OOP vocabulary

objects

- Something that represents an atomic entity.
- Something you can uniquely identify (identity).
- Something that encapsulates data as its state.
- Something you can send messages to, causing it to respond/behave in some way.
- Something whose behaviour depends on its internal state (which may change).
- Exists at runtime.

object ***has*** state + identity + behaviour

state

- All the data the object currently encapsulates.
- Data is defined in terms of named attributes (instance variables).
- The values of some attributes are fixed (immutable).
- The values of some attributes can change (mutable).
- The values of the attributes are the state of the object.
- If all attributes are immutable, then the object is also immutable.

identity

- Attribute values may change, so they don't uniquely characterise an object over time.
- Identity makes it possible to distinguish any object in an unambiguous way, independent from its state.
- Characterises the object to give persistence.
- Like a primary key in a database, or a unique handle.

interfaces

- An object can have public and private interfaces.
- Public interface defines the messages accepted from other objects.
- Private interface can only be used by the object itself.
- An object can conform to a pre-defined public interface.
    - A conforming object can be used wherever the interface is specified

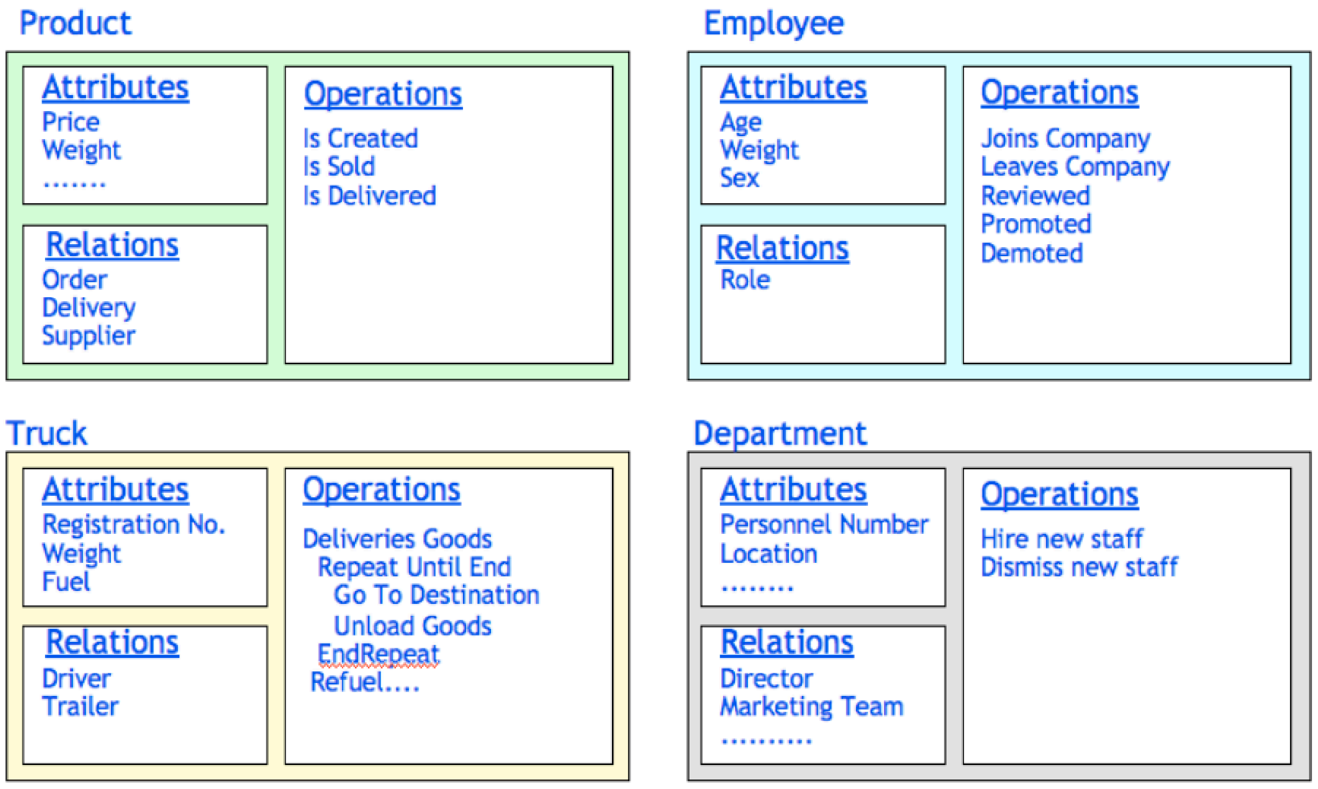
method

- An implementation of some specific behaviour.
    - e.g., Java code.
- A message is dynamically bound to a method.
    - Dynamic binding.
- The bound method is determined by the the object’s class.
- The same message can be send to different objects and bound to different methods.
    - Remember Java dynamic binding.

class

- A class is a template/blueprint/factory specifying a particular kind of object.
- A class can have many instance objects.
- Every object must be an instance of a class.
- Class have relationships to other classes to define the class structure of a system.

examples of classes:



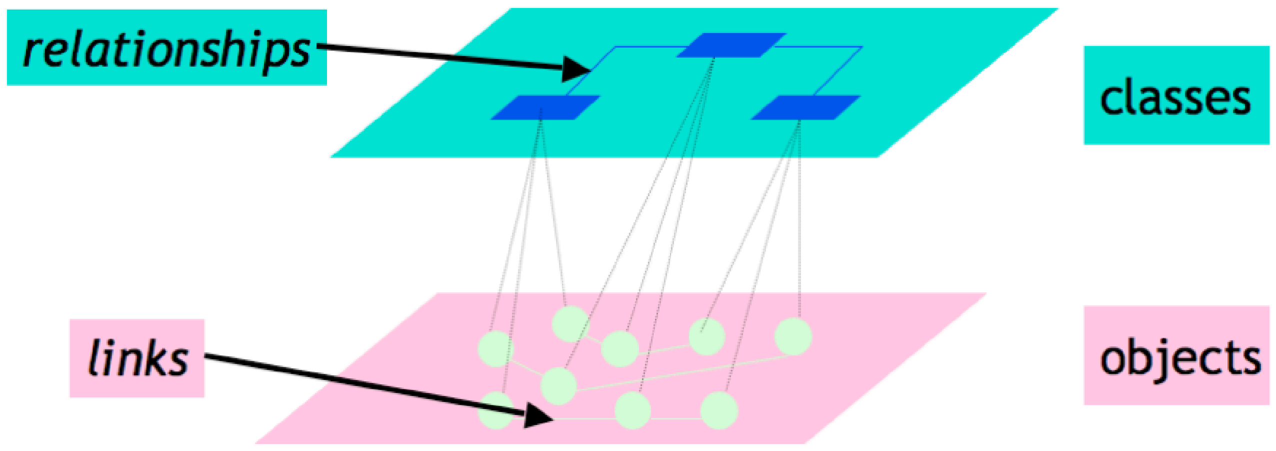
Instantiation: The process of creating a new object belonging to a class.
Instance: The resulting object

- Note that this object is a new ‘thing’ with its own identity.
- The state of the new object must be initialised.

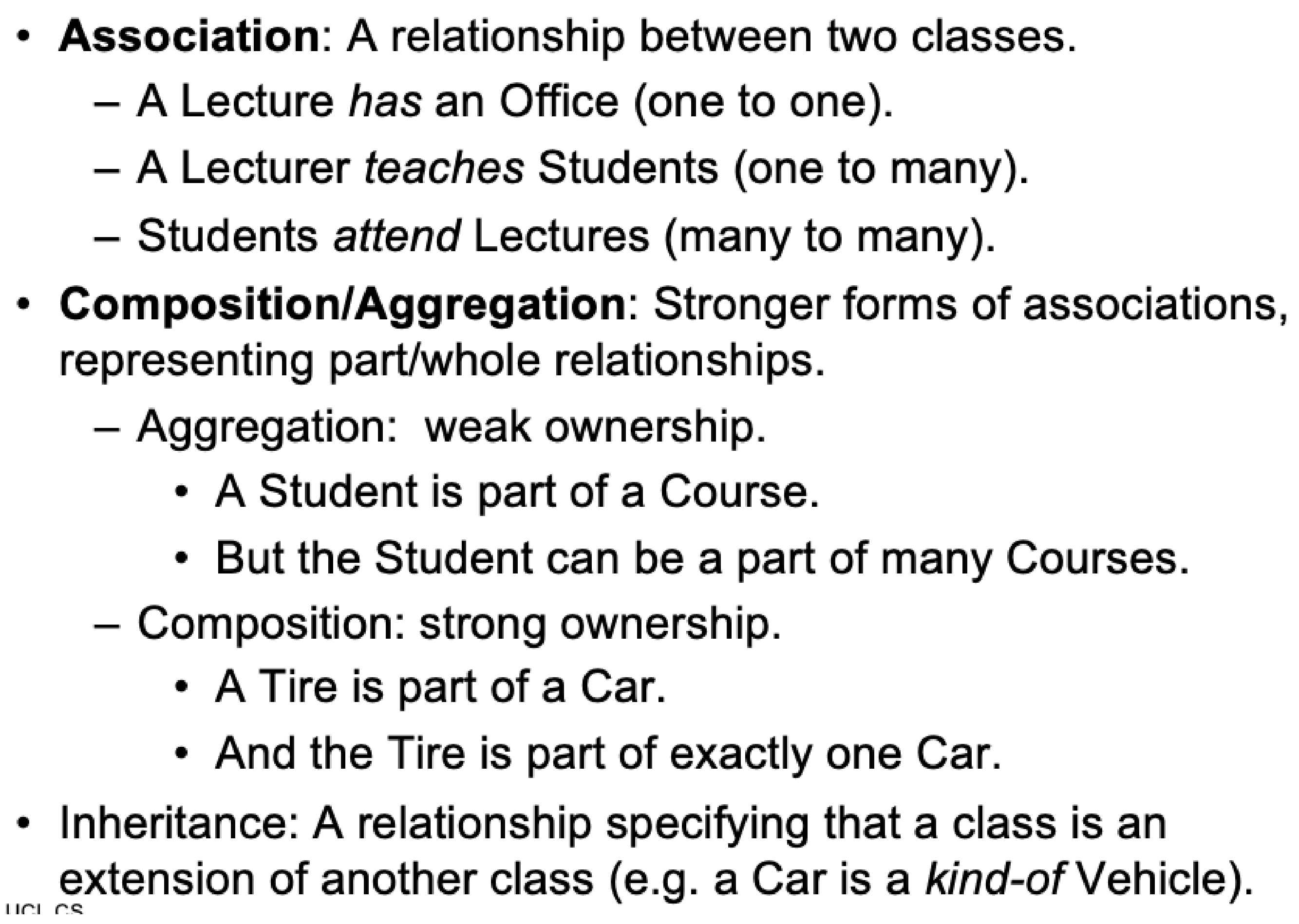
- The class of the object determines its public interface.
    - It will behave consistently with all other objects of its class.

classes vs objects

- Classes define the structure and behaviour of a system.
- Objects represent the actual system



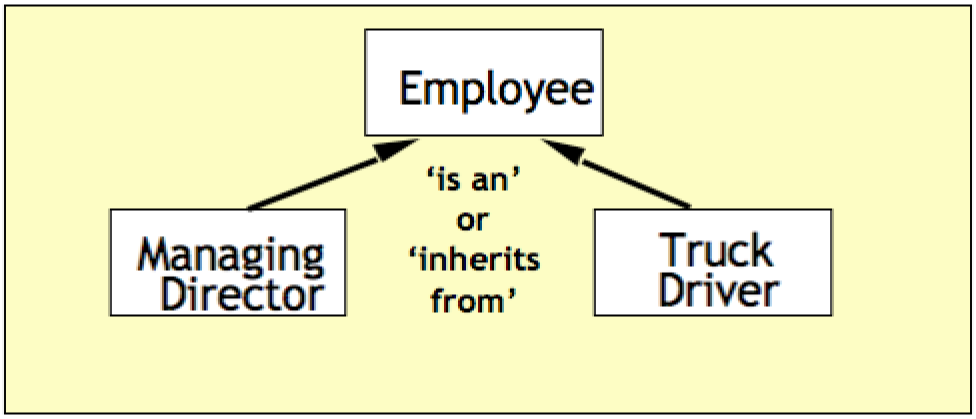
relationships:



- **Association**: A relationship between two classes.
    - – A Lecture *has* an Office (one to one).
    - – A Lecturer *teaches* Students (one to many).
    - – Students *attend* Lectures (many to many).
- **Composition/Aggregation**: Stronger forms of associations, representing part/whole relationships.
    - – Aggregation: weak ownership.
        - A Student is part of a Course.
        - But the Student can be a part of many Courses.
    - – Composition: strong ownership.
        - A Tire is part of a Car.
        - And the Tire is part of exactly one Car.
- Inheritance: A relationship specifying that a class is an extension of another class (e.g. a Car is a *kind-of* Vehicle).

inheritance

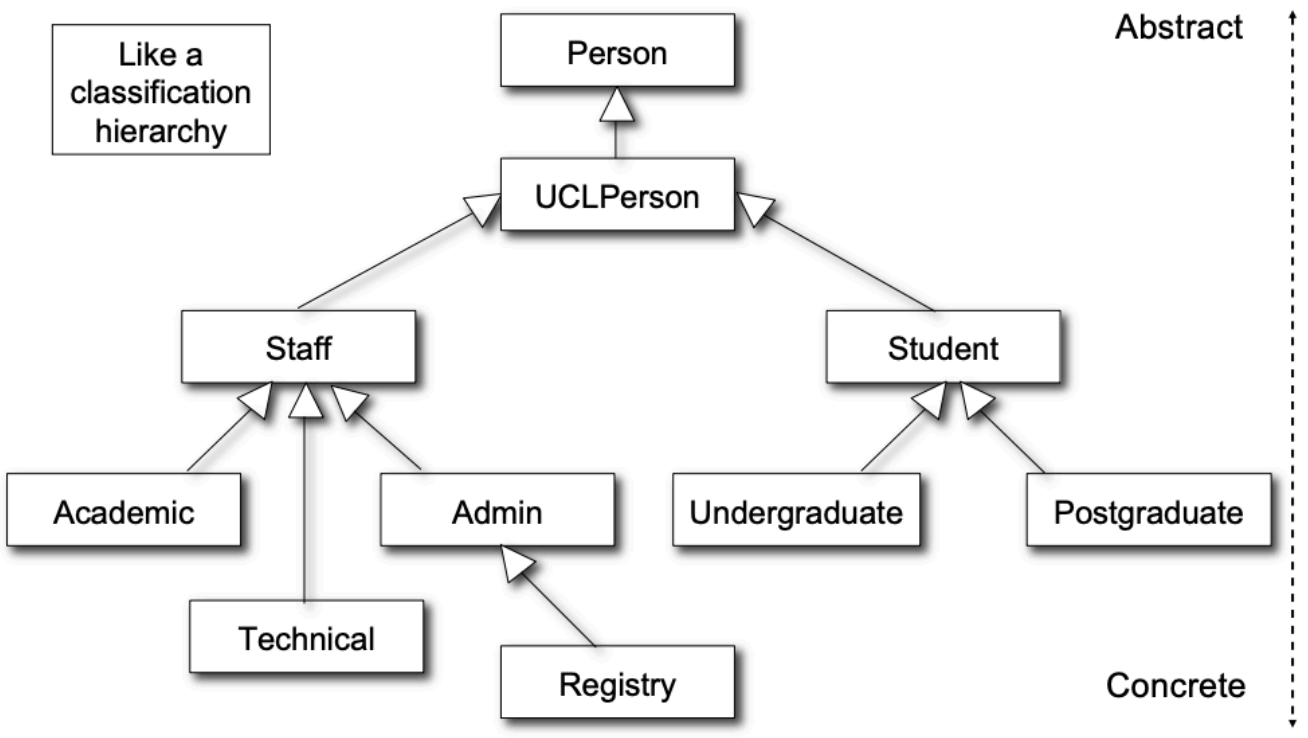
- Inheritance is a relationship between different classes that share common characteristics.
- 'If class B inherits class A, then both the operations and information structure in class A will become part of class B' [Jacobsen 1992].



Generalisation, Specialisation and Polymorphism

- A superclass may be inherited by a subclass.
- The subclass gains all the properties of the superclass and can add more.
- The **superclass** is a **generalisation**.
  - e.g.
    - Shape defines the abstract properties of shapes in general.
    - Number defines the common behaviour of numbers.
    - Employee defines common attributes (name, date of birth, etc.)
- The **subclass** is a **specialisation**.
  - e.g.
    - Square represents a specific kind of concrete shape.
    - Integer, Double define specific kinds of number representation.
    - Undergraduate defines specific attributes (e.g., year, registered modules)
- Leads to polymorphism.
  - Any message accepted by a superclass can be accepted by a subclass.
  - Example: Re-fuel a Vehicle.
    - Can be Airplane, a Car, or a Bus

hierachy:



abstract vs concrete classes:
Abstract classes provide a partial or abstract description.

- Not enough to create instance objects.
- Define a common set of public methods that all subclass objects must have
- common interface.
- Define a common set of attributes/operations can be shared via inheritance.
    - Do not need to be duplicated in all subclasses.

Concrete classes provide a complete description.

- Inherited + new attributes/operations.
- Inherit shared interface.
- Can be used to create instance objects.