ADPROC

Lecture 3

Unified Modelling Language (UML)

Objectives

- UML Notation and Representation
  - Use Case Diagrams
  - Class Diagrams
  - Interaction Diagrams
  - Statechart Diagrams
  - Activity Diagrams

Announcement: No lecture next week. Labs as normal.

Introduction

Sources:

- M. Fowler and K. Scott, *UML Distilled(3rd Ed.)*, Addison-Wesley, 2004, ISBN: 0-321-19368-7.
- D. Tagarden, A. Dennis, & B. Wixom, *System Analysis and Design with UML (4th Ed.)*, Wiley, 2013, ISBN: 978111809236-1.
- www.uml.org/#Links-General
- www.uml.org/#Links-Tutorials
- www.softdocwiz.com/UML.htm - UML Dictionary
- http://argouml.tigris.org/ - free UML software

- The Unified Modeling Language (UML) was developed by Grady Booch, Jim Rumbaugh and Ivar Jacobson in early 90's as a response to the need of representing visually the increasing complexity of systems in a clear and concise way.

Introduction

- The UML is now a standard for OO design and development (originally intended to support large C++ projects). Provides notation in the form of diagrams for modeling and documenting a broad range of systems, activities, and processes.
- A *model* can be considered as an abstraction of a system (problem), that consists of *objects* interacting by sending messages.
  - The *domain* is the actual world from which the system (problem) originates.
  - Objects have features and characteristics (*attributes*) and things they can do (*behaviour, functionality, service, etc.*).
  - The values of an object's attributes determine its *state*.

Modeling Concepts

Systems, Models and Views

- A system can be considered as an organized set of interacting objects
- To deal with the complexity we use decomposition ("Divide at impera") to divide it into subsystems
- Modeling (constructing abstractions) is another way of dealing with system's complexity
- Modeling means focusing on the important aspects of the system
- A ***model*** is an abstraction of a system and it consists of ***objects*** that interact each other
- Generally, more than one model is used to describe complex systems
- A **System model** is a set of all models built during the development
- A **view** depicts selected aspects of a model in order to visualize and make them understandable

Primitive and Abstract Data Types

Primitive data types (built-in – *byte, int, long, double, float, …*):

- set of values
- set of operations (unary, binary, …)
- way of representing (e.g. *int* in 32 bits)

Complex systems can not be represented with such types.
User-defined types – classes – encapsulate data (features, characteristics, state, structure) and functionality (operations, behaviour, actions, services, etc.).

- When solving a real problem, in the analysis phase of system development process, we specify what is the essential information for the problem and the system.
- Extracting this essential information involves creating ***abstractions***, or models for entities in the real world.
- The essential characteristics (data) of the entities are captured as *attributes (data fields)*, and the essential functionalities (operations, actions, behaviour) of those attributes are captured as *methods*.

UML Models

- **System development includes three different models for which UML uses different notations:**
  - **functional models**
    - ***use case diagrams*** **(*actors*** **and** ***use cases*****)**
  - **object models**
    - **class and instance diagrams, which comprise objects (attributes and functionality) and associations (relationships), used to describe the system structure;**
  - **dynamic models**
    - **interaction diagrams – sequence of *messages*, exchanged between *the objects* to describe behaviour;**
    - **statechart diagrams – describe dynamic behaviour of an *individual object* as a number of *states and transitions;***
    - **activity diagrams – represent a *set of operations* that describe system's behaviour.**

UML Notation and Representation

- **Use Case Diagrams**
  - to represent functionality of a system during the elicitation and analysis stages (*what* rather then *how*)
  - to describe what a system does from the standpoint of an external observer
  - it is a set of *scenarios* tied together for a single task or goal
  - a *scenario* is a sequence of steps describing what happens when someone interacts with the system

- *An Actor* represents (mistranslation from Swedish for *role*):
  - the roles that people (objects) play in the system;
  - different type of users (or systems) that interact with the system.

UML Notation and Representation

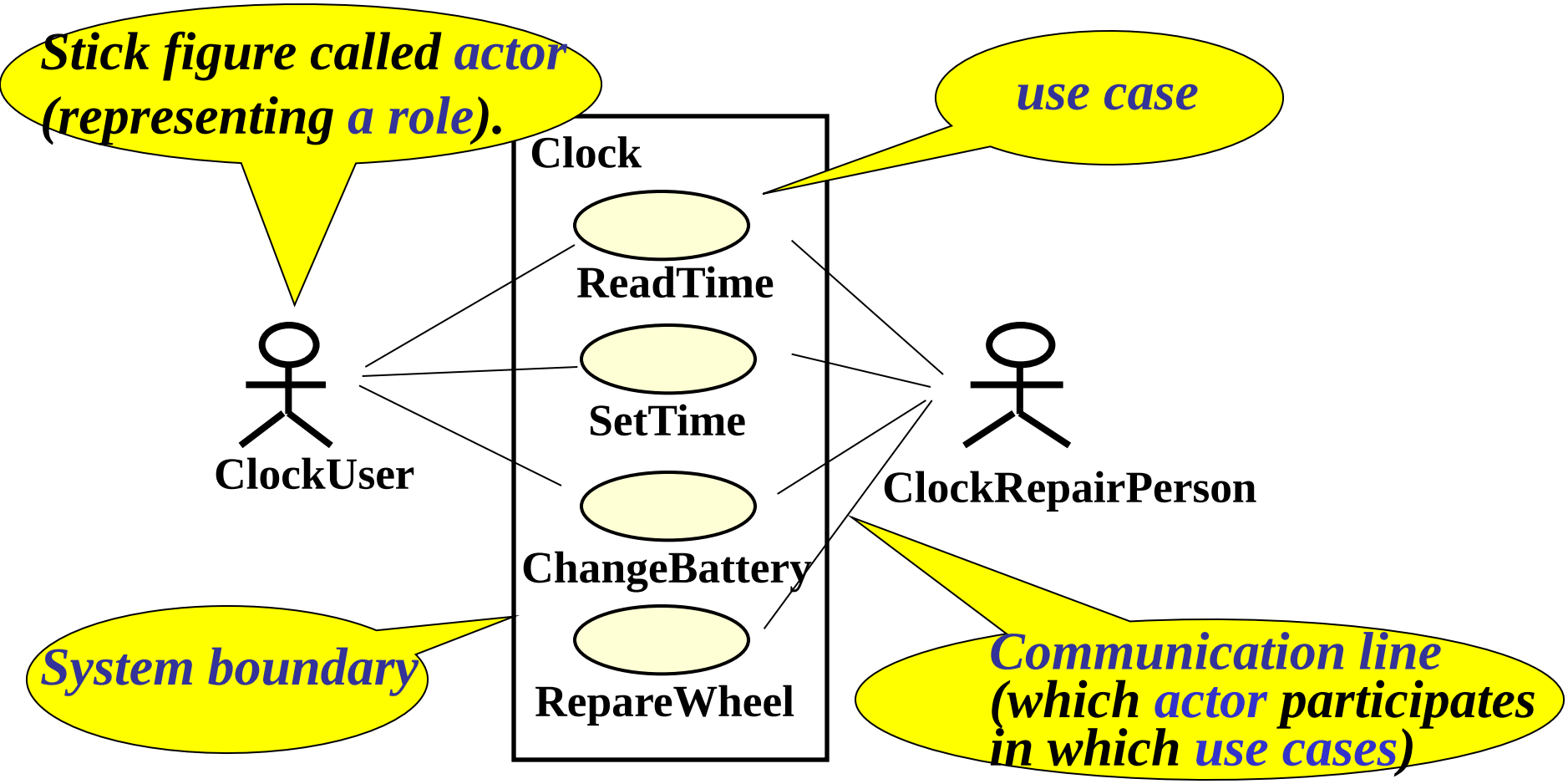

Fig.1. A simple Clock functionality, described with UML use case diagram.

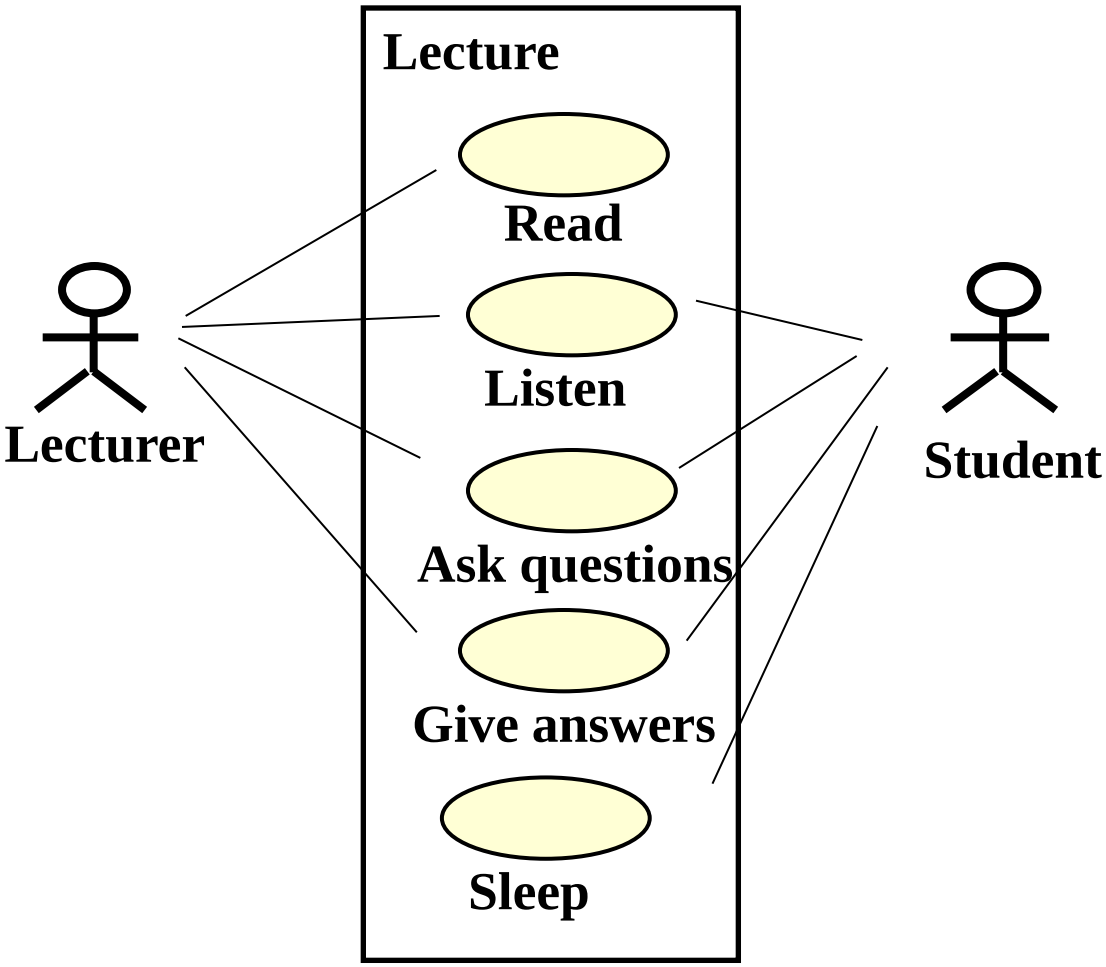

Fig.1a. A simple *Lecture* UML use case diagram.

Class diagrams

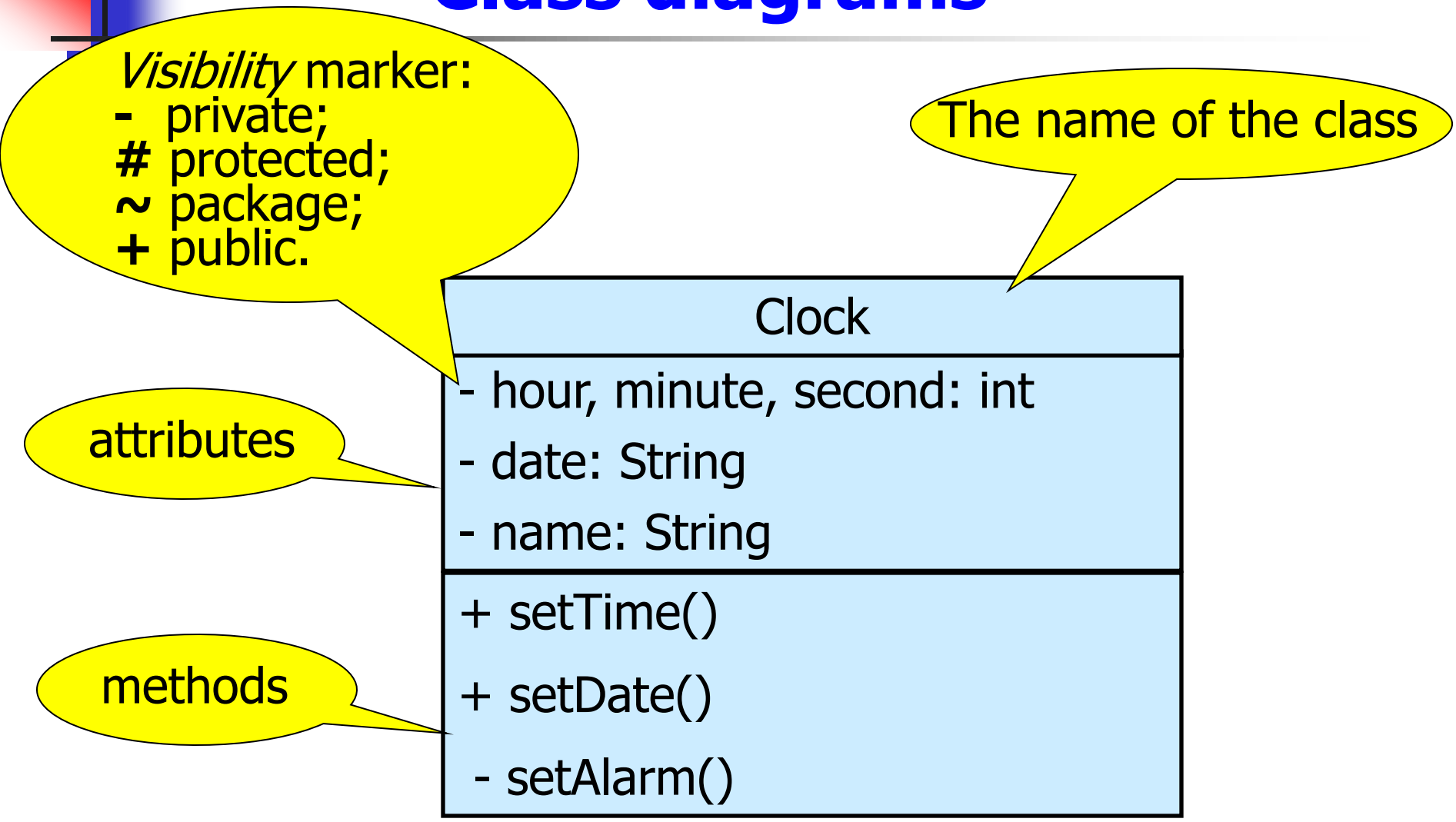

Fig. 2. A simple UML class diagram

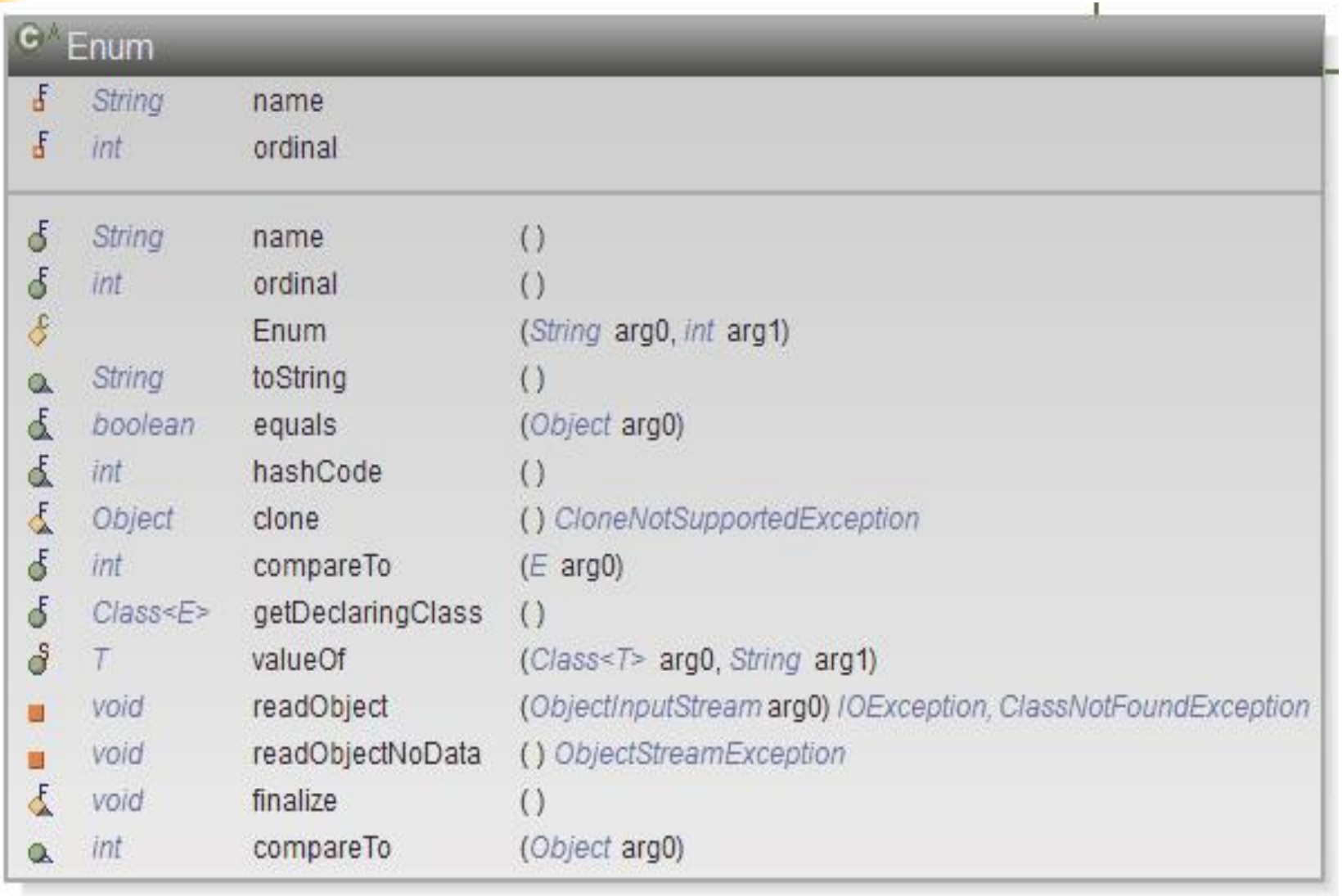

Fig. 2a. A UML class diagram (indicated by Eclipse symbols, exceptions thrown by a method are shown after the parameter list).
www.agilej.com/object-oriented/uml-class-diagram.html

Class diagrams

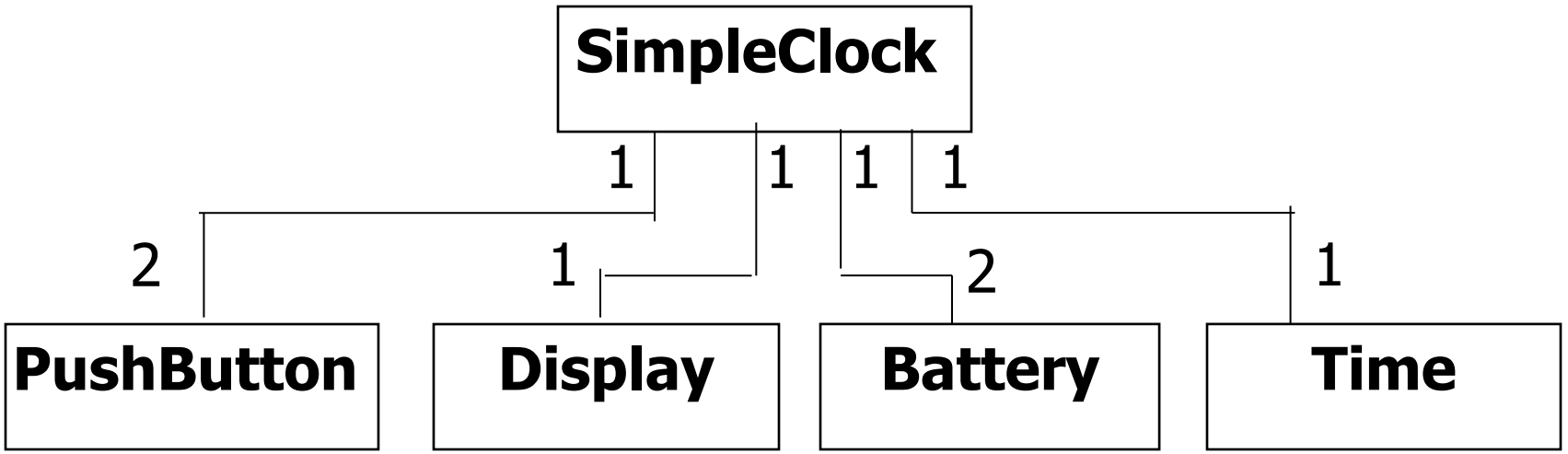

Fig.3. A class diagram of the *SimpleClock* class and its associations with other classes.

Class diagrams

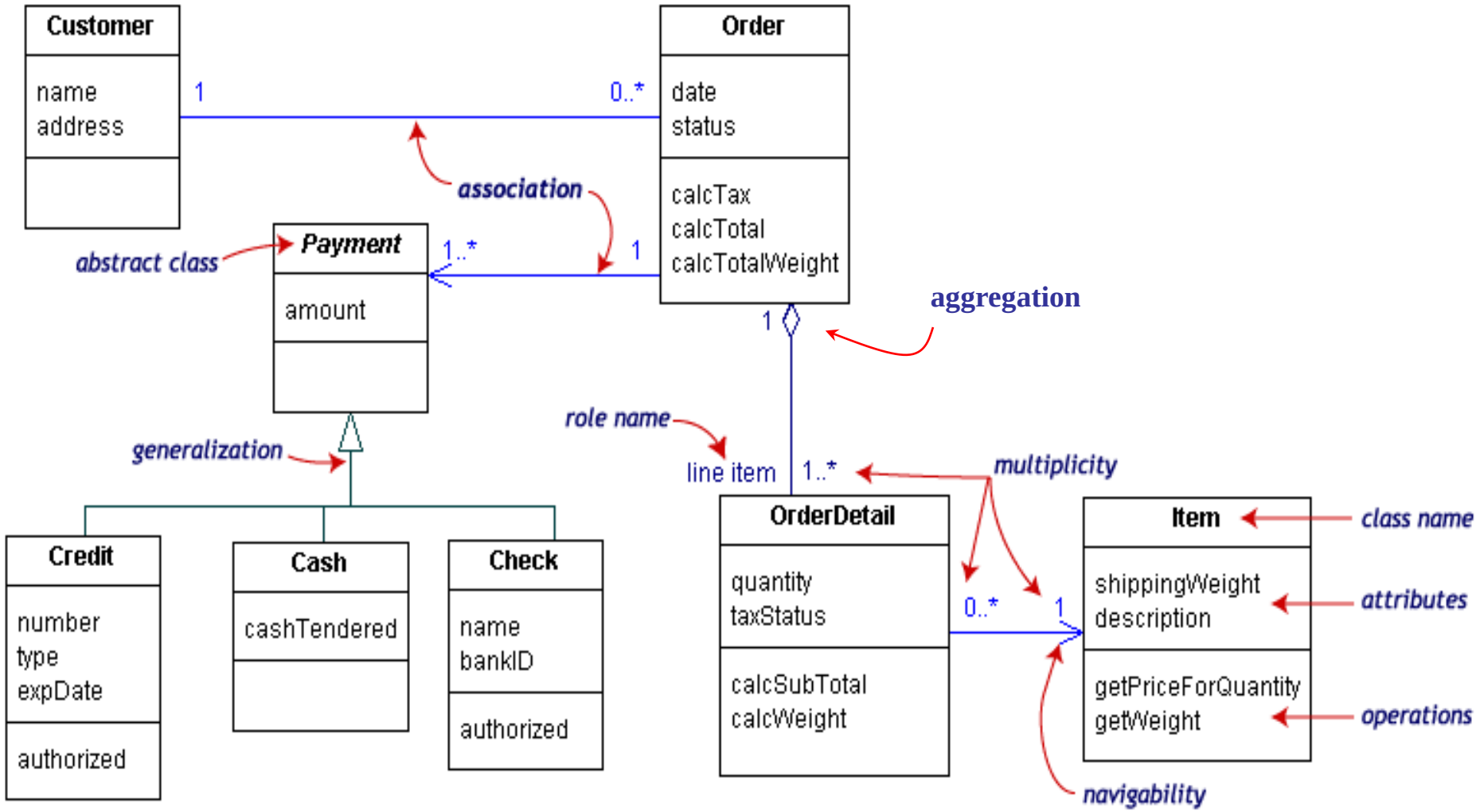

Fig. 4. A class diagram modelling a customer order from a retail catalogue (//bdn.borland.com/article/0,1410,31863,00.html#use-case-diagram).

Class diagrams

Types of relationship shown in fig. 4:

- **association** - a relationship between instances of the two classes, shown as a solid line between the classes (could be directed).
- **aggregation** - a part-of relationship, in which one class belongs to a collection. Shown as solid line with a diamond end pointing to the part containing the whole.
- **generalization** - an inheritance relationship, shown as a solid line with a triangle pointing to the superclass (*Payment* is an abstract class - its name is given in italics).

Class diagrams

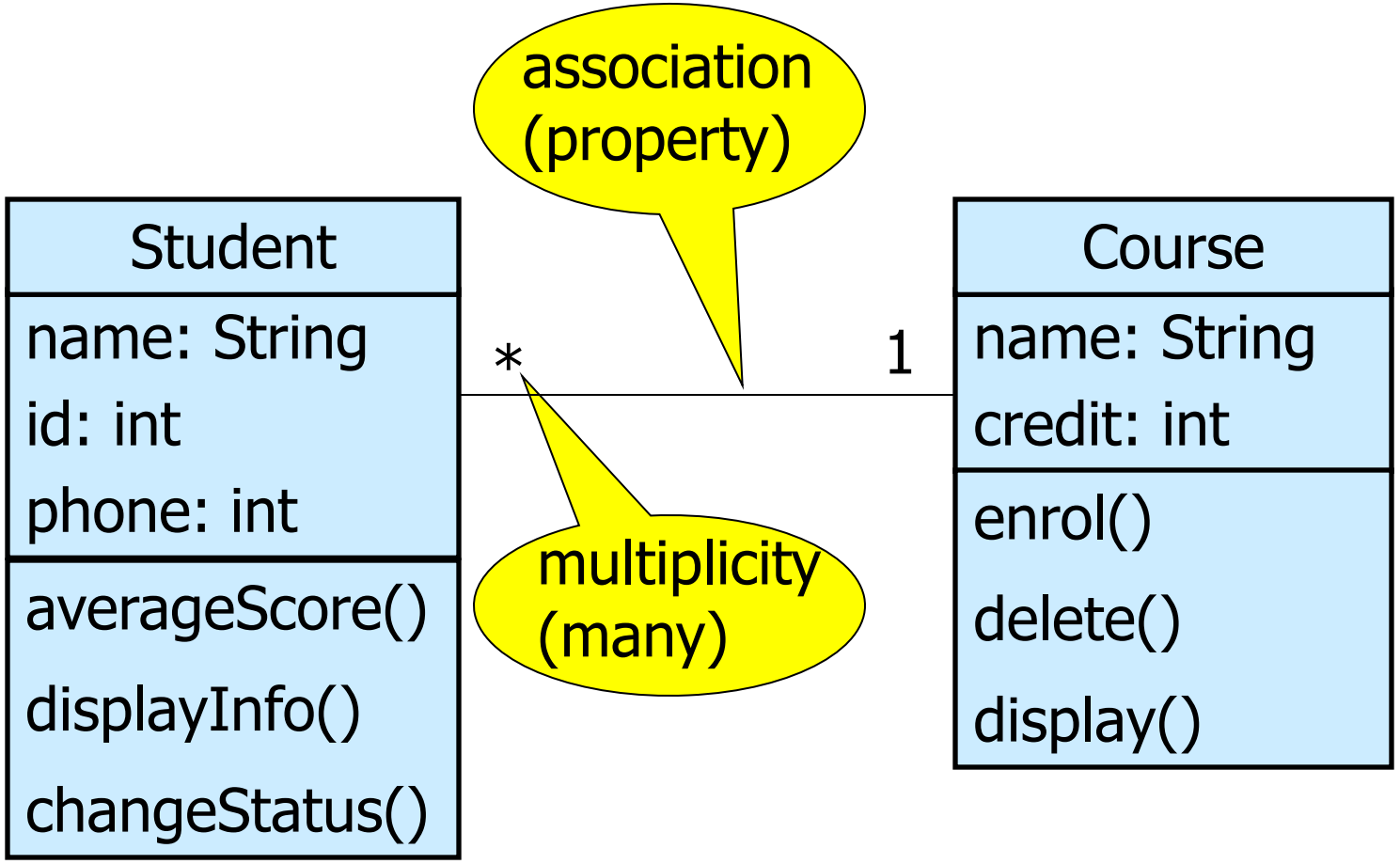

Fig. 5. A simple domain class diagram

Associations

- ***The multiplicity*** **of an association (property) describes how many objects may fill the property.**

  *** - no upper limit – zero or more (one course may have many students), (0..*)**

  1 - exactly one (a student can enrol in one and only one course), (1..1)

  0..1 - zero or one

  1..* - at least one

Instance (object) diagrams

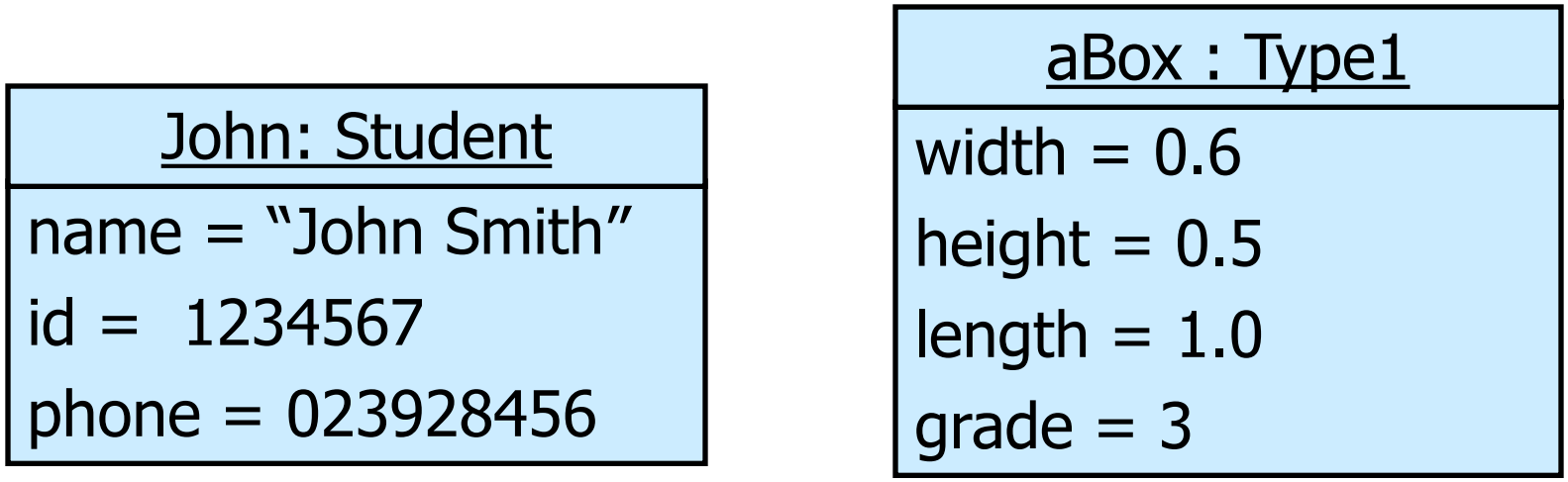

Fig. 6. Example of two instance diagrams

Interaction diagrams

- Formalized description and notation of the dynamic behaviour and collaboration of group of objects.
- Objects involved are called *participating objects*. These diagrams are used to describe their interaction.
- *Sequence* diagrams are one type of interaction diagrams (*there are others as well*).
- Within a sequence diagram, each object (the name underlined in Fig.7) is shown as a box at the top of a dashed vertical line, called *lifeline* .
- *Activation* - rectangles on the lifelines - they appear when a method is active.
- The arrows between the lifelines of the objects represent the *messages* sent between them.

Sequence diagrams

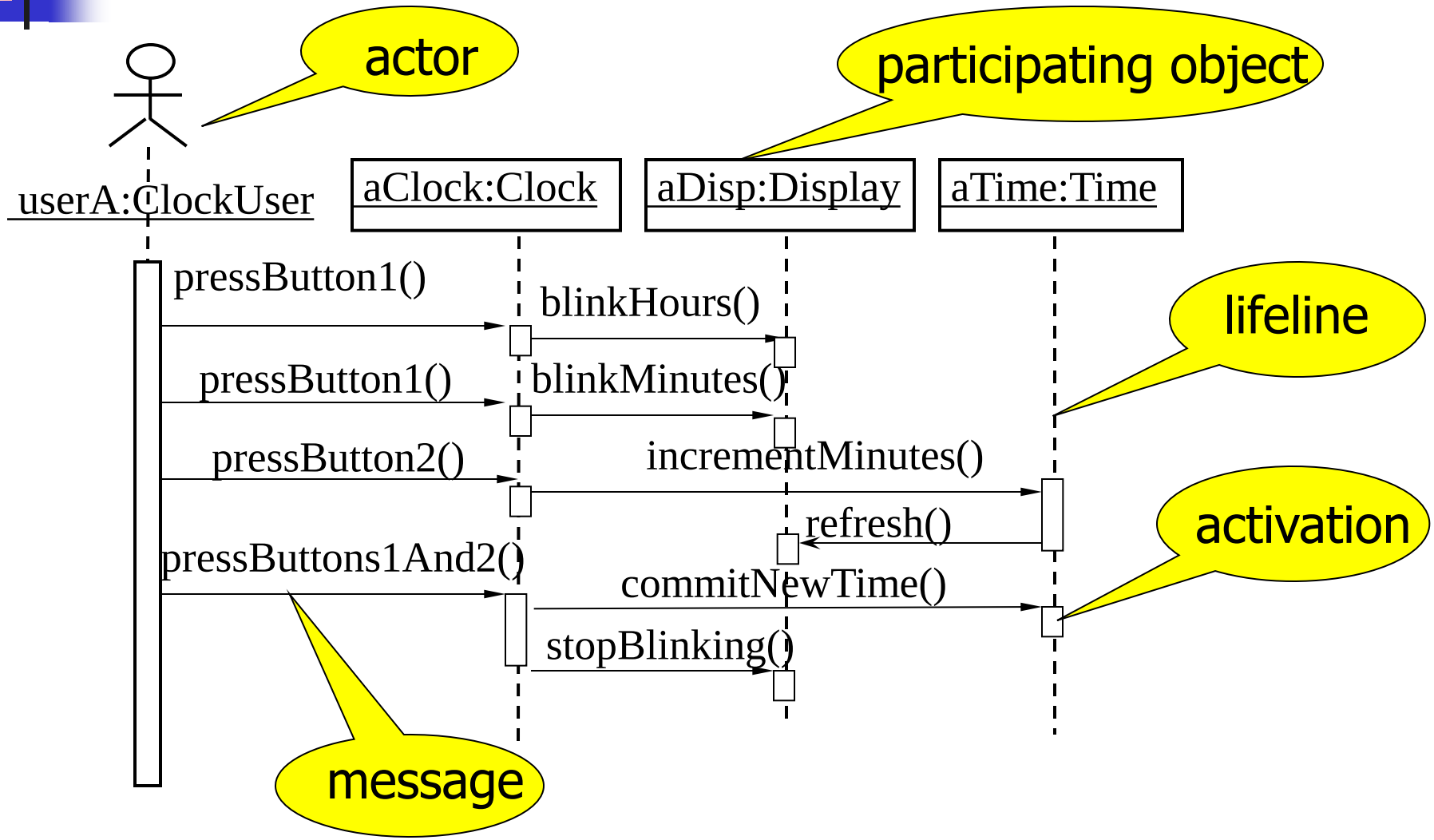

Fig. 7. Sequence diagram for *SetTime* use case scenario.

Statechart diagrams

The dynamic of a system can be described with a number of states and the transitions between them.

A ***state*** defines a particular set of attributes (data, characteristics) for an object.

A ***transition*** represents the change between the states.

A statechart diagram (Fig. 8) describes the possible states a particular object can get and the transitions between those states as a result of external events that reach this particular object.

Statechart diagrams

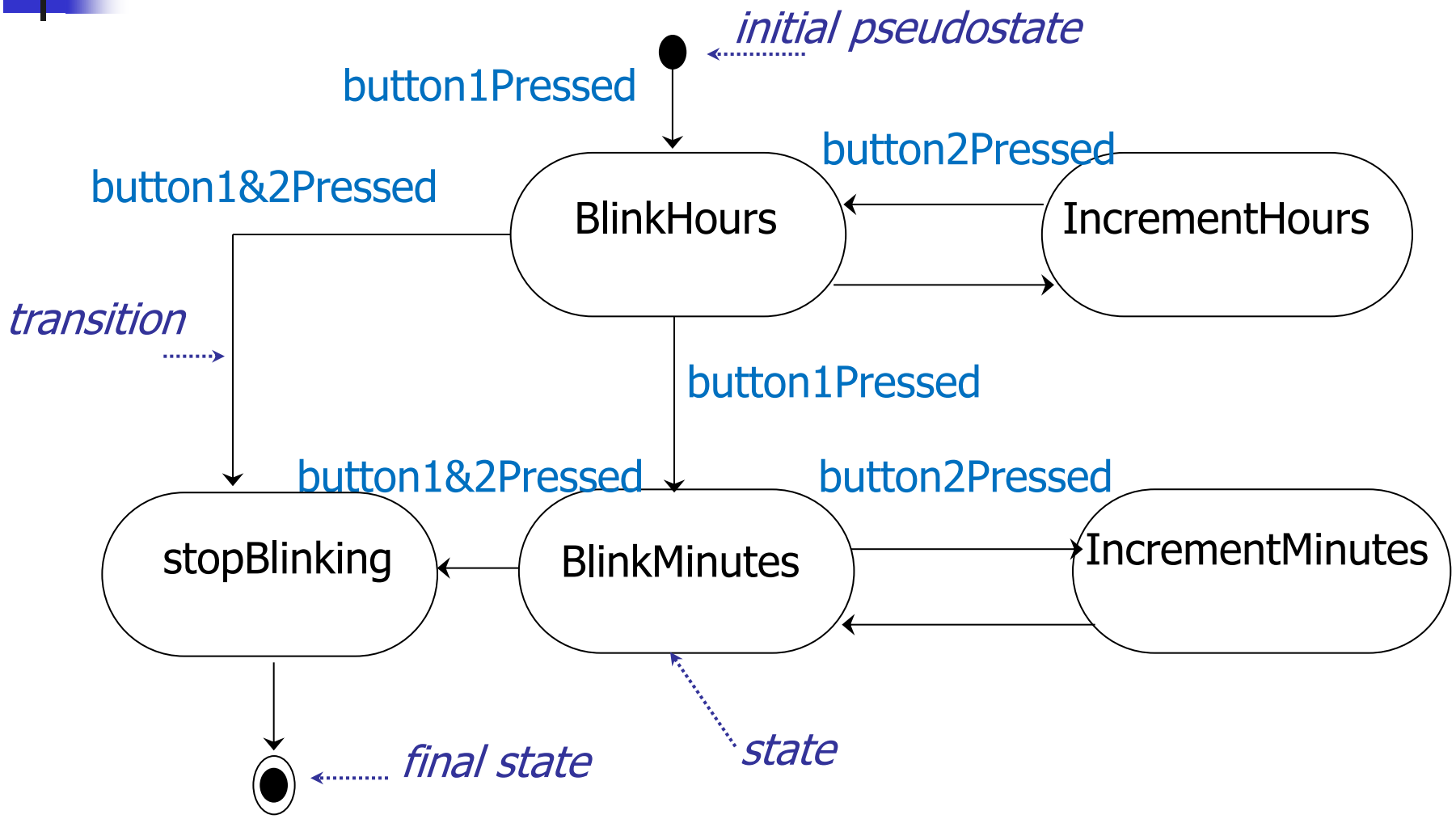

Fig. 8. A statechart diagram for the *SetTime* use case.