5COSC001W - OBJECT ORIENTED PROGRAMMING

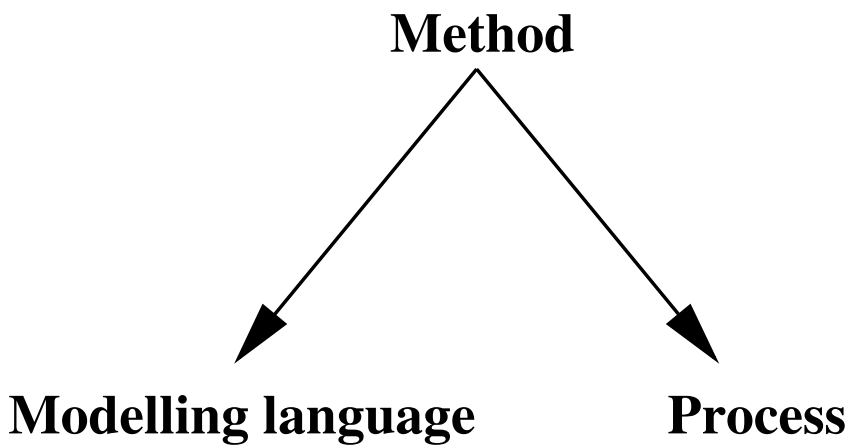
Lecture 8: All you need to know about UML

Dr Dimitris C. Dracopoulos

What is UML?

A collection of techniques (diagrams) used for object oriented software design.

- It unifies the methods of Booch, Rumbaugh and Jacobson, known as the *three amigos* in the software engineering community.
- It is a modelling language not a method.



UML techniques

The most well known techniques currently used in UML are (but UML still evolves):

- Use case diagrams
- Class diagrams
- Object diagrams
- Interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams
- Statechart diagrams
- Activity diagrams
- Physical diagrams

Use Case Diagrams

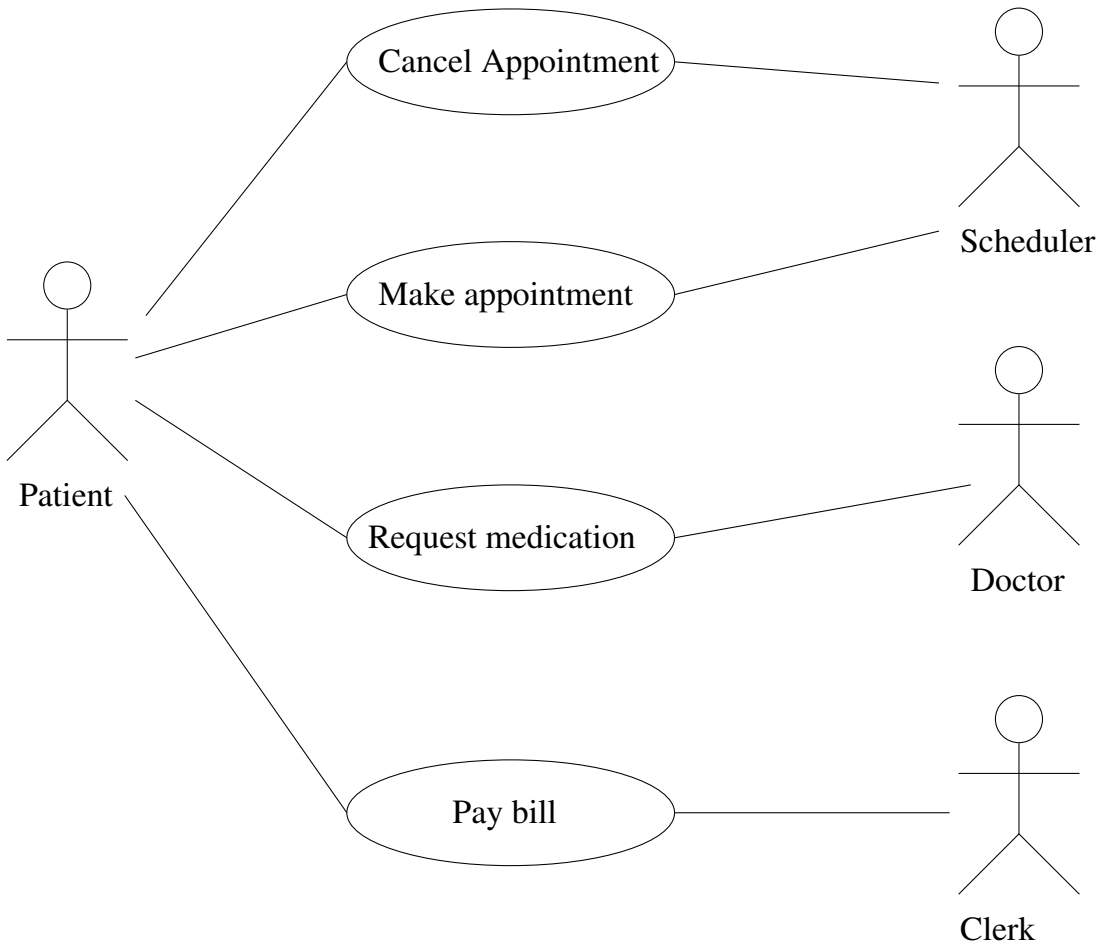
- Describe *what* we are building (helps to build the right system).
- It is a snapshot of one aspect of the system. The sum of all use cases is the external picture of the system.

A use case diagram consists of:

- *Actors:* an actor is a role that a user plays in respect to the system.
- *Use cases:* a use case is a scenario for a single task or goal.
- *Communications:* the associations between the actors and the use cases.

An example of a use case diagram

A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot. The patient visits the doctor and then he pays the bill.



How to draw a use case diagram

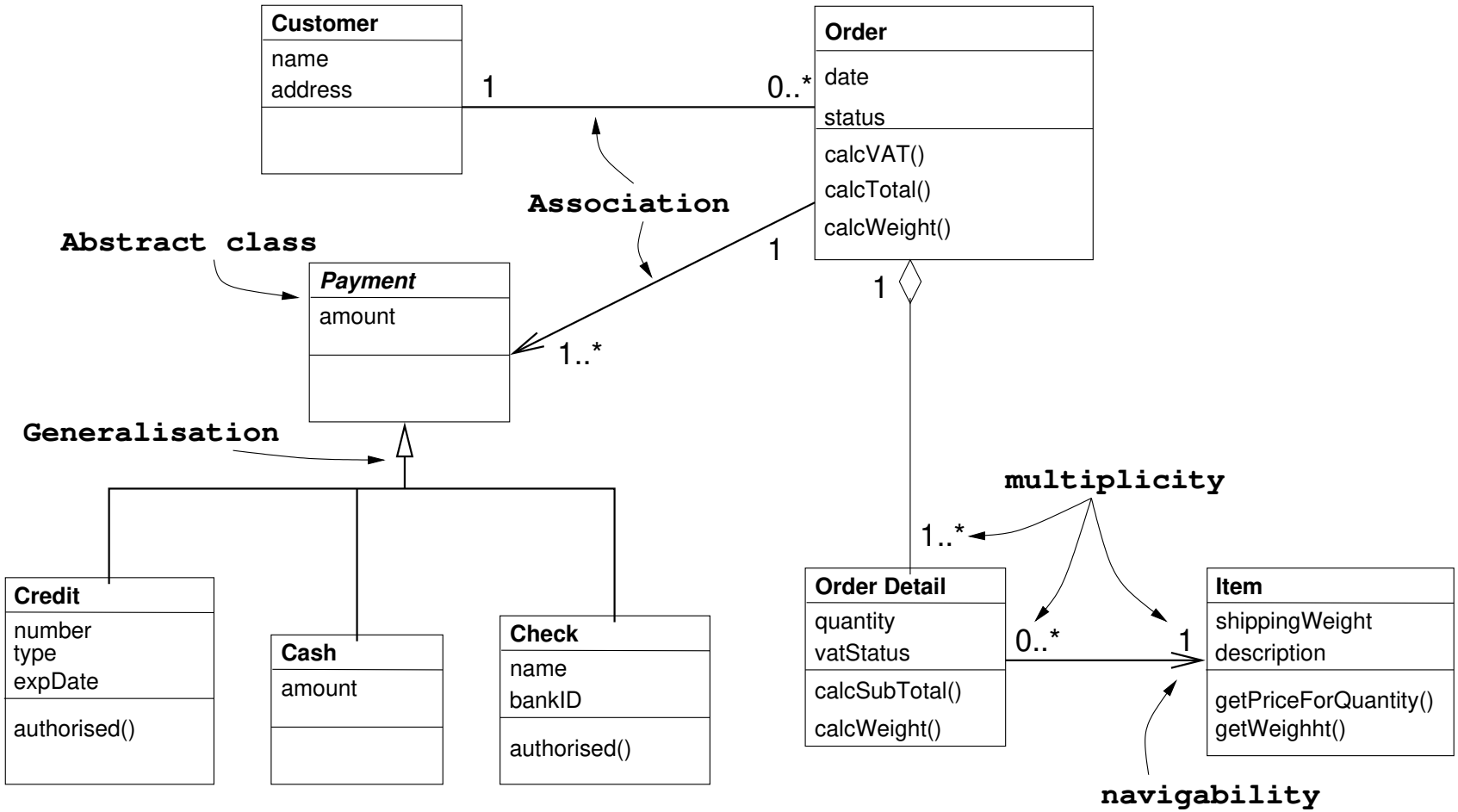
Use cases identify key features in the system that will reveal some of the fundamental classes which will be used in the implementation.
Ask the questions:

- Who will use this system?
- What can those actors do with the system?
- How does *this* actor do *that* with this system?
- How else might this work if someone else were doing this, or if the same actor had a different objective? (to reveal variations)
- What problems might happen while doing this with the system? (to reveal exceptions)

Class Diagrams

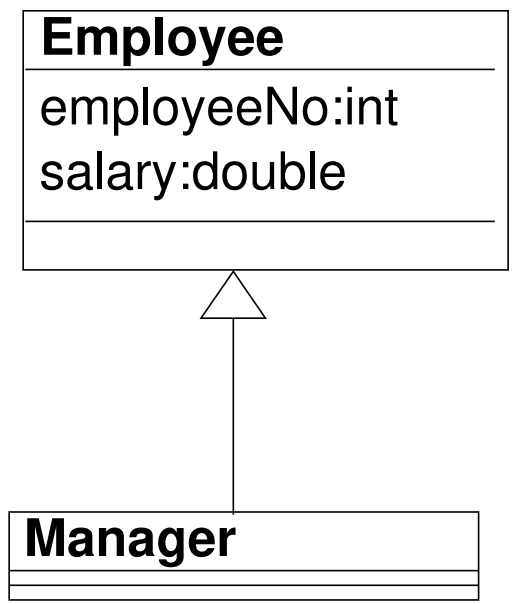
A class diagram describes the types of objects in the system and the static relationships that exist among them.
There are two main kinds of static relationships:

- *Associations:* For example, a customer may rent a number of videos.
- *Generalisation:* For example a manager is a kind of employee.


Customer
name
address
1
0..*
Order
date
status
calcVAT()
calcTotal()
calcWeight()
Association
Abstract class
Payment
amount
1
1..*
1
Generalisation
multiplicity
1..*
Credit
number
type
expDate
authorised()
Cash
amount
Check
name
bankID
authorised()
Order Detail
quantity
vatStatus
calcSubTotal()
calcWeight()
0..*
1
Item
shippingWeight
description
getPriceForQuantity()
getWeighht()
navigability

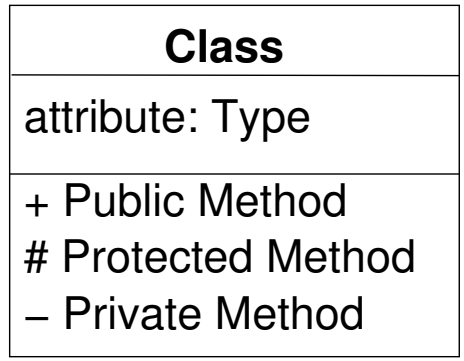
Generalisation and Inheritance

```
class Employee {
   private int employeeNo;
   private double salary;
}
class Manager extends Employee {

}
```



Manager *is an* Employee.

The Class in a class diagram

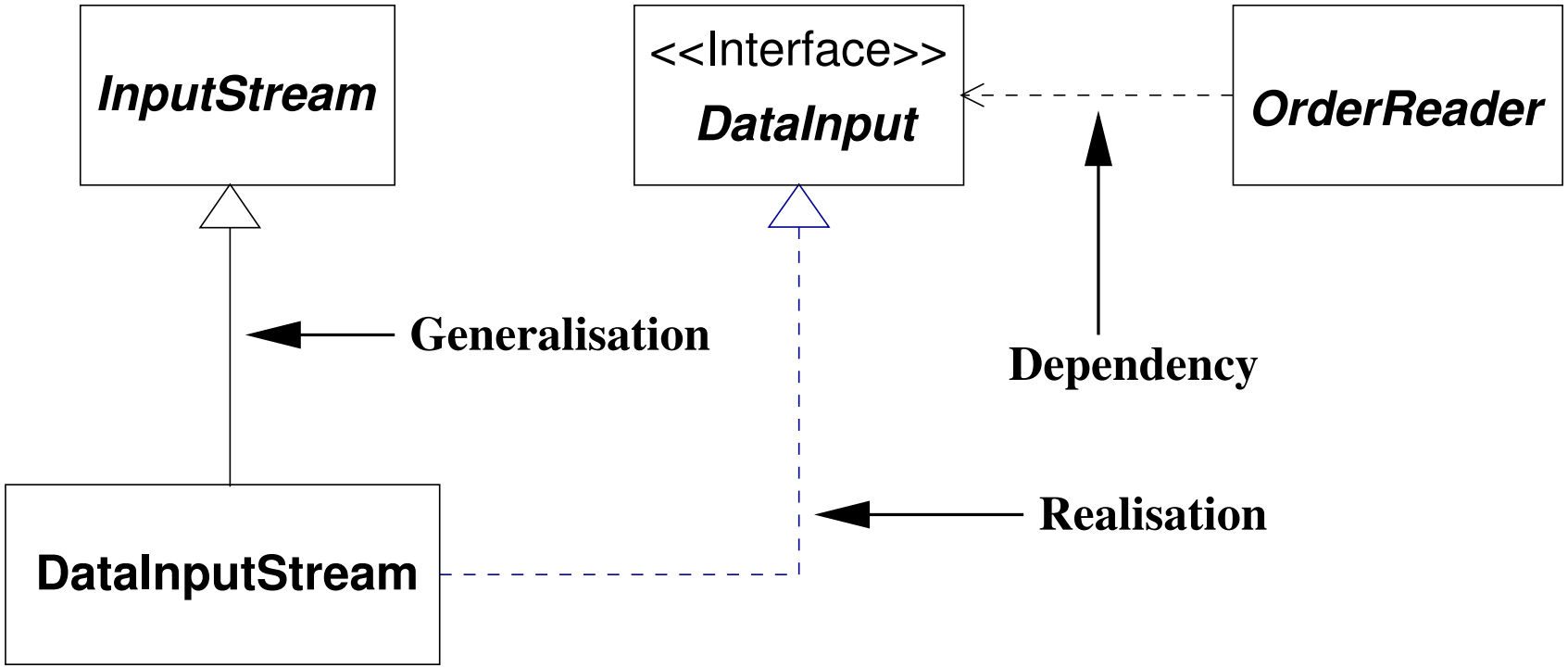


The full UML syntax for operations is:
visibility name(parameter-list): return-type{property-string}
where

- *visibility* is: + (public), # (protected), or – (private).
- *name* is a string.
- *parameter-list* contains comma-separated parameters similar to function parameters.
- *return-type* is the type(s) of the return value(s). Notice that more than one return value is allowed in UML.
- *property-string* indicates property values that apply to the given operation.

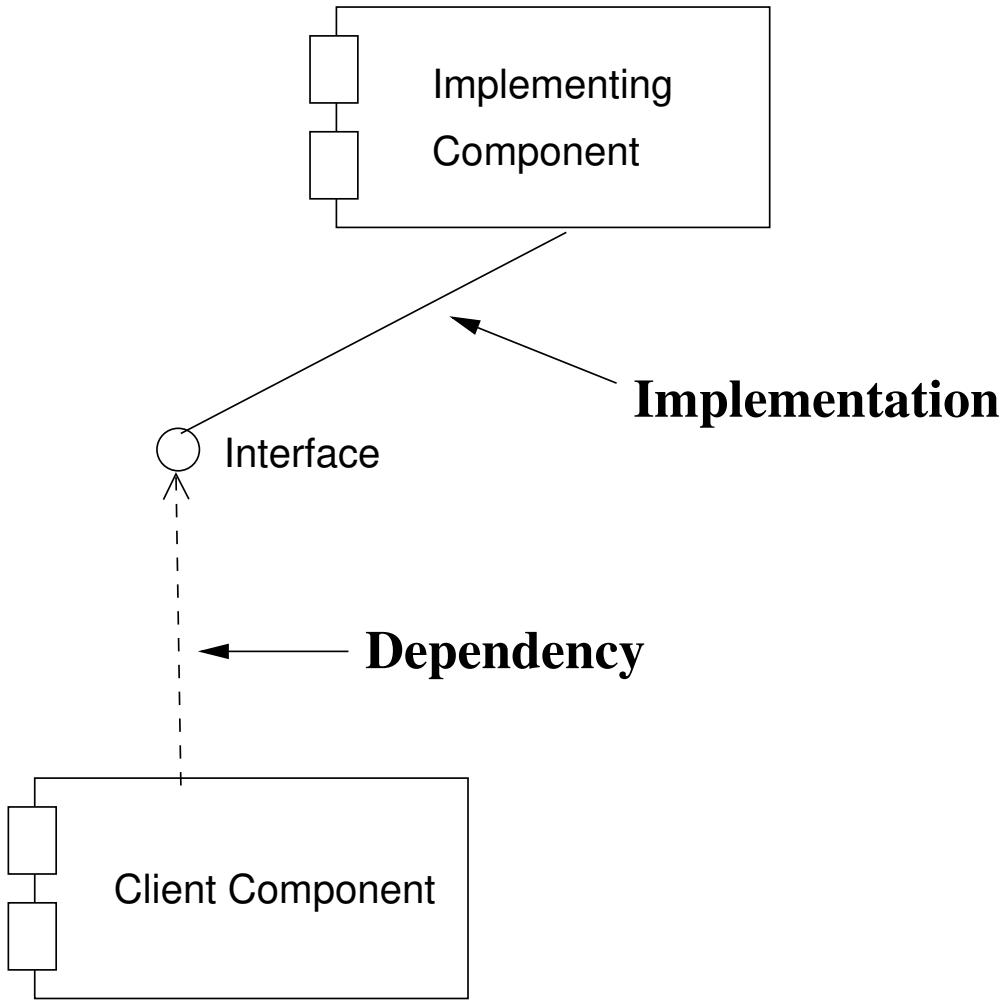
For example: *balanceOn(date: Date): Money*

Interfaces and Abstract Classes



- Realisation is similar to generalisation. It indicates that a class implements an interface.
- A dependency indicates that if a class (interface) changes then the dependent class (component) has to change as well. Dependencies are not necessary associated with interfaces, i.e. a class can depend on part of the implementation of another class.

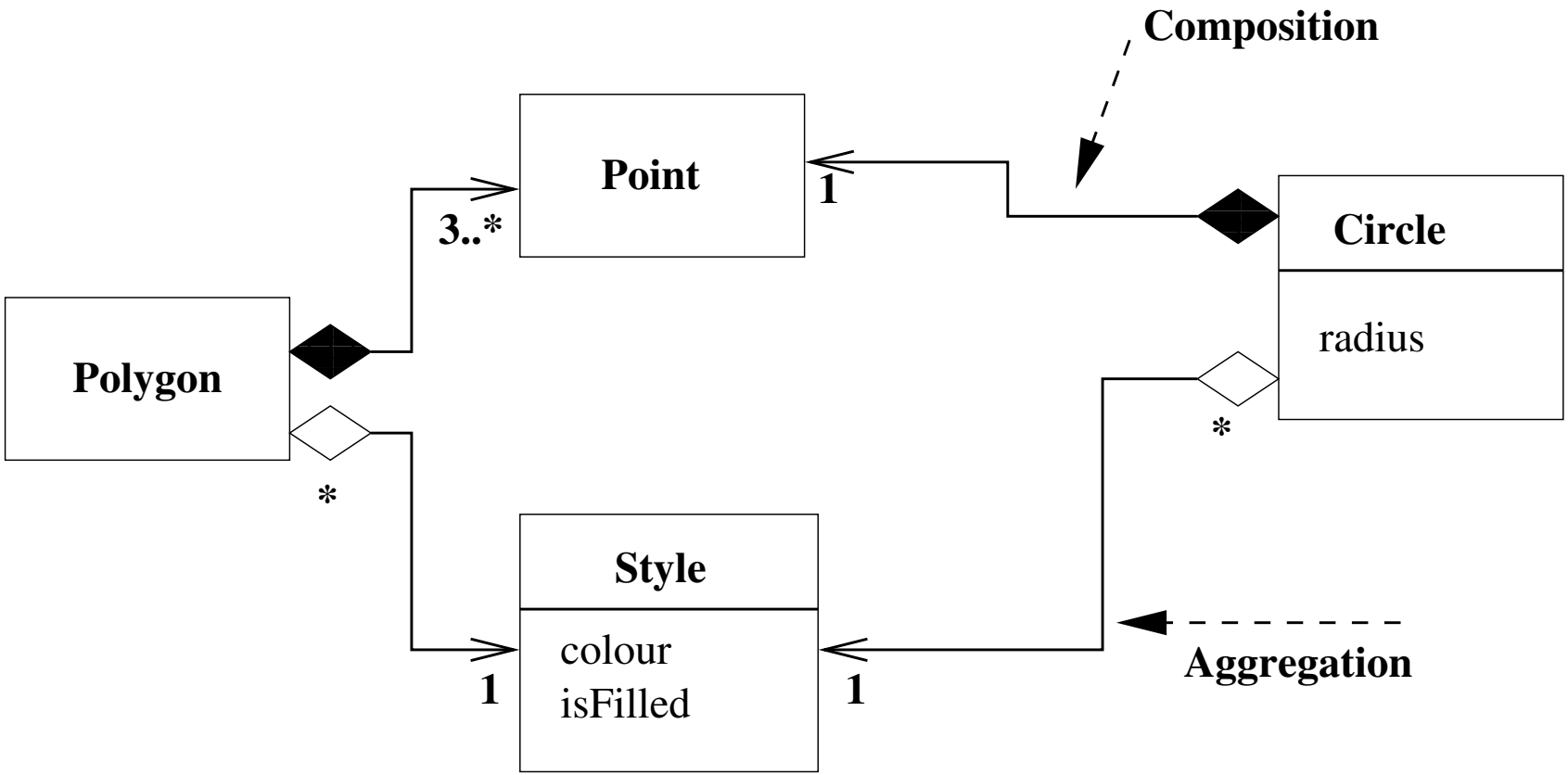
An alternative notation for realisation and dependency on interfaces



Abstract classes vs Interfaces

```
abstract class Shape {
    public abstract void draw();

    void print_hello() {
        System.out.println("Hello!");
    }
}

interface Geometry {
    double calcArea();
}

class Circle extends Shape implements Geometry {
    private double radius;

    public void draw() {
        System.out.println("Drawing circle");
    }
    public double calcArea() {
        return(...);
    }
}
```

Aggregation and Composition



Aggregation is the part-of relationship. An object is part of another object. Composition is a stronger variety of aggregation.

Composition in Java

```
class Engine {
    // ... details omitted
}


class Car {
    Engine eng;

    Car() {
        eng = new Engine();
    }

    public static void main(String[] args) {
        Car c = new Car();
    }
}
```

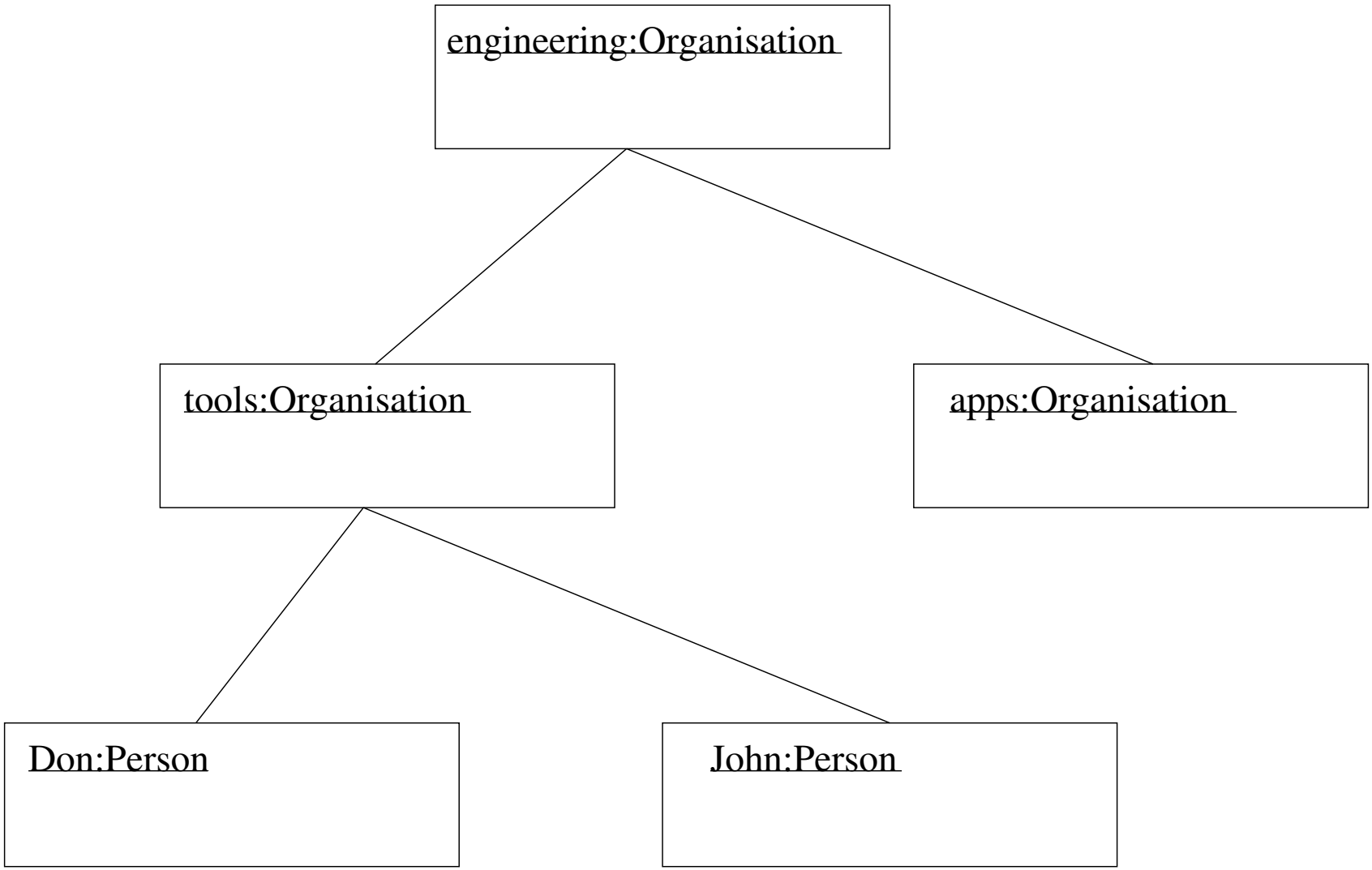
Aggregation in Java

```
class Color {
    // ....
}
class Fill {
    // ....
}
class Style {
    private Color color;
    private Fill isFilled;
}
class Circle {
    private Style style;
    Circle(Style s) {
        style = s;
    }
    // ... the rest of details omitted
}
class Polygon {
    private Style style;
    Polygon(Style s) {
        style = s;
    }
    // ... the rest of details omitted
}
```

Aggregation in Java (cont'ed)

```
class Paint {
    public static void main(String[] args) {
        Style style = new Style();
        Circle c = new Circle(style);
        Polygon pol = new Polygon(style);

        c = null;
        // object style still exists even if
        // garbage collector recycles!
    }
}
```

Object Diagrams



Statechart Diagrams

Describe the state of a system by showing all the states of objects and how an object's state changes as a result of events.

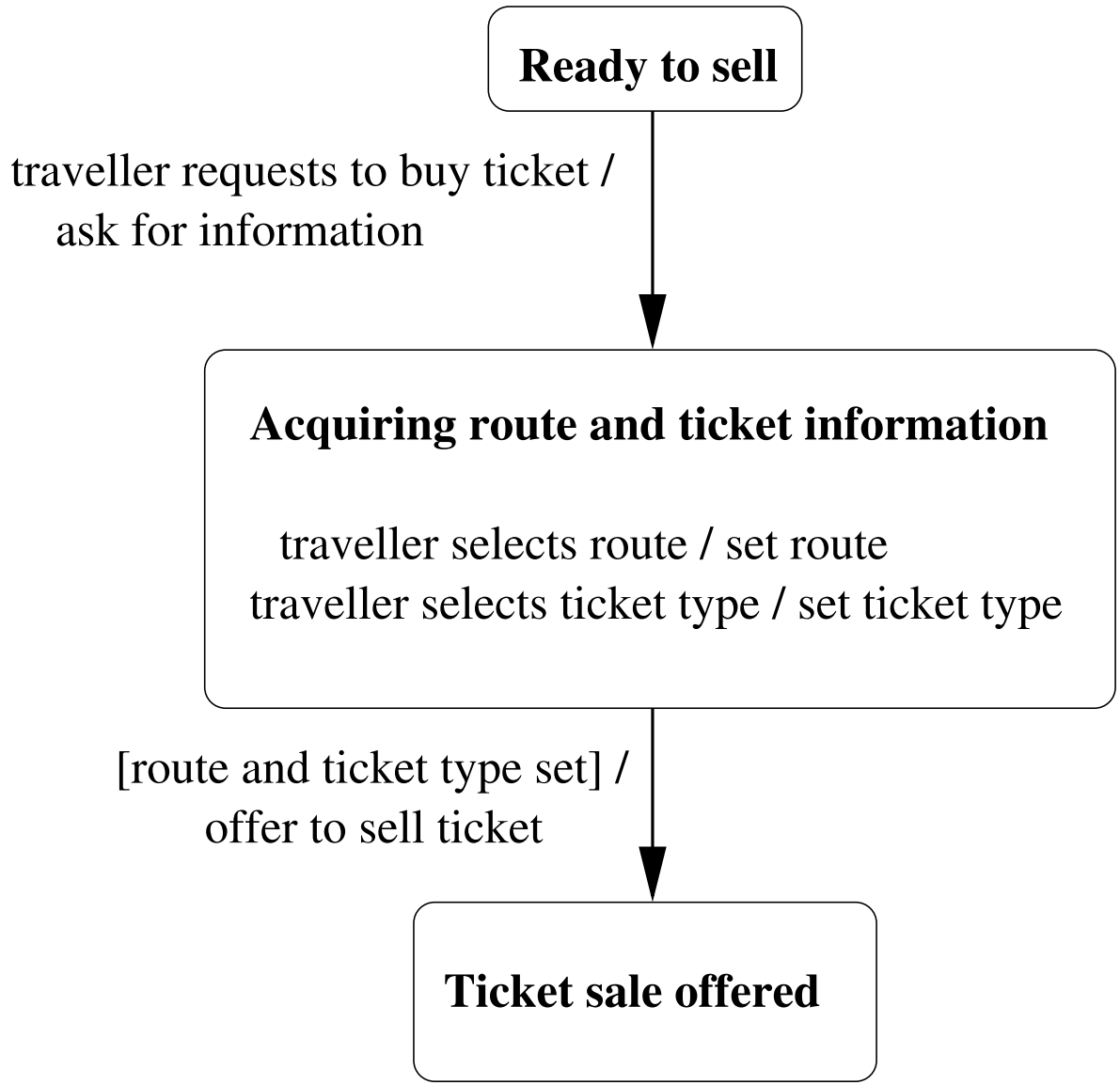
- Associated with each state there is an event list which defines the reactive behaviour in that state.
- Each entry in the event list has the form:

```
event expression[guard expression]/action
                                expressions
```

Two special events are:

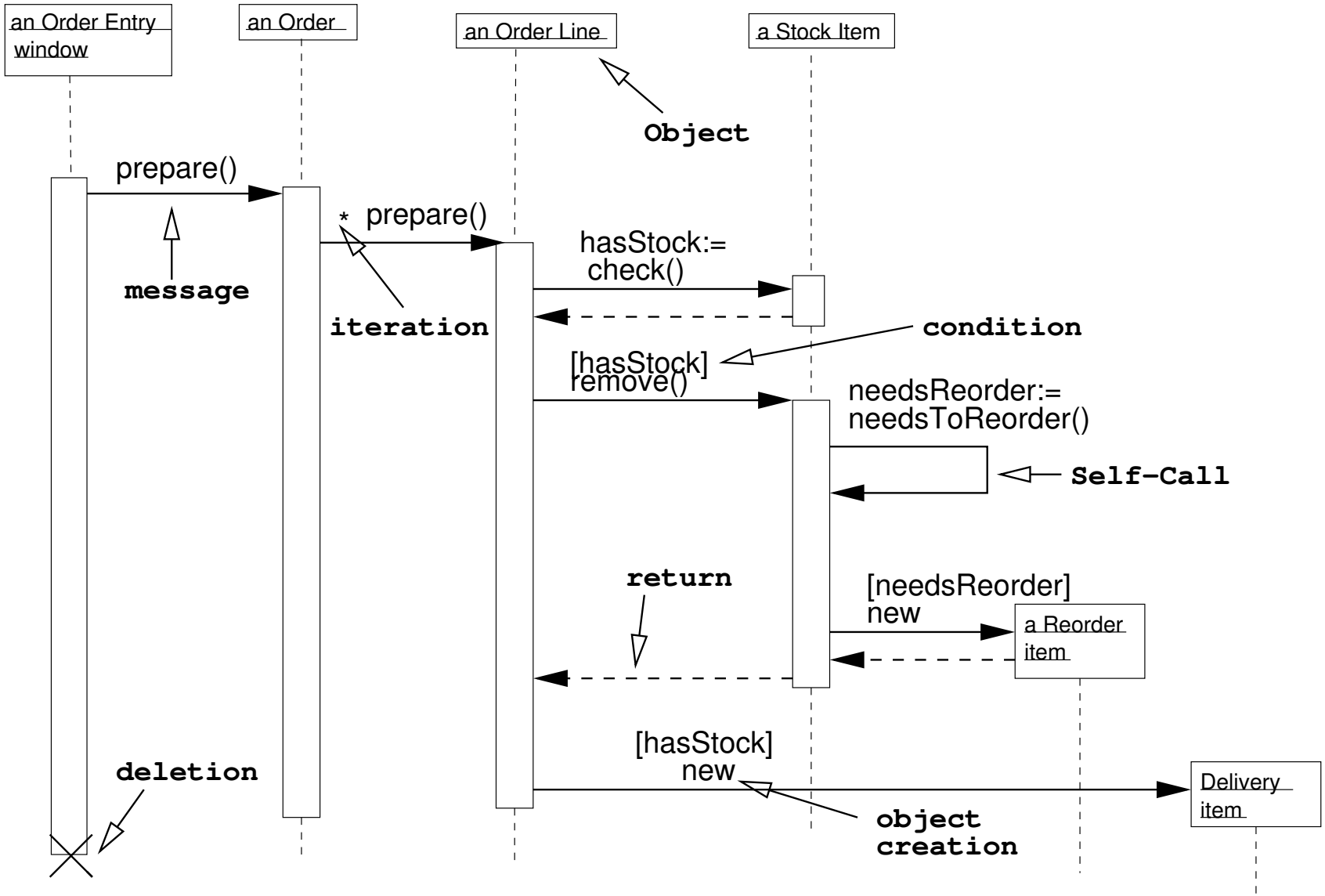
- *entry*: occurs when the state is entered.
- *exit*: occurs when the state is exited.

Statechart example: The train ticket machine

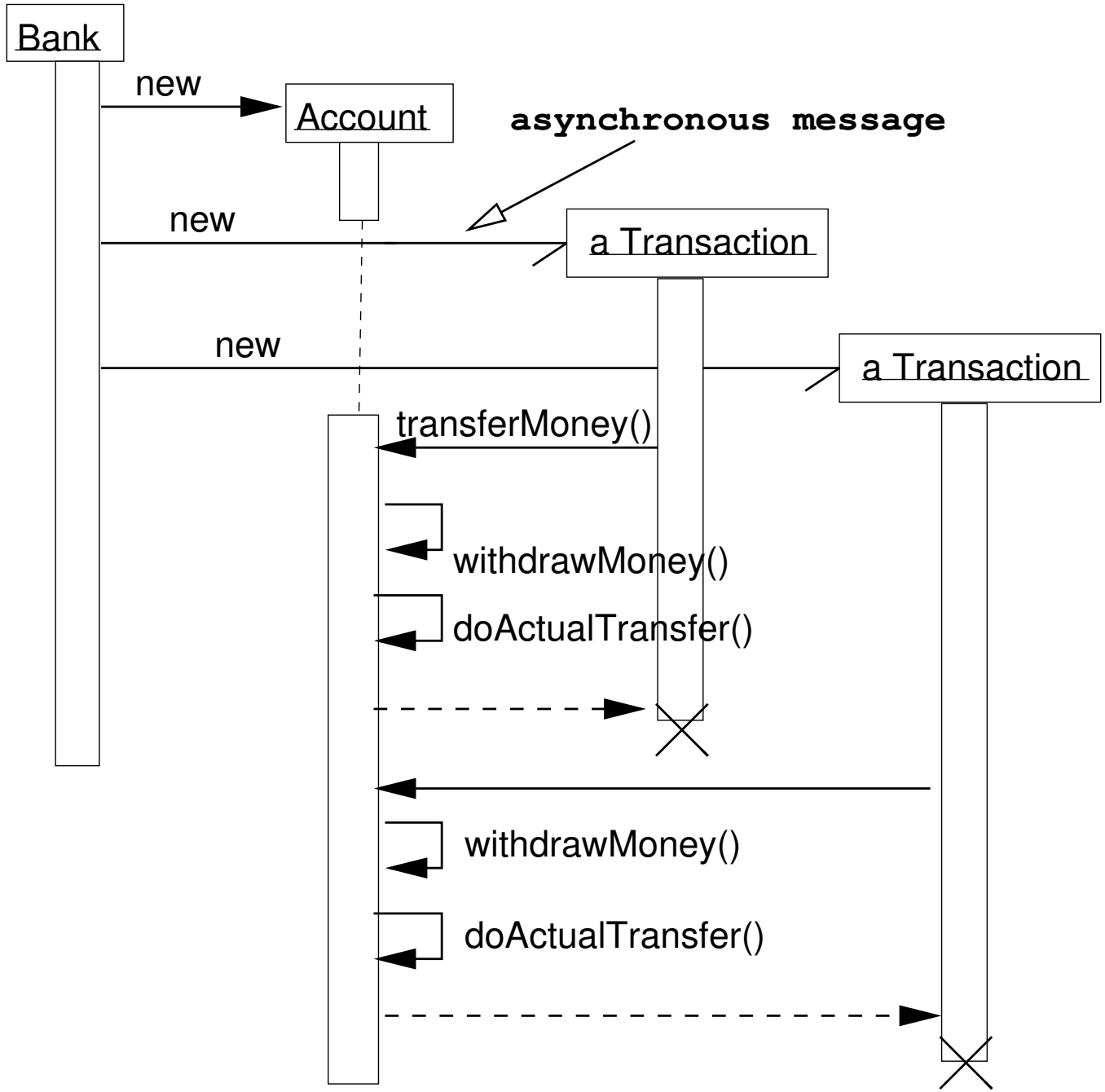


Sequence Diagrams

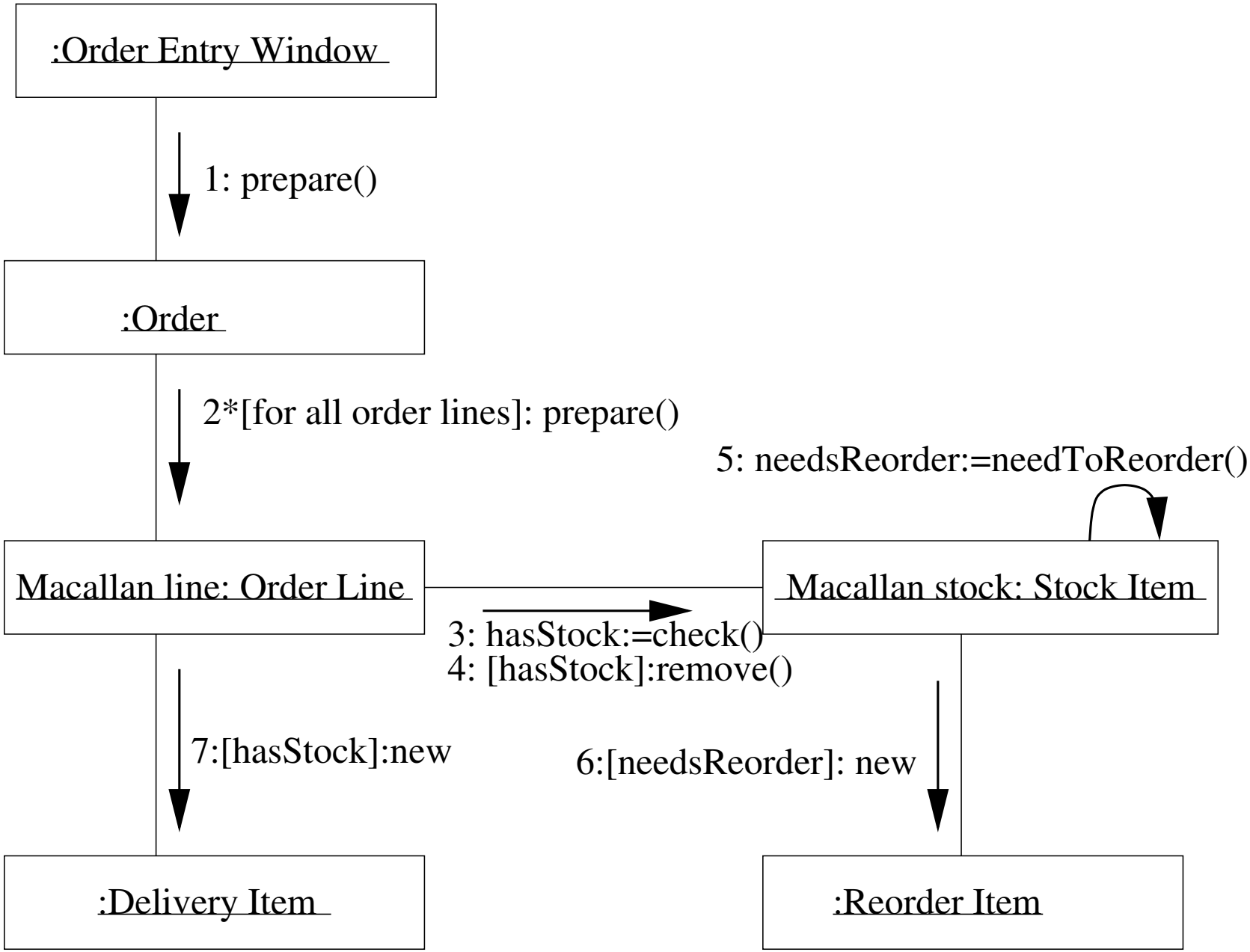
1. The Order Entry window sends a “prepare” message to an Order.
2. The Order then sends “prepare” to each Order Line on the Order.
3. Each Order Line checks the given Stock Item.
   - If this check returns “true”, the Order Line removes the appropriate quantity of Stock Item from stock and creates a delivery item.
   - If the Stock Item has fallen below the reorder level, that Stock Item requests a reorder.


an Order Entry window
an Order
an Order Line
a Stock Item
Object
prepare()
message
* prepare()
iteration
hasStock:= check()
condition
[hasStock] remove()
needsReorder:= needsToReorder()
Self-Call
return
[needsReorder] new
a Reorder item
[hasStock] new
Delivery item
deletion
object creation

```
class Transaction extends Thread {
    private Account account;
    Transaction(Account accnt) {
        account = accnt;
    }
    public void run() {
        account.transferMoney(100);
    }
}
class Account {
    public synchronized void transferMoney(int amount) {
        withdrawMoney(amount);
        doActualTransfer(amount);
    }
    public void withdrawMoney(int amount) { /* ...*/ }
    public void doActualTransfer(int amount) { /*...*/ }
}
class Bank {
    public static void main(String[] args) {
        Account currentAccount = new Account();
        for (int i=0; i < 2; i++)
            new Transaction(currentAccount).start();
    }
}
```

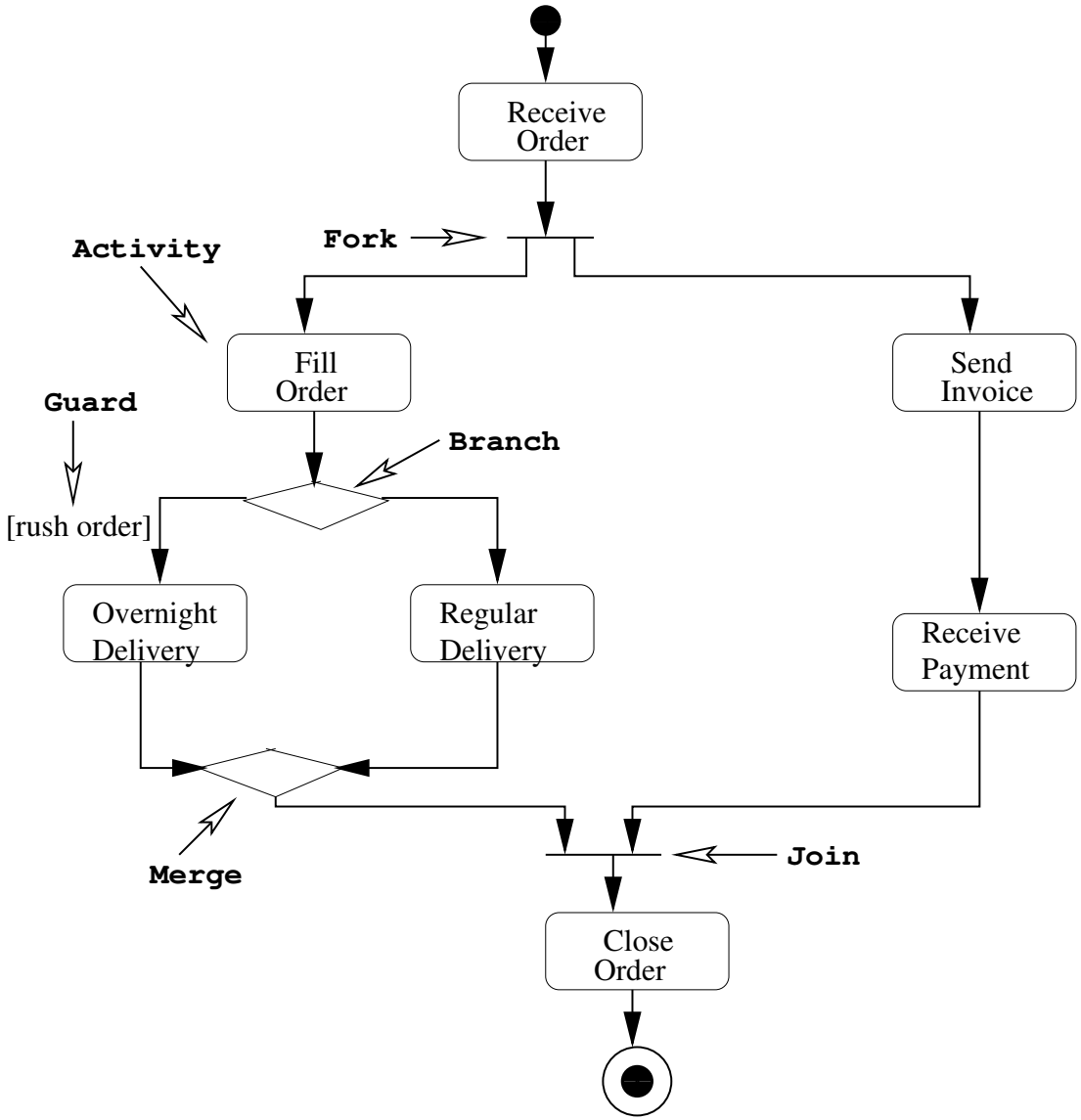

Bank
new
Account
asynchronous message
new
a Transaction
new
a Transaction
transferMoney()
withdrawMoney()
doActualTransfer()
withdrawMoney()
doActualTransfer()

Collaboration Diagrams



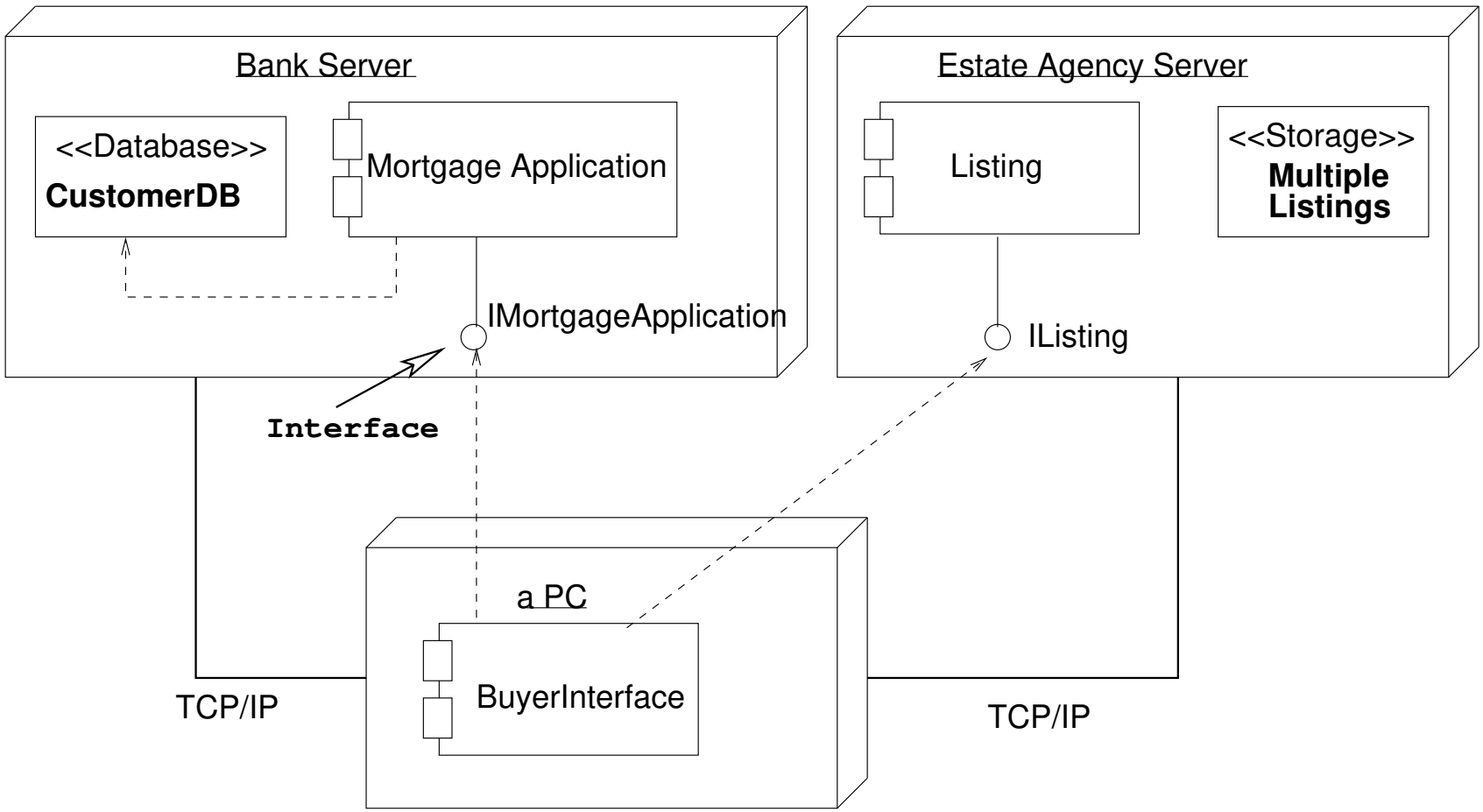
Activity Diagrams

A fancy flowchart borrowing ideas from flowcharts, Petri nets, event diagrams and state modelling.



Physical Diagrams

- *Deployment diagram:* shows the physical relationships among software and hardware components in the delivered system.
- *Component diagram:* these are the physical analogs of class diagrams. It shows the physical modules (packages) of code.


Bank Server
<<Database>>
CustomerDB
Mortgage Application
IMortgageApplication
Estate Agency Server
Listing
<<Storage>>
Multiple Listings
IListing
Interface
a PC
BuyerInterface
TCP/IP
TCP/IP

Tools for UML (Commercial and Public Domain)

- Rational Rose
- Together Control Center (TogetherSoft Corporation)
- Inkscape
- Xfig
- Any of your favourite drawing software