



# UML – part 3



# Classes and Objects

Olli Härmäläinen

# What is an object?

- An object is a representation of an entity, either real world or conceptual
- Objects can represent something concrete, such as
  - Joe's truck
  - my computer

or something more abstract e.g.

- a chemical process
- John's withdrawal at ATM this morning
- Mary's credit history

# What is an object?

- An object is a (presentation of a) concept, abstraction or thing with well-defined boundaries and meaning for an application.
- Each object in a system has three characteristics:
  - state
  - behaviour
  - identity

# State of an object

- One of the possible conditions in which an object may exist
- State typically changes over time
- State is defined by a set of properties (called attributes), with the values of the properties plus the relationships the object may have with other objects
- e.g. a course offered in the registration system may be in two state: open or closed.
  - If the number of the students registered for a course offering is less than 10, the state of the course is open.
  - When the tenth student registers for the course offering, the state becomes closed.

# Behaviour of an object

- Behaviour determines how an object responds to requests from other objects, and describes everything the object can do
- Behaviour is implemented by the set of operations for the object
- e.g. in the registration system, a course offered could have the behaviours *add a student* and *delete a student*
- N.B. the behaviours are usually identical for similar objects

# Identity of an object

- Each object is unique, even if its state and behaviour are identical to that of another object
- e.g. two courses, say Software Design for CAP05 and Software Design for CAP06, may have exactly the same characteristics i.e. the same state in the course registration system, but they still have (and must have) different identities
- in practice there is usually at least one attribute whose value is different for otherwise identical objects



# What is a class?

- Class is a description of a group of objects with common properties (attributes), common behaviour (operations), common relationships to other objects, and common semantics
- Class is like a mould to create objects
- Each object is an instance of exactly one class

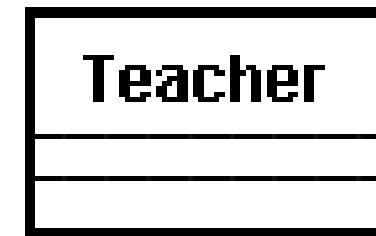
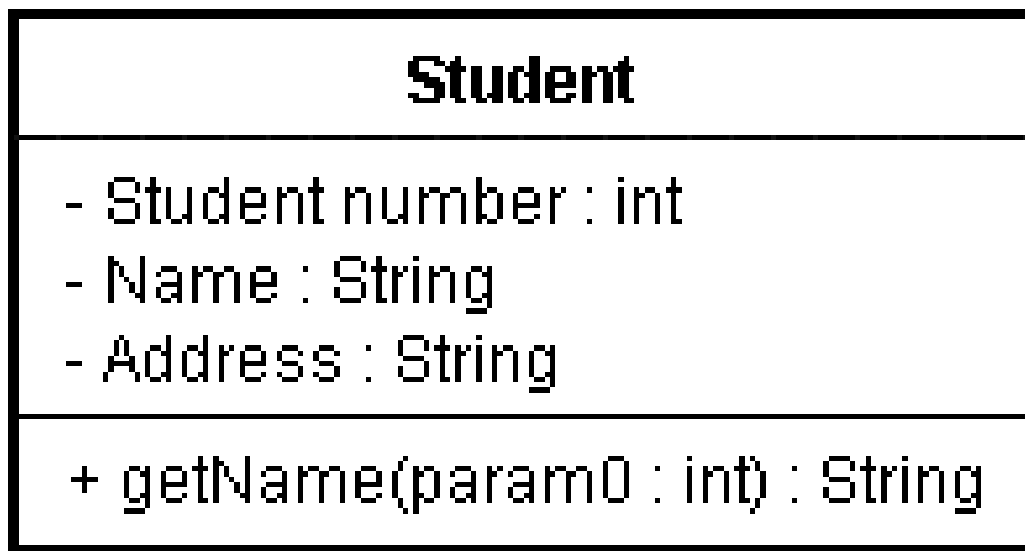


# What is a class?

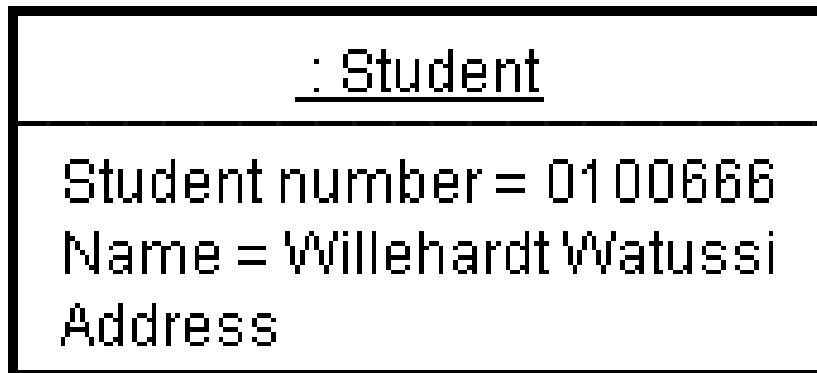
e.g. the CourseOffered class might have the following characteristics (i.a.):

- attributes
  - location
  - time table
  - number of credit points
- operations
  - retrieve location
  - retrieve time of the day
  - add a student for the course offered

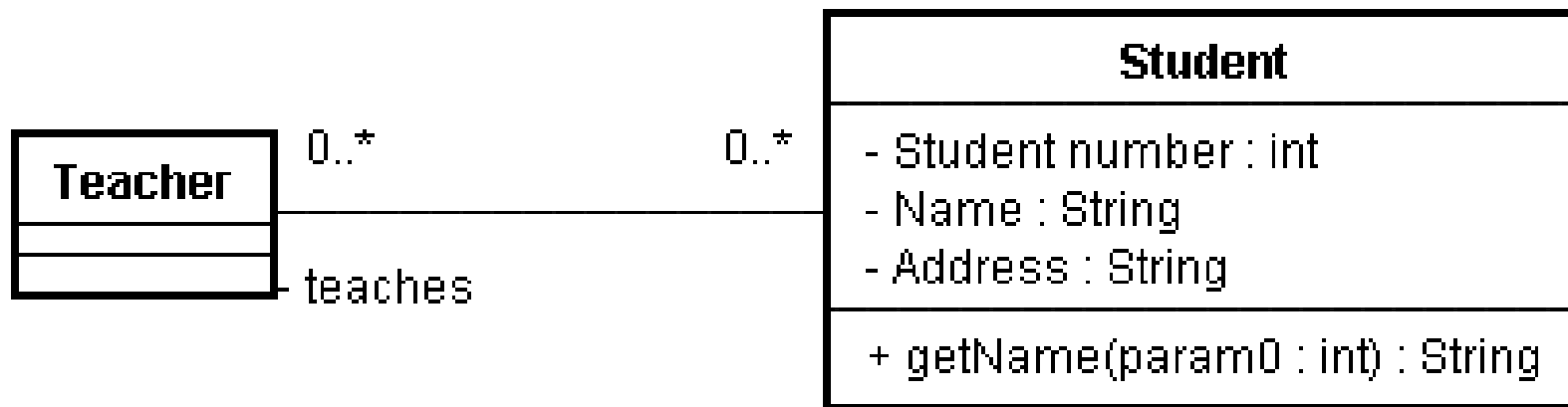
# UML notation for classes



# UML notation for objects



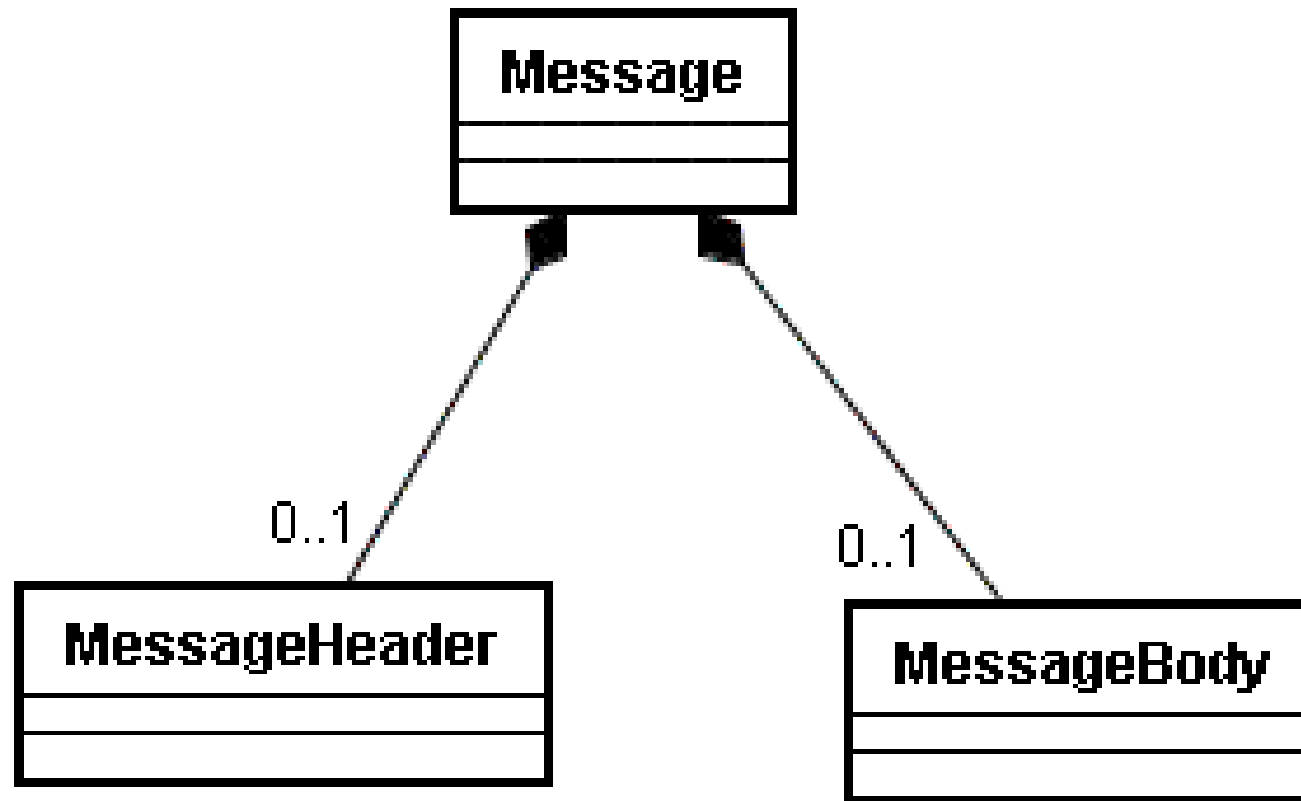
# Association between classes



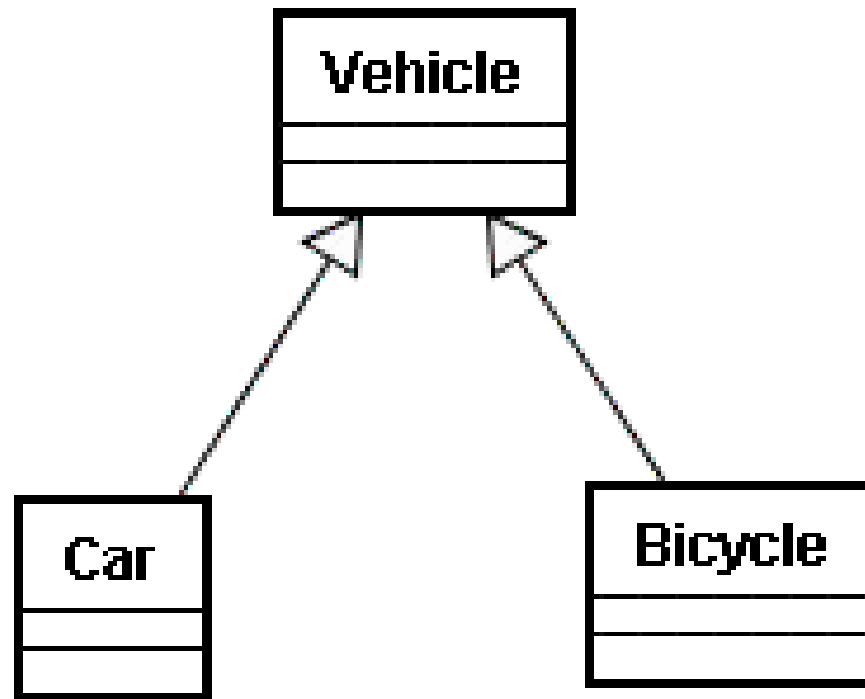
# Aggregation



# Composition



# Generalisation/inheritance





# What is an abstract class?

- Abstract classes cannot be instantiated (have no corresponding objects)
- Abstract classes may implement some methods, but typically also have some abstract methods



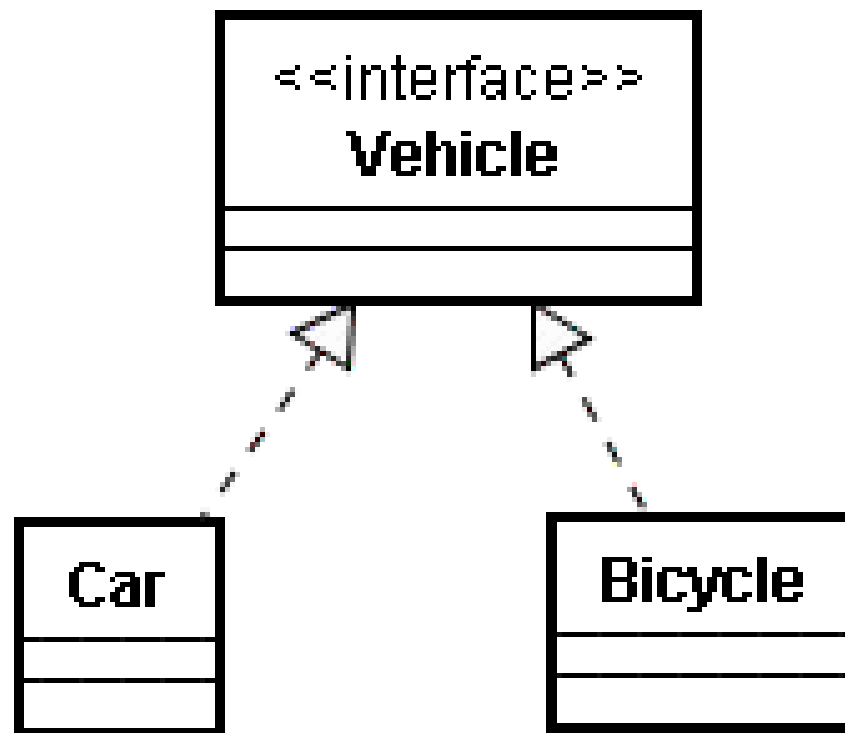
# What is an interface?

- An interface can be described is similar to an abstract class that has only abstract methods (methods that have no implementation)
- In the Java programming language, an interface is a reference type, similar to a class, that can contain only constants, method signatures, and nested types. There are no method bodies.
- A class can implement one or several interfaces => kind of multiple inheritance also in Java

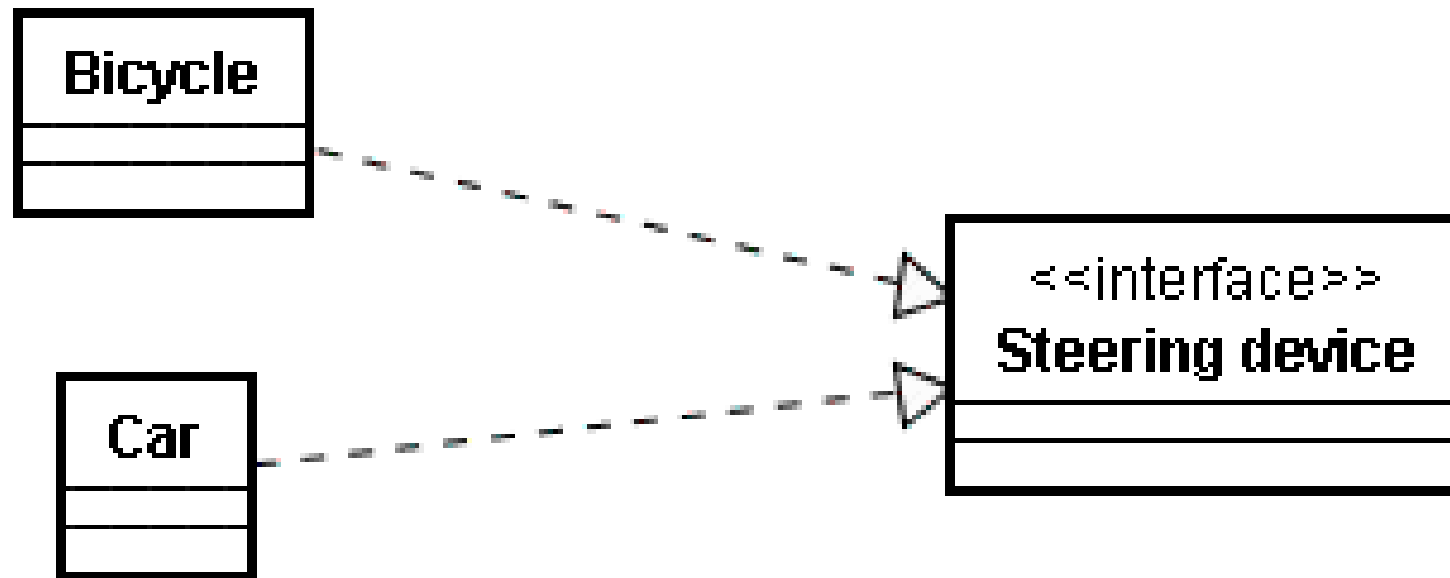
# What is an interface?


- Interfaces cannot be instantiated (i.e. they have no corresponding objects)
- Classes that implement an interface must replace the abstract (empty) methods with real ones
- A subinterface can extend one or several interfaces thus combining features from several sources

# Interface and Implementation



# Interface and Implementation





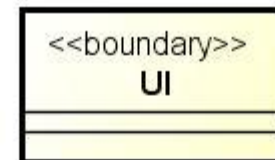
# Discovering classes and associations

Olli Härmäläinen

# Discovering classes

- entity classes
  - may reflect a real-world entity
  - to discover candidate classes check the nouns used to describe the responsibilities documented in the flow of events for the identified use cases
- boundary classes
  - provide the interface to an actor
- control classes
  - model sequencing behaviour specific to one or more use cases

# Alternative UML symbols for classes





# Boundary classes

- boundary classes handle the communication between the system and the inside of the system
- they provide the interface to a user or another system.
- boundary classes constitute the surroundings-dependent part of the system.
- boundary classes are used to model the system interfaces e.g user interface(s) and reports





## Control classes

- control classes model sequencing behaviour specific to one or more use cases
- control classes co-ordinate the events needed to realize the behaviour in the use case.
- control class can be thought as the class “running” or “executing” the use case.
- control classes typically are application dependent classes



# Entity classes

- an entity class models information and associated behaviour that is generally long lived. This type of class may reflect a real-world entity, or it may be needed to perform tasks internal to the system.
- entity classes are typically independent of their surroundings; they are not sensitive to how the surroundings communicate with the system
- they are often application independent and can be used in more than one application
- to find the entity classes, the first step is to examine the responsibilities (i.e. what the system must do) documented in the flow of events for the identified use cases



## Finding entity classes, example

- Nouns are marked with blue color
- Nouns are all candidates for entity classes
- Part of the candidates are rejected
- Several nouns might represent the same entity class

## Example - Access Control

An access control system is based on personal key cards. The card must always be presented for control every time a person wishes to pass a gate. Every control occasion is recorded by the system. The area which is controlled by the gates is divided into zones. Access rights are given based on these zones, staff category, weekday, and time.

# Discarding class candidates

- The initial list of nouns i.e. class candidates must be trimmed since it usually contains items that are outside the problem domain
- you should omit
  - unessential classes
  - repetitions
  - attributes
  - classes with something to do with control
  - role describing classes

# Discovering Associations

- associations are often found through verbs or adverbs using all the documents in the requirements analysis
- first a candidate list of all expressions that can indicate an association is composed
- next the association connected with already omitted classes are discarded
- associations with transient action and trivialities are omitted
- after trimming the associations are named with describing names and given the multiplicities

## Example - Access Control

- An access control system is based on personal key cards. The card must always be presented for control every time a person wishes to pass a gate. Every control occasion is recorded by the system. The area which is controlled by the gates is divided into zones. Access rights are given based on these zones, staff category, weekday, and time.



# Previous diagrams revisited

Olli Härmäläinen

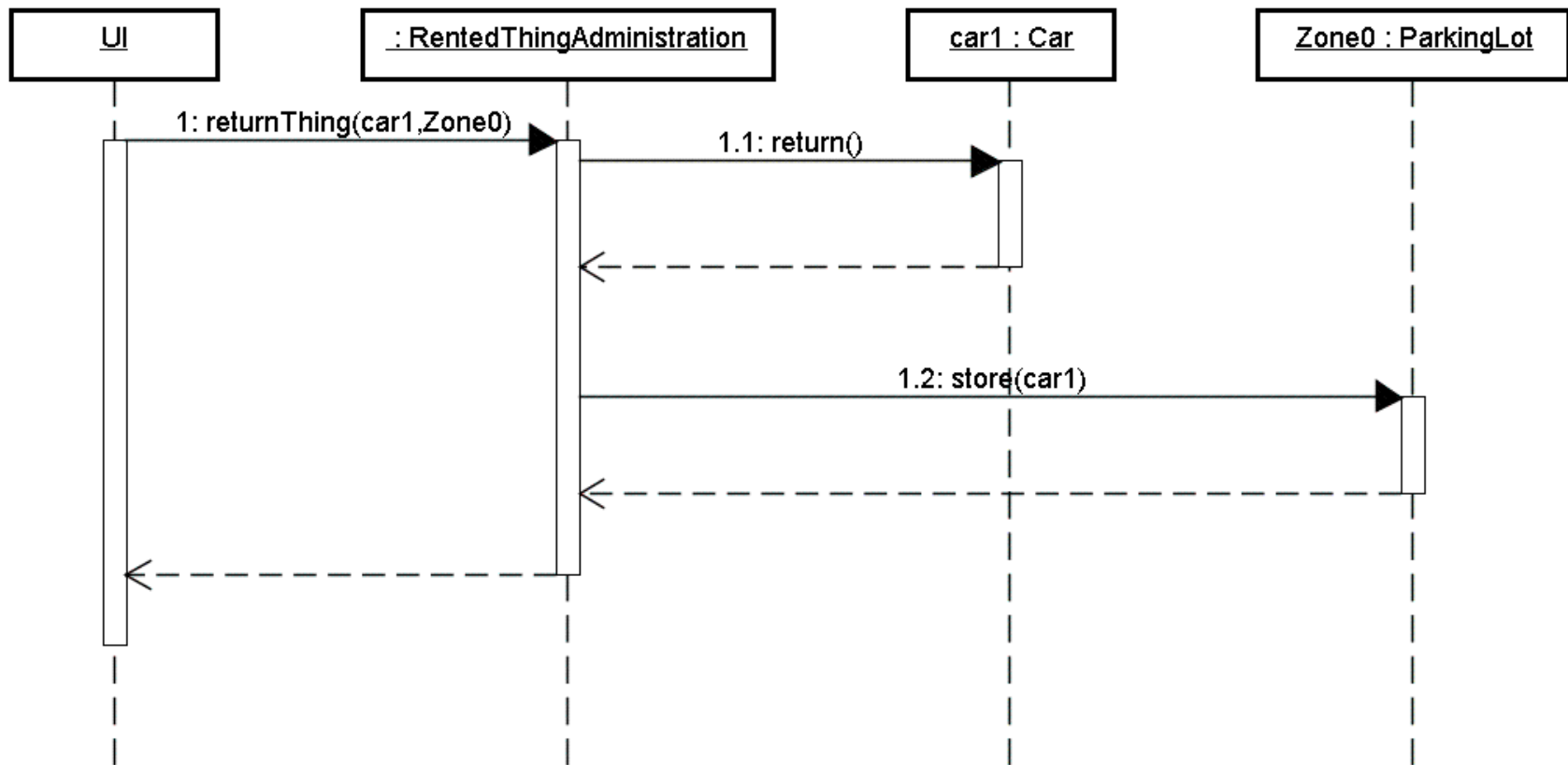




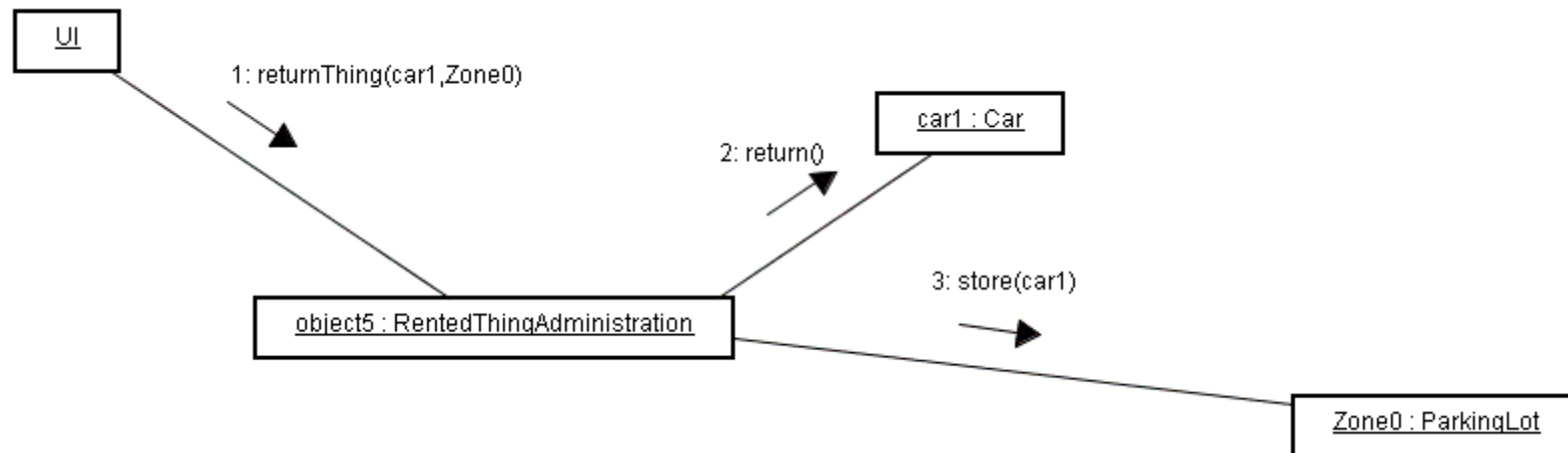
# Interaction Diagrams

- sequence diagrams emphasise the time ordering of messages
- collaboration diagrams emphasize the structural organisation of the objects that send and receive messages
- sequence diagrams and collaboration diagrams contain essentially the same information, but they are typically used in different phases of the development cycle

# Sequence Diagram



# Collaboration Diagram





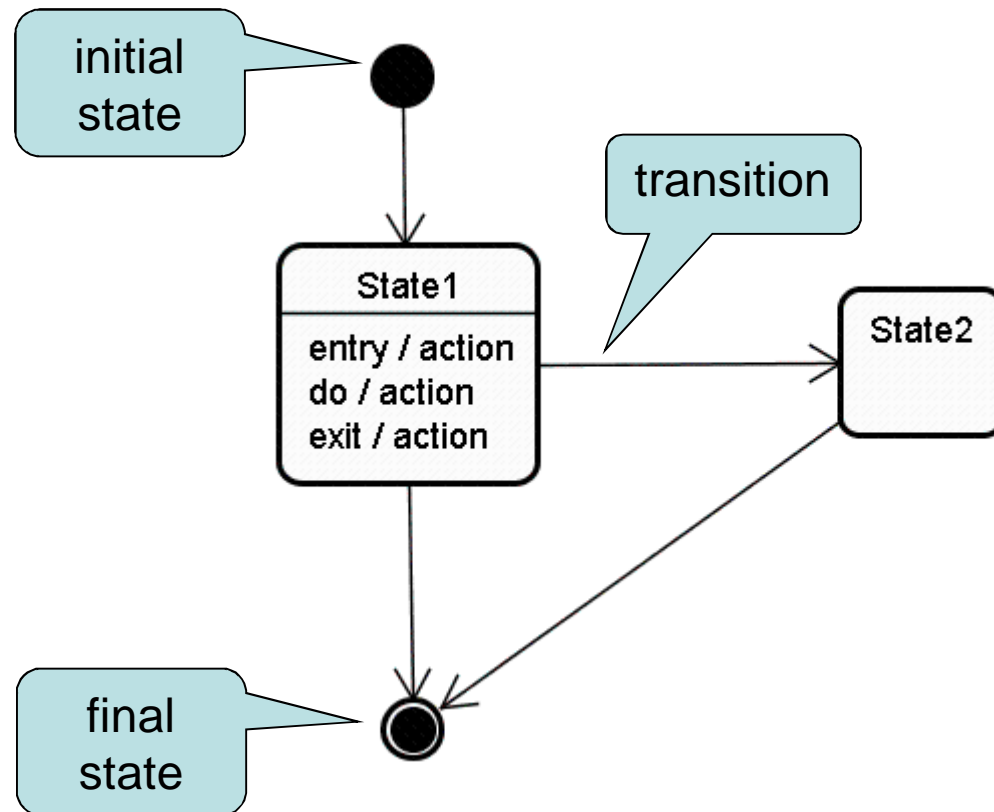
# State Diagrams

Olli Härmäläinen

# State Diagrams

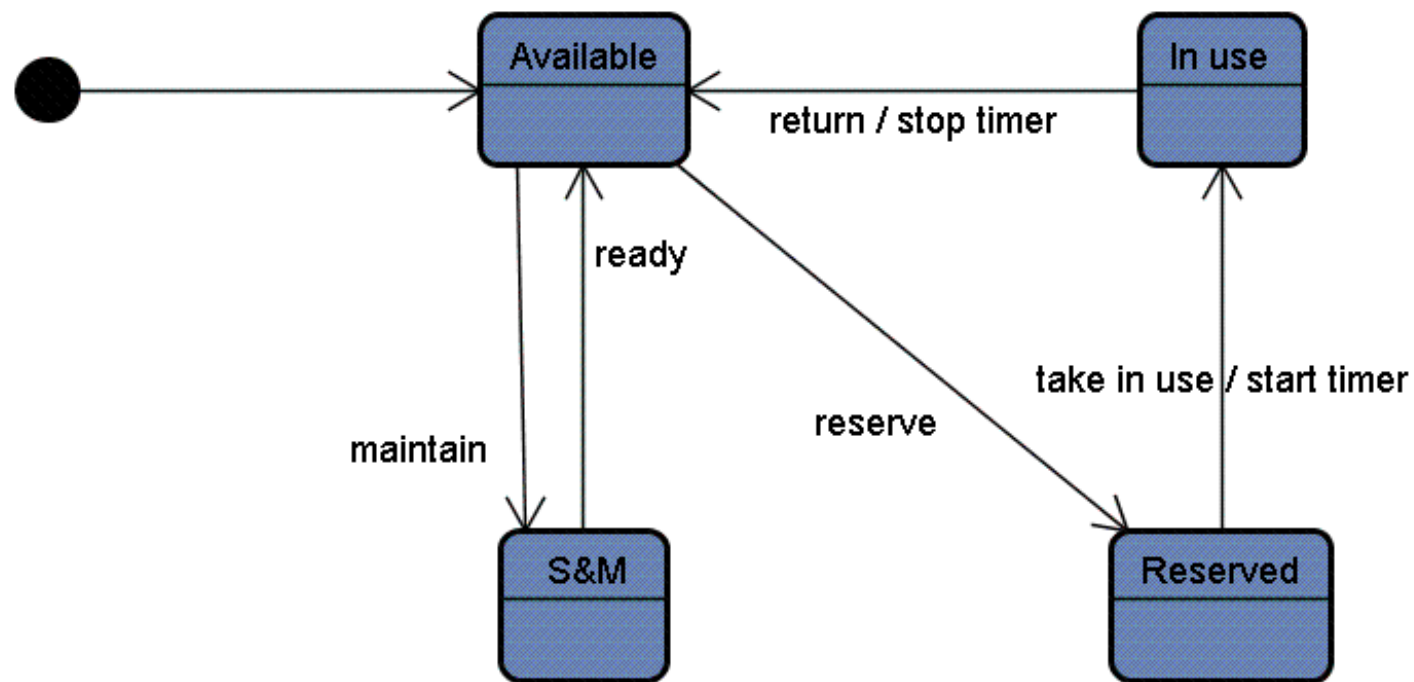
- State diagram (also known as statechart diagram) typically describes the dynamic behaviour of objects of a given class and how they react to external events
- State diagrams are reasonable only for classes whose realisations (objects) have clearly identifiable states during their life time and whose behaviour can be characterised by the manner in which they change their state while interacting with their environment
- Typically used to specify the run-time behaviour of active objects (i.e. the objects has its own process) or object that steer the activity of others

# State Diagram Symbols



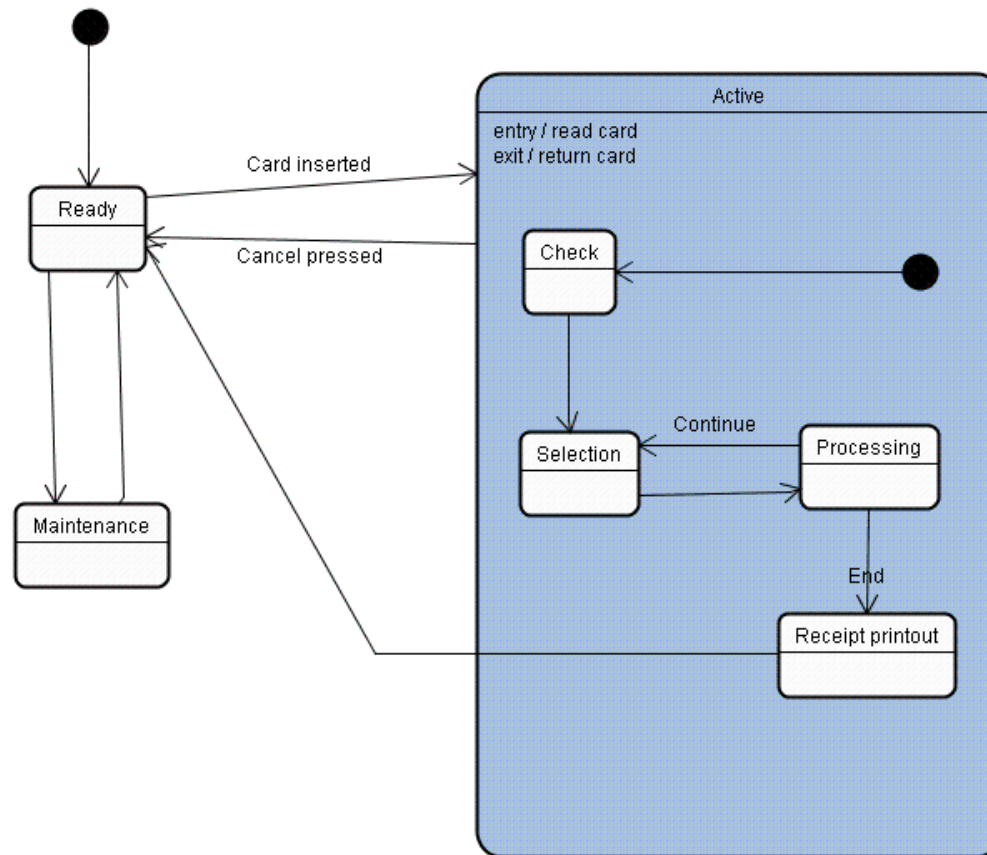
# State Diagram example

states of a car in the rental system



## State Diagram example 2

Super state in ATM – 'Cancel' may be pressed in any phase



© 2011 Pearson Education, Inc.





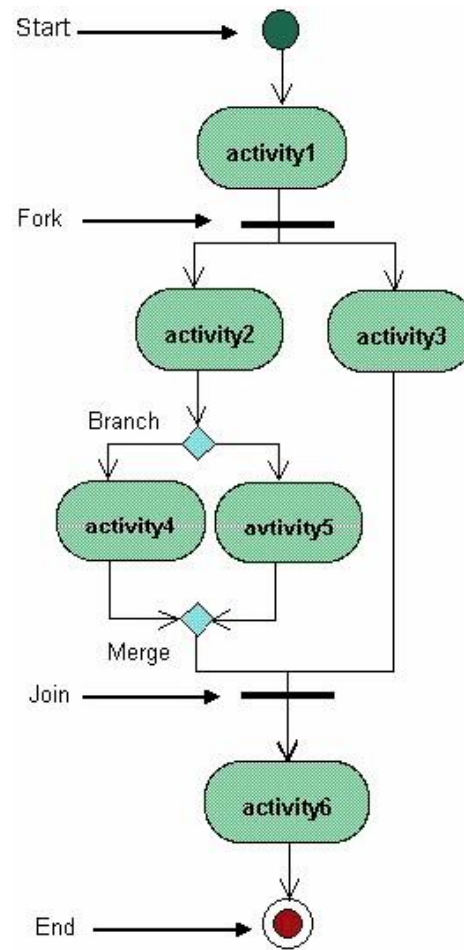
# Activity Diagrams, once again

Olli Härmäläinen

# Activity diagrams

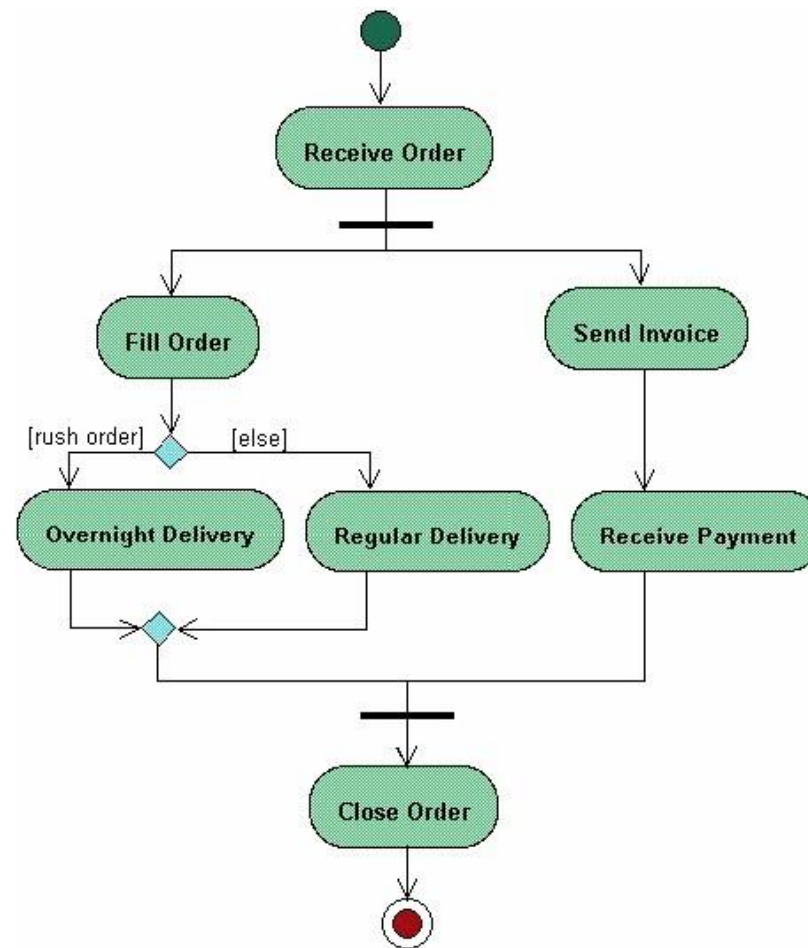
- an activity diagram describes the internal logic of a certain task
- activity diagram is formally just a special case of a statechart diagram, but it is typically used differently
- decision symbols and data items can be used in activity diagrams, which allows a usage similar to traditional flow charts and data flow diagrams
- parallel processing can also be described with special symbols (fork and join)
- swimlanes may be used to describe the locus, role, or processing unit of an activity

# Activity Diagram Tools



Olli Härmäläinen

# Activity Diagram, example 1



Olli Härmäläinen

# Activity Diagram, example 2

Car rental – general view with "swimlanes"

