Systems Concepts and Modeling
Chapter 2 – Lecture 1

Objectives

- Context models
- Interaction models
- Structural models
- Behavioral models
- Model-driven engineering

System Modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling now usually means representing a system using some kind of graphical notation based on diagram types in the Unified Modeling Language (UML).
- Models are used during:
  - the requirements engineering process to help derive the detailed requirements for a system
  - the design process to describe the system to engineers implementing the system and
  - after implementation to document the system’s structure and operation

A System: General Properties

- Made up of components, both physical and conceptual
- Receives inputs and transforms these into outputs
- Exists within an environment (collection of hardware and software tools used to build software system).
- Boundary divides things inside the system from things outside
- Exhibits behavior (working/functionality)
- Fulfils some specific purpose which varies according to particular viewpoints

Example of a System: Company Payroll

- **Key Inputs:** employee information
- **Key outputs:** payslips, cheques, cash
- **Physical components:** people, paper computers
- **Conceptual components:** basic salary

Systems: Definition

- A system is an assembly of components, connected together in an organized way and separated from its environment by a boundary. This organized assembly has an observable purpose which is characterized by how it transforms inputs from the environment into outputs to the environment.

- A system with no inputs or outputs is closed.

Software components

- Files
- Subroutines
- Library functions
- Classes
- Packages

Component dependency

A

B

- Component A depends on B
- A change to B may require a change to A
- Many types of component dependency

Modeling in Design

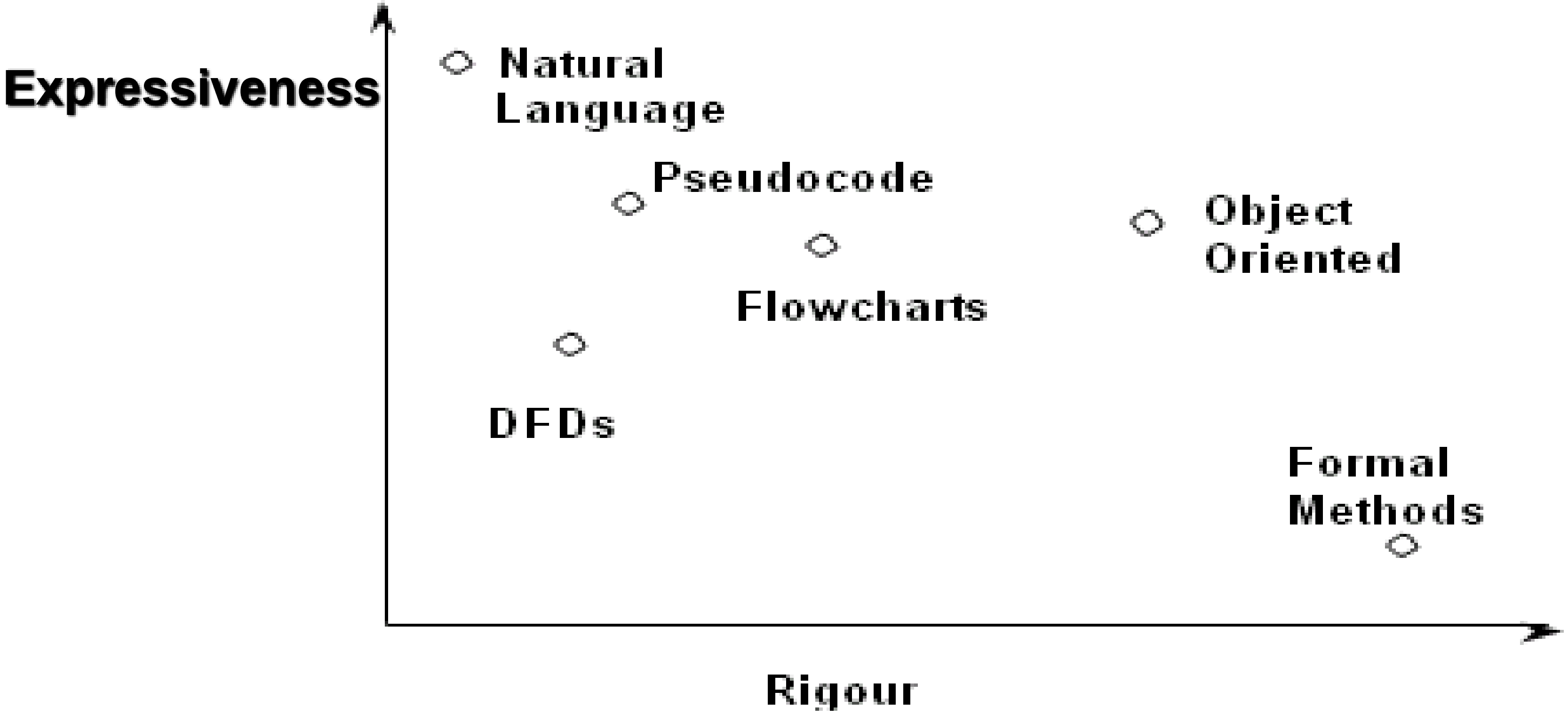
- The field of software engineering now incorporates object-oriented concepts and techniques, by which a system is viewed as a collection of self-contained objects that include both data and processes.
- Objects can be built as individual pieces and then put together to form a system, leading to modular, reusable project components.
- In 1997, the Unified Modeling Language (UML) was accepted as the standard language for object development.
- Some types of models support the analysis process
  - Class diagrams
  - Use case diagrams
  - Activity Diagrams

What is “object-oriented”?

- ❑ The object-oriented approach views a software system as a collection of self-contained objects, including both data and processes.
- ❑ Object-oriented systems focus on capturing the structure and behavior of software systems in modules (**objects**) that encompass both data and processes.
- ❑ Unified Modeling Language (UML) was accepted as the standard language for object development.
- ❑ Consequently, developers focused on building software systems more efficiently by enabling the software engineer to work with a system’s data and processes simultaneously as objects.
- ❑ The beauty of objects is that they can be reused over and over in many different systems and changed without affecting other system components.

Software Modeling Methods

- 1970s: flowcharts, data flow diagrams, Jackson structured design
- 1980s: formal methods
- 1990s: object oriented methods
- 2000s: OO methods consolidated in UML



What is UML?

- Unified Modeling Language
- Convergence of three leading OO methods:
  - OMT (James Rumbaugh)
  - OOSE (Ivar Jacobson)
  - Booch (Grady Booch)
- Reference: “The Unified Modeling Language User Guide”, Addison Wesley, 1999.
- Supported by several CASE tools (e.g Together)

UML and This Course

- You can model 80% of most problems by using about 20% UML

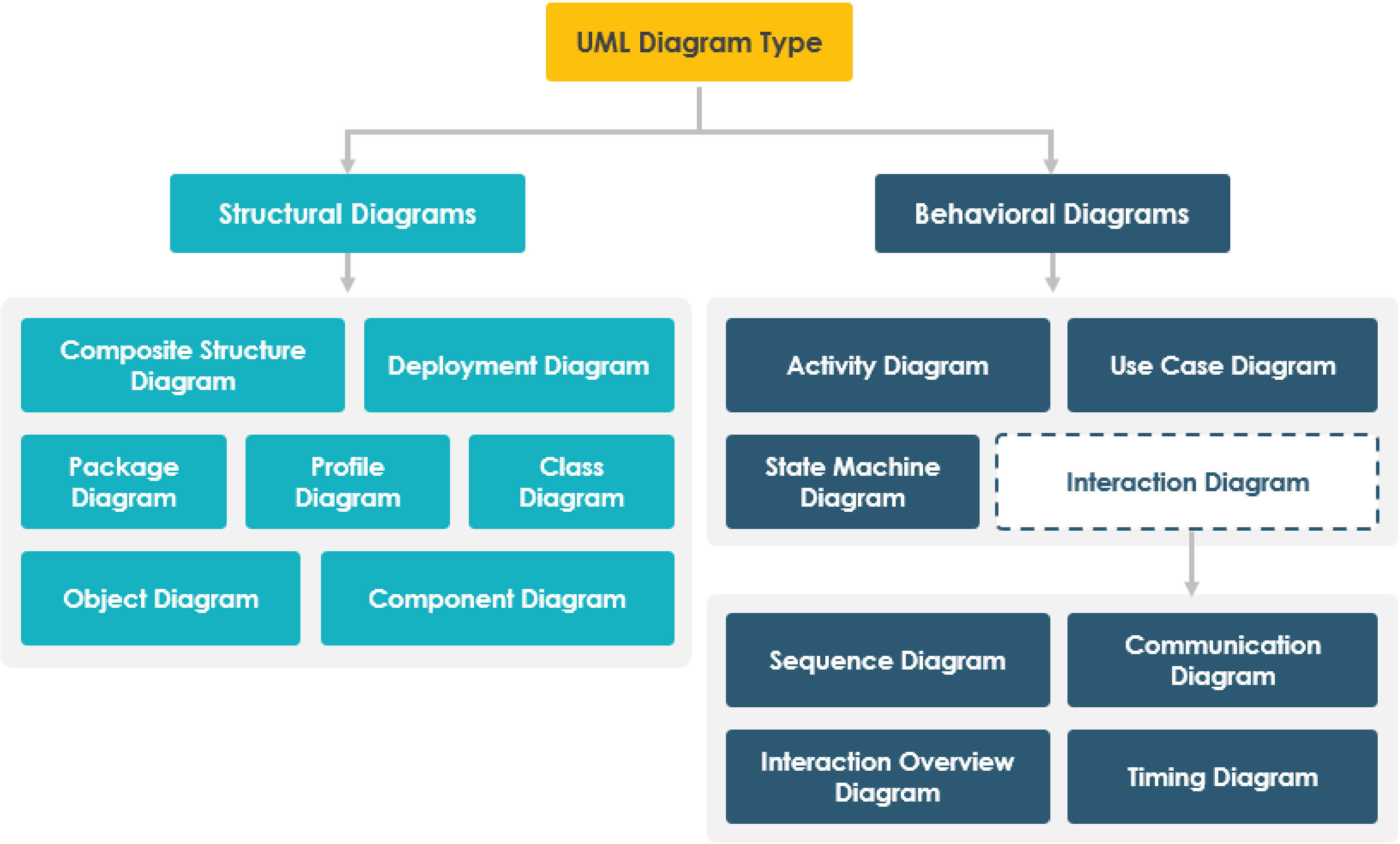
- In this course, we teach you those 20%

Benefits of an Object Approach

- ❑ Software Engineer break a complex system into small manageable components
- ❑ Work on the components individually
- ❑ Easily piece the components back together to form a system
- ❑ Modularity makes system development easier to grasp
- ❑ Modules easier to share among members of a project team
- ❑ User communication is enhanced
- ❑ Reusable pieces are formed that can be plugged into other systems efforts or used as starting points for other projects
- ❑ Save time; new projects do not have to start from scratch and learning curves are not as steep

Why UML?

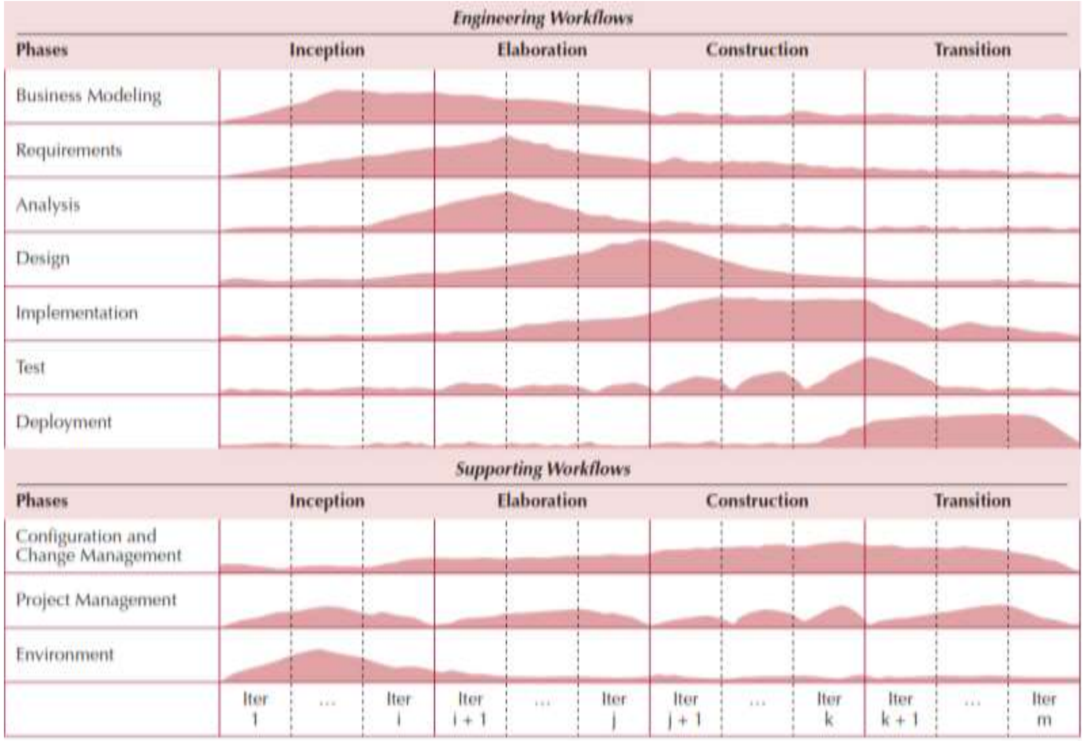
- Now the industry standard method for software engineering (design and documentation). When applied properly it makes software engineering possible ('round-trip engineering')
- All design/documentation and implementation can really be integrated

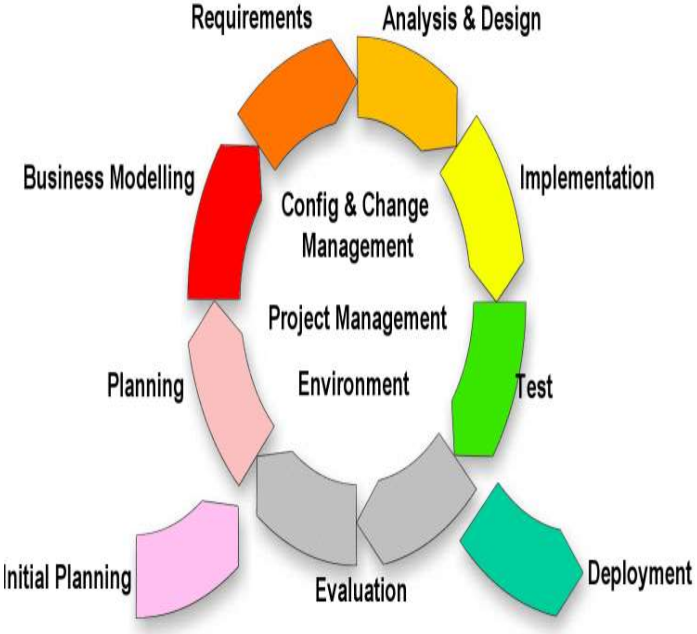


The Rational Unified Process (RUP)

- Rational Software Corporation has created a methodology called the *Rational Unified Process (RUP)* that define *how* to apply UML.
- A specific methodology that maps out when and how to use the various UML techniques for object-oriented analysis and design
- RUP is a rapid application development approach to building systems that is similar to the iterative development approach or extreme programming described in Chapter 2.
- RUP emphasizes iterative, incremental development, and prototyping.
- A two-dimensional process consisting of phases and workflows
  - Phases are time periods in development
  - Workflows are the tasks that occur in each phase

The Unified Process

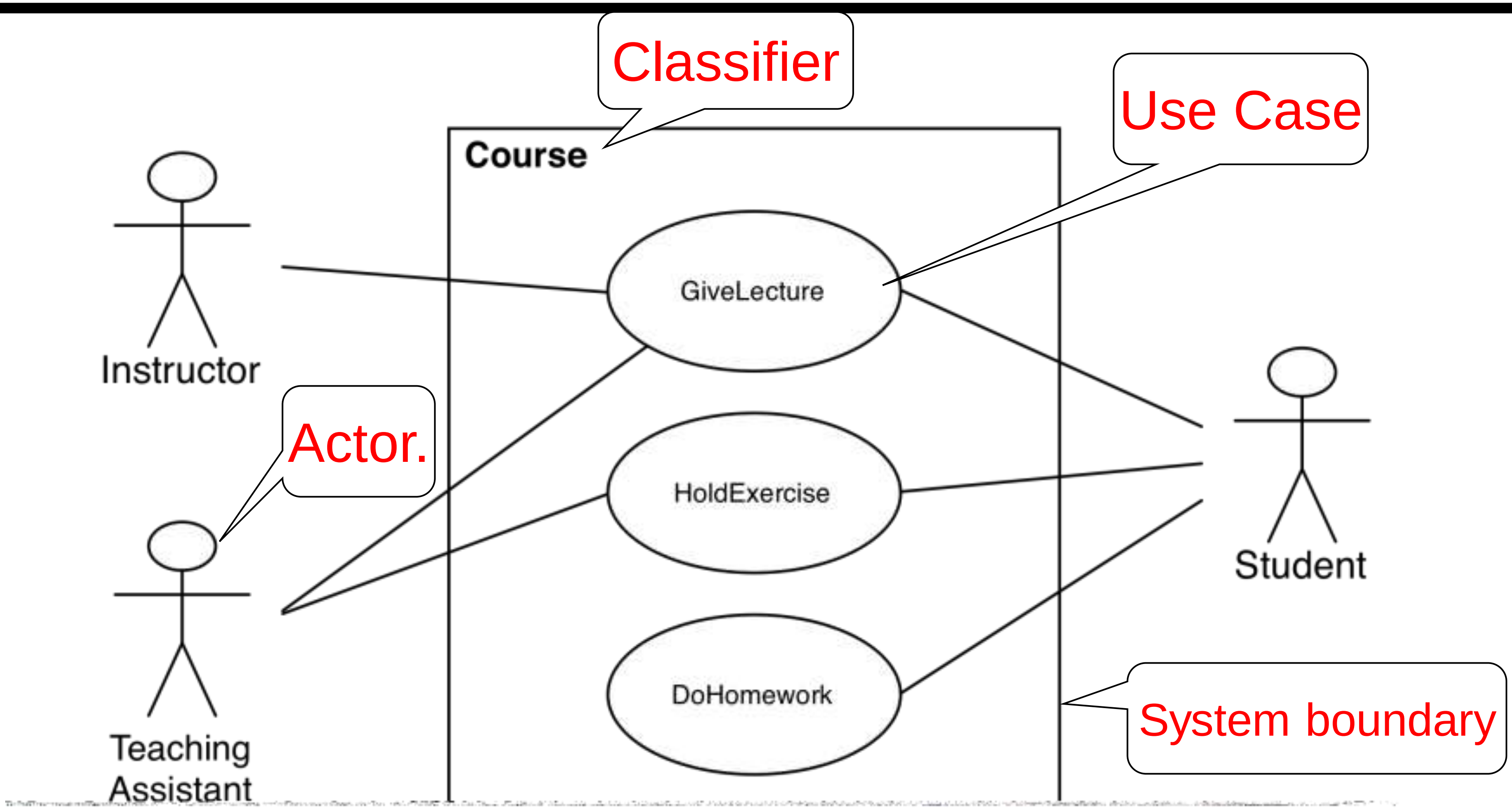




UML diagram types

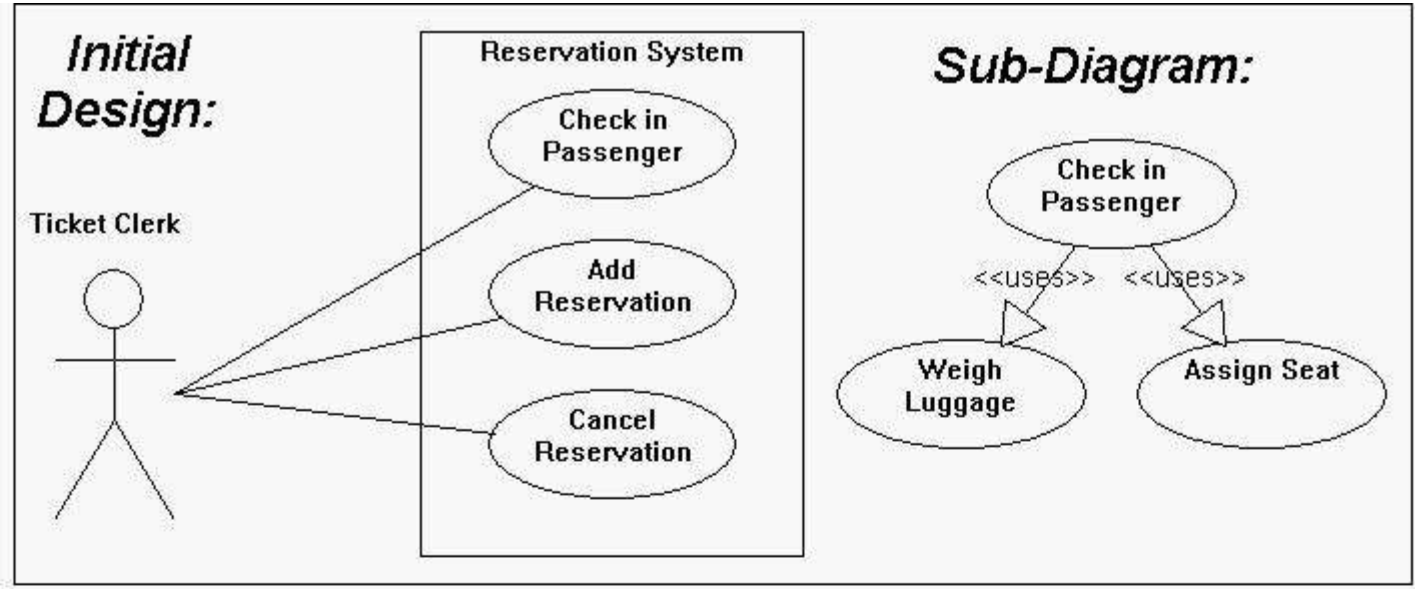
- Activity diagrams, which show the activities involved in a process or in data processing .
- Use case diagrams, which show the interactions between a system and its environment.
- Sequence diagrams, which show interactions between actors and the system and between system components.
- Class diagrams, which show the object classes in the system and the associations between these classes.
- State diagrams, which show how the system reacts to internal and external events.

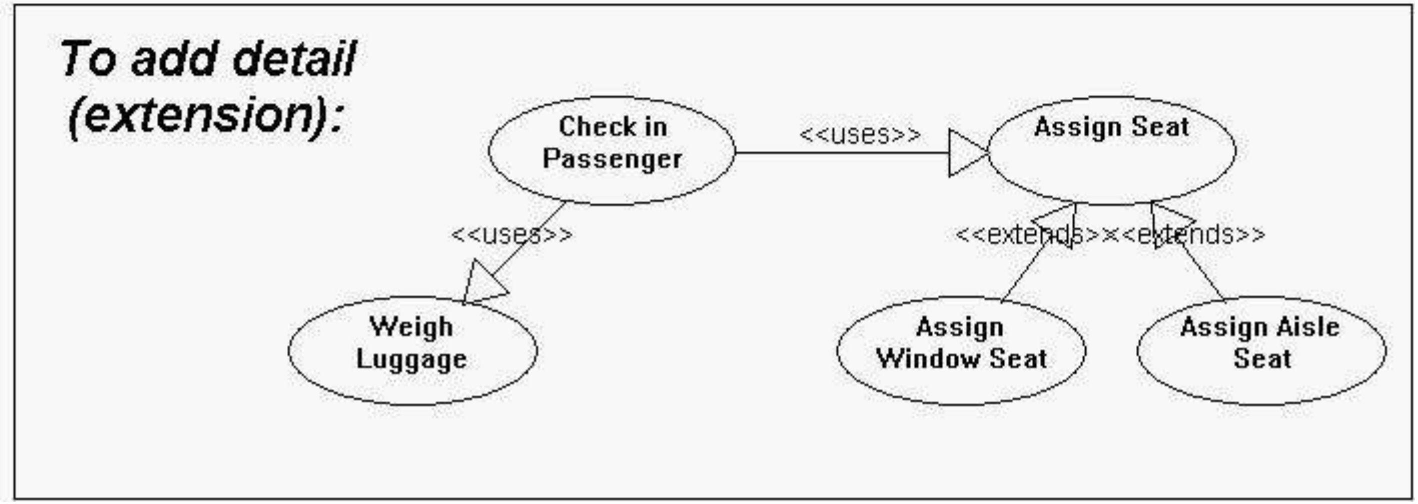
UML first pass: Use case diagram



Use case diagram represent the functionality of the system from user’s point of view

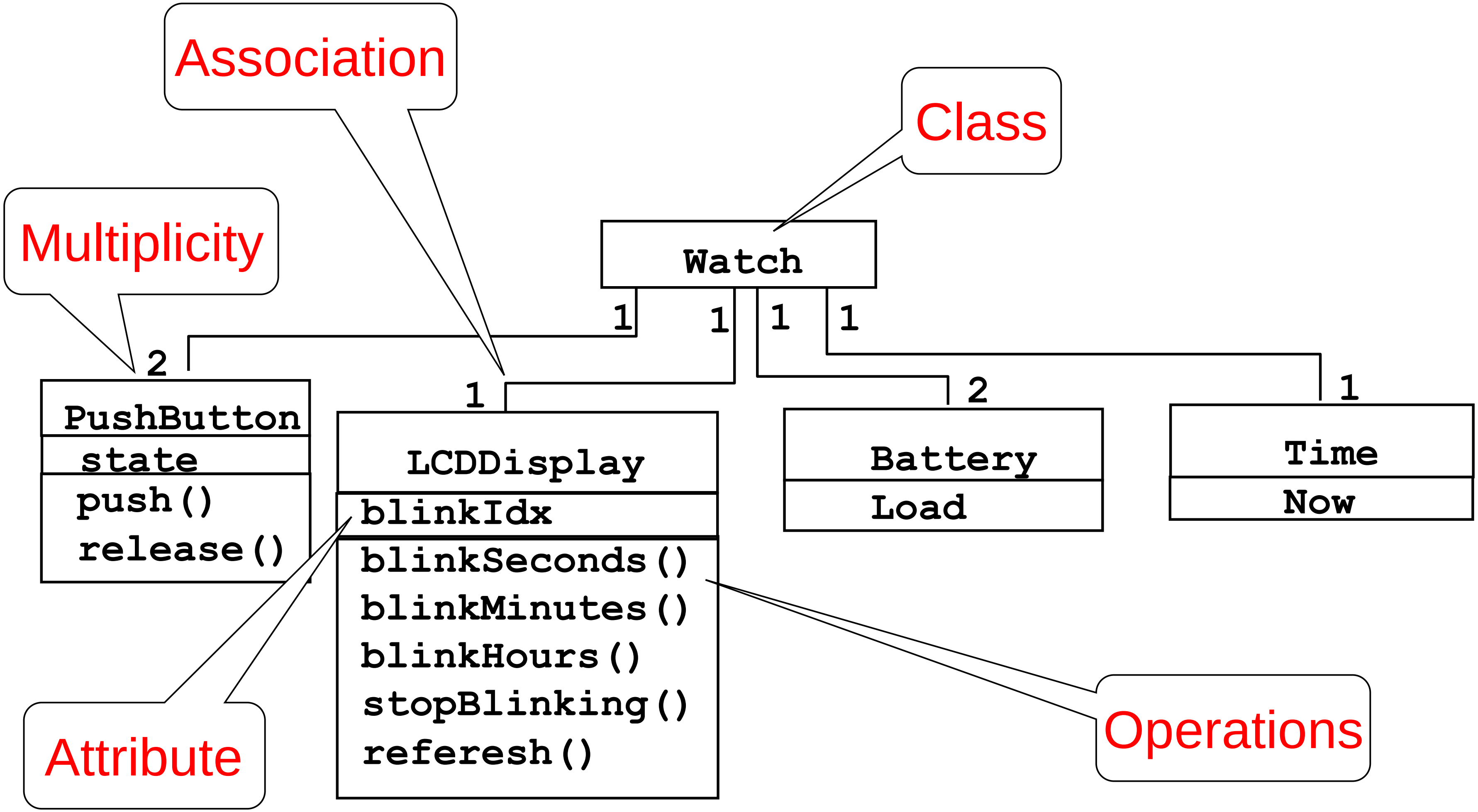
UML: Use Case Diagram





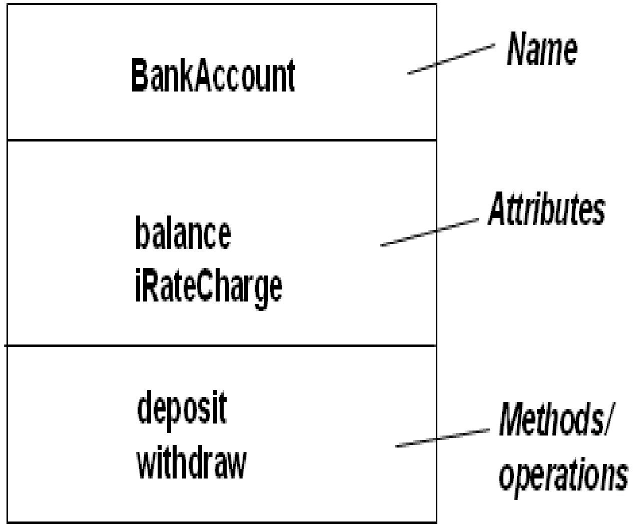
UML first pass: Class diagrams

Class diagrams represent the structure of the system



Class diagrams represent the structure of the system

UML: Class Diagram

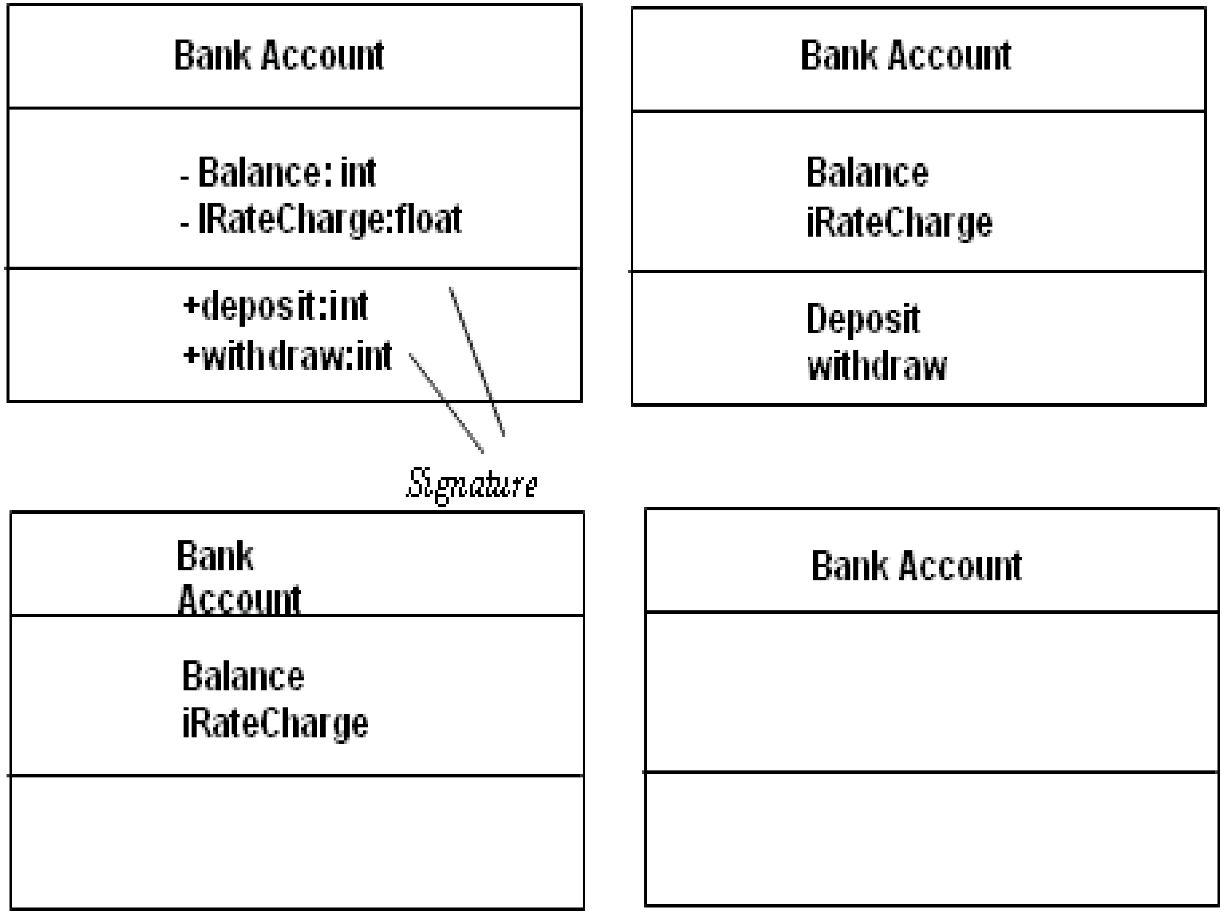


BankAccount

balance
iRateCharge

deposit
withdraw

UML Class: Different levels of detail



Bank Account

- Balance: int
- IRateCharge:float

+deposit:int
+withdraw:int

Bank Account

Balance
iRateCharge

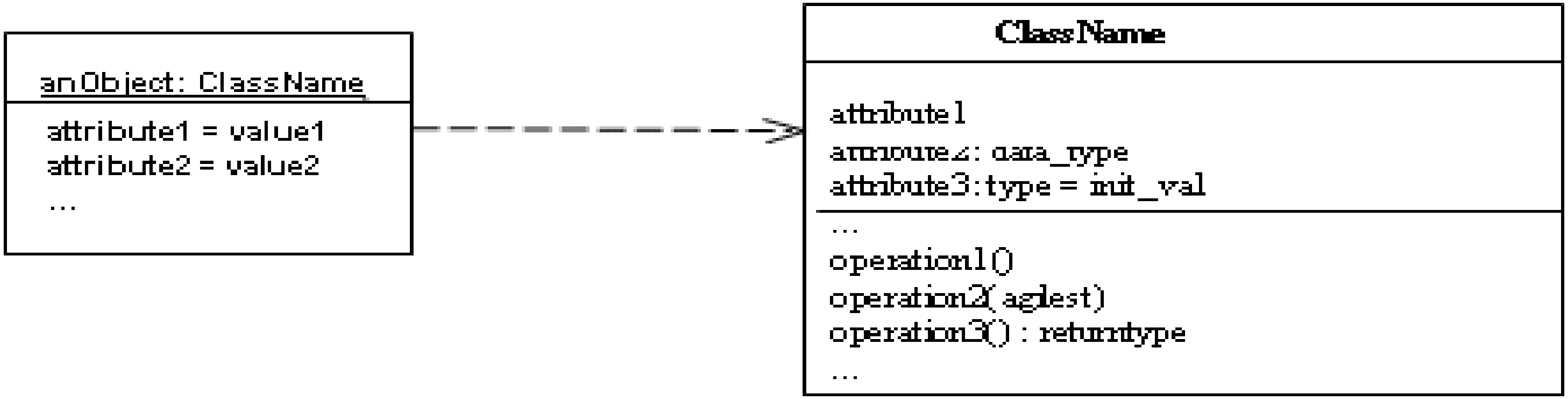
Deposit
withdraw

Bank
Account

Balance
iRateCharge

Bank Account

Objects and Classes



anObject: ClassName

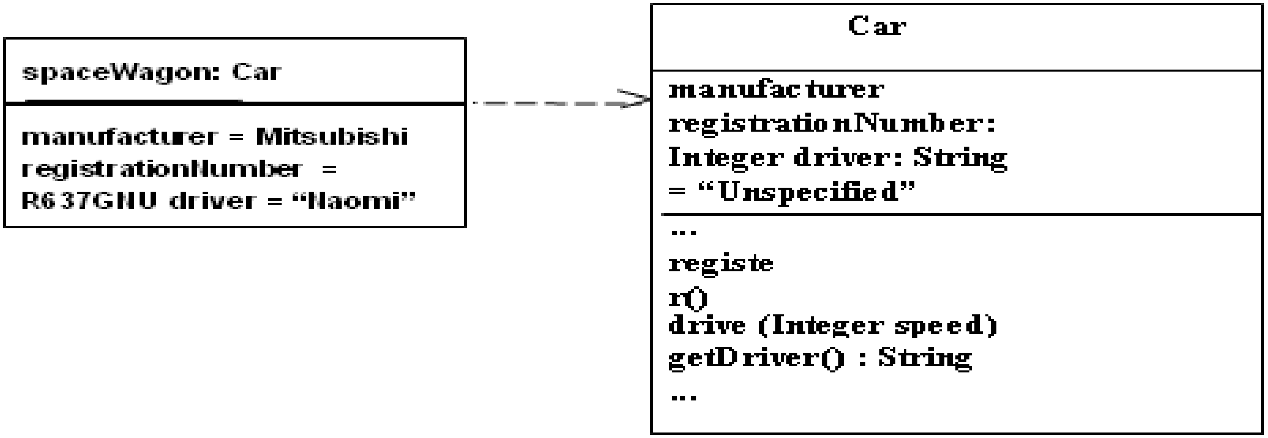
attribute1 = value1
attribute2 = value2
...

ClassName

attribute1
attribute2: data_type
attribute3: type = init_val
...
operation1()
operation2(arglist)
operation3() : returntype
...

Object (instance) .....is instantiated from .... Class

Example



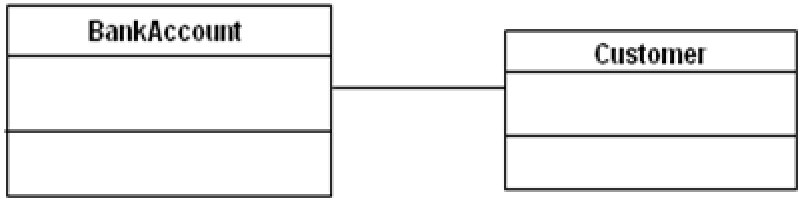
spaceWagon: Car

manufacturer = Mitsubishi
registrationNumber =
R637GNU driver = "Naomi"

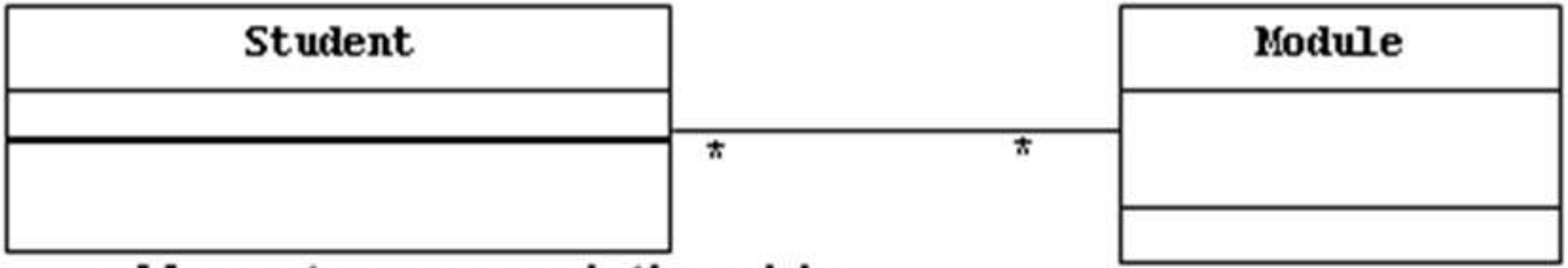
Car

manufacturer
registrationNumber:
Integer driver: String
= "Unspecified"
...
registe
r()
drive (Integer speed)
getDriver() : String
...

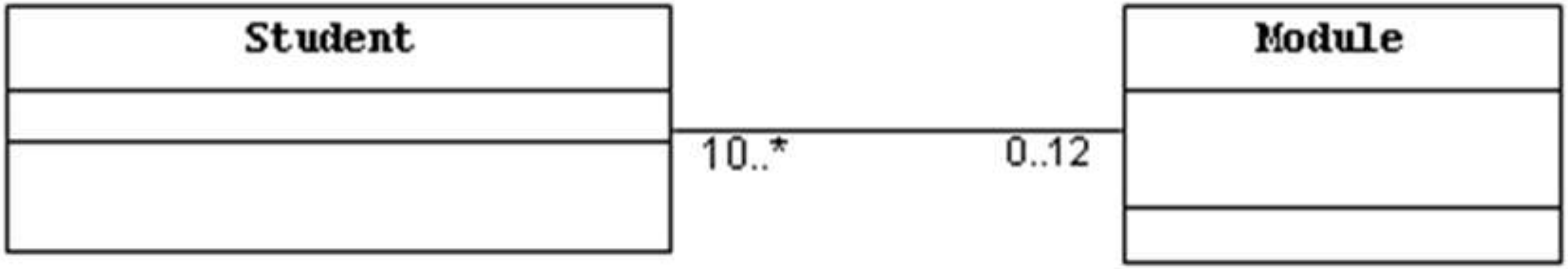
Class Diagrams



Associations: Multiplicity

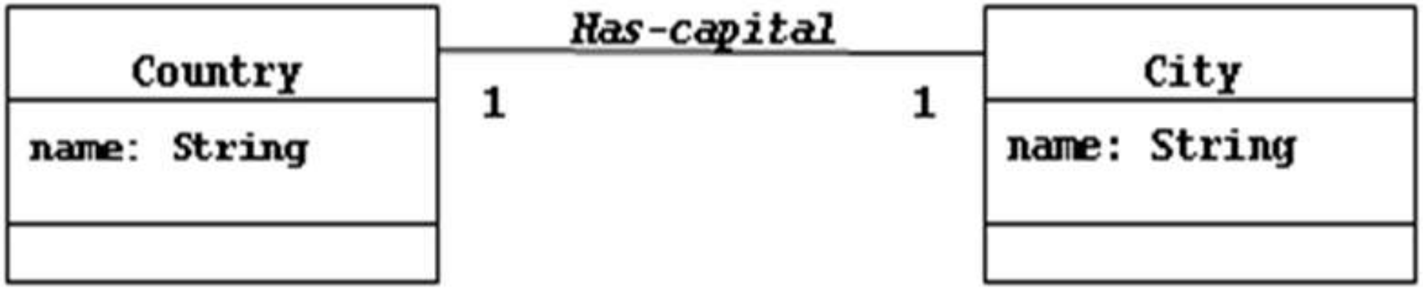


- Many-to-many relationship

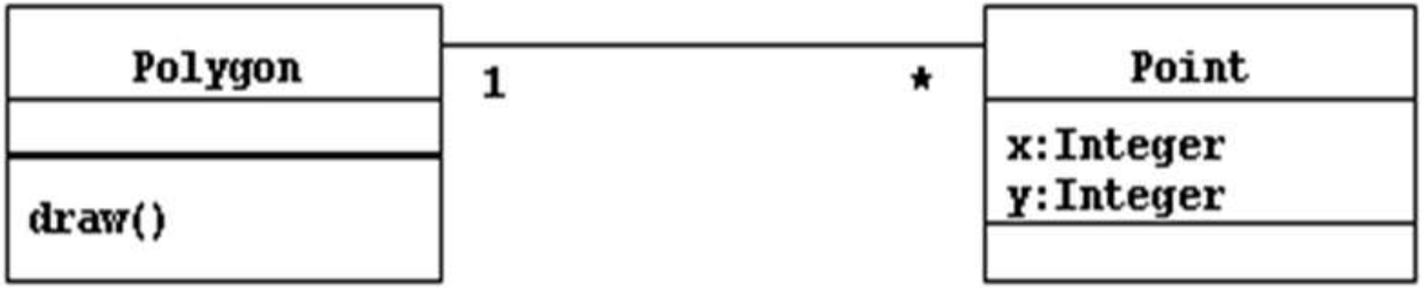


- A student takes between 0 and 12 modules
- A module is taken by at least 10 students

1-to-1 and 1-to-many Associations



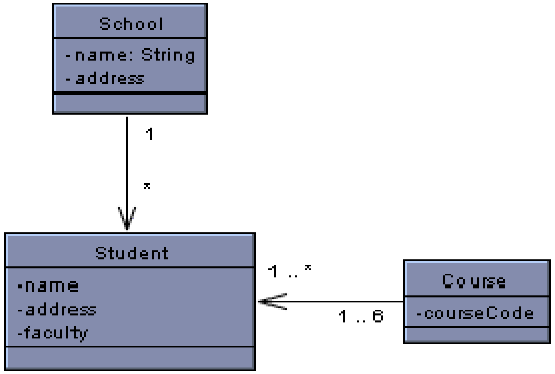
1-to-1 association



1-to-many association

Association, Aggregation and Composition

- **The association link indicates that two classes have a relationship: a student attends a school; a student takes courses.**



- **Each link has two ends which are called roles. Each role has a name, a multiplicity, a navigability and a type.**

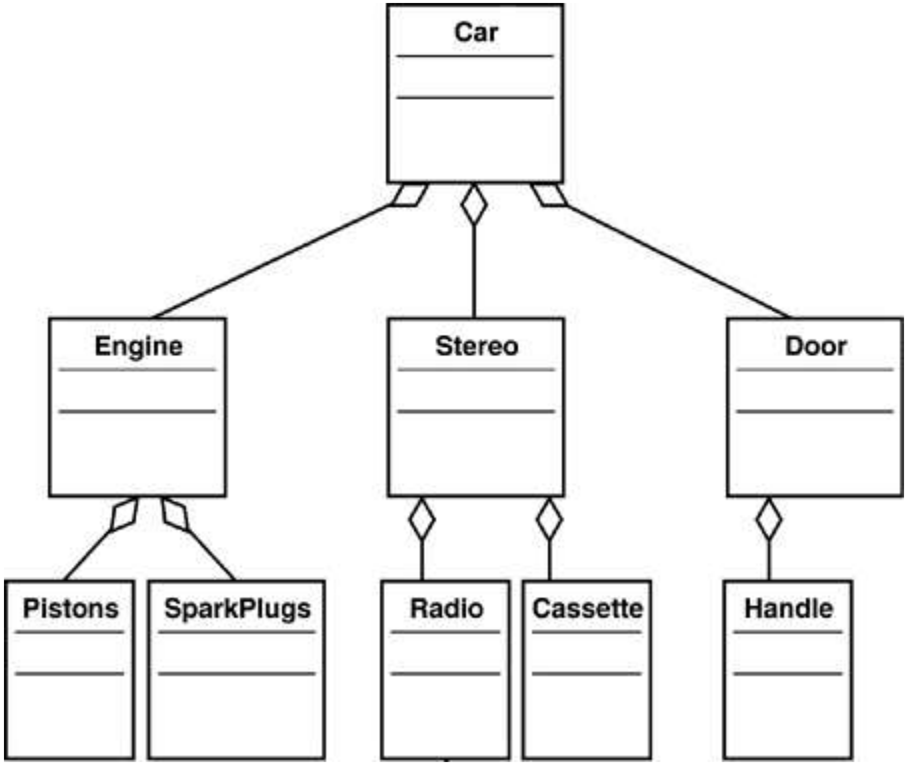
- The role can have one of three types: **association**, **composition** or **aggregation**.

- **Association** indicates that the two classes have a relationship.

Aggregation

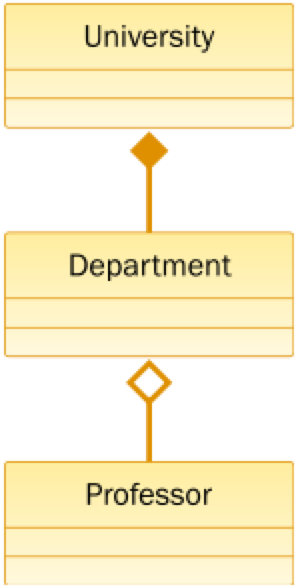
An aggregation is a special case of association denoting a **"consists of"** hierarchy.

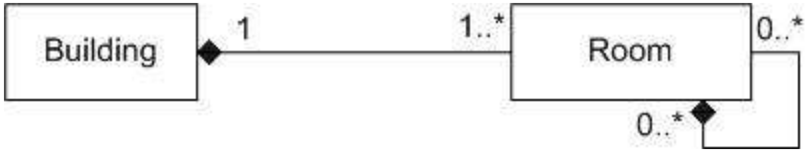
The aggregate is the **parent class**, the components are the **children class**.



Composition

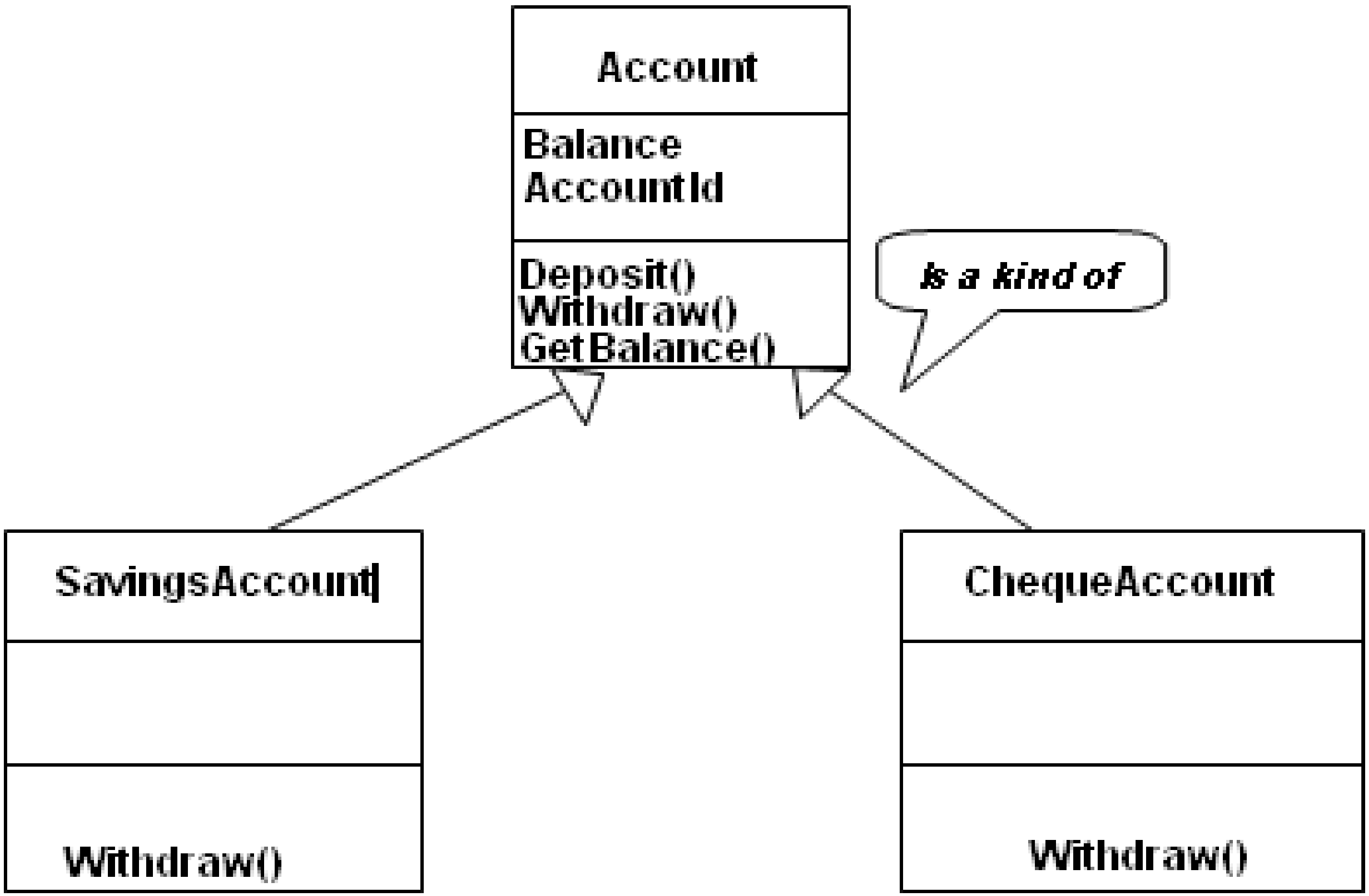
- **A solid diamond denotes *composition*, a strong form of aggregation** *where components cannot exist without the aggregate*.





Composition between two classes

Generalization/Inheritance



Inheritance

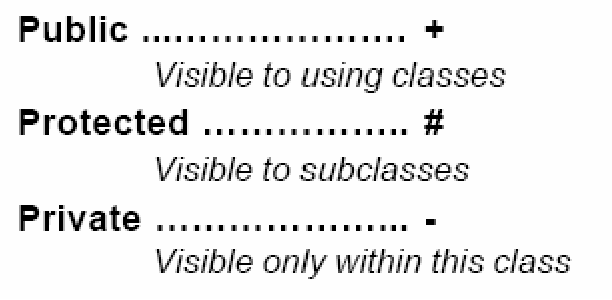
- **Classes inherit the attributes and operations of their ‘parents’ i.e. from the generalization to the specialization**
- **Operations and attributes may be re-defined**
- **Additional operations or attributes must be defined**
- **Operations and attributes may *not* be removed in the specialization**

Associations: which type?

- If in doubt use simple association

- Use aggregation or composition for “has a“ relationship
- s
- Use inheritance for “is a“ relationships

Visibility and Scope ...



Public .... **+**

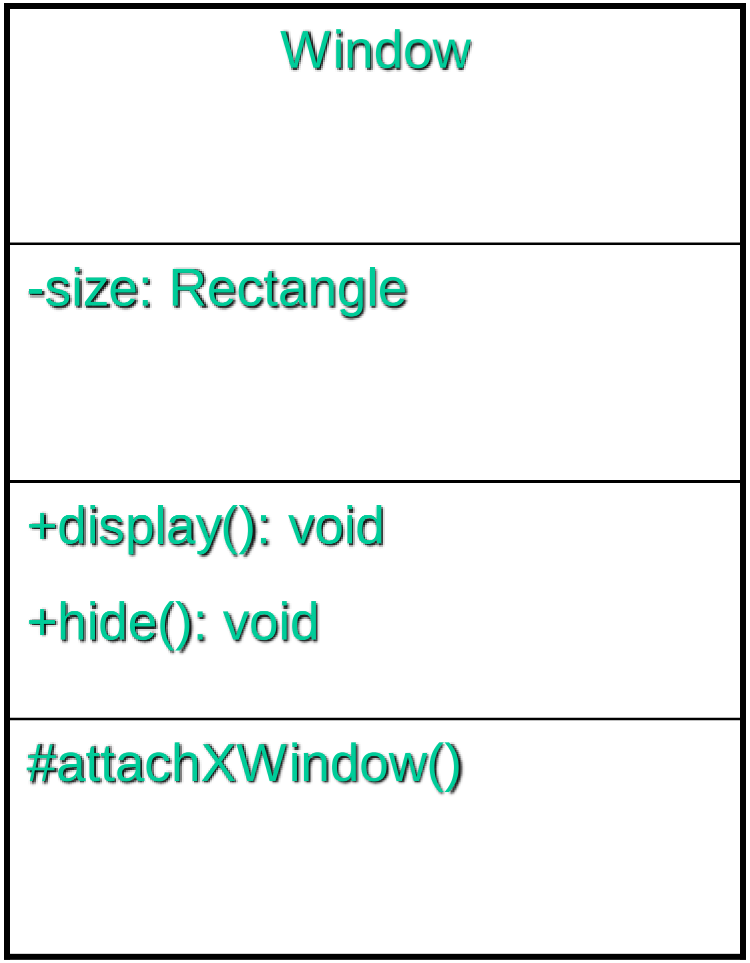
Visible to using classes

Protected .... **#**

Visible to subclasses

Private .... **-**

Visible only within this class



Window

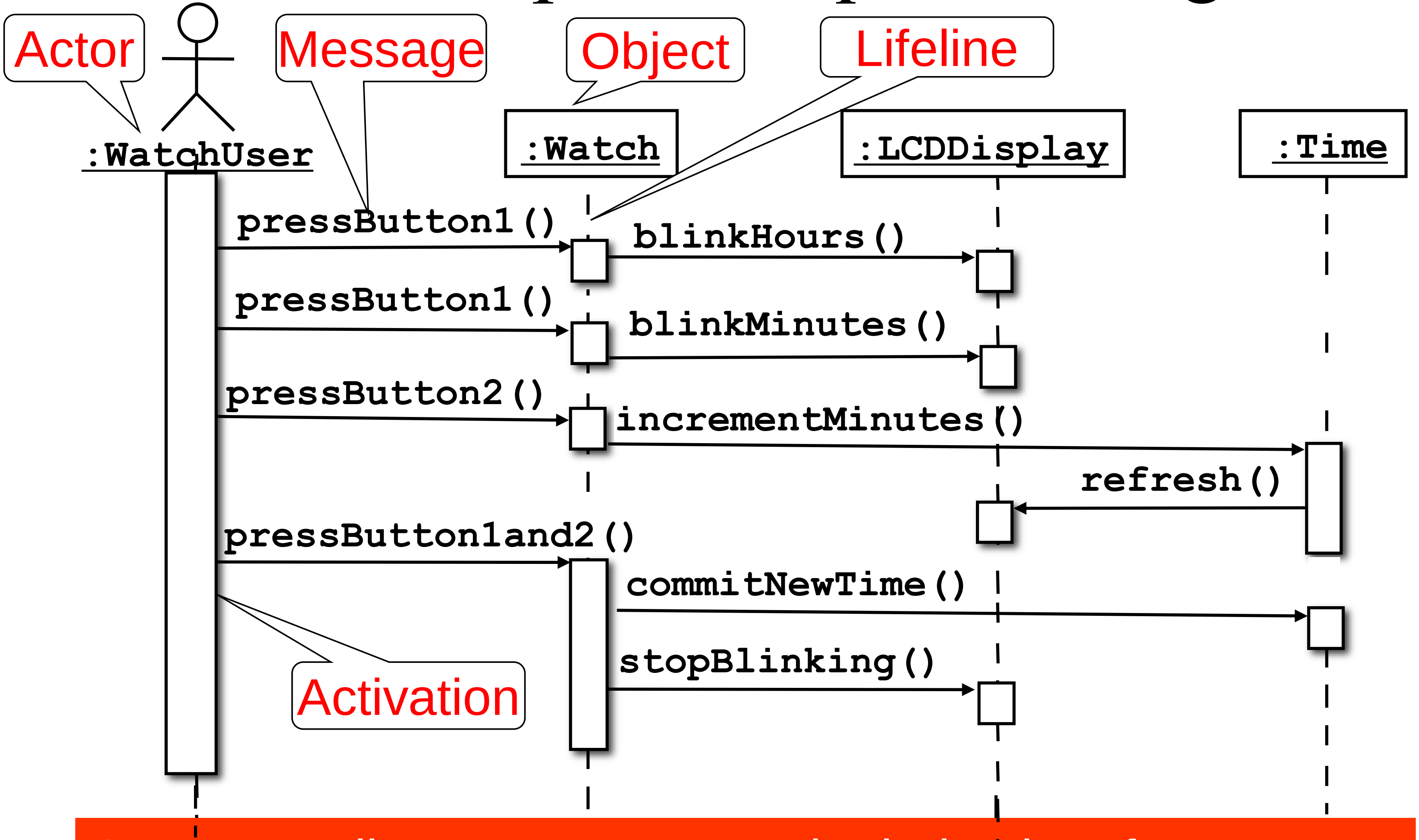
-size: Rectangle

+display(): void

+hide(): void

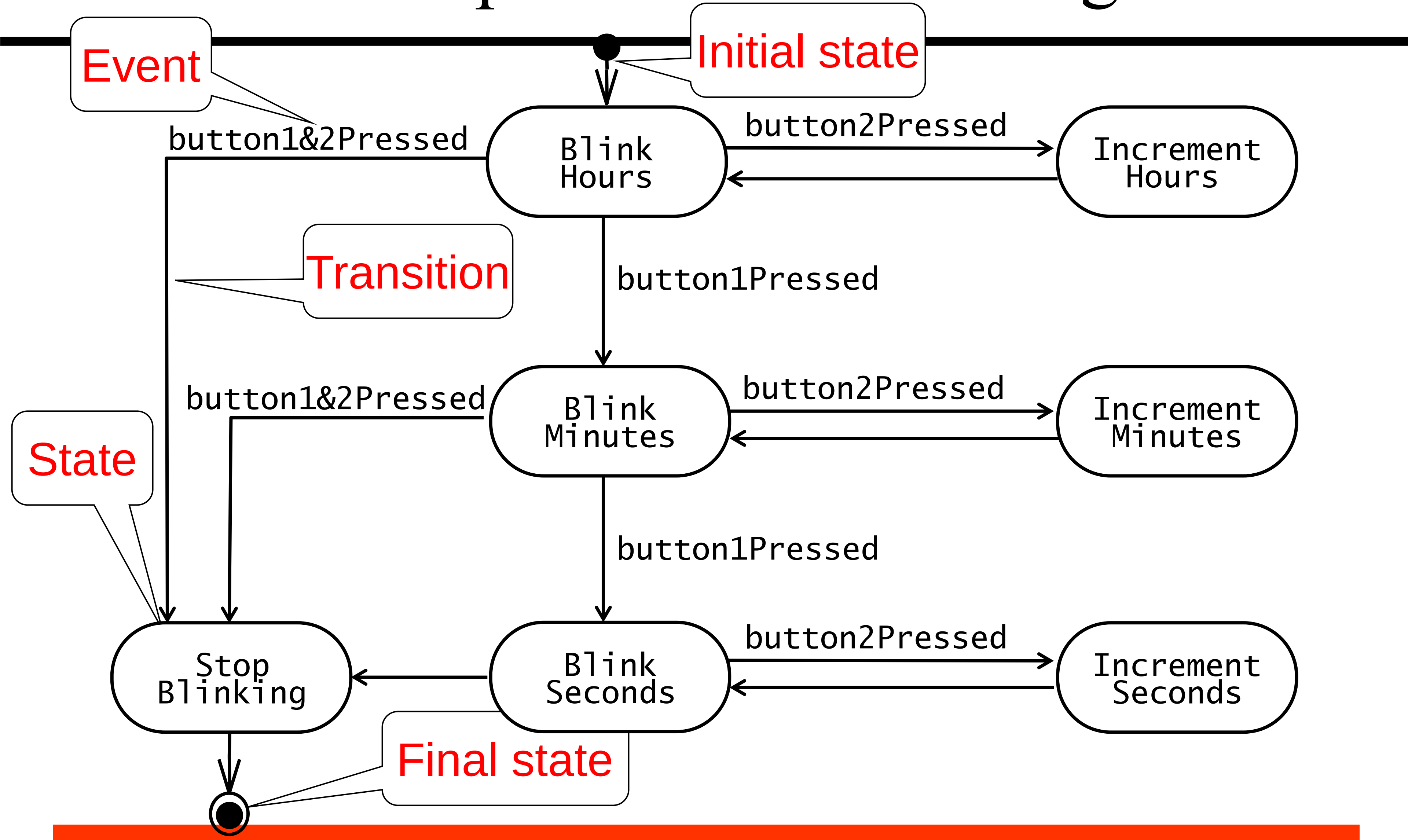
#attachXWindow()

UML first pass: Sequence diagram



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

UML first pass: Statechart diagrams



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

References

- Bernd Bruegge & Allen H. Dutoit Object-Oriented Software Engineering: Using UML, Patterns, and Java
- Software Engineering, Ivan Marsic, 2020
- Sommerville, I. (2015). Software Engineering 10. Pearson.
- Gustafson, D., 2002. Schaum's Outline of Software Engineering. McGraw-Hill, Inc.