Information Systems Concepts

What Is Object-Orientation?

Roman Kontchakov

Birkbeck, University of London

Based on Chapter 4 of Bennett, McRobb and Farmer:
You’d have to be living face down in a moon crater not to have heard about object-oriented programming.

    Tom Swan

Object-oriented programming is an exceptionally bad idea which could only have originated in California.

    Edsger Dijkstra
Outline

- Object-Orientation Concepts
  - Section 4.2 (pp. 91 – 106)
- Object-Orientation Benefits
  - Section 4.3 (pp. 106 – 109)
An object is “an abstraction of something in a problem domain, reflecting the capabilities of the system to keep information about it, interact with it, or both.”

Coad and Yourdon (1990)

“We define an object as a concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand. Objects serve two purposes: they promote understanding of the real world and provide a practical basis for computer implementation.”

Rumbaugh et al. (1991)
Object

- “Objects have state, behaviour and identity.”
  Booch (1994)

- Identity (Who am I?)
  - each object is unique

- State (What do I know?)
  - the conditions of an object at any moment that affect how it behaves

- Behaviour (What can I do?)
  - the way in which an object responds to messages
## Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Identity</th>
<th>States</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>A person</td>
<td>‘Hussain Pervez’</td>
<td>Studying</td>
<td>Speak</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resting</td>
<td>Walk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Qualified</td>
<td>Read</td>
</tr>
<tr>
<td>A shirt</td>
<td>‘My favourite button-down white denim shirt’</td>
<td>Pressed</td>
<td>Shrink</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dirty</td>
<td>Stain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Worn</td>
<td>Rip</td>
</tr>
<tr>
<td>A sale</td>
<td>‘Sale no 0015, 15/06/02’</td>
<td>Invoiced</td>
<td>Earn loyalty points</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cancelled</td>
<td></td>
</tr>
<tr>
<td>A bottle of ketchup</td>
<td>‘This bottle of ketchup’</td>
<td>Unsold</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opened</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Empty</td>
<td></td>
</tr>
</tbody>
</table>

**A coffee machine object?**
Identity != Equality

- Different objects must have different identities
- Different objects may have exactly the same state
  - e.g., twin brothers, two interchangeable blue pens, etc.

[Java]
if (obj1 == obj2)  tests identity
if (obj1.equals(obj2)) tests equality
Object has

State

Behaviour

Identity

(equal ≠ identical)

Class

- A class is “a set of objects that share the same specifications of features (attributes, operations, links), constraints (e.g. when and whether an object can be instantiated) and semantics”

  OMG (2004)

- Moreover, “The purpose of a class is to specify a classification of objects and to specify the features that characterize the structure and behaviour of those objects”

  OMG (2004)
Class

- An object = An instance of some class
  - Every object must be an instance of some class
- A class = A set of objects that share the same
  - structure
    - what information it holds
    - what links it has to other objects
  - behaviour
    - what things it can do
<table>
<thead>
<tr>
<th>Identity</th>
<th>Class</th>
<th>In C++</th>
<th>In Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Structure</td>
<td>Member</td>
<td>Fields</td>
</tr>
<tr>
<td>(values)</td>
<td>Specification</td>
<td>Variables</td>
<td></td>
</tr>
<tr>
<td>Operations</td>
<td>Behaviour</td>
<td>Member</td>
<td>Methods</td>
</tr>
<tr>
<td></td>
<td>Specification</td>
<td>Functions</td>
<td></td>
</tr>
</tbody>
</table>
Ways of thinking about a class

- A **factory** that manufactures objects according to some blueprint
- A **set** that specifies what features its member objects have
- A **template** that allows us to produce any number of objects of a given shape
Class

an abstraction (generic description) for a set of objects

Object

an instance of a class

Message Passing

- Objects collaborate to fulfil some system function, and they communicate by sending each other messages:
  - A question message asks an object for some information
    - How much is the balance?
  - A command message tells an object to do something
    - Withdraw 100 pounds
Message Passing: Example

- Buying a loaf of bread:

```
What do you have? →

← Two white, one wholemeal

I'd like a wholemeal →

← I'd like some money

Here's some money →

← Here's the loaf

← Here's your change
```
Encapsulation

‘Layers of an onion’ model of an object

An outer layer of operation signatures...

...gives access to middle layer of operations...

...which can access inner core of data

Message from another object requests a service.

Operation called only via valid operation signature.

Data accessed only by object’s own operations.

An object’s data is hidden (encapsulated).
Encapsulation

Consider an object representing a circle.

A circle would be likely to have operations allowing us to discover its radius, diameter, area and perimeter. We could store any one of the four attributes and calculate the other three on demand. Let's say we choose to store the **diameter**. Without encapsulation, any programmer who was allowed to access the diameter might do so, rather than going via the **getDiameter** operation.

If, for a later version of our software, we decided that we wanted to store the **radius** instead, we would have to find all the pieces of code in the system that used direct access to the diameter, so that we could correct them (and we might introduce faults along the way).

With encapsulation, there is no problem.
Encapsulation

Object data is hidden

Operations **encapsulate** manipulation of the data

Generalization / Specialization

- Classification is hierarchical in nature
  - A person may be an employee, a customer or a supplier
  - An employee may be paid monthly, weekly or hourly
  - An hourly-paid employee may be a driver, a cleaner or a sales assistant.
- Every instance of the specific class (subclass) is also an instance of the more general class (superclass)
- A subclass is a (kind of) its superclass
Generalization / Specialization

Person

Employee
  - Monthly-paid
  - Weekly-paid

Customer
  - Hourly-paid

Supplier

More general
(superclasses)

More specific
(subclasses)

Driver
Cleaner
Sales Assistant
Taxonomies

- Animal
  - Mammal
    - Whale
    - Dog
    - Cat
      - Domestic Cat
      - Tiger
  - Fish
  - Bird

More general (superclasses)
More specific (subclasses)
Inheritance

- A subclass always inherits **all** the characteristics (data structure and behaviour) of **all** its superclasses
- The definition of a subclass always includes at least one detail not derived from any of its superclasses
Generalization

A subclass inherits the structure and behaviour of its superclass


Not a good visualization of generalization, because subclasses inherit types, not values (a nose not a long nose)!
Generalization in UML

A superclass has general characteristics that are inherited by all subclasses.

The symbol for generalization.

Subclasses have specialized characteristics that are unique to each subclass.
Advantages of using Generalization

Employee
- dateOfAppointment
- dateOfBirth
- department
- employeeNumber
- lineManager
- name

This new subclass requires no change to the existing structure.

MonthlyPaidEmployee
- monthlySalary

HourlyPaidEmployee
- hourlyRate
- hoursWorked

WeeklyPaidEmployee
- weeklyWage
Multiple Inheritance

- We may want the `Part-Time BSc Student` class to be a sub-class of both the `BSc Student` class and the `Part-Time Student` class.
Generalization: Exercise

- How shall we group these classes into a generalization hierarchy?
Polymorphism

- Polymorphism allows one message to be sent to objects of different classes
- Sending object need not know what kind of object will receive the message
- Each receiving object responds appropriately, i.e., different kinds of objects may respond to the message in different ways

poly morph ic = having many shapes
Polymorphism: Example
Polymorphism: Example
Polymorphism: Example

“calculatePay” for different kinds of employees

1: getTotalPay()

2a: calculatePay()

:FullTimeEmployee

Fixed monthly amount depends only on employee grade

2b: calculatePay()

:PartTimeEmployee

Variable monthly amount depends on grade and hours

2c: calculatePay()

:TemporaryEmployee

Fixed monthly amount depends on grade, but no pension deductions
Polymorphism: What is behind?

```java
if (x instanceof FullTimeEmployee)
    calculatePay1(x);
else if (x instanceof PartTimeEmployee)
    calculatePay2(x);
else if (x instanceof TemporaryEmployee)
    calculatePay3(x);

x.calculatePay();
```
Benefits of Object-Orientation

- Object-Orientation concepts and techniques improve both software quality and software productivity
  - Abstraction, Modularity and Reusability
  - Event-Driven Programming and GUI Programming
  - Model Transition and Iterative/Incremental Lifecycle
Take Home Messages

- Object-Oriented Concepts
  - Object and Class
  - Encapsulation
  - Generalization
  - Inheritance
  - Polymorphism

- Object-Oriented Benefits