COMP201 – Software Engineering 1
Lecture 21 – UML Class Models

Lecturer: T. Carroll

Email: Thomas.Carrroll2@Liverpool.ac.uk

Office: G.14

See Vital for all notes

Special Office Hour:

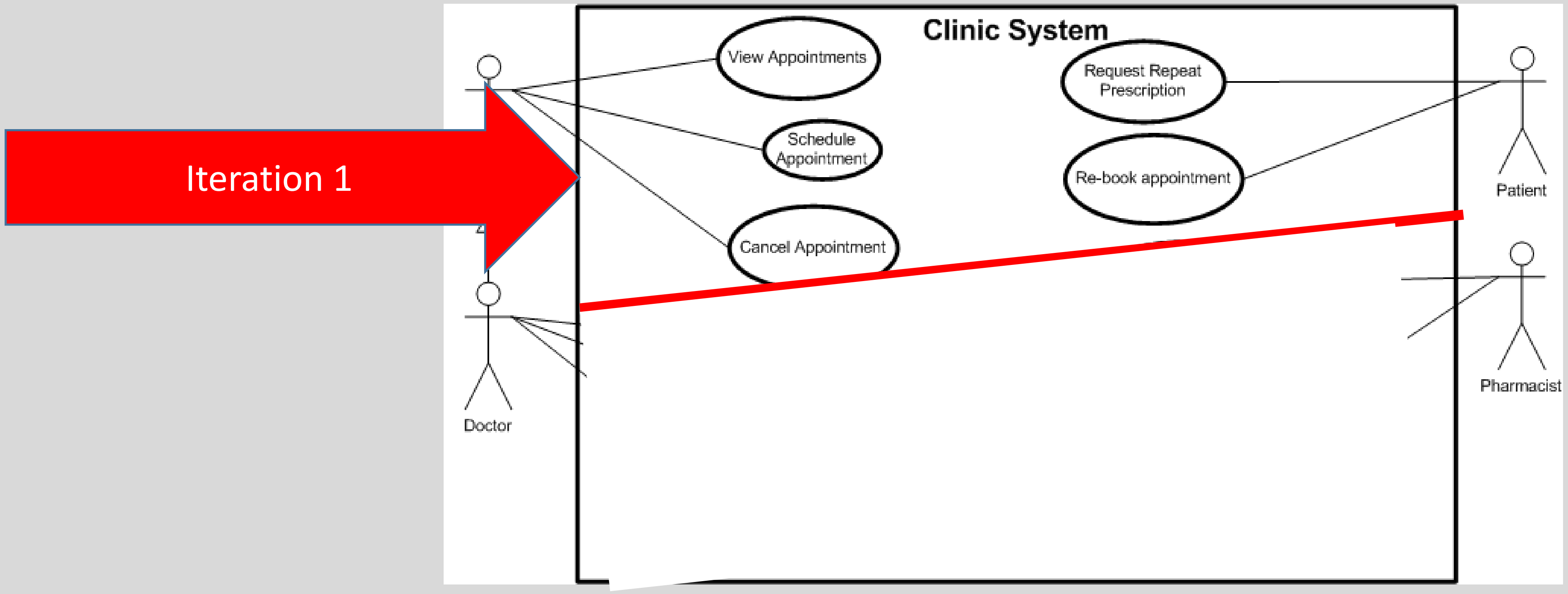
Monday 11th November
1200-1300
Ashton Lecture Theatre.

Recap –
Clinic System

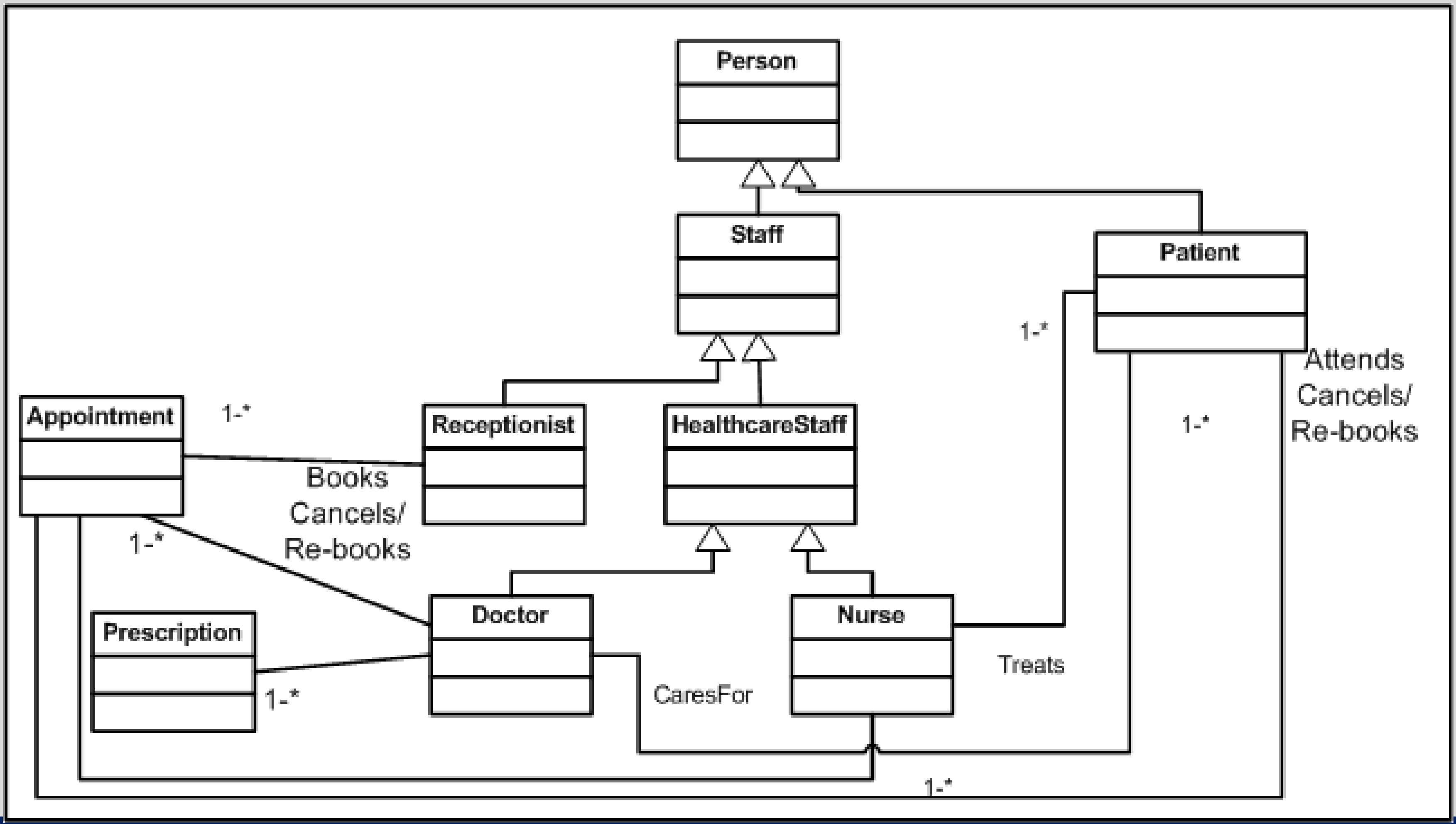
Recap – Lecture 20

- Introduction to UML through a *case study*
  - Clinic system:
    - Staff management (Receptionist, Doctor, Nurse)
    - Patient management
    - Appointments and Prescriptions
- Use case diagrams give a **user oriented** perspective
  - Actors: Users playing a role
  - Use cases: Tasks the user does with the system (don't show workflow)
- Identify candidate use cases by finding **active verb phrases**
- Identify candidate classes by finding **nouns/noun phrases**
- Class diagrams show the **association** between classes

Use Case Diagram of Clinic System



Revised Health Class Model



Today

Lecture 21

- Finish case study:
  - Sequence Diagram
  - State Diagram
- UML Class Models:

Sequence Diagrams

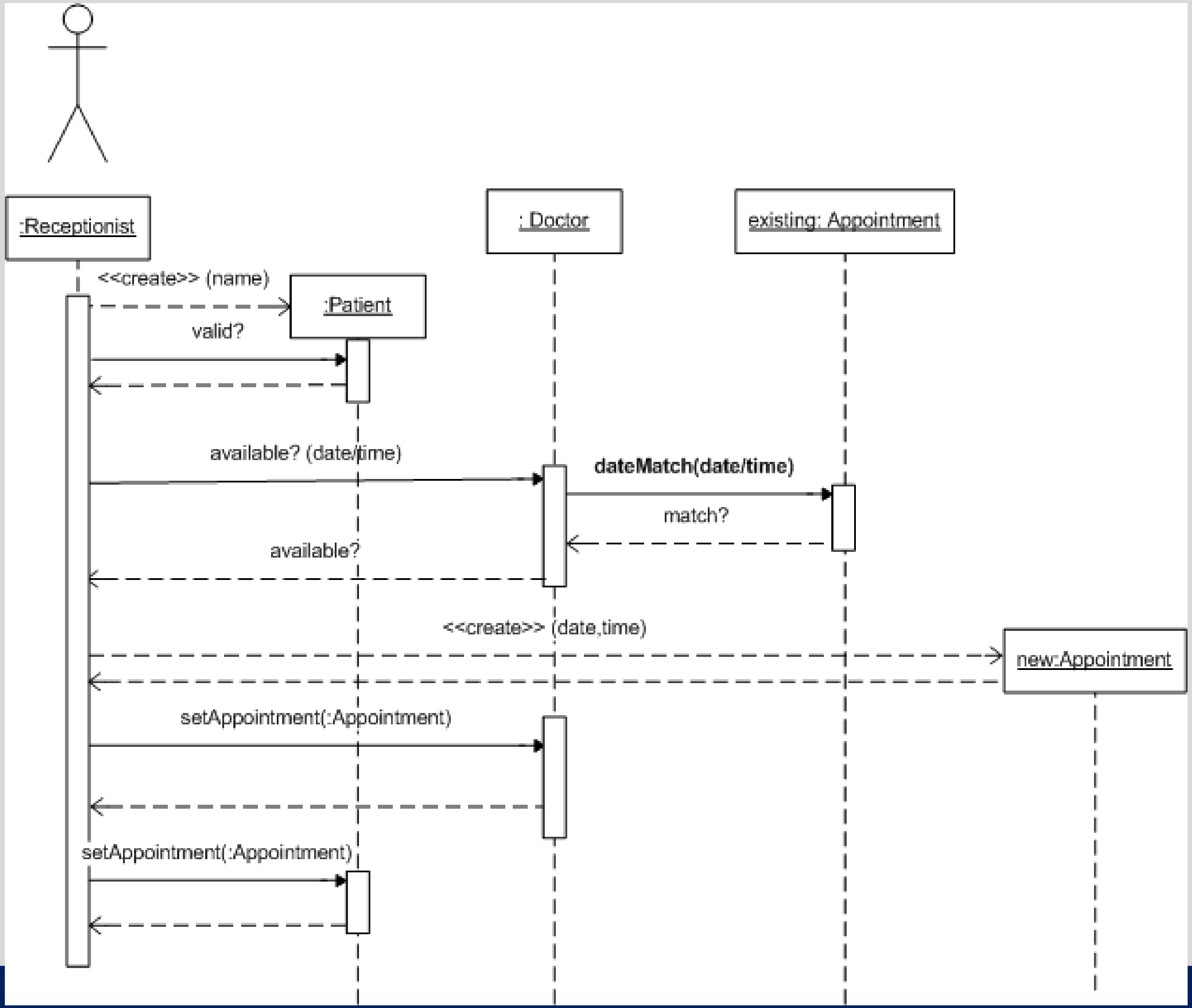
The System in Action

- 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

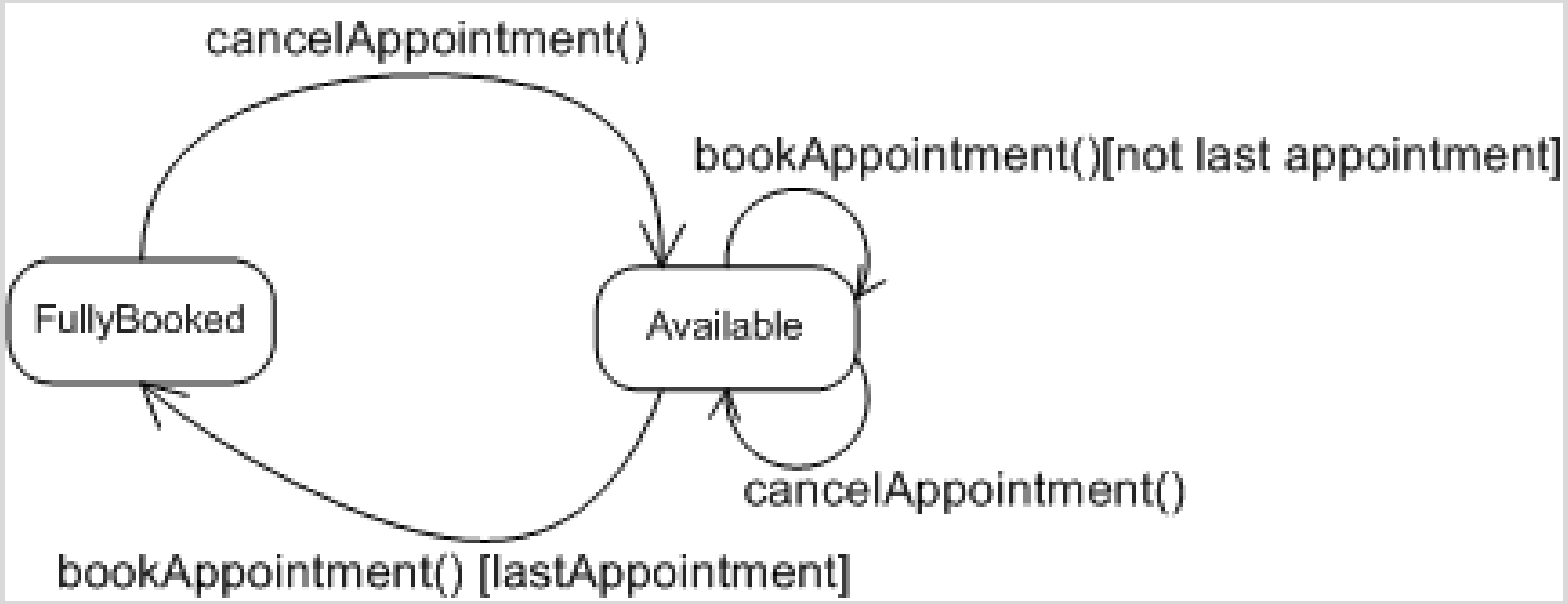


State Diagrams

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


cancelAppointment()
bookAppointment()[not last appointment]
FullyBooked
Available
cancelAppointment()
bookAppointment() [lastAppointment]

In detail:
Class Models

What Makes a Good Class Model?

- A class model shows the **classes** of the system and the **association** between them.
- A class model should *help us* to meet objectives:
  1. Build a system which satisfies all current requirements;
  2. Build a system which will be easy to maintain and evolve
- **A good class model** consists of classes which don't depend on the particular functionality required today
- Good classes have **good names**

All required behaviour must be provided by the objects

Build a system with encapsulated modules, loose coupling, high cohesion

Deriving Classes

- We have seen the ***noun identification technique*** to find potential classes.
- There are two main (extreme) techniques used to find classes in general:
  - **Data-driven-design** (DDD)
    - Identify all the data of the system
    - Divide it into classes
    - Assign particular responsibilities (methods) to these classes
  - **Responsibility-driven-design** (RDD)
    - Identify all the responsibilities of the system
    - Divide them into classes
    - Find the data each class requires.

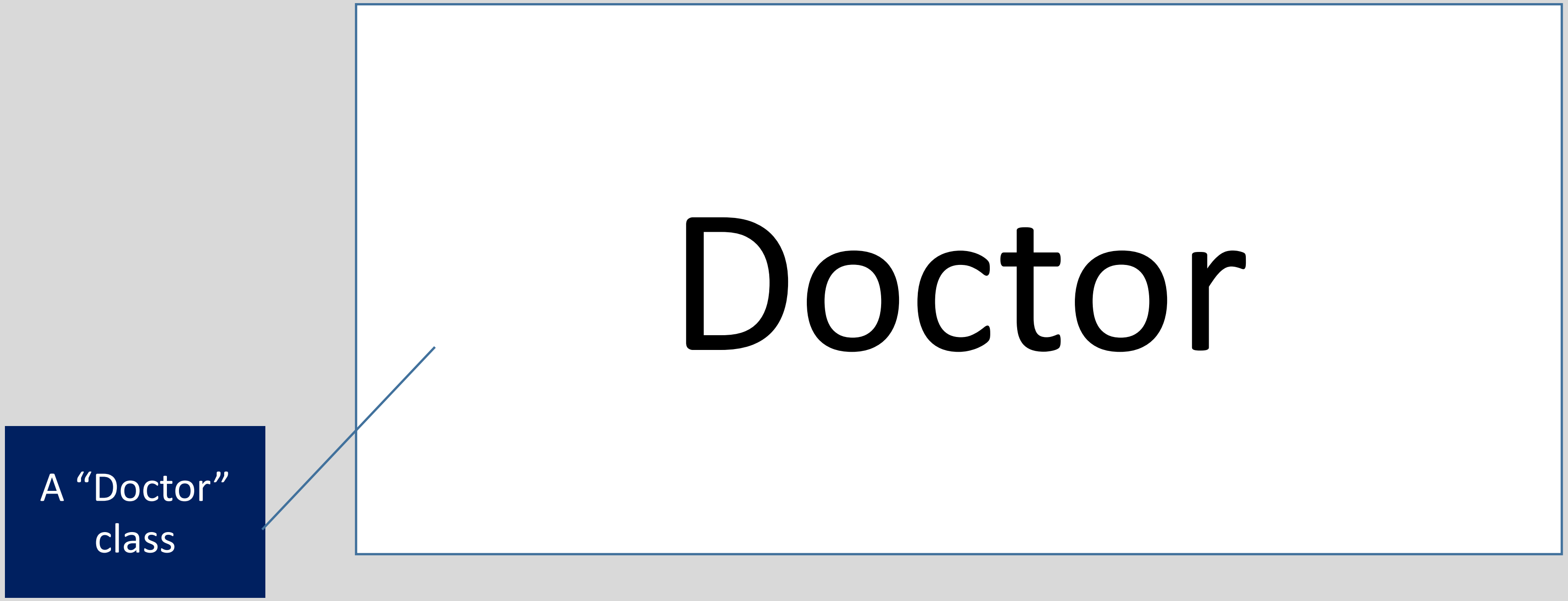
What’s in a name?
How to name classes

- No Plurals
  - Persons
  - People
- No Verbs
  - Encrypt
  - Dial
- No General Process Descriptors
  - Encryption
  - Dialling
  - Printing

- Person
- Encryptor
- EncryptionHelper
- Dialler
- Printer
- PrintManager

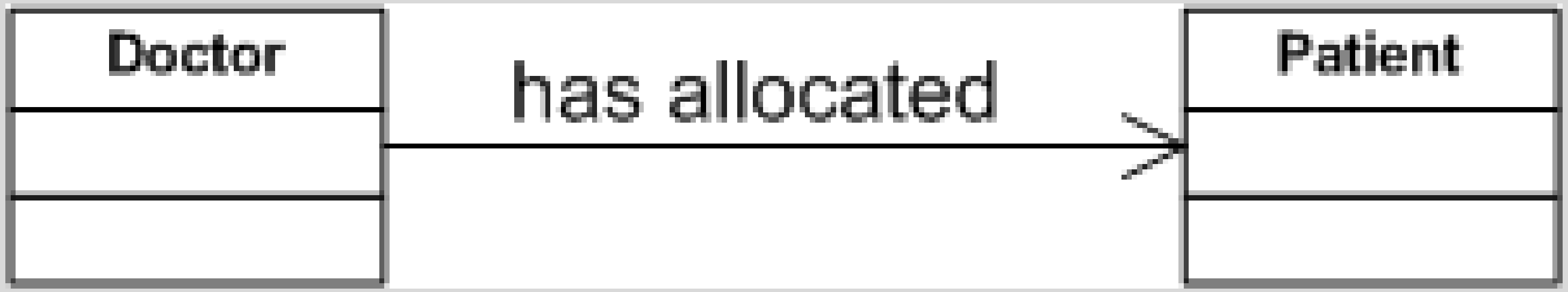
A Very Simple Class Model

- In UML Class Models, a class is shown as a rectangle containing the class name:



Associations

- Classes (sort of) correspond to nouns
- Associations (sort of) correspond to verbs.
- Associations express the relationship between classes.
- There are instances of classes (objects)
- There are instances of associations (**links** in UML)



Associations – relationships between classes

- Classes (sort of) correspond to nouns -- Associations (sort of) correspond to verbs.

Example Associations:

- an object of class A sends a message to an object of class B
- an object of class A creates an object of class B
- an object of class A has an attribute of type class B
- an object of class A receives a message with an object of class B as argument

An object of class A has to know about an object of class B

Multiplicity

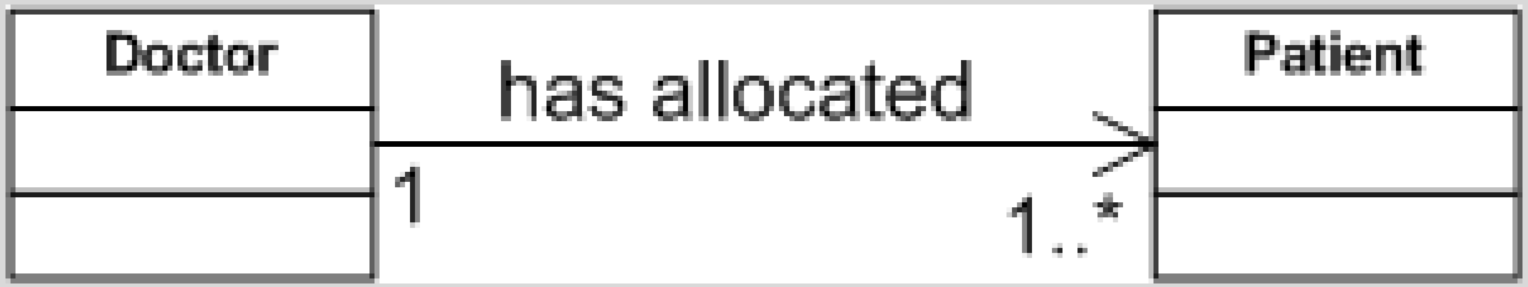
- Multiplicity allows us to quantify an association
    - Eg: For cases where a Doctor has multiple Patients

We can specify:

- **an exact number** simply by writing it, e.g. **1**
- **a range of numbers** using *two dots* between a pair of numbers, e.g., **1..10**
- **an arbitrary, unspecified number** using ***** (upto infinity)

Example

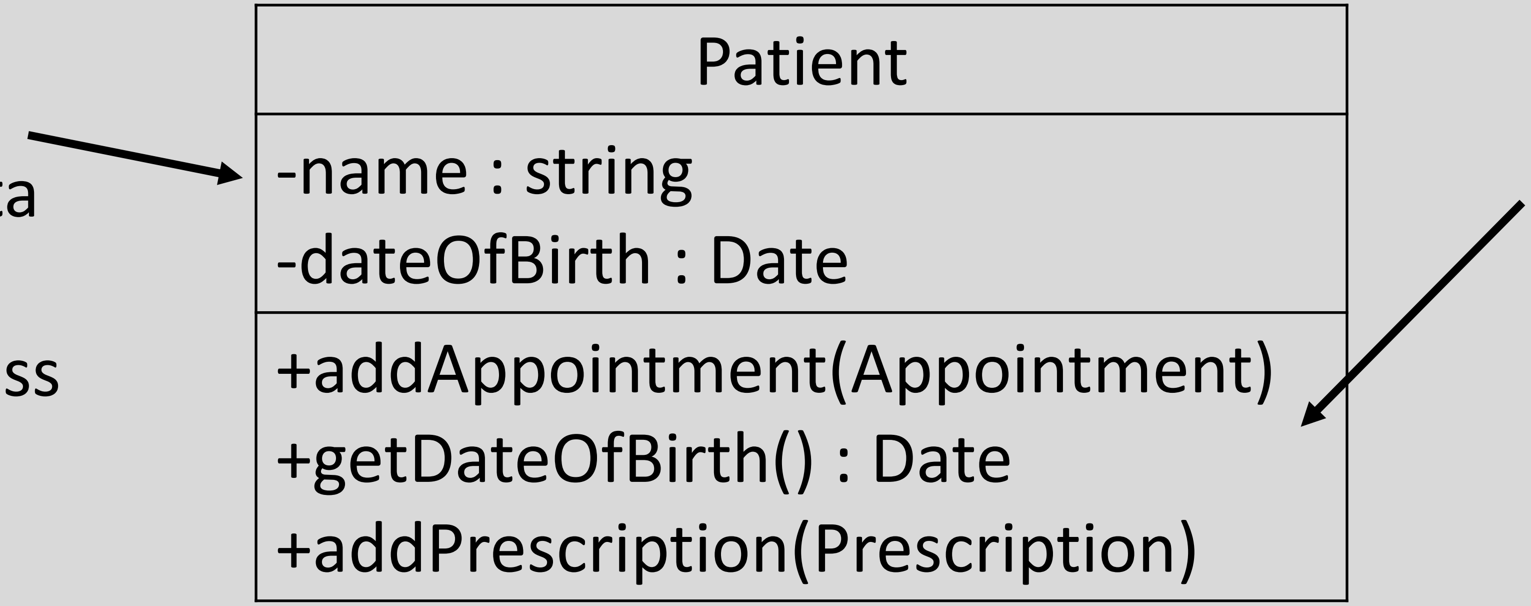
- *1 Doctor* is associated by *has allocated* with (possibly) multiple patients.
  - **1** at the Doctor end of the association (alternately we can use 1..1)
  - **1..*** at the Patient end of the association



```
class Doctor{
       private:
              string name;
              const string gmcNumber;
              vector<Patient> patients;
       …
```

Attributes and Operations

Attributes
describe the data contained in an object of the class and their type

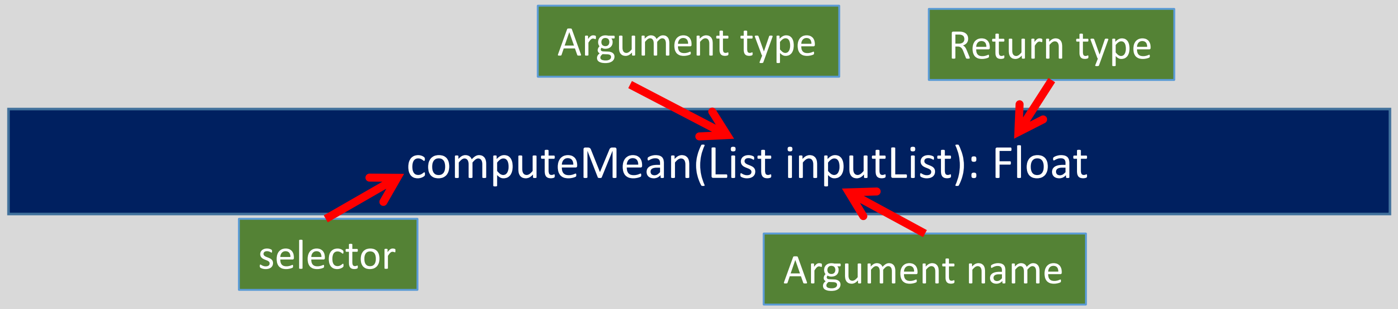


Operations
define the ways objects may interact

+	**Public**
-	**Private**
#	**Protected**

Operation Signatures

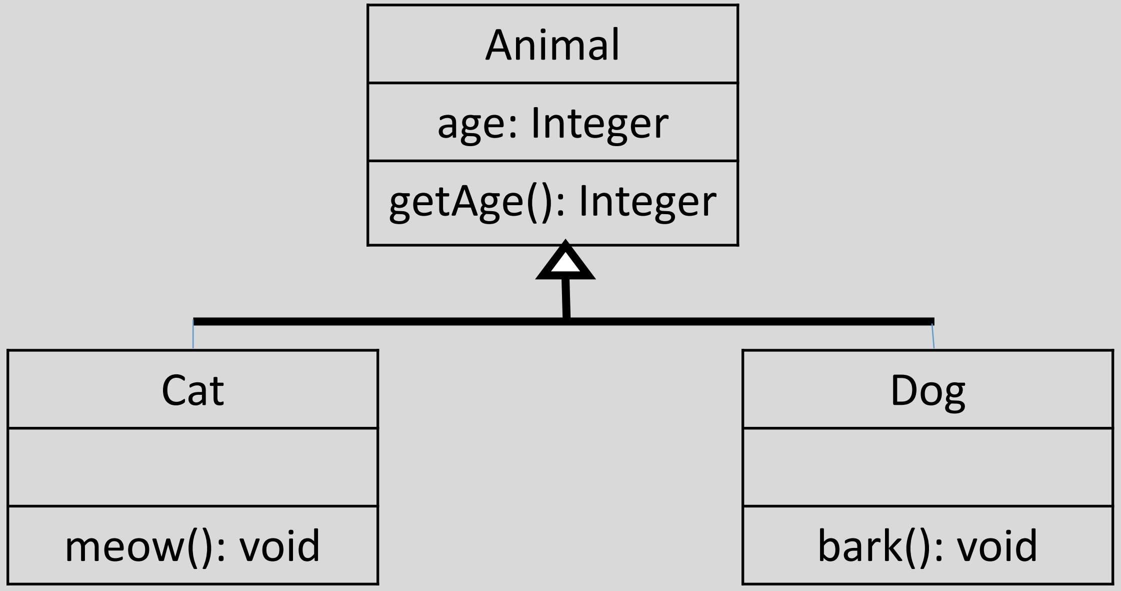
- The **signature** of an operation gives the
  - *Selector*
  - *Names*
  - *types* of all formal parameters (arguments)
  - *return type*



Generalization

- If class **A** is a generalization of class **B**:
  - the interface of class **B** must conform to the interface of class **A**.
- Every attribute and operation of **A** will also be supported by **B**.
- **B** may contain some *extra* operations and data specific to its class.

Recall
Dynamic Binding



CRC Cards

CRC Cards

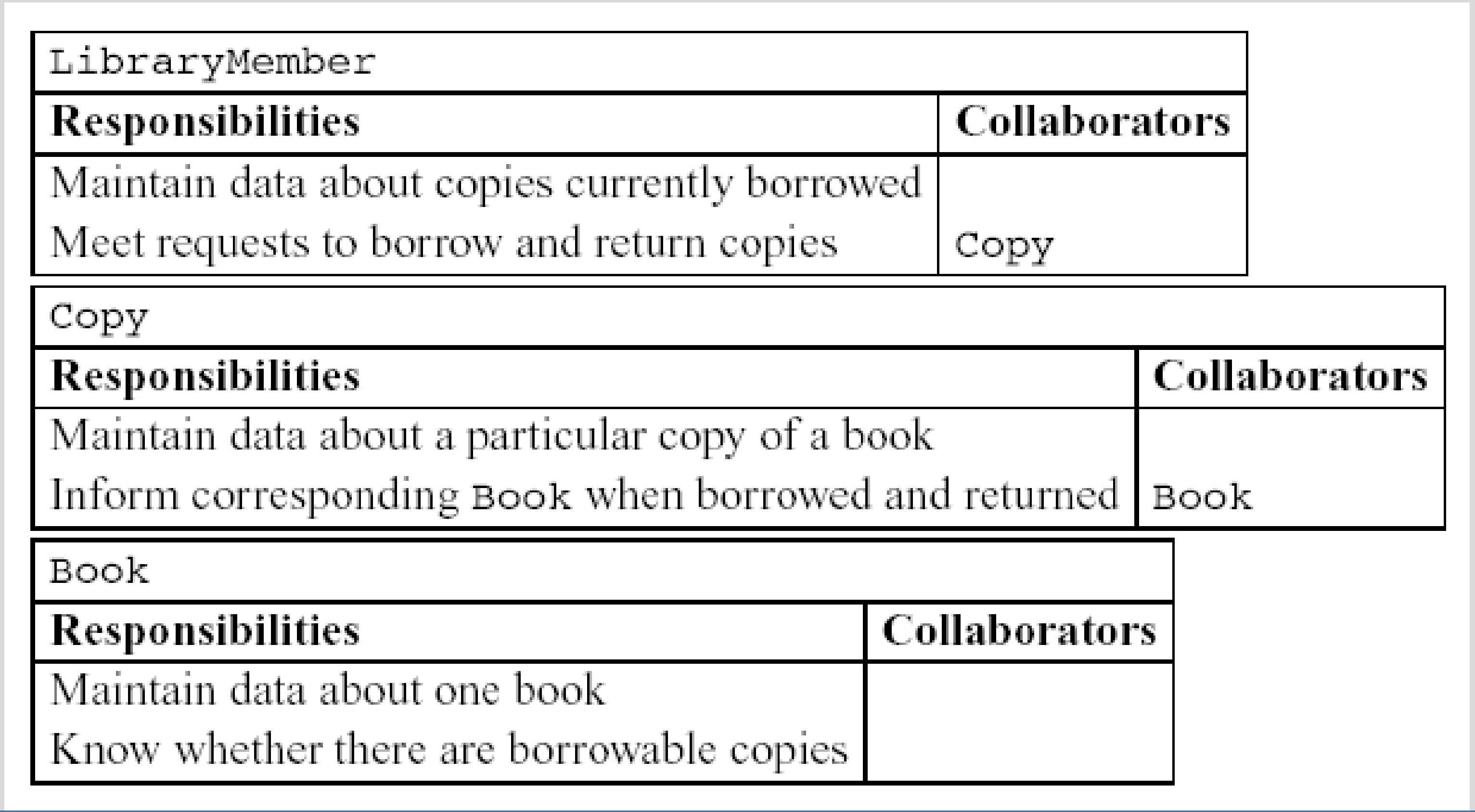
- Classes, Responsibilities, Collaborations.
- CRC Cards help check for a good design and guide refinement
- CRC is not part of UML
  - adds some very useful insights throughout a development.

Creating CRC Cards

- **The name of a class**, at the top
- **The responsibilities of the class**, on the left-hand side
  - Reasons for class's existence
  - no more than 3 or 4 per class
- **The collaborators of the class**, which help to carry out each responsibility, on the right-hand side.
  - Try to limit the amount of collaborators

Class Name	
Responsibilities	Collaborators
R1,R2,…	C1, C2…

CRC Card Example



LibraryMember	
Responsibilities	**Collaborators**
Maintain data about copies currently borrowed	
Meet requests to borrow and return copies	`Copy`

Copy	
Responsibilities	**Collaborators**
Maintain data about a particular copy of a book	
Inform corresponding `Book` when borrowed and returned	`Book`

Book	
Responsibilities	**Collaborators**
Maintain data about one book	
Know whether there are borrowable copies	

Questions: Do these three classes conform to our notion of a good design? What is their level of cohesion and coupling?

CRC Card of a Bad Object Class

- Here is an example of a CRC card for a bad object class

Word_Processor_Object	
Responsibilities	**Collaborators**
1. Spellcheck the document	Dictionary
2. Print the document	Printer
3. Open a new document	File I/O
4. Save the document	Networking API
5. Email document	

Why is this a bad class according to the principals of good design we have identified?

How can it be improved?

Lecture Key Points

- Sequence diagrams add detail to the use case, by showing the workflow and the objects involved
- State diagrams show how an objects state can be changed by messages received
- Class diagrams represent the static (as opposed to the dynamic) nature of the system to be built.
- We discussed how classes and their associations can be found and the concept of multiplicity.
- We discussed class attributes and operations
- We looked at representations of generalization.
- We learned about CRC cards