Teaching London Computing

A Level Computer Science

Topic 8: Software Development & Inheritance

William Marsh
School of Electronic Engineering and Computer Science
Queen Mary University of London

Aims

- Review software development lifecycle
- Introduce
  - Structure modelling
  - State modelling
  - Object modelling
- Introducing inheritance in modelling and Python

Software Lifecycles and Modelling

Are Lifecycles Important?

- Comments on mark breakdown table on the AQA practical project

The table does not imply that students are expected to follow a traditional systems life cycle approach when working on their projects, whereby a preceding stage must be completed before the next can be tackled. It is recognised that this approach is unsuited to the vast majority of project work, and that project development is likely to be an iterative process, with earlier parts of the project being revisited as a result of discoveries made in later parts. Students should be encouraged to start prototyping and writing code early on in the project process.

OCR Draft Curriculum

- 1.2.3 Software Development

a. Understand the waterfall lifecycle, agile methodologies, extreme programming, the spiral **model** and rapid application development.
b. Writing and following algorithms.
c. The relative merits and drawbacks of different methodologies and when they might be used.

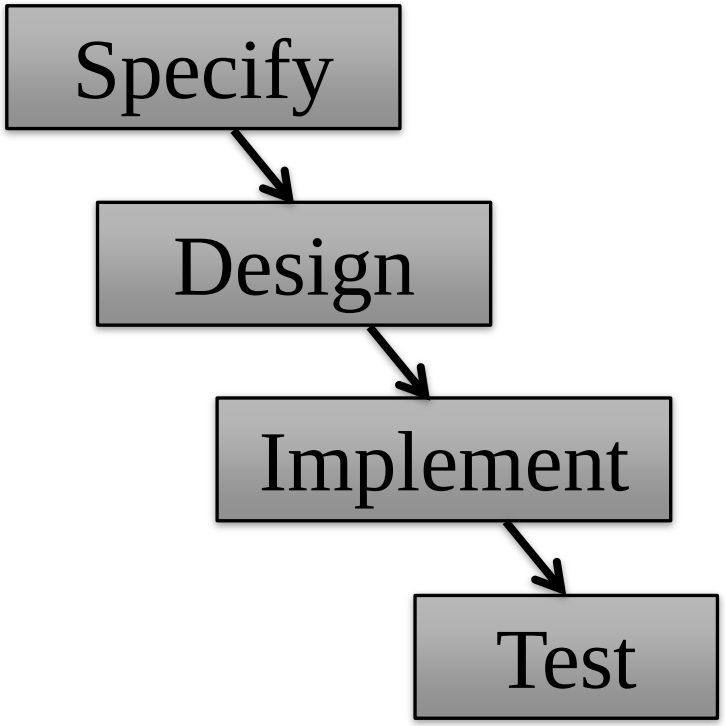
Hacking

- Just write a program
- Linus Torvald – author of Linux

- Good approach for very good programmers
- BUT
  - Not repeatable (c.f. factory)
  - Does not work with teams
- **Problem**: how to organise a large project

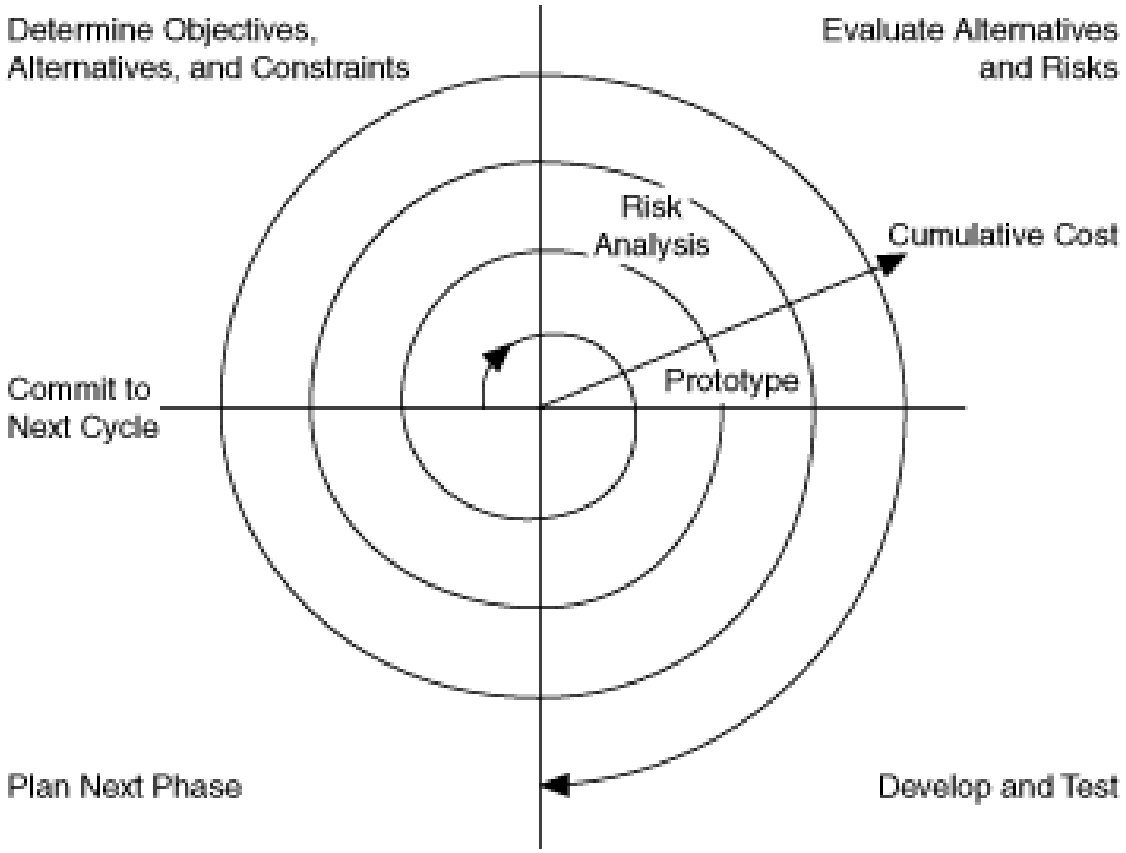
Waterfall Life Cycle Model

- Key ideas
  - S/W development stages
  - Sequence: complete in order
  - Client knows requirements
  - Documents
- Critique
  - **Design problems: separate parts** of implementation don't work together
  - **Requirements problems**: client not satisfied at end
  - … illusion of progress



Spiral & Prototypes: Insights

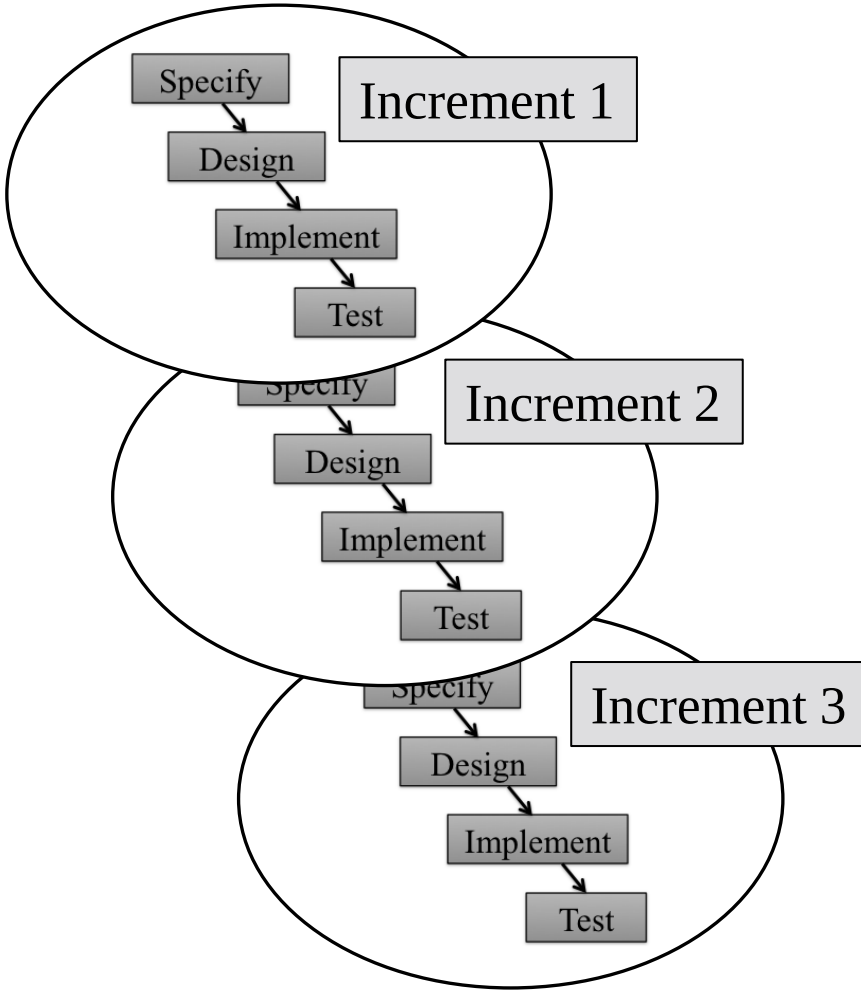
- Implementing helps to understand a problem
  - *Implement*
  - *Refine*
- Prototypes: users give feedback on working systems
  - *Show them a prototype*



Agile, Extreme, RAD

- Lots of books; written by ‘gurus’

- Emphasises
  - Incremental development (iterations)
  - Early working systems
  - User feedback
  - Automated testing



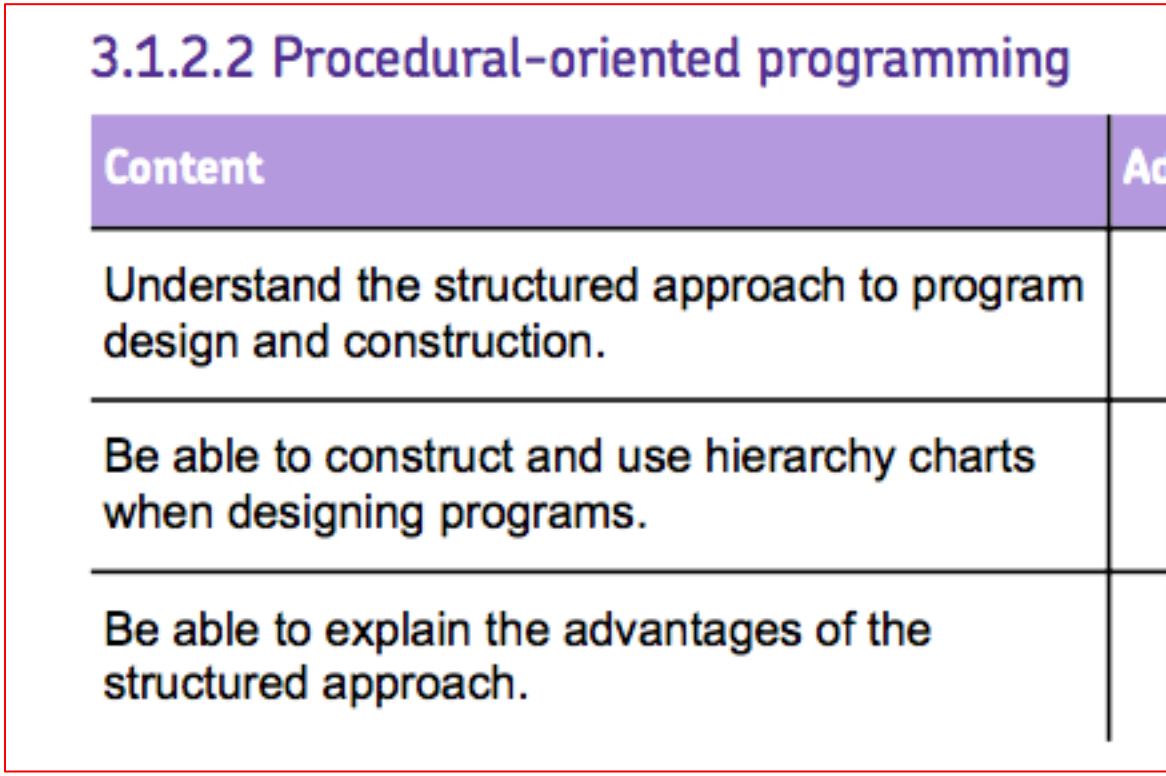
Summary

- Modern lifecycles give less emphasis to **a specification**
- Models
  - Still important
  - Abstract view of problem or solution
- Pseudo code is a model but not very abstract
- Flow charts are abstract compared with assembly code but not with a modern high level language

Structure Modelling

Structured Programming

- Early (1970s) and important idea
  - Top-down decomposition into functions
  - Abstract view of data
  - Became OOP
- Unstructured
  - Use of gotos
  - Global data



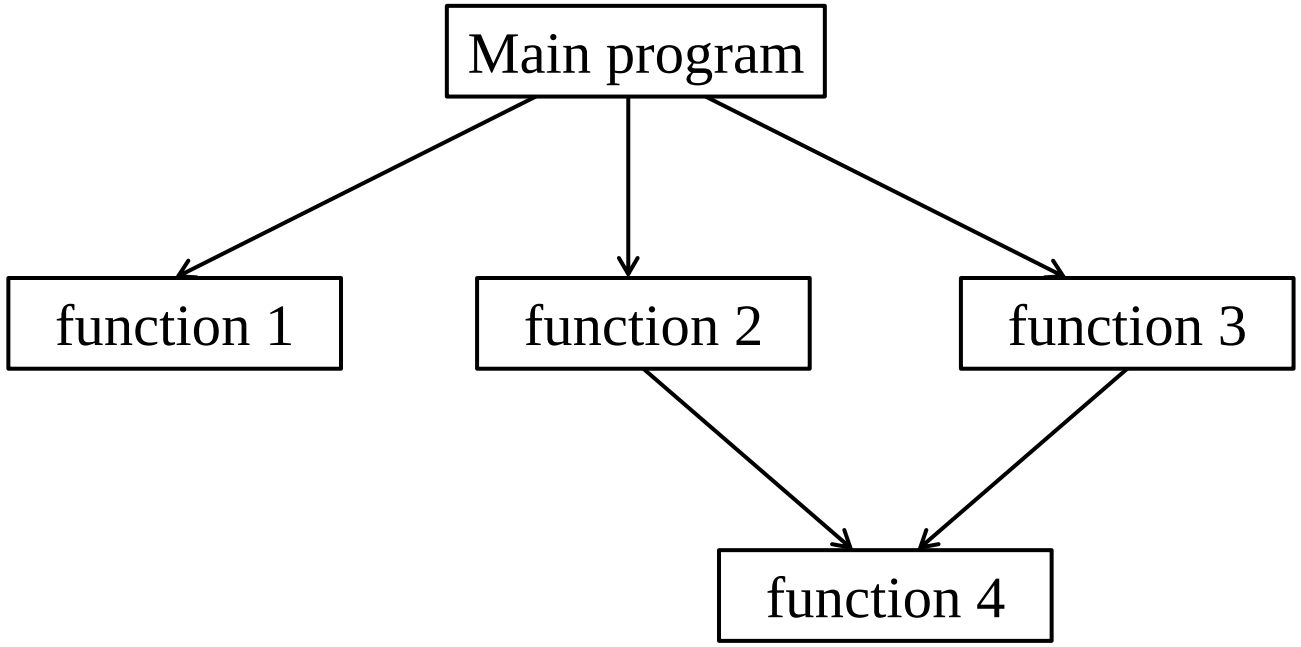
3.1.2.2 Procedural-oriented programming

Content	Ac
Understand the structured approach to program design and construction.	
Be able to construct and use hierarchy charts when designing programs.	
Be able to explain the advantages of the structured approach.	

Global data: hard to see where it is used. Therefore hard to change program

Call Hierarchy

- Shows one function calling another



Exercises 2.1, 2.2

- A simple version of the minesweeper game is available. Download it and review the code. Draw a function-call structure chart.

- A very simple version of '20 Questions' randomly selects from a fixed set of animals (for example). The player can see the answers of 3 questions (e.g. *how many legs?*) chosen from 5. The player then guesses the animal.

  Suggest a functional decomposition for this problem.

State Modelling

States

A state is a condition during the life of an object ... during which it satisfies some condition, performs some action or waits for some event.

- States should
  - have a name
  - be mutually exclusive
  - cover all the possibilities
- Possible states of a bank account:

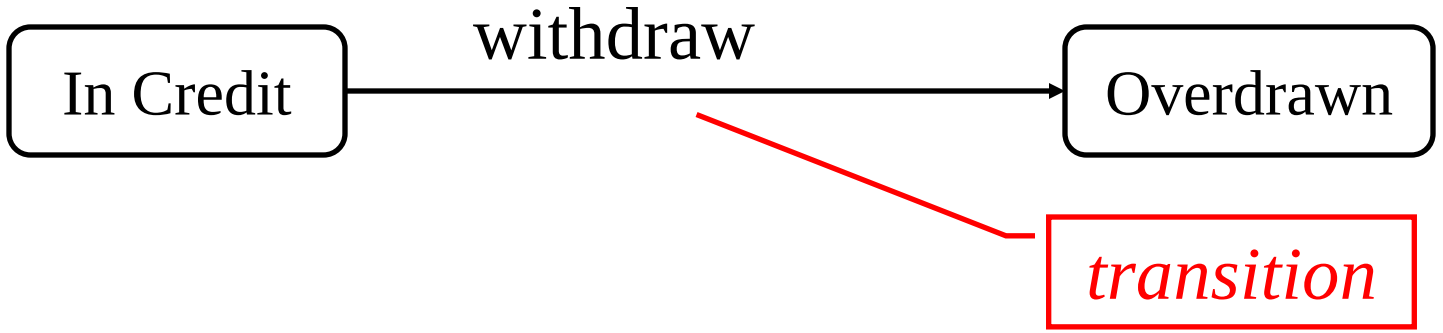
In Credit | Overdrawn | Closed

Events

- Events cause a change of the state of an object
- Event for a telephone
  - ring
  - answer
  - hang up
  - switch off
- A change of state is called a **transition**

Events

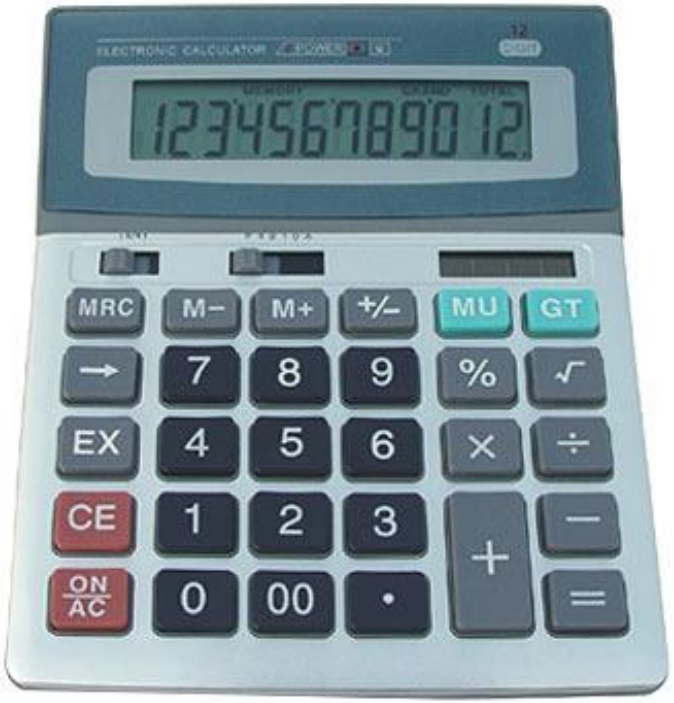
- Events causes a transition between states



- Some events are allowed only when an object is in certain states
  - e.g. can’t withdraw money from an overdrawn account

Exercise 3.1

- Draw a state-transition model of a simple calculator
  - How much detail?



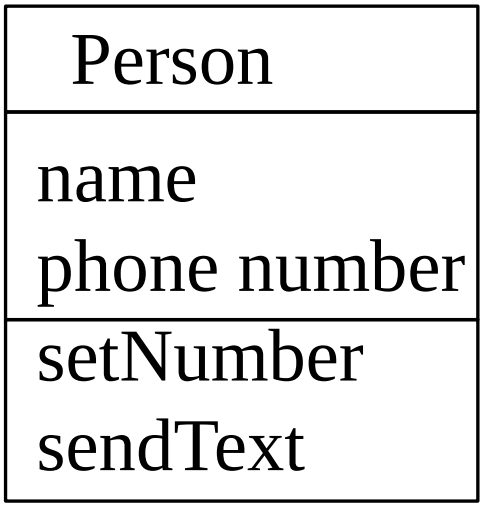
Exercise 3.2

- In minesweeper, a square in the minefield
  - Can hide a mine or no mine
  - The user can
    - Flag (claim a mine is present)
    - Test (claim no mine is present)
- Draw a state model of this. What transitions are needed?
- How is the game ended?

Object Modelling and Inheritance

Pictures of Classes

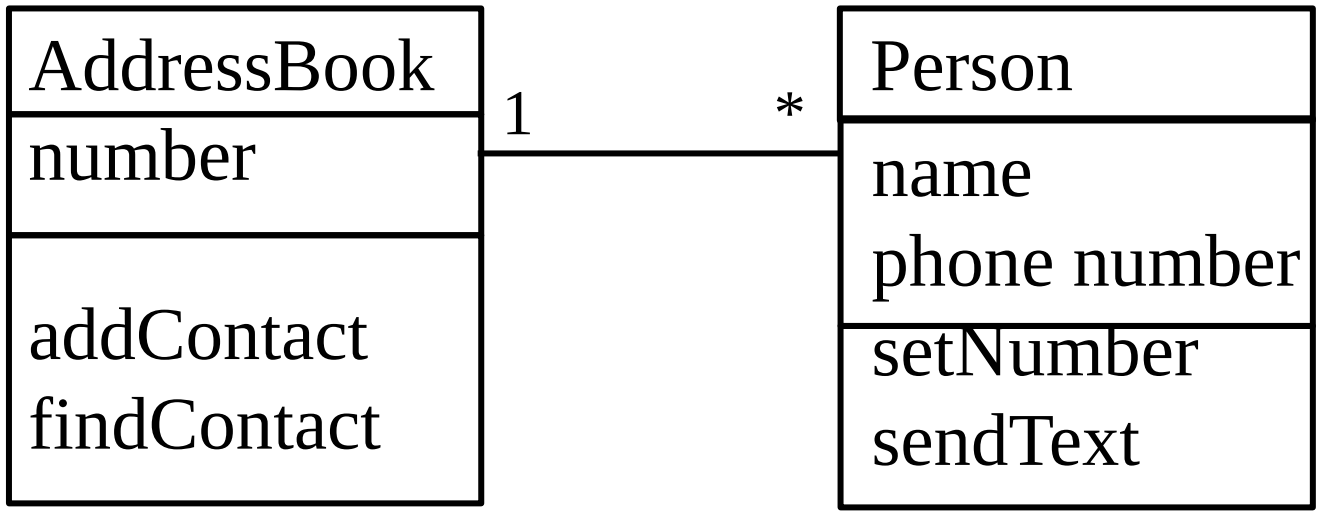
- Classes
  - Thinking about which classes
  - Attributes
  - Methods



- Relationships between classes
  - How are classes related?

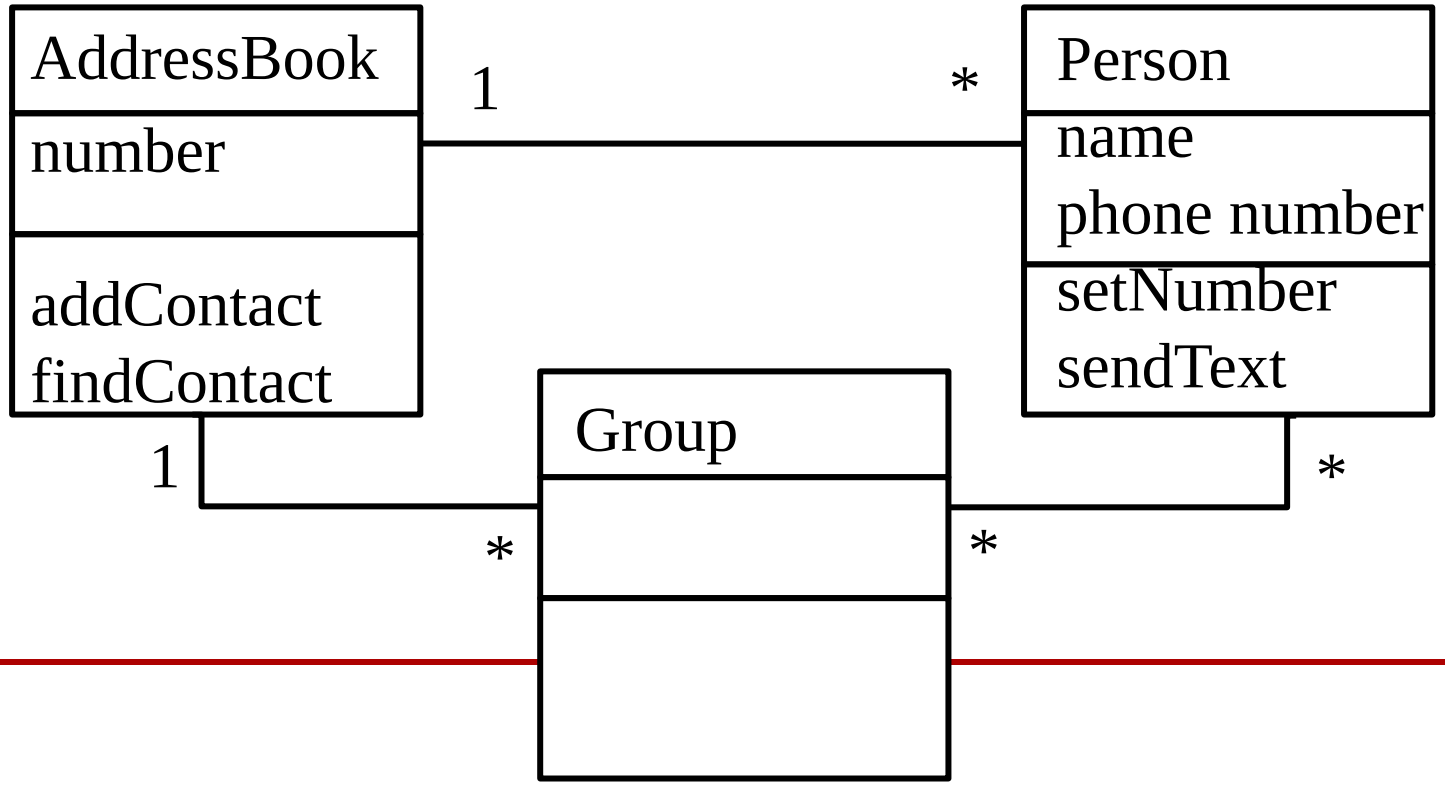
How Are Classes Related?

- An AddressBook contains many Persons



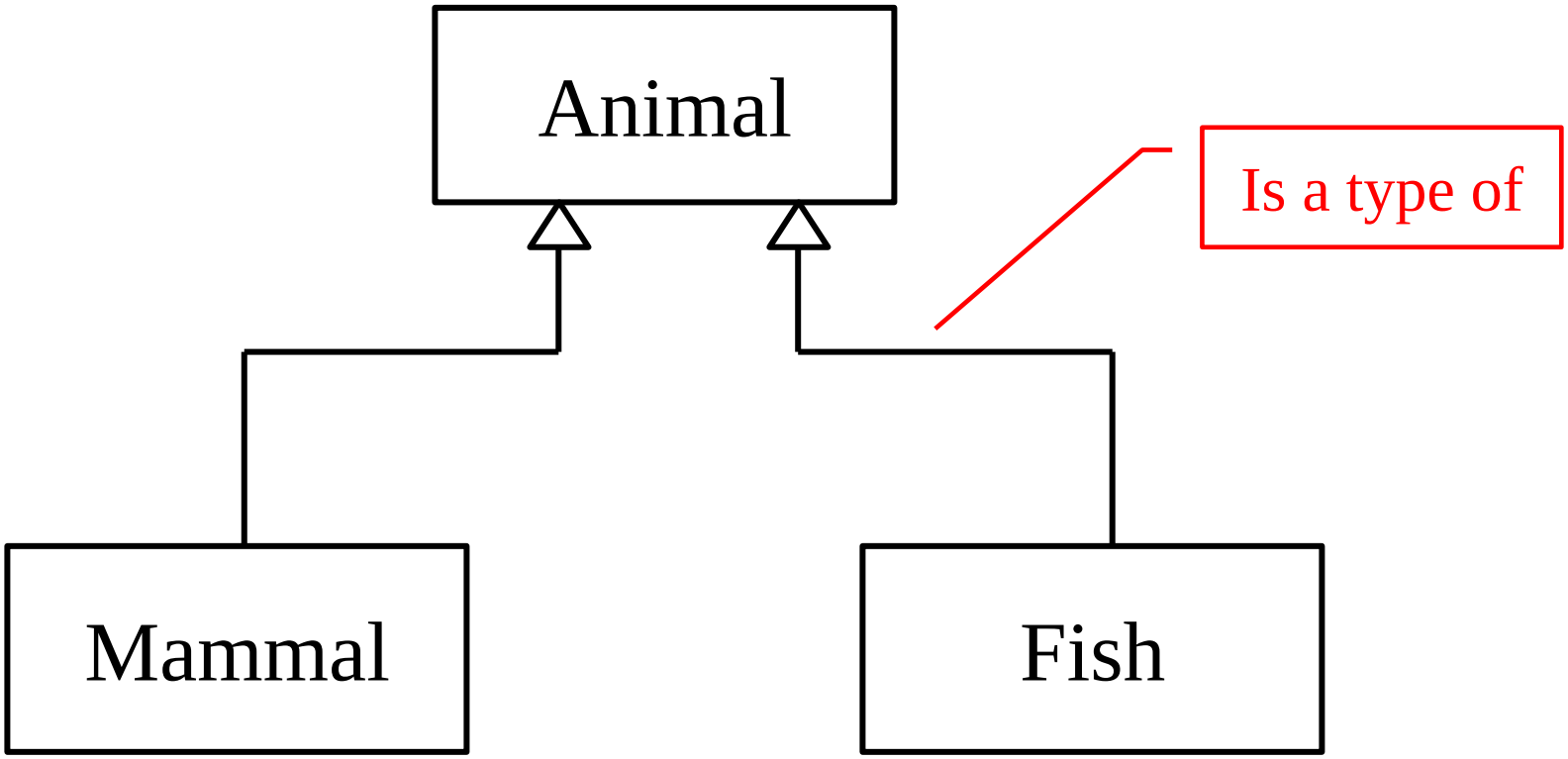
Knows-About and Has-A

- An AddressBook contains many Persons
- The AddressBook contains groups
- People are in groups



Inheritance – Classification

- Classify in a hierarchy

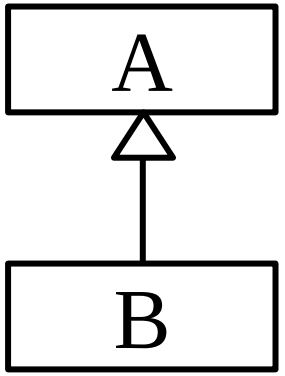


Language of Inheritance

- Fish is a **sub-class** of Animal
- Fish **is a** Animal
- Fish **inherits** from Animal
  - Methods are inherited
  - Attributes are inherited
- Animal **is the parent** of Fish
- Fish **is a child of** Animal

Inheritance and Overriding

- If A has a method M1, then B inherits M1
- B can have another method M2, not in A
- If A has a method M3, B can define (**override**) M3 so it behaves differently



Exercises 4.1

- Sketch an Object Model for the minesweeper game

- The school has a directory listing students and staff. Different information is held about the different categories of people. Discuss how inheritance can model this.

Inheritance In Python

Inheritance in Python

```
class Mammal:

    def __init__(self, n):
        self.name = n
        self.legs = 0

    def setLegs(self, l):
        self.legs = l

    def numLegs(self):
        return self.legs

    def hasFur(self):
        return True

    def canSwim(self):
        return False
```

```
class Dog(Mammal):

    def __init__(self):
        super().__init__("dog")
        self.setLegs(4)

    def canSwim(self):
        return True
```

Inherits

Superclass constructor

Overrides

Exercises 5.1, 5.2

- Enter the classes Mammal and Dog from the slide
  - Create a Dog. Find out if the dog has fur, can swim and see
  - Add one or more new subclasses, such as: Dolphin, Bat or Cow
- Create some classes to draw shapes in turtle graphics (see sheet)

Is Inheritance Useful?

- Very in Graphical User Interface libraries
  - Much behaviour inherited
  - Override when special behaviour needed

- In **many** programs, ‘knows about’ relations are more useful

Summary

- Changing attitudes to lifecycles
- Different types of modelling
- Structure:
  - Easy to use but limited
  - Encourages structured programming
- Object and state modelling
  - Both analysis and design