Bournemouth University

software systems research centre

Software Systems Modelling: Classes and Objects: The abstract perspective

Dr Keith Phalp

UML - Pure Class

A class diagram shows a set of classes, interfaces, collaborations and their relationships.

Class diagrams are the most common diagram type you'll use to model OO systems.

One purpose of the class diagram is to define a foundation for other diagrams where other aspects of the system are shown.

Class diagrams only model the static **design** view of the system.

Bournemouth University

Who stole the Analysis Model (1)?

software systems research centre

UML diagrams concerned with design:

Class diagram, Object diagram.

UML diagrams concerned with implementation:

Component diagram.

UML diagrams concerned with architectural deployment:

Deployment diagram.

Bournemouth University

Who stole the Analysis Model (2)?

software systems research centre

Diagrams concerned with dynamics of the system:

Sequence diagram, Collaboration diagram,

Activity diagram, Statechart diagram.

And Use Case Diagrams? Well, they're concerned with the 'Use Case View' of the system - whatever that means.

We can say: the UML does not support an Analysis model of the system.

Bournemouth University

So how do we analyse with the UML?

ssrc software systems research centre

We can develop a business class model

We can do design and pretend it's analysis

Or...

We ignore the fact that UML does not explicitly support analysis and do analysis anyway.

(Without adding any new notation - it's all a question of abstraction, description and context.)

Bournemouth University

Fowler's Perspectives (1)

software systems research centre

Martin Fowler identifies 3 perspectives of use for UML class diagrams

Conceptual

- similar to C & Y OOA

Specification

- concerned with interfaces of classes

Implementation

- the implementation code is apparent

Perspectives (2)

Conceptual

Diagram represents the concepts in the domain under study. No regard to software or implementation should be made. The concepts relate to the classes that implement them but there is no concern for code. Often called a “Business Model”, this equates to a typical OO analysis diagram of the problem domain.

Bournemouth University

Perspectives (3)

software systems research centre

Specification

a fundamental principle of OO that is often ignored is that of interface. The difference between implementation and interface is usually blurred.

To have a class diagram that deals with interface only is of importance when we discuss class responsibility. (CRC cards are useful for this too.)

Implementation

We show all the classes with all their implementation. Often the specification diagram is better but you'll see this more.

Classes and Objects

- A class is a set of objects that share a common structure and behaviour.
  - Egbert the Aardvark belongs to the class 'Aardvark'.
- A class is a blueprint of state and behaviour from which objects are instantiated
  - E.g., Keith is an object of class ‘Unknown Alien Species’.

Object Oriented Analysis

- OOA is all about viewing a solution in terms of Classes & Objects.
- It is argued that '...by thinking about systems in an object-oriented way we can...'
  - Begin to understand the problem domain.
  - Give the system an intuitive structure.

Finding Classes (and objects)

- **Object -** An abstraction of something in a problem domain, reflecting the capabilities of a system to keep information about or interact with it; an encapsulation of Attribute values and their exclusive Services.
- **Class -** A description of one or more Objects with a uniform set of Attributes and Services, including a description of how to create new objects in the Class.
- **Class-&-Object -** A term meaning “a Class and the Objects in that Class.”

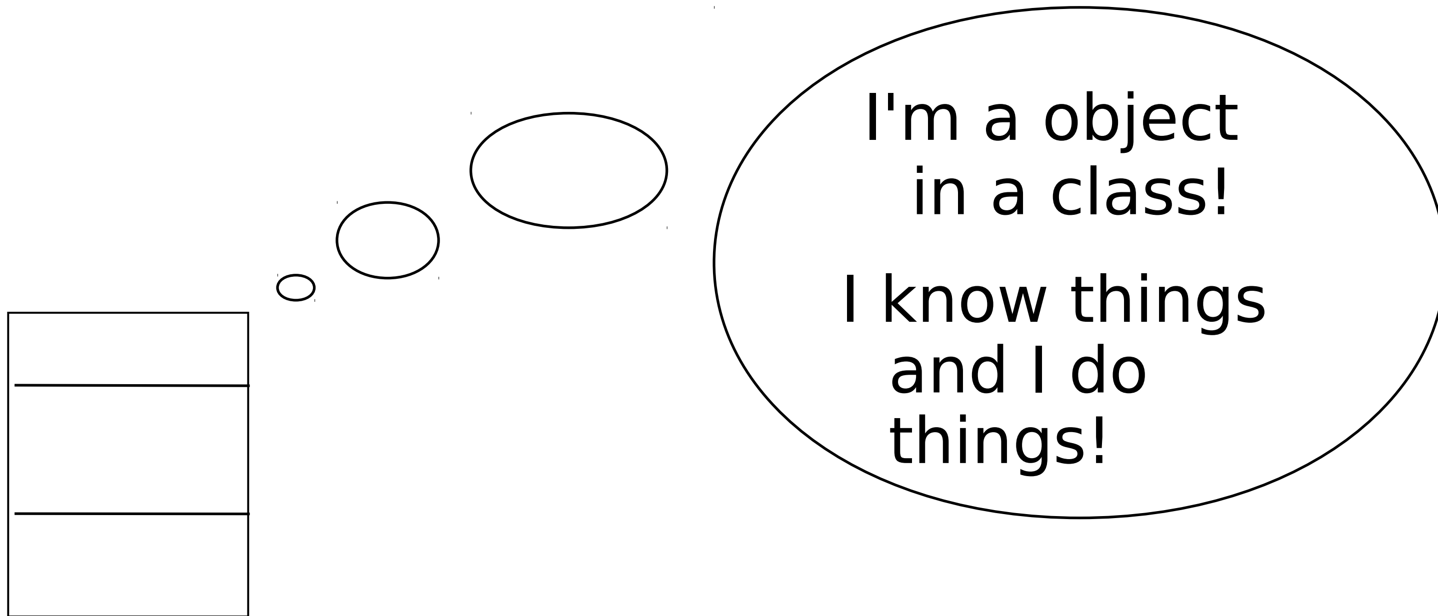
Finding Classes (and objects) 2

- Roles Played e.g., Bank Customer.
- Things or Events Remembered e.g., Sensor Log.
- Devices e.g., Sensor , Robot Arm.
- Operational Procedures e.g., Calculate Interest.
- Sites e.g., Launch Pad, Tube Station.
- Other Systems.
- Structure - inheritance and ‘part-of’ hierarchies

Roles

- The OOA view is: What role or roles do human beings play in the system?. Two types of role.
  - A user who interacts with the system. E.g., Pilot.
  - A Person who does not interact directly with the system but about whom information is kept. E.g., Customer Financial Record.
- With our strategic (business) modelling we

Anthropomorphism



How to name Classes

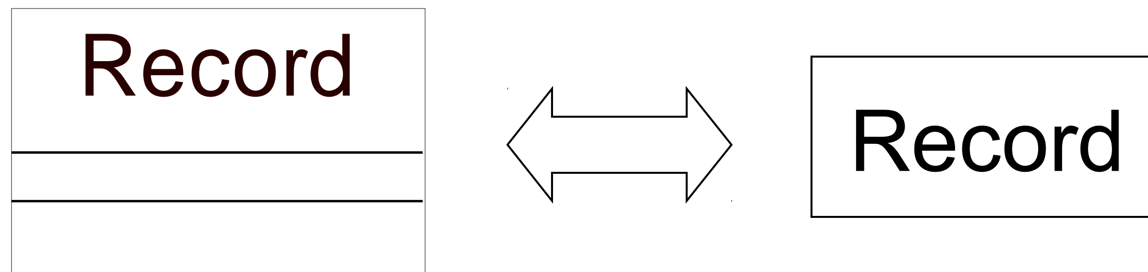
- Use a singular noun or adjective & noun
- Describe a single Object in the Class
- Adhere to the standard vocabulary for the problem domain.

Bournemouth University

Where to look

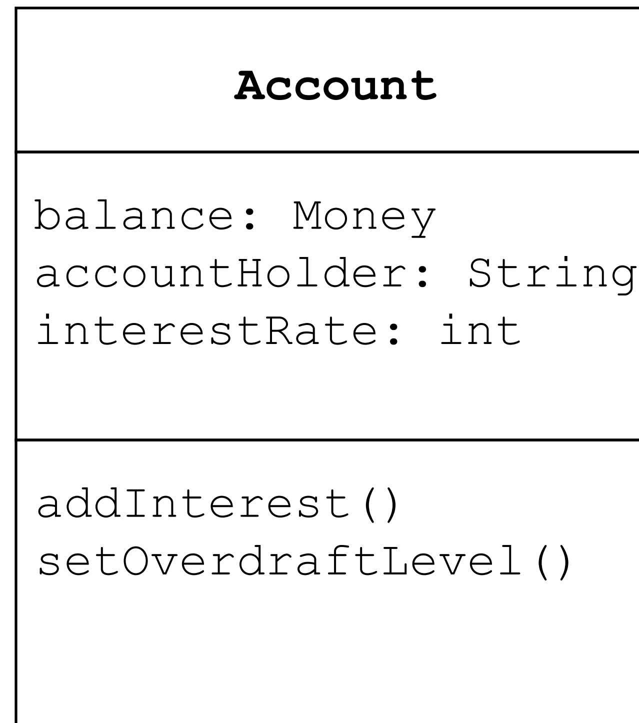
software systems research centre

- Observe first-hand
- Listen actively
- Check previous OOA results (REUSE!)
- Check other systems
- Read
- Prototype.



A simple UML class. Note that attributes and operations are left empty here - they can be filled as you want, dependent on the perspective you wish to model.

Simple class syntax



Class name compartment

Attributes compartment

Operations compartment

"A class is a description of a set of objects that share the same attributes, operations relationships, and semantics."

Bournemouth University

software systems research centre

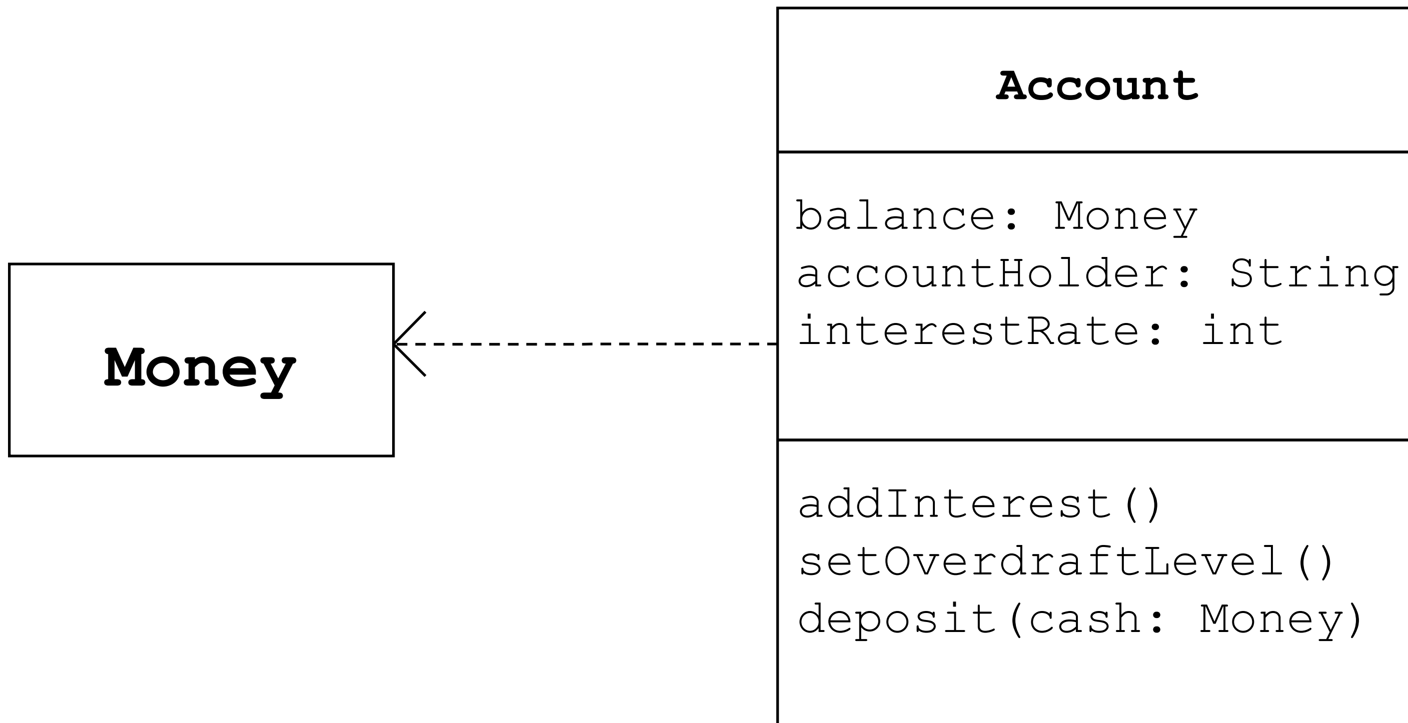
Connecting classes

There are 4 types of connections in UML

- **Dependencies** which are using relationships.
- **Generalizations** which are inheritance relationships.
- **Associations** which are structural relationships.
- **Aggregation**, which shows whole-part relationships.

Dependencies

A dependency shows that one class uses another. A change in one will affect the other.



Inheritance

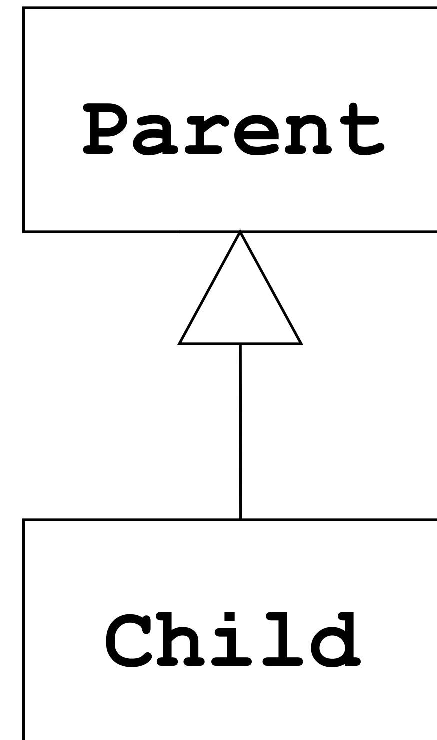
- Defines an ‘Is a’ relationship between classes.
- A class may inherit services and attributes from other classes
  - A Bird squawks.
  - A Penguin squawks and swims.
  - A Penguin is a bird.

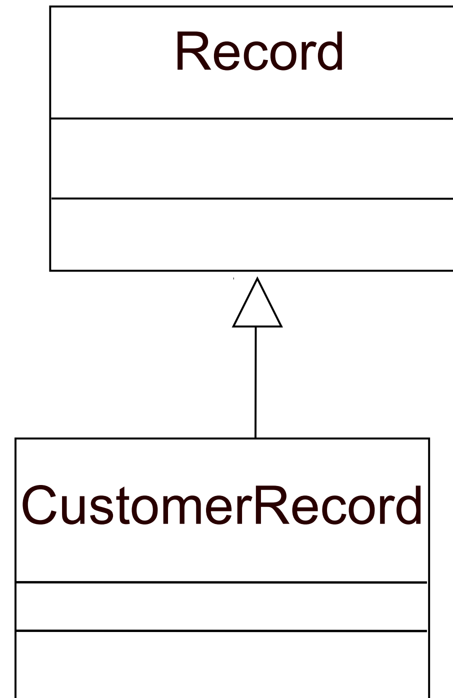
Generalization

"Generalization

implies

substitutability"





CustomerRecord is a specialisation of Record.

The triangle arrow head indicates that Record is the parent and CustomerRecord the child.

Inheritance 2

Superclass

Bird Class has a beak length and squawks.

Subclass

Penguin Class has a fin colour and swims.

Penguin inherits from bird so it also has a beak length and squawks

A Penguin 'is a' Bird

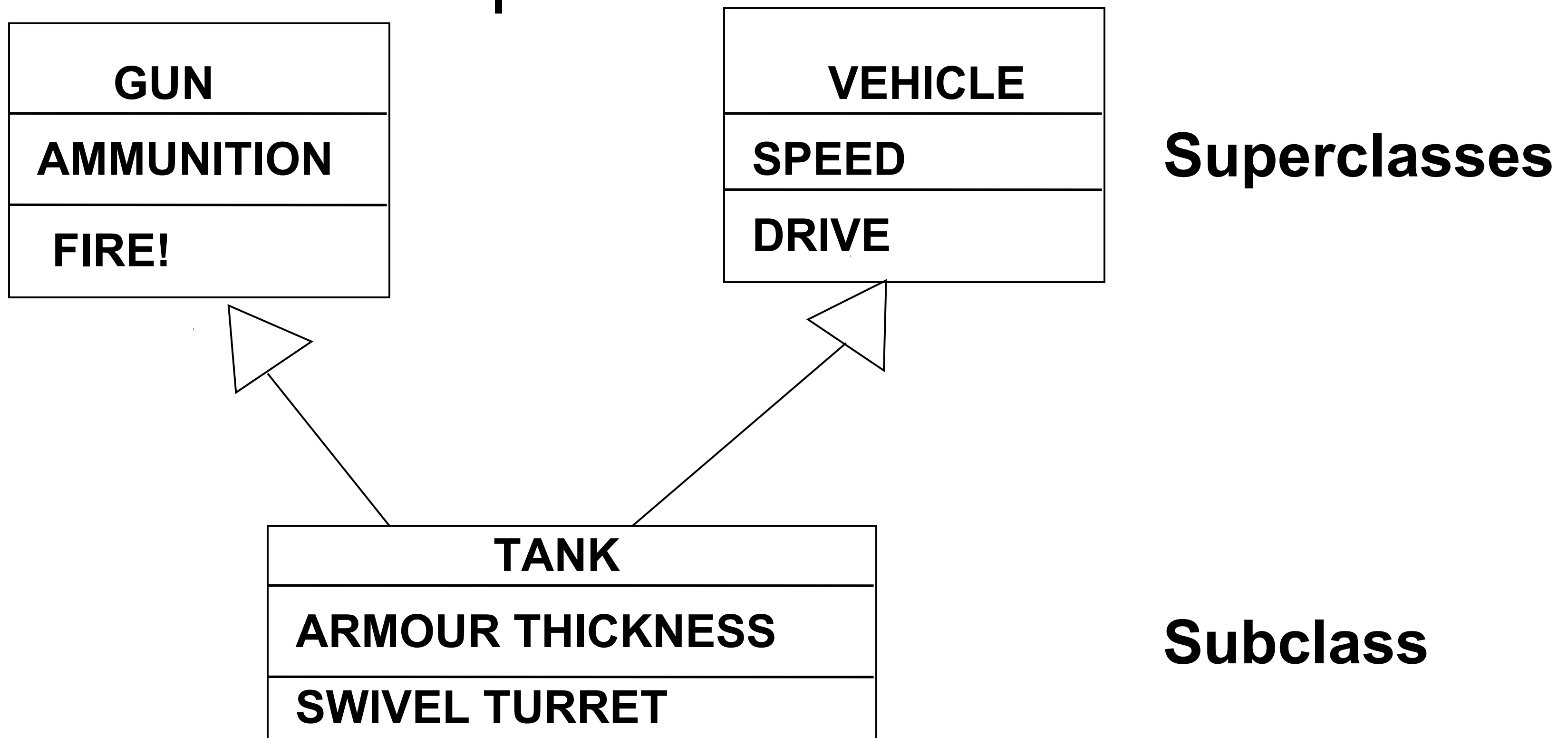
Inheritance 3

- Inheritance is a Generalisation-Specialisation relationship
  - A ‘Penguin’ Class is a specialisation of ‘Bird’.
- Inheritance is a hierarchy where new classes may be built as specialisations of those already in existence.
- Inheritance is said to allow:
  - Greater understanding of the problem domain.

Reuse: of existing classes

Multiple Inheritance

- A subclass may be derived from more than one superclass.

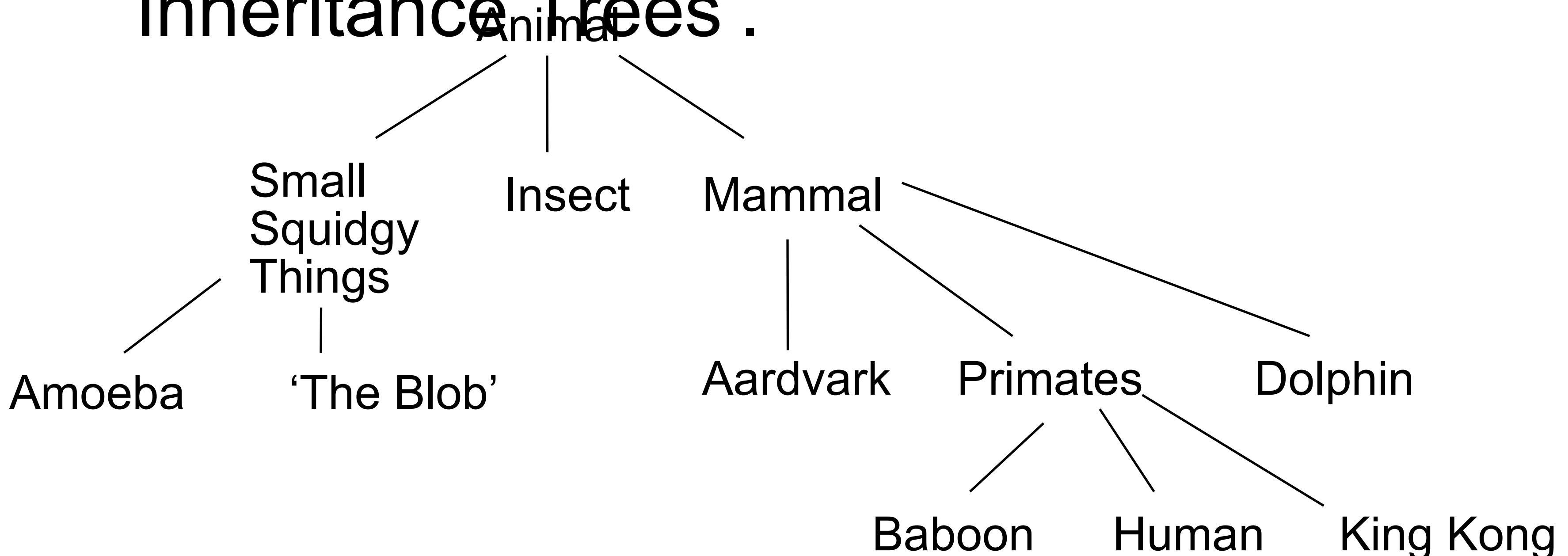


Bournemouth University

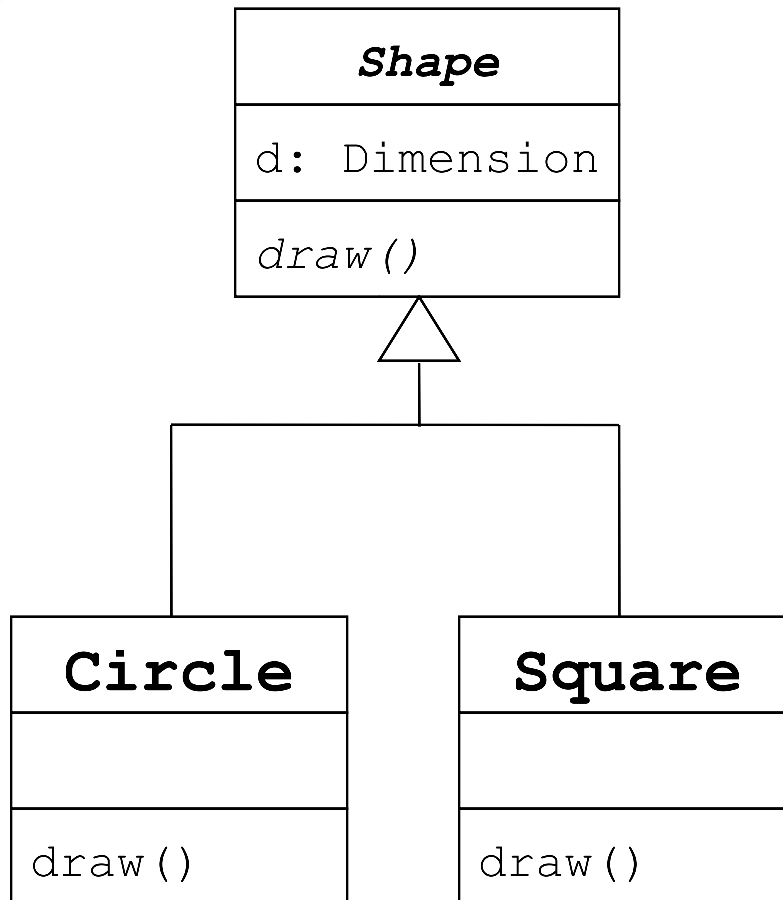
software systems research centre

Class Hierarchies

- Hierarchies of Classes inheriting from each other, sometimes called 'Inheritance Trees'.



Abstract classes

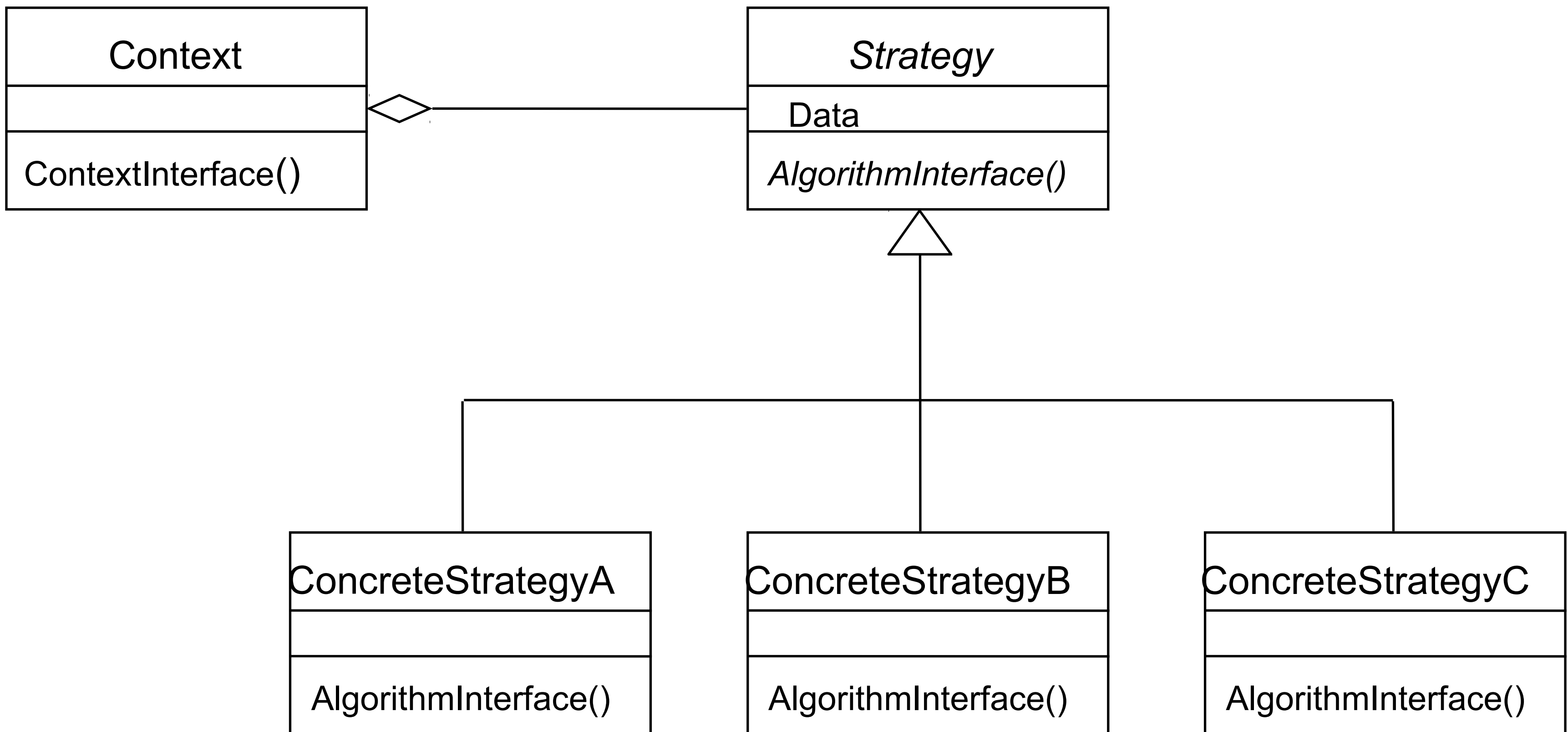


{abstract}
Shape

Bournemouth University

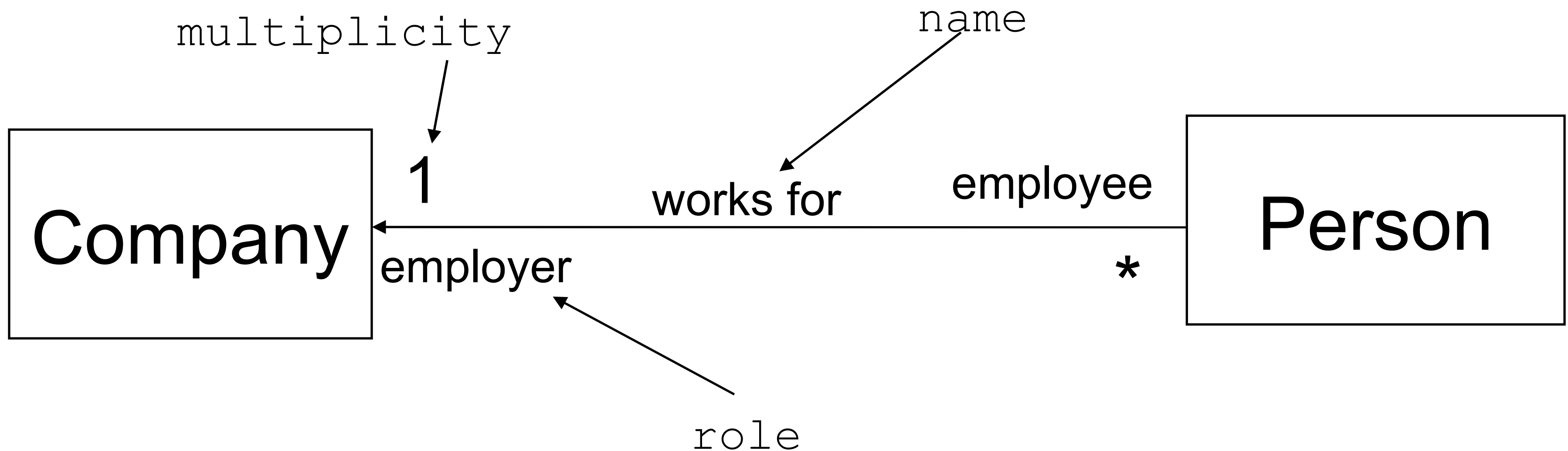
A simple Class Diagram showing an abstract class

software systems research centre



Bournemouth University

Associations



Associations are implicitly bidirectional but the direction arrow '→' makes association unidirectional - it shows navigability

Bournemouth University

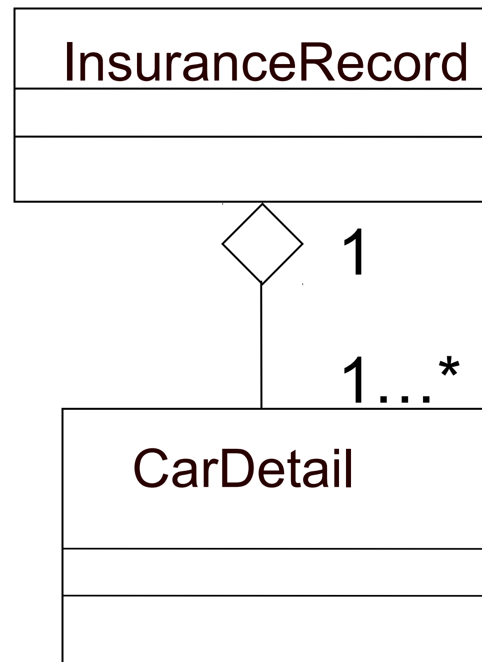
software systems research centre

Whole-Part

- A class & objects may be composed of other classes & objects.
  - This is called a ‘whole-part’ relationship.
  - Allows compositional hierarchies.
  - E.g., a car is composed of an engine, seats, boot, wheels and so on.

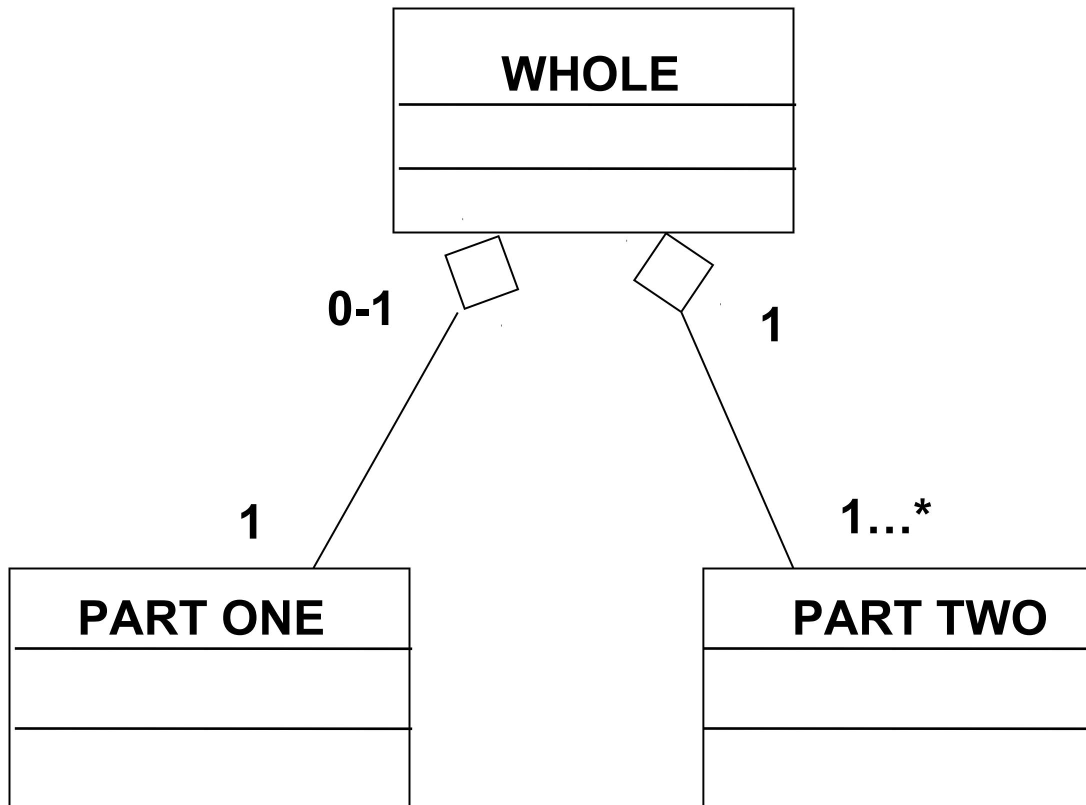
Composition

- Defines a ‘is part of’ relationship.
- Classes (and any objects they instantiate) may be nested inside other classes.
- The human body has two lung objects inside it
  - So the lung class & objects are ‘part-of’ the human body class.
  - As a consequence a human *object* will always have lung objects inside it (as the



The diamond shows the whole-part “direction” - CarDetail is part of InsuranceRecord. The cardinality shown indicates that for every InsuranceRecord there can be 1-to-many CarDetail classes.

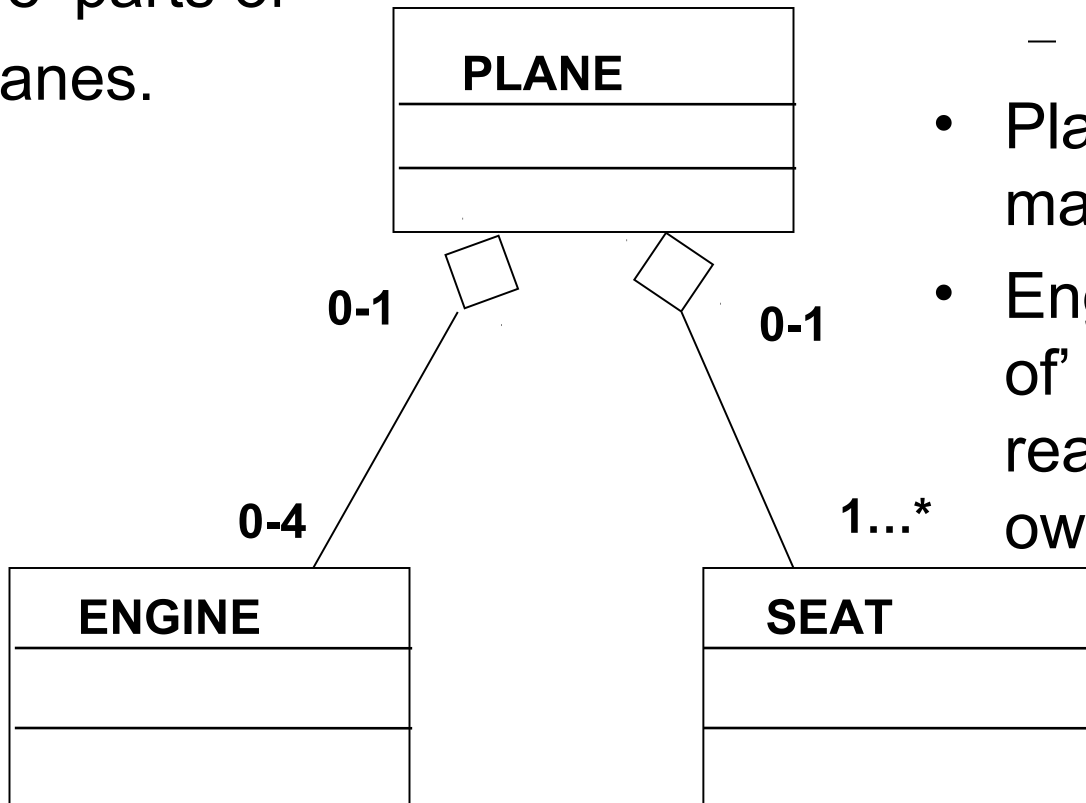
Notation & Guidelines



- Draw whole - part structures from top to bottom.
- Number above each part is a range.
- Number under whole states how many wholes there are at any one time.

Whole - Part Example

Engines and Seats are ‘parts of’ planes.



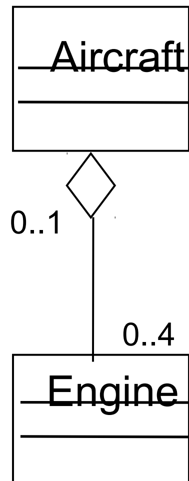
- A plane composed of 0 to 4 engines.
  - (0 being a glider!)
- Plane composed of 1 to many seats.
- Engine & seat may be ‘part of’ a single plane or reasoned about on their own.

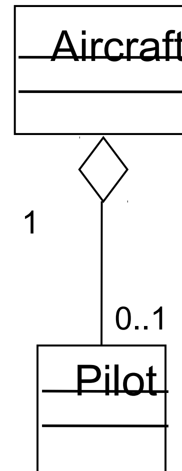
Kinds of Whole-Part Structure

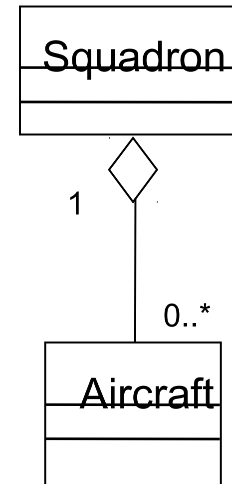
- Assembly of Parts
  - E.g., An aircraft is composed of engines.
- Container & Contents
  - E.g., A pilot sits inside an aircraft.
- Collection of Members
  - E.g., A squadron is composed of aircraft.

Another Whole-Part View

Assembly-parts **Container-contents** **Collection-members**







(kick one) (look inside) (observe)

Bournemouth University

software systems research centre

Whole-Part Advice

- Consider each Object as a whole. For potential parts, ask:
  - Is it in the problem domain?
  - Is it within the system’s responsibilities?
  - Does it capture just a status value?
    - If so, then just include a corresponding Attribute within the whole.
  - Does it provide a useful abstraction?
- Also, consider each Object as a part.

Bournemouth University

software systems research centre

Overview of Behaviour

- Behaviour is about the dynamic properties of a model or system.
  - Contrast with a static model.
- Behaviour in objects is brought about through ‘services’.
- Examine categories of service.
- Describe strategies for the identification and description of services.

Bournemouth University

software systems research centre

Services and Behaviour

- A Service is a specific behaviour that an Object is responsible for exhibiting.
  - Dobbin the Mule carries holidaymakers at Butlins.
  - Karl’s car drives forward (usually).
- Operations and methods are synonyms.
- ‘What I do’.

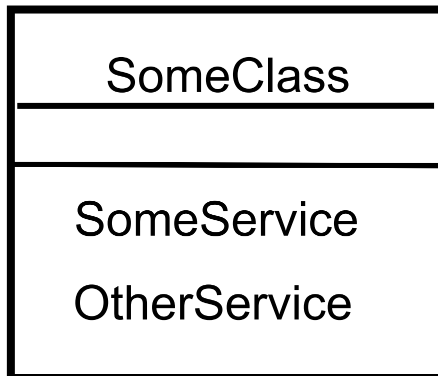
Bournemouth University

Services, States and Behaviour

software systems research centre

- An object is a dynamic entity - the values of its attributes collectively define a ‘state’.
- Changes of state are the result of the invocation of services.
  - (For roles it was the invocation of events).
- State changes can be captured through the use of STDs.
  - Other mechanisms?

Service Notation



Services are shown in the bottom section of the class symbol. Services may be considered to be either simple or complex

Algorithmically Simple Services

C	Create
R	Read
U	Update
D	Delete

All objects are given these CRUD services for free.

We only add them to the model in special cases.

Simple Service Categories

- Create: creates and initialises a new object.
- Read: gets (access) the value of an internal attribute.
- Update: sets the value of an attribute.
- Delete - disconnects (releases) an external object.

and

Bournemouth University

ssrc software systems research centre

Algorithmically Complex Services

- **Calculate**: calculates a result from the attribute values of an object.
- **Monitor**: monitors an external system or device.
  - Deals with external system inputs and outputs, or with device data acquisition and control.
  - May need some companion Services, such as Initialise or Terminate.

Bournemouth University

software systems research centre

Service Identification and Definition

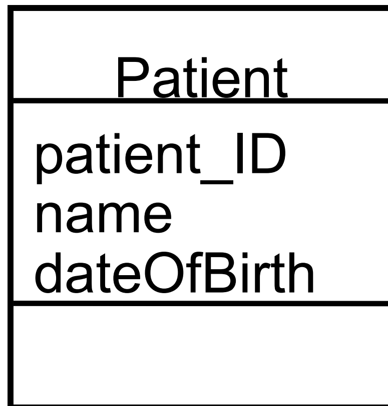
Two broad strategies

- Introspection - to see how an object changes state over time; Object State Diagrams.
  - Or similar.
- Interconnection - to see how an object relates with other objects; Messages and Scenarios.

Attributes

- Attribute: some data (state information) for which each Object in a Class has its own value.
  - A property, quality or characteristic.
  - 'What I know'.
- 'Any property, quality, or characteristic that can be ascribed to a person or thing'. [Webster, 1977].
  - E.g., Dobbin has a resting heart rate of 70 bpm

Attribute Notation



Attributes are shown in the centre section of the class symbol.

Identifying Attributes

- Anthropomorphic Questions:
  - How am I described in general?
  - How am I described in the problem domain?
  - How am I described in the context of the system's responsibilities?
- and then...
  - What do I need to know?
  - What state do I need to remember over time?

Sensible Attributes: Atomicity

- Make each attribute an ‘atomic concept’ - i.e., a single or tightly coupled set of values.

Good Values:

  - Name (‘Zargonn, Lord of the 9 Moons of Jubberwak’)
  - Address (‘9 Acacia Avenue, Surbiton, London’)

- Bad Values
  - The things a horse does in the evening

Bournemouth University

software systems research centre

Attribute Positioning

- Place attributes within the Class (& object) it best describes.
  - E.g., Current_Methane_Level would be best placed inside Methane_Sensor.
- In a Gen-Spec (Inheritance) Structure...
  - Attributes that are generally applicable to a number of subclasses should be in a superclass (rather than repeated).
  - Specialised attributes should be in a

Bournemouth University

software systems research centre

Attribute Verification

AIRCRAFT
ENGINE_TYPE
FLY

When is the attribute not applicable?

BANANA
FREIGHT_COMPANY
CONSUME

Is this attribute appropriately placed?

- Check for values which may not always be applicable.
- Single attribute classes?
  - Ask: is this a good abstraction?