Unified Modling Language

- UML is a modeling language.
  - It is not a programming language.

- Why we model software system before we build them?
  - Modern software system are typically large and complex
  - Software has to be integrated with other systems
  - Modeling allows architects and developers
    - To visualize entire system
    - To reason about aspects of those system
    - To consider alternative and evaluate risks
    - To communicate design

- Models are not the same as the thing they represents
  - They are abstraction of some features of a system with a certain purpose.

- UML contains data about all the element of the model and there relationship.

UML Diagram

Structural

- Object
- Class
- Component
- Deployment
- Package
- Composite structure

They all show static structural aspects of the system being modeled

Behavioural

- Use case
- Sequence
- Communication
- Interaction overview
- Timing
- State machine
- Activity

Shows the aspects of the system that reflects behavior that are executed usually in response to a series of events

Frame → Frames are used to contain diagram.

Classifier

→ When you create a model in uml, the types of thing which you can put into that model are defined in a meta model called classifier

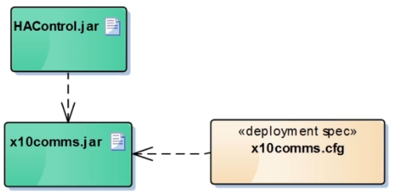
→ Classifiers have features and properties. This may be structural or behavioral.

Comments

- Comments/ Notes are texual annotations that can be applied to pretty much anything in a diagram.
- Constraints are shown in curly braces, usually inside a note symbol.
- They are connected to model element by a dashed line.

Dependencies

- It means that one element in a model is dependent on another.
- They are shown as an arrow with open arrow head.
- They may also be shown with a stereotype to indicate that they have a specific purpose.



Stereotypes

- It is a applied to a model element to show that it is extending the meaning of that type of element for some specific purpose in a system or domain.
- It is ahown by in guillements «stereotype»

★ Class diagram

Purpose of class diagram

① Model the concepts that are used in the business, in the language used to reffer to them.

Business Domain　Problem Domain　Solution Domain

- It helps to clarify the vocabulary of the business.
- A class diagram like this is called the domain model.

② Analyse usecases and identify the classes required to implement each usecase.

- A common use of class diagram in the early stages of a project is to model the classes that collabrates to deliver the functionality of a particular usecase.
- These will usually includes the classes of the domain model together with classes that manages interaction with the user and display of a user interface.
- The class diagram from a set off use cases can be used to build up a picture of all the required classes.
- This approach is known as Robustness analysis.

3 To model the overall structure of the system in terms of the classes interfaces and there relationships.

4 Reason about the system and make design decisions about how it will work.

5 Provide the basis for implementation in an object oriented language.

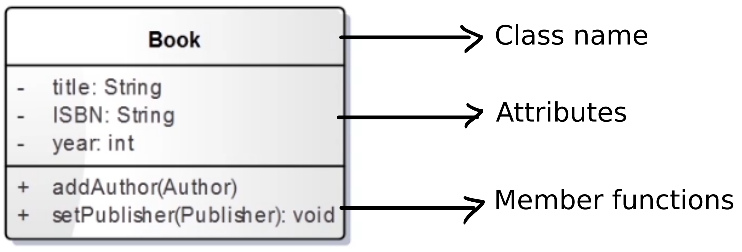
⇒ Class diagram allows us to reason about the problem and its solution.

⇒ Class diagram form the basis for the implementation in a programming language.

Basic Notations of Class Diagram

⇒ Simplest notation of a class is a rectangle with the name of the class at the top.

⇒ Class may be divded up into compartments



Associations

→ Relationship between classes in the model that reflects relationship in the real world.

→ Simplest notation of an association is a solid line drawn between two classes

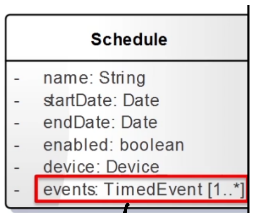
→ Association may have a name. Name is usually a verbe phrase that allows us to read the meaning of the association.

Attributes and Operations

Attributes
- Visibility
- Type
- Initial value
- Derived attributes

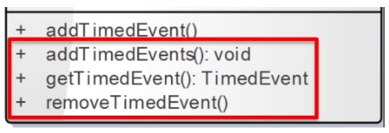
Operations
- Visibility
- Return values
- Parameters
- Getters and setters

- The visibility of an attribute can be + public, - private, # protected or ~ package
- Normally attributes are private, or protected if the class is part of an inheritance hierarchy
- The attribute type is separated from the name by a colon
- The type can be a primitive type, a class from a library or a class from the same system
- If there can be multiple values, the multiplicity is shown in square brackets

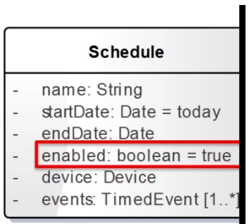


This indicates that there must be atleast 1 timed events but there is no fixed upper bound.

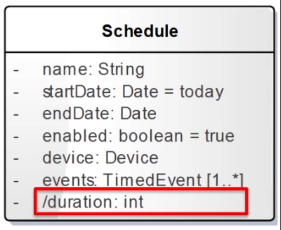
- The visibility of an operation can be + public, - private, # protected or ~ package
- Normally operations are public, or protected if the class is part of an inheritance hierarchy
- The type of the return value from an operation is shown separated from the operation by a colon
- If there is no return value, it can be left blank, or you can use the word void
- Name and type of each parameter are in the model
- Name and type are separated by a colon



- The default or initial value that should be assigned to an attribute when an object of that class is created can be shown after the type, separated from it by an equals sign


Schedule
- name: String
- startDate: Date = today
- endDate: Date
- enabled: boolean = true
- device: Device
- events: TimedEvent [1..*]

- Sometimes attribute values can be derived from other attributes, for example, a person's age from their date of birth
- These are shown with a slash before the name of the attribute


Schedule
- name: String
- startDate: Date = today
- endDate: Date
- enabled: boolean = true
- device: Device
- events: TimedEvent [1..*]
- /duration: int

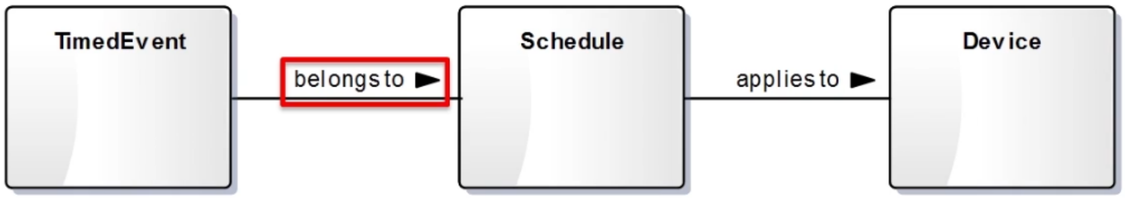
- Successive parameters in the list are separated by commas
- Display of names and types is optional
- Operations to get the value and set the value of attributes are known as getters and setters

Association

Association Name

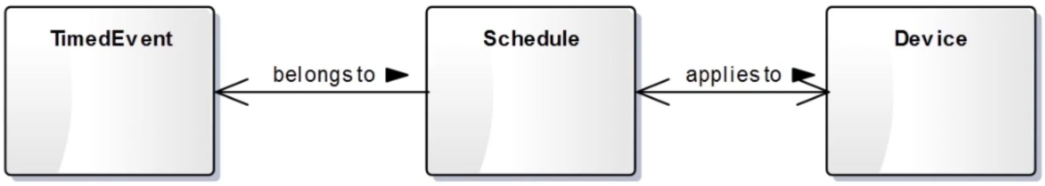
→ It can be shown as a label next to the line representing that association

→ The direction in which that name is to be read can be shown by a small arrow next to the label.



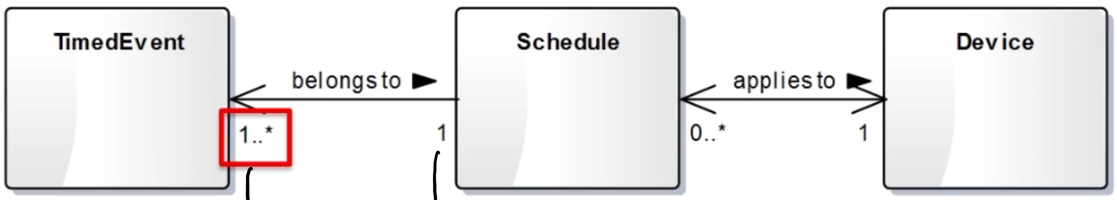
{Example}

⇒ Association end can have open arrow. This shows that the association is navigable from the class at the opposit end.



{Example}

⇒ Multiplicity of the association end can be shown with a label.



{Example}

→ This indicate each TimedEvent must belong to exactly 1 Schedule.

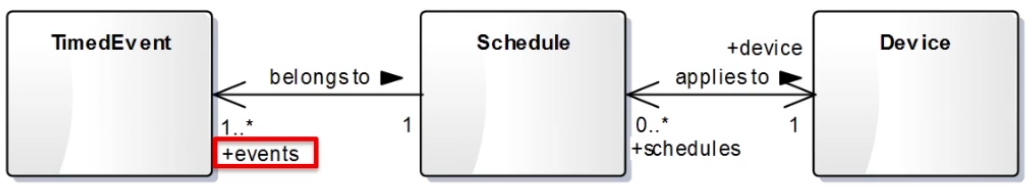
→ This indicate that Schedule have atleast 1 or more timed event associated to it.

- Format of multiplicity labels

1	exactly one	*	same as 0..*
0..1	zero or one (optional)	1..7	a specific range
1..*	one or more	1,3,5	a list of values
0..*	zero or more	1,3,7..10	a list of values and ranges

Association Role

Each end of an association has a role name, which defaults to the name of the class at that end with an initial lower case letter



{Example}

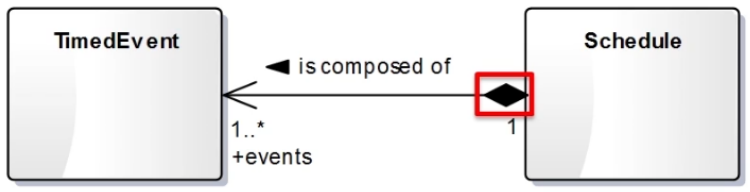
Composition and Aggregation

⇛ Both of these representation relationship where objects of one class are made up of one or more other classes.

Composition

Composition represents a stronger whole-part relationship

If a class is composed of others, deleting an instance of the whole results in deletion of the instances of the part.

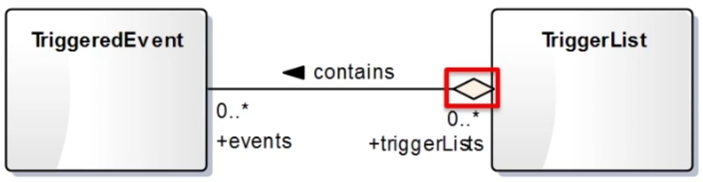


{Example}

Aggregation

Aggregation is the weeker whole-part relationship.

If a class is an aggregation of others, deleting instances of the whole leaves the instances of the part untouched.



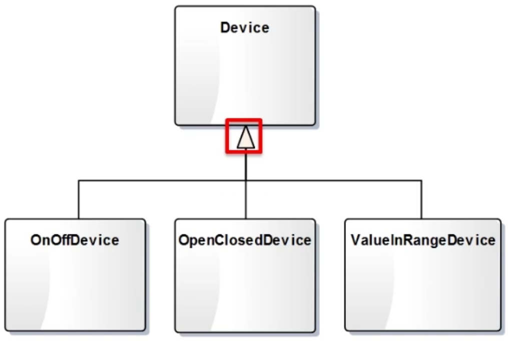
{Example}

Generalization and Specialization

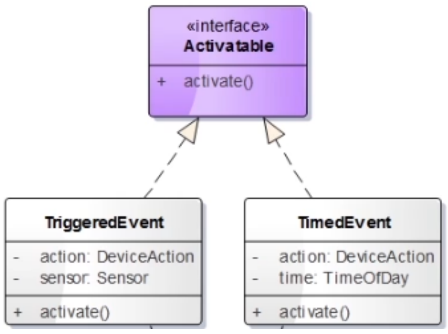
⇛ Generalization and Specialization are technical term for inheritance.

- Each subclass is a specialization of the superclass
- The superclass is a generalization of the subclasses

- A generalization/ specialization relationship is shown with an unfilled triangle at the superclass end of the relationship

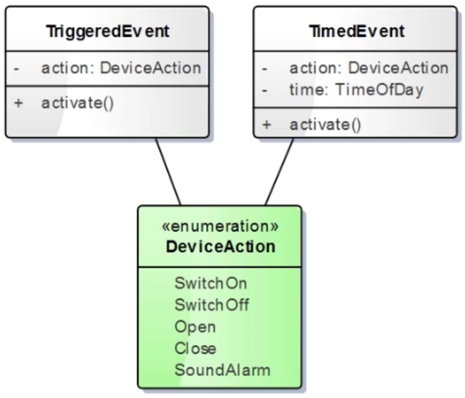


Interface declares operations that must be implemented by classes that realize the interface.



The name of the interface has the stereotyped <<interface>> above it.

Enumeration can be shown as bellow.


TriggeredEvent
- action: DeviceAction
+ activate()
TimedEvent
- action: DeviceAction
- time: TimeOfDay
+ activate()
«enumeration»
DeviceAction
SwitchOn
SwitchOff
Open
Close
SoundAlarm