University of Leicester

MSc programme (induction week) – Computer Science Department

INTRODUCTION TO UML

Some of this material is based on

Bernd Bruegge and Allen H. Dutoit (2009) ‘Object-Oriented Software Engineering: Using UML, Patterns, and Java’, Pearson, 3rd edition.

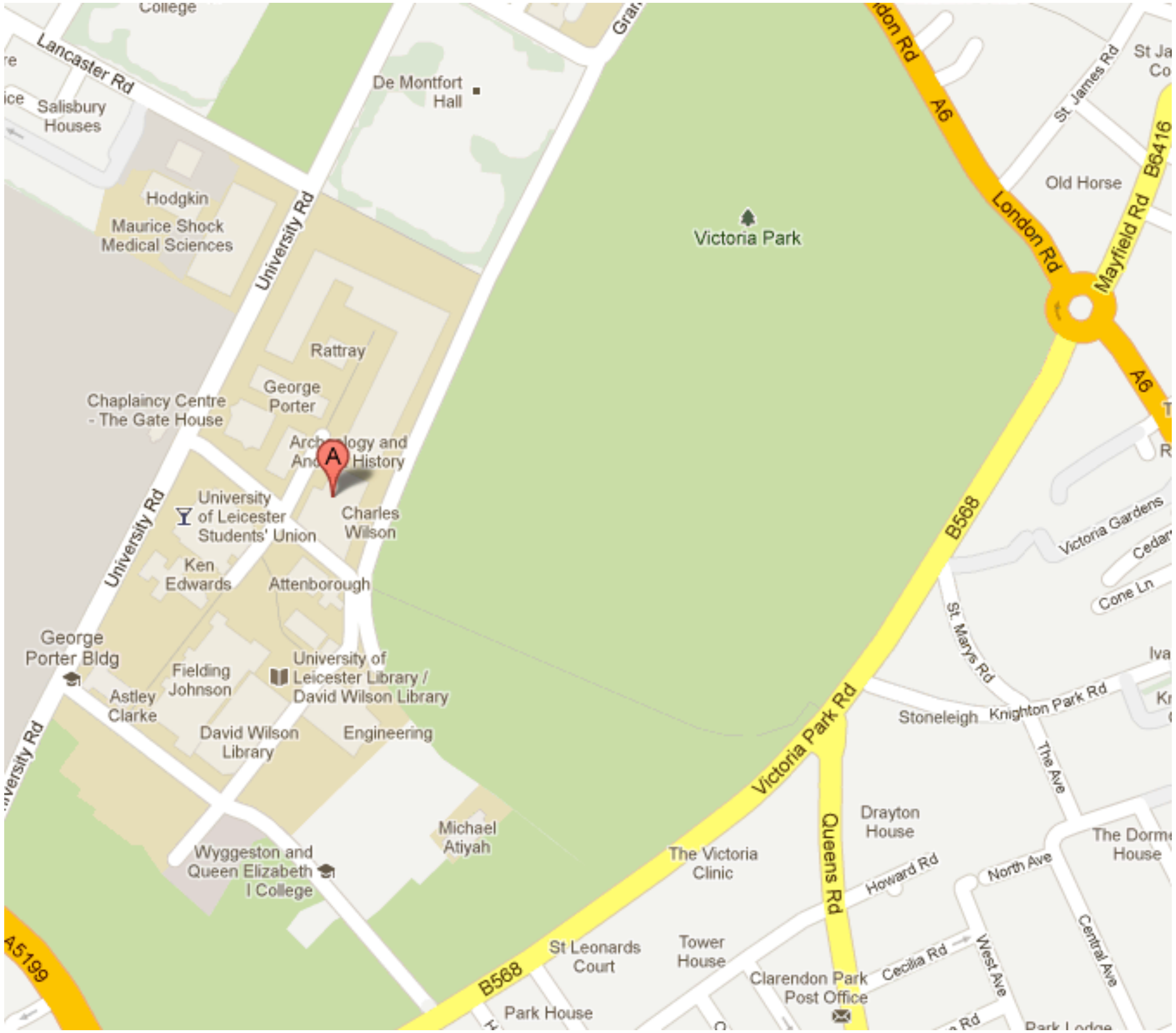
Overview: modelling with UML

- What is modelling?
- What is UML?
- Use case diagrams
- Class diagrams
- Sequence diagrams
- Activity diagrams

What is modelling?

- Modelling consists of building an abstraction of reality.
- Abstractions are simplifications because:
  - **They ignore irrelevant details and**
  - **They only represent the relevant details.**
- What is *relevant* or *irrelevant* depends on the purpose of the model.

Example: street map



Why model software?

- Software is getting increasingly more complex:
  - **Windows XP > 40 million lines of code.**
  - **A single programmer cannot manage this amount of code in its entirety.**
- Code is not easily understandable by developers who did not write it.
- We need simpler representations for complex systems:
  - **Modelling is a means for dealing with complexity.**

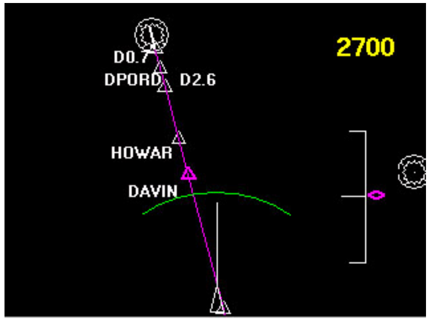
Application and Solution Domain

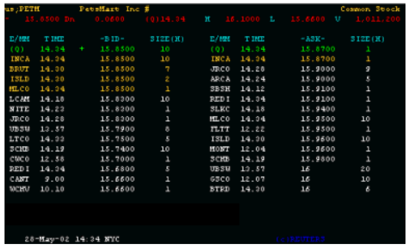
- Application Domain (Requirements Analysis):
  - **The environment in which the system is operating**

- Solution Domain (System Design, Object Design):
  - **The available technologies to build the system**

Object-oriented Modelling

Application Domain
(Phenomena)



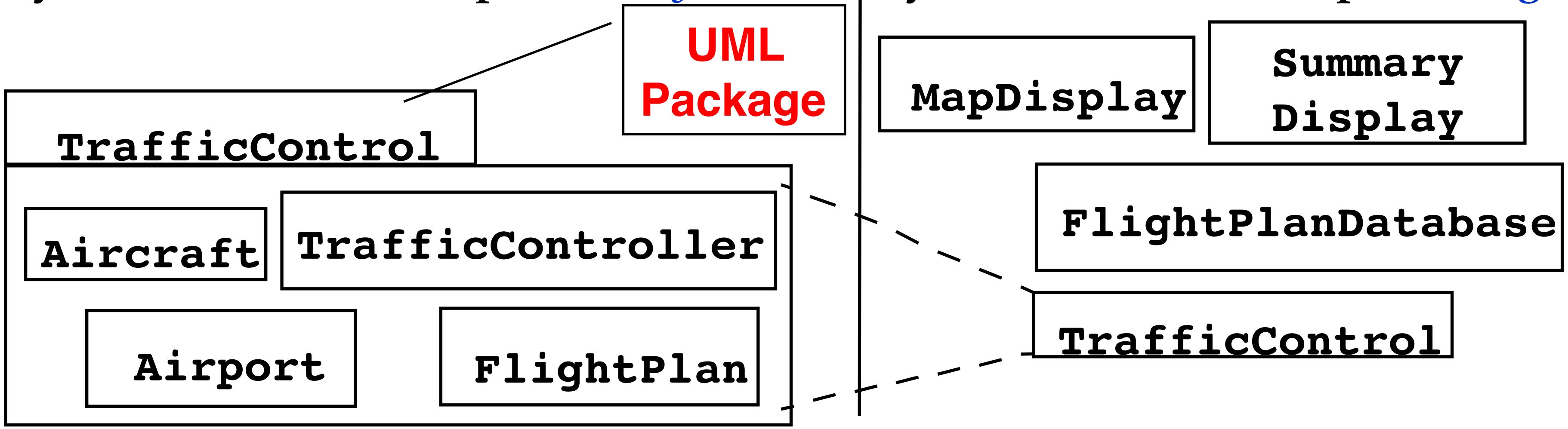


```
us;PETM          PetsMart Inc #                              Common Stock
-   15.8500 Dn    0.0600    (Q)14.34   H   16.1000  L   15.6600  V  1,011,200

E/MM  TIME        -BID-    SIZE(H)    E/MM  TIME       -ASK-    SIZE(H)
(Q)   14.34  +   15.8500     10       (Q)   14.34     15.8700      1
INCA  14.34      15.8500     10       INCA  14.34     15.8700      1
BRUT  14.30      15.8500      7       JRCO  14.28     15.9000      9
ISLD  14.30      15.8500      2       ARCA  14.24     15.9000      5
MLCO  14.34      15.8500      1       SBSH  14.12     15.9100      1
LCAM  14.18      15.8300     10       REDI  14.34     15.9100      1
NITE  14.23      15.8300      1       SLKC  14.18     15.9400      1
JRCO  14.28      15.8300      1       MLCO  14.34     15.9500     10
UBSW  13.57      15.7900      6       FLTT  12.22     15.9500      1
LTCO  14.33      15.7500      5       ISLD  14.30     15.9600     10
SCHB  14.19      15.7400     10       MONT  12.04     15.9600      1
CWCO  12.58      15.7000      1       SCHB  14.19     15.9800      1
REDI  14.34      15.6800      5       UBSW  13.57     16              20
CANT   9.00      15.6600      1       GSCO  12.07     16              10
WCHV  10.10      15.6600      1       BTRD  14.30     16               6


  28-May-02 14:34 NYC                              (c)REUTERS
```

Solution Domain
(Phenomena)

System Model (Concepts) *(Analysis)*

System Model (Concepts) *(Design)*



What should be done first? Coding or Modelling?

- It all depends….
- Forward Engineering
  - **Creation of code from a model**
  - **Start with modelling**
  - **Greenfield projects**
- Reverse Engineering
  - **Creation of a model from existing code**
  - **Interface or reengineering projects**
- Roundtrip Engineering
  - **Move constantly between forward and reverse engineering**
  - **Reengineering projects**
  - **Useful when requirements, technology and schedule are changing frequently.**

What is UML? Unified Modelling Language

- Convergence of different notations used in object-oriented methods, mainly
  - **OMT (James Rumbaugh and collegues), OOSE (Ivar Jacobson), Booch (Grady Booch)**

- They also developed the Rational Unified Process, which became the Unified Process in 1999

Origins

- OO programming languages
- OO analysis and design techniques
  - **business modelling**
  - **analysis of requirements**
  - **design of software systems**
- UML: industry standard that merges the best features of different notations

What UML is not

- UML is not a programming language per se
- UML is not a software modelling tool
- UML is not a method, methodology or software development process

Why UML?

- De facto standard for OO modelling
- Unified modelling language
- UML provides extension mechanisms

Main diagram notations

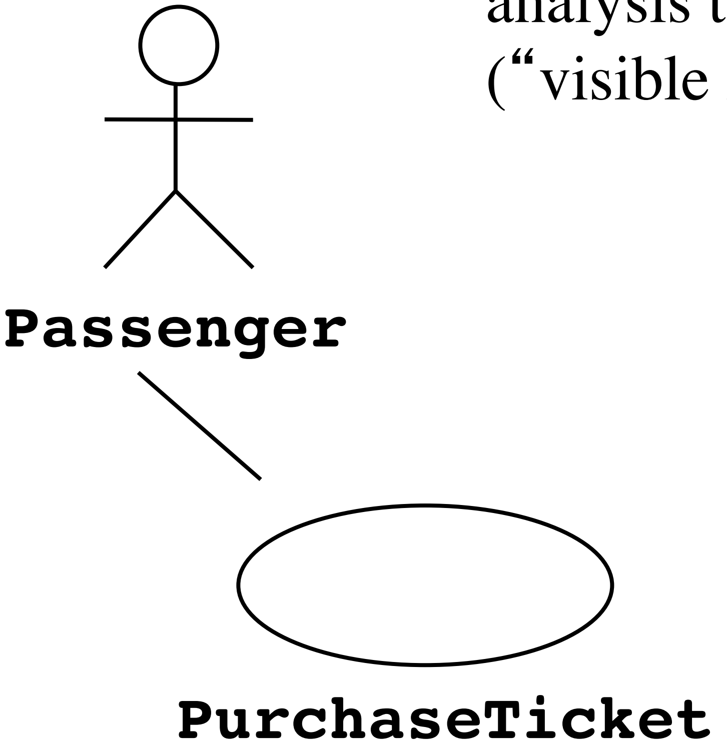
- Use case diagrams
- Class diagrams and object diagrams
- Component diagrams
- Interaction diagrams
- Activity diagrams
- State machines
- Deployment diagrams

UML overview

- Use case diagrams
  - **Describe the functional behaviour of the system as seen by the user.**
- Class diagrams
  - **Describe the static structure of the system: objects, attributes, associations.**
- Sequence diagrams
  - **Describe the dynamic behaviour between objects of the system.**
- Statechart diagrams
  - **Describe the dynamic behaviour of an individual object.**
- Activity diagrams
  - **Describe the dynamic behaviour of a system, in particular the workflow.**

UML Use Case Diagrams

Used during requirements elicitation and analysis to represent external behaviour ("visible from the outside of the system")



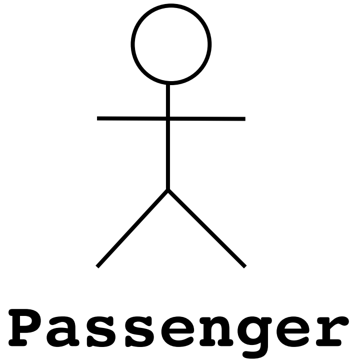
An ***Actor*** represents a role, that is, a type of user of the system

A ***use case*** represents a class of functionality provided by the system

Use case model:

The set of all use cases that completely describe the functionality of the system.

Actors

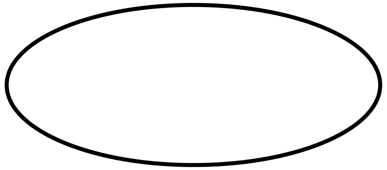


- An actor is a model for an external entity which interacts (communicates) with the system:
  - **User**
  - **External system (Another system)**
  - **Physical environment (e.g. Weather)**
- An actor has a unique name and an optional description
- Examples:
  - **Passenger: A person in the train**
  - **GPS satellite: An external system that provides the system with GPS coordinates.**

Optional Description

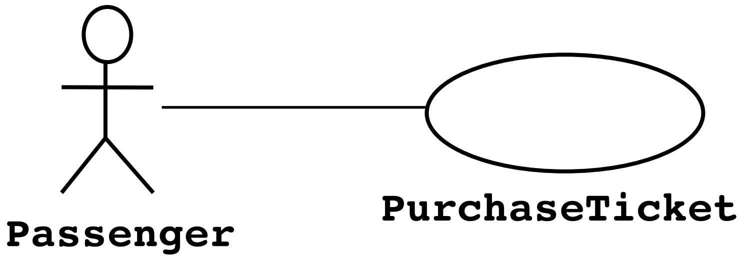
Name

Use Case



- A use case represents a class of functionality provided by the system
- Use cases can be described textually, with a focus on the event flow between actor and system
- The textual use case description consists of 6 parts:
    1. **Unique name**
    2. **Participating actors**
    3. **Entry conditions**
    4. **Exit conditions**
    5. **Flow of events**
    6. **Special requirements.**

Textual Use Case Description Example



1. Name: `Purchase ticket`

2. Participating actor: `Passenger`

3. Entry condition:

- `Passenger` stands in front of ticket distributor
- `Passenger` has sufficient money to purchase ticket

4. Exit condition:

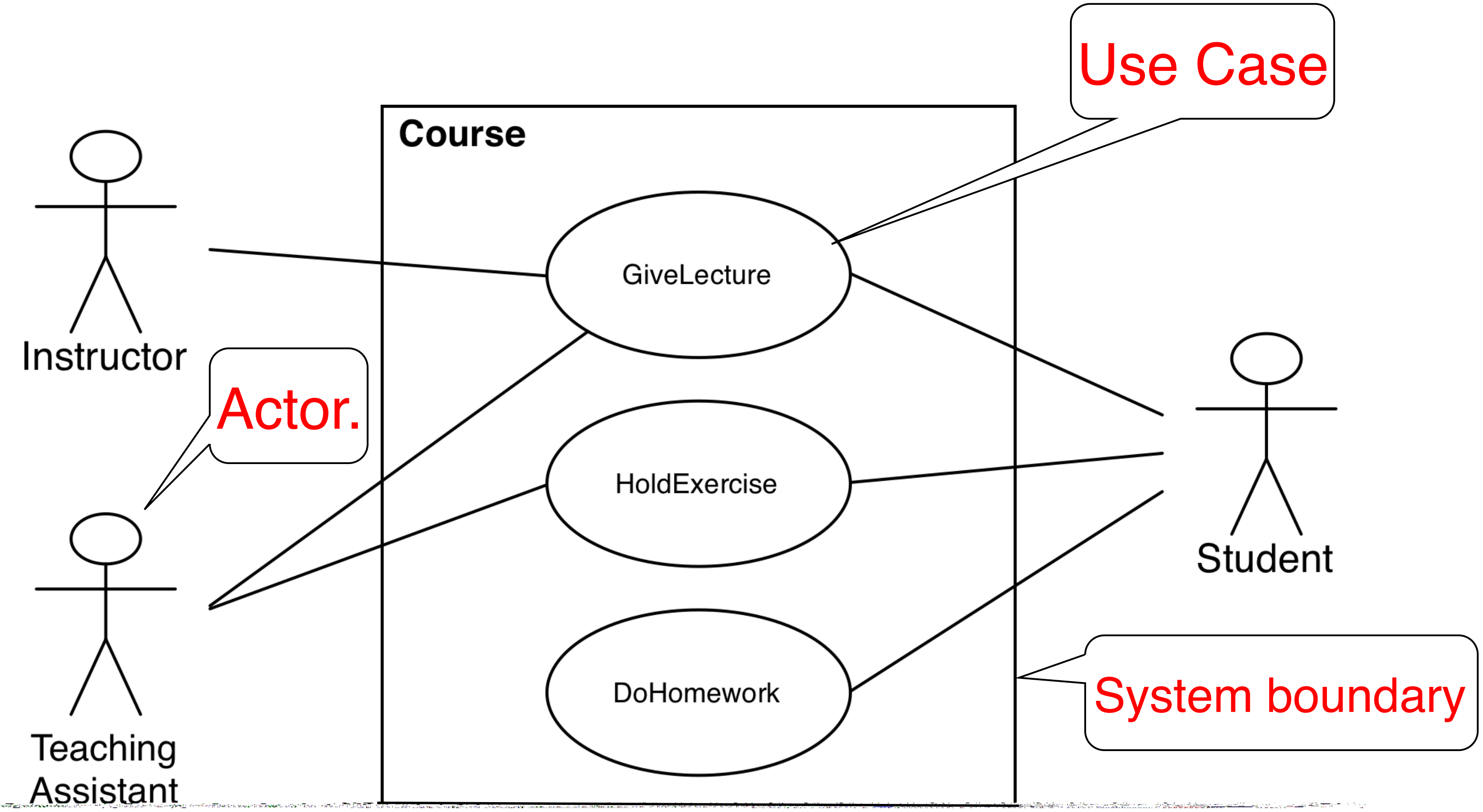
- `Passenger` has ticket

5. Flow of events:

1. **`Passenger` selects the number of zones to be traveled**
2. **Ticket Distributor displays the amount due**
3. **`Passenger` inserts money, at least the amount due**
4. **Ticket Distributor returns change**
5. **Ticket Distributor issues ticket**

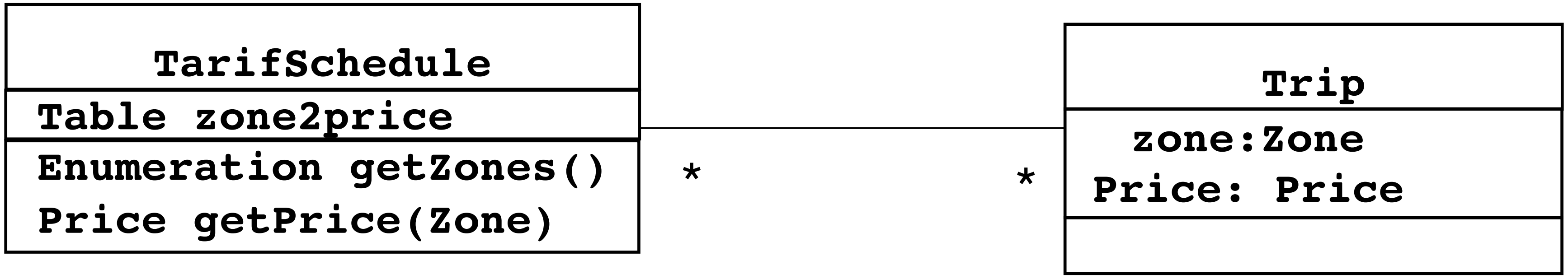
6. Special requirements: None.

Use Case Models should be packaged

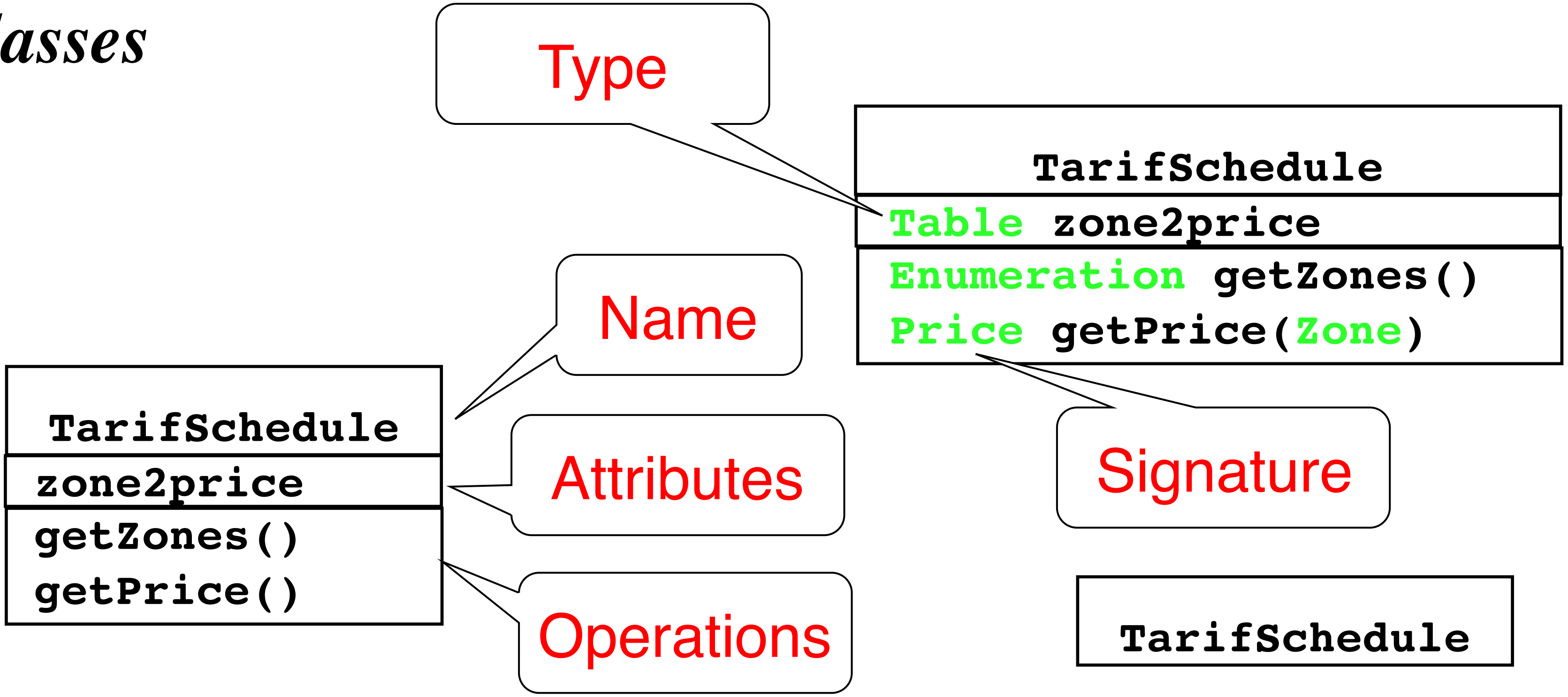


Class Diagrams

- Class diagrams represent the structure of the system
- Used
  - **during requirements analysis to model application domain concepts**
  - **during system design to model subsystems**
  - **during object design to specify the detailed behaviour and attributes of classes.**



Classes

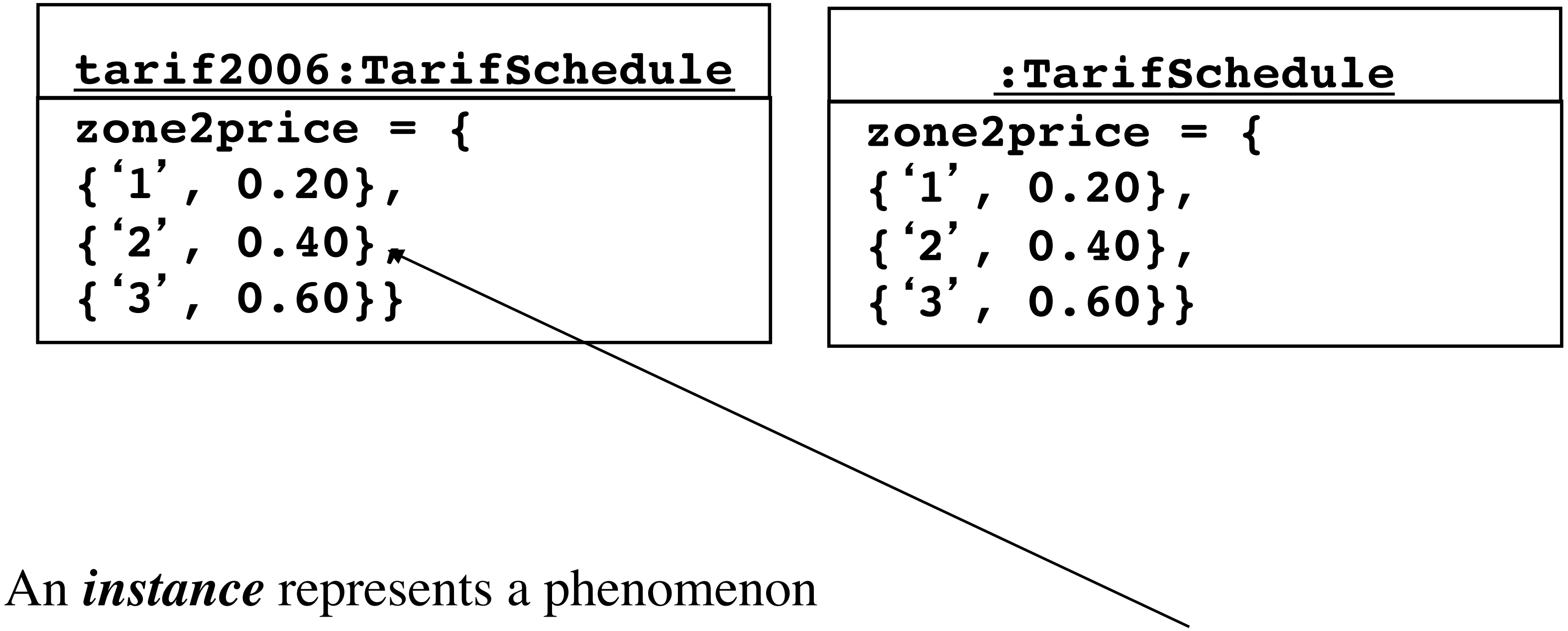


- A ***class*** represents a concept
- A class encapsulates state ***(attributes)*** and behaviour ***(operations)***

Each attribute has a ***type***
Each operation has a ***signature***

The class name is the only mandatory information

Instances



tarif2006:TarifSchedule

zone2price = {
{‘1’, 0.20},
{‘2’, 0.40},
{‘3’, 0.60}}

:TarifSchedule

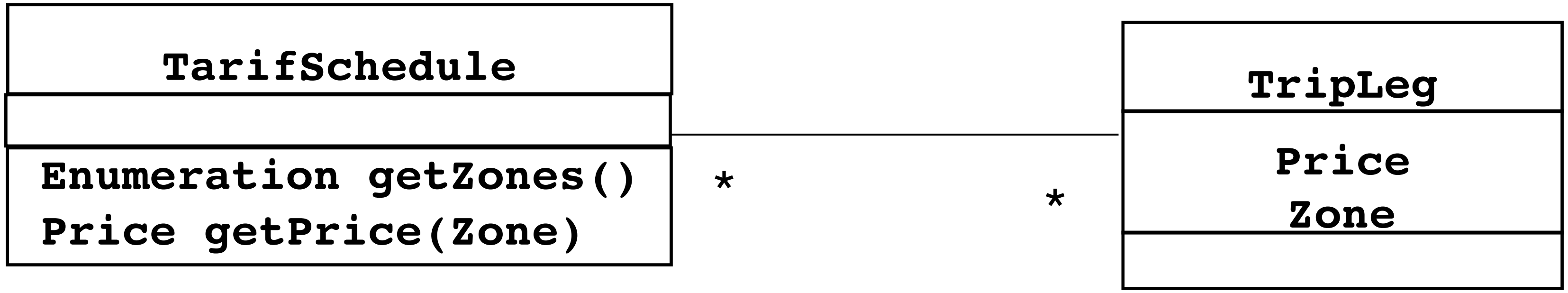
zone2price = {
{‘1’, 0.20},
{‘2’, 0.40},
{‘3’, 0.60}}

- An ***instance*** represents a phenomenon
- The attributes are represented with their ***values***
- The name of an instance is underlined
- The name can contain only the class name of the instance (anonymous instance)

Actor vs Class vs Object

- Actor
  - An entity outside the system to be modelled, interacting with the system (“Passenger”)
- Class
  - An abstraction modelling an entity in the application or solution domain
  - The class is part of the system model (“User”, “Ticket distributor”, “Server”)
- Object
  - A specific instance of a class (“Joe, the passenger who is purchasing a ticket from the ticket distributor”).

Associations

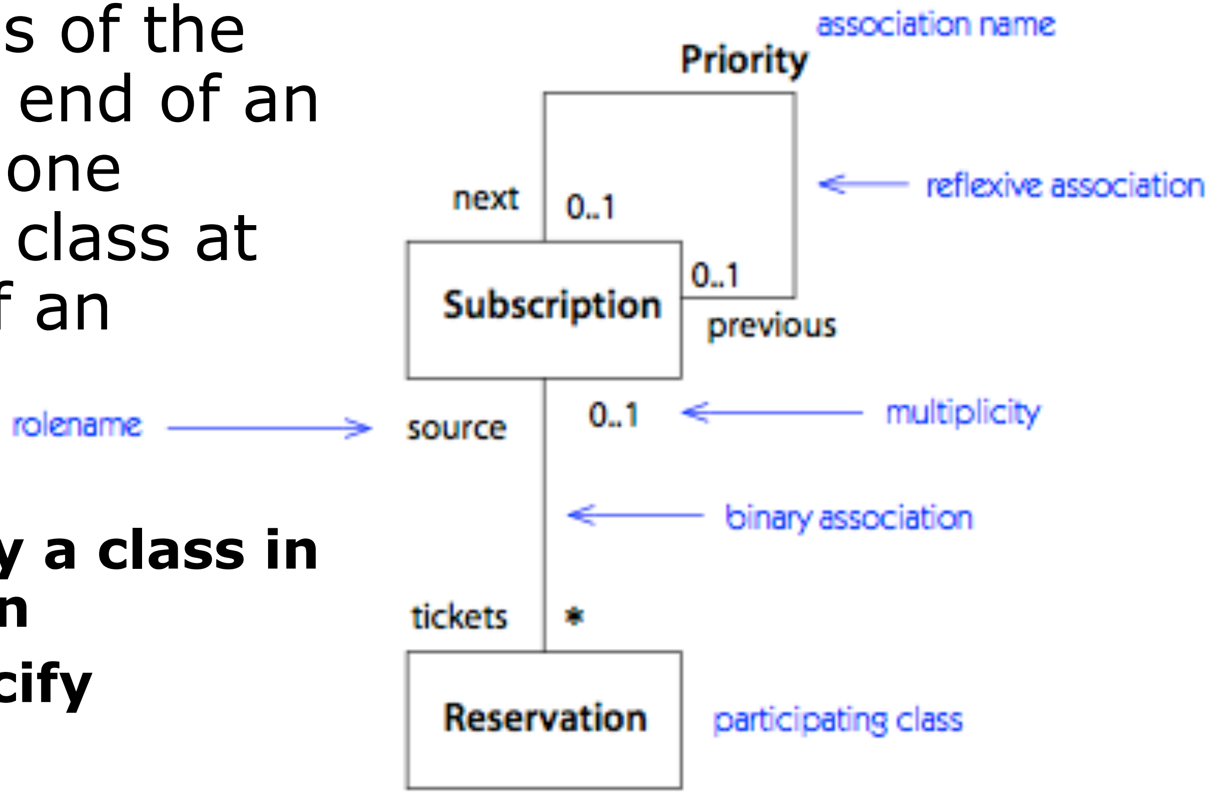


Associations denote collaborations between classes by means of message exchange.

The multiplicity of an association end denotes how many objects the instance of a class can legitimately reference.

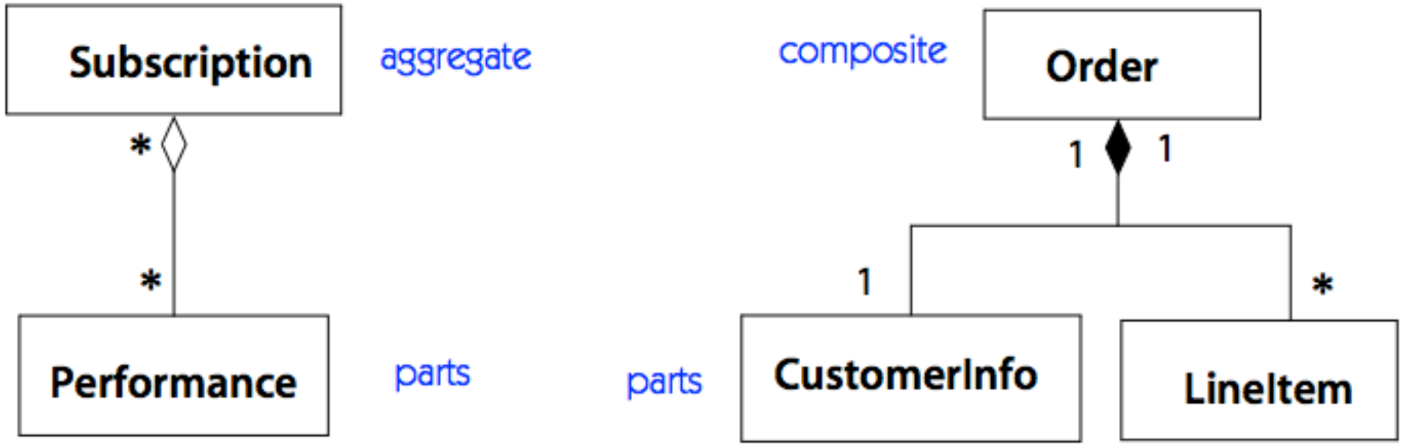
Association properties

- Name
- Multiplicity: number of object instances of the class at the far end of an association for one instance of the class at the near end of an association
- Role names
  - **role played by a class in an association**
  - **useful to specify methods**

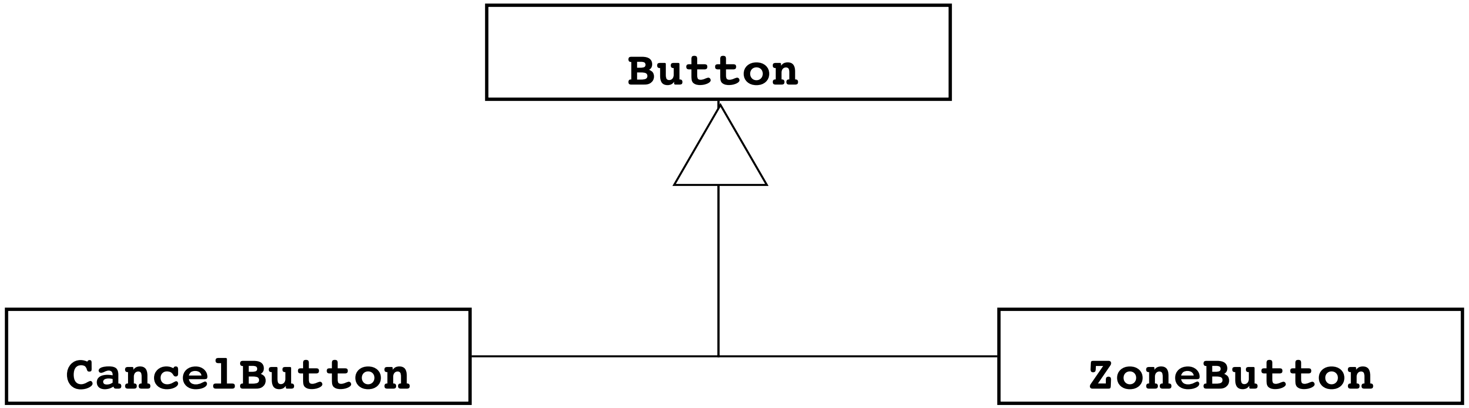


Aggregation

- An *aggregation* is a special case of association denoting that one class may consist of, or include, instances of another class.
- A solid diamond denotes *composition*: the *life time of the component instances* is controlled by the aggregate.



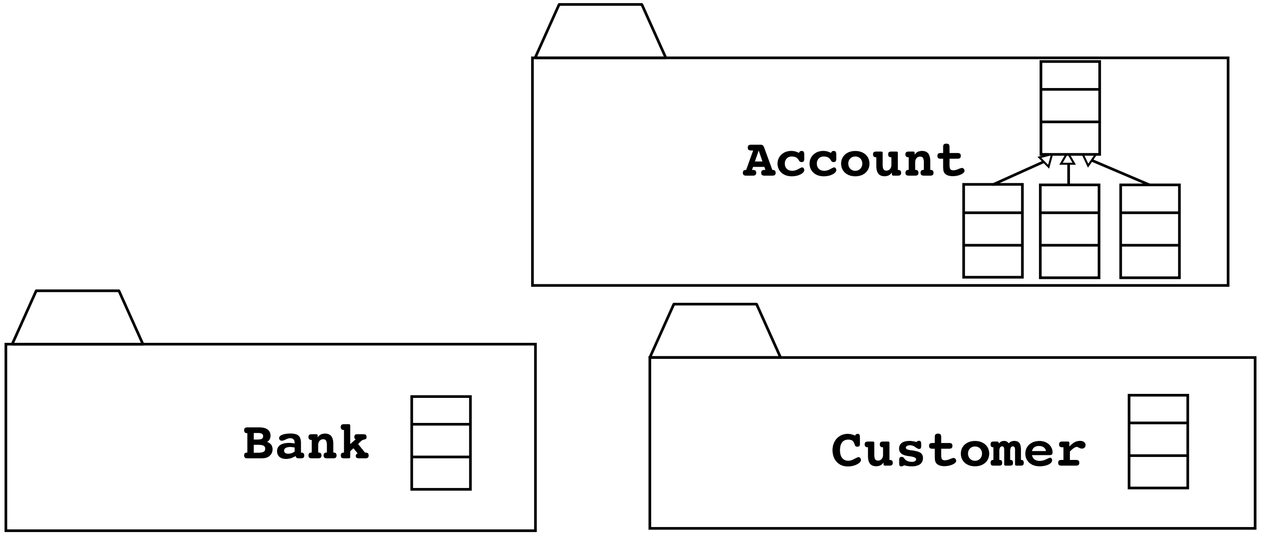
Inheritance



- *Inheritance* is another special case of an association denoting a "kind-of" hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The **children classes** inherit the attributes and operations of the **parent class.**

Packages

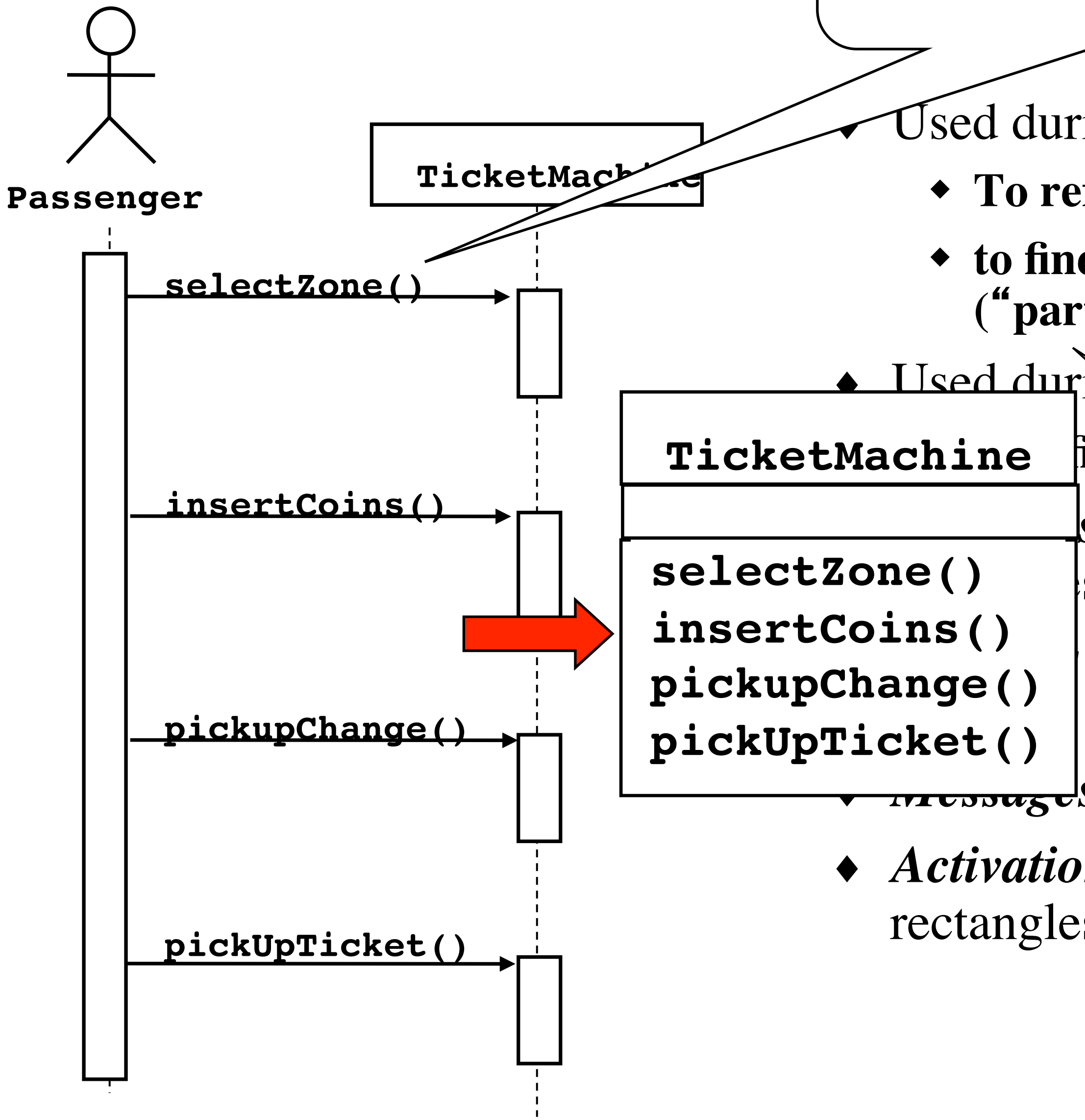
- Packages help you to organize UML models to increase their readability
- We can use the UML package mechanism to organize classes into subsystems



- Any complex system can be decomposed into subsystems, where each subsystem is modelled as a package.

Sequence Diagrams

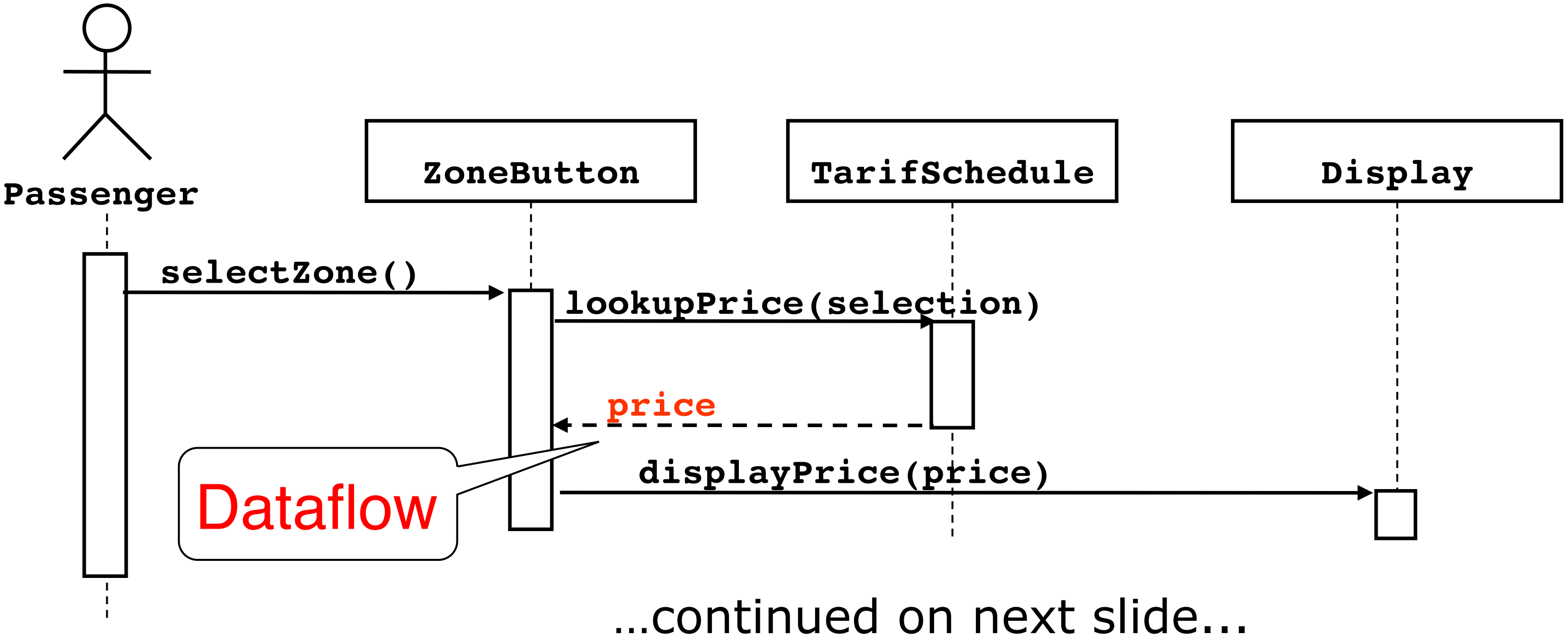
Focus on control flow



- Used during analysis
  - **To refine use case descriptions**
  - **to find additional objects (“participating objects”)**
- Used during system design
  - **fine su ystem interfaces**
- *s* are repr
  es. *Actors* b
- are repres
- *Messages* are represented by arrows
- *Activations* are represented by narrow rectangles.

Messages -> Operations on participating Object

Sequence Diagrams can also model the Flow of Data

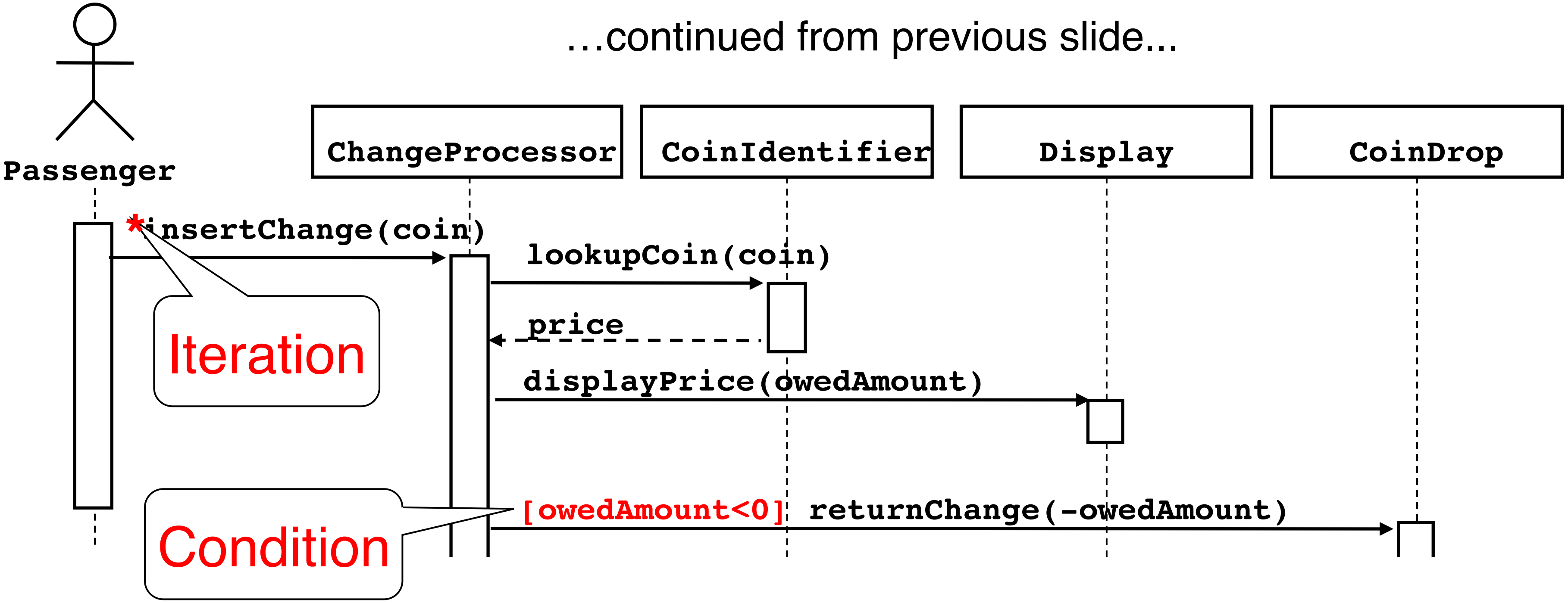


…continued on next slide...

- The source of an arrow indicates the activation which sent the message
- Horizontal dashed arrows indicate data flow, for example return results from a message

Sequence Diagrams: Iteration & Condition

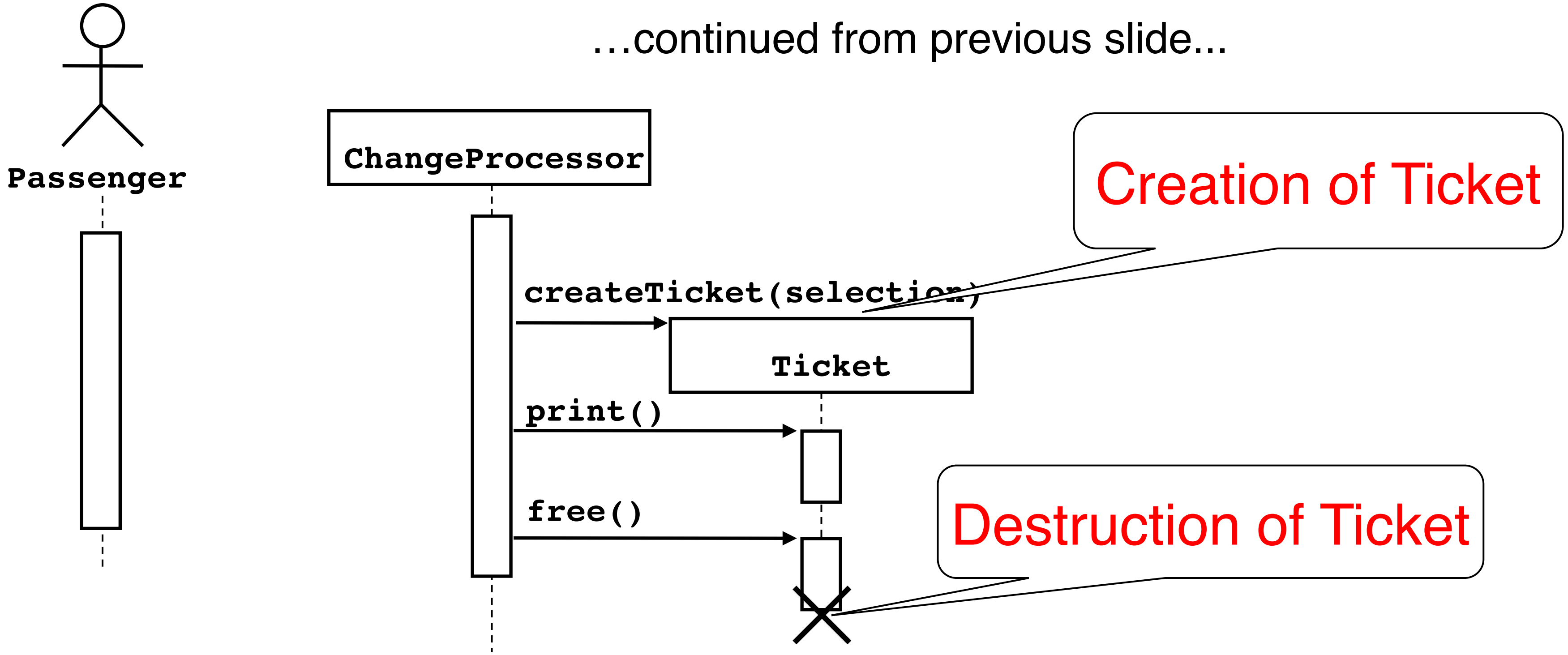
…continued from previous slide...



...continued on next slide...

- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [ ] before the message name

Creation and destruction



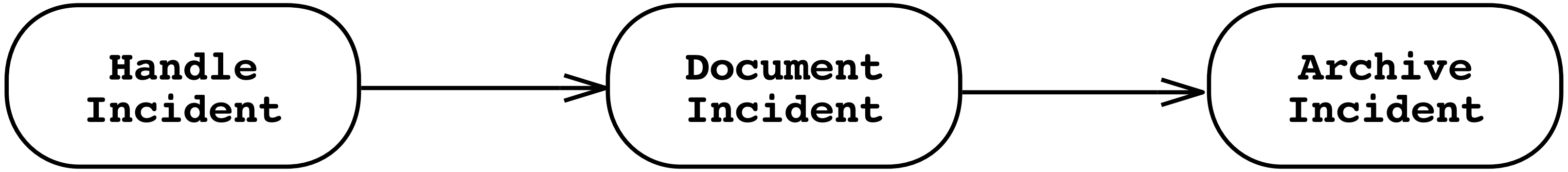
- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
  - **In garbage collection environments, destruction can be used to denote the end of the useful life of an object.**

Sequence Diagram Properties

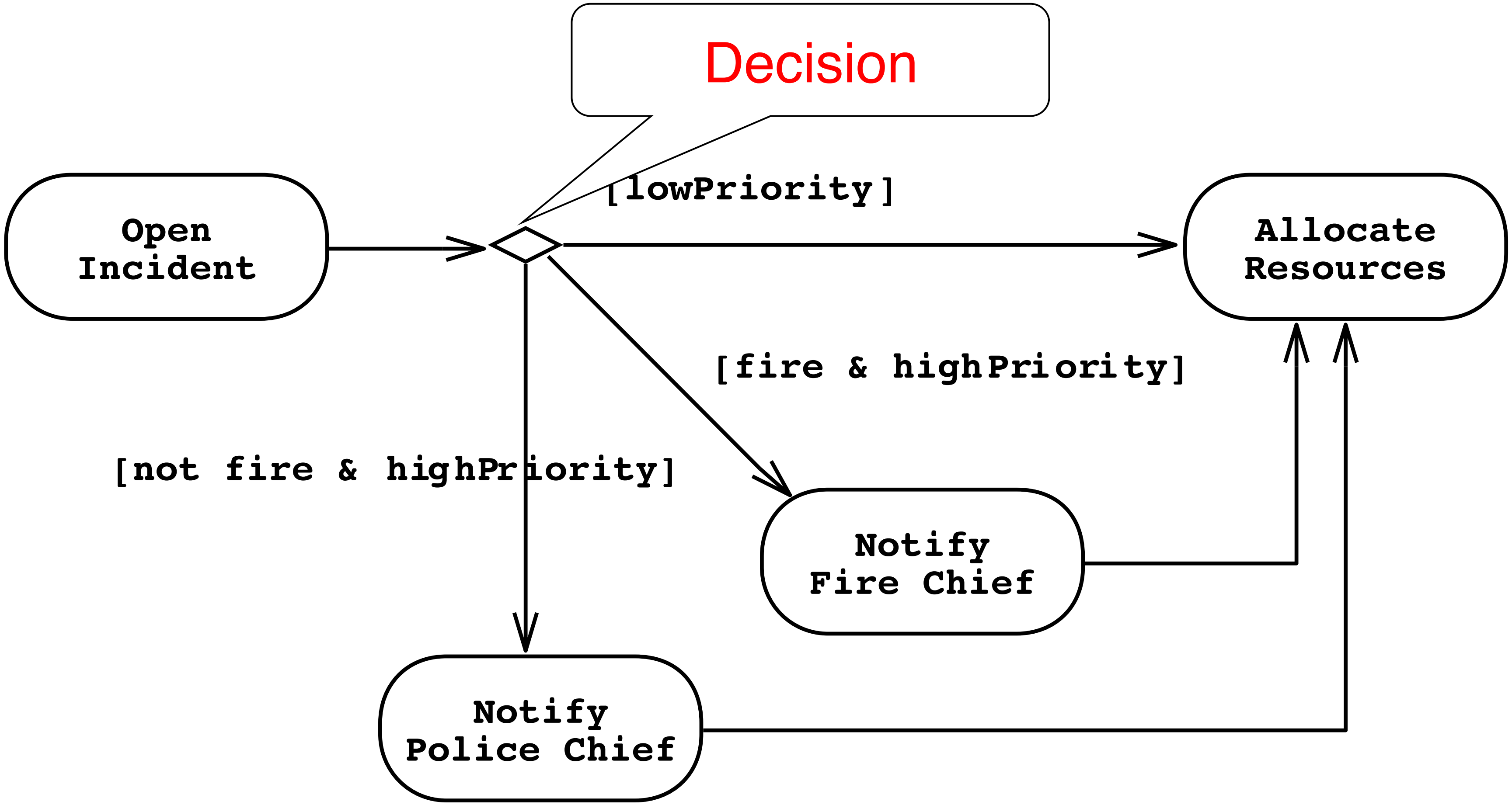
- UML sequence diagram represent *behaviour in terms of interactions*
- Useful to identify or find missing objects
- Time consuming to build, but worth the investment
- Complement the class diagrams (which represent structure).

Activity Diagrams

- An activity diagram is a special case of a state chart diagram
- The states are activities ("functions")
- An activity diagram is useful to depict the workflow in a system

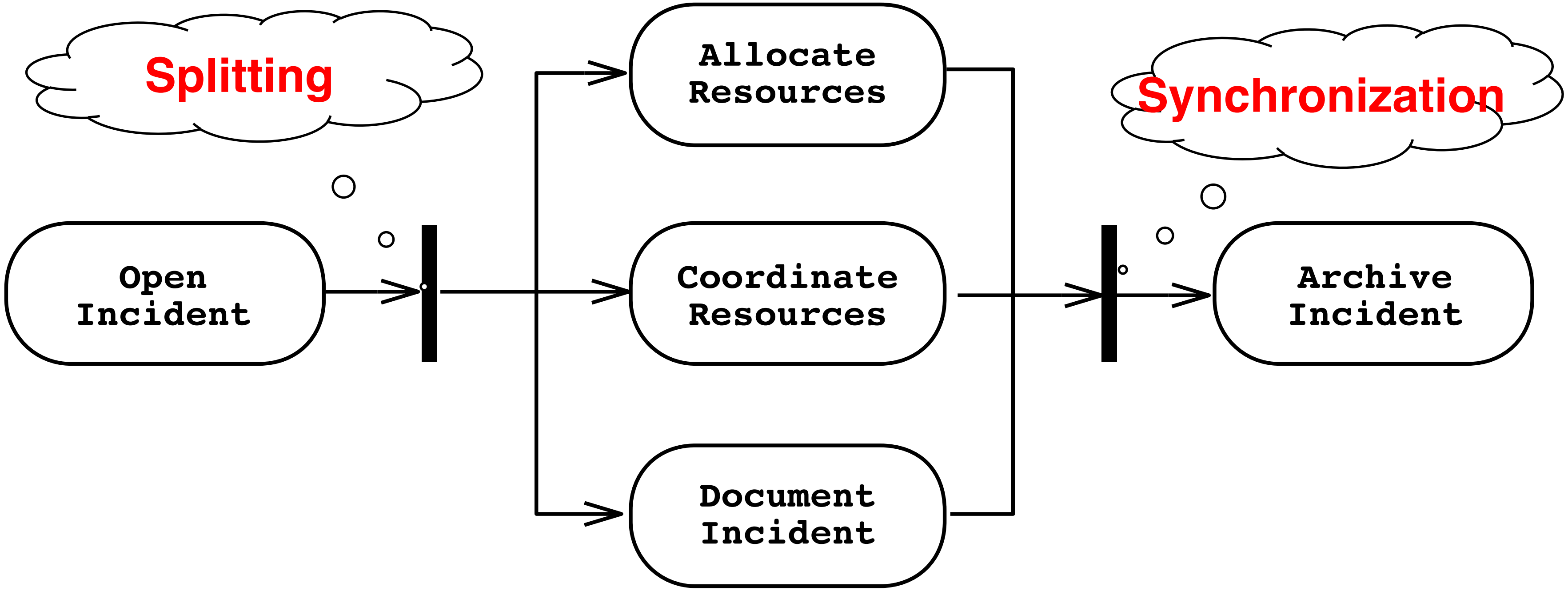


Activity Diagrams allow to model Decisions



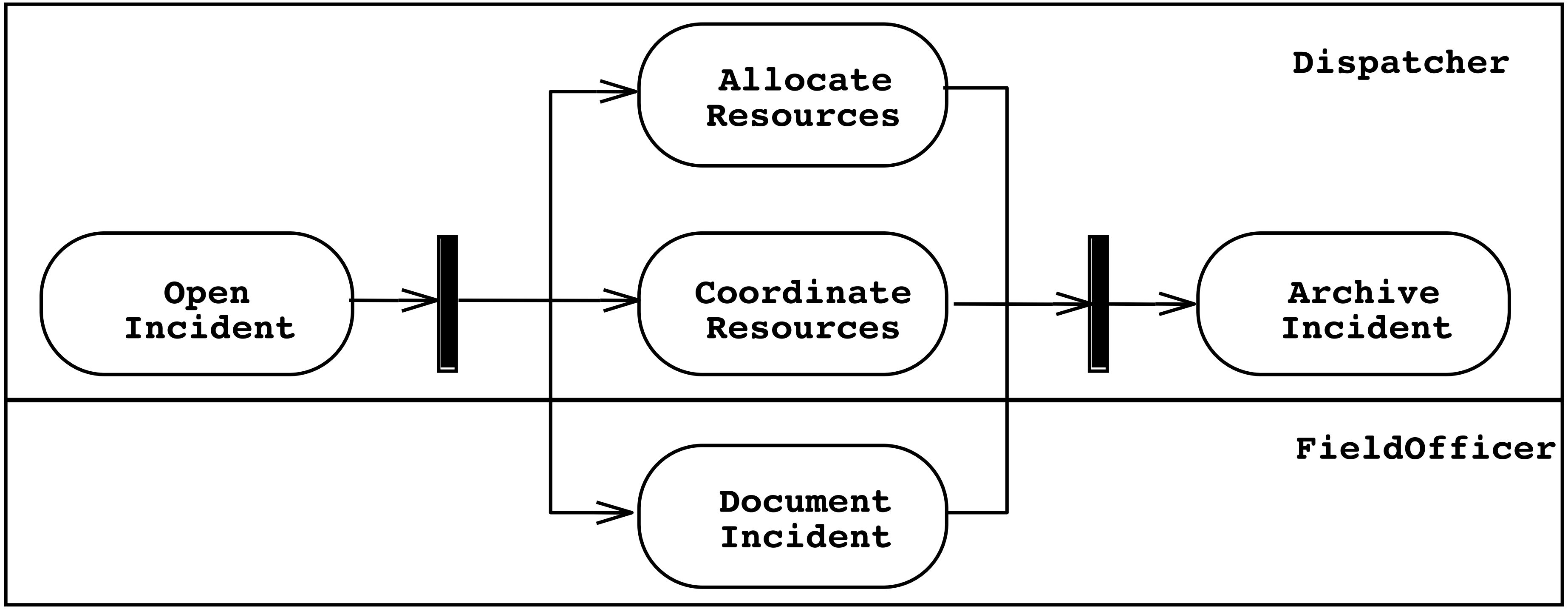
Activity Diagrams can model Concurrency

- Synchronization of multiple activities
- Splitting the flow of control into multiple threads



Activity Diagrams: Grouping of Activities

- Activities may be grouped into swimlanes to denote the object or subsystem that implements the activities.



UML Summary

- UML provides a wide variety of notations for representing many aspects of software development
  - **Powerful, but complex**
- UML is a programming language
  - **Can be misused to generate unreadable models**
  - **Can be misunderstood when using too many exotic features**
- We concentrated on a few notations:
  - **Functional model: Use case diagram**
  - **Object model: class diagram**
  - **Dynamic model: sequence diagrams, statechart and activity diagrams**

Additional References

- Martin Fowler
  - **UML Distilled: A Brief Guide to the Standard Object Modelling Language, 3rd ed., Addison-Wesley, 2003**
- Grady Booch, James Rumbaugh, Ivar Jacobson
  - **The Unified Modelling Language User Guide, Addison Wesley, 2nd edition, 2005**
- Commercial UML tools
  - **Rational Rose XDE for Java**
    - **http://www-306.ibm.com/software/awdtools/developer/java/**
  - **Together (Eclipse, MS Visual Studio, JBuilder)**
    - **http://www.borland.com/us/products/together/index.html**
- Open Source UML tools
  - **http://java-source.net/open-source/uml-modeling**
  - **ArgoUML,UMLet,Violet, …**