Software Engineering

Dr. Fatma ElSayed

Computer Science Department
fatma.elsayed@fci.bu.edu.eg

Course Information

Contents

- Introduction to Software Engineering
- Software Processes
- Requirements Engineering
- System Modelling Part I
- System Modelling Part II
- System Architecture
- Software Testing Strategies
- Software Testing Techniques
- Technical Metrics for Software

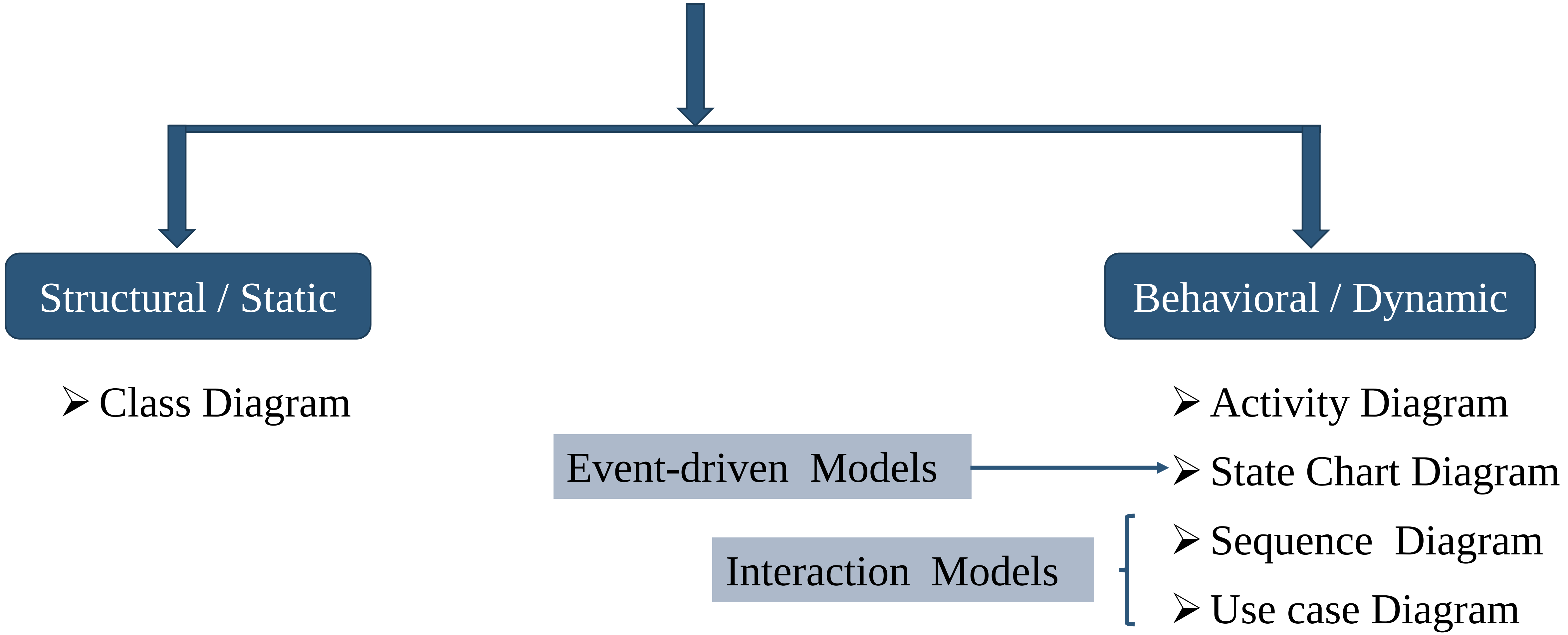
Course Information

Contents

- Introduction to Software Engineering
- Software Processes
- Requirements Engineering
- System Modelling Part I
- ➡ System Modelling Part II
- System Architecture
- Software Testing Strategies
- Software Testing Techniques
- Technical Metrics for Software

Chapter 5: System Modelling

Static vs Dynamic Models



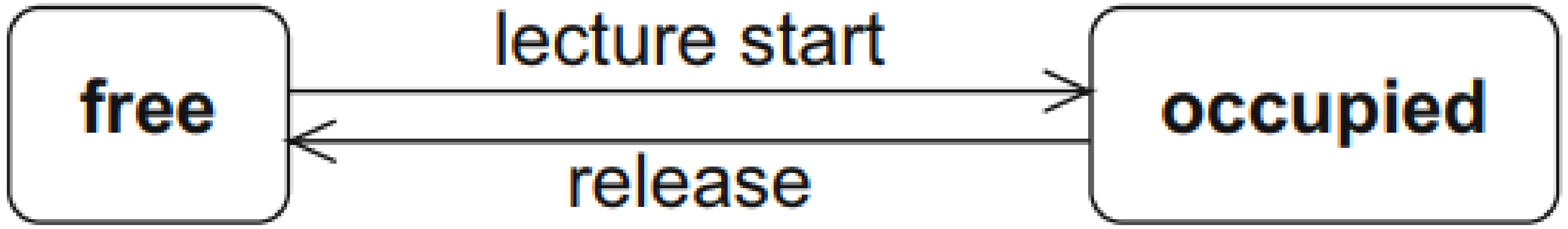
Event-Driven Models

- Event-driven modelling shows how a system responds to **external** and **internal** events. It is based on the assumption that a system has a **finite number of states** and that **events** *(stimuli)* may cause a **transition** from one state to another.
- The UML supports event-based modelling using **State Machine Diagram or state chart diagram** *(state diagrams)*.
- **State diagrams** show system states and events that cause transitions from one state to another. They **do not show the flow of data** within the system but may include additional information on the computations carried out in each state.

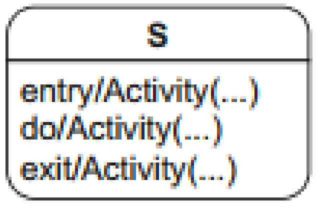
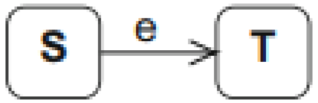
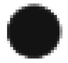
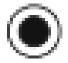
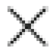
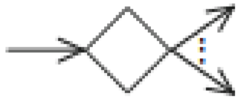
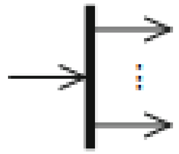
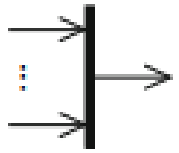
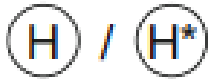
State Diagram: Simple Example

- A state diagram shows the states of a **single object**.

- As a simple example consider a **lecture hall** (object) that can be in one of two states: free or occupied. When a lecture starts in the lecture hall, the state of the lecture hall changes from free to occupied. Once the respective event in the lecture hall has finished and the hall has been released again.

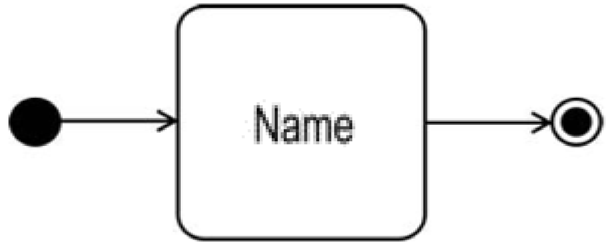


State Diagram Elements

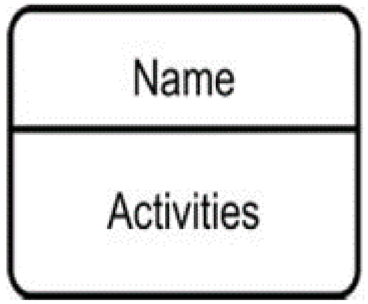
Name	*Notation*	*Description*
State	 S entry/Activity(...) do/Activity(...) exit/Activity(...)	Description of a specific "time span" in which an object finds itself during its "life cycle". Within a state, activities can be executed on the object.
Transition		State transition e from a source state S to a target state T
Initial state		Start of a state machine diagram
Final state		End of a state machine diagram
Terminate node		Termination of an object's state machine diagram
Decision node		Node from which multiple alternative transitions can proceed
Parallelization node		Splitting of a transition into multiple parallel transitions
Synchronization node		Merging of multiple parallel transitions into one transition
Shallow and deep history state		"Return address" to a substate or a nested substate of a composite state

State Diagram

- A state is shown as a rectangle with round corners and is **labeled** with the name of the state.



- If internal activities are specified for a state, it is divided into two sections: the **upper** section contains the **name** of the state; the **lower** section includes **internal activities**.

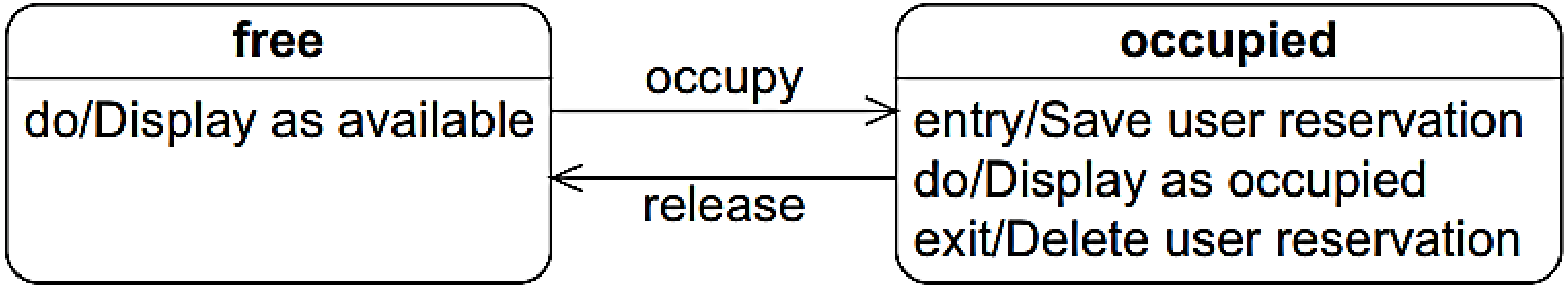


Adding Details to the State Icon

Three frequently used categories of activities :

1. **Entry:** What happens when the system *enters* the state.
2. **Exit:** what happens when the system *leaves* the state**.**
3. **Do:** what happens while the system is *in* the state.

* **Note**: You can add others as necessary.



Adding Details to the State Icon

Example **(Fax Machine)** has 2 states:

1. **Faxing (Sending a fax)**
2. **Idle**

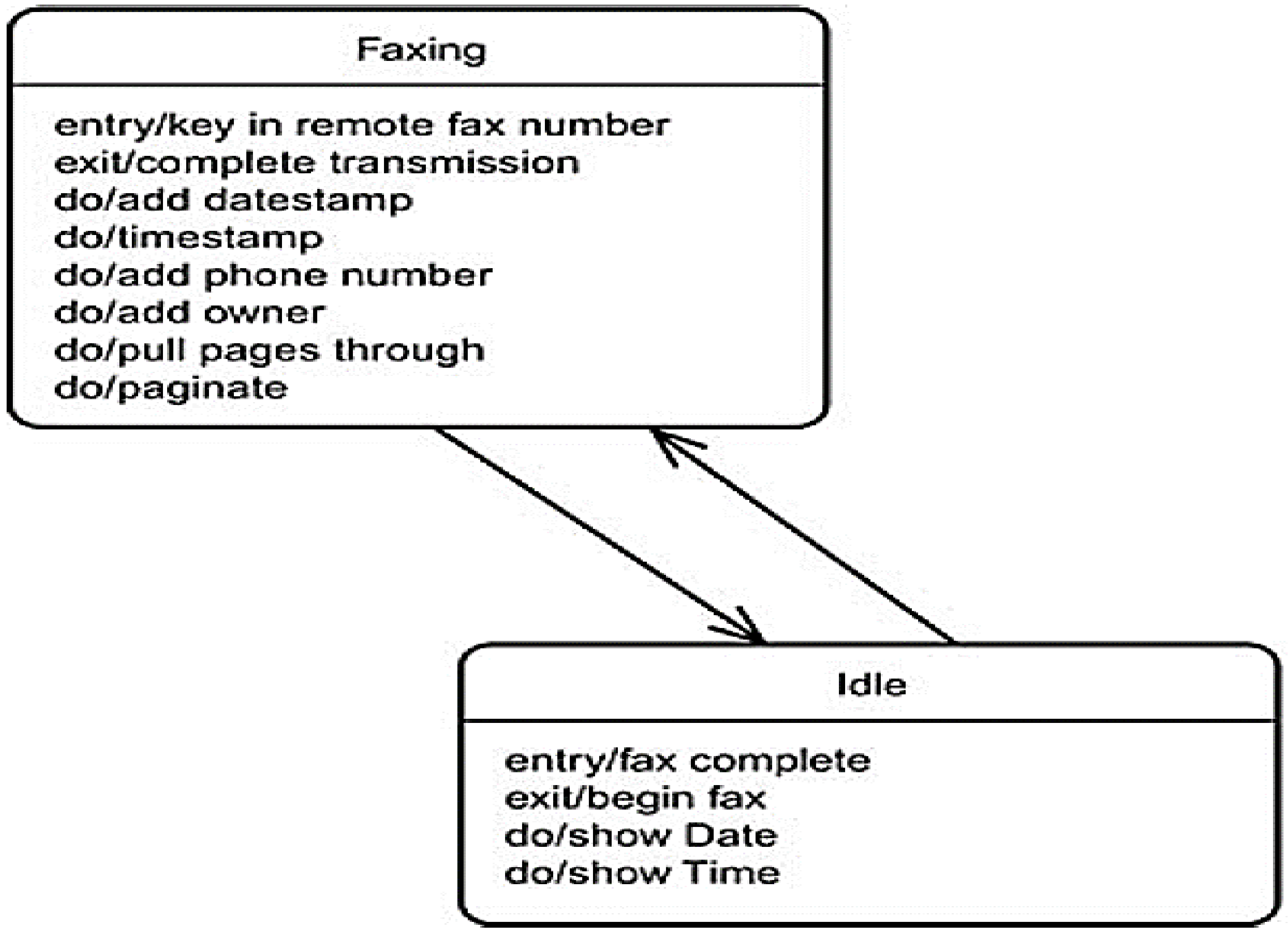
Faxing State Activities

1. Adding a date stamp and time stamp
2. Adding phone number
3. Adding name of owner.
4. Pulls the pages through,
5. Paginates
6. Completes the transmission.

Idle State Activities

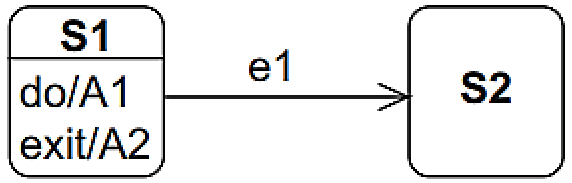
1. Fax machine presents the date and time on a display.

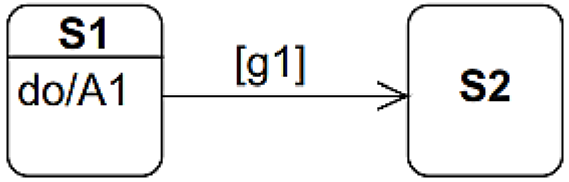
Adding Details to the State Icon

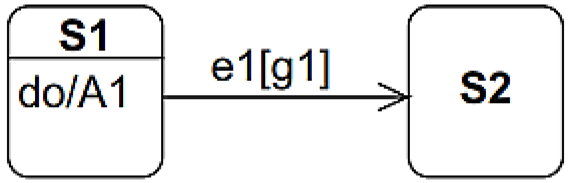


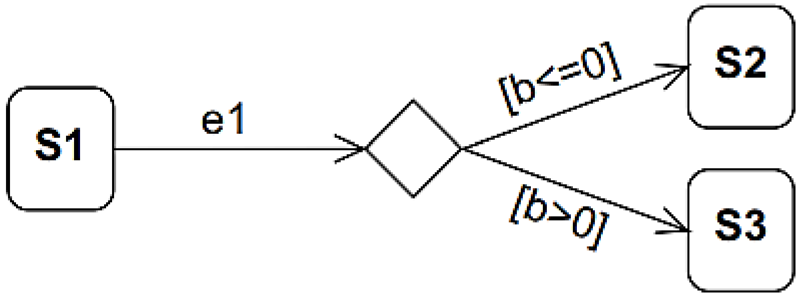
Adding Details to the Transition

- You can specify various properties for a transition:
  - The **event** (also called "**trigger**") that triggers the state transition.
  - The **guard** (also called "**condition**") that enables the execution of the transition.

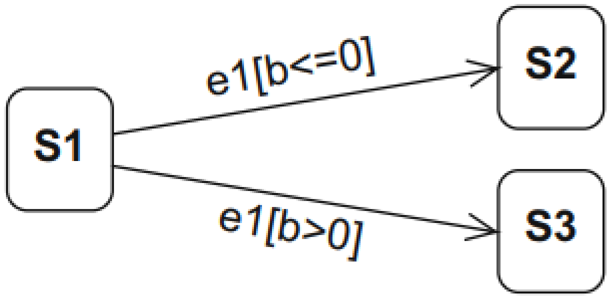




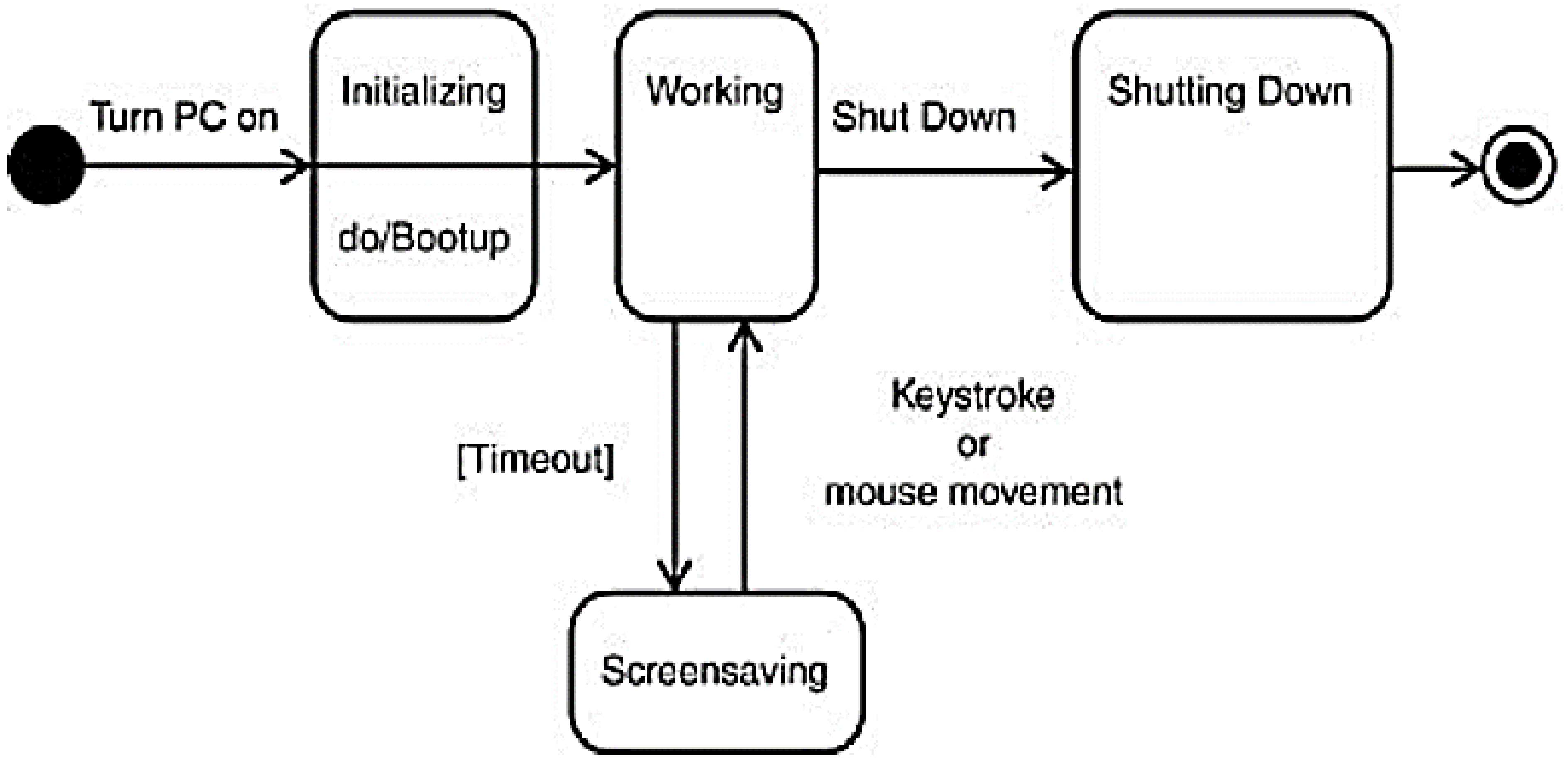




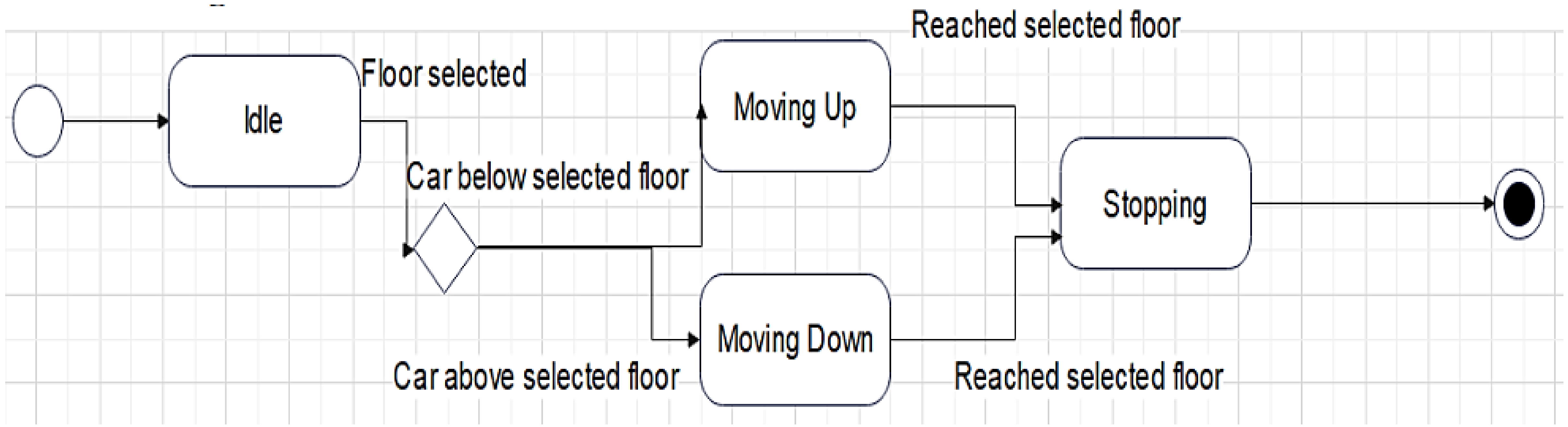
Equivalent to



Adding Details to the Transition



Adding Details to the Transition



* **hint:** *Car is the Elevator*

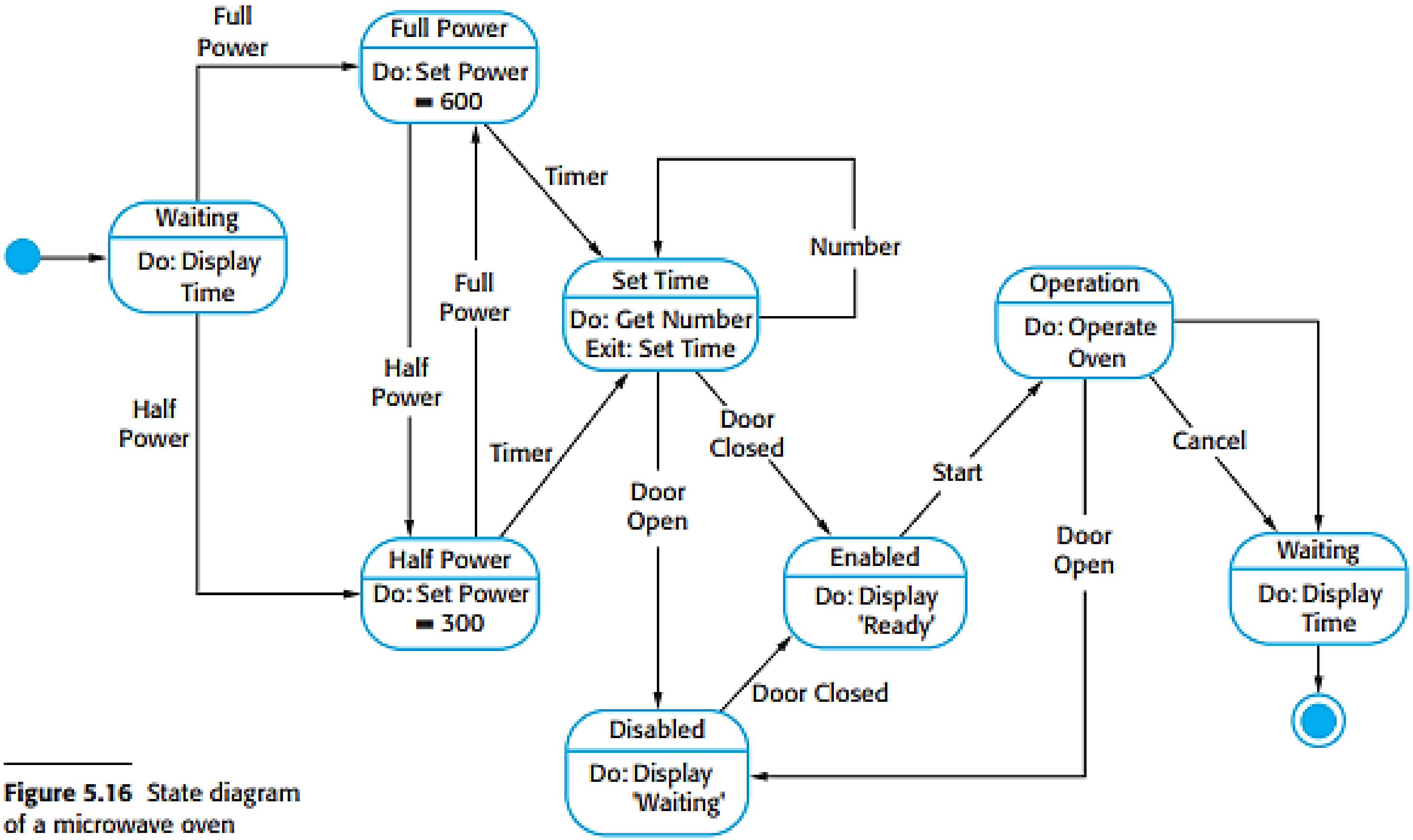


Figure 5.16 State diagram of a microwave oven

Sequence Diagram

Interaction Models

- All systems involve interaction of some kind. This can be:
  - User interaction, which involves **user inputs** and **outputs** (helps to identify user requirements).
  - Interaction between the **system** being developed and **other systems** (highlights the communication problems that may arise).
  - Interaction **between the components** of the system (helps us understand if a proposed system structure is likely to deliver the required system performance and dependability).

Two related approaches to interaction modelling:

1. **Use case modelling,** which is mostly used to model interactions between a system and external actors.
2. **Sequence diagrams,** which are used to model interactions between system components, although external agents may also be included.

Sequence Diagram

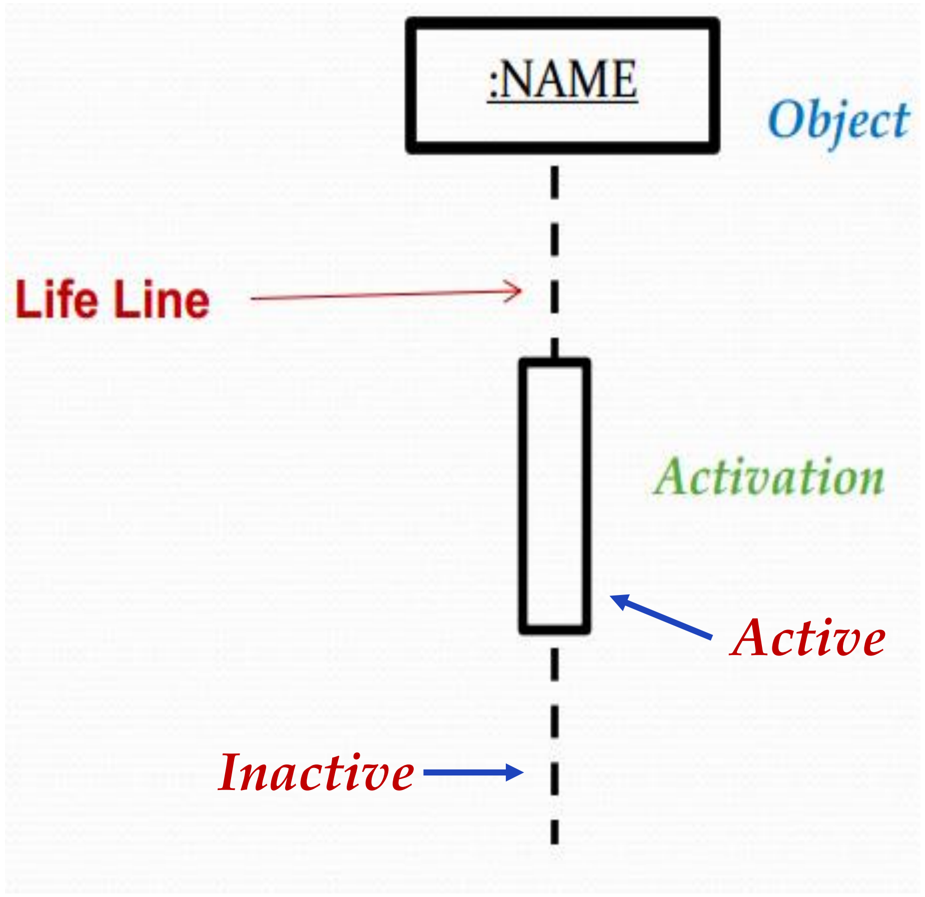
- A **sequence diagram** shows the sequence of interactions that take place during a particular use case or use case instance.
- Each diagram represent <u>one of the flows </u>through a **use case.**
- Each Use Case requires **one or more** sequence diagrams to describe its behavior.
- Sequence diagrams are interaction diagrams **ordered by time.**

Shows, step-by-step, flows through a use case:

- What **objects** are needed for the flow.
- What **messages** the objects send to each other.
- What **actor** initiates the flow.
- What **order** the messages are sent.

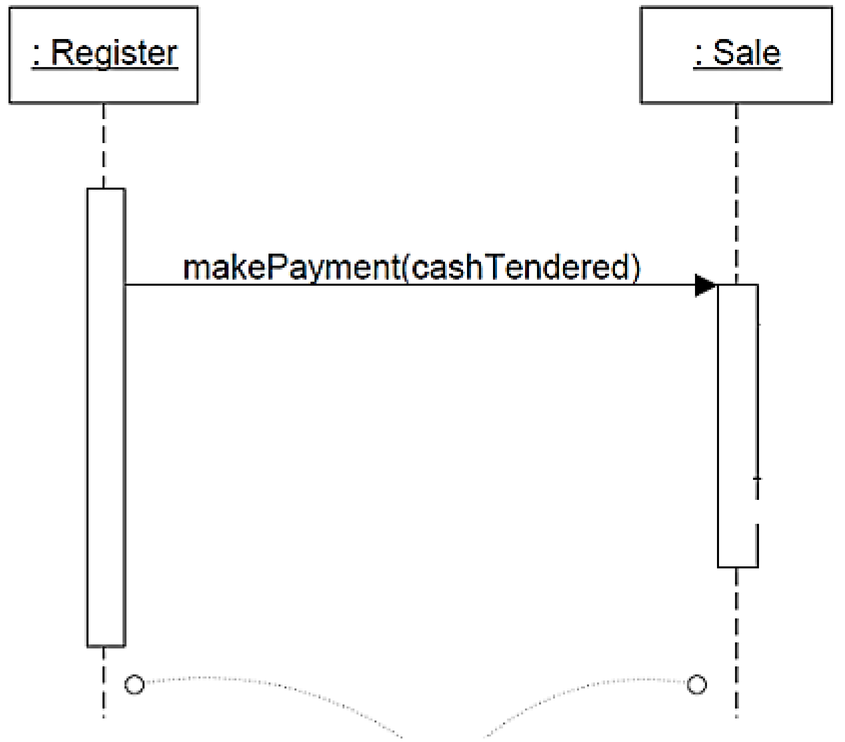
Object Lifeline

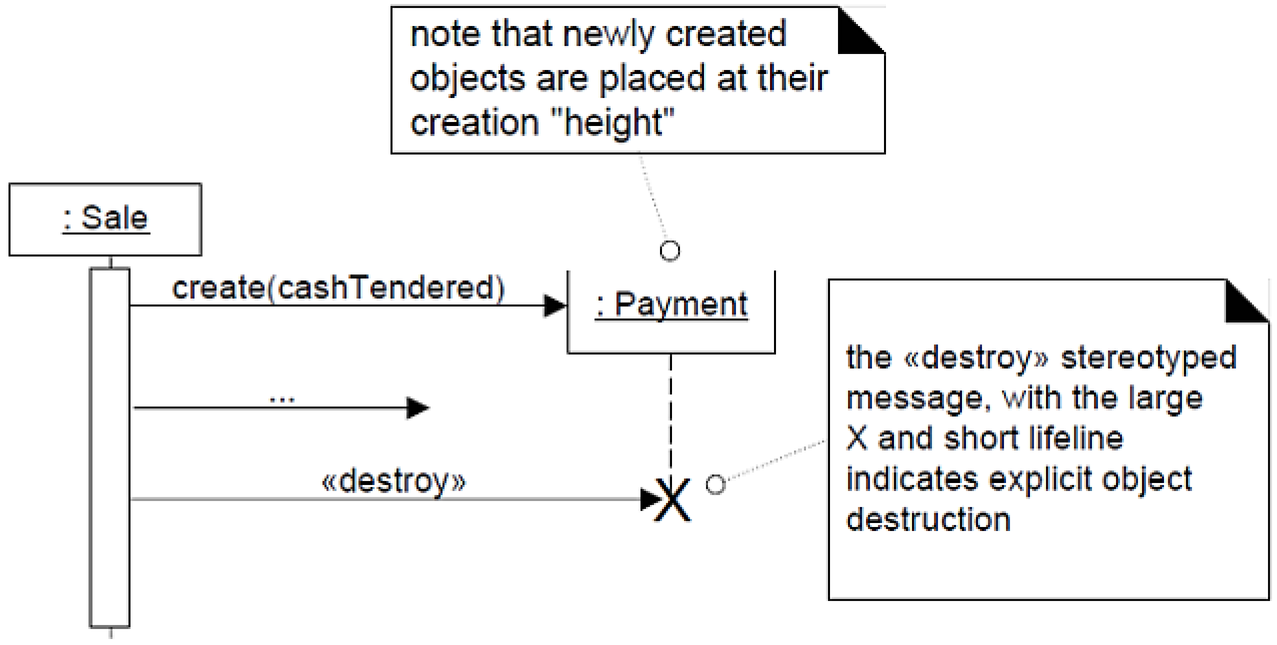
- A **lifeline** is shown as a vertical, usually dashed line that represents the lifetime of the object associated with it.
- At the top end of the line is the head of the lifeline, a rectangle which contains an expression in the form ***roleName:Class*.**
- This expression indicates the **role** name and the **class** of the object associated with the lifeline; either name may be **omitted**.
- If the class is omitted, the colon can also be omitted; if only the class is specified, it must be preceded by a colon.



Object Lifeline

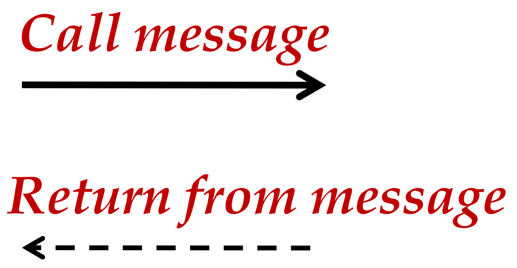
- A The sequence diagram is a two-dimensional diagram.
  - The top-to-bottom dimension shows the passage of time. ***[vertical]***
  - The left-to-right dimension is the layout of the objects. ***[horizontal]***




note that newly created objects are placed at their creation "height"
: Sale
create(cashTendered)
: Payment
...
«destroy»
the «destroy» stereotyped message, with the large X and short lifeline indicates explicit object destruction

Exchanging Messages

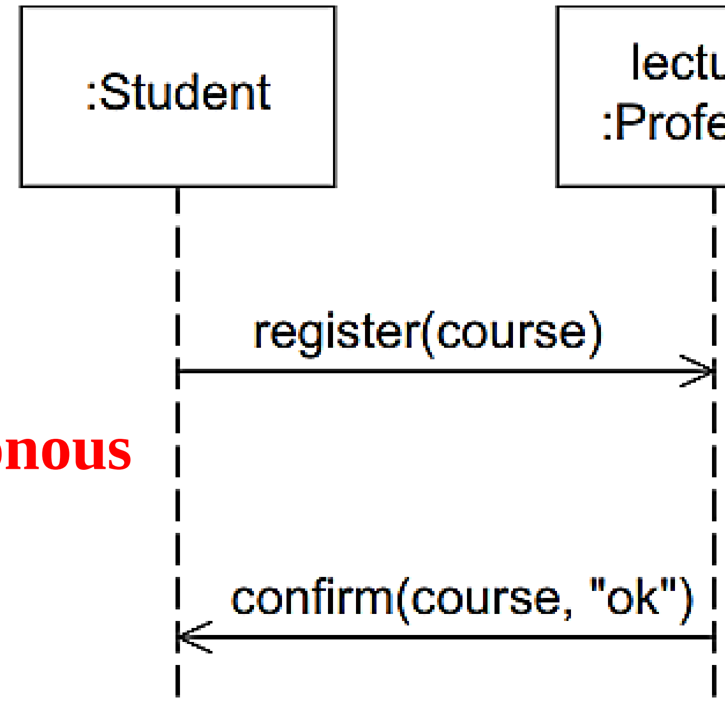
- A message is shown as an arrow from the **sender** to the **receiver**.
- The type of the arrow expresses the type of communication involved, and there are two types of messages:
  1. A ***synchronous*** message is represented by an arrow with a continuous line and a **filled triangular arrowhead**.
     - the sender **waits** until it has received a response message before continuing.
     - The response message is represented by a <u>dashed line</u> with an open arrowhead.

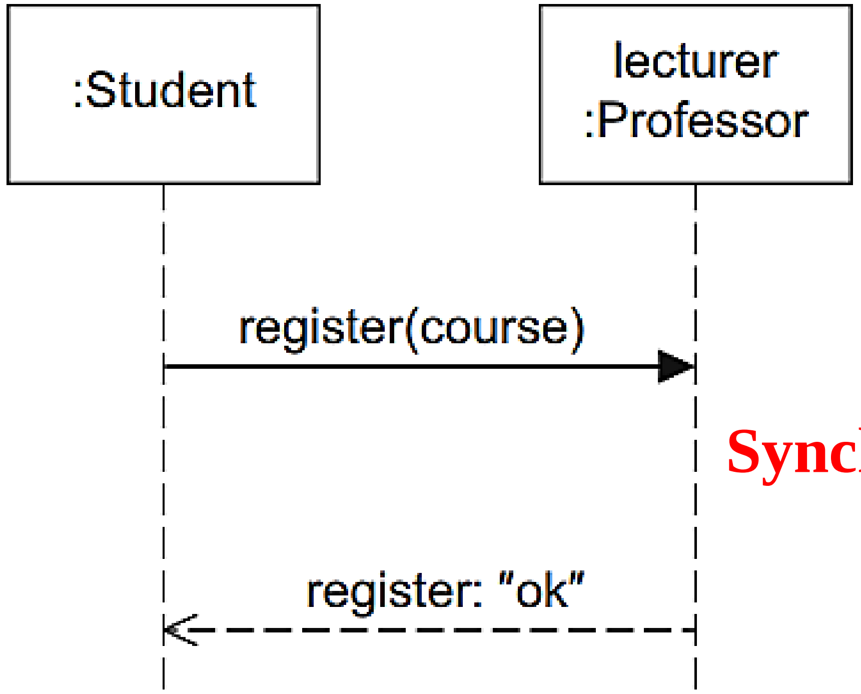


Exchanging Messages

2. An **asynchronous** message is represented by an arrow with a continuous line and an **open arrowhead**.
   - The sender continues after having sent the message.

Asynchronous

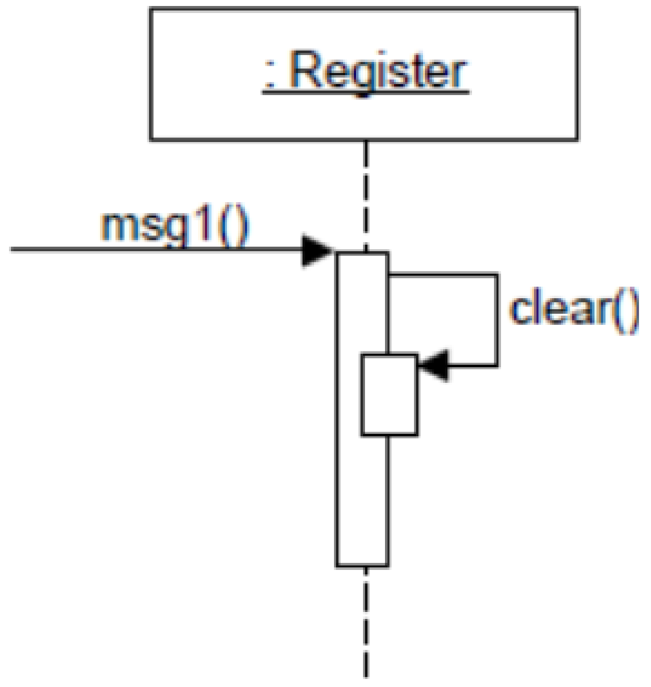


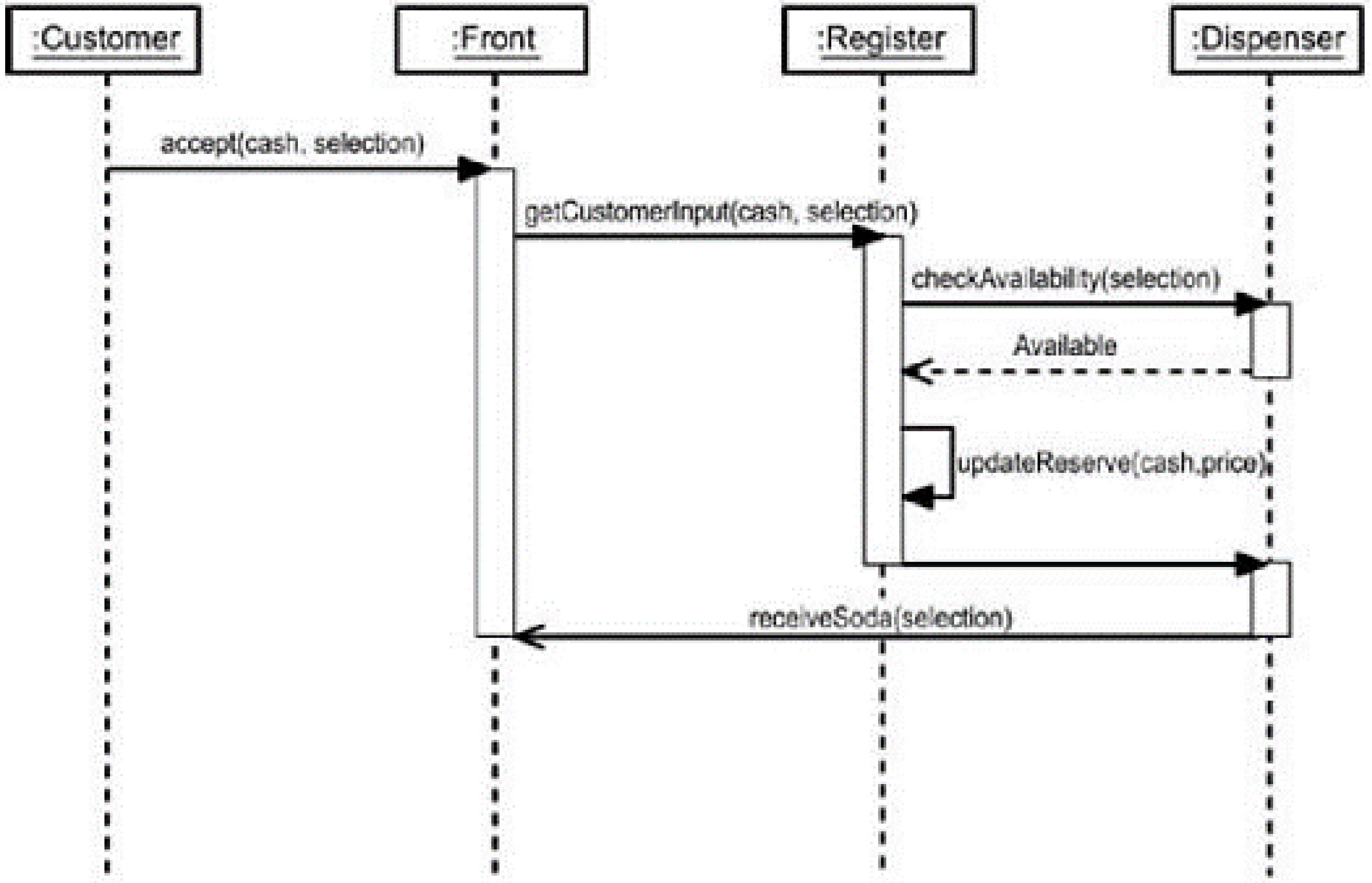


Synchronous

Register for a course

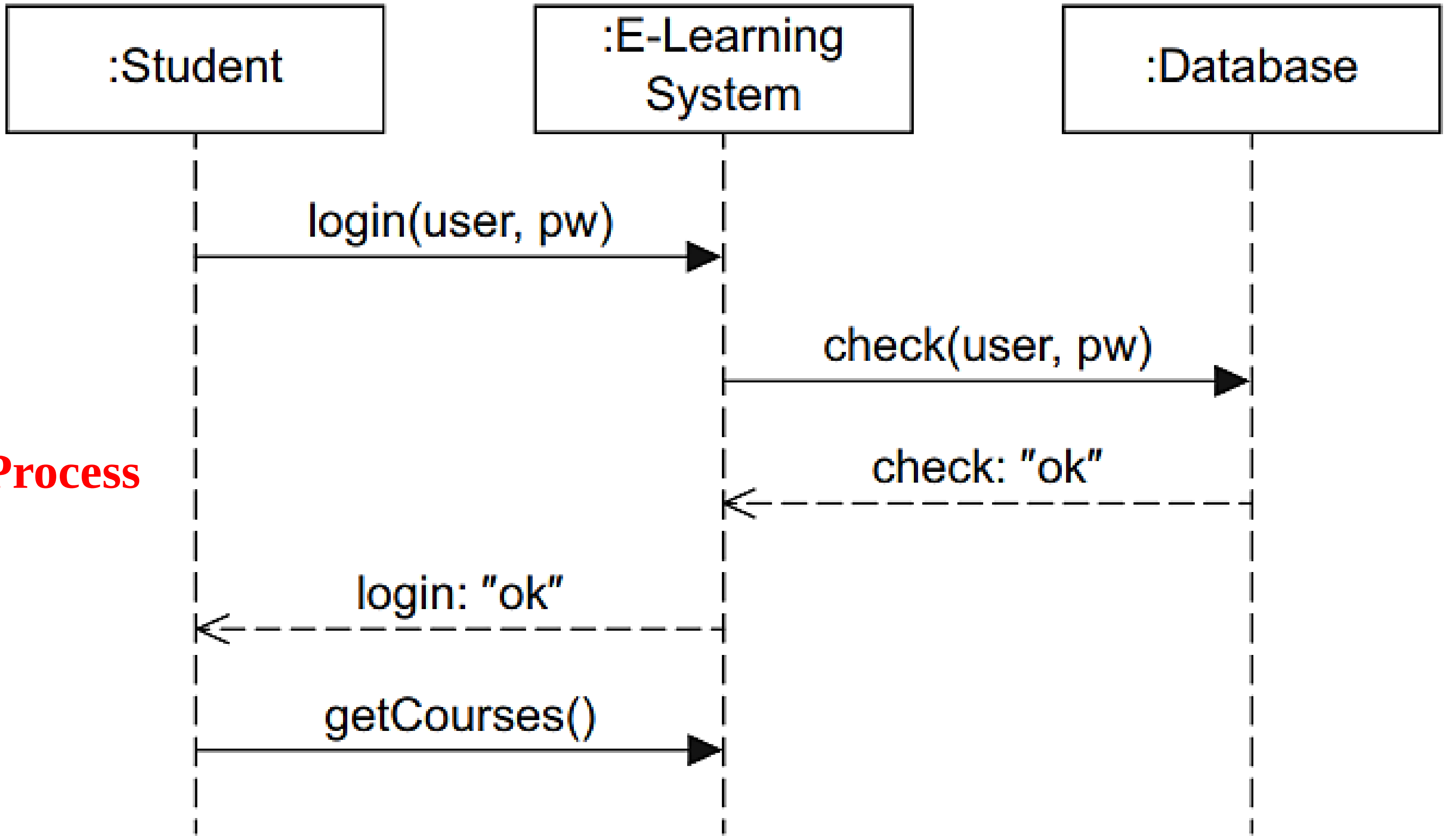
Message to Self





The Soda Machine: buying soda

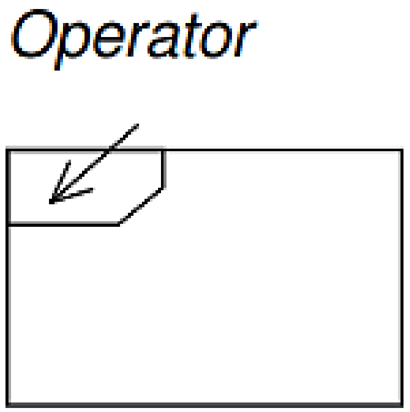
Sequence Diagram: Example

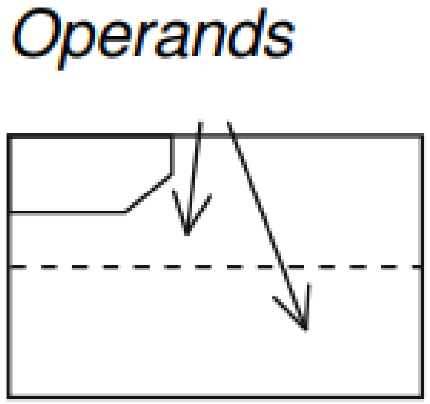


Login Process

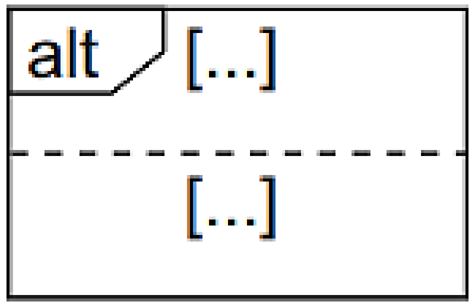
Alternatives and Iterations

- In a sequence diagram, you can use ***combined fragments (operators)*** to describe a number of possible execution paths.

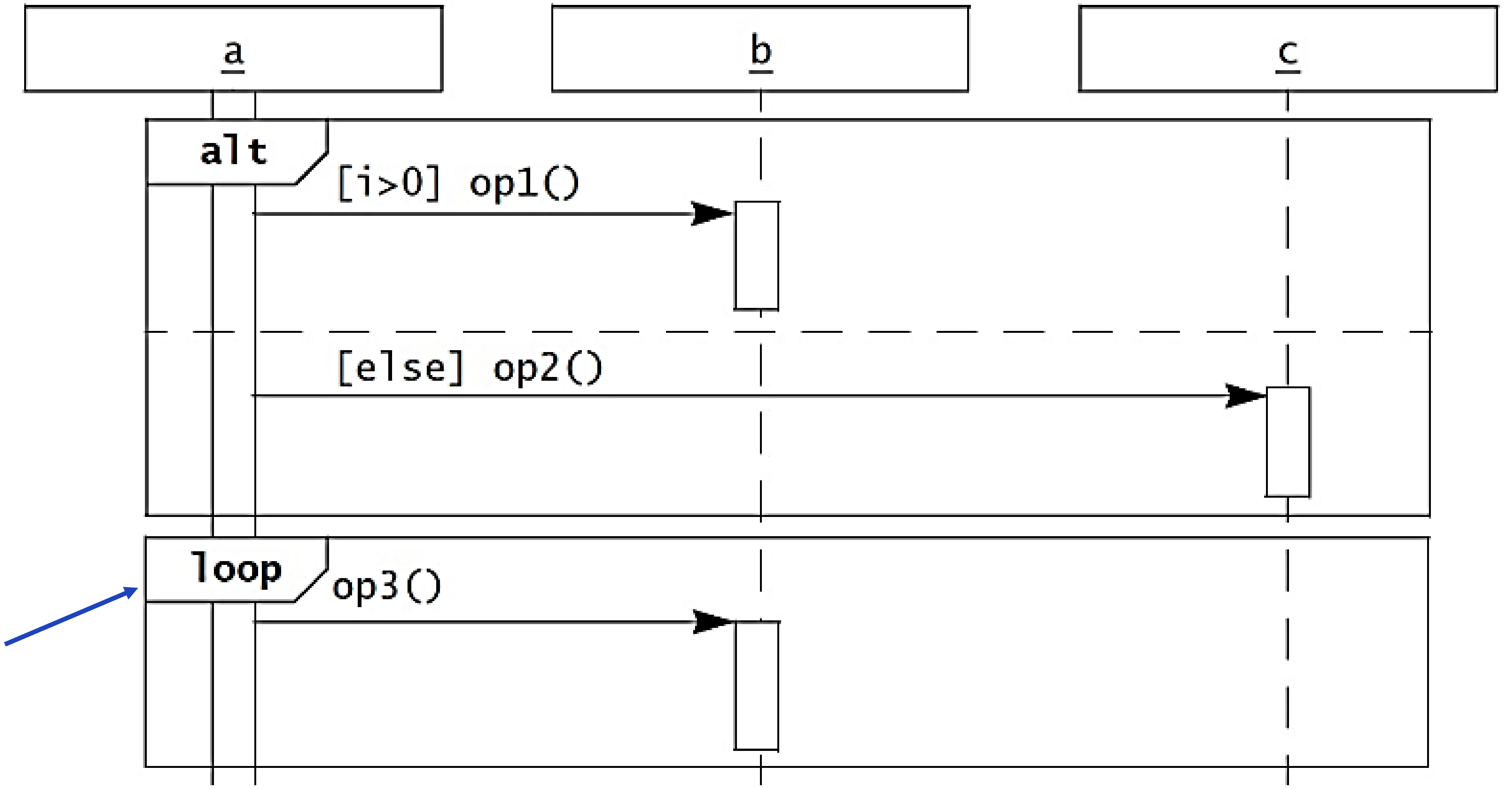




- You can use an **alt** fragment to represent alternative sequences, each operand represents an alternative path in the execution.
- Each operand has a ***guard***.



Alternatives and Iterations: Example

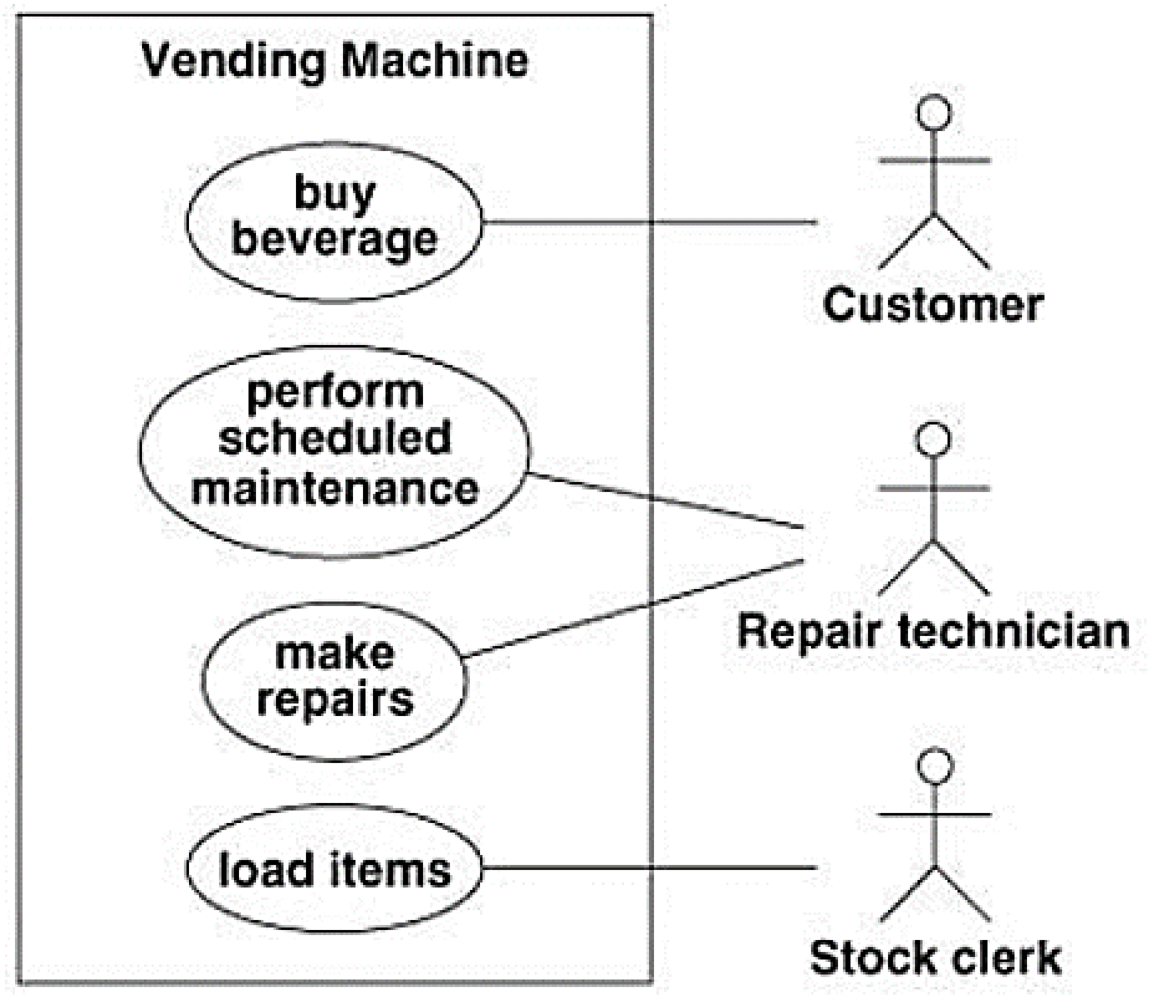


Loop

System Sequence Diagram (SSD)

For a use case scenario, an SSD shows:

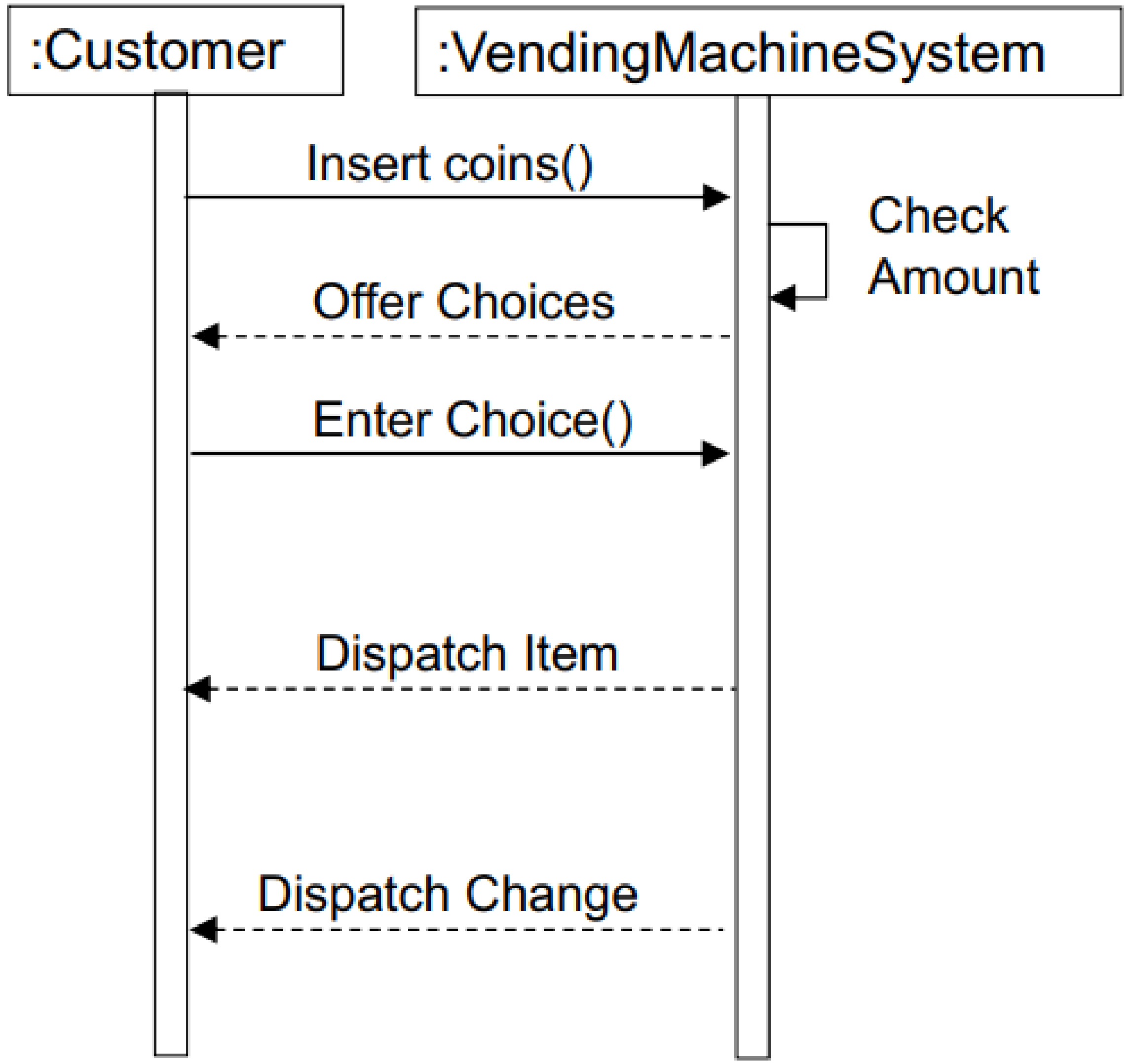
- The System (as a black box).
- The external actors that interact with System.
- The System events that the actors generate.



System Sequence Diagram (SSD)

The System events are:

- Insert coins()
- Enter choices()



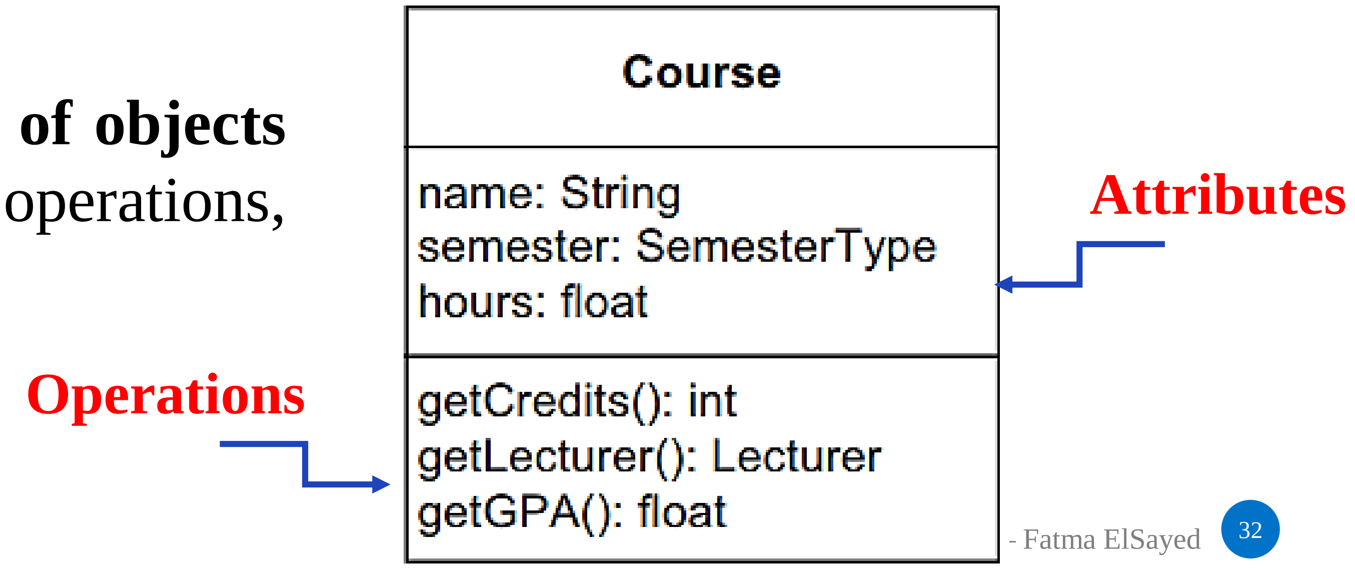
Class Diagram

Structural Models: Class Diagram

- Class diagrams are used when developing an **object-oriented** system model to show the **classes** in a system and the **associations** between these classes.
- A class model captures the **static structure** of the system by characterizing:
    - The **classes and objects** in the system,
    - The **attributes** and **operations** for each class of objects, and
    - The **relationships** among the objects.
- Class models are the **most important** OO models.
- In OO systems we build the system around objects, not functionality.
- Represent **structural and not behavioral** relationships that exist among system entities.

Objects

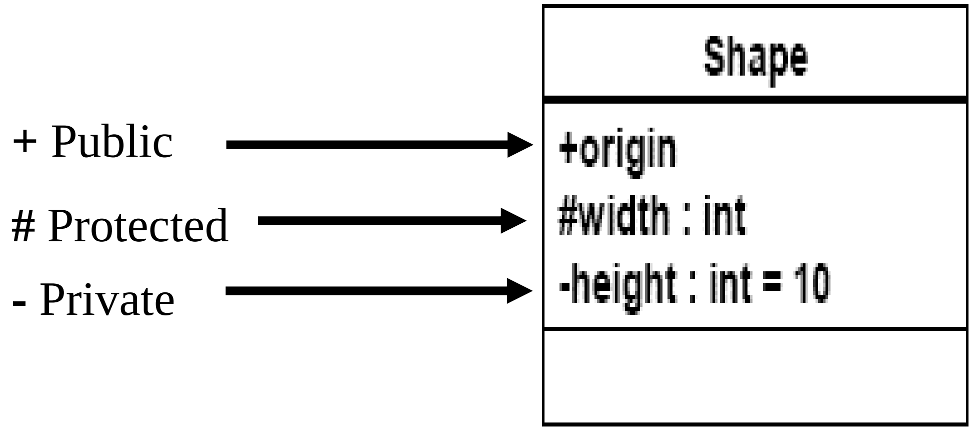
- **Objects** represent something in the **real world**, such as a patient, a prescription, a doctor, etc.
- Objects often appear as **proper nouns** in the problem description or discussion with the customer.
- The choice of objects depends on the analyst's judgment and the problem at hand. There can be more than one correct representation.
- A class is the description of a **set of objects** (having similar attributes, operations, relationships, and behavior).

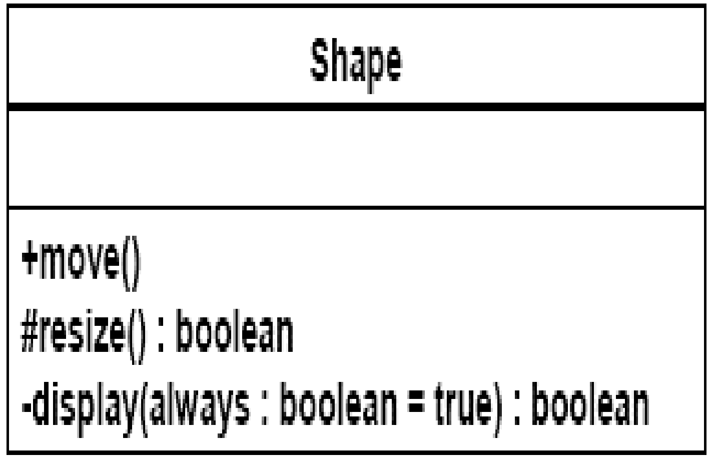


Class Attributes & Operations

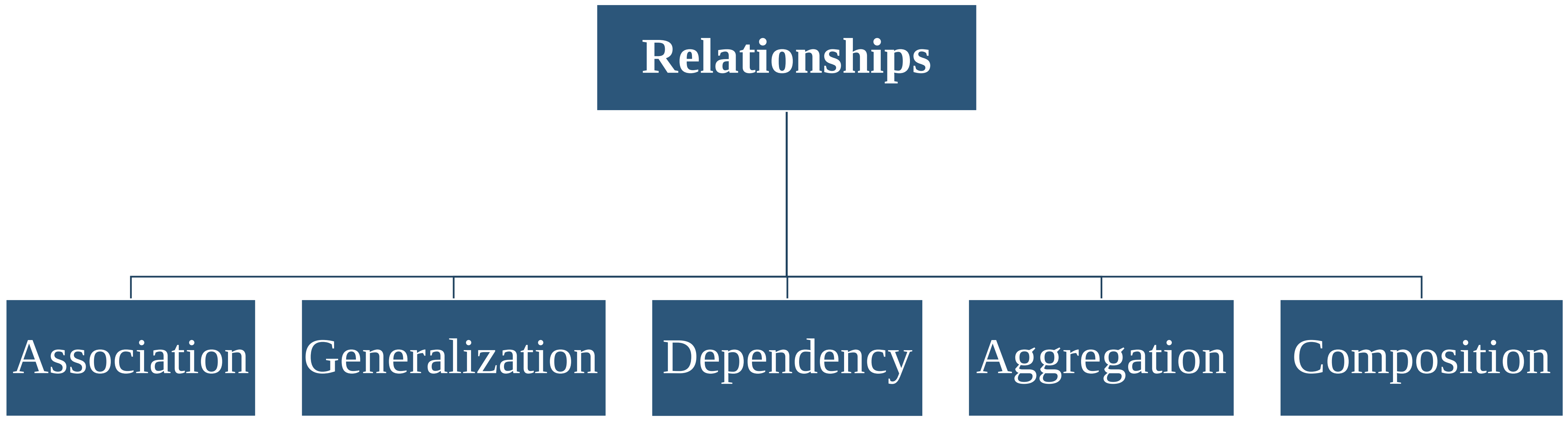
- An attribute allows you to **store** information that is known for all instances.
- Attribute declaration may include visibility, type and initial value

- Operations enable objects to **communicate** with one another and to act and react.
- An operation may include visibility, parameters, and return type:



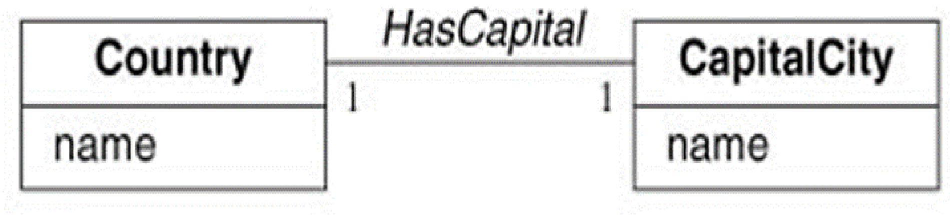


Relationships

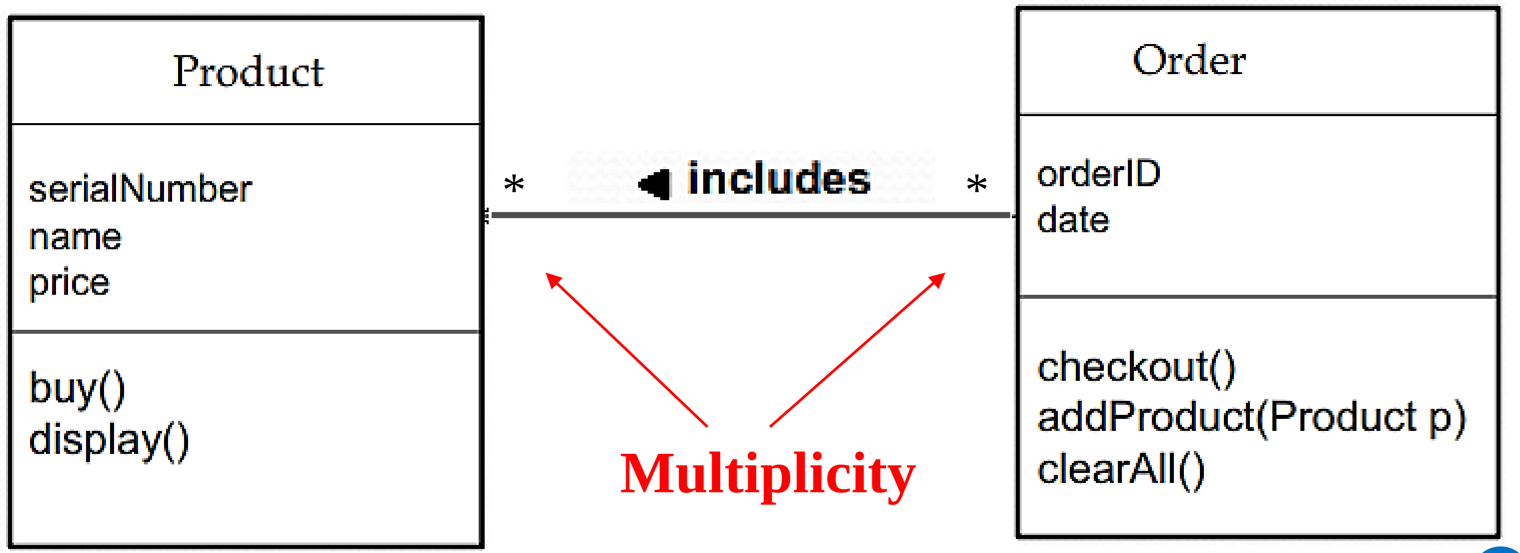


Associations

- An association is a **relationship** between classes is represented by a **line** with a name.

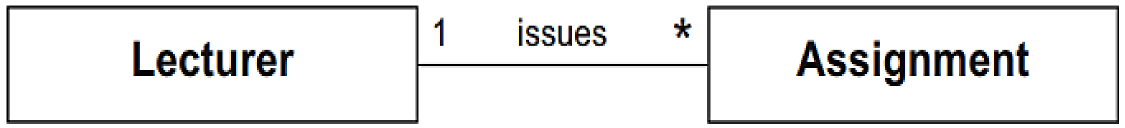


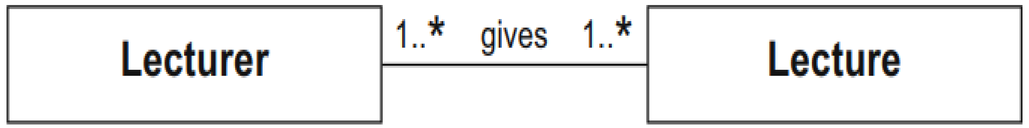
- An **optional** "reading direction arrow" indicates the **direction** to read the association name.

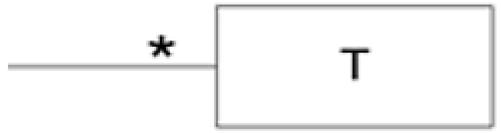


Multiplicity

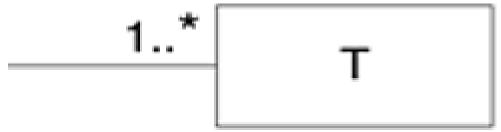
- **Multiplicity** specifies the **number** of instances of one class that may relate to a single instance of an associated class.



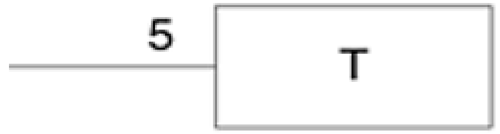




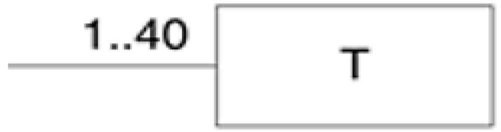
Zero or more



One or more



Exactly 5

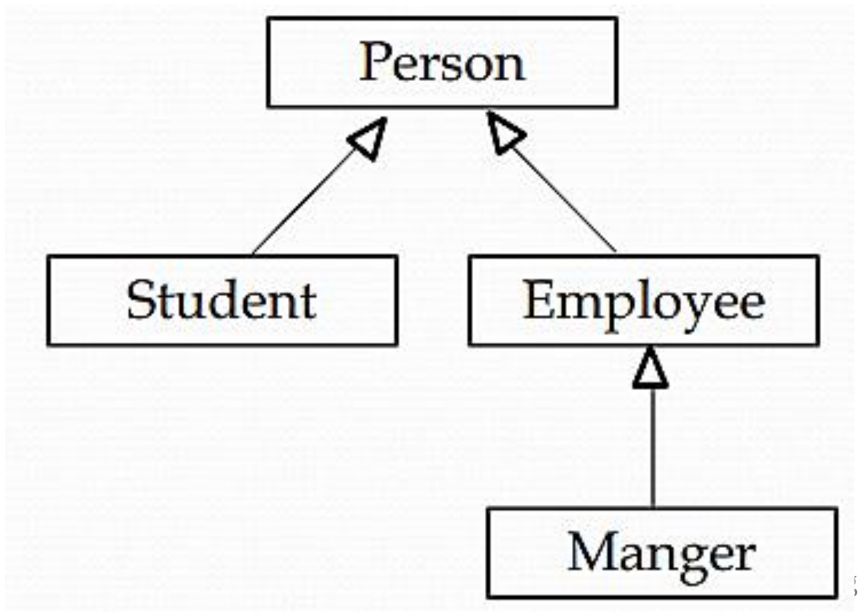


One to 40

Generalization

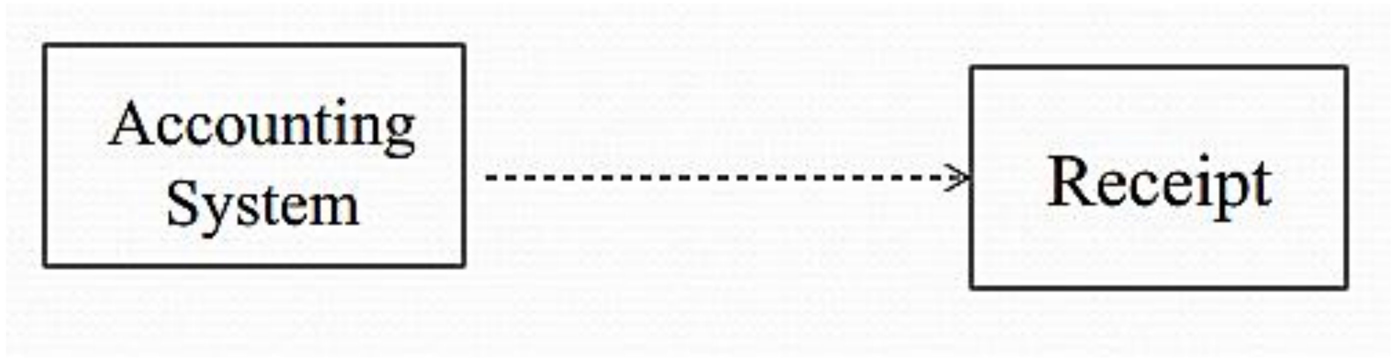
- The generalization relationship is also referred to as **inheritance**.
  1. A class might have no parents, which is a **base class or root class**.
  2. A class might have no children, in which case it's a **leaf class**.
  3. If a class has exactly one parent, it has a **single inheritance**.
  4. If a class has more than one parent, it has **multiple inheritance**.

Inheritance is called **"is-a"** *relationship.*



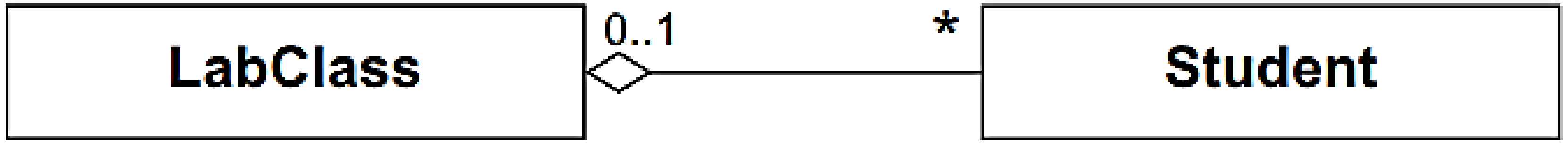
Dependency

- A kind of relationship in which one class **uses** another. This is called a **dependency**.



Aggregation

- An aggregation is a special form of association that is used to express that instances of one class **are parts of** an instance of another class.
- Represented by a **diamond** at the association end of the class that stands for the ***“whole”***.
- Expresses a **weak belonging** of the parts to a whole, meaning that parts also exist **independently** of the whole.

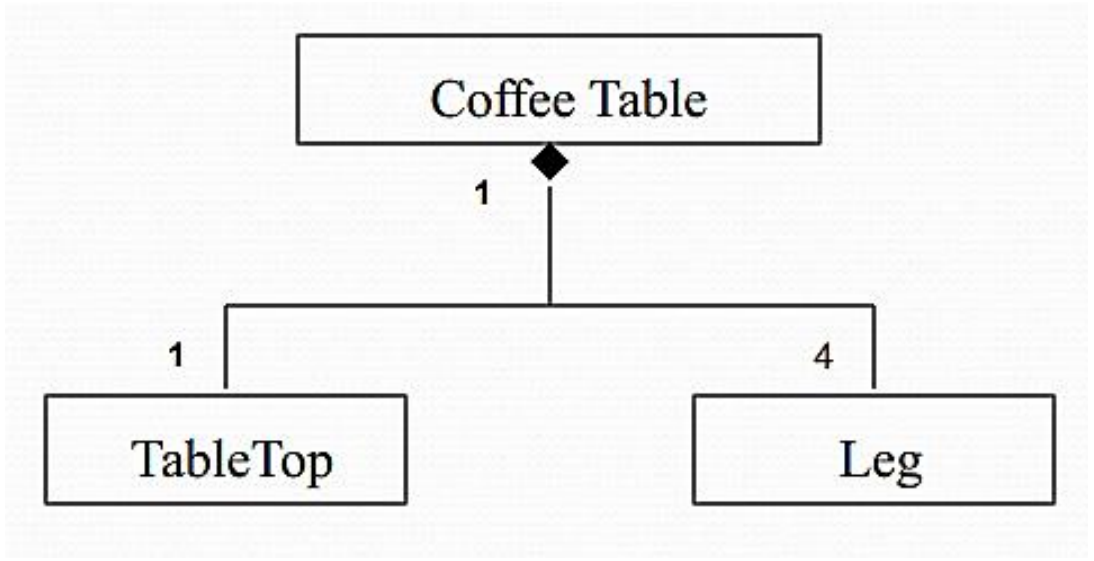


Whole　　*Part*

Composition

A <u>strong</u> form of aggregation

- The part object may belong to ***only one*** whole
- Multiplicity on the whole side ***must be zero or one***.
- The life time of the part is ***dependent*** upon the whole.
- The composite must manage the ***creation*** and ***destruction*** of its parts.



Thank You