# SWE209
# Object Oriented Analysis and Design

## Modelling with UML - 2

# Note

- This presentation is based on the slides and content of the course main textbook.
- Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014
- https://ase.in.tum.de/lehrstuhl_1/component/content/article/43-books/217

# Agenda

Introduction

Use Case Diagrams

Class Diagrams

Interaction Diagrams

State Machine Diagrams

Activity Diagrams

Diagram Organization

Diagram Extensions

# Introduction

- Describe the five main UML diagrams in detail.
- Use case diagrams → Functionality of the system
- Class diagrams → Static structure of a system
- Interaction diagrams → Behavior of the system in terms of interactions among objects
- State machine diagrams → Behavior of nontrivial objects
- Activity diagrams → Data or control flow through a system

# Use Case Diagrams

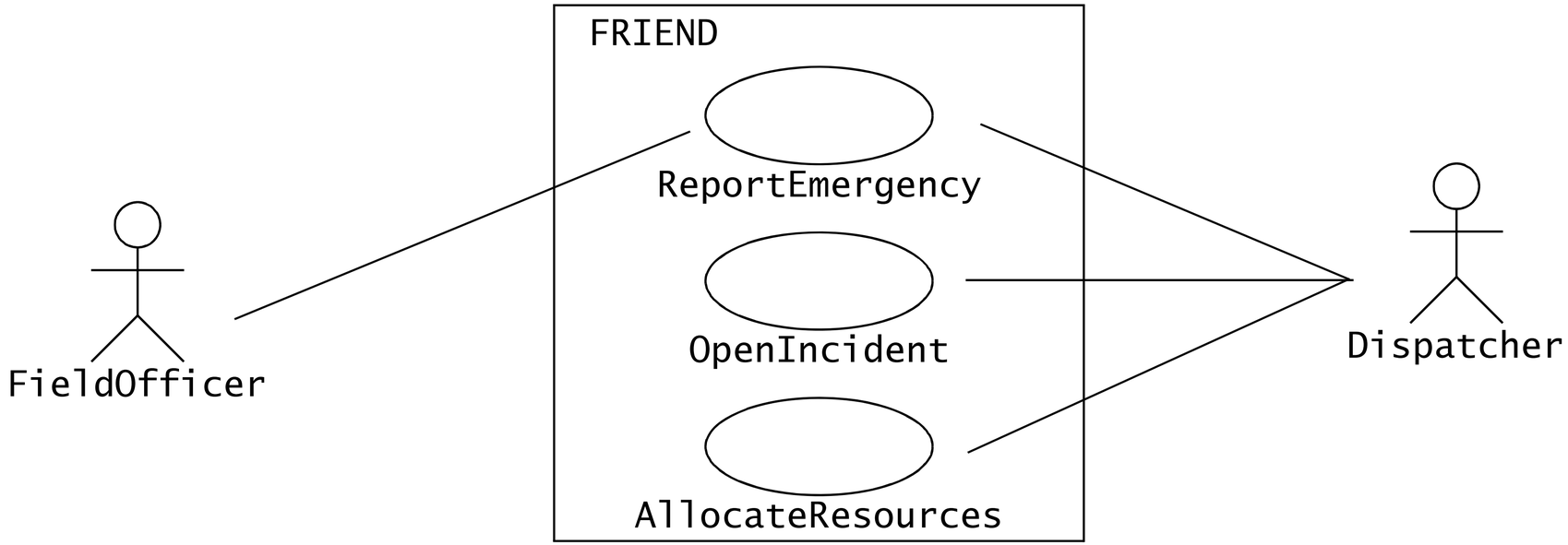
# Use Case Diagrams

- Use case diagram = Actors + Use cases
- Actor → External entity that interact with the system.
  - User role → System administrator, a bank customer, a bank teller, etc.
  - Another system → Central database, fabrication line, etc.
- Use case → The behavior of the system as seen from an actor's point of view.
  - Actor's point of view → External view
  - External behavior

# Use Case Diagrams

- A use case describes a function.
- Actors initiate use cases.
- Use cases also can initiate other use cases.
- Actors and use cases exchange information → Communicate

# Use Case Diagrams




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Use Case Diagrams

Bernd Bruegge, Allen H. Dutoit
Object-Oriented Software Engineering:
Using UML, Patterns and Java
3rd Edition, Pearson, 2014.

| | |
|---|---|
| *Use case name* | `ReportEmergency` |
| *Participating actors* | Initiated by `FieldOfficer`<br>Communicates with `Dispatcher` |
| *Flow of events* | 1. The `FieldOfficer` activates the "Report Emergency" function of her terminal.<br>2. `FRIEND` responds by presenting a form to the `FieldOfficer`.<br>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.<br>4. `FRIEND` receives the form and notifies the `Dispatcher`.<br>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.<br>6. `FRIEND` displays the acknowledgment and the selected response to the `FieldOfficer`. |
| *Entry condition* | • The `FieldOfficer` is logged into `FRIEND`. |
| *Exit condition* | • The `FieldOfficer` has received an acknowledgment and the selected response from the `Dispatcher`, OR<br>• 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.<br>• The selected response arrives no later than 30 seconds after it is sent by the `Dispatcher`. |

# Use Case Diagrams

- Use cases are written in natural language.
  - Easy communication with the clients and the users.
  - Participants from other disciplines can understand the system.
  - Developers can capture special requirements.

# Use Case Diagrams

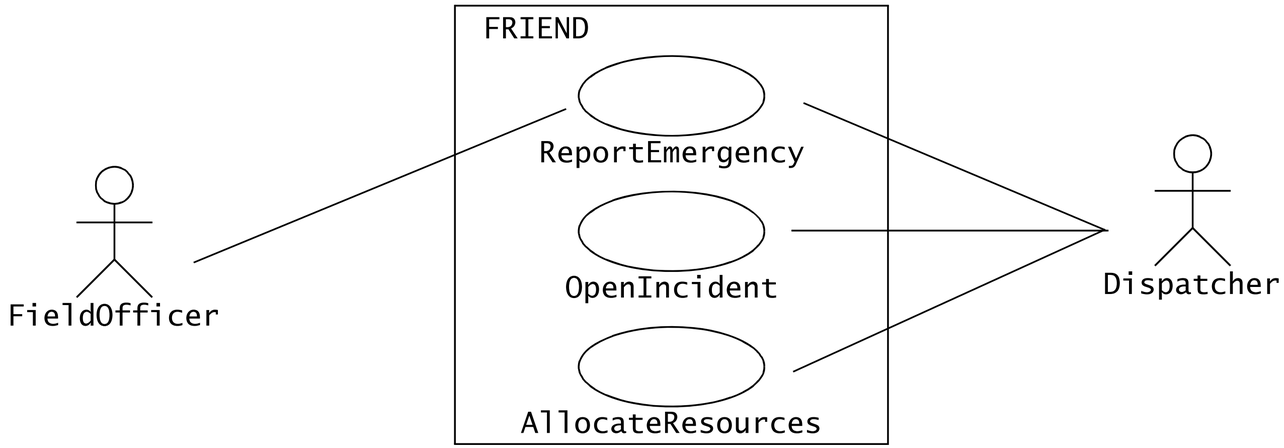
## Relationship

- Use case diagrams can include four types of relationships:
  - Communication
  - Inclusion
  - Extension
  - Inheritance

# Use Case Diagrams

## Relationship → Communication Relationship

- Communication → Information exchange between actors and use cases
- Communication relationship → Used to denote access to functionality.

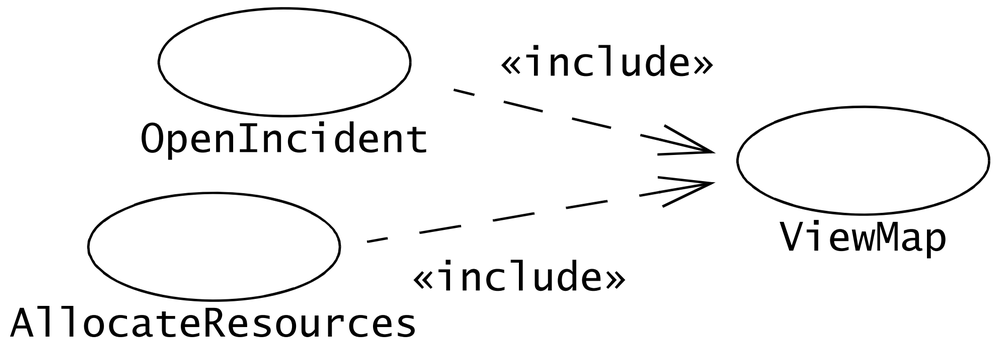



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Use Case Diagrams

## Relationship → Include Relationship

- Used to reduce complexity and redundancy. (fazlalık)
- Identifying commonalities in different use cases. (ortak noktalar)
- Include common use cases in other use cases.




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Use Case Diagrams

## Relationship → Include Relationship

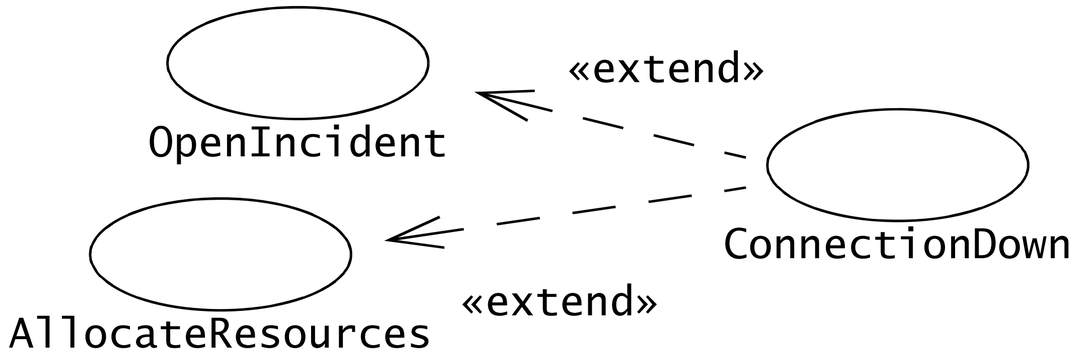
| | |
|---|---|
| *Use case name* | `AllocateResources` |
| *Participating actor* | Initiated by `Dispatcher` |
| *Flow of events* | . . . |
| *Entry condition* | • The `Dispatcher` opens an `Incident`. |
| *Exit condition* | • Additional `Resources` are assigned to the `Incident`.<br>• `Resources` receives notice about their new assignment.<br>• `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`. |

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Use Case Diagrams

## Relationship → Extend Relationship

- Reduce complexity in the use case model.
- Extend another use case by adding events.
- Typical application → Specification of exceptional behavior




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Use Case Diagrams

## Relationship → 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* | . . . |

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Use Case Diagrams

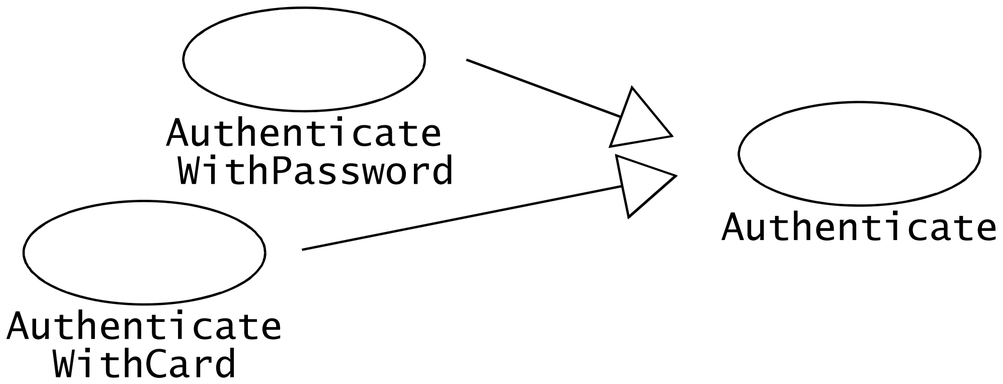
## Relationship → Extend Relationship

- The difference between the include and extend relationships
  - The location of the dependency
- Exception cases are modeled with extend relationships.
  - Help, error, unexpected condition
- Behavior commonly shared are modeled with include relationships.

# Use Case Diagrams

## Relationship → Inheritance Relationship

- Reduce the complexity of a model.
- One use case can specialize another more general one by adding more detail.




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Use Case Diagrams

## Relationship → 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.<br>2. The field terminal acknowledges the card and prompts the actor for her personal identification number (PIN).<br>3. The `FieldOfficer` enters her PIN with the numeric keypad.<br>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. |

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Use Case Diagrams

## Relationship → Inheritance Relationship

- Extend relationships and inheritance relationships are different.
- Extend relationship
  - Each use case describes a different flow of events to accomplish a different task.
- Inheritance relationship
  - Specialization and generalization describe the same task, each at different abstraction levels.

# Use Case Diagrams

## Scenario

- Scenario → An instance of a use case describing a concrete set of actions.

| | |
|---|---|
| *Scenario name* | `warehouseOnFire` |
| *Participating actor instances* | `bob, alice:FieldOfficer`<br>`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.<br>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 acknowledgment.<br>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.<br>4. Alice receives the acknowledgment and the ETA. |

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

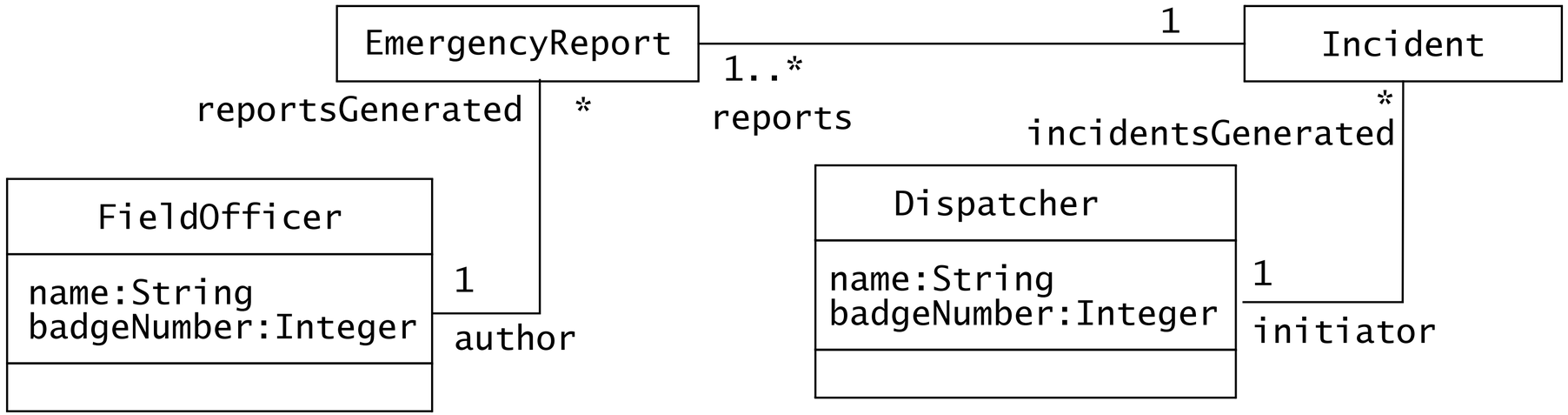
# Class Diagrams

# Class Diagrams

- Describe the structure of the system.
- Class → Abstraction → Attributes and behavior of a set of objects
- Object → Entity that encapsulate state and behavior.
- Each object has an identity.
  - Referred individually
  - Distinguish from other objects

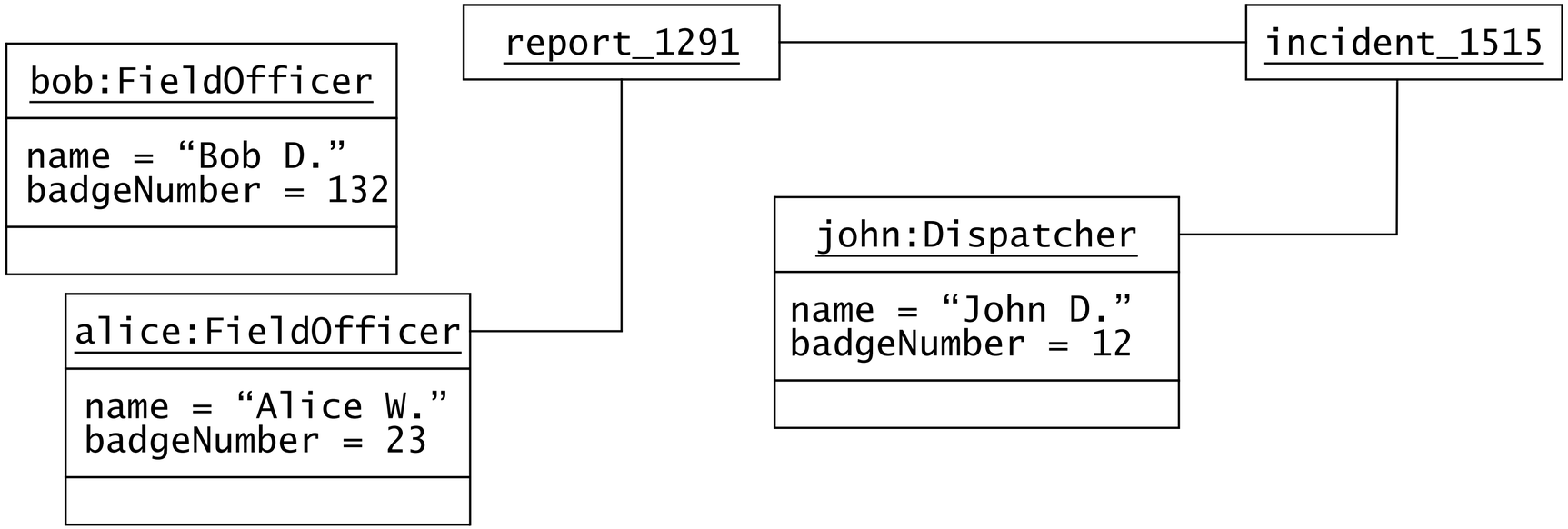
ayırt etmek

# Class Diagrams




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

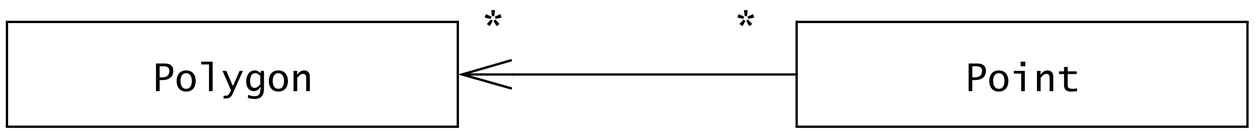



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Associations and Links

- Associations are relationships between classes and represent groups of links.
- A link represents a connection between two objects.
- Association → Symmetrical (bidirectional) or asymmetrical (unidirectional).

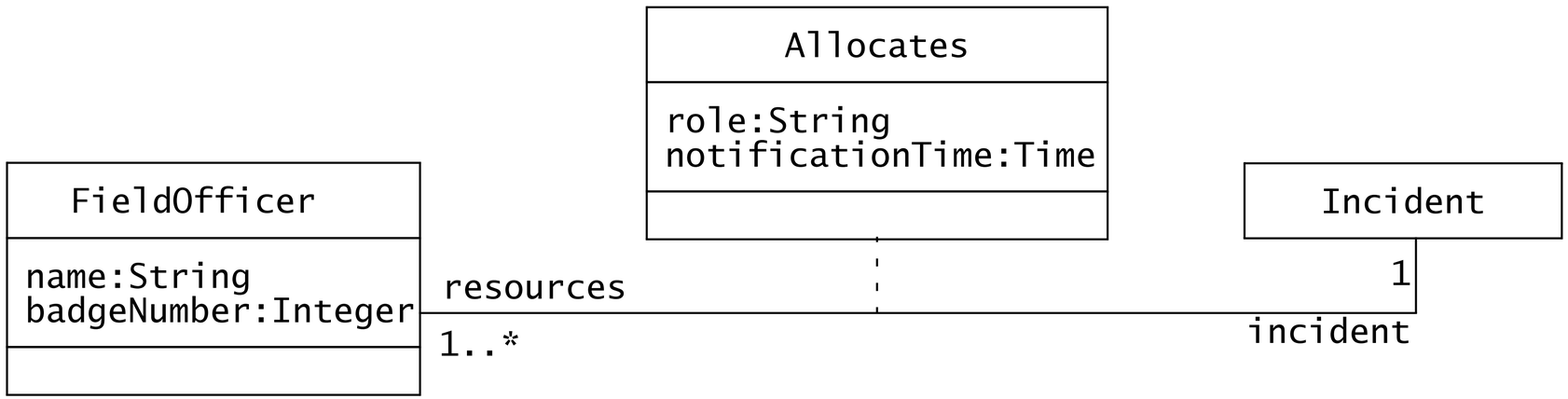



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Association Class

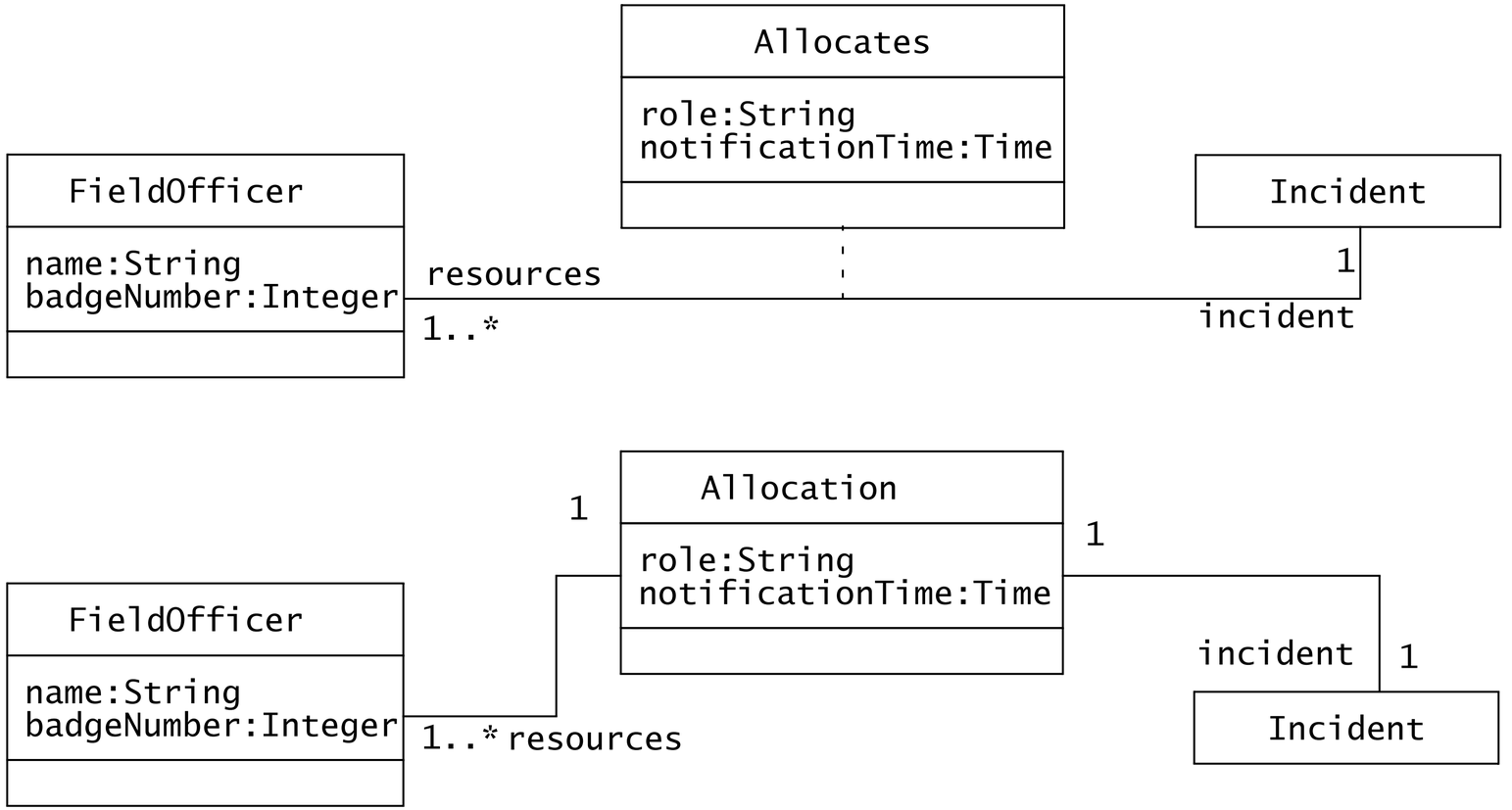
- Associations can have attributes and operations.




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Association Class

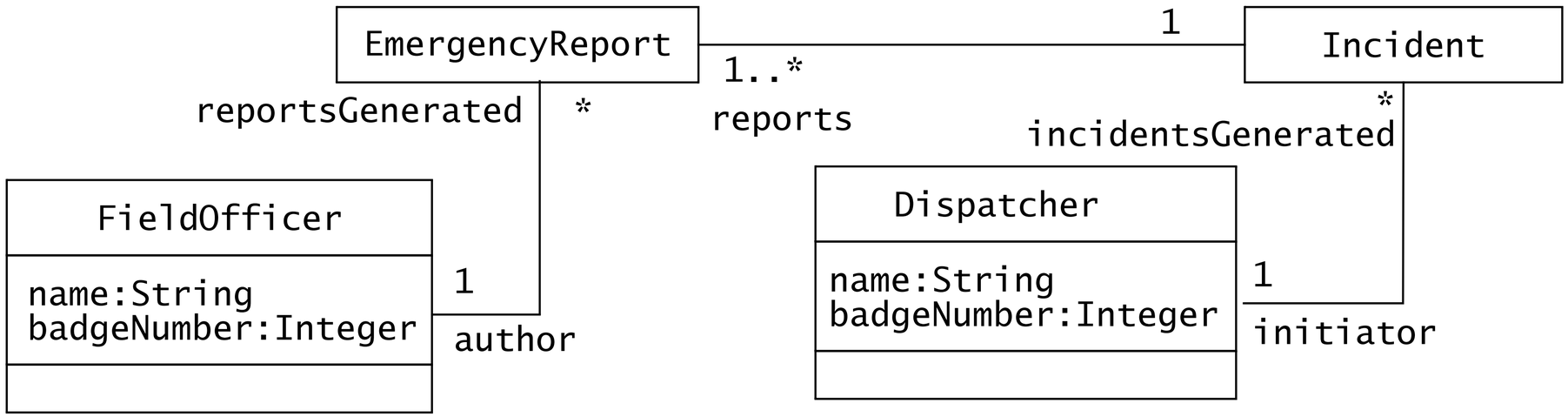



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Roles

- Allow us to distinguish associations.
- Clarify the purpose of the association.

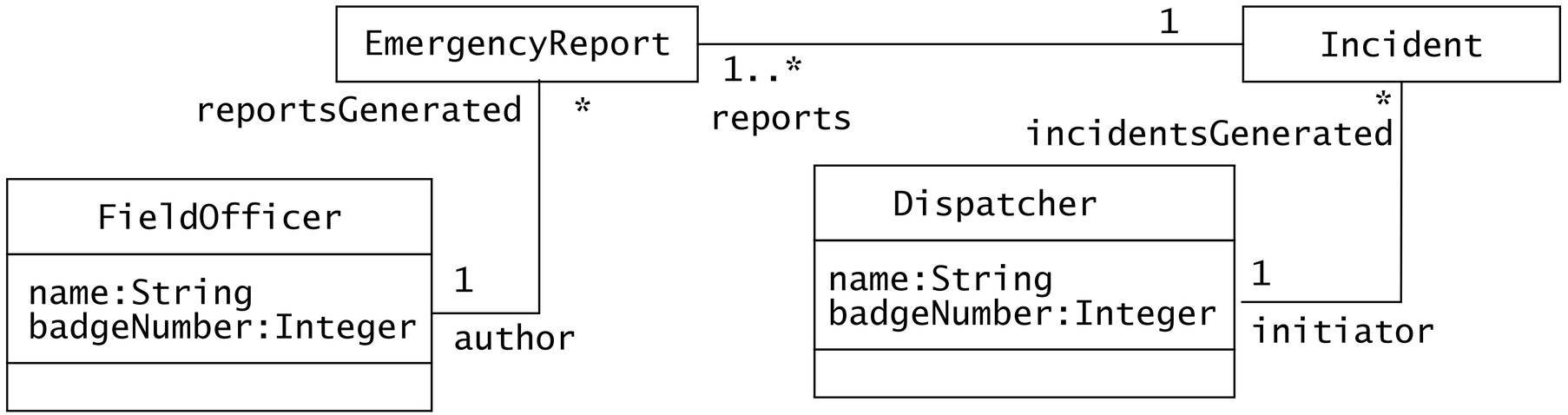



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Multiplicity

- Integers indicating the number of links.

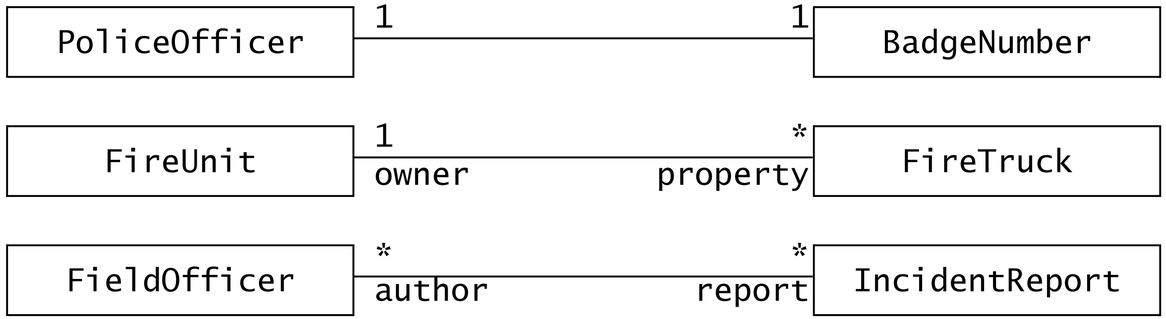



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Multiplicity

- Three common types of associations:
  - One-to-one association → Multiplicity 1 ↔ 1
  - One-to-many association → Multiplicity 1 ↔ 0..n (star) or 1..n
  - Many-to-many association → Multiplicity 0..n or 1..n on both ends

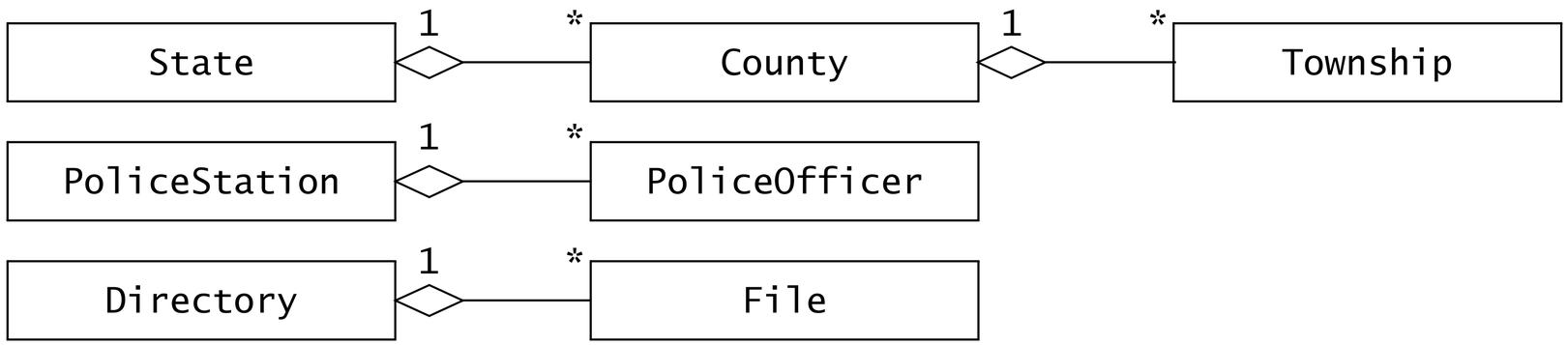



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Aggregation

- A special case of an association.
- Denotes containing or composition.

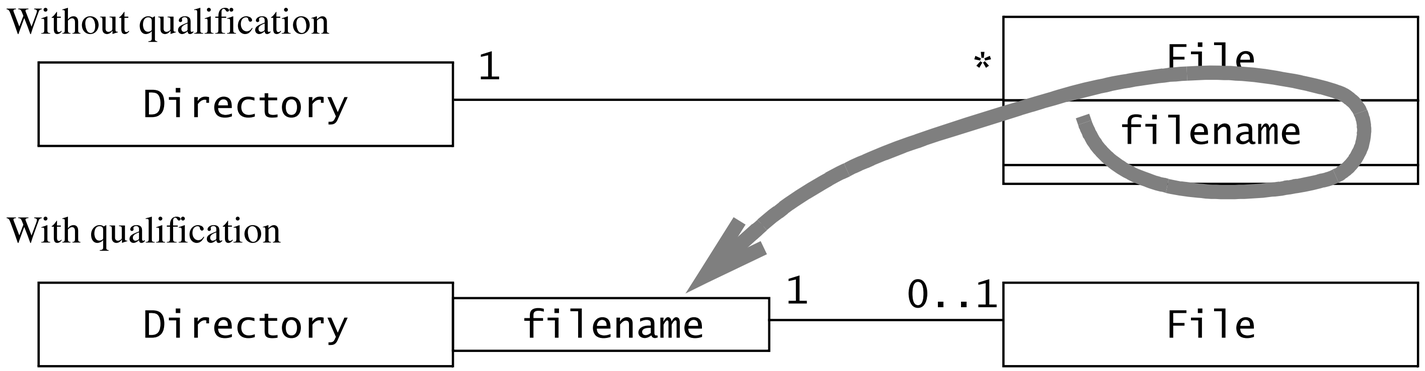



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Qualification

- A technique for reducing multiplicity by using keys.
- The relationship is called a qualified association.




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

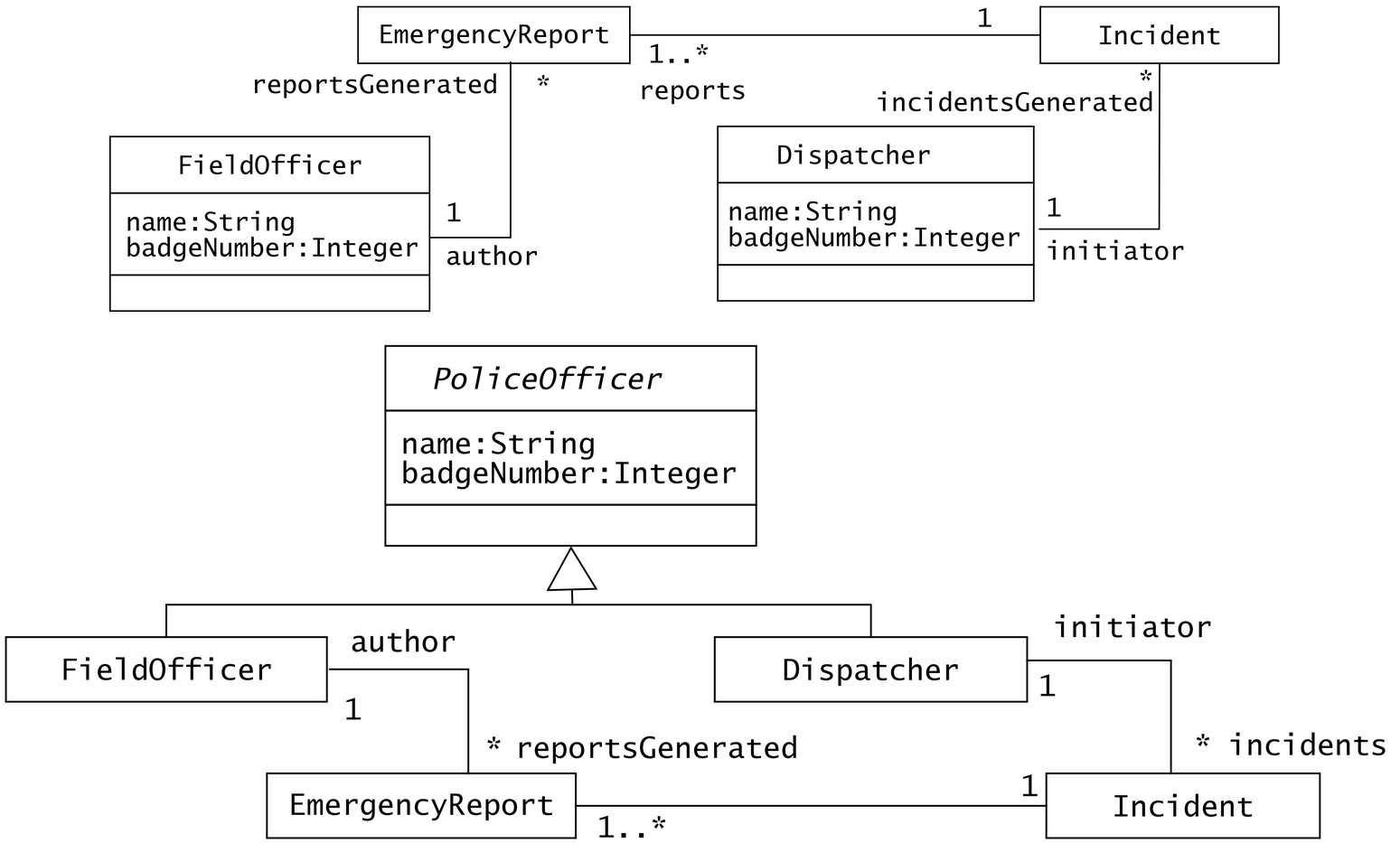
# Class Diagrams

## Inheritance

- The relationship between a general class and one or more specialized classes.
  - Generalization → Superclass
  - Specialization → Subclass

# Class Diagrams

## Inheritance




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Class Diagrams

## Operation vs Method

- Specification of behavior → Operation
- Implementation of behavior → Method
- UML distinguishes operations from methods.
- In practice, developers usually simply refer to methods.

# Interaction Diagrams

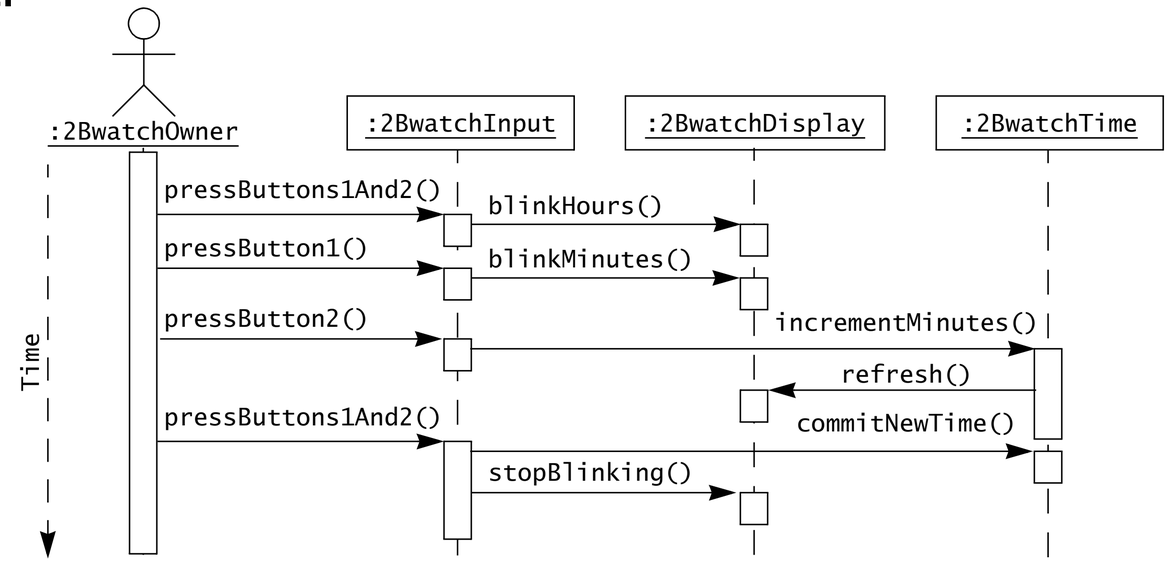
# Interaction Diagrams

- Describe patterns of communication among a set of interacting objects.
- Interaction → Sending messages
- Interaction diagrams
  - Sequence diagrams
  - Communication diagrams

# Interaction Diagrams

## Sequence Diagram

- Sequence diagram
  - Objects → Horizontal
  - Time → Vertical




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

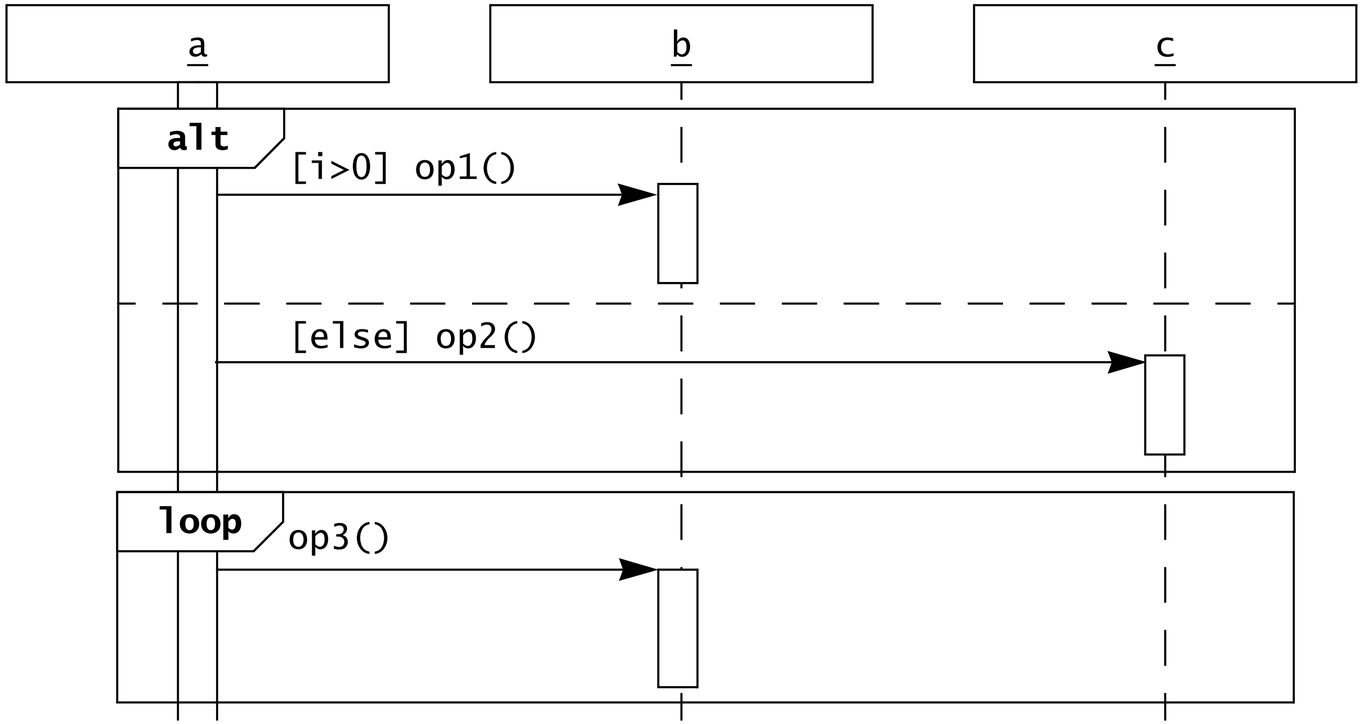
# Interaction Diagrams

## Sequence Diagram

- Sequence diagrams can be used to describe
  - Abstract sequence → All possible interactions
  - Concrete sequences → One possible interaction
- Abstract sequence → Iterations and conditionals

# Interaction Diagrams

## Sequence Diagram - Iteration and Conditional

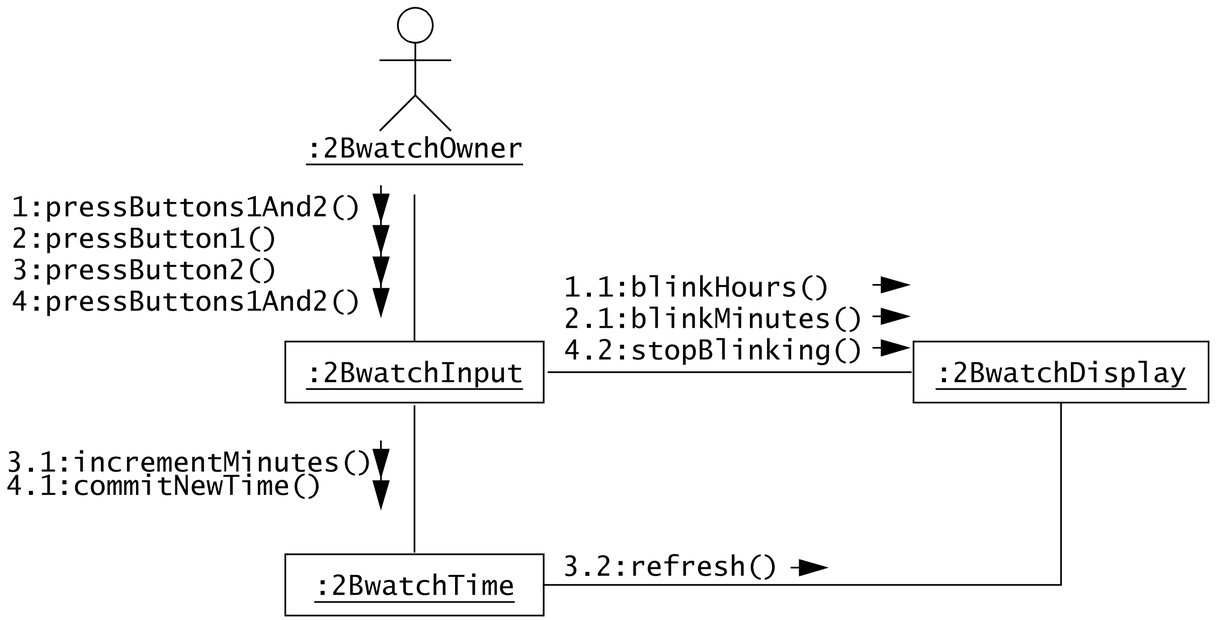



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Interaction Diagrams

## Communication Diagram

- Depict the same information as sequence diagrams.
- Represent the sequence of messages by numbering the interactions.

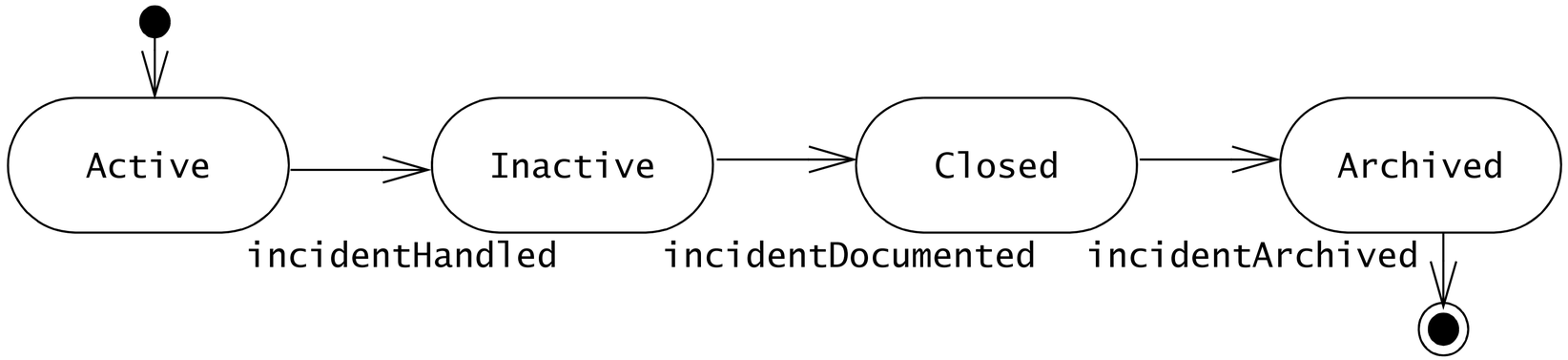



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# State Machine Diagrams

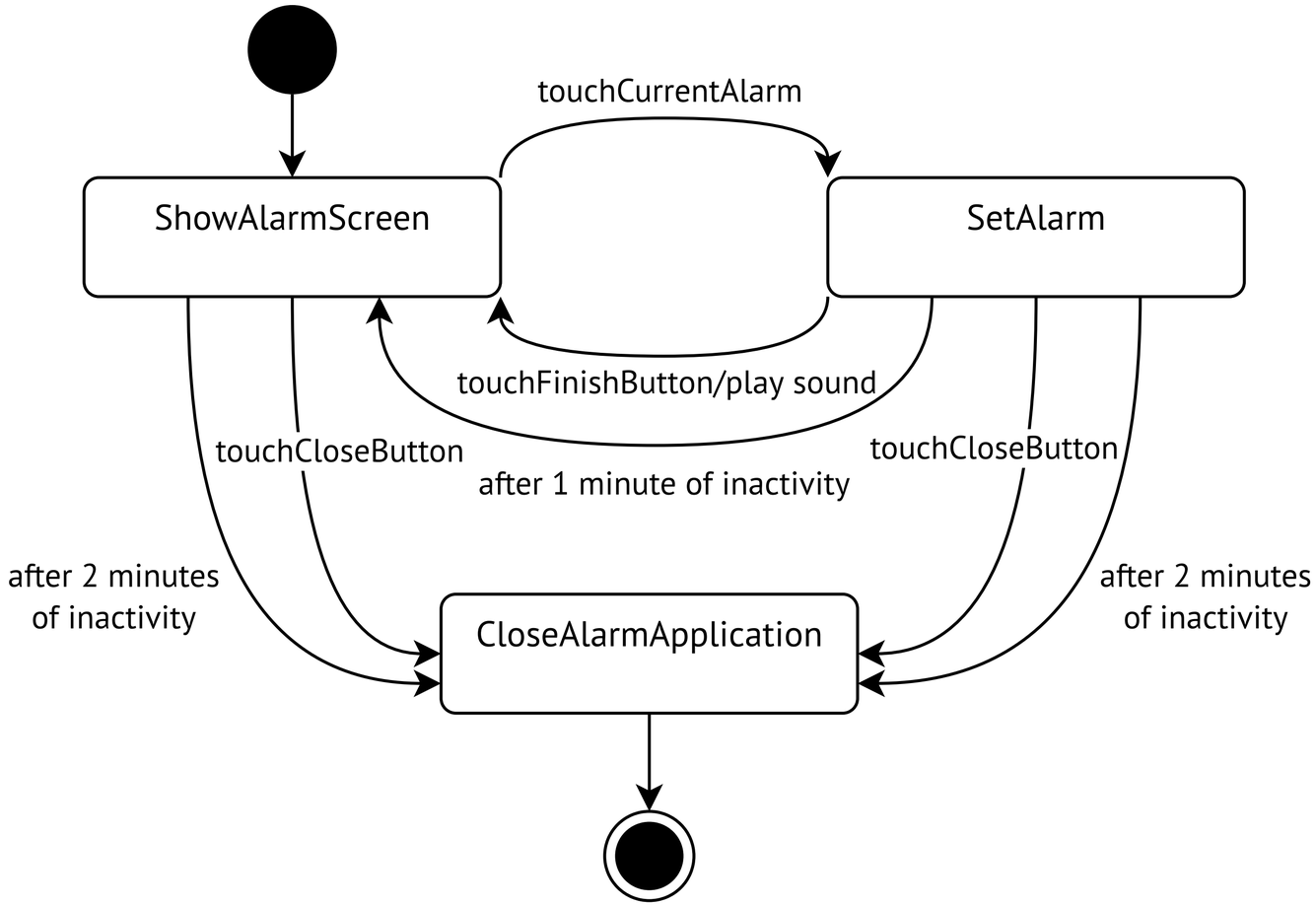
# State Machine Diagrams

- Describe the sequence of states of objects.
- State → Condition satisfied by the attributes of an object
- Transition → Change of state triggered by events, conditions, or time.




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# State Machine Diagrams



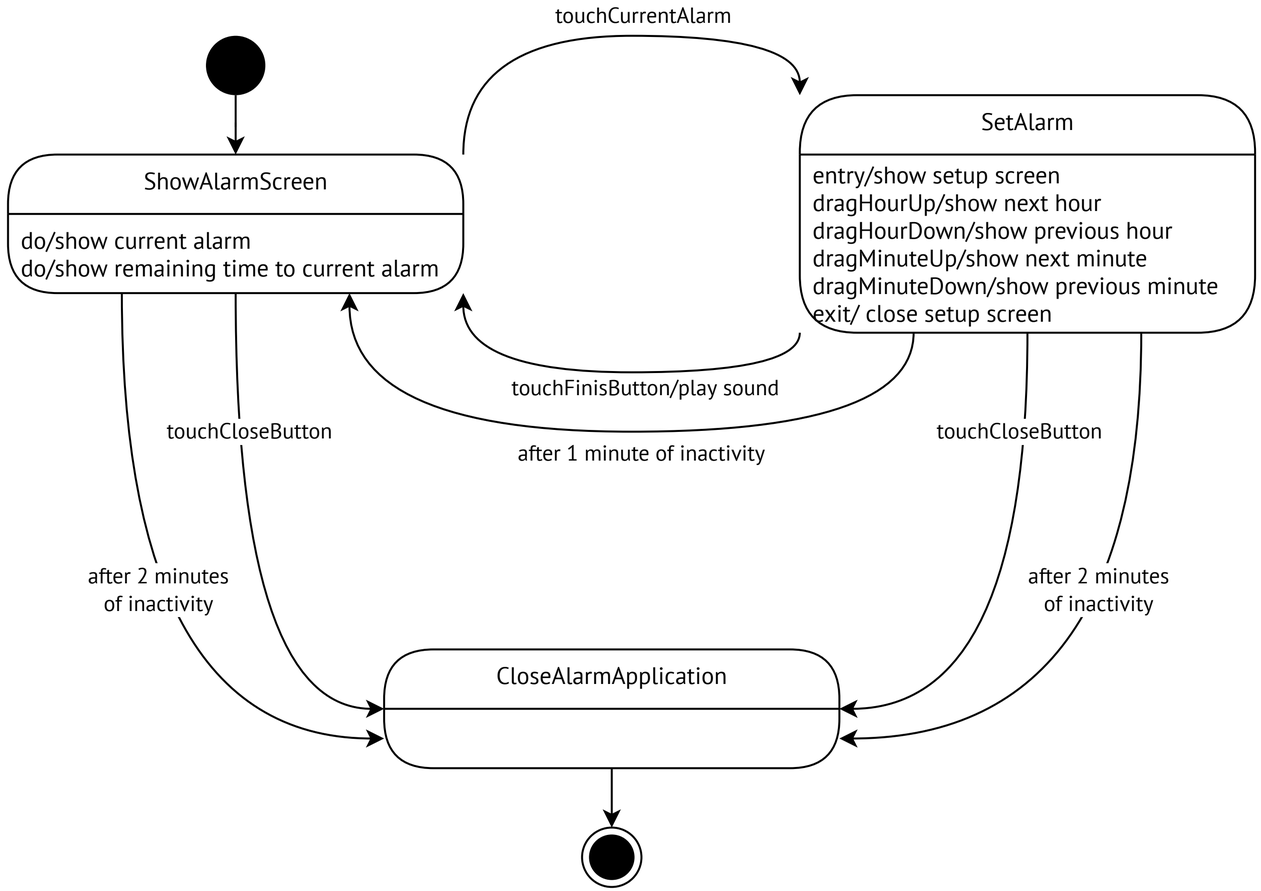
# State Machine Diagrams

- Four terms of state machine diagrams;
  - Action
  - Internal transition
  - Activity
  - Nested state machine

# State Machine Diagrams

## Actions

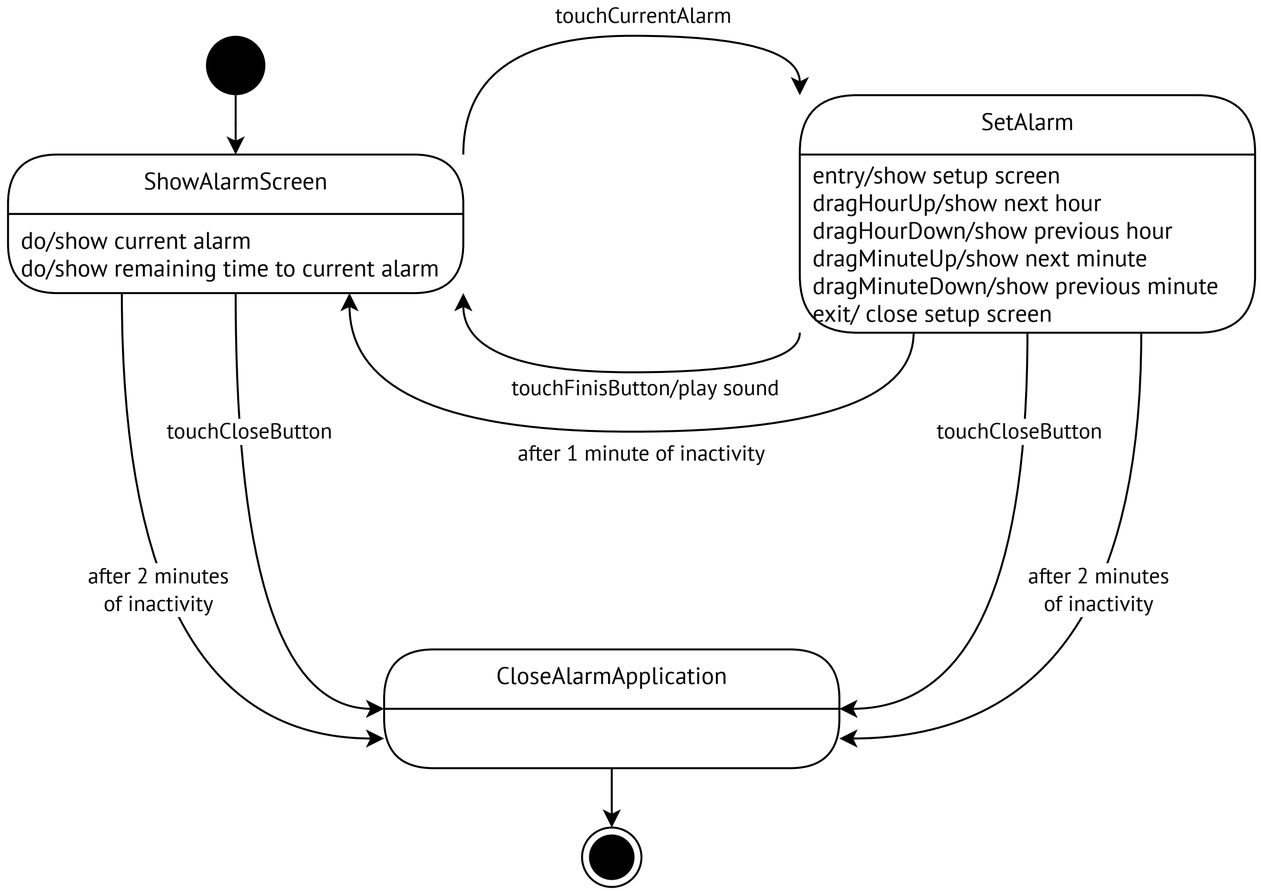
- Behavior within the states.
- Fundamental units of processing
- Can take a set of inputs
- Produce a set of outputs
- Can change the state of the system.
- Not interruptible.



# State Machine Diagrams

## Actions

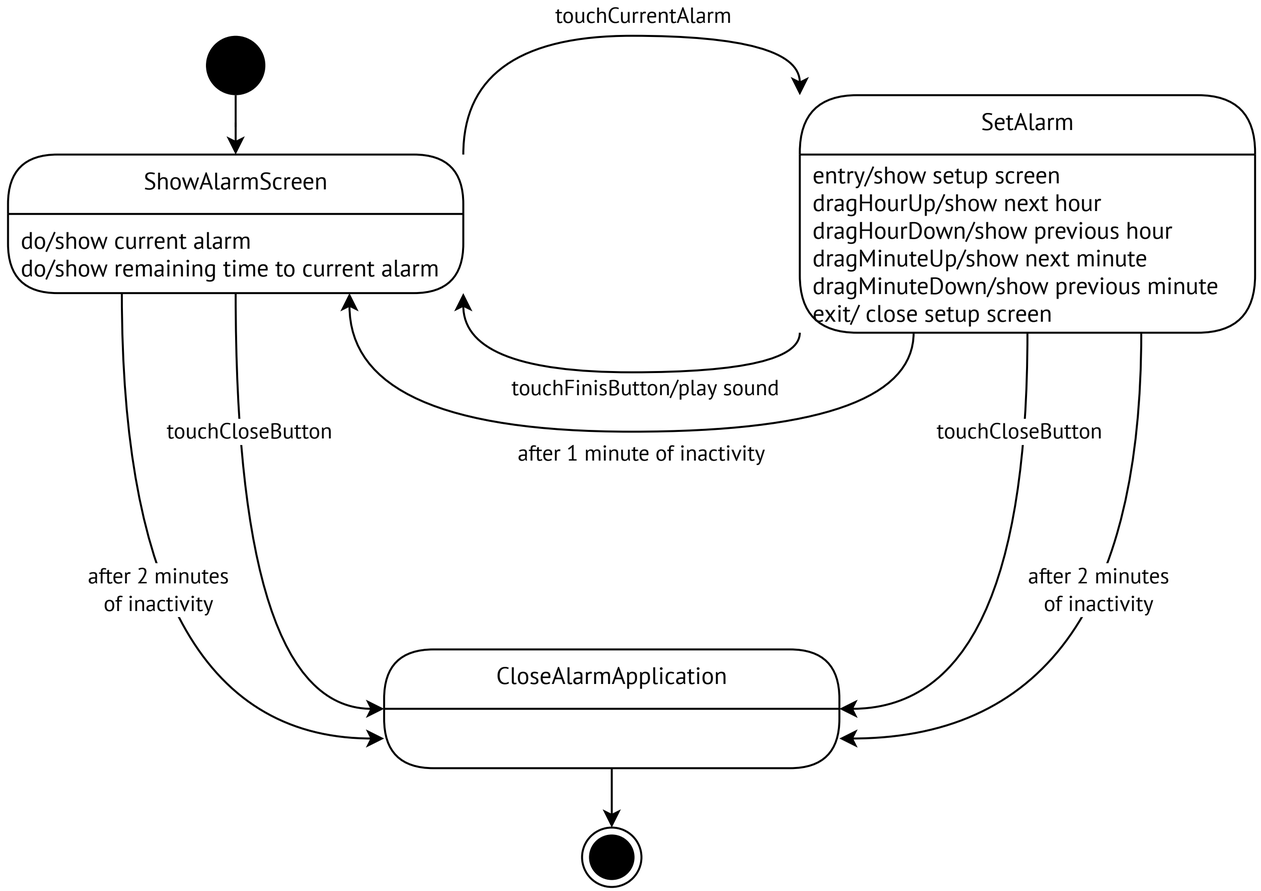
- Happen in three different times:
  - when a transition is taken
  - when a state is entered
  - when a state is exited



# State Machine Diagrams

## Internal Transition

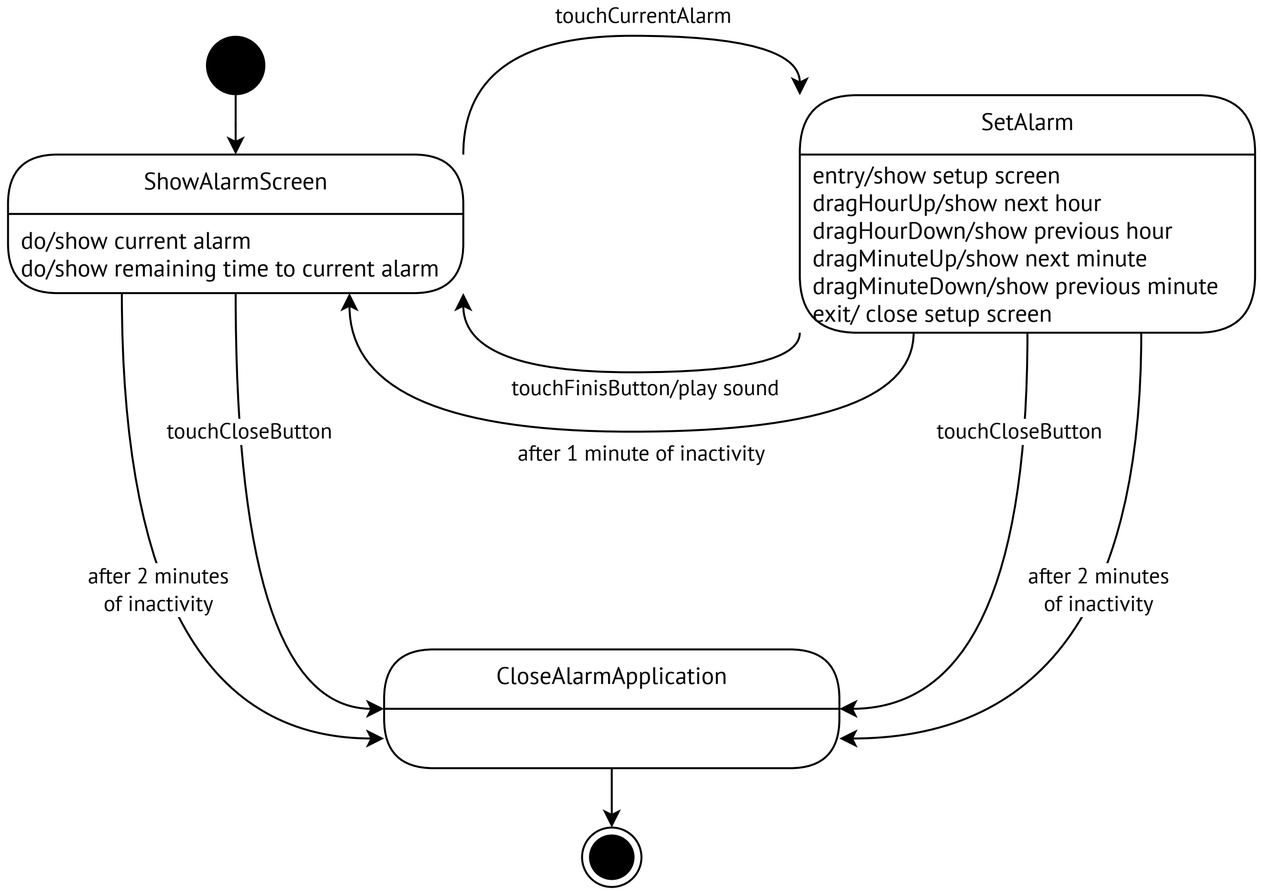
- A transition that does not leave the state.
- Triggered by events.
- Can have actions associated with them.
- Does not result in the execution of any exit or entry actions.



# State Machine Diagrams

## Activity

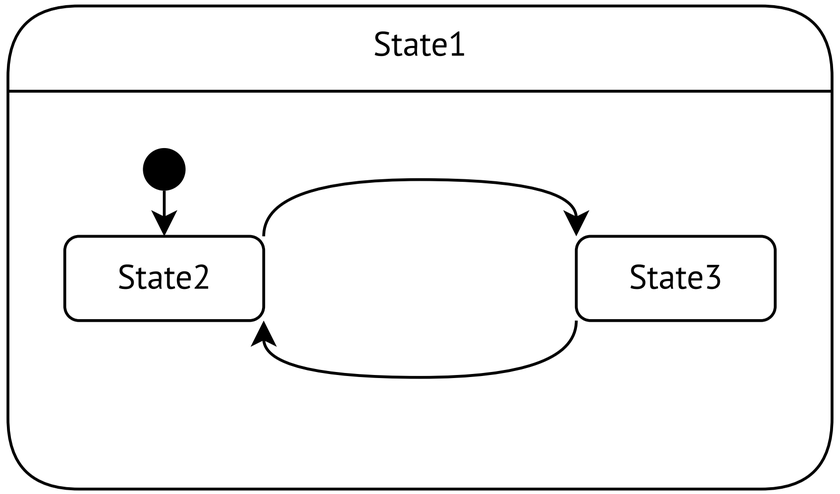
- A coordinated set of actions.
- Can take a substantial amount of time
- Is interrupted on exit.
- Activities are associated with state using the do label.



# State Machine Diagrams

## Nested State Machines

- Reduce complexity.
- Can be used instead of internal transitions.
- Each state could be modeled as a nested state machine.



# Activity Diagrams

# Activity Diagrams

- Sequencing and coordination of lower level behaviors.
- Denotes how a behavior is realized in terms of;
  - One or several sequences of activities
  - The object flows needed for coordinating the activities


Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Activity Diagrams

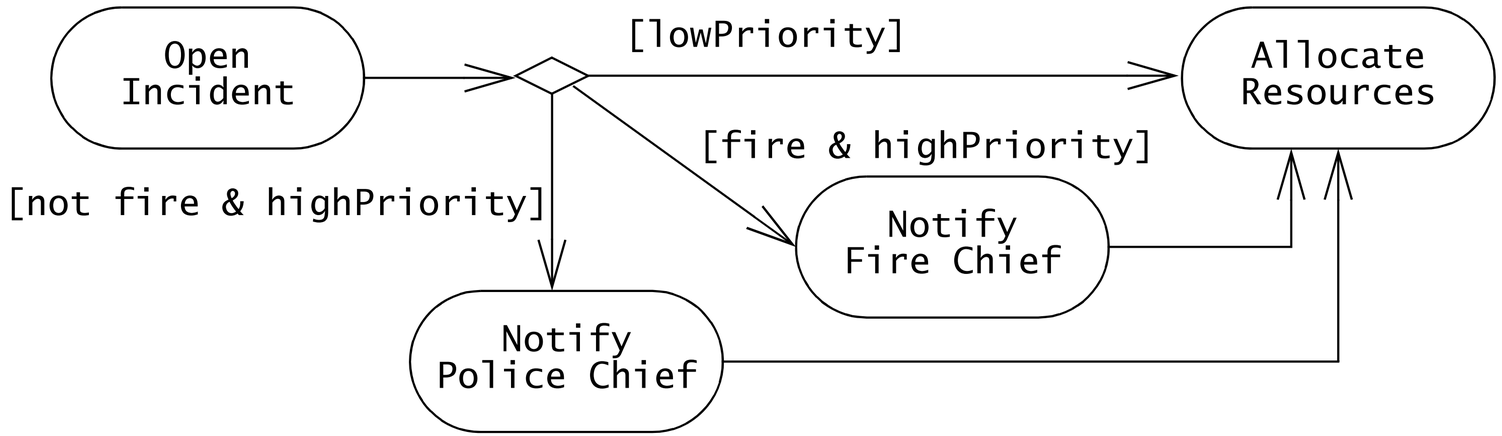
## Control Nodes

- Coordinate control flows in an activity diagram.
- Provide mechanisms for representing decisions, concurrency, and synchronization.
- Main control nodes
  - Decisions
  - Fork nodes
  - Join nodes

# Activity Diagrams

## Control Nodes → Decisions

- Decisions are branches in the control flow.
- Denote alternatives based on a condition of the state of an object or a set of objects.

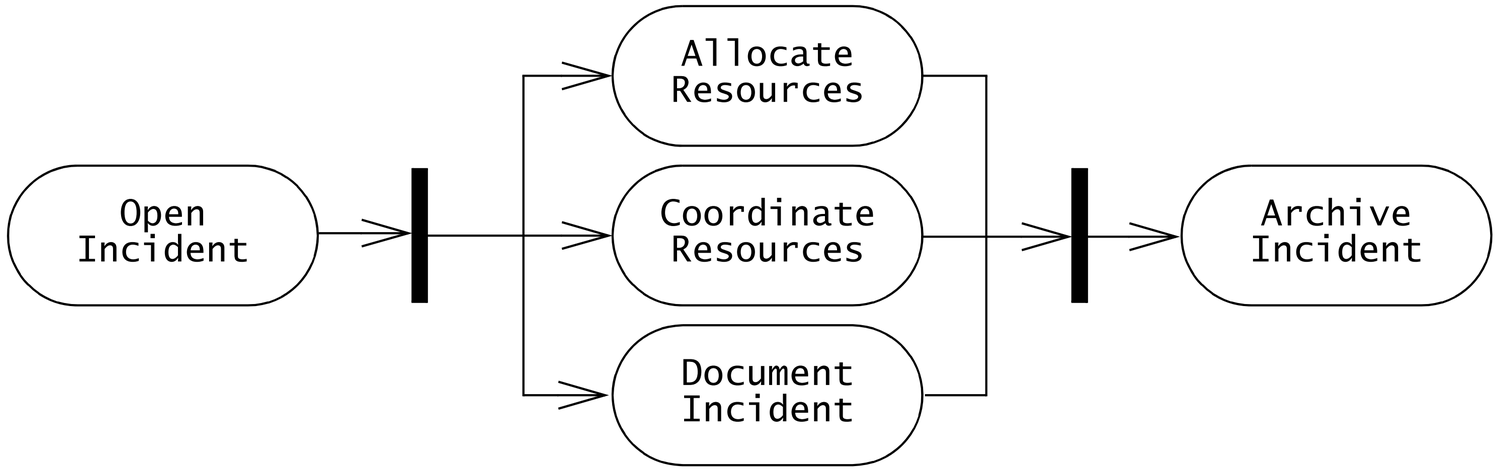



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Activity Diagrams

## Control Nodes → Fork and Join Nodes

- **Fork nodes** and **join nodes** represent concurrency.
- Fork nodes denote the splitting of the flow of control into multiple threads.
- Join nodes denotes the merging of the flow of control into a single thread.

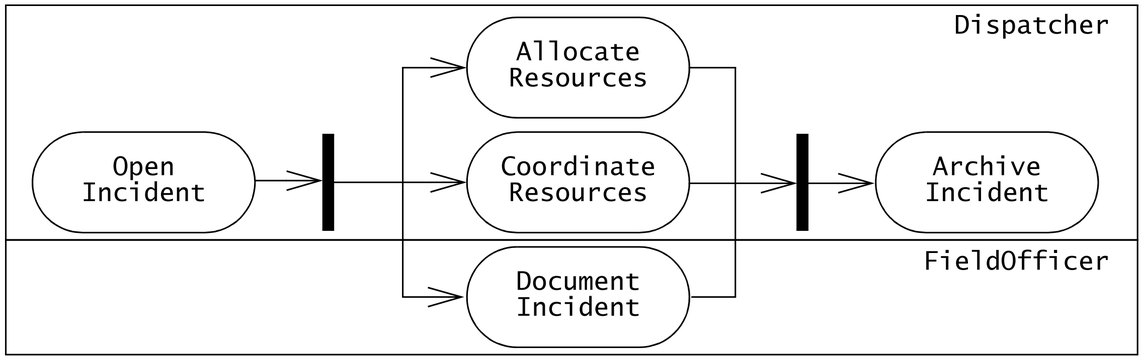



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Activity Diagrams

## Swim-Lane

- Used to group activities.
- Also called activity partitions.
- Aim is to denote the object or subsystem that implements the actions.
- Transitions may cross swim-lanes.



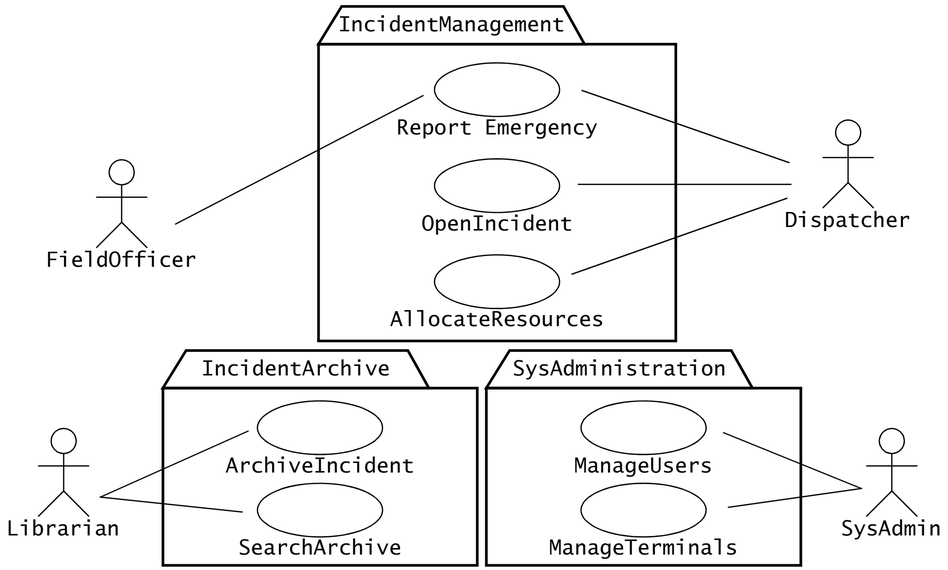

Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Diagram Organization

# Diagram Organization

## Packages

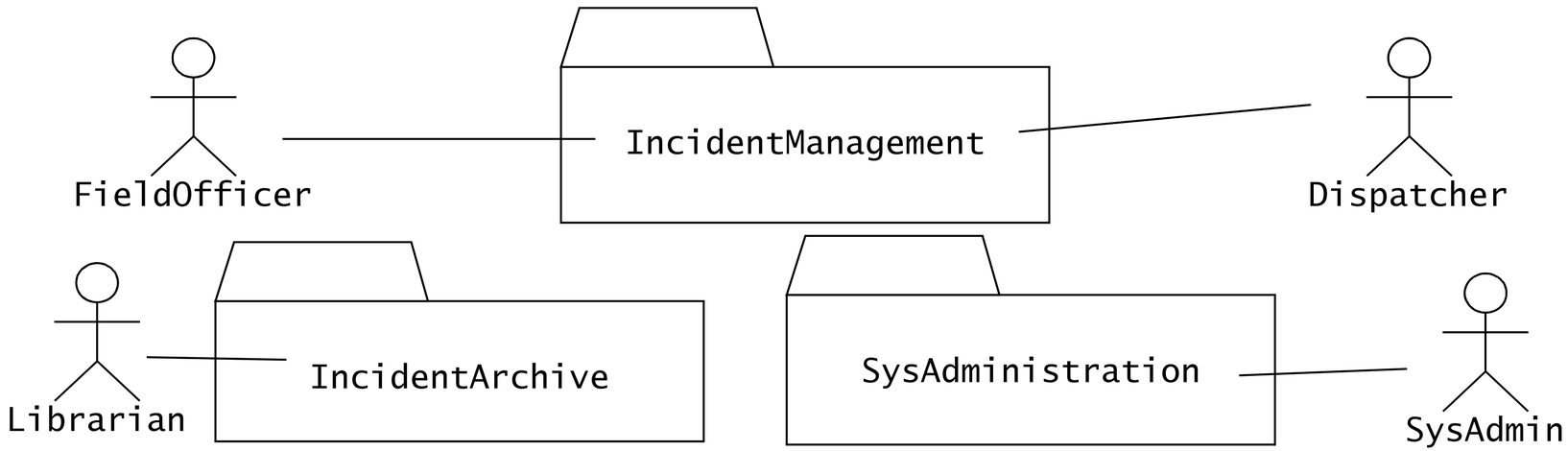
- Models can quickly become complex as developers refine them.
- Manage the complexity of models by grouping related elements into **packages**.




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Diagram Organization

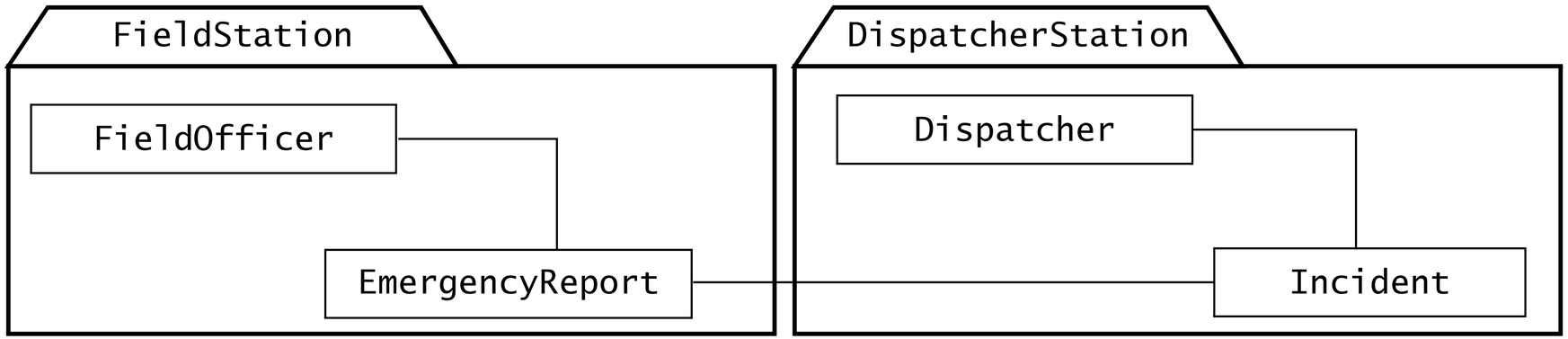
## Packages




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Diagram Organization

## Packages

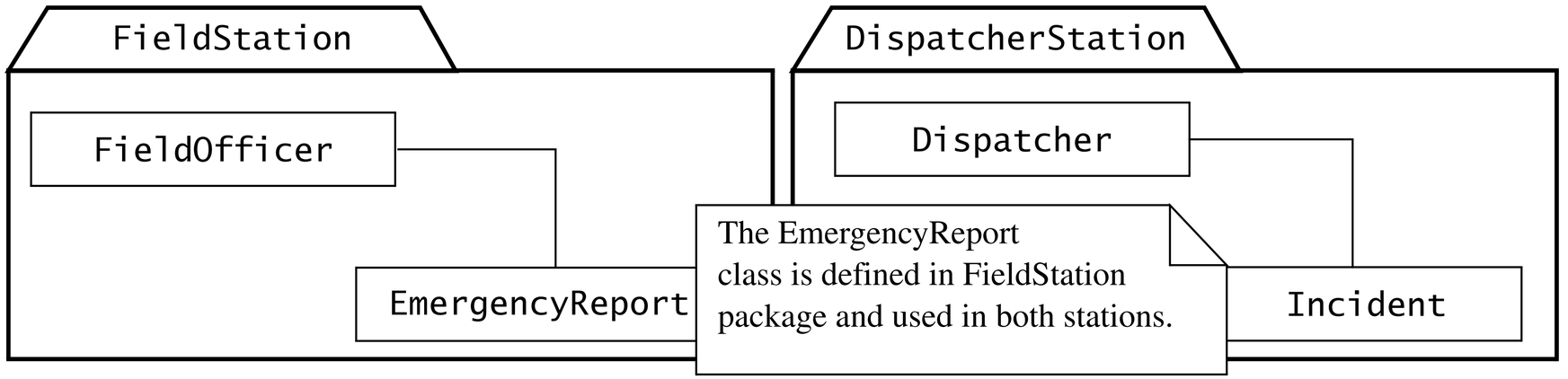



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Diagram Organization

## Notes

- Comments attached to diagrams.
- Used for attaching information to models and model elements.




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Diagram Extensions

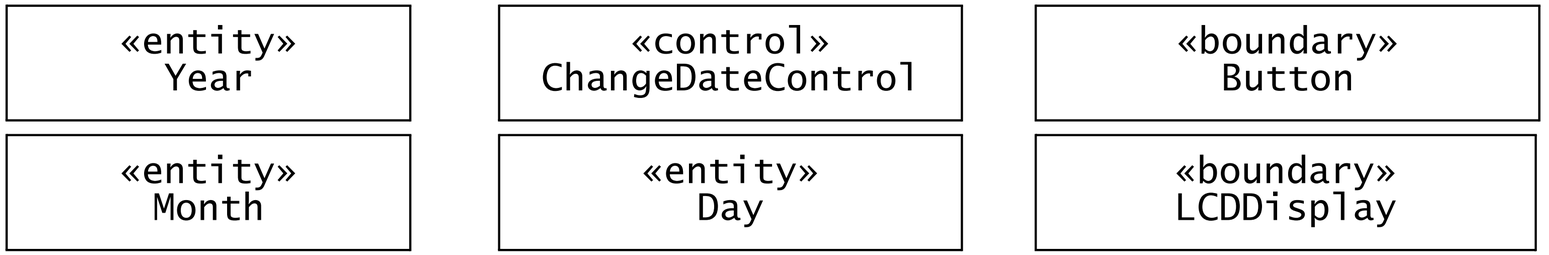
# Diagram Extensions

- The goal of the UML:
  - Provide a set of notations to model a broad class of software systems.
- Sometimes fixed notations may not be sufficient.
- Solution: Extension mechanisms
  - Enabling the modeler to extend the language.
- Two of them: **stereotypes** and **constraints**.

# Diagram Extensions

## Stereotype

- Allows developers to classify model elements in UML.
- A stereotype is represented by string enclosed by angle brackets
- Attached to the model element to which it applies, such as a class or an association.

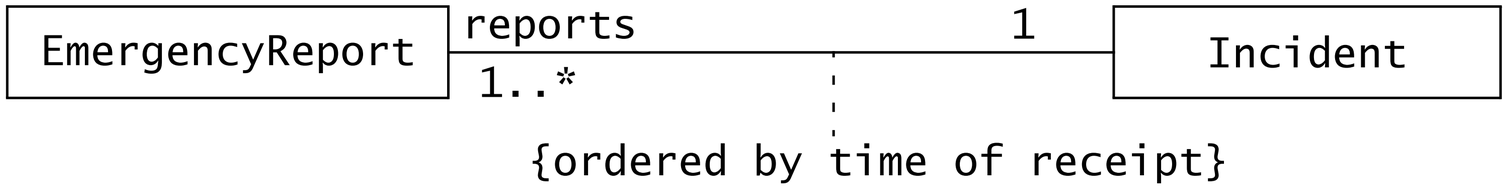



Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# Diagram Extensions

## Constraint

- A rule that is attached to a model element restricting its semantics.
- Represent phenomena that cannot otherwise be expressed with UML.
- Constraints can be expressed in two ways:
  - By using an informal string
  - By using a formal language such as OCL (Object Constraint Language).




Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.

# References


- Bernd Bruegge, Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, 3rd Edition, Pearson, 2014.
- Object Management Group, OMG Unified Modeling Language Superstructure, Version 2.2., http://www.omg.org/2009.

# Thank you.