COMP201 – Software Engineering 1
Lecture 20 – Intro to UML

Lecturer: T. Carroll

Email: Thomas.Carroll2@Liverpool.ac.uk

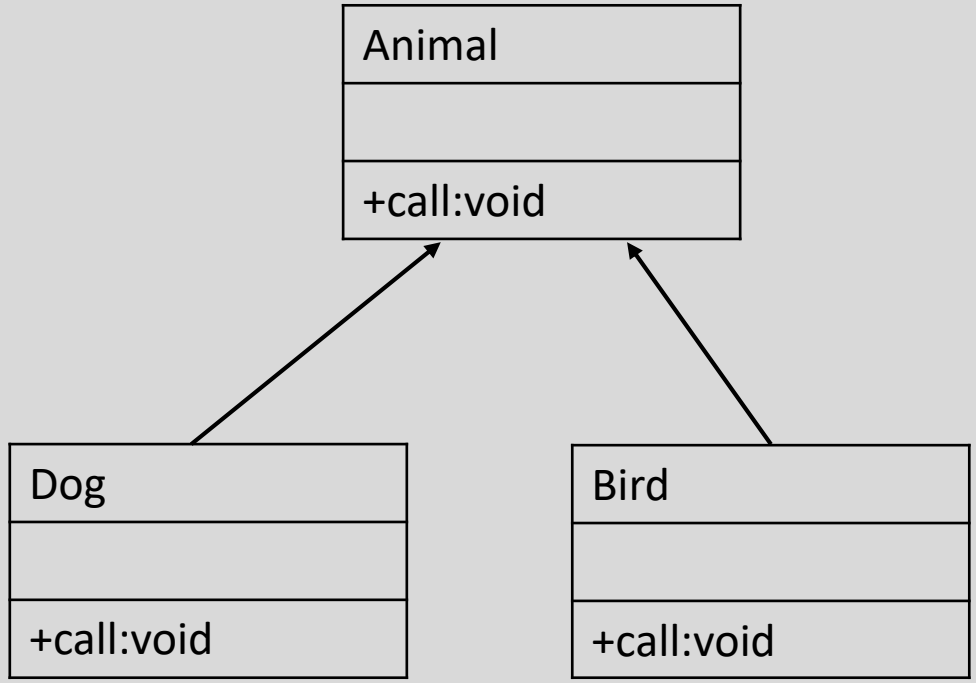
Office: G.14

See Vital for all notes

Recap – Polymorphism and Dynamic Binding

Object Polymorphism

- Polymorphism allows the programmer to use a subclass anywhere that a superclass is expected.




```
Animal myPet = new Animal();

Animal myDog = new Dog();

Animal myBird = new Bird();
```

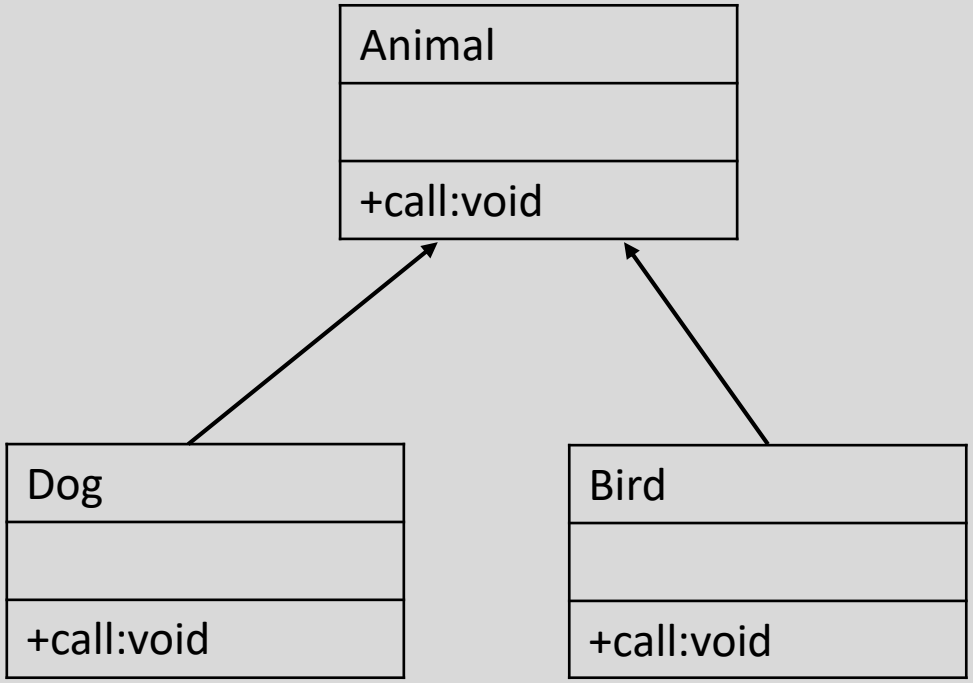
```
Bird myBirdy = new Dog();

Dog myDoggo = new Animal();
```

If the types follow a IS-A relation, then Polymorphism will work

Dynamic Binding

- Dynamic Binding is when an object determines (possibly at run time) which code to execute as the result of a method call on a base type.




```
Animal myPet = new Animal();
Animal myDog = new Dog();
Animal myBird = new Bird();

myPet.call();
myDog.call();
myBird.call();
```

Which particular methods will be called?

Coming Up…

Unified Modelling Language (UML)

- UML is a language for:
  - Specifying
  - Visualizing and
  - Documenting

  the parts of a system under development
- Different types of UML models:
  - Use case models – describing a system from the users' point of view
  - Static models – describing the elements of the system and their relationship
  - Dynamic models – describing the behaviour of the system over time
- UML has been adopted by the Object Management Group (OMG) as an Object-Oriented notation standard
- Today, we use a *case study* to **introduce** various UML diagrams

Requirements

The Problem

- You have been contacted to develop a computer system for a clinic.
- The clinic needs the following types of service:
  - Staff management
  - Booking appointments
  - Keeping records of patients and prescribed medication
- You must build an interactive system which handles all of these aspects online.

The most difficult part of any design project is understanding the task

Clarifying the Requirements

- Different users will have different, sometimes conflicting, priorities
- Users are not likely to have clear, easily expressed views of what they want
- It is hard to imagine working with a system of which you have only seen a description

Facts about the Requirements

- Staff management: Doctors, Nurses, Admin staff
- Patients
- Appointments
- Prescribed Medications

TASK:
Specify the facts about the requirements that an ideal system would satisfy.

Use Case Model

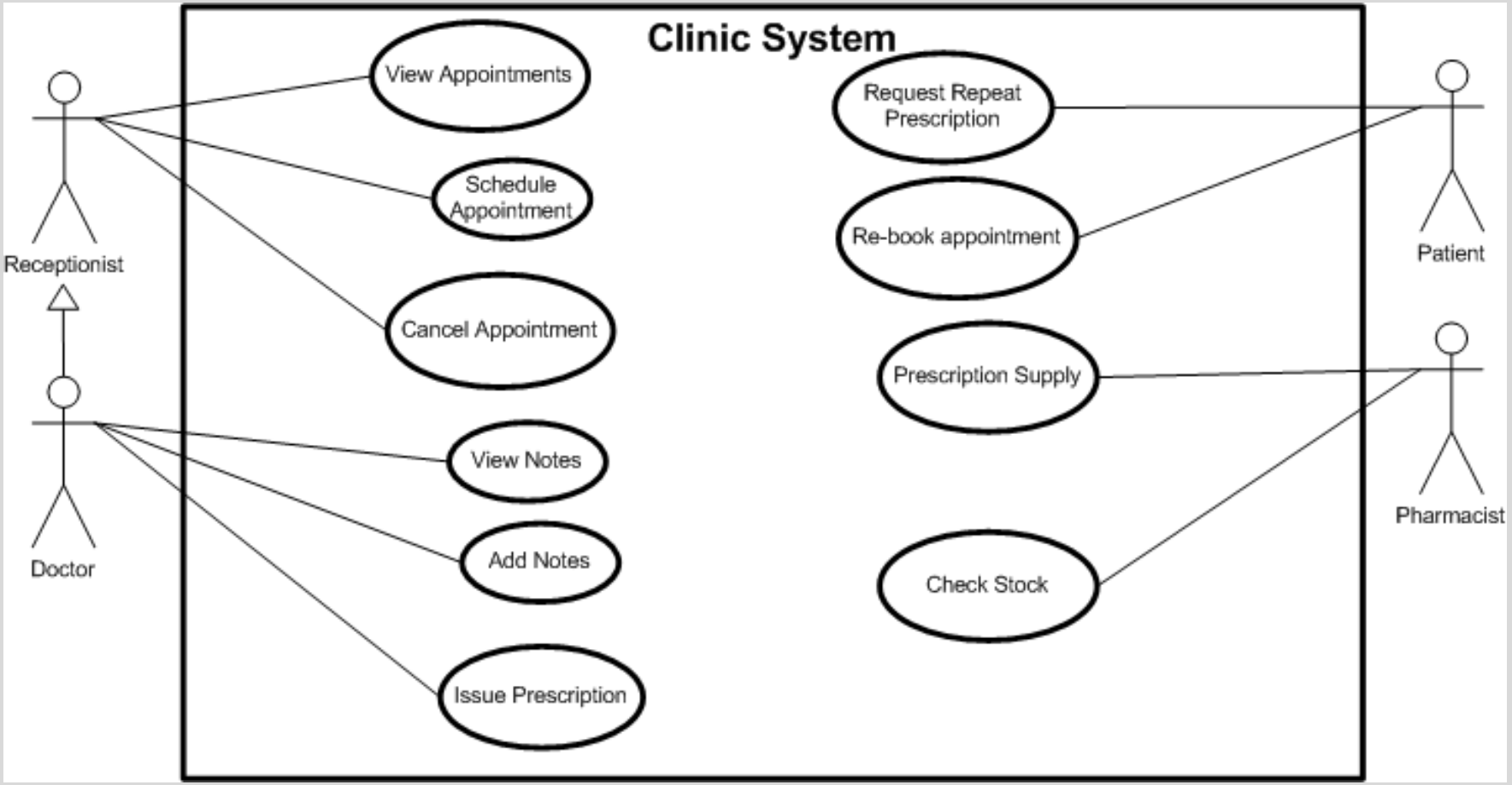
Use Case Model

- A high quality system must meet the needs of its users.
- We take a user-oriented approach to systems analysis.
- We identify the users of the system
- We identify the tasks they must undertake with the system.
- We seek information about which tasks are most important, so that we can plan the development

…“Users”? …“Tasks”?

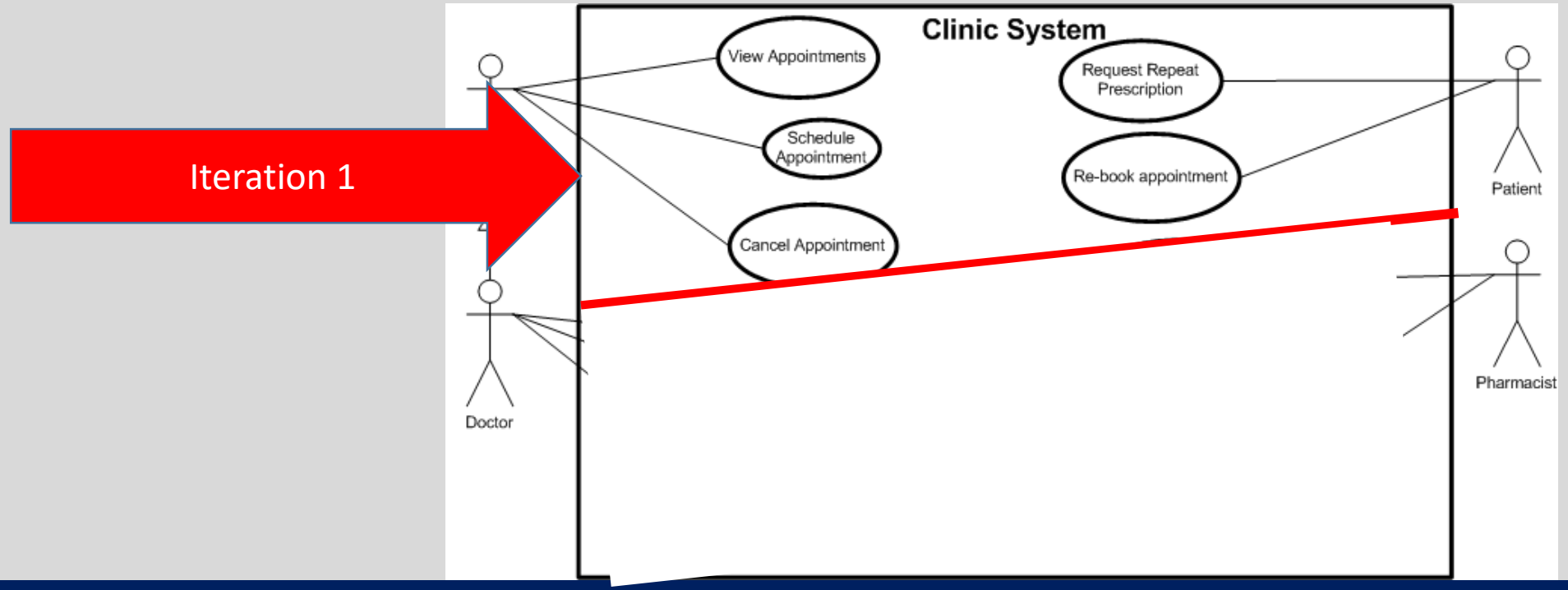
- UML uses as technical terms “actors” and “use cases”
- An actor is a *user* of a system in a particular role
  - can also be an external system
- A use case is a *task* which an actor needs to perform with the help of the system

Use Case Diagram of Clinic System



Scope and Iterations

- To limit the risk, we develop in iterations.
  - The first iteration delivers a system with only the most basic and essential functionality;
  - Later iterations enhance the system
- Use cases to help identify suitable dividing lines between interactions



Limiting Requirements

- It is important **not to invent new requirements** for the system.
- Perhaps it would be good to inform the doctor that a drug is out of stock via their online system?
  - But this may not be what is wanted by the Doctor
  - This may cause an overload of information!

This is **NOT your** system!
This system belongs to the customer and you should deliver software to THEIR requirements,
NOT what you THINK should be delivered

Use Case Advantages

- It may be easier to identify the amount of time required to implement all the required features of the system.
  - Such details can often be optimistically overlooked.
- We can identify which requirements are important to key personnel in the company.
  - By providing this functionality early, we can show the potential value of the software and avoid the project being cancelled.

Use Case Advantages

- We may decide to implement more risky use cases first
- This allows flexibility to tackle any problems that arise
- Use cases can be used to derive validation checks on the developed system
- This can show it provides all required functionality.

Identifying Classes

Identifying Classes

- Identifying the right classes is one of the main skills of OO development.
- We start the process of identifying the *key domain abstractions* using the ***noun identification technique***.

1. Take a coherent, concise statement of the requirement of the system
2. Underline its *noun and noun phrases*, that is, identify the words and phases that denote things
3. This gives a list of candidate classes
4. Whittle down and modify to get an initial class list for the system

Task: Identify Candidate Classes

Clinic system

Before seeing a doctor or nurse, the patient needs to make an appointment.

The appointment will be made by the receptionist.

Before making the appointment, the receptionist needs to ask the patient which doctor they wish to see and if the appointment is a standard appointment or urgent appointment.

The receptionist will use this information, check the appointment schedule, and find a free slot. They will then make the booking.

Receptionists can also cancel appointments.

When the patient sees the doctor, the doctor will sometimes issue a prescription.

Each doctor has a maximum of 2000 patients registered to them.

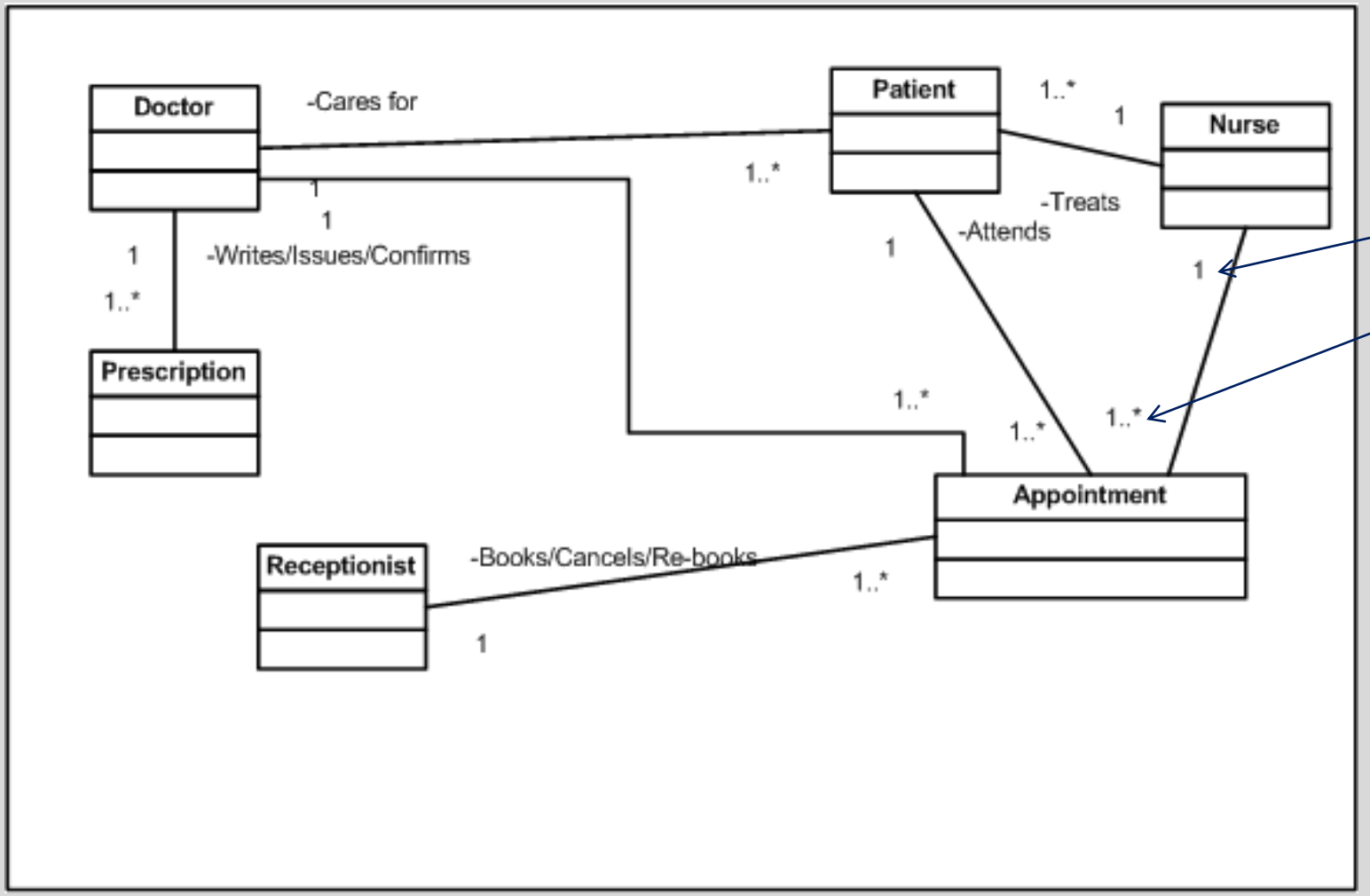
The patient may request a repeat issue of their prescription.

Class Diagrams

Relations between Classes

- Next we identify and name important real-world relationships or associations between our classes
- We do this for two reasons:
  - To clarify our understanding of the domain, by describing our objects in terms of how they work together;
  - To sanity-check the coupling in our system, i.e. make sure that we are following good principles in *modularising our design*

Initial Class Model of the Health Clinic

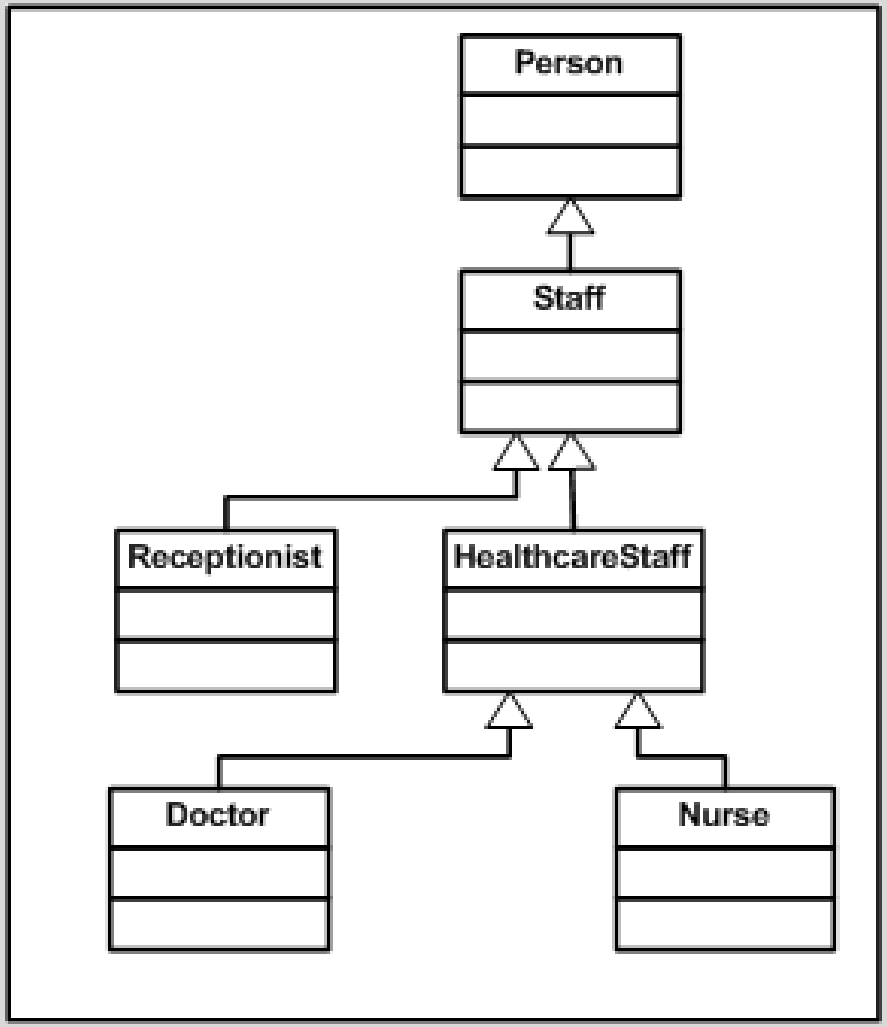


These are ***multiplicities*** which we shall study in detail soon

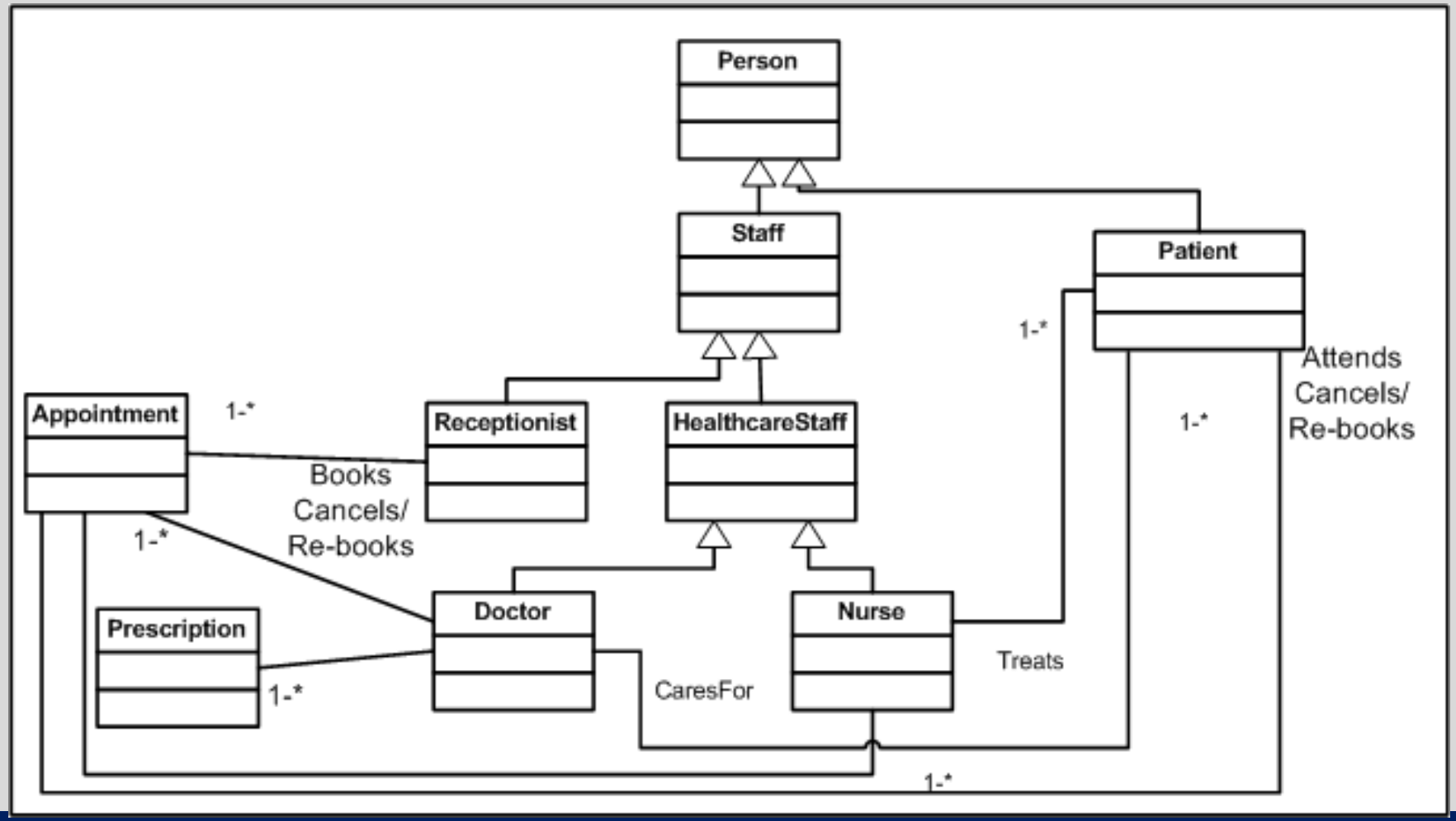
A Closer Look….

- We notice:
  - Doctor shares all the same associations that Nurse does
- Recording this in the class diagram will clarify our understanding of the situation
- Inheritance:
  - Dr and Nurse are both healthcare staff
  - Healthcare staff are staff
  - Receptionist is also staff

Revised Health Class Model (hierarchy)



Revised Health Class Model



Sequence Diagrams

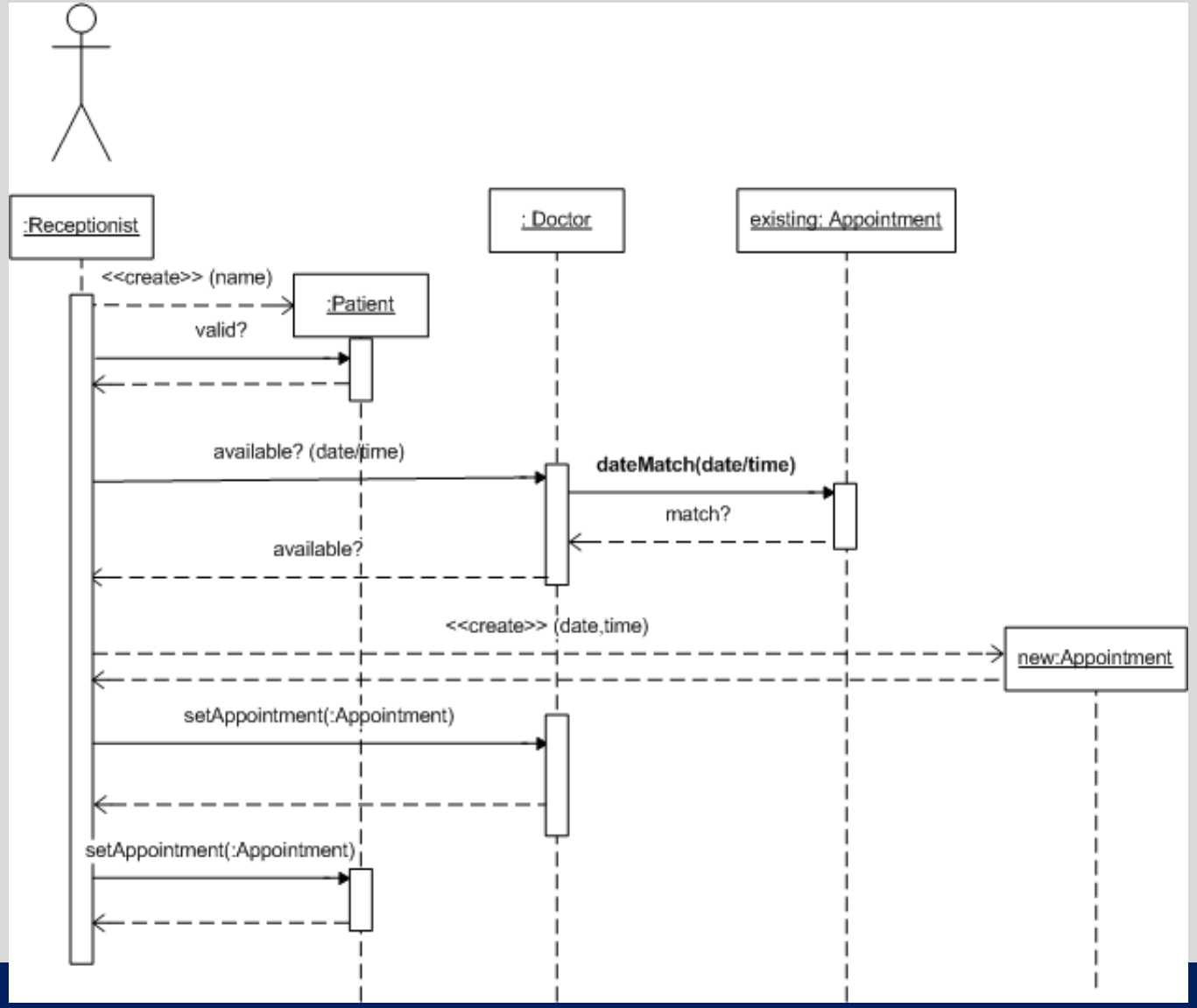
The System in Action

- A class diagram gives a **static view** of the system
- We know nothing about the **dynamic behaviour**
- In UML we can use interaction diagrams to show how messages pass between objects of the system to carry out some task
  - This will also show how the various classes realize the different use cases we identified in the use case diagram

An Example Sequence Diagram: Booking an Appointment

- Consider what happens in the appointment booking scenario:
  - A patient wishes to make an appointment
  - The **receptionist must check that the person is a valid patient**
  - Then the doctor object must be checked to see if there are any available appointments
  - If there are suitable slots available, a new appointment should be created and assigned to the doctor.

Interaction Shown on a Sequence Diagram

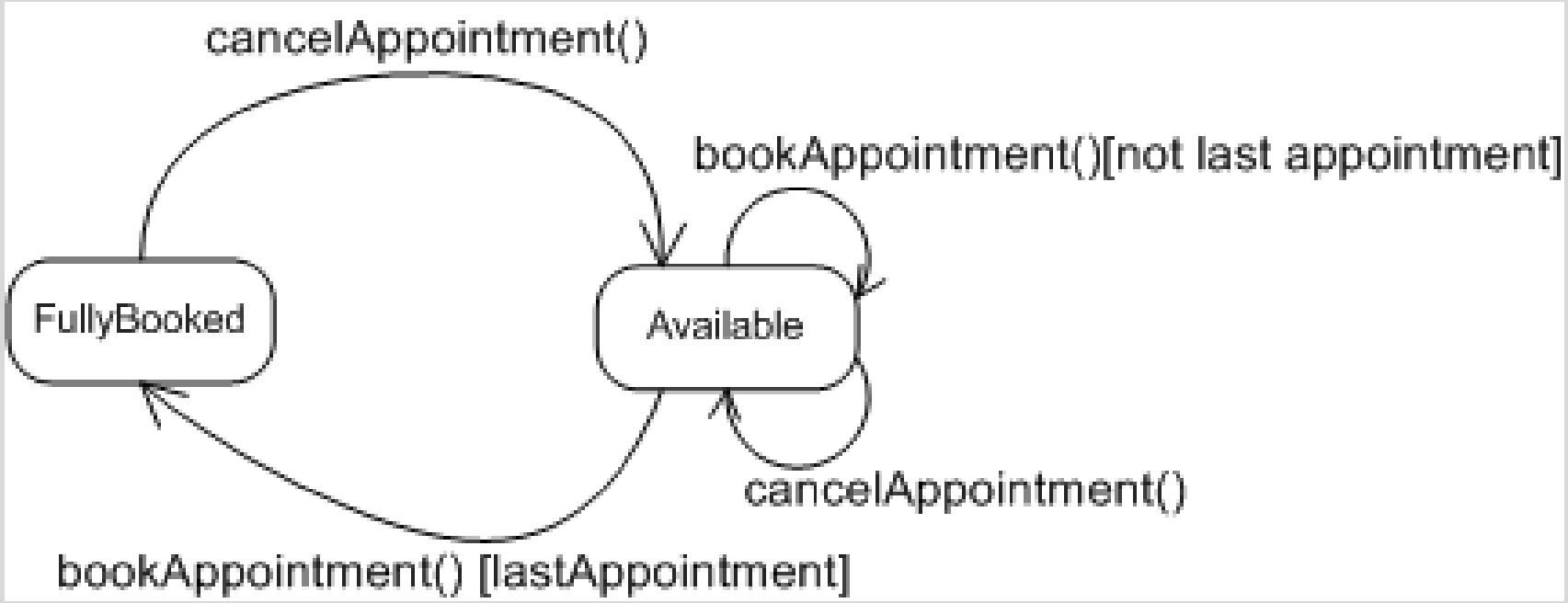


State Diagrams

State Diagrams

- Objects in the system have a **state**
- Eg: A Doctor can be *available* or *fully booked*
- Running methods on the object can cause a change in state
  - i.e., by booking appointments to the doctor object we change its internal state.
- Changes in object states can be modelled by a **state diagram**.

Example State Diagram: Doctor Availability



Lecture Key Points

- We have seen an *introduction* to the **Unified Modelling Language** (UML)
- We studied an introductory case study based on a health clinic system with an introduction to:
  - Use case diagrams
  - Class diagrams
  - Sequence diagrams
  - State diagrams