Modeling with UML
Chapter 2

CS 4354
Fall 2012

Jill Seaman

Modeling with UML: in the textbook

- 2.1 Introduction
- 2.2 Overview of UML
  - ✦Introduction to the 5 UML diagrams
- 2.3 Modeling concepts
- 2.4 A Deeper view into UML
  - ✦The 5 types of UML diagrams in greater detail

2.1 Introduction

- UML is a notation to articulate complex ideas
- To enable accurate communication a notation must:
  - ✦Have well-defined semantics
  - ✦Be well suited for representing the desired information
  - ✦Be well understood among project participants
- UML fulfills these requirements
- UML resulted from a unification of many existing notations.
- The goal of UML is to provide a standard notation that can be used by all object-oriented development methods.

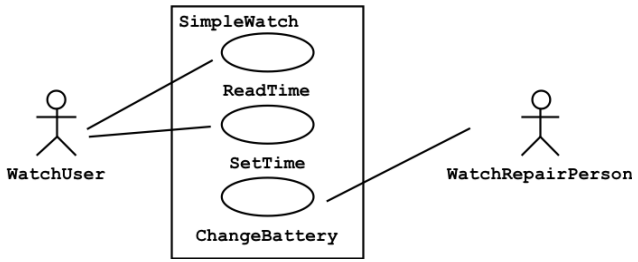
2.2 An Overview of UML

- A brief introduction to five UML notations:
  - ✦Use case diagrams
  - ✦Class diagrams
  - ✦Interaction diagrams (sequence diagrams)
  - ✦State machine diagrams
  - ✦Activity Diagrams.

Use case diagrams

- Use cases are used during requirements elicitation and analysis.
- Purpose: to represent the full functionality of the system.
- A single use case:
  - ✦describes one function provided by the system
  - ✦yields a visible result for an actor
- An actor
  - ✦is any entity that interacts with the system
  - ✦can be a person or another system
  - ✦must be outside the system

Use case diagram for a simple watch

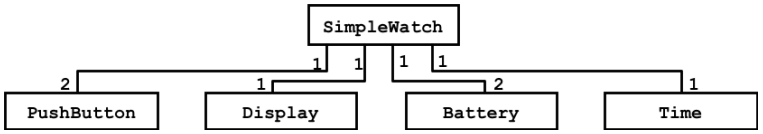


- Actors are stick figures
- Use cases are ovals
- The boundary of the system is the box

Class diagrams

- Used to describe the internal structure of the system.
- 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

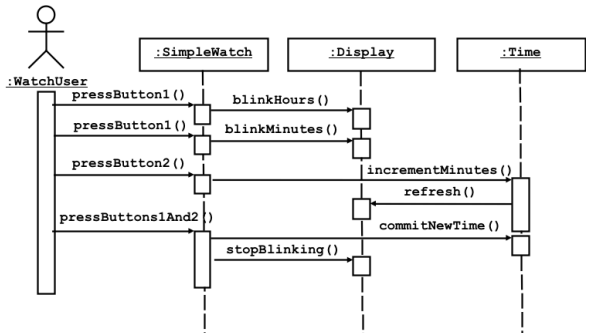


- Boxes are classes
- Lines show associations (between objects)
- Numbers show how many objects must be associated

Interaction diagrams (sequence diagram)

- Represent the dynamic behavior of the system
- Visualize communication among objects
- Show interactions among various objects that are internal to the system.

Sequence diagram for a simple watch

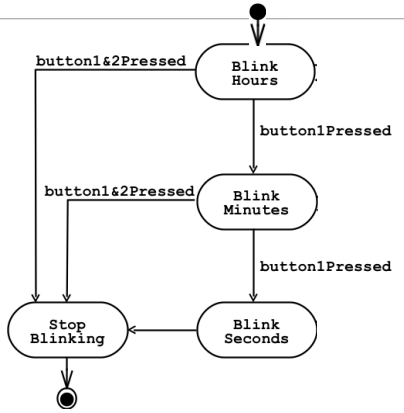


- Actor and objects across the top
- Vertical lines are time lines of the objects
- Labeled arrows are stimuli (or messages) sent to another object

State machine diagrams

- Describe the dynamic behavior of an individual object
- Description uses a number of states and transitions between these states.
  - ✦A graph: states are nodes, transitions are edges
- Transitions occur as a result of external events

State diagram for the watch display

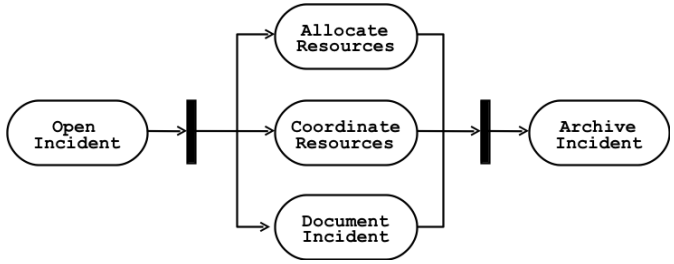


- small black circle: start state
- small black circle inside another circle: finish state
- two arrows are missing labels.

Activity diagrams

- Describe the behavior of a system in terms of activities
- Shows the flow of a set operations
  - ✦ control flow
  - ✦ data flow
- Execution of an activity can be triggered by:
  - ✦ completion of other activities
  - ✦ availability of objects
  - ✦ external events

Activity diagram representing management of an incident



- Round rectangle: activity
- Arrow: control flow
- Thick bar: synchronization (concurrency)

2.3 Modeling concepts Systems, Models, and views

- System an organized set of communicating parts
- Subsystem a system that is a subcomponent of another system.
  - ✦ an object-oriented system is composed of other systems or of objects

Software systems can become too complex to be managed by one individual

- A Model is
  - ✦ a means for dealing with complexity.
  - ✦ an abstraction of a system that focuses on interesting aspects and ignores irrelevant details
  - ✦an abstraction describing a subset of a system.
  - ✦NOT a diagram or notation

Systems, Models, and views cont.

- To build an airplane
  - ✦ build a scale model to test aerodynamic properties
    - ‣ ignore details of instrument panel and engine
  - ✦ build a flight simulator to train pilots
    - ‣ ignore details of exterior of the plane
- Modeling:
  - ✦ uses divide-and-conquer to simplify: build a separate model for each type of problem that needs to be solved
  - ✦ can incrementally refine simple models into more detailed ones
- System model: the set of all models built during development

Three models of a (software) system

- Functional Model: describes the functionality of the system from the user's point of view (functional requirements).
  - ✦ represented in UML using use case diagrams
- Object Model: describes the *structure* of the system in terms of objects, attributes, associations, and operations.
  - ✦ represented in UML using class diagrams
- Dynamic Model describes the internal *behavior* of the system.
  - ✦ represented in UML using sequence diagrams, state diagrams, and activity diagrams.

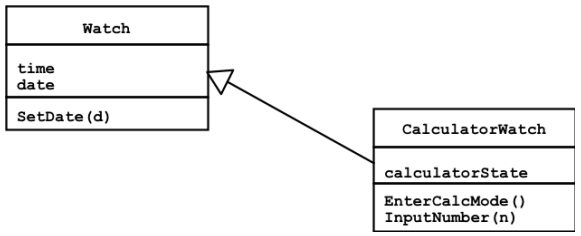
Systems, Models, and views cont.

- View: depicts selected aspects of a given model
- Views may overlap
- Notations: graphical or textual rules for representing views
- A UML class diagram is a graphical view of the object model
  - ✦ you could make a textual view of the object model

Classes, abstract classes, and objects

- A Class is an abstraction in object-oriented modeling and programming
  - ✦ captures both structure and behavior
  - ✦ similar to an Abstract Data Type
- 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
  - ✦ base class is also called the generalized class, or the superclass.
  - ✦ refined class is also called the specialized class, or the subclass.

Class diagram showing inheritance



- Arrow points to superclass
- Top box: class name
- Middle box: attributes
- Third box: operations

CalculatorWatch has functionality that Watch doesn't have

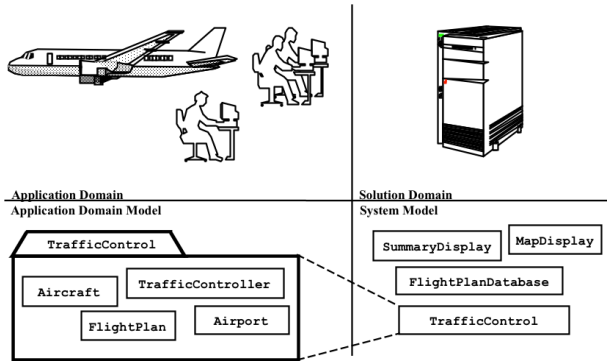
Classes, abstract classes, and objects cont.

- An object is
  - ✦ an instance of a class
  - ✦ has values for each attribute in the class
  - ✦ class's operations work over the object.
  - ✦ each object is an instance of exactly one class
- An Abstract Class is a superclass that does not have any concrete instances.
  - ✦ it is never instantiated
  - ✦ it exists to represent shared attributes and operations
  - ✦ Watch is NOT an abstract class, some watches are not CalculatorWatches

Object-oriented modeling

- The application domain is all aspects of the customer's "problem".
  - ✦ physical environment
  - ✦ users and other people
  - ✦ their work processes, etc.
  - ✦Object-oriented analysis models the application domain
- The solution domain is the modeling space of all possible solutions.
  - ✦ represents system design and object design
  - ✦ richer and more volatile than application domain, more detail.
  - ✦ Object-oriented design models the solution domain
- Both use the same representation (classes, associations, etc)

Application domain and Solution Domain

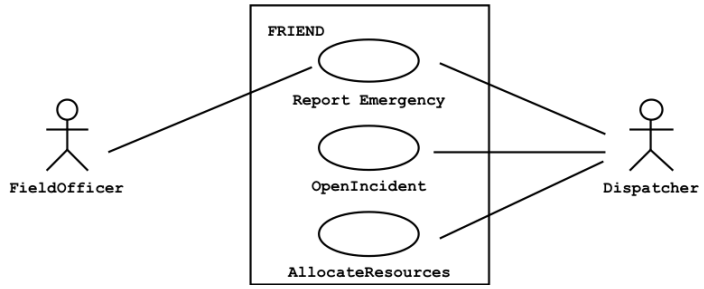


Note the System Model contains the Application domain model.

2.4.1 Use Case Diagrams (in detail)

- An Actor is an external entity that interacts with the system.
  - ✦ different kinds of users, other systems
- A Use case describes the behavior of the system from an actor's point of view.
  - ✦ describes a function provided by the system
  - ✦ described as a set of events.
  - ✦ yields a visible/observable result for the actor
- When actors and use cases exchange information they are said to communicate.

Example use case diagram



- Three separate use cases
- FRIEND: First Responder Interactive Emergency Navigational Database

Use Case: textual descriptions

- Uses a template with six fields
  - **Name**: unique in the system model
  - **Participating actors**: actors interacting with use case
  - **Entry conditions**: must be true to initiate use case
  - **Flow of events**: describes the sequence of interactions, numbered for reference. Actor steps on left, system response on right.
  - **Exit conditions**: will be true after use case is complete
  - **Quality requirements**: not related to functionality (constraints on performance, etc.)
- These fields are not "standard", may see many variations.

Use case textual description for Report Emergency

Use case name	ReportEmergency
Participating actors	Initiated by FieldOfficer Communicates with Dispatcher
Flow of events	1. The FieldOfficer activates the "Report Emergency" function of her terminal. 2. FRIEND responds by presenting a form to the FieldOfficer. 3. The FieldOfficer fills out the form by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form. 4. FRIEND receives the form and notifies the Dispatcher. 5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the report. 6. FRIEND displays the acknowledgement and the selected response to the FieldOfficer.
Entry condition	The FieldOfficer is logged into FRIEND.
Exit condition	The FieldOfficer has received an acknowledgement and the selected response from the Dispatcher, OR The FieldOfficer has received an explanation indicating why the transaction could not be processed.
Quality requirements	The FieldOfficer's report is acknowledged within 30 seconds. The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

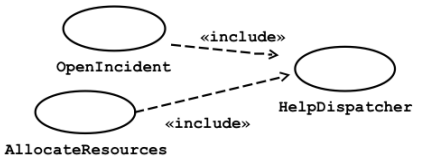
Use Case Diagrams:

- 4 kinds of relationships
  - **Communication**
    indicates information is exchanged between actor and use case
    represented by a solid line in diagram
    described by textual description
  - **Inclusion**
  - **Extension**
  - **Inheritance**

Include relationship

- Used when multiple use cases share a common step or event
- For example, assume the Dispatcher can at any time press a key to access a street map:
  - ✦ Make a new use case called “ViewMap”
  - ✦ It will be included in OpenIncident and AllocateResources use cases
- Two use cases are related by an include relationship if one includes the other in its flow of events.
- Textual description of the including use case:
  - ✦ Add to flow of events at point where it occurs OR
  - ✦ Add to Quality requirements if it can occur any time

Example include relationship use case diagram



- Dashed arrow points to the included use case

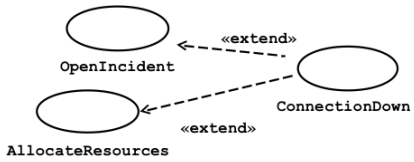
Textual representation of include relationship

Use case name	AllocateResources
Participating actor	Initiated by Dispatcher
Flow of events	…
Entry condition	The Dispatcher opens an Incident.
Exit condition	Additional Resources are assigned to the Incident. Resources receives notice about their new assignment. FieldOfficer in charge of the Incident receives notice about the new Resources.
Quality requirements	At any point during the flow of events, this use case can **include** the ViewMap use case. The ViewMap use case is initiated when the Dispatcher invokes the map function. When invoked within this use case, the system scrolls the map so that location of the current Incident is visible to the Dispatcher.

Extend relationship

- Used to add events to an existing use case, especially exceptional behavior
- For example, assume the network connection between the Dispatcher and FieldOfficer may fail:
  - ✦ Make a new use case called “ConnectionDown”
  - ✦ Describes events that will happen when connection is lost
  - ✦ It will extend the OpenIncident and AllocateResources use cases
- Textual description of the extending use case:
  - ✦ Add to Entry condition the names of the extended use cases
  - ✦ Do not need to make any changes to the original, extended use case

Example extend relationship use case diagram



- Dashed arrow points to the extended use case

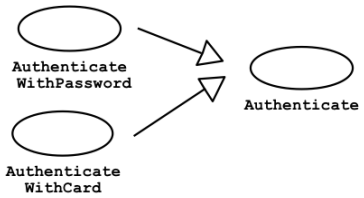
Textual representation of extend relationship

Use case name	ConnectionDown
Participating actor	Communicates with FieldOfficer and Dispatcher.
Flow of events	…
Entry condition	This use case **extends** the OpenIncident and the AllocateResources use cases. It is initiated by the system whenever the network connection between the FieldOfficer and Dispatcher is lost.
Exit condition	…

Inheritance relationship

- Used when one use case is a specialization of another
  - ✦It adds more detail
- For example, assume FieldOfficers are required to authenticate before they can use Friend:
  - ✦ Early stages of modeling has one use case called Authenticate
  - ✦ Later stages add two specific ways of authenticating, called AuthenticateWithPassword and AuthenticateWithCard
  - ✦ New use cases give details of authenticating by their method
- Textual description of the specialized use case:
  - ✦ Inherit initiating actor, and entry and exit conditions from general use case.

Example inheritance relationship use case diagram



- Solid arrow points to the generalized use case

Textual representation of inheritance relationship

Use case name	AuthenticateWithCard
Participating actor	**Inherited** from Authenticate use case.
Flow of events	1. The FieldOfficer inserts her card into the field terminal. 2. The field terminal acknowledges the card and prompts the actor for her personal identification number (PIN). 3. The FieldOfficer enters her PIN with the numeric keypad. 4. The field terminal checks the entered PIN against the PIN stored on the card. If the PINs match, the FieldOfficer is authenticated. Otherwise, the field terminal rejects the authentication attempt.
Entry condition	**Inherited** from Authenticate use case.
Exit condition	**Inherited** from Authenticate use case.

Scenarios

- A **scenario** is an instance of a use case describing a concrete set of actions.
  - ✦A use case describes all possible scenarios
- Scenario uses three fields:
  - ✦ The **name** of a scenario is underlined to indicate that it is an instance.
  - ✦The **participating actor instances** field indicates which actor instances are involved in this scenario. Actor instances also have underlined names.
  - ✦The **flow of events** of a scenario describes the sequence of events step by step.

WarehouseOnFire scenario for the ReportEmergency use case

Scenario name	warehouseOnFire
Participating actor instances	bob, alice: FieldOfficer john: Dispatcher
Flow of events	1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the "Report Emergency" function from her FRIEND laptop. 2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appears to be relatively busy. She confirms her input and waits for an acknowledgement. 3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice. 4. Alice receives the acknowledgement and the ETA.

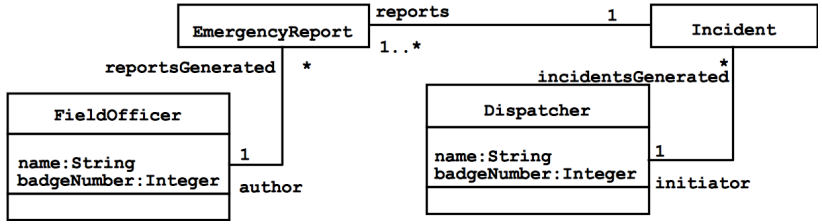
2.4.2 Class Diagrams (in detail)

- Used to describe the internal structure of the system.
- 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
  - ✦Objects, entities with state (values for attributes) and behavior.
  - ✦Associations that can occur between objects in various classes

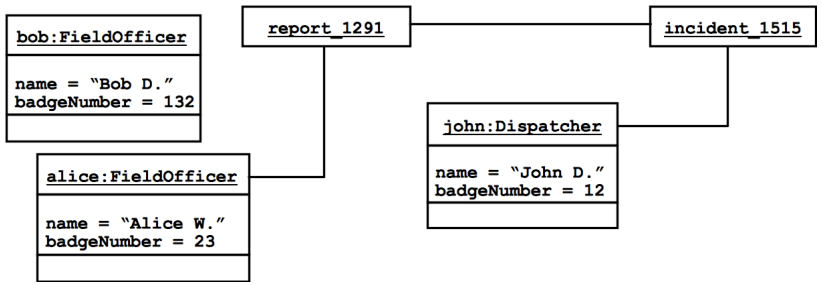
Class Diagrams

- Classes and objects are boxes composed of three compartments:
  - ✦Top compartment: name
  - ✦Center compartment: attributes (may be omitted for simplicity/delay)
  - ✦Bottom compartment: operations (may be omitted for simplicity/delay)
- Conventions:
  - ✦Object names are underlines to indicate they are instances
  - ✦Class names start with uppercase letter
  - ✦Named objects start with lowercase

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



UML object diagram: **objects** that participate in the warehouseOnFire scenario.

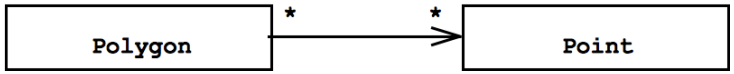


Note object names are underlined, multiple instances of FieldOfficer.

Associations and links

- A link represents a connection between two objects.
- Associations are relationships between classes and represent the fact that links may (or do) exist between object instances.
  - ✦Links and 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.

Example of a unidirectional association

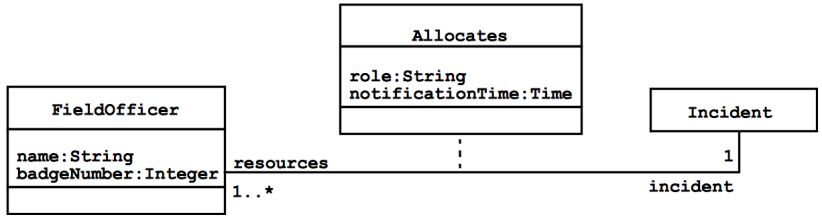


This system supports navigation from the Polygon to the Point, but not vice versa. Given a specific Polygon, it is possible to query all Points that make up the Polygon. But a given Point does not know which Polygon(s) it belongs to.

*Note: the diagram in the book is wrong

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.



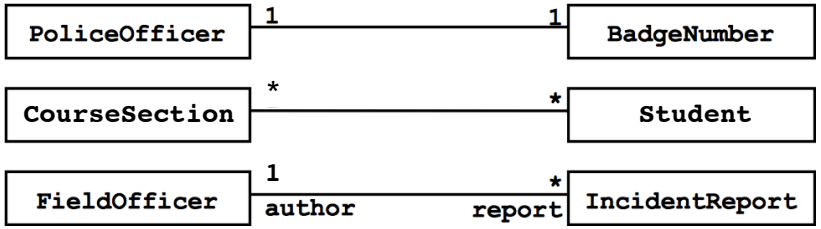
Roles

- 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.
- Previous slide:
  - ✦The FieldOfficers allocated to a given incident are called resources.

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.
- * 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.

Examples of multiplicity

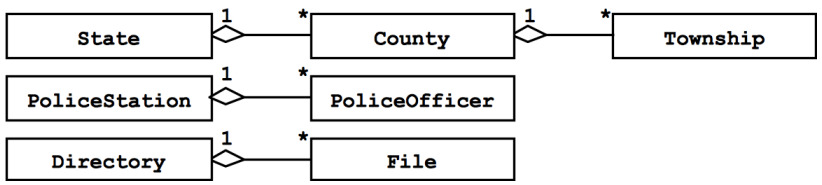


How many reports can a FieldOfficer write?
How many authors of a report can there be?

Aggregation

- Aggregation is a kind of association that specifies a whole/part relationship between the aggregate (whole) and component part.
  - ✦Useful to denote hierarchical relationships (usually recursive).
  - ✦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

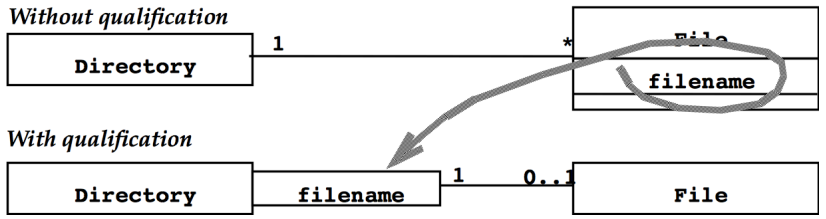


None of these are recursive.

Could any of them be composites?

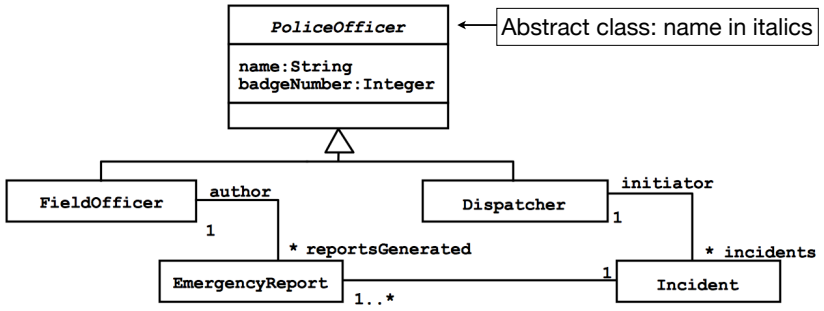
Qualification

- A qualified association is a (directed) association which allows us to restrict the objects referred to in an association using a key attribute.
- The multiplicity on the target end indicates how many instances of the target class with a given value of the key attribute can participate with a given source object.
- Only one File per filename can be in a Directory
- Note apparent reduction in multiplicity.

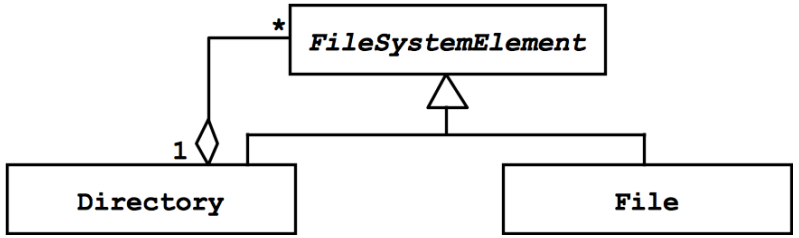


Inheritance

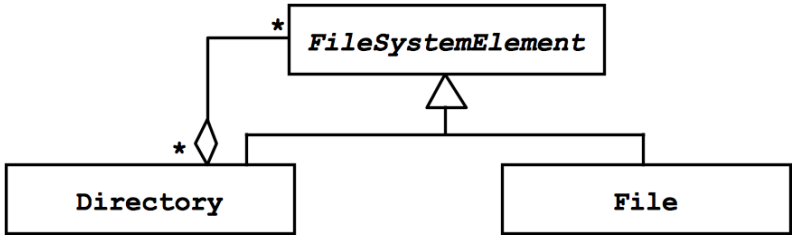
- Inheritance is a relationship between a base class and a more refined class.
  - ✦ the refined class has attributes and operations of its own, as well as the attributes and operations of the base class (it inherits them).



Example of a hierarchical file system



Example of a non-hierarchical file system



What's the difference?

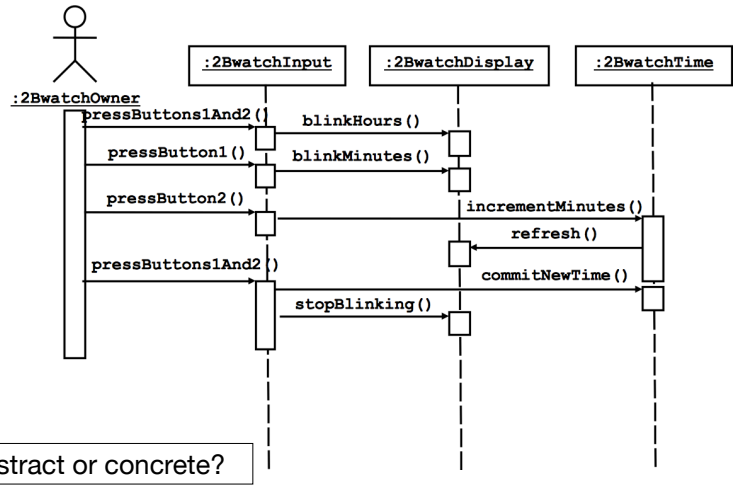
2.4.3 Interaction Diagrams (in detail)

- Represent the dynamic behavior of the system
- Describe patterns of communication among a set of interacting objects.
- An object interacts with another object by sending **messages**.
  - ✦The message must be an operation of the receiving object.
- Arguments may be passed along with a message
  - ✦they correspond to the parameters of the receiver's operation.
- In UML, interaction diagrams can be either:
  - ✦sequence diagrams or
  - ✦communication (formerly collaboration) diagrams.

Sequence Diagrams

- Objects across the top (horizontal axis)
- Time goes down (vertical axis)
- Diagram components:
  - ✦Solid horizontal arrows: messages
  - ✦Labels on arrows: message names (arguments optional)
  - ✦Return arrows (at end of operation, going back) are optional
  - ✦Vertical rectangle: an activation (lifetime) of an operation
  - ✦Interactions involving actors may not be operations.
- Sequence diagrams may be:
  - ✦abstract (representing all possible interactions)
  - ✦concrete (representing a specific interaction)

Sequence diagram: setting the time on 2Bwatch.

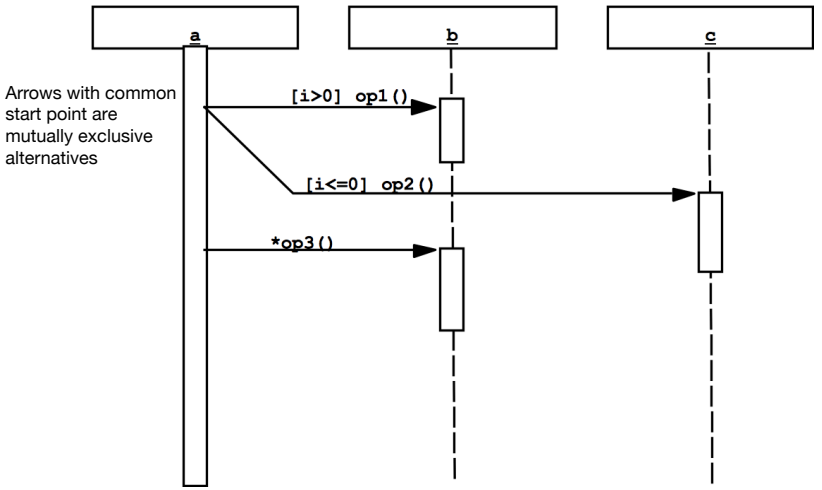


Abstract or concrete?

Abstract Sequence Diagrams

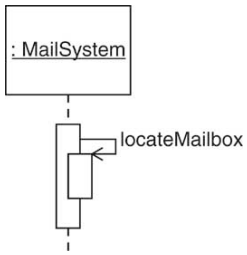
- Notation for iteration (loops)
  - ✦Repeated message(s) in a box labeled “loop” (see book)
  - ✦Repeated message has an asterisk (see next slide)
    - ➡Optional: indicate basis of iteration in brackets: *[for all order lines] op3()
- Notation for conditions (alternatives)
  - ✦Alternative message(s) in a partitioned box labeled “alt” (see book)
    - ➡each partition has a guarded message (see book): [i>0] op1()
  - ✦Conditional message is marked with a guard (see next slide)
  - ✦May be easier to draw a separate diagram for each alternative

Conditions and iterators in sequence diagrams.

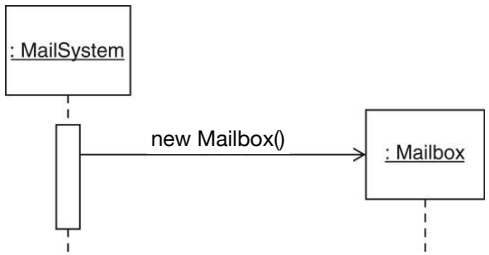


Sequence Diagrams: additional notations

- Self-call (operation calls itself)
  - ✦Message arrow back to original activation:



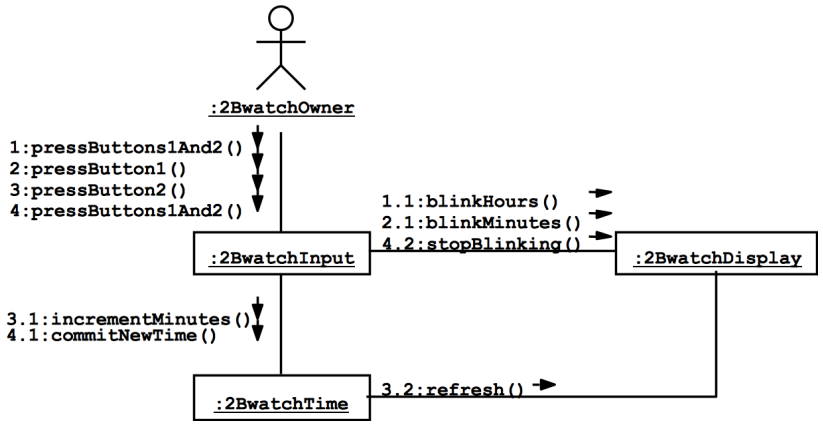
- Creating new instances:
  - ✦"new Class()" message points to object's box:



Communication (formerly Collaboration) Diagrams

- Depicts the same information as sequence diagrams
- Objects/Actors can be anywhere
- Messages between objects are numbered sequentially
  - ✦More difficult to understand sequence
  - ✦Relationships between objects are more obvious
  - ✦Can specify other information, such as grouping

Collaboration diagram: Setting the time on 2Bwatch



2.4.5 Activity Diagrams (in detail)

- Describe the behavior of a system in terms of activities
- Represent the sequencing and coordination of lower level behaviors.

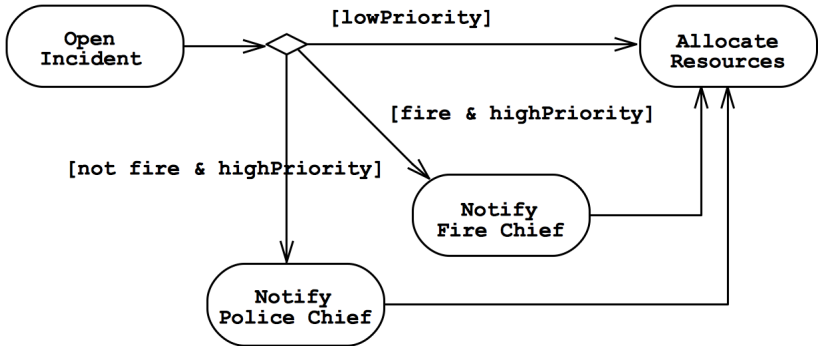
- Rounded rectangles represent actions and activities.
- Edges between activities represent control flow.

- Activity diagrams can be hierarchical:
  - ✦A given activity in a rounded rectangle could be further detailed in its own separate activity diagram.

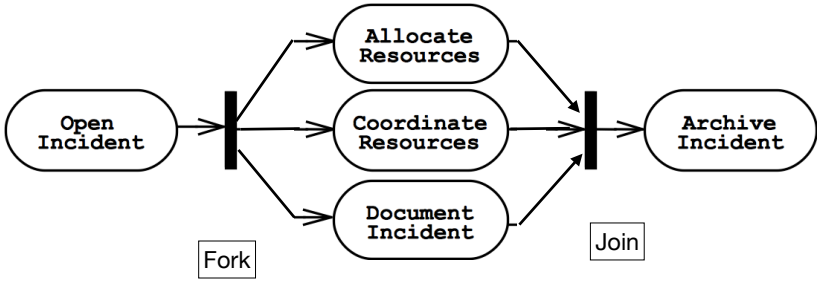
Activity Diagrams: control nodes

- Decisions (branches, alternates)
  - ✦Diamond with one (or more) incoming arrow(s), two or more outgoing arrows.
  - ✦Outgoing edges labeled with guards (conditions) that select that arrow.
- Fork nodes and Join nodes (concurrency)
  - ✦Fork: denotes splitting control into multiple threads
  - ✦Join: denotes synchronizing threads back into one
  - ✦Also may denote activities that may be done in any order

Decision in the Handle Incident process.



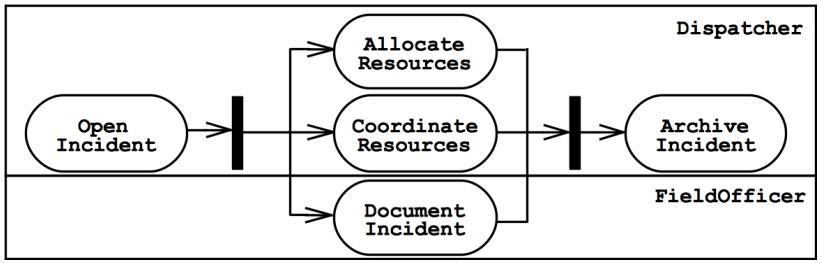
Concurrency in another process.



Activity Diagrams: swimlanes

- Swimlanes (activity partitions)
  - ✦Rectangles enclosing a group of activities
  - ✦Denotes object of subsystem that implements the activities
  - ✦Edges may cross swimlane boundaries

Swimlanes in the other process.



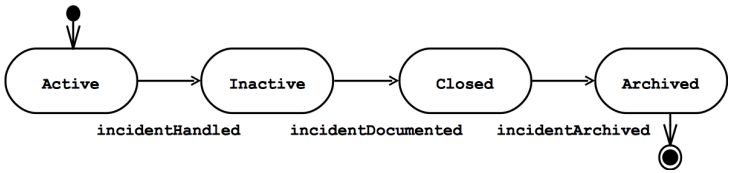
2.4.4 State Machine Diagrams (in detail)

- Describe the dynamic behavior of an individual object
- Describes the sequence of states an object goes through in response to external events

States and Transitions

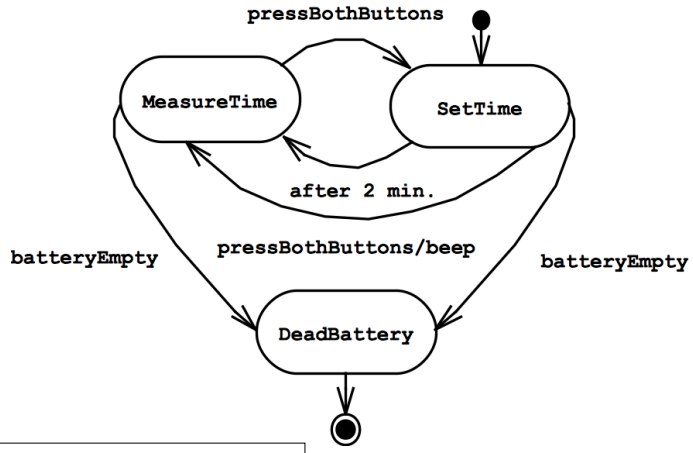
- A state is a condition satisfied by the values of attributes of an object.
  - ✦An Incident can exist in four states: Active, Inactive, Closed and Archived
  - ✦These are nodes in the graph
  - ✦A node can have some activity that is performed when the node is entered.
- A transition represents a change of state triggered by events, conditions, or time.
  - ✦Transitions are directed edges in the graph
  - ✦labelled by the event causing the transition:
    Event [Guard] / Action
    Each part is optional

State Machine diagram for Incident class



Solid black circle is start
Concentric black circle is finish
Transitions labeled by events

State Machine diagram for 2Bwatch

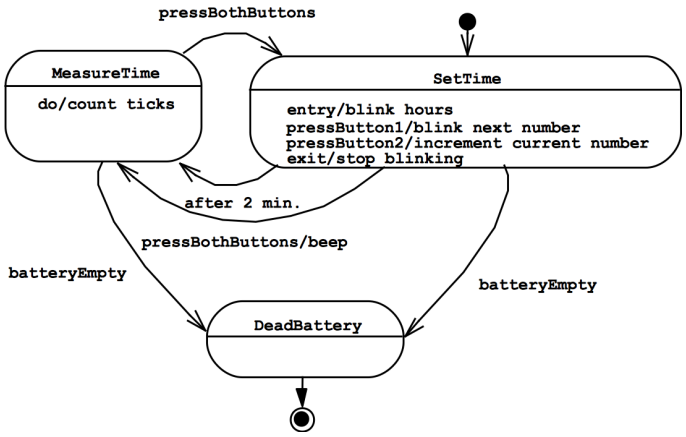


/beep is the action that happens when both buttons are pressed

Actions and Activities

- Action can happen in three places
  - ✦During a transition (/beep)
  - ✦when a state is entered
  - ✦when a state is exited
- An internal transition is a transition that doesn't leave the state
  - ✦Triggered by events
  - ✦Can have actions
  - ✦Does not trigger enter or exit actions
- An activity is a coordinated set of actions
  - ✦can be interrupted
  - ✦used to describe behavior in a state, indicated by "do/action"

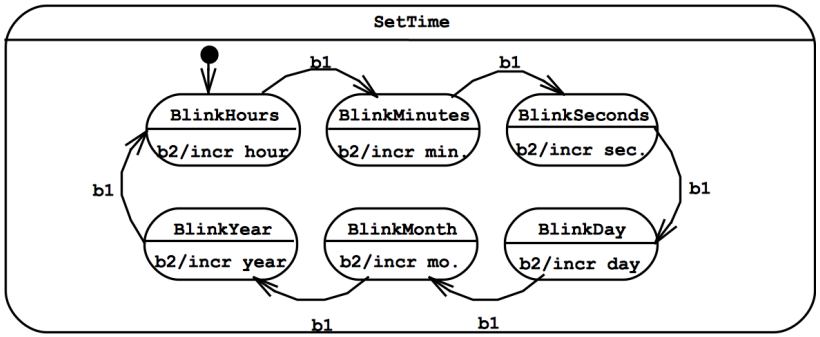
Refined State Machine diagram for 2Bwatch



Includes activities that happen when the watch is in a given state

Nested State Machine example: SetTime state

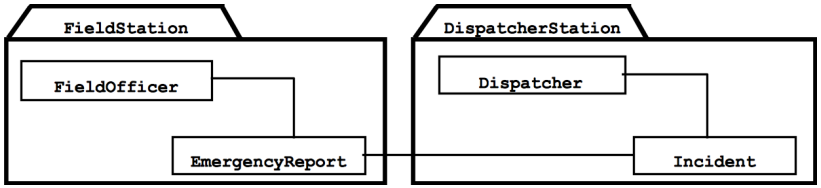
Instead of internal transitions, use a separate state diagram



b1 = pressButton1 b2 = pressButton2

2.4.6 Diagram Organization: packages

A Package is a grouping of model elements (usually classes)



The diagram is often shown without the internal elements, to show relationships among the packages.

2.4.7 Diagram Extensions: stereotypes

A stereotype is an extension mechanism that allows developers to classify model elements in UML.

For example, we classify objects into three types: entity, boundary, and control.