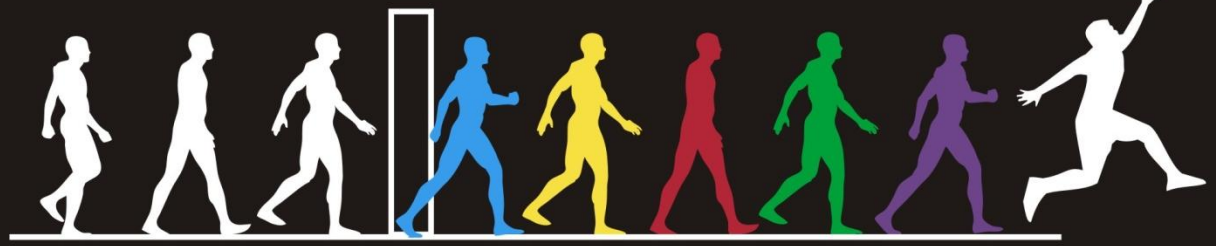
TGMC Technical Session

Bringing it All Together!



The Great Mind Challenge'11

Initiate Collaborate Innovate

Agenda

- **SRS Session**
- **Resources available in TGMC DVD kit**
- **Installation of all software's**
- **Small Application development**

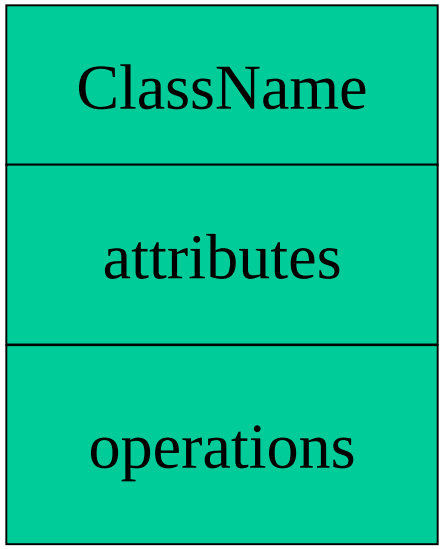
SRS Session

- **SRS Format**
- **Structural Diagrams**
  - Class Diagram
- **Behavioral Diagrams**
  - Sequence diagram
  - Use case model
  - Activity diagram
- **Database Diagrams**
  - ER diagram
  - Schema diagram

SRS Session

- **Tools and Technologies**
- **Sample SRS**

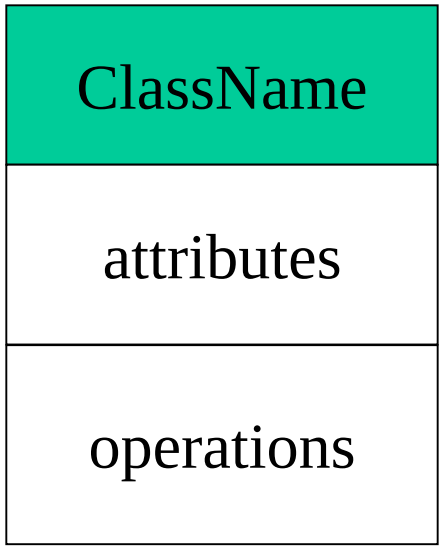
Classes



A *class* is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

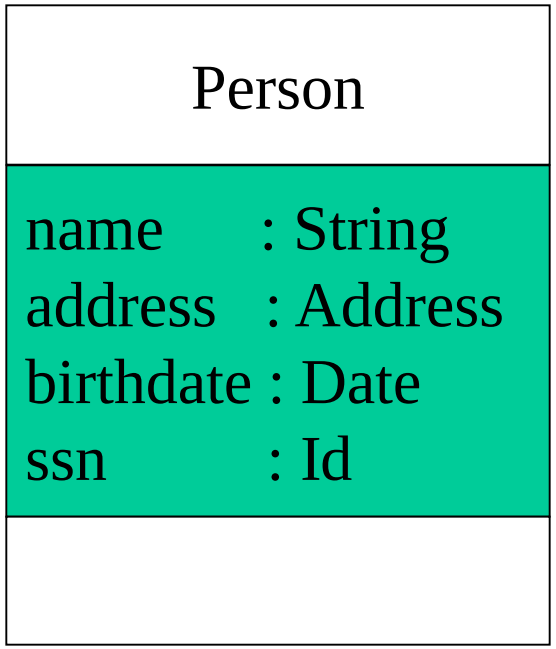
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartments.

Class Names



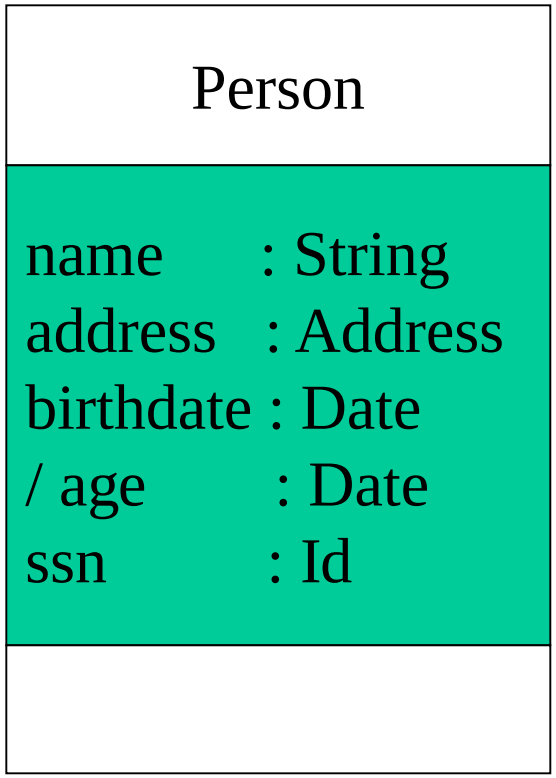
The name of the class is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

Class Attributes



An *attribute* is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment.

Class Attributes (Cont'd)



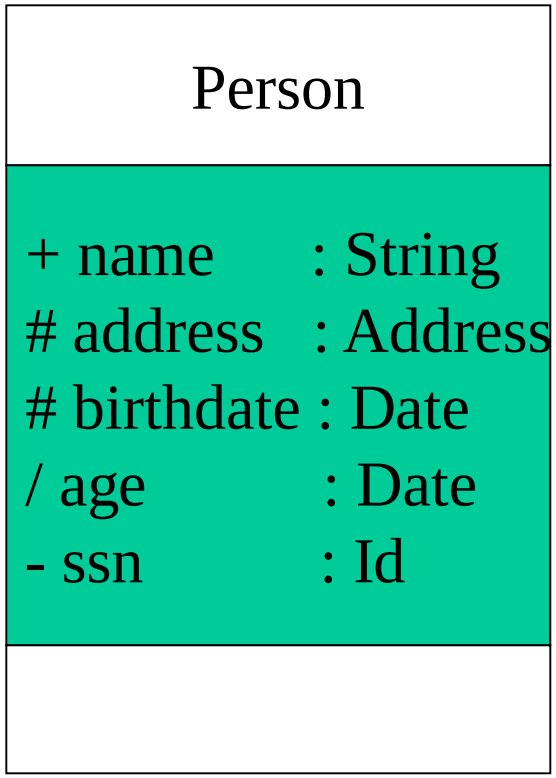
Attributes are usually listed in the form:

attributeName : Type

A *derived* attribute is one that can be computed from other attributes, but doesn't actually exist. For example, a Person's age can be computed from his birth date. A derived attribute is designated by a preceding '/' as in:

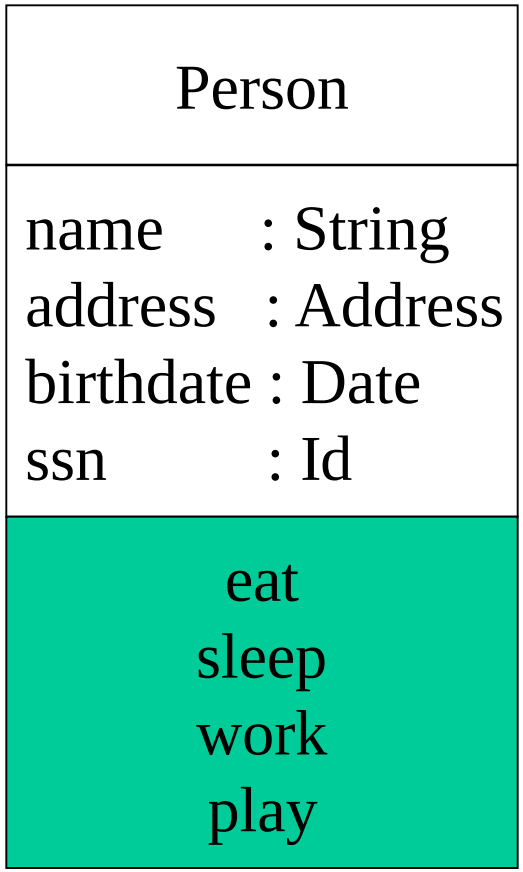
/ age : Date

Class Attributes (Cont'd)



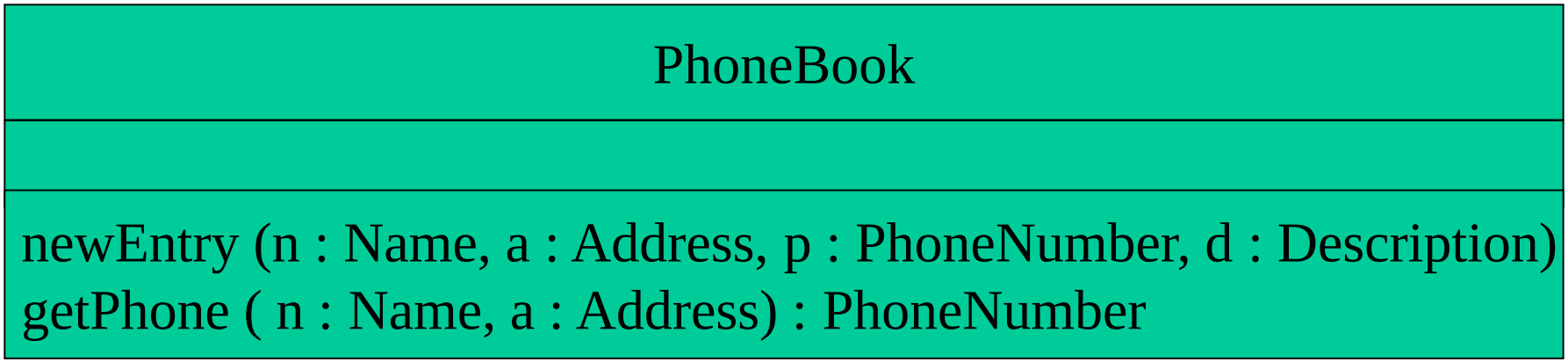
Attributes can be:
+ public
# protected
- private
/ derived

Class Operations



Operations describe the class behavior and appear in the third compartment.

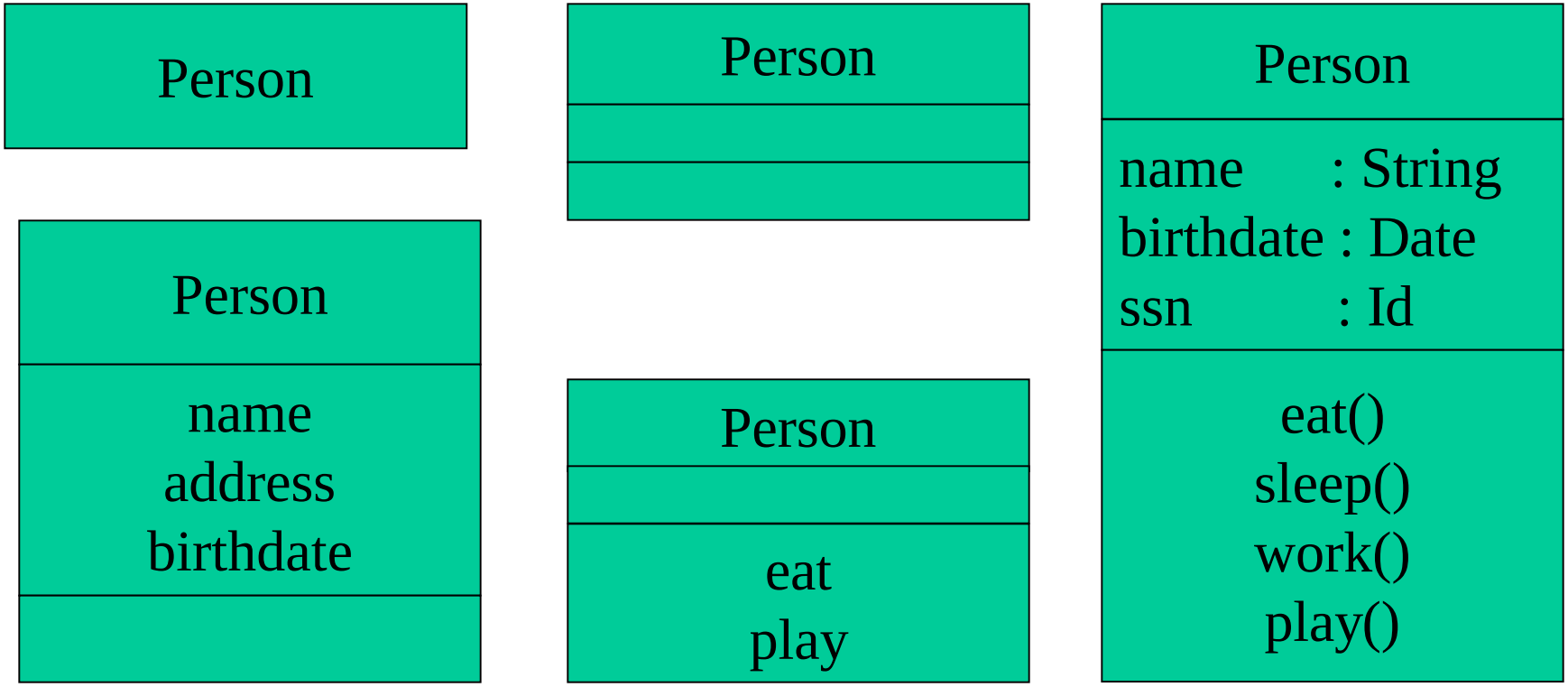
Class Operations (Cont'd)



You can specify an operation by stating its signature: listing the name, type, and default value of all parameters, and, in the case of functions, a return type.

Depicting Classes

When drawing a class, you needn't show attributes and operation in every diagram.



Relationships

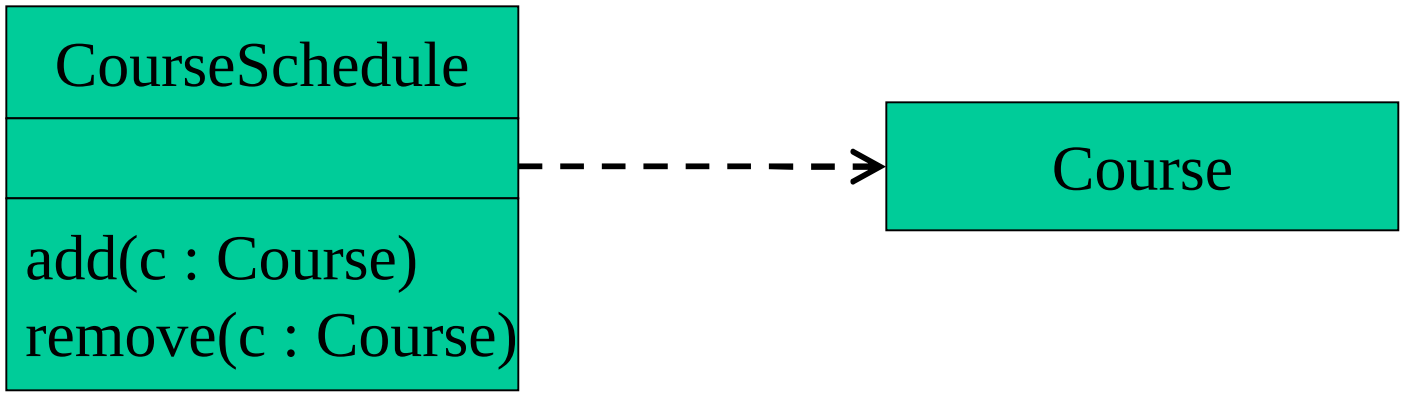
In UML, object interconnections (logical or physical), are modeled as relationships.

There are three kinds of relationships in UML:

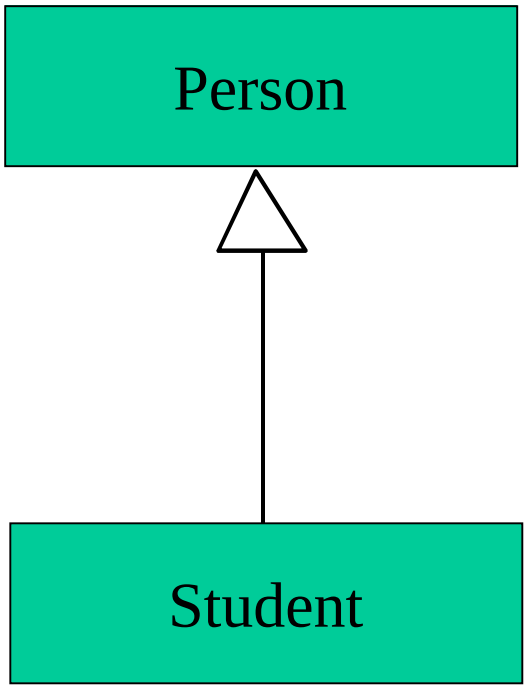
- dependencies
- generalizations
- associations

Dependency Relationships

A *dependency* indicates a semantic relationship between two or more elements. The dependency from *CourseSchedule* to *Course* exists because *Course* is used in both the **add** and **remove** operations of *CourseSchedule*.



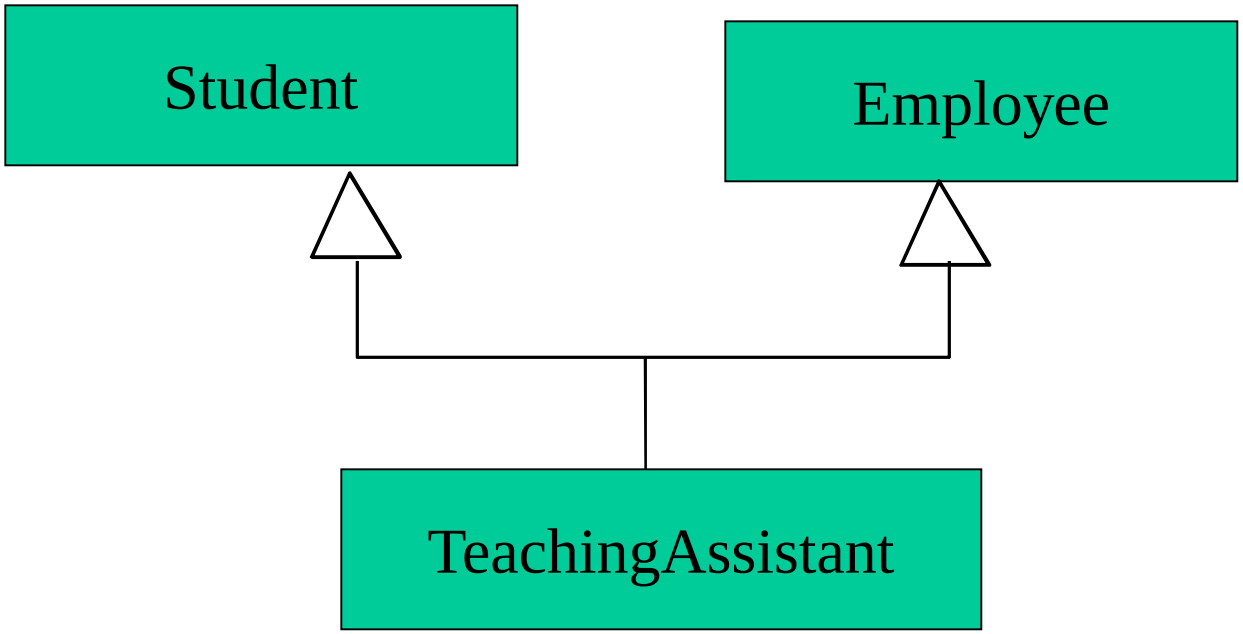
Generalization Relationships



A *generalization* connects a subclass to its superclass. It denotes an inheritance of attributes and behavior from the superclass to the subclass and indicates a specialization in the subclass of the more general superclass.

Generalization Relationships (Cont'd)

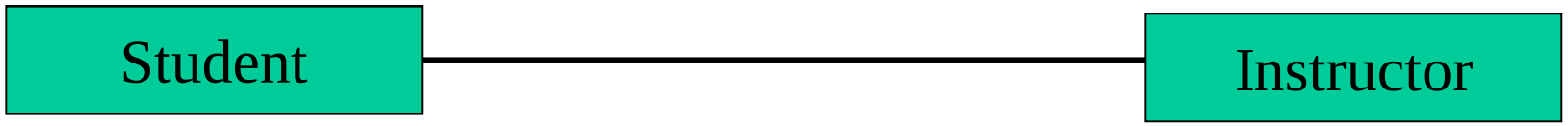
UML permits a class to inherit from multiple superclasses, although some programming languages (*e.g.*, Java) do not permit multiple inheritance.



Association Relationships

If two classes in a model need to communicate with each other, there must be link between them.

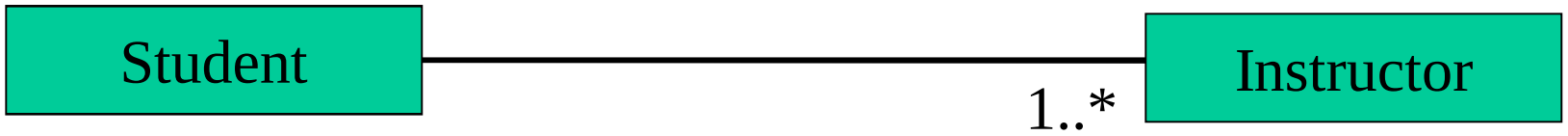
An *association* denotes that link.



Association Relationships (Cont'd)

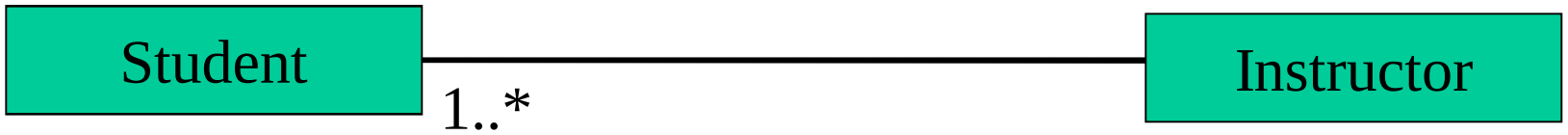
We can indicate the *multiplicity* of an association by adding *multiplicity adornments* to the line denoting the association.

The example indicates that a *Student* has one or more *Instructors*:



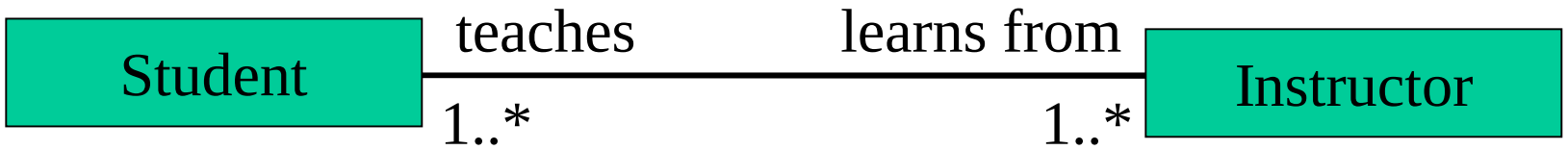
Association Relationships (Cont'd)

The example indicates that every *Instructor* has one or more *Students*:



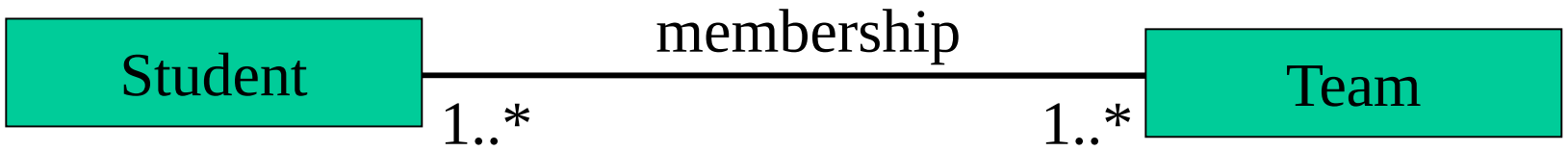
Association Relationships (Cont'd)

We can also indicate the behavior of an object in an association (*i.e.*, the *role* of an object) using *rolenames*.



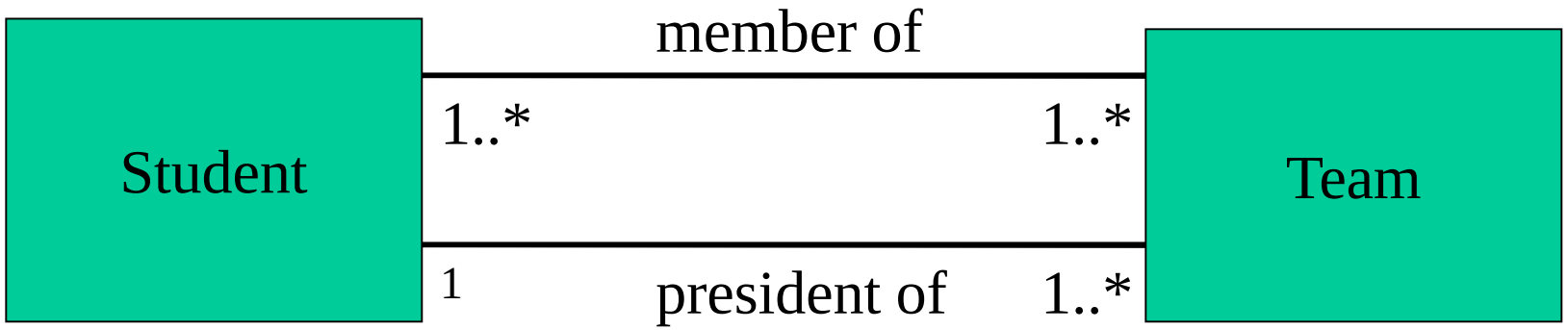
Association Relationships (Cont'd)

We can also name the association.



Association Relationships (Cont'd)

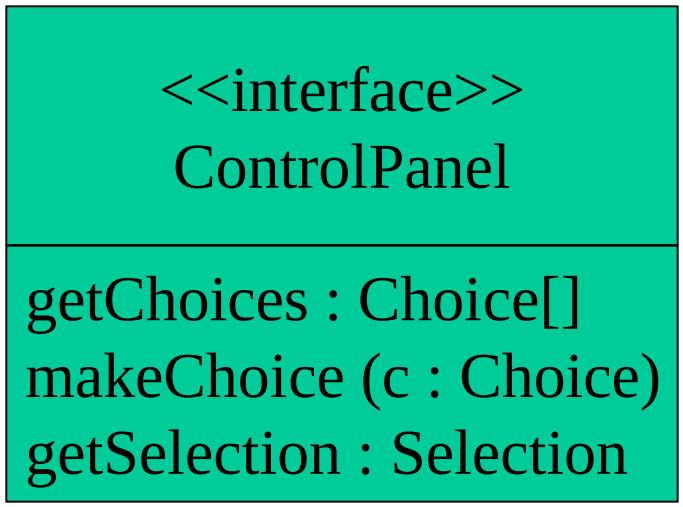
We can specify dual associations.



Interfaces

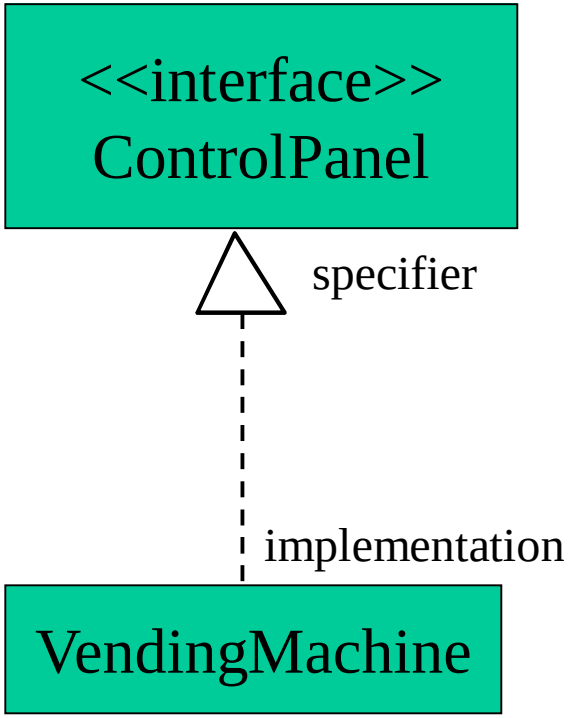
An *interface* is a named set of operations that specifies the behavior of objects without showing their inner structure. It can be rendered in the model by a one- or two-compartment rectangle, with the *stereotype* <<interface>> above the interface name.

Interface Services



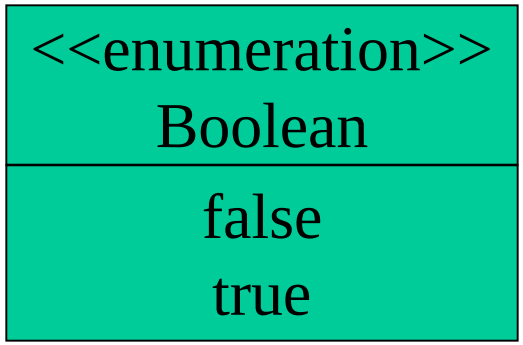
They specify the services offered by a related class.

Interface Realization Relationship



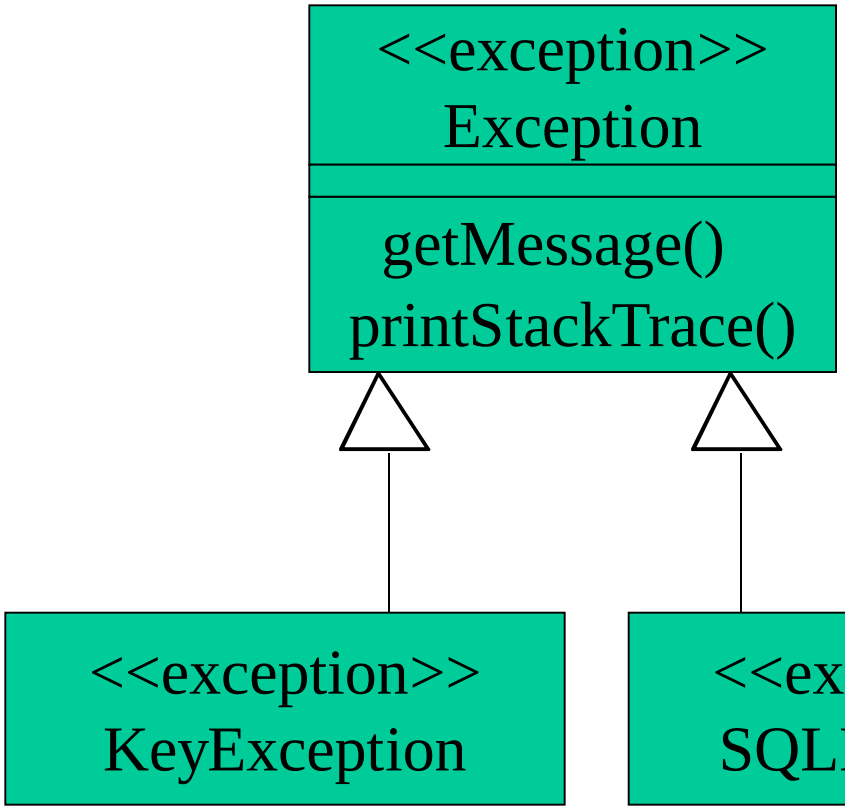
A *realization* relationship connects a class with an interface that supplies its behavioral specification. It is rendered by a dashed line with a hollow triangle towards the specifier.

Enumeration



An *enumeration* is a user-defined data type that consists of a name and an ordered list of enumeration literals.

Exceptions



Exceptions can be modeled just like any other class.

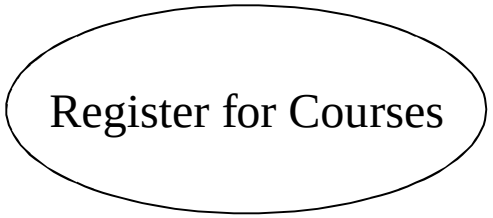
Notice the <<exception>> stereotype in the name compartment.

Use Case

"A *use case* specifies the behavior of a system or a part of a system, and is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor."
- *The UML User Guide, [Booch,99]*

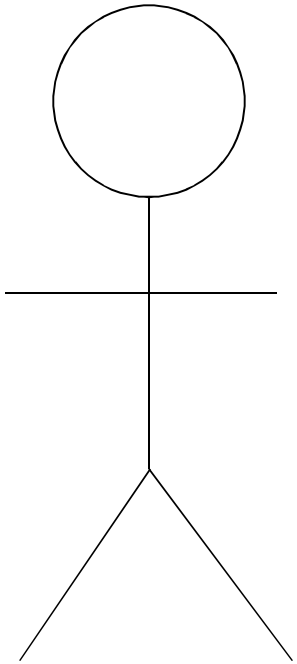
"An *actor* is an idealization of an external person, process, or thing interacting with a system, subsystem, or class. An actor characterizes the interactions that outside users may have with the system."
- *The UML Reference Manual, [Rumbaugh,99]*

Use Case (Cont'd)



A use case is rendered as an ellipse in a use case diagram. A use case is always labeled with its name.

Use Case (Cont'd)

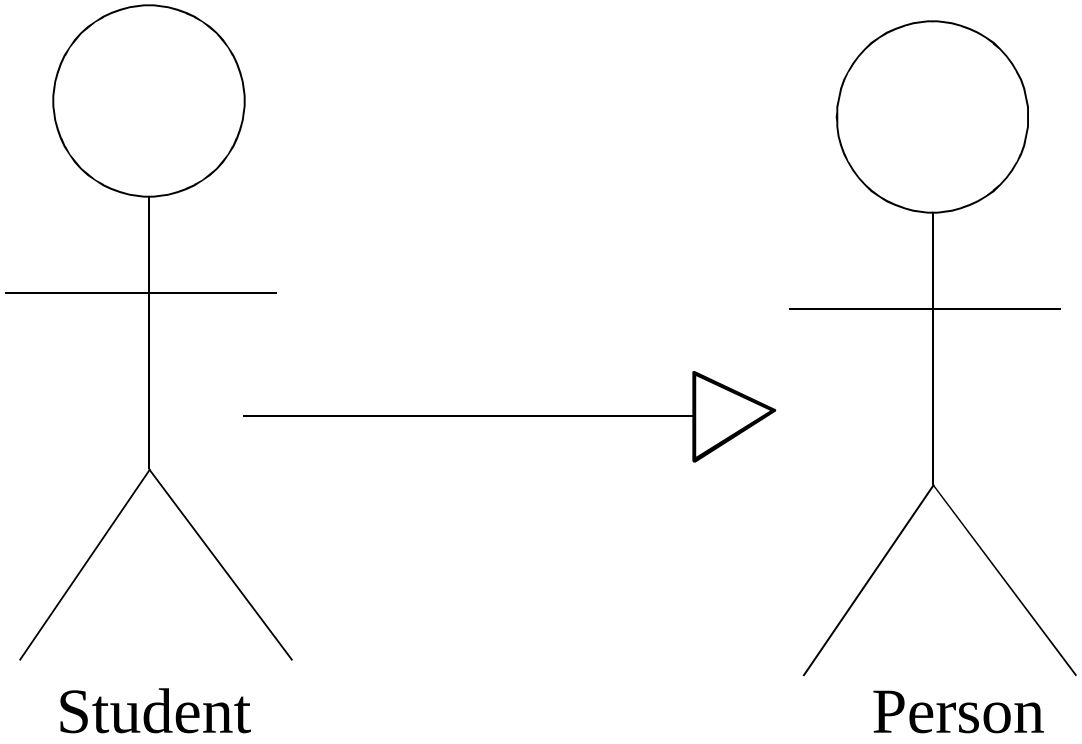


Student

An actor is rendered as a stick figure in a use case diagram. Each actor participates in one or more use cases.

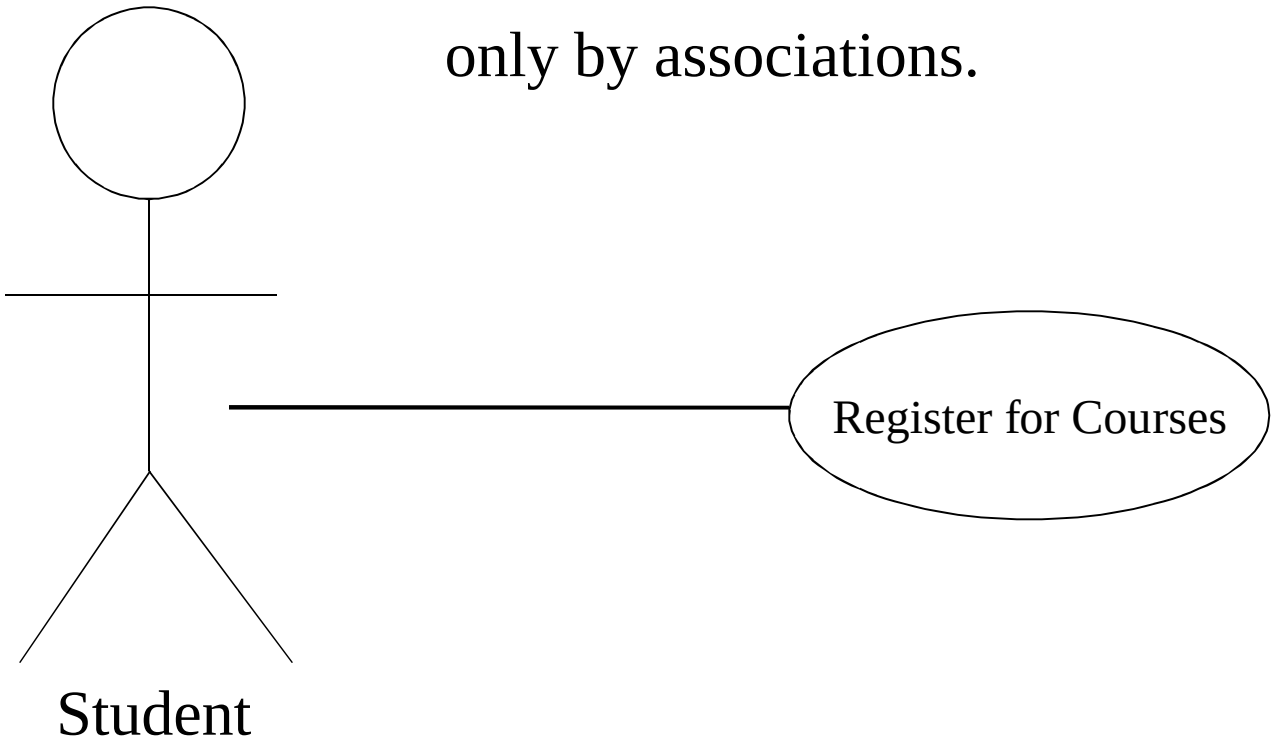
Use Case (Cont'd)

Actors can participate in a generalization relation with other actors.



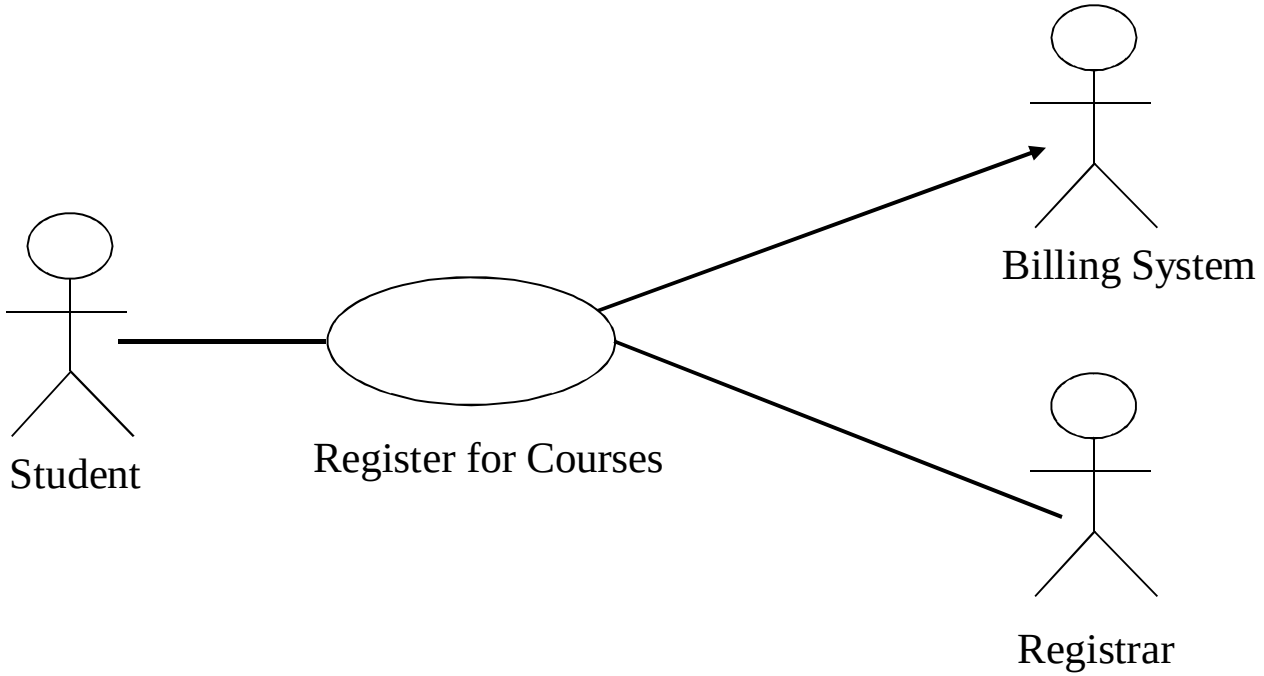
Use Case (Cont'd)

Actors may be connected to use cases only by associations.



Use Case (Cont'd)

Here we have a *Student* interacting with the *Registrar* and the *Billing System* via a "*Register for Courses*" use case.

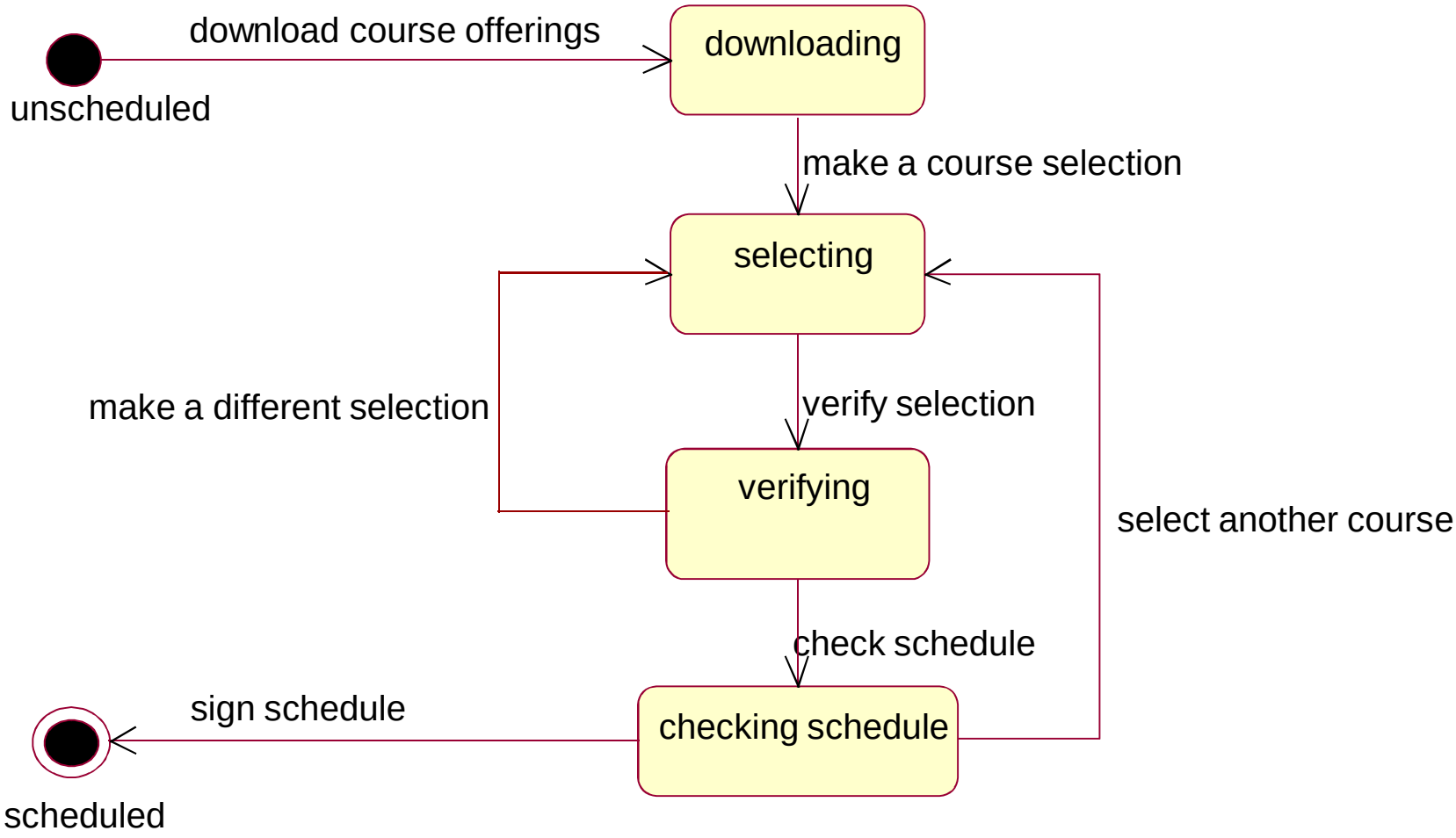


State Machine

An object must be in some specific state at any given time during its lifecycle. An object transitions from one state to another as the result of some event that affects it. You may create a state diagram for any class, collaboration, operation, or use case in a UML model .

There can be only one start state in a state diagram, but there may be many intermediate and final states.

State Machine

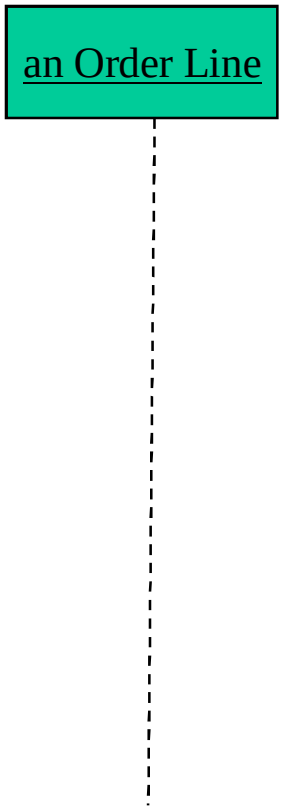


Sequence Diagram

A *sequence diagram* is an interaction diagram that emphasizes the time ordering of messages. It shows a set of objects and the messages sent and received by those objects.

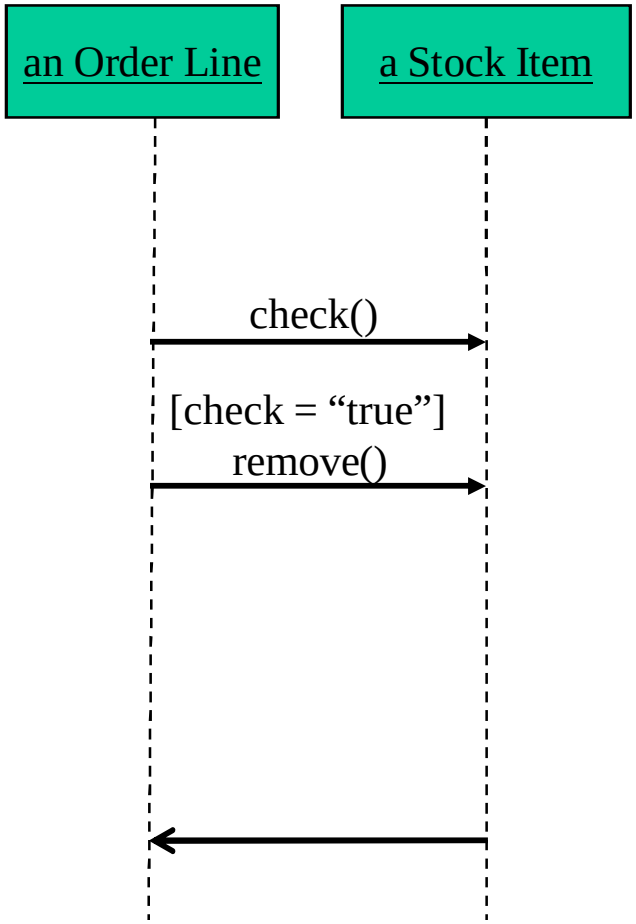
- *The UML User Guide, [Booch,99]*

Sequence Diagram



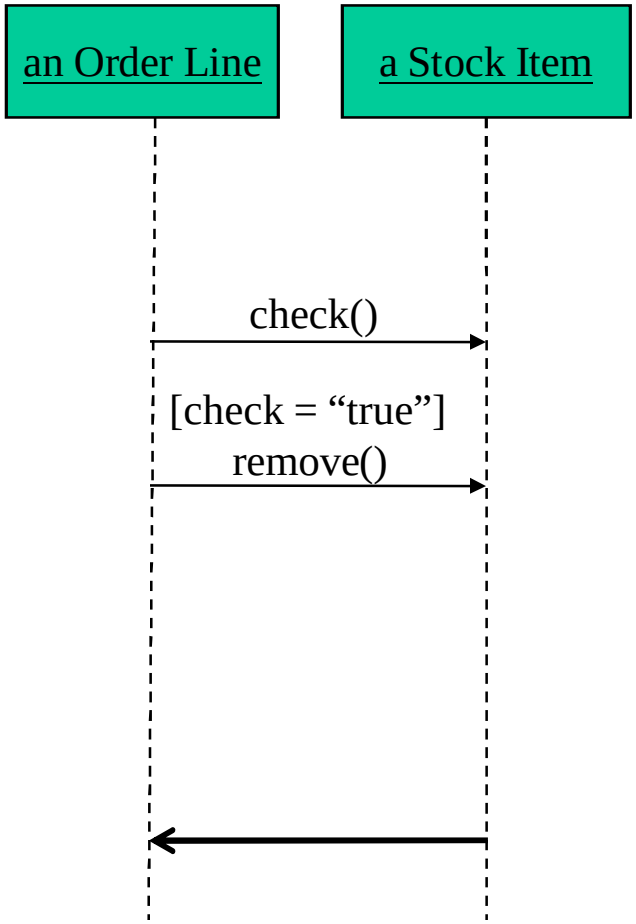
An object in a sequence diagram is rendered as a box with a dashed line descending from it. The line is called the *object lifeline*, and it represents the existence of an object over a period of time.

Sequence Diagram



Messages are rendered as horizontal arrows being passed from object to object as time advances down the object lifelines. Conditions (such as [check = “true”]) indicate when a message gets passed.

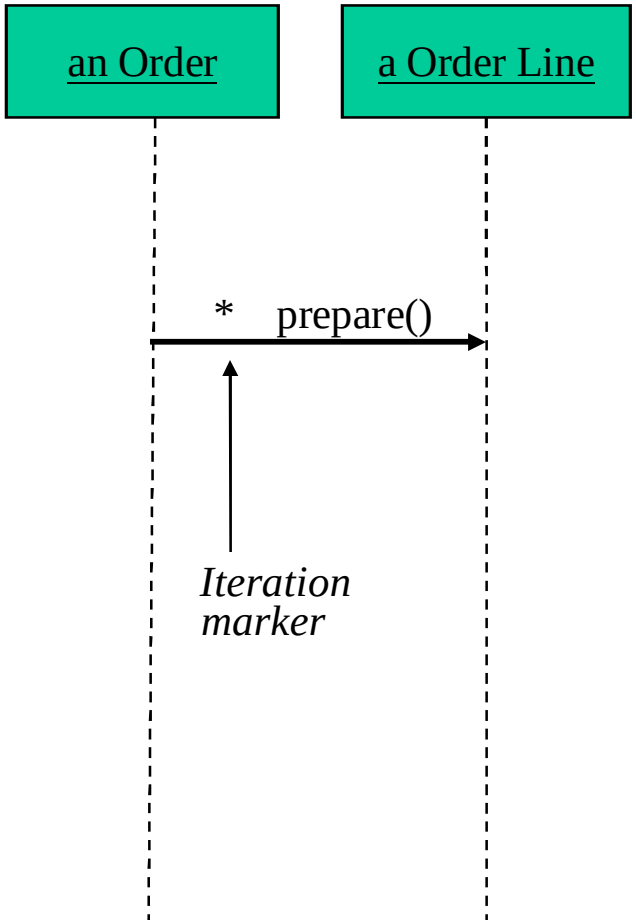
Sequence Diagram



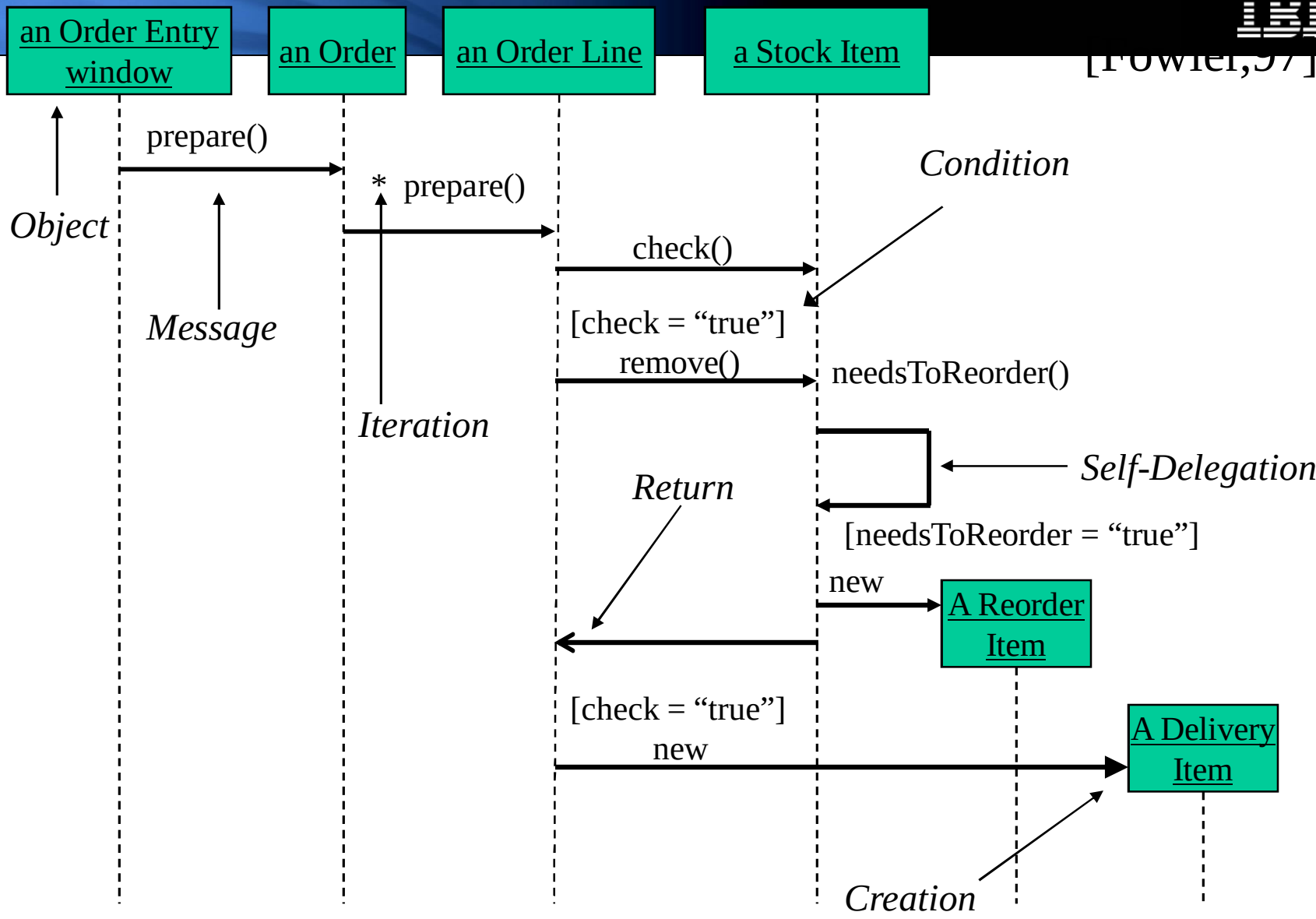
Notice that the bottom arrow is different. The arrow head is not solid, and there is no accompanying message.

This arrow indicates a **return** from a previous message, not a new message.

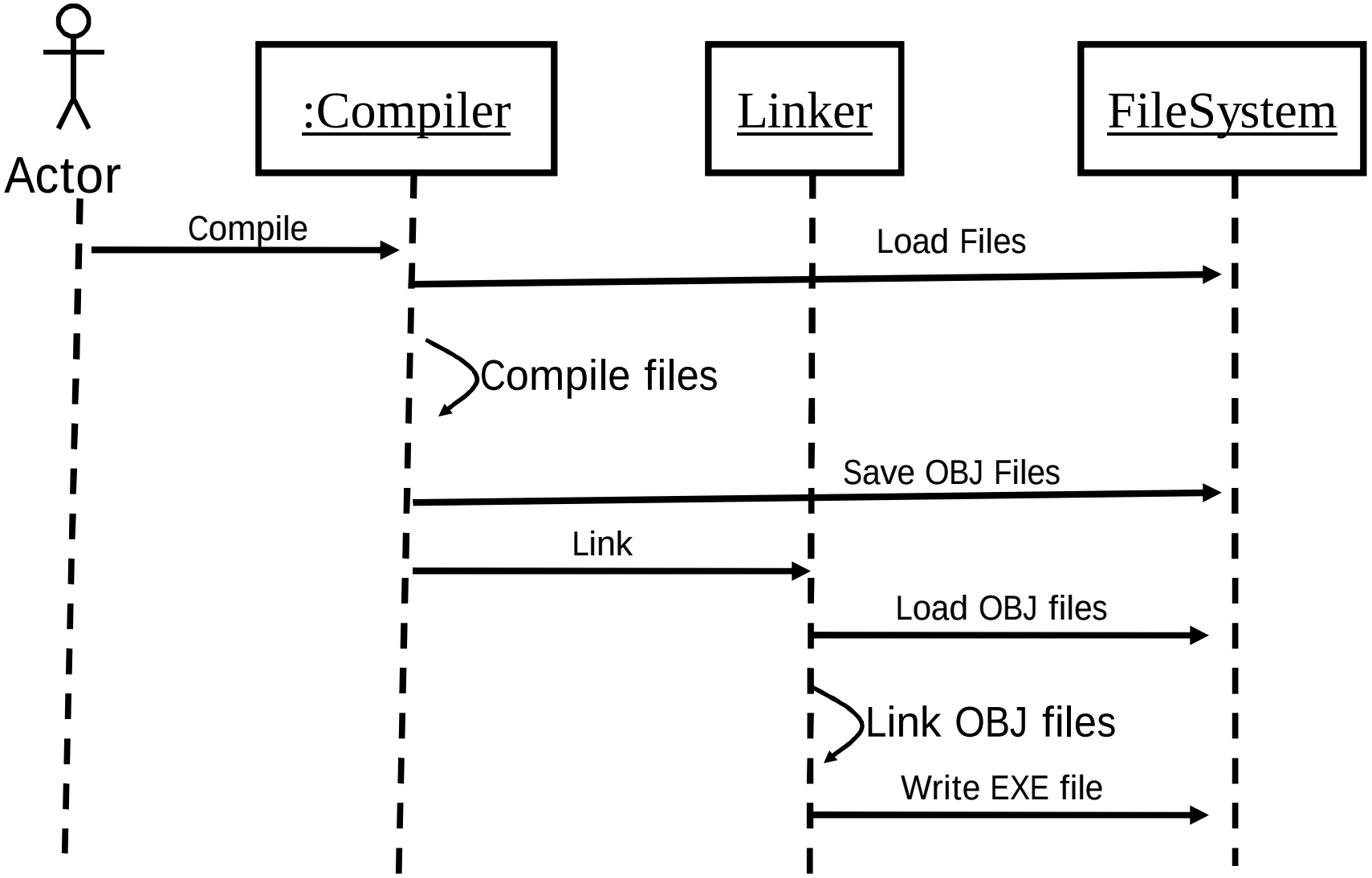
Sequence Diagram



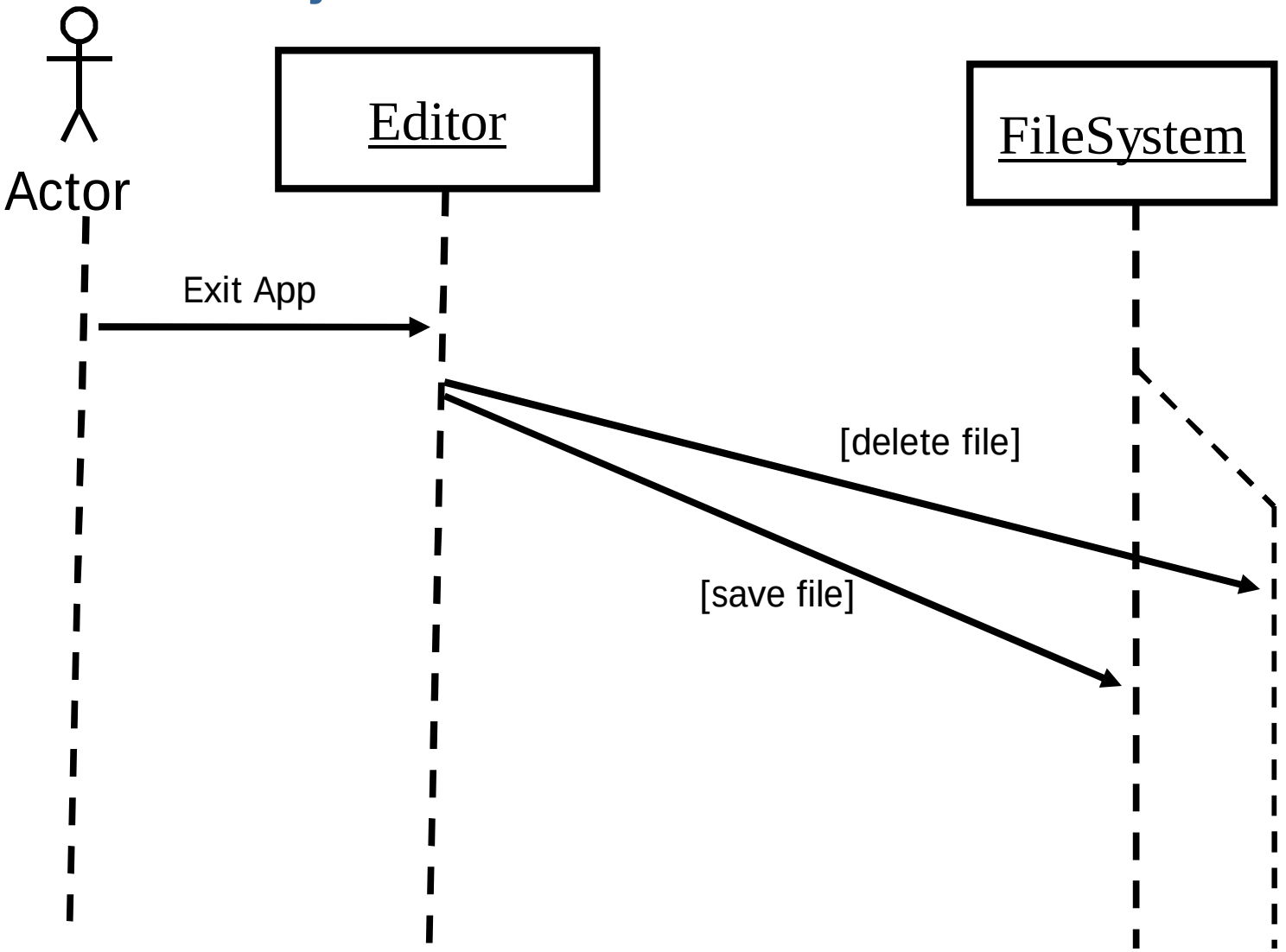
An iteration marker, such as * (as shown), or *[i = 1..n] , indicates that a message will be repeated as indicated.



Sequence Diagram – Compilation



Alternative Flow: flow changes to alternative lifeline branch of the same object

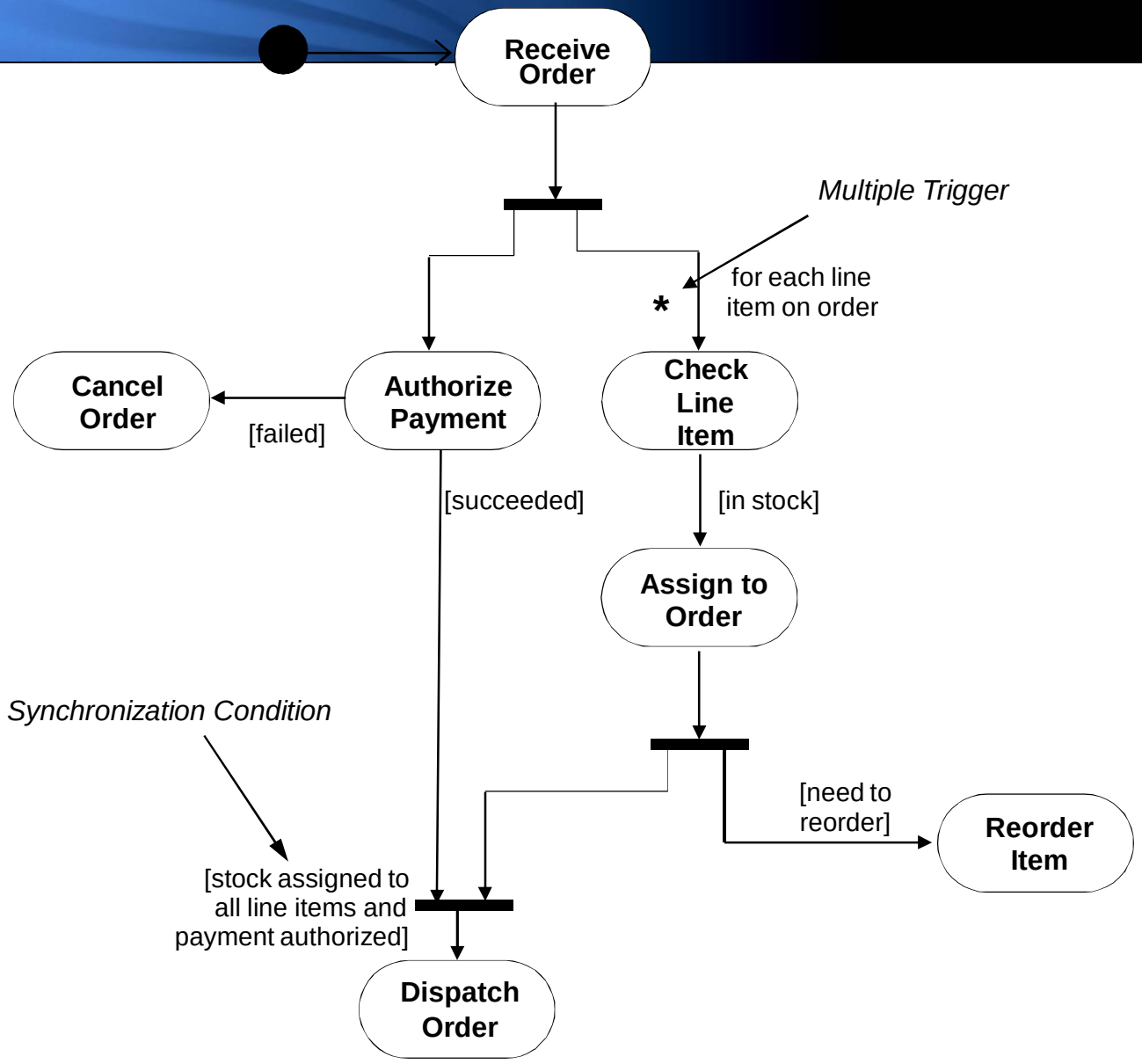


Activity Diagram

An activity diagram is essentially a flowchart, showing the flow of control from activity to activity.

Use activity diagrams to specify, construct, and document the dynamics of a society of objects, or to model the flow of control of an operation. Whereas interaction diagrams emphasize the flow of control from object to object, activity diagrams emphasize the flow of control from activity to activity. ***An activity is an ongoing non-atomic execution within a state machine.***

- *The UML User Guide, [Booch,99]*



[Fowler,97]

Database Diagrams

Introduction

- **Entity Relationship Modeling (ERM)**
  - a technique used to analyze & model the data in organizations using an Entity Relationship (E-R) diagram.

Definitions

- **Entity**
  - an aggregation of a number of data elements
  - each data element is an attribute of the entity
- **Entity type**
  - a class of entities with the same attributes
- **Relationship**
  - an association between two or more entities that is of particular interest

ERD Development Process

- Identify the entities
- Determine the attributes for each entity
- Select the primary key for each entity
- Establish the relationships between the entities
- Draw an entity model
- Test the relationships and the keys

A Simple Example

- **STUDENTs attend COURSEs that consist of many SUBJECTs.**
- **A single SUBJECT (i.e. English) can be studied in many different COURSEs.**
- **Each STUDENT may only attend one COURSE.**

Identify the entities

Any entity can be classified in one of the following categories:

- **Regular :**
  - any physical object, event, or abstract concept that we can record facts about.
- **Weak :**
  - any entity that depends on another entity for its existence.

Determine the Attributes

- **Every Entity has attributes.**
- **Attributes are characteristics that allow us to classify/describe an entity**
- **e.g., entity STUDENT has the attributes:**
  - student number
  - name
  - date of birth
  - course number

Key Attributes

- **Certain attributes identify particular facts within an entity, these are known as KEY attributes.**

- **The different types of KEY attribute are:**
  - Primary Key
    - Composite Primary Key
  - Foreign Key

Key Definitions

- **Primary Key:**
  - One attribute whose value can uniquely identify a complete record (one row of data) within an entity.
- **Composite Primary Key**
  - A primary key that consists of two or more attribute within an entity.
- **Foreign Key**
  - A copy of a primary key that exists in another entity for the purpose of forming a relationship between the entities involved.

ER Diagram Components

Every entity diagram consists of the following components:

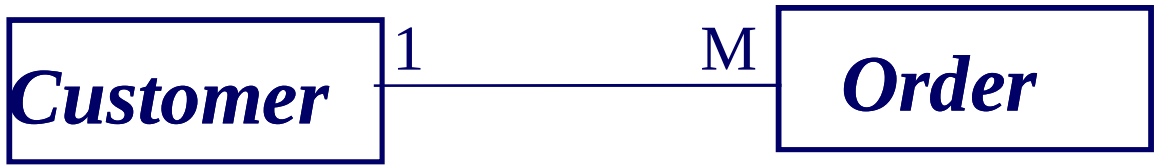
Entity (labelled box)

Relationship line

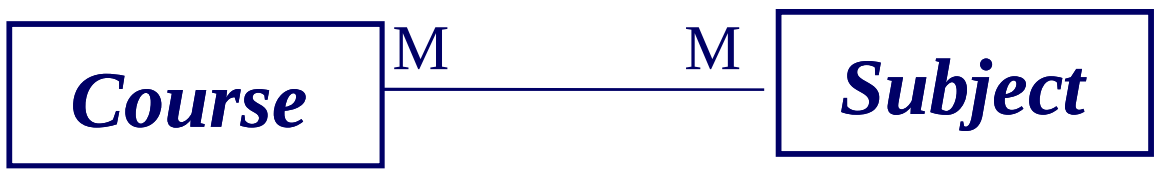
Degrees of a Relationship

One-to-one (1:1)

One-to-many (1:n)



Many-to-many (n:m)

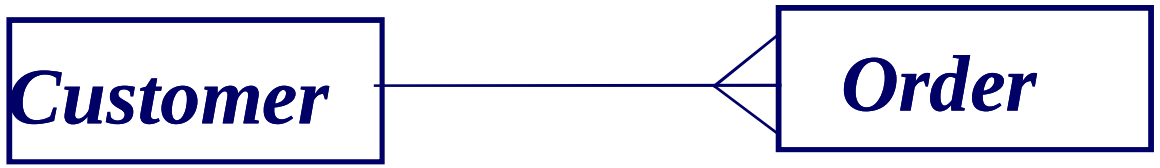


NOTE: Every many to many relationship consists of two one to many relationships working in opposite directions

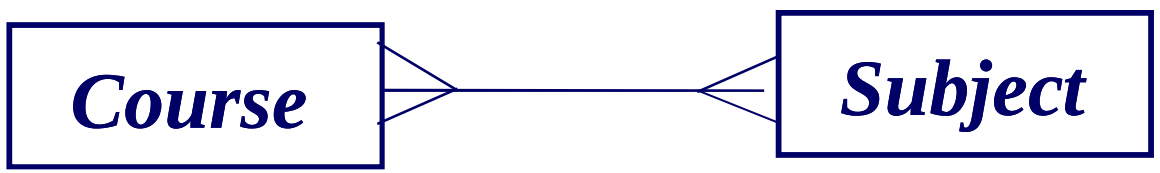
Degrees of relationship, alternative representation

One-to-one (1:1)

One-to-many (1:n)



Many-to-many (n:m)



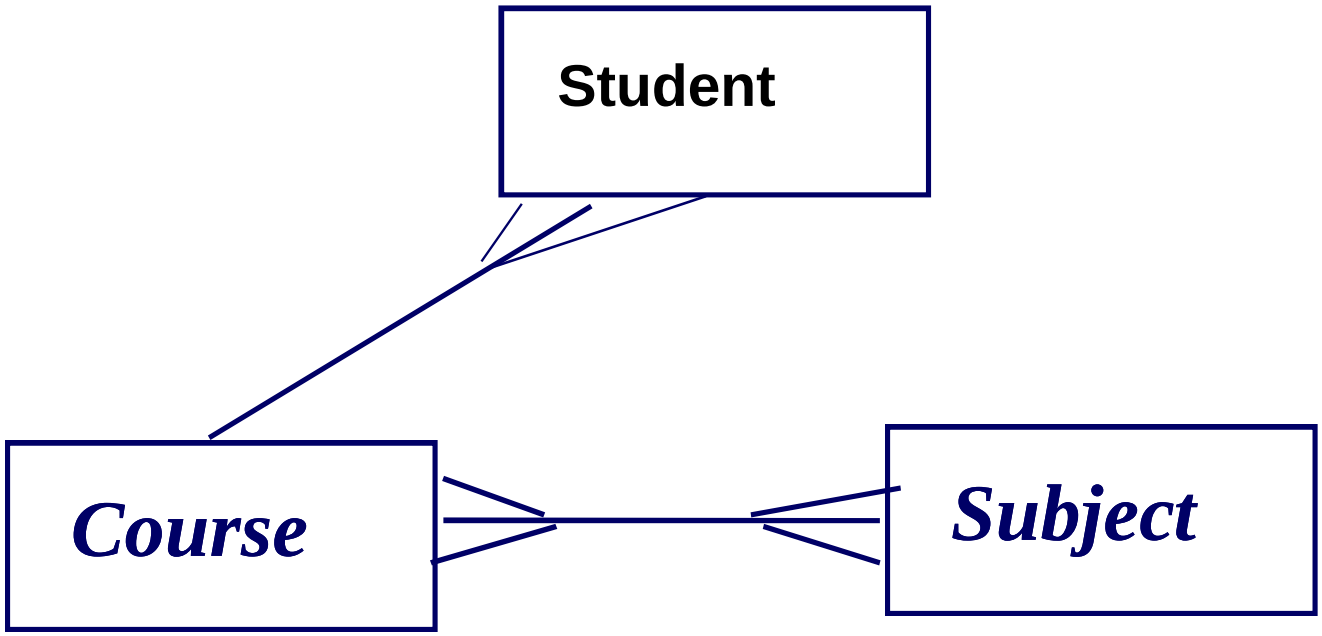
NOTE: Every many to many relationship consists of two one to many relationships working in opposite directions

Notation for optional attributes

A person must own at least one car. A car doesn't have to be owned by a person, but if it is, it is owned by at least one person. A person may own many cars.

- ○ optional relationship
- ● mandatory relationship

A Sample ER Diagram



A Student Record Entity Diagram

Summary of ER Diagram Designing

- Identify the entities
- Determine the attributes for each entity
- Select the primary key for each entity
- Establish the relationships between the entities
- Draw an entity model

Tools and Technologies

- **See Technical Resources page on TGMC website**

Best of Luck

For TGMC contest and Technical queries: tgmc@in.ibm.com

For SRS submission:

Save your SRS in PDF format.

Best of Luck

- **Ask your faculty mentor to register at**
  - www.ibm.com/in/university
    - For all the softwares and e-Books download