COMP201 – Software Engineering I
Lecture 19 – Object Oriented Design

Lecturer: T. Carroll
Email: Thomas.Carroll2@Liverpool.ac.uk
Office: G.14
See Vital for all notes

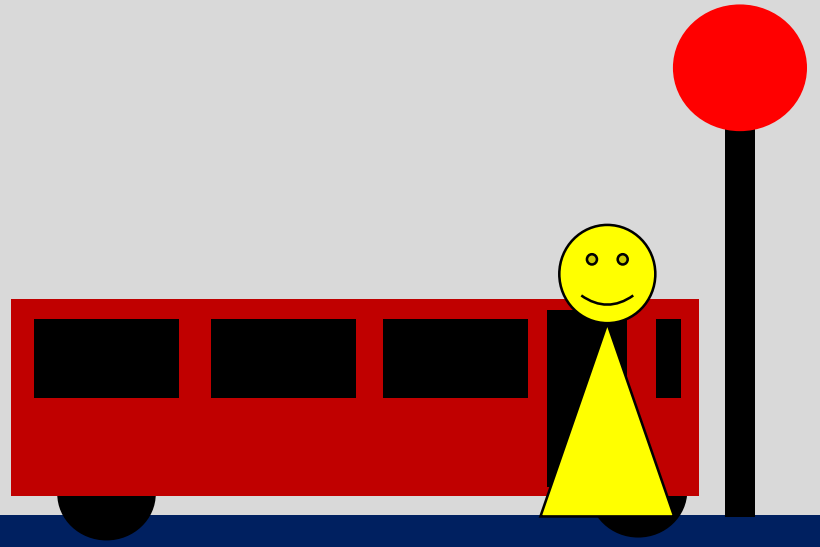
Today

Coming Up…

- We continue to explore the question “what are good systems like?” by describing the object oriented paradigm.
- We shall answer these questions:
  - What is an object?
  - How do objects communicate?
  - How is an object’s interface defined?
  - What have objects to do with components?
- Finally we consider inheritance, polymorphism and dynamic binding.

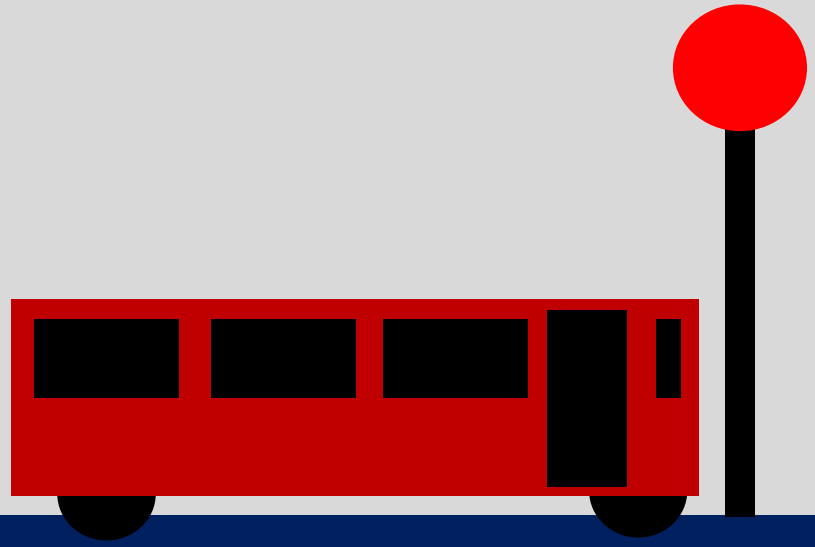
Object Orientation

- To organise the software as a collection of discrete objects
- Objects incorporate both **data structure** and **behaviour**
- System functionality is expressed in terms of object services



Object Orientation

- To organise the software as a collection of discrete objects
- Objects incorporate both **data structure** and **behaviour**
- System functionality is expressed in terms of object services



Object Concepts

- Don’t think of what an object holds – but what it will do for you
  - Consequently no public data members
  - Think only about the methods (the *public interface*)
- Objects are potentially reusable components.
- An object may represent something in the real world
  - But often not

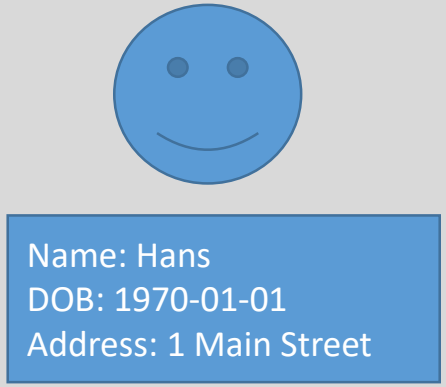
What is an Object?

- An object is a thing which has
  - behaviour,
  - state and
  - identity
- Advantages:
  - Shared data areas are eliminated (communicate by message passing)
  - Objects are independent
  - Leads to easier maintenance

Object State

All the data which it currently encapsulates

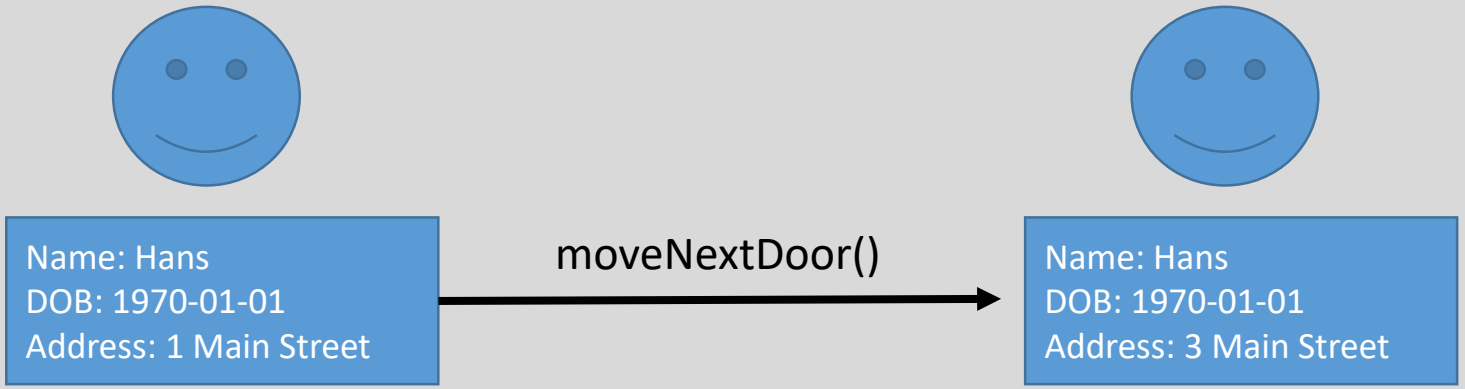
- An object normally has a number of named *attributes*
  - Some can be mutable (variable)
  - Some can be immutable (constant)



Object Behaviour

The way an object acts and reacts

- An object understands certain messages,
  - it can receive the message and act on it.
- The set of messages that the object understands is normally fixed.



Object Identity (a bit tricky…)

- Objects are not defined just by the current values of their attributes
- An object has a continuous existence
  - Values of the object's attributes could change
  - Still be the same object!

```
private Clock myClock = new Clock(12:00);
private Clock yourClock = new Clock(13:00);
myClock.setTime(13:00);
```

Are myClock and yourClock the same?

Messages

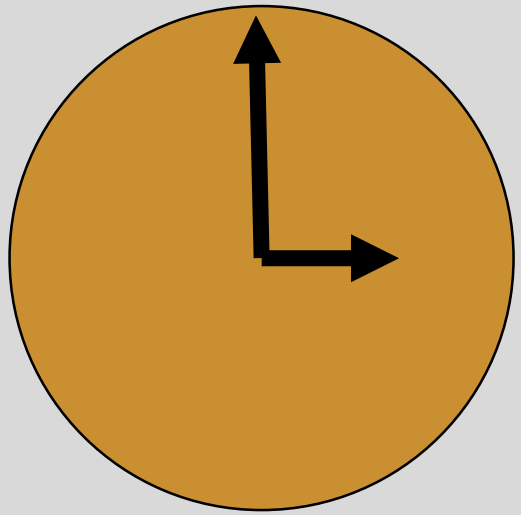
- A message includes a **selector**
  - Eg: setTime
- A message may include one or more **arguments**
  - Eg: setTime(Time theTime)
- Often, for a given selector there is a single “correct” number of arguments
- Sometimes, we can use overloading
  - Eg: setTime(Time theTime, Timezone theTimeZone, Boolean DST)

Interfaces

- The object's public interface defines which messages it will accept
- So typically an object has two interfaces:
  - The public interface (any part of the system can use)
  - The larger private interface (which the object itself and other privileged parts of the system can use)
- An object can send to itself any message which it is capable of understanding

Example

- Consider an object which we'll call myClock, which understands the messages:
  - getTime()
  - setTime (07:43)
  - setTime (12:30)
  - setTime (Time newTime)
- How does it implement this functionality?
- The outside world doesn't need to know



Maybe it has a "time" attribute?

Maybe it passes a message to another object?

- The information should be hidden

Object Classification

- Objects with the same data structure (attributes) and behaviour (operations) are grouped into a class
- Each class defines a possibly infinite set of objects
- Each **object** is an instance of a **class**
- Each object knows its class
- Each instance:
  - has its own value for each attribute (state)
  - shares the attribute names and operations with other instances of the class
  - shares "static" i.e. class variables
- A class encapsulates data and behaviour, *hiding the implementation details* of an object

Object Interface Specification

- Object interfaces have to be specified so that the objects and other components can be designed in parallel.
- Objects may have several interfaces which are viewpoints on the methods provided.
- Hiding information inside objects means that changes made to an object do not affect other objects in an unpredictable way.

Digression: Why have Classes?

- Why not just have objects, which have state, behaviour and identity as we require?
- Classes in object oriented languages serve two purposes:
  - Convenient way of describing a collection (a class) of objects which have the same properties
  - In most modern OO languages, classes are used in the same way that types are used in many other languages
    - To specify what values are acceptable

Classes and Types

- Often, people think of classes and types as being the same thing (indeed it is sometimes convenient and not misleading to do so). It is not strictly correct however.

- Remember that a class does not only define what messages an object understands!
- It also defines what the object does in response to the messages.

What have Objects to do with Components?

- The hype surrounding object orientation sometimes suggests that any class is automatically a reusable component.
- This is NOT TRUE!
- Reusability of a component is not a property of the component itself
- Reusability depends upon the *context* of development and proposed reuse.
- In order to reuse a single class you have
    - To be writing in the same programming language and
    - using a compatible architecture

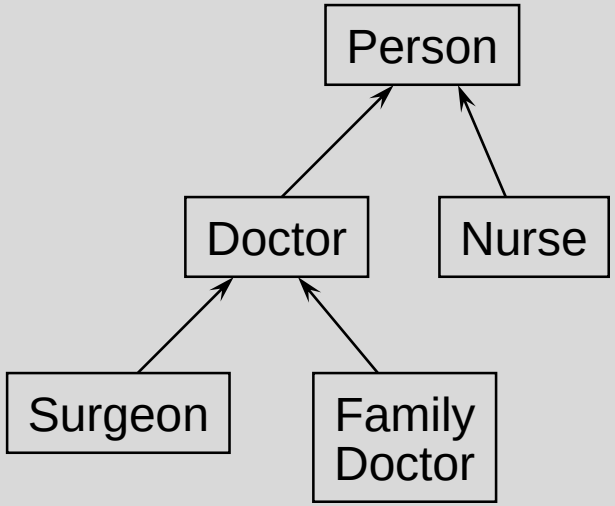
Object Inheritance

- Inheritance is the sharing of attributes and operations among classes based upon a hierarchical relationship
- A class can be defined broadly and then refined into successively finer subclasses
- Each subclass incorporates or inherits all of the properties of its super class and its own unique properties

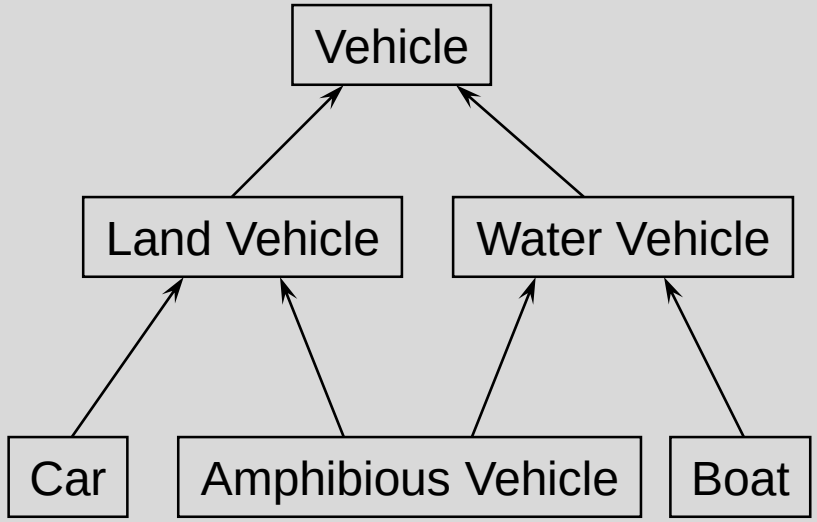
Subclass ←→ Superclass

- A **subclass** is an extended, specialized version of its **superclass**.
- It includes the operations and attributes of the superclass, and possibly extra ones which extend the class.

Object Inheritance



single



multiple

Inheritance - Warning

- One should not abuse inheritance
  - It is not just a way to be able to access some of the methods of the subclass
  - A subclass must inherit **all** the superclass
- Composition is often “better” than inheritance

- **An object class is coupled to its super-classes**.
- Changes made in a super-class propagate to all sub-classes.

Object Polymorphism

- Polymorphism allows the programmer to use a subclass anywhere that a superclass is expected.
- E.g., if Saloon is a subclass of type Car then if an argument expects a variable of type Car, it should also be able to take any variable of type Saloon.
- Polymorphism reduces the amount of code duplication required

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.
- Eg: C is a subclass of B and both have a method named printName(). Then if we write:

```
B temp = new C(); // (This is allowed by polymorphism)
temp.printName();
```

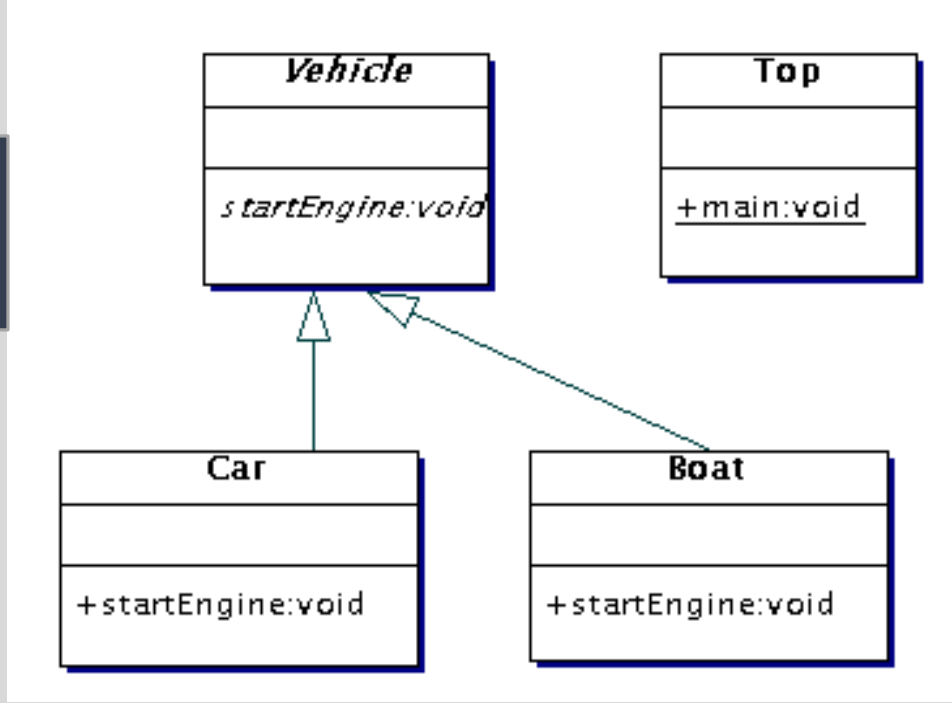
  we would invoke the printName() method of object C, **not** of object B
- This is ***dynamic binding***

Dynamic Binding - Example

```
Vehicle v = null;
v = new Car();
v.startEngine();
v = new Boat();
v.startEngine();
```

Call Car startEngine() method

Call Boat startEngine() method



Unified Modelling Language(UML)

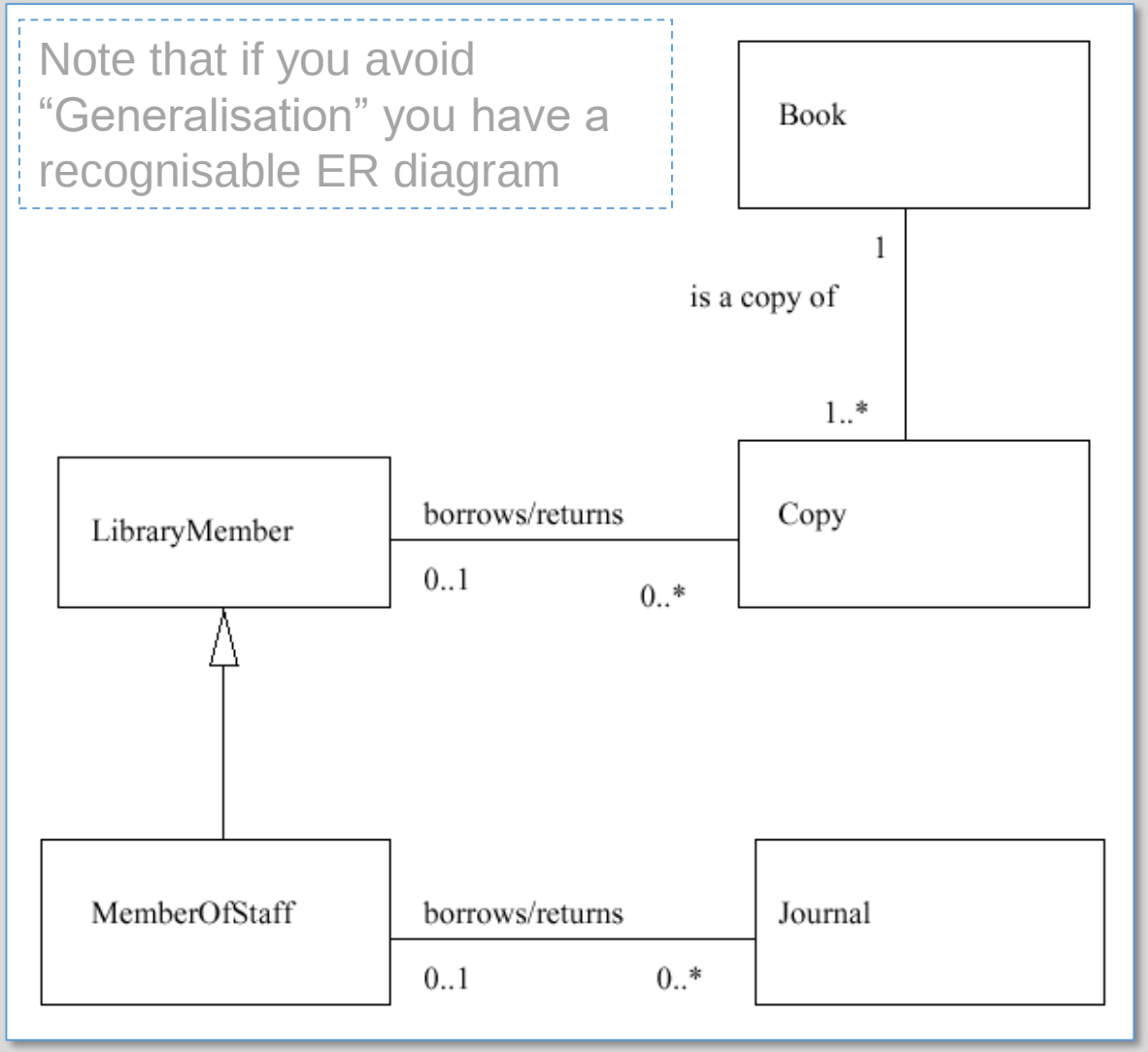
- Unify notations
- UML is a language for:
  - Specifying
  - Visualizing and
  - Documenting

  the parts of a system under development
- UML has been adopted by the Object Management Group (OMG) as an Object-Oriented notation standard

UML – Some Notation

- **Object classes** are rectangles with:
  - the name at the top
  - attributes in the middle compartment
  - operations in the bottom ompartment.
- Associations between object classes are shown as lines linking objects
- Inheritance is referred to as generalisation.
  - It uses an *open triangular arrow* head

Library System UML Example



CASE Tools/Workbenches

- Computer Aided Software Engineering (CASE)
- A coherent set of tools to support a software engineering method
- These workbenches may:
  - support a specific SE
  - provide support for creating several different types of system model.

CASE Tool Components

- Diagram editors
- Model analysis and checking tools
- Repository and associated query language
- Data dictionary
- Report definition and generation tools
- Forms definition tools
- Code generation tools
- Import/export translators

Key Points

- Object Oriented Design is an approach to design so that design components have their own private state and operations.
- Objects should have a constructor as well as inspection operations. They provide services to other objects.
- Objects may be implemented *sequentially* or *concurrently*.

Key Points

- Object interfaces should be defined precisely using e.g. a programming language like Java.
- Object-oriented design potentially simplifies system evolution.
- The Unified Modeling Language provides different notations for defining different object models.