convertit le modèles vers des modèles de plus bas niveau
2. Génère du code (quasi-)executable

Avantages:

- Le code est généré à partir des modèles ("pour la plupart")
- Portabilité, interopérabilité

Introduction à la notation UML

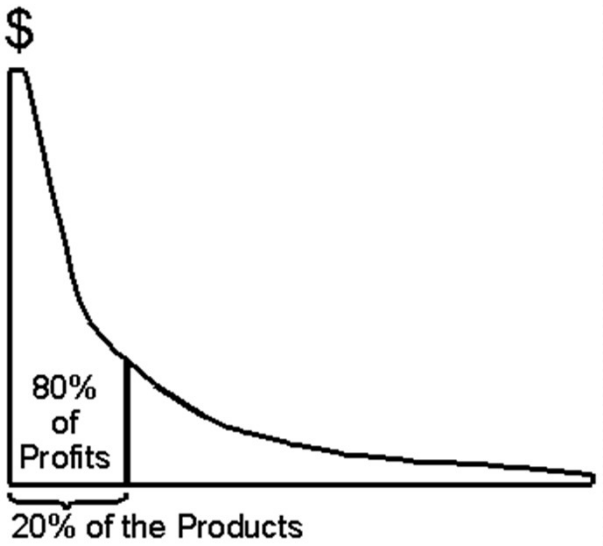
What is UML?

- UML (Unified Modeling Language)
  - Non proprietary standard for modeling software systems, OMG
  - Convergence of notations used in object-oriented methods
    - OMT (James Rumbaugh and collegues)
    - Booch (Grady Booch)
    - OOSE (Ivar Jacobson)
- Current Version: UML 2.5
  - Information at the OMG portal http://www.uml.org/
- Commercial tools: Rational – RSA (IBM),Together (Borland), Visual Architect (business processes, BCD)
- Open Source tools: ArgoUML, Omondo, **Papyrus**

UML: First Pass

- You can solve 80% of the modeling problems by using 20 % UML
- We teach you those 20%
- 80-20 rule: Pareto principle

Vilfredo Pareto, 1848-1923
Introduced the concept of Pareto Efficiency,
Founder of the field of microeconomics.



UML First Pass

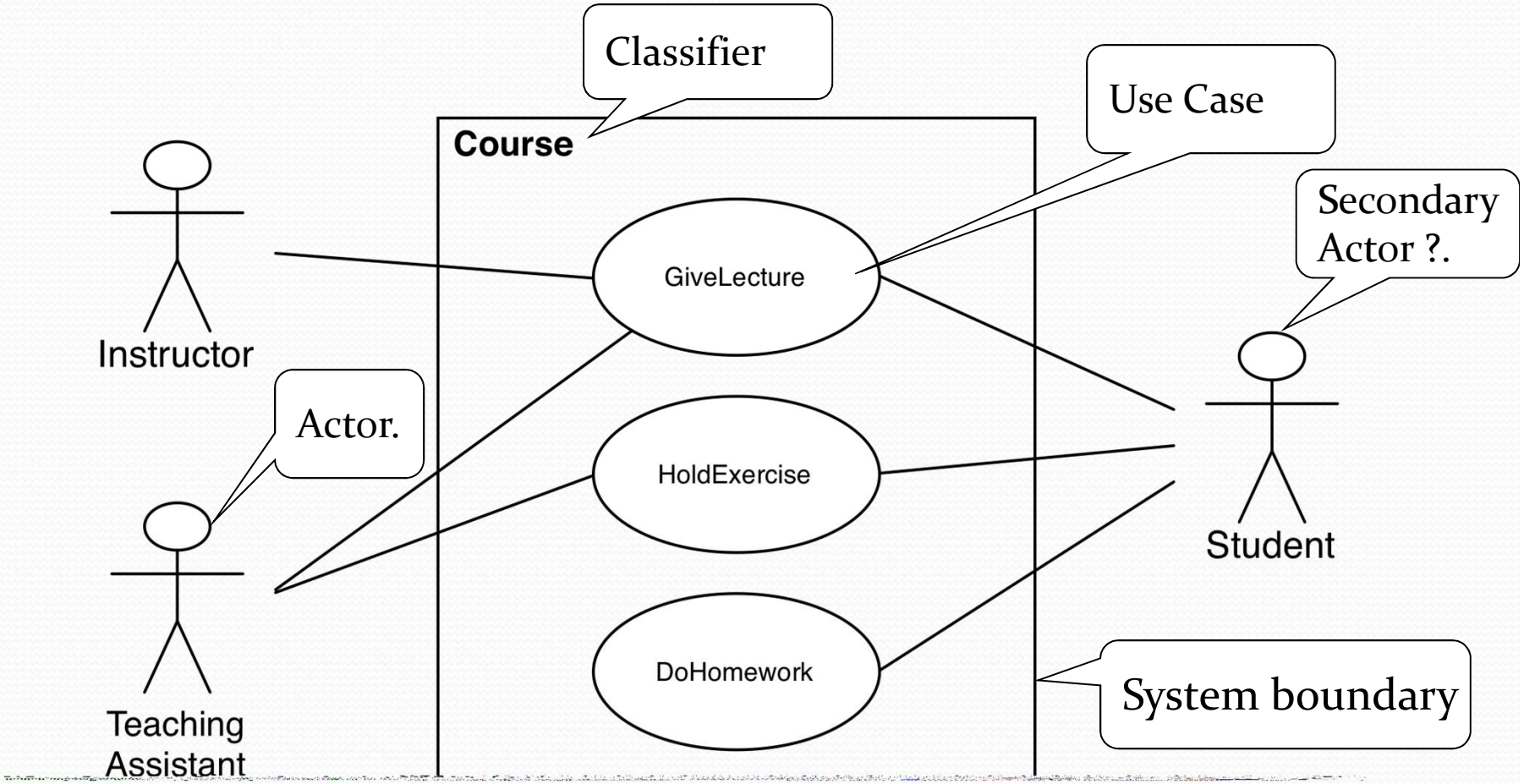
- Use case diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Objects, attributes, associations
- Sequence diagrams
  - Describe the dynamic behavior between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object
- Activity diagrams
  - Describe the dynamic behavior of a system, in particular the workflow.

UML Core Conventions

- All UML Diagrams denote graphs of nodes and edges
  - Nodes are entities and drawn as rectangles or ovals
  - Rectangles denote classes or instances
  - Ovals denote functions

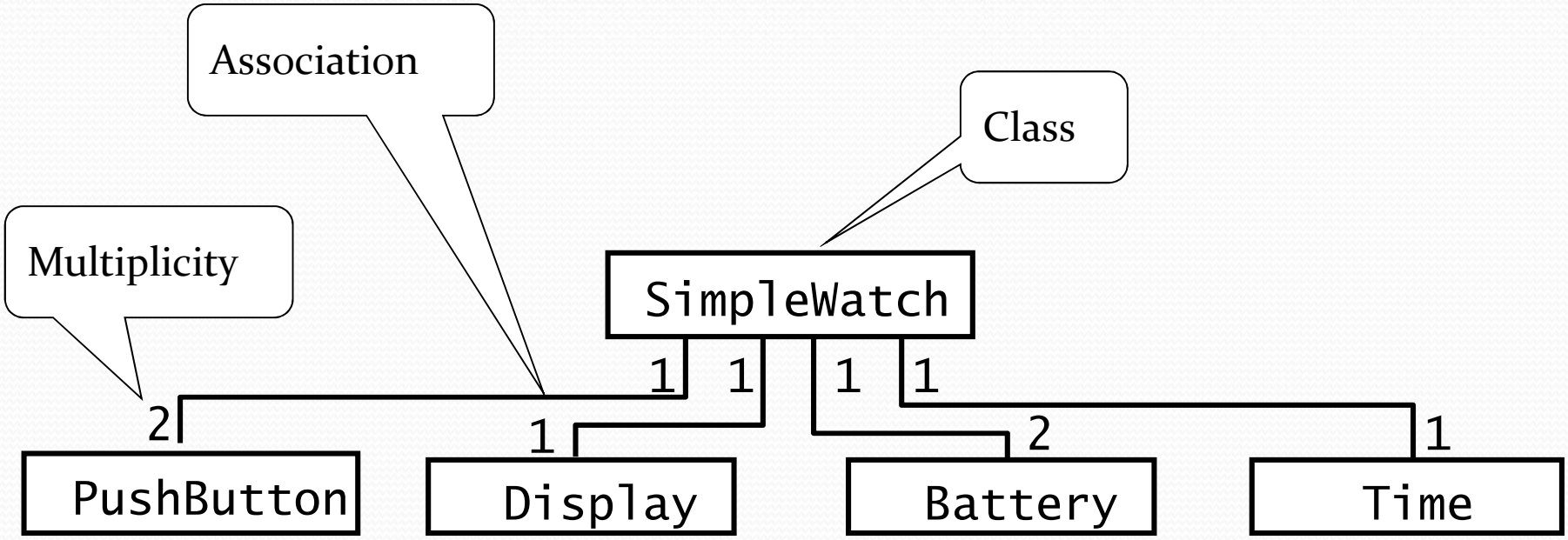
- Names of Classes are not underlined
  - `SimpleWatch`
  - `Firefighter`
- Names of Instances are underlined
  - `myWatch:SimpleWatch`
  - `Joe:Firefighter`
- An edge between two nodes denotes a relationship between the corresponding entities

UML first pass: Use case diagrams



Use case diagrams represent the functionality of the system from user's point of view

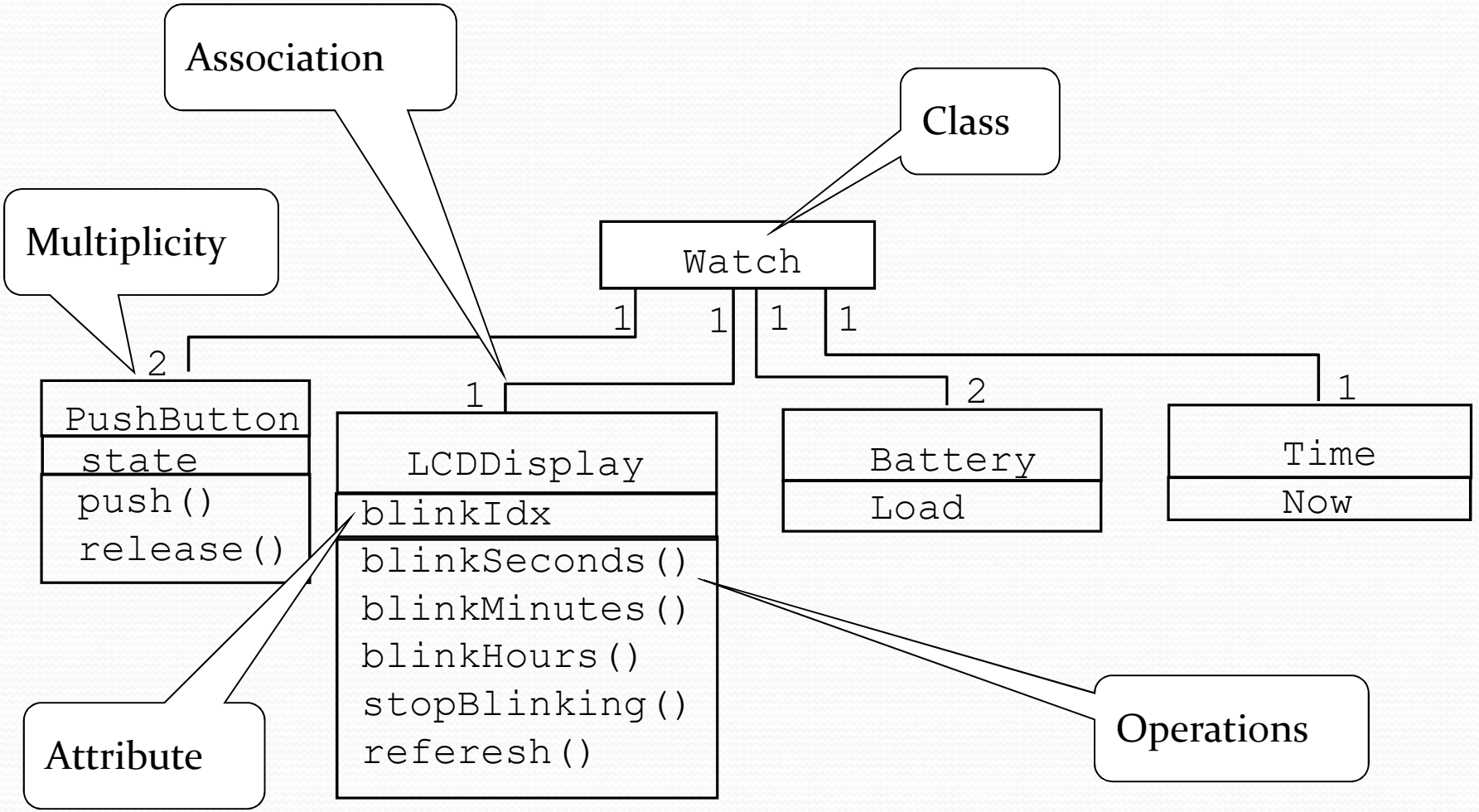
UML first pass: Class diagrams



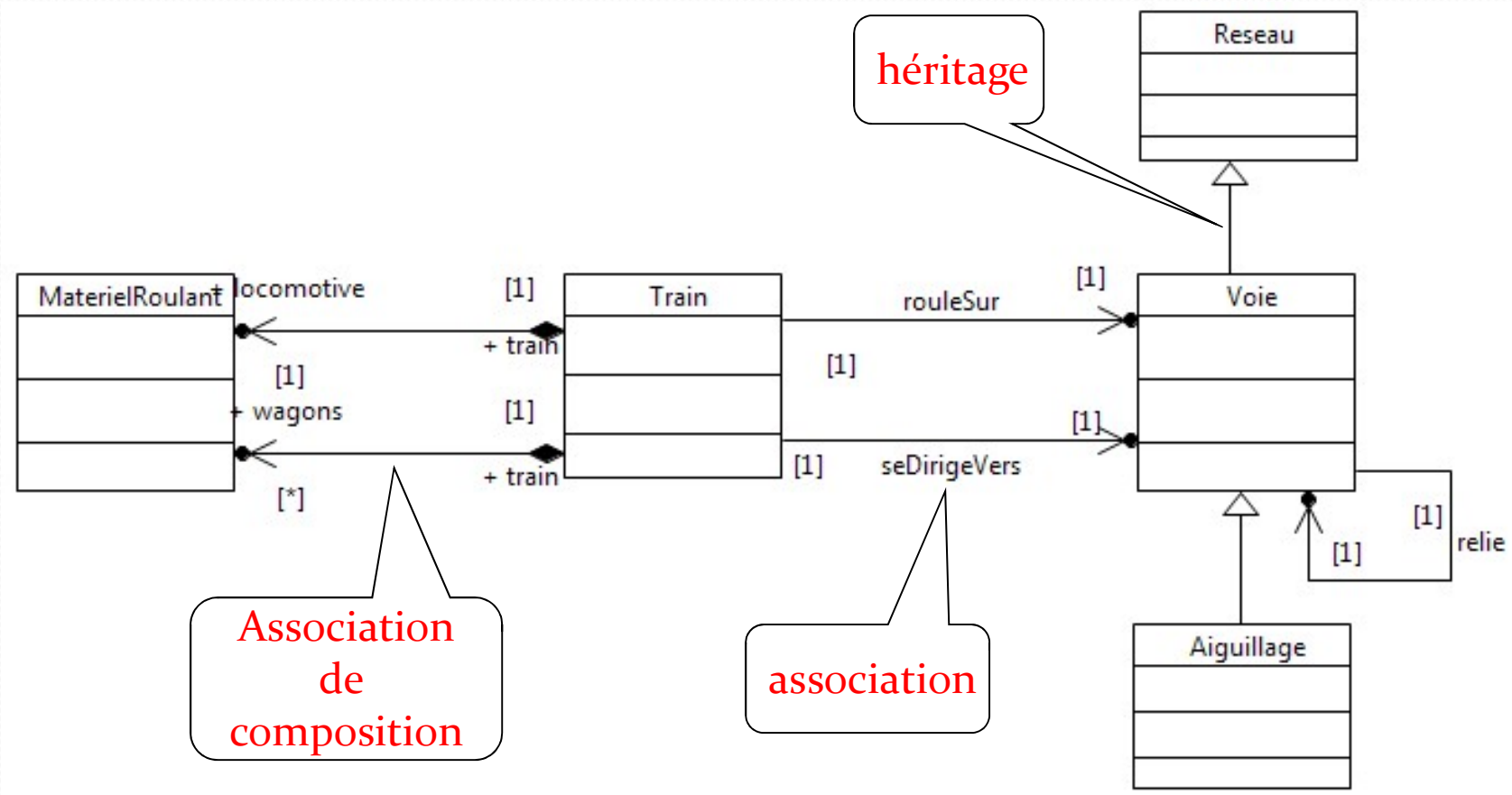
Class diagrams represent the structure of the system

UML first pass: Class diagrams

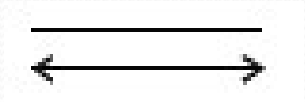
Class diagrams represent the structure of the system



Suivez la flèche !



Relations entre classes/ liens entre objets



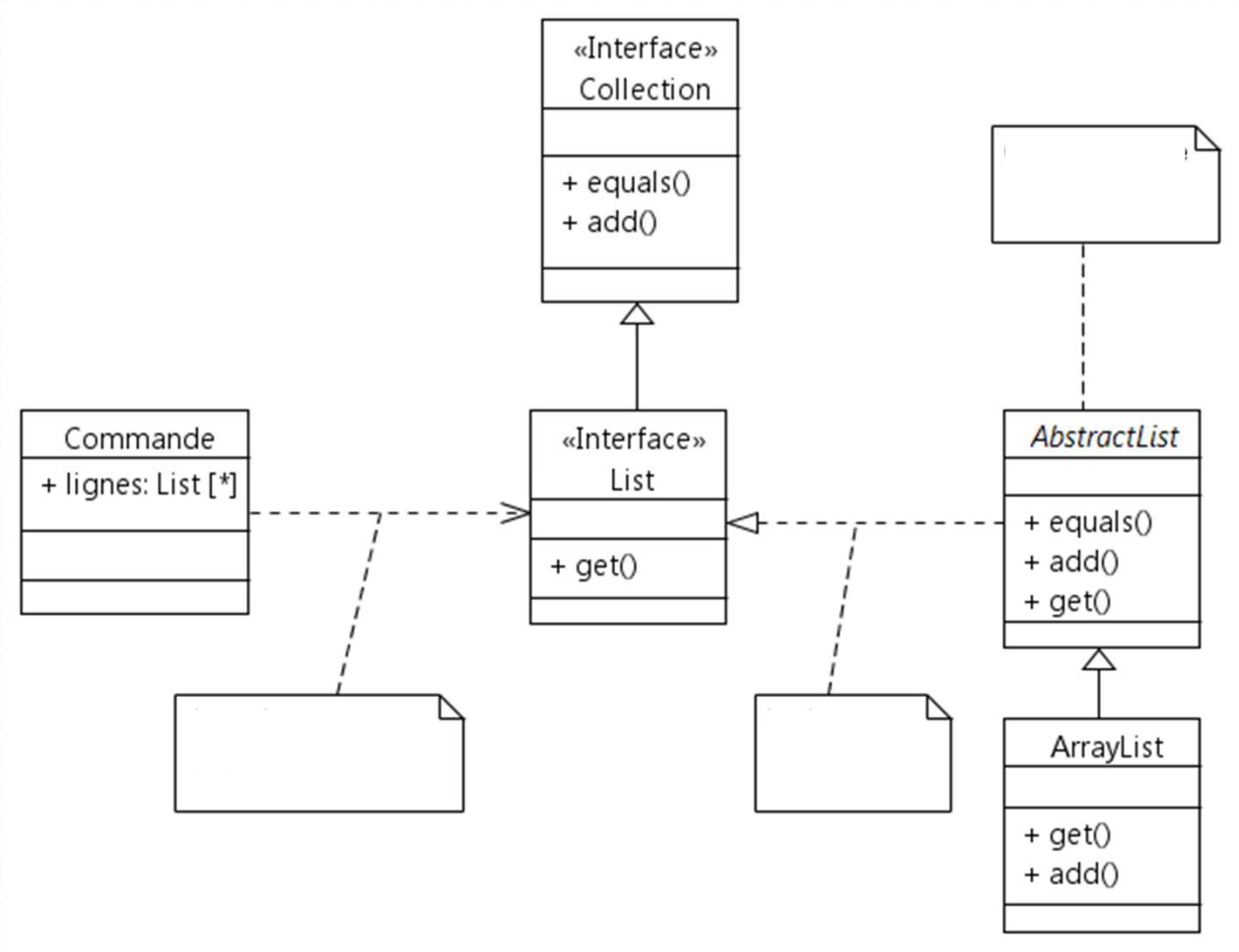
- Association
  - les instances des classes sont liées
  - possibilité de communication entre objets
  - relation forte : composition

- Généralisation/spécialisation
  - les instances de la sous-classe sont des instances de la super-classe (niveau conceptuel)
  - héritage (niveau implémentation)

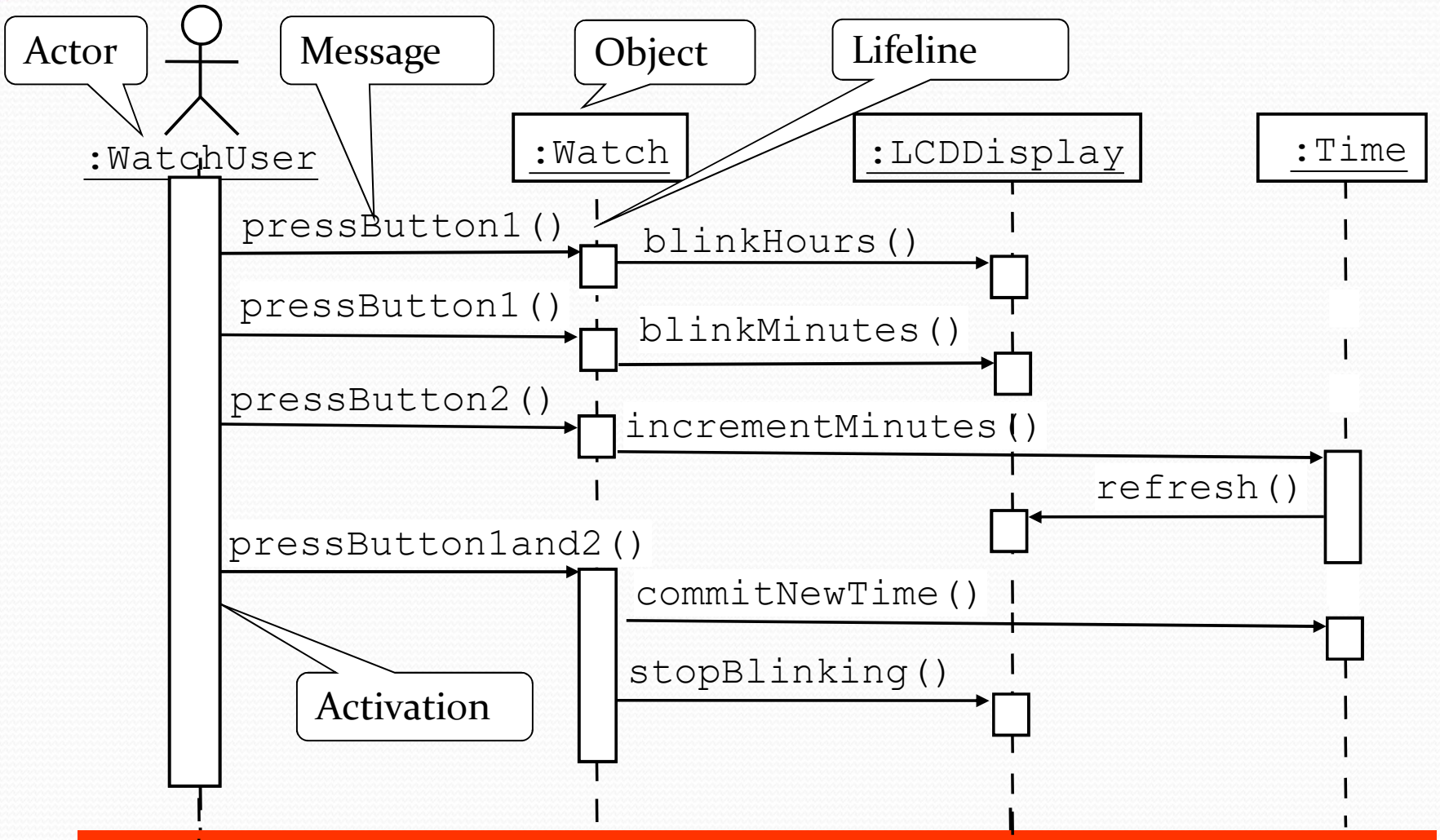
- Dépendance
  - la modification d'une classe peut avoir des conséquences sur une autre

- Réalisation
  - une classe réalise une interface

Interfaces et classes abstraites

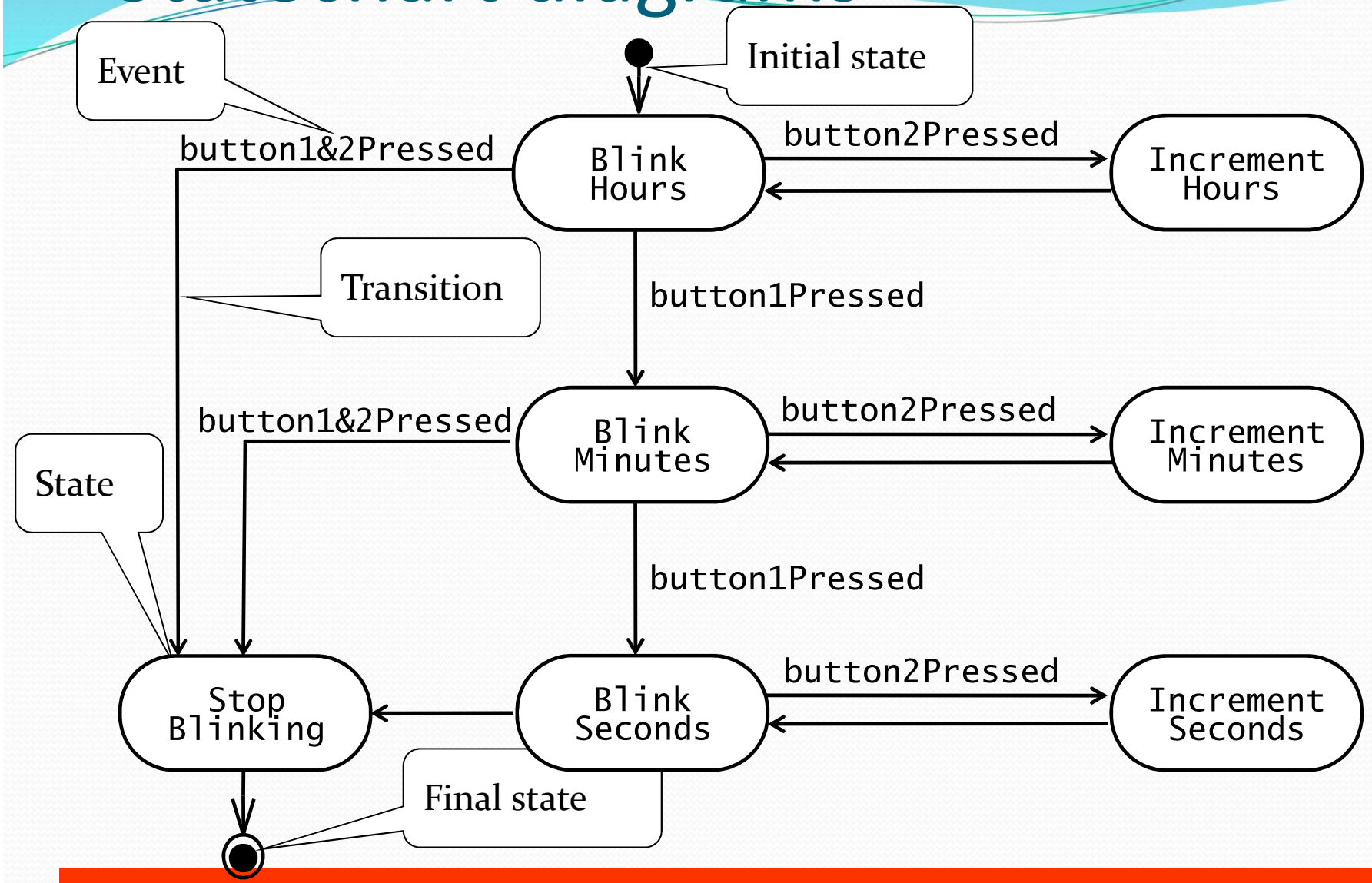


UML first pass: Sequence diagram



Sequence diagrams represent the behavior of a system as messages ("interactions") between *different objects*

Statechart diagrams



Represent behavior of *a single object* with interesting dynamic behavior.

Other UML Notations

UML provides many other notations, for example

- Deployment diagrams for modeling configurations
  - Useful for testing and for release management
- We introduce these and other notations as we go along in the lectures
  - OCL: A language for constraining UML models.

What should be done first? Coding or Modeling?

- It depends….
- Forward Engineering
  - Creation of code from a model
  - Start with modeling
  - Greenfield projects
- Reverse Engineering
  - Creation of a model from existing code
  - Interface or reengineering projects
- Roundtrip Engineering
  - Move constantly between forward and reverse engineering
  - Reengineering projects
  - Useful when requirements, technology and schedule are changing frequently.

UML Basic Notation Summary

- UML provides a wide variety of notations for modeling many aspects of software systems
- Today we concentrated on a few notations:
  - Functional model: Use case diagram
  - Object model: Class diagram
  - Dynamic model: Sequence diagrams, statechart.