Modeling with UML
Chapter 2, part 2

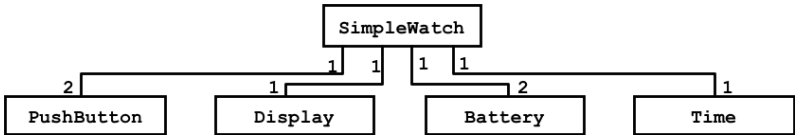
CS 4354
Summer II 2015

Jill Seaman

Class diagrams

- Used to describe the internal structure of the system.
- Also used to describe the application domain.
- They describe the system in terms of
  - ✦Classes, an abstract representation of a set of objects
  - ✦Attributes, properties of the objects in a class
  - ✦Operations that can be performed on objects in a class
  - ✦Associations that can occur between objects in various classes

Class diagram for a simple watch

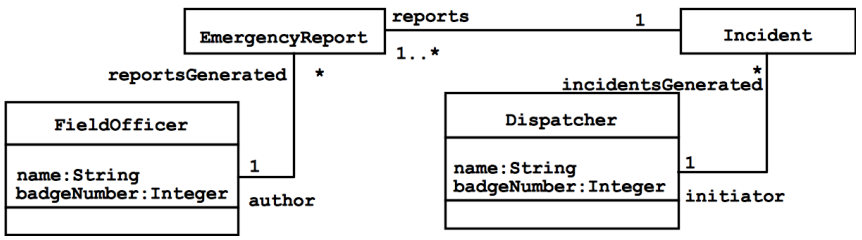


- Boxes are classes
- Lines show associations (between objects)
- Numbers show how many objects must be associated
- This diagram does not show attributes or operations

Class Diagrams: details

- Classes are boxes composed of three compartments:
  - ✦Top compartment: name
  - ✦Center compartment: attributes
  - ✦Bottom compartment: operations
- Lines between classes represent associations between classes
  - ✦These can have role names and multiplicity constraints
- Conventions:
  - ✦Class names start with uppercase letter
  - ✦Attributes can (and should) have types specified
- Attributes and Operations are sometimes omitted for simplicity

UML class diagram: **classes** that participate in the ReportEmergency use case.



Associations and links

- A link is some connection between two objects.
- Associations are relationships between classes and represent the fact that links may (or do) exist between object instances.
  - ✦Associations are noted with a line between the boxes.
- Associations can be symmetrical (bidirectional) or asymmetrical (unidirectional).
  - ✦Unidirectional association is indicated by using a line with an arrow
  - ✦The arrow indicated in which direction navigation is supported.
  - ✦If the line has no arrows, it's assumed to be bidirectional.

Roles

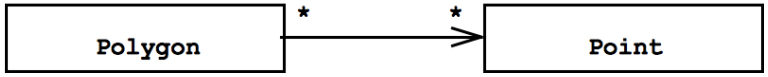
- Each end of an association can be labeled by a role.
- Allows us to distinguish among the multiple associations originating from a class.
  - ✦An employee can belong to a department and be the head of the department.
- Roles clarify the purpose of the association.
- If there is no role name specified, you can use the name of the class at the unspecified end.
- Previous diagram:
  - ✦The FieldOfficers who generate reports are called authors (or the author of the report).

Multiplicity

- Multiplicity: a set of integers labeling one end of an association
- Indicates how many links can originate from an instance of the class at the other end of the association.
- This is generally an upper bound.
- * is shorthand for 0..n, called "many"

- Most associations belong to one of these three types:
  - ✦A **one-to-one** association has a multiplicity 1 on each end.
  - ✦A **one-to-many** association has a multiplicity 1 on one end and 0..n or 1..n on the other.
  - ✦A **many-to-many** association has a multiplicity 0..n or 1..n on both ends.

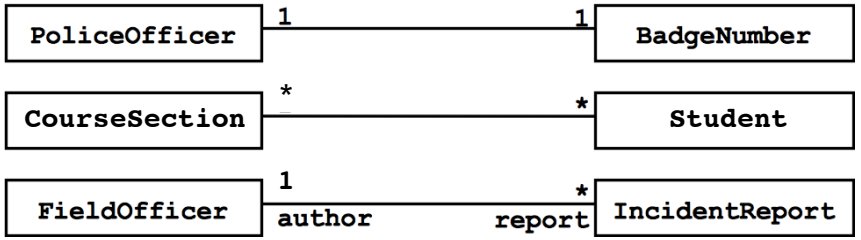
Example of a unidirectional association



This system supports navigation from the Polygon to the Point, but not vice versa. Given a specific Polygon, it is possible to query all Points that make up the Polygon. But a given Point does not know which Polygon(s) it belongs to.

*Note: the diagram in the book is wrong, it has two arrows.

Examples of multiplicity

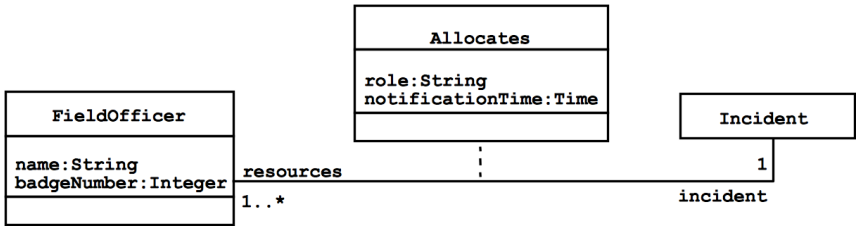


A CourseSection contains many Students
A Student is enrolled in many CourseSections

How many reports can a FieldOfficer write?
How many authors of a report can there be?

Association class

- Association class: an association with attributes and/or operations
- Depicted by a class symbol that contains the attributes and operations and is connected to the association symbol with a dashed line.

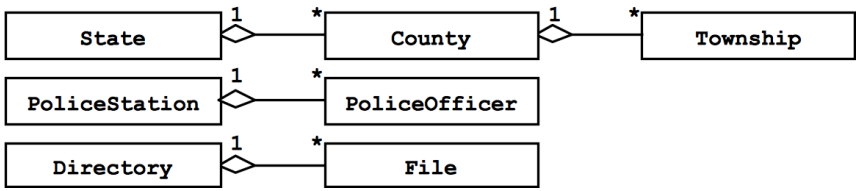


These kinds of classes are not commonly used.

Aggregation and Composition

- Aggregation is a kind of association that specifies a whole/part relationship between the aggregate (whole) and component part.
  - ✦Useful to denote hierarchical relationships (directory contains files)
  - ✦Specified with an open diamond on the aggregate (whole) side.
- Composition is a special case of aggregation where the composite object has sole responsibility for the life cycle of the component parts.
  - ✦The composite is responsible for the creation and destruction of the component parts.
  - ✦An object may be part of only one composite.
  - ✦Specified with a closed diamond on the composite (whole) side.

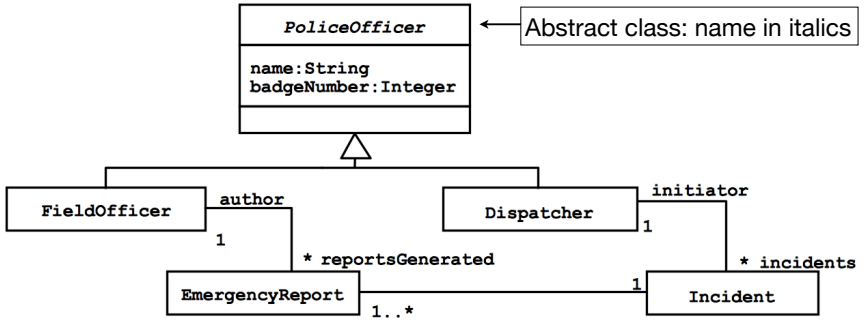
Examples of a aggregations



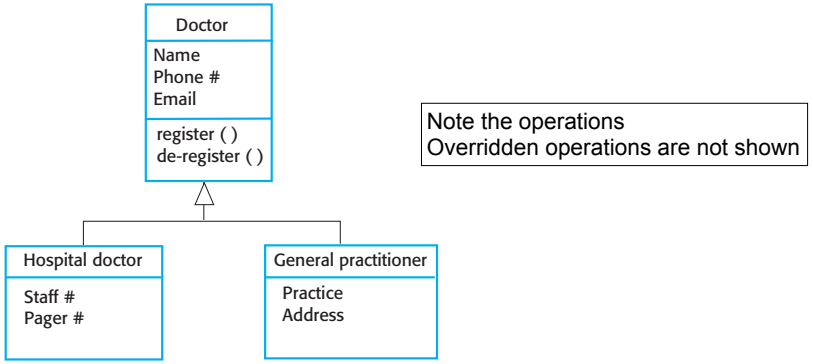
Could any of these be composites?
- can a County belong to more than one State?
- can a County exist without a State?

Inheritance (or generalization)

- Inheritance is a relationship between a base class and a more refined class.
  - ✦ the refined class has attributes and operations of its own, as well as the attributes and operations of the base class (it inherits them).

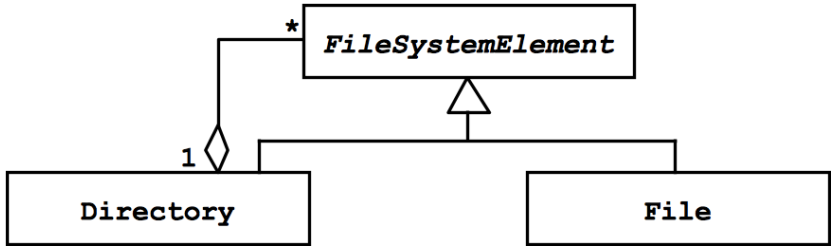


Inheritance example



Note the operations
Overridden operations are not shown

Example of a hierarchical file system



Associations are not attributes!

- Do not add an attribute to a class to represent the end of an association already in the diagram
  - ✦Among other problems, this is redundant
- Associations in a diagram do NOT specify how they should be implemented
- There are various ways to record in an implementation the fact that one object is related to another.
  - ✦One class may contain a reference to an object it is associated with
  - ✦One class may contain a list of the objects it is associated with
  - ✦The class may run a method/function to determine which objects it is associated with.

Attributes are like associations

- A Customer's name attribute indicates that a Customer has a name
  - ✦The name type might even be a Class, like a String
- The attribute types are usually small, simple classes
- The attribute usually has only one value, often its own copy

- In UML, the syntax of an attribute is:
  - ✦ *visibility name : type = defaultValue*
  - ✦visibility, type and defaultValue are optional
  - ✦visibility: + (public), # (protected), or - (private)

Operations

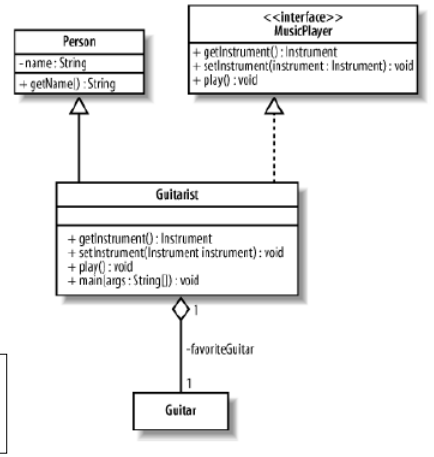
- An operation is a process that a class knows how to carry out
  - ✦Normally we don't show the setters and getters, these can often be inferred
- In UML, the syntax of an operation is:
  - ✦ *visibility name (parameter-list) : return-type*
  - ✦visibility: + (public), # (protected), or - (private)
  - ✦parameter-list contains comma separated parameters, with syntax like attributes.
- An operation is something that is invoked on an object (like a signature or prototype), without a definition.
  - ✦So overriding definitions are generally not shown in the class diagram

Class diagram with an interface

This diagram says that objects:

a) Persons have a name
b) Guitarists have a name
c) Guitars have a name
d) MusicPlayers have a name

Interface: labelled with <<interface>>
Implementors: point to the interface using dashed, open arrow



When and how to use Class Diagrams

- All the time.
- Try to keep them simple, don't use unnecessary notation.
  - ✦ Especially if you are using them to model the application domain.
  - ✦ If you want to specify the implementation very specifically, you will use more of the notation.
- Don't draw models for every part of the program (at least not all in great detail)
- Focus first on concepts, then add detail as the design process continues.

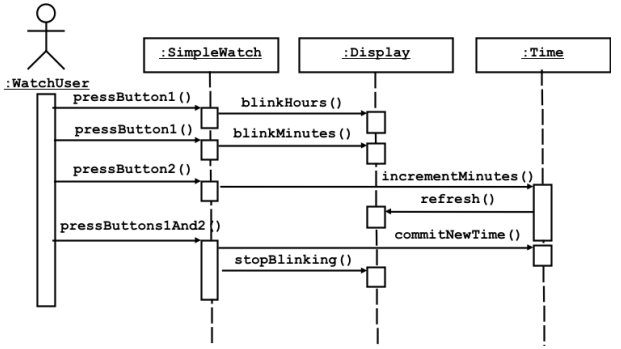
Sequence Diagrams

- Represent the dynamic behavior of the system
- In UML, **Interaction diagrams** are models that describe how groups of objects collaborate in some behavior.
- There are two kinds of Interaction diagrams:
  - ✦**Sequence Diagrams**
  - ✦**Collaboration Diagrams** (we will not talk about these).
- Both of these diagrams describe patterns of communication among a set of interacting objects.

Sequence Diagrams: basics

- An **object** of a given class is shown as a box at the top of a dashed vertical line.
  - ✦The dashed line is the lifeline, representing the object's lifetime.
- An object interacts with another object by sending **messages**.
  - ✦The message must be an operation of the receiving object.
  - ✦The message is shown as an arrow between the lifelines of the object
- Arguments may be passed along with a message
  - ✦they correspond to the parameters of the receiver's operation.
  - ✦variables can be used to label the return value of the operation

Sequence diagram for a simple watch



- Actor and objects (not classes) across the top
- Vertical lines are lifelines of the objects
- Labeled arrows are messages sent to another object

Objects, lifelines, and activation boxes

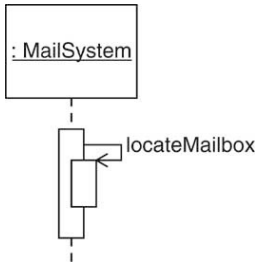
- **Objects** are represented with a box containing a name and the Class the object is an instance of.
  - ✦ *name : Class*
  - ✦ The underline indicates this is an object, not a class.
  - ✦ Only one part (the name or the Class) is required to be specified.
- The dotted line below the object is the object's **lifeline**
  - ✦ Vertical rectangle: an **activation box** representing the duration of an operation.
  - ✦ There must be a message pointing to the top of the box indicating the operation the box corresponds to.

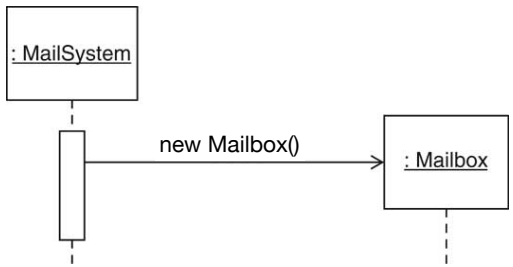
Messages and return arrows

- **Messages** are represented with a solid horizontal arrow from one object's lifeline to another.
  - ✦ The call originates in the object at the source of the arrow
  - ✦ It is received by the object at the end of the arrow
  - ✦ The order in which the messages occur is top to bottom on the page.
  - ✦ The message must be labeled with the name, but can also include the arguments, and a variable to label the operation's result value.
- **Return arrows** are dashed arrows from the bottom of the activation box back to the lifeline of the object that sent the initial message.
  - ✦ These are optional!
  - ✦ They might be labeled with the return value (rather than using a variable)

Self-call and Create new

- Self-call (object calls one of its own methods)
  - ✦Message arrow back to original activation:
- Creating new instances:
  - ✦"new Class()" message points to object's box:

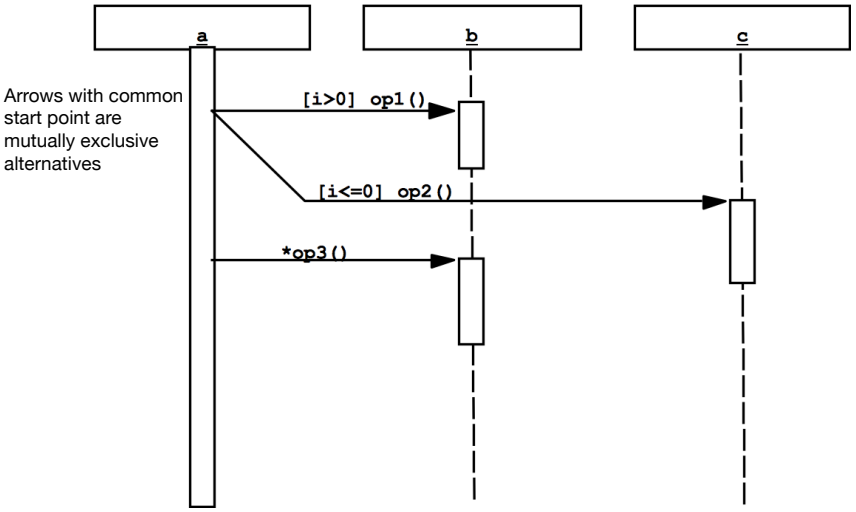




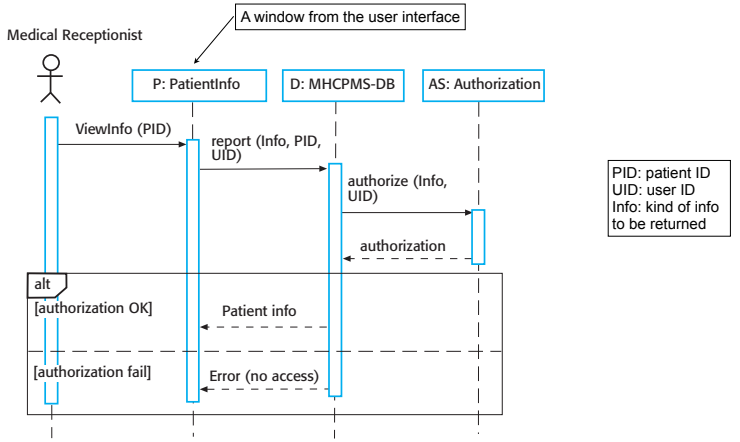
Iteration and branching

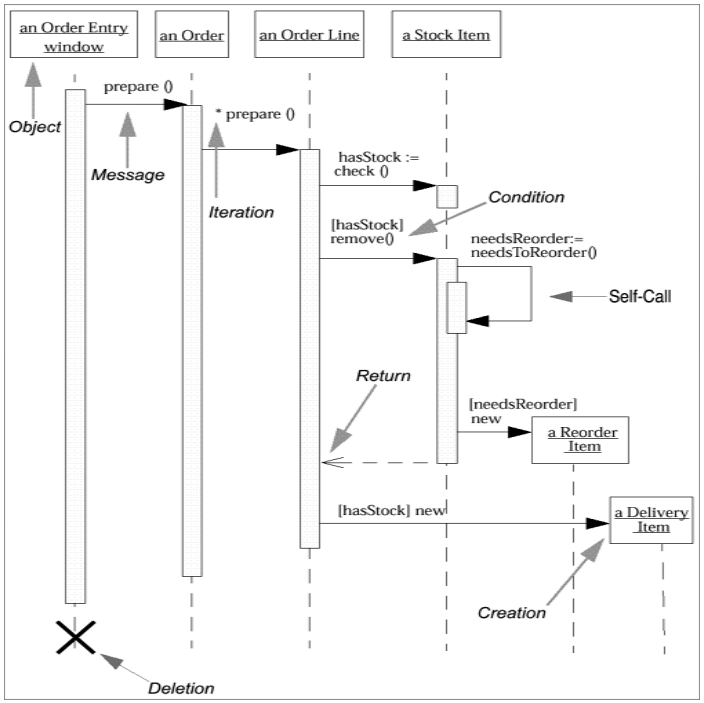
- Notation for iteration (loops)
  - ✦Repeated message has an asterisk (*op3( ))
    - ➡Optional: indicate basis of iteration in brackets: *[for all order lines] op3()
- Notation for branching (alternatives)
  - ✦Conditional messages are marked with a guard (a condition inside square brackets) OR
  - ✦Alternative messages are placed in a partitioned box labeled "alt"
    - ➡each partition has a guard
  - ✦May be easier to draw a separate diagram for each alternative
- If you really want to model control flow, you should use an activity diagram instead.

Branching and iteration in sequence diagrams.



Using alt box to show branching.



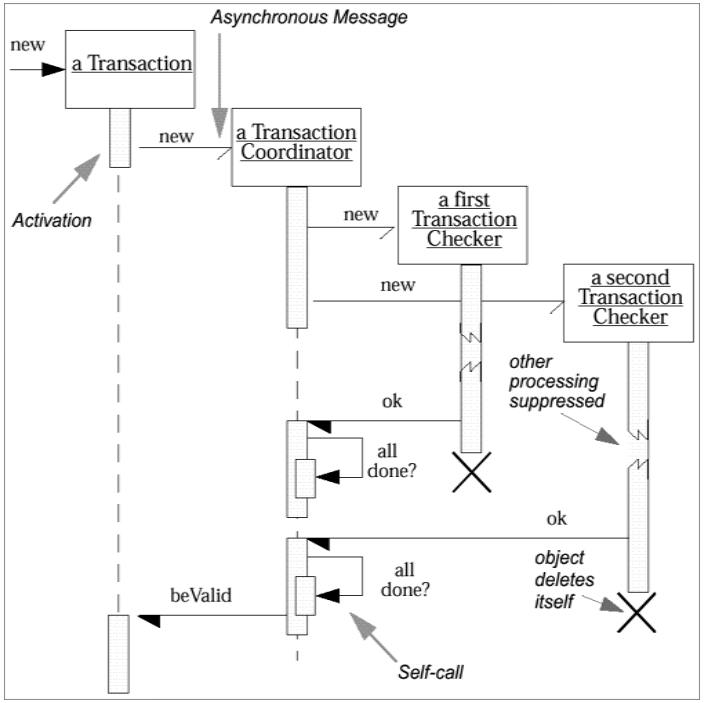


Sequence Diagrams: good practices

- The sequence diagram must be consistent with a given class diagram that fully specifies the classes and their operations
  - ✦ messages to an object's lifeline must correspond to valid operations for that object's class; arguments must be specified, and return values labeled.
- Arguments should be defined somewhere in the diagram:
  - ✦ A Name of another object in the diagram,
  - ✦ An attribute of another object or
  - ✦ A variable labeling the result of a previous message call.
- Activation boxes should not overlap horizontally unless one box's message has called the other.

Sequence diagrams for concurrency/threads

- **asynchronous messages** are represented with a half-arrowhead.
  - ✦ An asynchronous message does not block the caller, it continues simultaneously.
  - ✦ It is ok for activation boxes to overlap horizontally if one is not called from another.
- An asynchronous message can do one of three things:
  - ✦ Create a new thread, linking to the top of an activation
  - ✦ Create a new object
  - ✦ Communicate with a thread that is already running
- Object deletion is shown with a large X.
- Note: GUIs are often asynchronous.



When and how to use Sequence Diagrams

- When you want to look at the behavior of several objects within a single use case.
- When the order of the method calls in the code seems confusing.
- When you are trying to determine which class should contain a given method.
  - ✦to uncover the responsibilities of the classes in the class diagrams
  - ✦to discover even new classes
- During Object-Oriented Design, sequence diagrams and the class diagram are often developed in tandem.