What is Object-Oriented Programming?

For part 1 of the course, you are recommended to read material in addition to the lecture notes. A particularly good text to support part 1 is *An Introduction to Object-Oriented Programming*, 3rd edition, by Timothy Budd.

Supplementary reading:

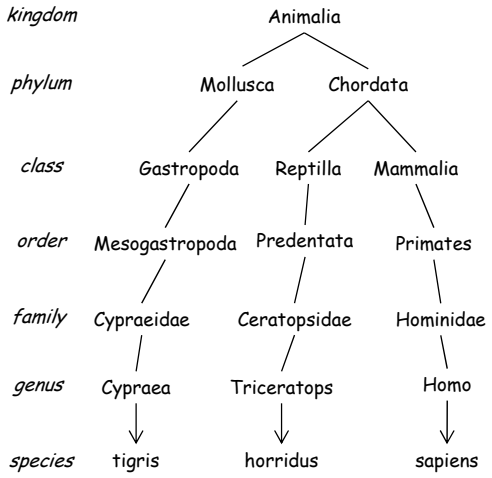
Java text (e.g. Budd Ch. 1, 2)

Java tutorial (http://www.cs.auckland.ac.nz/JavaCache/tutorial)

Trail: "Learning the Java Language"
Lesson: "Object-Oriented Programming Concepts"

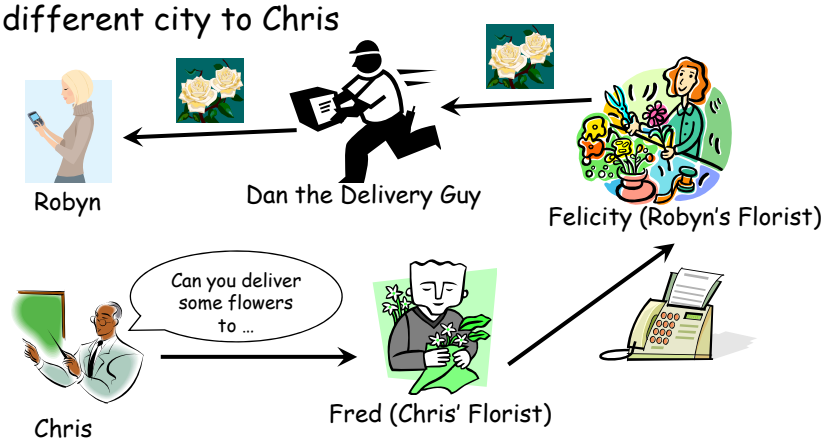
What is OOP?

- OOP is an approach to software development that:
  - **Scales** from trivial to complex problems
  - Can lead to **change-resilient** software
  - Promotes **reuse** of existing code
- Effective OOP requires:
  - One to solve problems using an object-oriented mindset
    - Fortunately, this mindset is similar to that used in everyday life
  - Logic, intelligence, and creativity
  - An ability to build and use **abstractions**

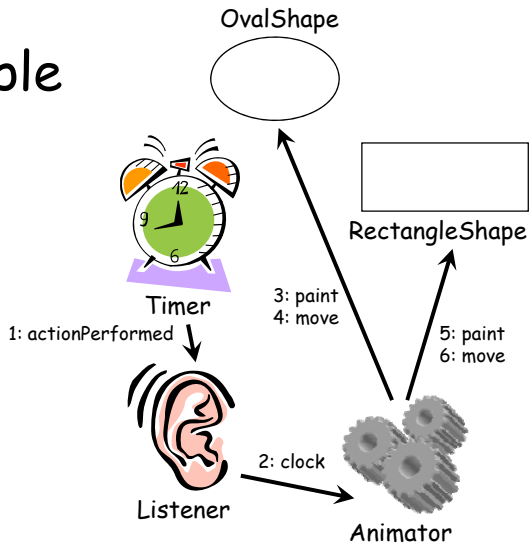


A way of viewing the world

- Chris wants to send flowers to Robyn, who lives in a different city to Chris



Another example



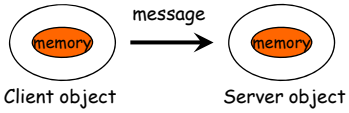
Objects, messages & methods

- **Object**
  - An object remembers things (i.e. has memory) and exhibits behaviour (i.e. an ability to process messages)
  - Objects have a **responsibility** to service particular messages
- **Message**
  - A message is sent to an object
  - A message contains a **request** for service
- **Method**
  - If an object is agreeable to process a message, it uses a particular method to process the request
  - A method is an algorithm – the details of which are of no concern to the sender of the message

An object-oriented program is structured as a community of interacting agents called *objects*. Each object has a role to play. Each object provides a service or performs an action that is used by other members of the community.

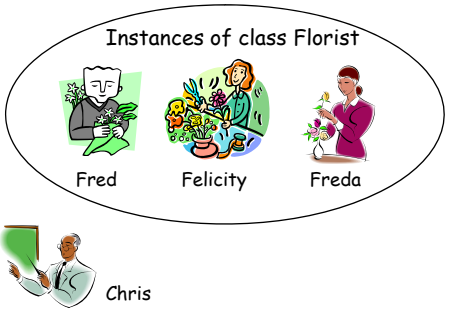
Information hiding

- **Information hiding** is a fundamental principle
  - When a **client** object sends a **server** object a message, the client need not know the actual means by which the request will be processed



- Why might information hiding be a good thing?

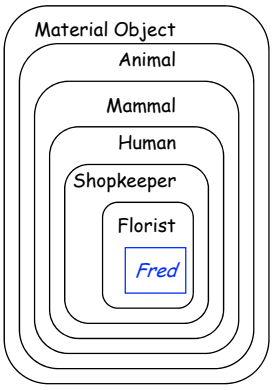
Classes and instances

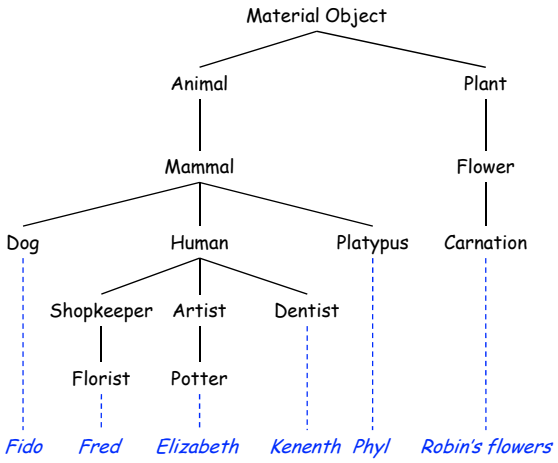


- Chris knows how to communicate with Fred
- If Chris comes across other florists – instances of the same class as Fred – Chris can assume they'll understand the same messages as Fred

All objects are instances of a class. The method invoked by an object in response to a new message is determined by the class of the receiver object. All objects of a given class use the same method in response to similar messages.

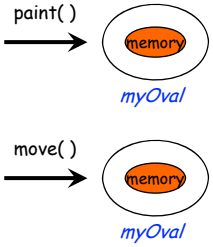
Inheritance

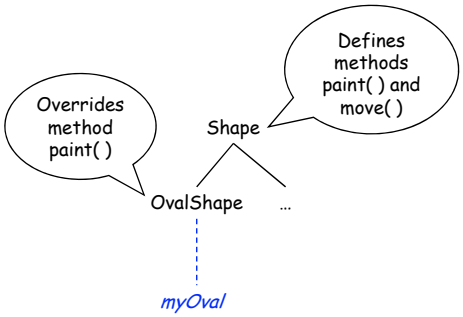




Classes can be organized into a hierarchical *inheritance* structure. A *child class* (*subclass*) will inherit attributes from a *parent class* (*superclass*) higher in the tree. An *abstract class* is a class (e.g. Mammal) for which there are no direct instances, it is used only to create subclasses.

Method binding

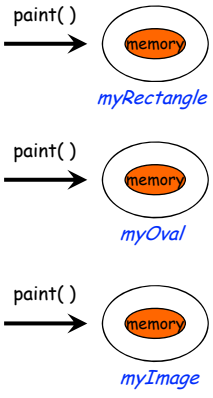


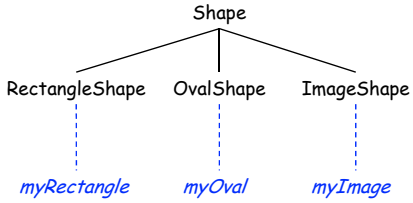


The search for a method to invoke in response to a given message begins with the *class* of the receiver. If no appropriate method is found, the search is conducted in the *parent class* of this class. The search continues up the parent class chain until either a method is found or the parent class chain is exhausted.

If methods with the same name can be found higher in the class hierarchy, the method executed is said to *override* the inherited behaviour.

Polymorphism



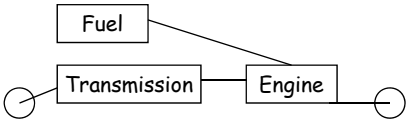


myRectangle, myOval, and myImage will all react to a paint( ) message but will use different methods in their response. This is known as *polymorphism*.

Abstraction

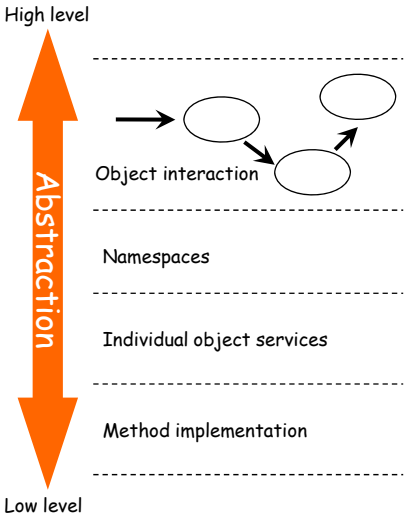
Abstraction is the purposeful suppression, or hiding, of some details of a process or artefact, in order to bring out clearly other aspects, details, or structures.

- Abstraction is essential when working with complex systems



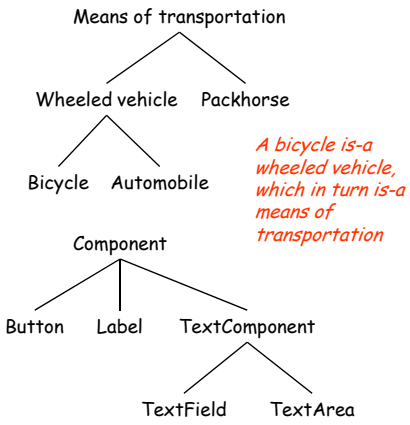
- An abstraction of a car engine
  - A device that takes fuel as input and produces a rotation of the drive shaft as output
  - This rotation is too fast to connect to the wheels of the car directly, so a transmission is used to reduce a rotation of several thousand revolutions per minute to a rotation of several revolutions per minute

Abstraction & software development

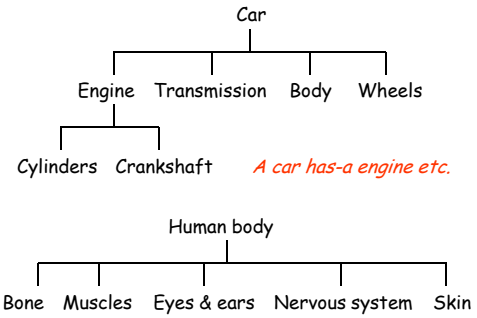


- Object interaction
  - The key objects in a community and the messages they send
- Namespaces
  - A grouping of logically-related classes
  - E.g. javax.swing, java.util
- Individual object services
  - An object's interface, detailing the services offered
- Method implementation
  - Consideration of the data structures and algorithms used to implement a method

Specialisation



Division-into-parts



Both specialisation (*is-a*) and division-into-parts (*has-a*) are widely practised forms of abstraction in OOP.

Interface & implementation

- An object's *interface* represents **what** the object does
- An object's *implementation* describes **how** the interface is realised
- Java offers language support for interfaces
  - Use of interfaces promotes information hiding and allows for **easy interchange** of implementations

```
public interface Date {                    What is offered
   int compareTo( Date that );
   String toString( );
   void advance( int n );
}
```

```
public class DateImpl1 implements Date {
   private int d;

   How it is offered. This implementation is based on
   a day-in-epoch number where the earliest date
   (say 1 January 2000) is represented as zero

   public DateImpl1( int y, int m, int d )
         throws Exception { …}
   public int compareTo( Date that ) { … }
   public String toString( ) { … }
   public void advance( int n ) { … }
}
```

```
public class DateImpl2 implements Date {
   private int d, int m, int y;
   …
}                                          Alternative implementation
```

```
void doSomethingUseful( Date d ) { …
}
                    Variable d can refer to any
                    implementation of Date
```

Summary of OOP concepts

- Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending and receiving messages. A message is a request for action bundled with whatever arguments may be necessary to complete the task
- Each object has its own memory, which can include references to other objects
- Every object is an instance of a class. A class simply represents a grouping of similar objects
- The class is the repository for behaviour associated with an object. That is, all objects that are instances of the same class perform the same actions
- Classes are organised into a singly rooted tree structure, called the inheritance hierarchy. Memory and behaviour associated with instances of a class are automatically available to any class associated with a descendent in this tree structure