A-level
COMPUTER SCIENCE

Object Oriented programming

PowerPoint slides

Published: Autumn 2017

Contents

Object oriented programming – A practical approach

Christine Swan

About me

Christine Swan

Experienced teacher
(28 years total, 18 years Computer Science)
Computing at School Master Teacher
Member of Raspberry Pi Foundation
Specialist Leader in Education

Learning outcomes

What are your expected outcomes for today?

Aims:

- To enable you to understand the fundamental concepts of object oriented programming and apply them in Visual Basic *(the ideas will transfer to other languages such as Java).*
- To be able to create class diagrams and answer A-level standard questions.
- To have a basic understanding of some advanced OOP (object oriented programming) concepts and be able to start applying them to scenarios.

AQA

Agenda

Time	Session
09.45	Coffee and Welcome
10.15	Introduction to OOP
11.30	Coffee
11.45	Practical application of OO – setting up classes and inheritance
12.45	Lunch
13.30	Class diagrams and advanced concepts
14.30	Coffee
14.45	Specimen questions and further practical work
15.30	Plenary and review
15.45	Close

AQA

Identify individual needs from the day

Now you have had a minute to think about your own needs.

It may be useful to edit this in throughout the day to track your progress.

Please ask questions as you need.

You have also been provided with a **learning mat**. This will be useful to use with your students.

AQA

AQA A-level specification, page 42

4.1.2.3 Object-oriented programming

Content	Additional information
Be familiar with the concepts of: • class • object • instantiation • encapsulation • inheritance • aggregation • composition • polymorphism • overriding.	Students should know that: • a class defines methods and property/attribute fields that capture the common behaviours and characteristics of objects • objects based on a class are created using a constructor, implicit or explicit, and a reference to the object assigned to a reference variable of the class type • in the Unified Modelling Language (UML) composition is represented by a black diamond line and aggregation by a white diamond line.
Know why the object-oriented paradigm is used.	
Be aware of the following object-oriented design principles: • encapsulate what varies • favour composition over inheritance • program to interfaces, not implementation.	Students would benefit from practical experience of programming to an interface, but will not be explicitly tested on programming to interfaces or be required to program to interfaces in any practical exam.
Be able to write object-oriented programs.	Practical experience of coding for user-defined classes involving: • abstract, virtual and static methods • inheritance • aggregation • polymorphism • public, private and protected specifiers.
Be able to draw and interpret class diagrams.	Class diagrams involving single inheritance, composition (black diamond line), aggregation (white diamond line), public (+), private (-) and protected (#) specifiers.

AQA

Why object oriented programming?

The paradigm shift and the benefits of programming in object oriented paradigm

Imagine you have a pile of books and I ask you to find my DICTIONARY. It just so happens to be the 5th one down in the pile BUT I don't want you to disturb any of the other books and you don't know where it is yet.

Can you do it?

Not easily!

AQA

Imperative (procedural) programming

I have now put my books onto two shelves.

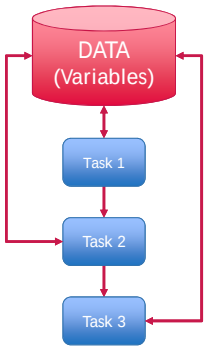
Can you find the DICTIONARY now?

Putting my books onto shelves has made the task much easier.

AQA

Imperative (procedural) programming paradigm

- In this paradigm, programs tell the computer exactly what to do, line by line.
- Programs use sequence, selection and iteration to control program flow and manipulate values of variables and constants stored in memory directly.
- Procedures and functions are modules of code that perform tasks.
- They use can be passed parameters and use local variables.
- Functions, return a value or values to the procedure/function that called it.
- Example languages: C, BASIC and Pascal



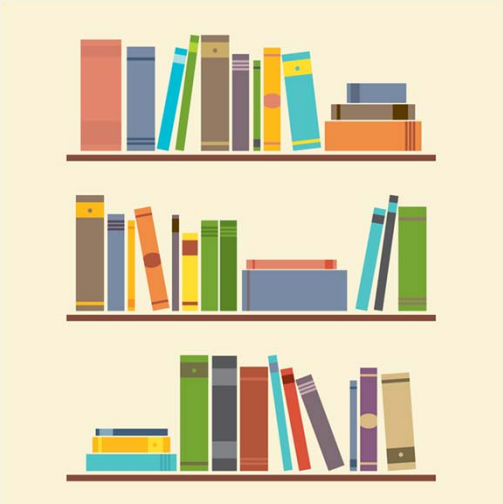
AQA

Back to the book analogy

- The first imperative programs were just one long list of commands. Just like our unsorted pile of books.
- Then tasks were modularised (put into procedures and functions), just like our separate book shelves.
- However, this doesn't prevent anyone going to each shelf taking out a book and putting it on another shelf. *(Such as changing a variable in one function which accidentally alters another).*
- This could be a real problem if we have made a reference for it's location, as we would in a typical library.

AQA

Book analogy



Now the book has been moved we have to find it then start again with organising into meaningful piles/shelves or modules.

AQA

Problems with imperative programming

This style of programming has been utilised successfully for many years but there are some drawbacks:

- Programs can become very complex and convoluted. Some programs, which solve real world problems, can end up with hundreds of functions and procedures communicating with each other. They pass data to one another through parameters and this can become very difficult to keep track of.
- If data changes in one module (or our book is moved) but not in another, this can result in disaster. With larger programs, it is also tempting to use global variables but this goes against sensible programming techniques.
- Maintaining imperative programs can prove difficult, especially when communicating modules make changes that have unforeseen consequences in other parts of the code.

AQA

So why object oriented?

The basic unit of an object oriented program is a class.

Classes define the attributes (properties, data) and operations (methods, behaviours) that an object of that class will have. As such, it can be thought of as a blueprint.

An instance of the class is called an object.

Every object will have the same data associated with it and support the same behaviours.

More importantly, the programmer doesn't need to know the exact code that carries out a particular operation and it only affects the particular object in question.

In this way accidental changes are prevented and do not impact on anything else in the program.

AQA

So why object oriented?

Reusability is key:

- Once a class has been written, created, and debugged, it can be used again and again. It can be distributed to other programmers for use in their own programs.
- In OOP an existing class can be used without modifying it and we can add additional features and capabilities to it. This can be done by deriving a new class from the existing one. The new class will inherit all the capabilities of the old one, but we can add new features that are its own.
- Reusability is using a class over and over in different programs. The ease with which existing software can be reused is an important benefit of OOP.

Let us find out how >>>>>>>>>>>>

AQA

Starter task

An Introduction to objects

- Can you name some properties of a car?
- What can a car do?
- Are there other types of car?
- What do they have in common?

What is a Class?

- A class is a definition of the properties and operations associated with objects of that class e.g. has 4 wheels, can move etc.

What is an Object?

- An object is an instance of a class – the picture shows a specific car.

AQA

Objects vs non-objects

Task 1 - So what is an object?

Task 2 - Even more challenging – a non-object?

OBJECT	NON-OBJECT
A red bicycle	The bottom tube of the frame
A mini car	The colour of the car
A 750cc motorbike	The noise of the motorbike
An A380 plane	The number of seats on the plane

AQA

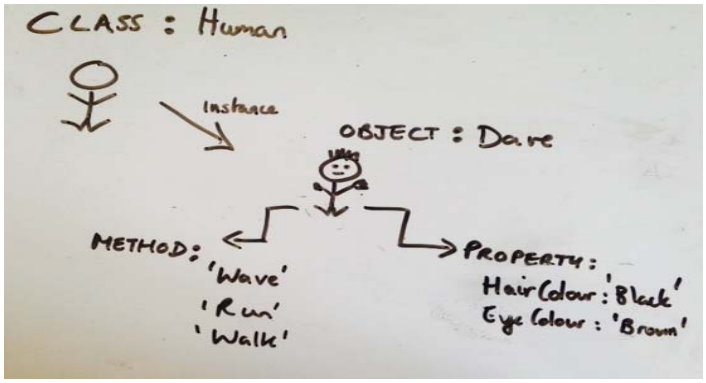
Objects

- Objects in the real world have
  - An identity (its name/identifier)
  - A state (value of a property)
  - Behaviour (methods … can do stuff or have things done to it)
- Let's look at the ball

Identity	my_car
State	colour = black
Behaviour	move()

AQA

Classes and objects



- A **class** can be thought of as a blueprint for objects.
- An **object** is an instance of a class.
- **Instantiation** is the process of creating an object of the class.

AQA

Objects

Rather than focus on real world objects, objects in an OO program share the same characteristics.

- Let's think about a program for holding student bank accounts:

Identity Dave Smith's account

State Balance = £5.00 OverdraftLimit = £100

Behaviour makeDeposit() checkBalance()

makeWithdrawal()

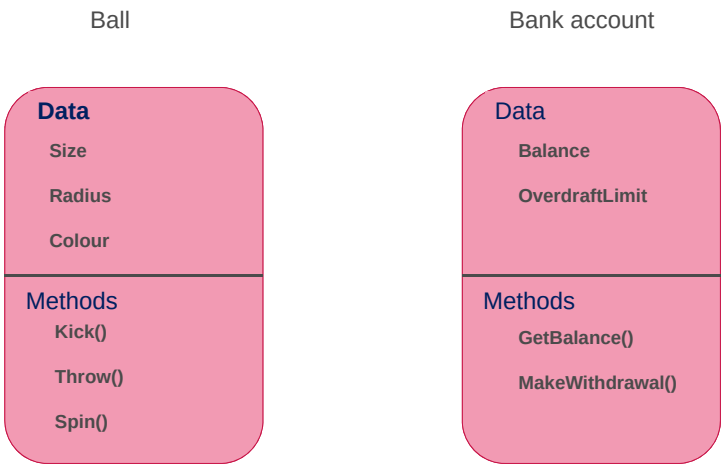
AQA

Unified Modelling Language

- The Unified Modelling Language was developed in 1996 to standardise object oriented software design
- The current version is 2
- There are many different types of diagram but your students need to be able to draw CLASS DIAGRAMS

AQA

So how can we represent this diagrammatically?



AQA

Further examples

- A useful analogy for A-level students first getting to grips with OOP is the car class.
- It is easy to understand the idea that there are many different types of car.
- All cars have properties such as colour, number of seats, make, engine size.
- They all have common methods such as drive and park. Some of which can be carried out in different ways.
- We can further expand to help explain more advanced concepts.

AQA

Task

For the following classes, identify some **properties** and **methods**.

1. **Car**
2. **Person**
3. **Dog**
4. **Customer**

Now we have the basics let us think about classes

- A class can be thought of as a blueprint or a master copy of a related group of items. We can liken this to a cookie cutter. There is only one cookie cutter but we can use it to make many cookies.
- In the example shown we can use the gingerbread man cutter to make many gingerbread cookies.
- Each gingerbread cookie is an individual as it is made from a different piece of gingerbread dough and even though each cookie has the same basic properties (e.g. 2 eyes, a mouth etc), each one can have different values. For example one has blue smarties for eyes, the other green smarties.
- You can create any amount of cookie objects and you can also destroy them (yum, yum) but this does not affect the cookie cutter; it can be used again and again.

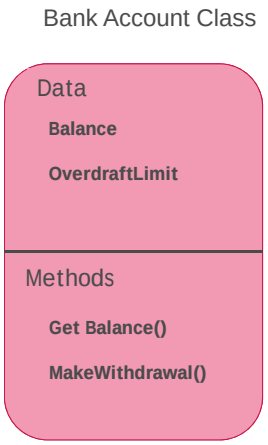
Classes, objects and encapsulation

- The concept of grouping data and methods together within the same object is known as **Encapsulation** or information hiding. This promotes separation of the implementation and the interface and is a key principle of OO programming.
- The subroutines (or **methods**) act on the data within the object.
- This helps overcome the issue seen in procedural languages where one subroutine changes the value of data to be used in another, sometimes with serious implications.
- In OO the data for an object can only be directly manipulated by the methods of that object.

AQA

Encapsulation and information hiding

- **Information Hiding** is when the data within an object can only be accessed or modified by the methods within that object.
- Going back to our bank account class we can see the data (properties) and methods which act on it. This means that to withdraw from the an account we can only use ***MakeWithdrawal()***.
- The brackets will hold the parameter to be passed to the method e.g. amount.



AQA

Encapsulation and access modifiers

- Properties of a class are generally declared as **private**. This is so they cannot be accessed outside the class.
- Methods (procedures and functions) can be declared as public, protected, private and friend.
- The keywords (public, protected, private and friend) are known as **Access Modifiers**.
- Access modifiers control the accessibility of types which includes classes, their methods and properties (members).
- Class methods declared as **public**, enable the properties of the class to be accessed from outside the class.

AQA

Access modifiers in Visual Basic

Access modifier	Description
Public	A public member of a class can be accessed from any program that instantiates that class.
Private	Private member declared within a class module is accessible only from within that class module.
Protected	This applies to class members only. This defines a method that is accessible only from within its own class or from a derived class.
Friend*	Defines a type that is accessible only from within the program in which it is declared. So all of the classes in your program/project have access but not those outside of it.

* Not required by AQA

AQA

Defining a class – a general approach

Follow these steps

1. Name the class.
2. Decide on the properties of the class.
3. Declare the properties as private or protected.
4. Declare the methods of the class (procedures and functions).
5. Declare the methods as public.
6. Write the code for the methods (if required).

AQA

Visual Basic example

```
Public Class BankAccount
       Private _AccountNumber As String     'data
       Private _Balance As Decimal
        'property
        Public Balance() As Decimal
           Get
               Return _Balance
           End Get
        End Property
       'method
      Public Sub Deposit(ByVal Amount As Decimal)
        Balance = Balance + Amount
       End Sub

End Class
```


AQA

Java example

```
class GoldAccount
{
// variable declarations
Private String accountNumber;
Private String accountHolder;
private int balance;

// methods
 public int getBalance()
 {
   return balance ;
 }
etc  etc
}
```

NB Constructor not included in this program fragment

AQA

Summary - for a typical OOP

- Properties of a class are declared as **private**. This is so they cannot be accessed outside the class.
- Methods (procedures and functions) are **public** and are written to enable the properties of the class to be accessed from outside the class.
- **Encapsulation** is a term used to describe the combination of the object and its methods. This promotes separation of the implementation and the interface.
- Because the properties of the class are private, you can control changes to the actual object. In addition, you have the flexibility to change the implementation without affecting anything outside the class. This principle is known as **information hiding**.

AQA

Practical Activity

Let us try to get to grips what we have learned so far by trying our first OO program.

Open the practical activity handout and try

Topic 1 – The Basics - Your first object oriented program

Topic 2 - Visibility

AQA

Constructors

- A constructor is a special method which in OOP allows us to have some control as to how objects are initialised.
- It instantiates objects of a class and allocates memory.
- Every defined class will have a default constructor.
- We can also define our own constructors.
- In Visual Basic a constructor method is always called **Sub New**.
- When using the ***New*** keyword we are also telling VB to allocate memory for a new object of the class.

AQA

Practical activity

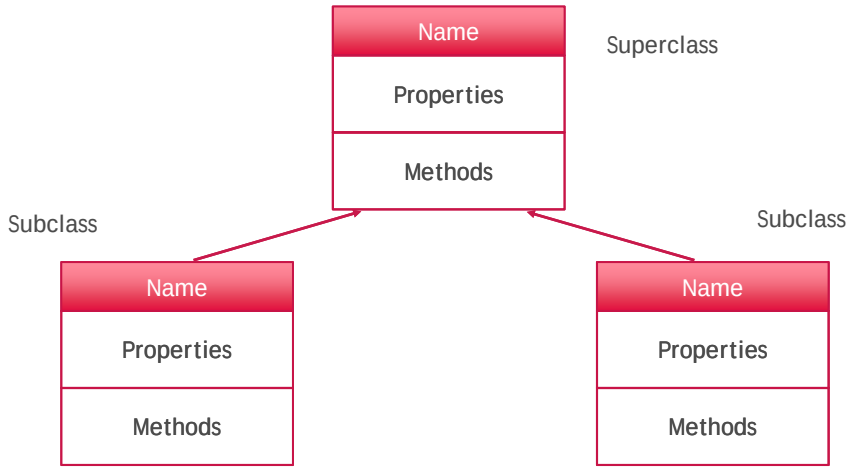
Back to the practical activity handout and try

Topic 3 – Constructors

Basic class diagrams

- Class diagrams are used to depict the classes within a model or design for a program.
- They are one diagram in the UML or Unified Modelling Language which is the common method for modelling object oriented programs.
- There are many different UML diagram types but we will use some basic class diagrams to consolidate ideas already covered and the concepts of inheritance, aggregation and polymorphism.

Basic class diagrams

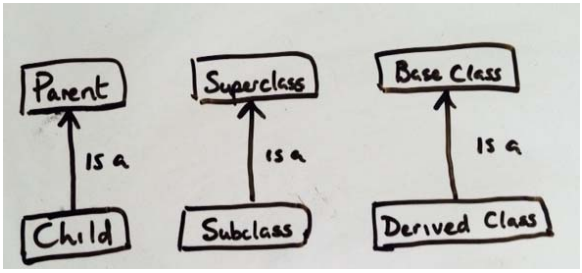


Inheritance

- **Inheritance** is where one class can inherit methods and properties from another class. This promotes efficiency because code from one class can be reused in its sub-class.
- The **Superclass** will consist of general properties and methods.
- The **Subclass** inherits all of the properties and methods from the superclass but will also have its own that may be more specific.
- A class can have several subclasses for example, we could have a superclass called Bank Account class with subclasses Student, Gold and Platinum Accounts each with different account characteristics.

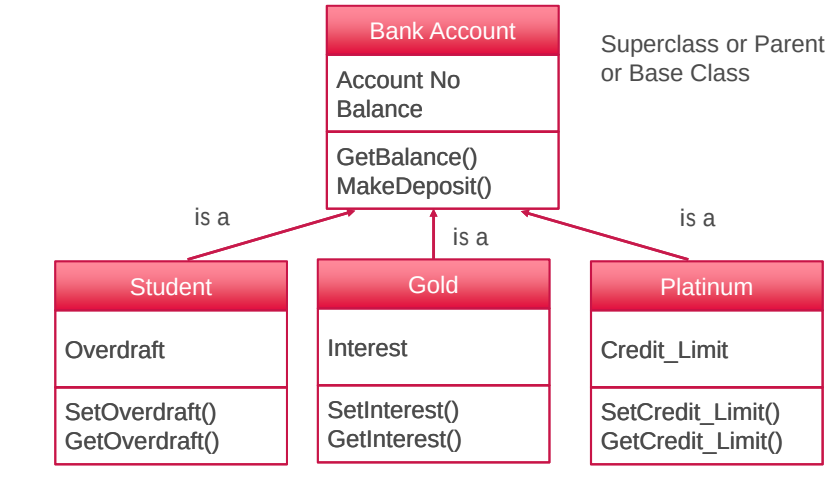
Inheritance

- It will depend on which text book you pick up (and language used) as to the naming conventions used for describing inheritance relationships.
- A parent can have many children but a child can only have one parent.
- Note the “is a” relationship.



AQA

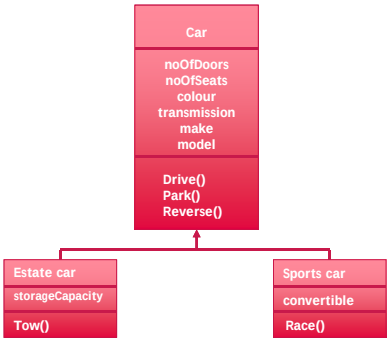
Inheritance - basic class diagram



Child, Sub or Derived Classes

AQA

Back to the car analogy



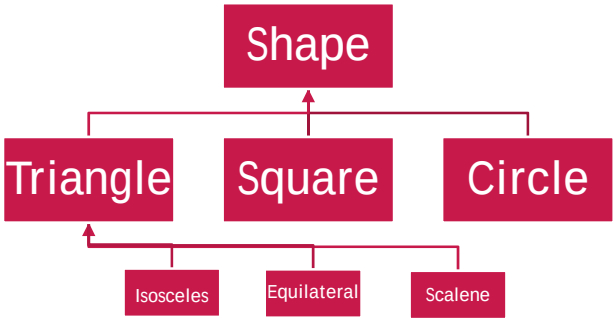
- A Sports Car is a subclass of Car, as is an Estate Car. The “is – a” relationship is upheld as they are both cars.
- There are some common properties such as Colour, NoOfSeats.
- There are common methods such as Drive and Park.
- Each class has will inherit the properties and methods of the Superclass but also have their own.

An object can be instantiated such as “MyCar”

AQA

Inheritance hierarchy

- A subclass can itself be a superclass for another subclass leading to a hierarchy of classes.



- Note that the “is-a” relationship is transitive, if a scalene triangle is a triangle and the triangle is a shape, then it follows that a scalene triangle is a shape.

AQA

Inheritance task June 2014 Q8

8 An event-driven, object-oriented programming language lets the programmer create a Graphical User Interface (GUI) from components such as forms and buttons. The components of the GUI are implemented using a class hierarchy and inheritance.

8 (a) Explain what is meant by inheritance.

[1 mark]

AQA

Inheritance task June 2014 Q8

8 (b) One GUI component is a **Selector**. Selectors come in two different types: **ComboBox** and **ListBox**.

Selector Type	Description
ComboBox	A combo box lets the user make an input either by typing into the box **or** by picking a single item from a list.
ListBox	A list box lets the user select options from a list. The user cannot type into a list box. There are two different types of list box: • **SingleSelectionListBox**: The user can only select one item from a list. Whenever an item is selected, the previously selected item is deselected. • **MultipleSelectionListBox**: The user can select one or more items from a list. Whenever an item is selected, it is added to the list of selected items.

Draw an inheritance diagram for the classes: Selector, ComboBox, ListBox, SingleSelectionListBox and MultipleSelectionListBox.

[3 marks]

AQA

Inheritance task June 2014 Q8

8 (c) The **Selector** class has data fields **Items** and **NumberOfItemsInList**:

- **Items**: an array that stores the list of strings that will appear in the selector.
- **NumberOfItemsInList**: a number that indicates how many items there are in the selector.

It also has a procedure that the programmer can call to add an item to the list of strings (**AddItemToList**) and a procedure that is called by the operating system whenever the user selects an item from the list (**SelectItemFromList**).

The **Selector** class does not include a procedure to display the items in the list as the way items are displayed is different for each type of selector.

The class definition for **Selector** is:

```
Selector = Class
             Public
               Procedure AddItemToList
               Procedure SelectItemFromList
             Private
               Items: Array of String
               NumberOfItemsInList: Integer
           End
```


AQA

Inheritance task June 2014 Q8

The **ComboBox** class needs the following additional data fields:

- **TextTyped**: Stores the characters that have been typed by the user if they have made their input by typing rather than picking an option from the list.
- **SelectedItemNumber**: Stores the position in the list of the item that has been selected by the user, if one has been selected.
- **AllowNonListInputs**: A True or False value that indicates whether the user should be allowed to type in text that is not one of the items in the list.

The class will need to implement the operation of selecting an item from the list differently from the way the **Selector** class implements this operation, but the operation of adding an item to the list will be implemented in the same way by both of these classes.

The class must provide subroutines to:

- display the combo box
- respond to the operating system's notification of a key press
- return the text that has been typed in
- return the selected item number
- set the value of **AllowNonListInputs** flag to True or False, to indicate whether or not the user is allowed to type text that is not in the list.

Write the class definition for the **ComboBox** class.

[5 marks]

Solution

AQA

Practical activity

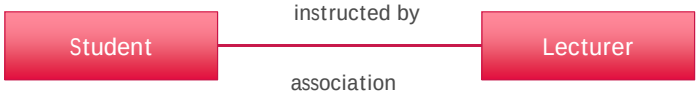
Open the practical activity handout and try

Topic 4 - Inheritance

AQA

Association

- **Association** in OOP is how two or more objects are related to each other. (Could be one to one, one to many).
- Generally the name of the association will specify the relationship between the objects.
- In a class diagram this association is shown with a solid line



- In this situation a student can be instructed by many lecturers and a many students can be instructed by a single lecturer. The key is that they are independent and both can be created or destroyed independently of each other.

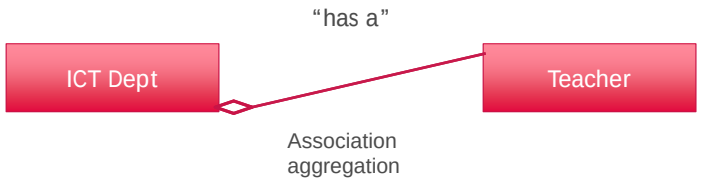
AQA

Association aggregation

Aggregation is a way of creating new objects that contain objects which already exist.

Association aggregation is when an object which contains other objects is destroyed, the other objects will still exist.

For example, if we have a teacher who belongs to a department, and the department is destroyed, there is still a teacher object which exists.



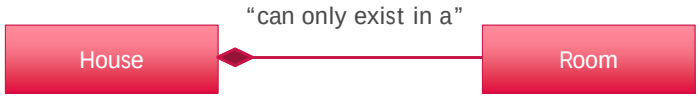
Association aggregation

Notice the "open" arrow head

AQA

Composition aggregation

- This is when an object that contains other objects and if destroyed, then the objects will no longer exist.
- In other words the child object cannot exist without the parent object, so the association is strong.
- For example if we have a house object which contains room objects, if we delete the house then the rooms can no longer exist.



Composition Aggregation

Notice the "solid" arrow head

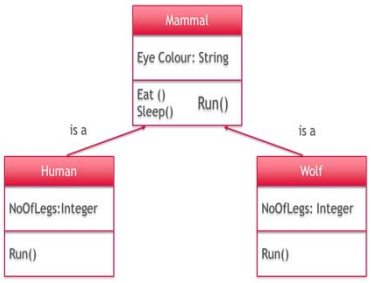
AQA

Polymorphism

- The word **Polymorphism** means "**many forms**".
- In OOP it describes a situation when a method in the super class is inherited in the subclass but is redefined to suit the data and purpose of the sub-class.
- Fictional examples which are often used relate to animals, because the principle is easier to understand.
  - For instance the Mammal superclass.
  - Subclasses which inherit from the superclass could be **Human** and **Wolf**.

AQA

Polymorphism

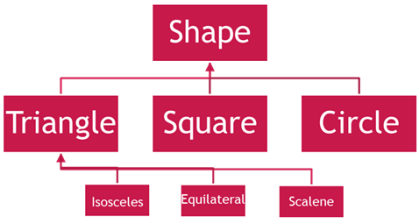


- In the example shown, the super class and both subclasses have a **Run()** method.
- However, we know that humans and wolves run in different ways.
- This is absolutely fine but the mechanism for how the Run() method will function will depend on which class you are using.
- The data required for the method may be different for the specific method too.
- Polymorphism enables the original method (from the superclass) to be redefined (in the appropriate sub-class) so that it can work with a new algorithm/data.

AQA

Polymorphism

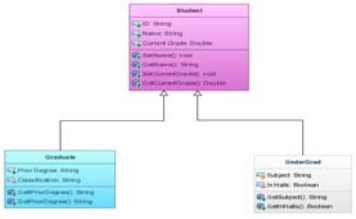
- Another example which can help illustrate polymorphism is using a base class **Shape**.
- Polymorphism enables the programmer to define different area methods for any number of derived classes, such as circle, triangle and rectangle.
- Irrespective of the shape an object is, applying the area method to it will return the correct results.



AQA

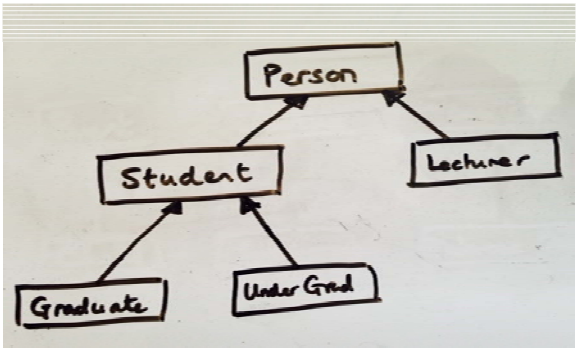
Overriding

- We have actually already been exploring a concept known as overriding.
- The most basic definition is **"when a method in the sub-class class provides a specific implementation of a method that is already provided by the super class or parent class."**
- Let us look go back to our student example mentioned earlier and expand on it.



AQA

Overriding



- This simplified class diagram is called an **inheritance diagram**
- Consider the relationship shown in the diagram.
- For any of the classes an arrow points to it's superclass (the arrow designating the '***is a'*** relationship).

AQA

Overriding

- Suppose the person class has properties *name* and *age*, as well as methods such as *GetName()*, *GetAge()* and *PrintName()*.
- Every one of the subclasses inherits these properties and methods.
- The Student class may have additional properties such as Student_ID and a method CalcGrade(). These additional features are inherited by the subclasses Graduate and UnderGrad.
- If we suppose that the Graduate and UnderGrad use different algorithms to calculate the grades then these methods are redefined in these classes. This concept is known as Overriding.

AQA

Overriding and polymorphism

- A method that has been overridden in at least one subclass is therefore polymorphic.
- So polymorphism is really the mechanism used to select the most appropriate method for a particular object.
- In practical terms in OO languages like Java and Visual Basic, method calls are determined by the object (not a reference) therefore the appropriate method is performed.
- For example :
  - StudentX.CalcGrade()
  - GraduateY.CalcGrade()
  - UnderGradZ.CalcGrade ()

AQA

Practical Activity

Open the practical activity handout and try

Topics 5 and 6 – Polymorphism

The Basics and Overriding

AQA

Inheritance/overriding - task

7 An object-oriented program is being written to store details of the hardware devices that are connected to a computer network in a college. This will be used by the network manager to perform an audit of the equipment that the college owns.

Two different types of devices are connected to the network. They are printers and computers. The computers are categorised as being laptops, desktops or servers.

A class **Device** has been created and two subclasses, **Printer** and **Computer** are to be developed. The **Computer** class will have three subclasses: **Laptop**, **Desktop** and **Server**.

7 (a) Draw an inheritance diagram for the six classes.

AQA

Inheritance/overriding - task

7 (b) The **Device** class has data fields **MACAddress**, **DeviceName** and **Location**.

The class definition for **Device** is:

```
Device = Class
           Public
             Procedure AddDevice
             Function GetMACAddress
             Function GetDeviceName
             Function GetLocation
           Private
             MACAddress: String
             DeviceName: String
             Location: String
         End
```

The **Computer** class has the following additional data fields:

- **ProcessorName**: Stores the name of the company that manufactured the processor.
- **RAMCapacity**: Stores the capacity of the RAM installed in the computer, in gigabytes.
- **HDDCapacity**: Stores the capacity of the Hard Disk Drive installed in the computer, in gigabytes.

Write the class definition for **Computer**.

AQA

Inheritance/overriding - task

7 (c) The **Laptop** class has the additional data field **BluetoothInstalled**. This field will indicate whether or not the laptop is fitted with a Bluetooth module.

Write the class definition for **Laptop**.

Solution

AQA

Methods – Static, virtual and abstract

The class methods that we have been learning about so far are known as **instance methods**.

Instance methods include **constructors**, **accessors** (eg getRadius() and **mutators** (eg newHeight += Height).

Instance methods operate on individual objects, eg

MyCircle.Circumference(4)

Student.ComputeGrade()

- **Static methods** - are those which will perform and operation for the **whole class**. In Visual Basic we can use the **Shared** keyword to achieve this.

AQA

Methods – Static, virtual and abstract

- **Virtual methods -** are those that can be overridden. In Visual Basic when we define our base classes, unless otherwise stipulated, our methods can be overridden in subsequent derived classes.
- In order to prevent a method from being overridden we can use the **NotOverridable** Keyword.
- **Abstract methods** - this kind of method has no implementation, just a header. It will appear in an abstract Base Class just to ensure completeness. The method will have to be overridden in each of the derived classes.

AQA

Methods – Static, virtual and abstract

For example

In the example the Card class is **abstract**. We cannot make an instance of card but we can instantiate a Birthday card or a Christmas card from the derived classes.

AQA

Practical activity

Open the practical activity handout and try

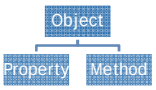
Topic 7 – Static, Virtual and Abstract Methods

AQA

UML and class diagrams

- Class diagrams are used to depict the classes within a model or design for a program and they originate in UML or Unified Modelling Language.

So why do we use UML ?



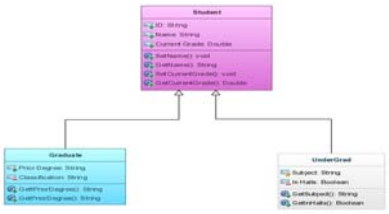
- Helps us to express program design graphically
- Facilitates use of abstraction
- Is the de-facto standard for OO design

There are many types of UML diagram but for A-level we need to draw and interpret class diagrams.

AQA

Class diagrams

- A class diagram depicts classes and their interrelationships.
- They are useful for describing structure and behaviour.
- Provide a conceptual model of the system in terms of classes and their relationships.



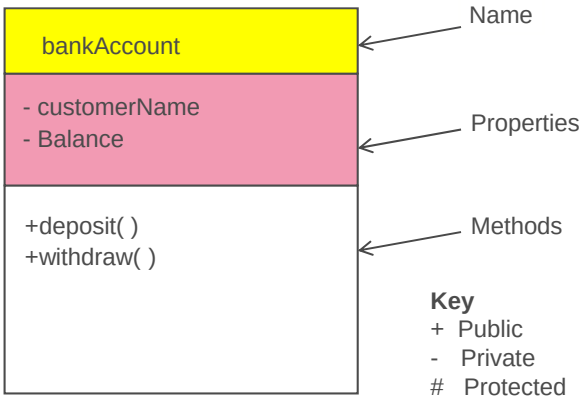
AQA

Class diagrams

- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Properties (Data)
  - Methods (Procedures and Functions)
- Modifiers are used to indicate visibility of attributes and operations.
  - **'+'** is used to denote *Public* visibility (everyone)
  - **'#'** is used to denote *Protected* visibility (friends and derived)
  - **'-'** is used to denote *Private* visibility (no one)

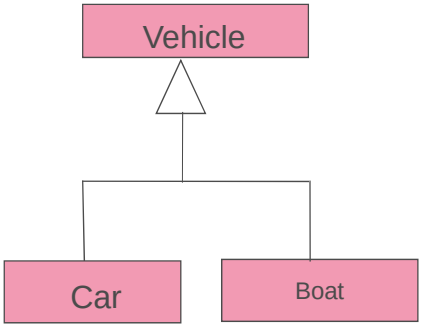
AQA

Class diagram – the basics



AQA

Generalisation



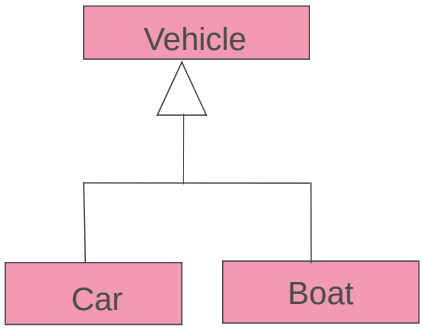
Generalisation is the process of creating a superclass by combining properties and methods from two or more subclasses.

In this case, we would look at common characteristics of Boats and Cars.

Generalisation is rather like inheritance “upside down”.

AQA

Specialisation

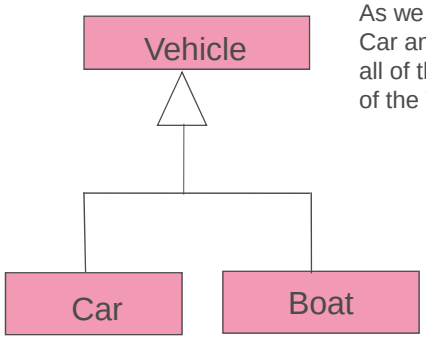


Specialisation is the process of creating a subclasses by refining the properties, methods and associations from the superclass.

In this case, we would look at what makes Boats and Cars specific vehicles.

AQA

Relationships - inheritance



As we have previously seen, the Car and Boat subclasses inherit all of the properties and methods of the Vehicle superclass.

AQA

Multiple inheritance

- Some languages, such as C++, Python, Perl and LISP support multiple inheritance where a subclass can inherit from 2 or more superclasses.
- This can cause issues during compilation and memory allocation as well as giving the programmer a headache!
- Fortunately, Visual Basic, Java and C# do not support it and knowledge of MI is not required for A-level.

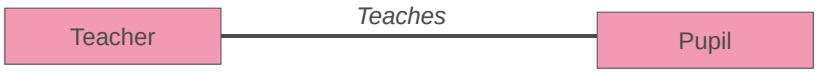
AQA

Relationships - association

If two classes in a model need to communicate with each other, there must be relationship between them.

An *association* denotes that link or relationship.

Association Diagram

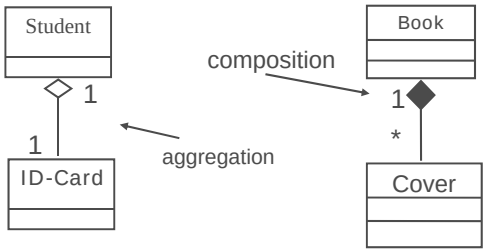


AQA

Association symbols

- **Aggregation**:
  This is symbolised by a clear unfilled diamond
  - The destruction of one object does not result in the destruction of the other object in the relationship

- **Composition**:
  This is symbolised by a black filled diamond
  – this is a strong version of aggregation as the parts live and die with the whole



Multiplicity values (How many objects will exist at each end of relationsip)

- * = 0, 1 or more
- 1 = exactly 1
- 2..5 = between 2 and 5
- 5..* = 5 or more

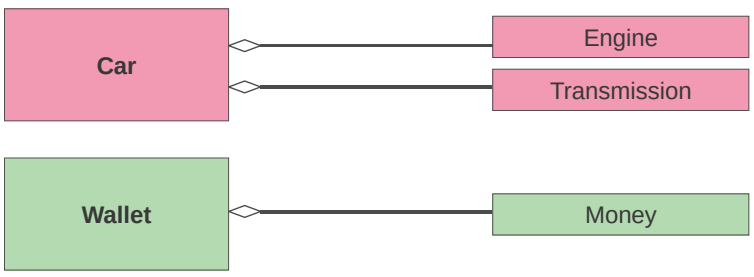
AQA

Aggregation

- We can model objects that contain other objects by way of special associations called **aggregation** and **composition**.

- **Association Aggregation** is when an object which contains other objects is destroyed, the other objects will still exist. These associations are denoted by a hollow-diamond/arrow on the association.

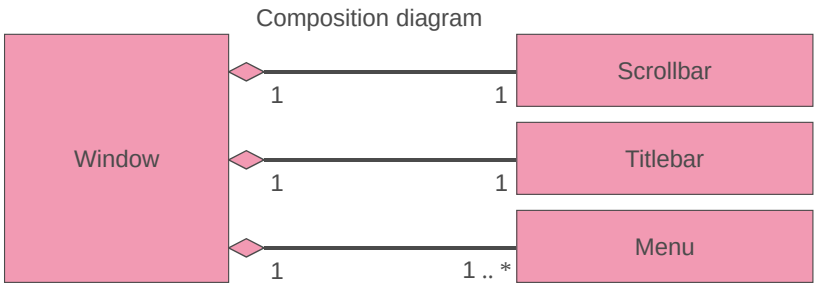
Association Aggregation Diagram



AQA

Composition

A ***composition*** indicates a strong ownership and objects are created and destroyed as a whole. Compositions are denoted by a filled-diamond on the association. **Notice** : the notation is as in E-R diagrams to denote 1 to 1 and 1 to many relationships in this example.



AQA

Task 1

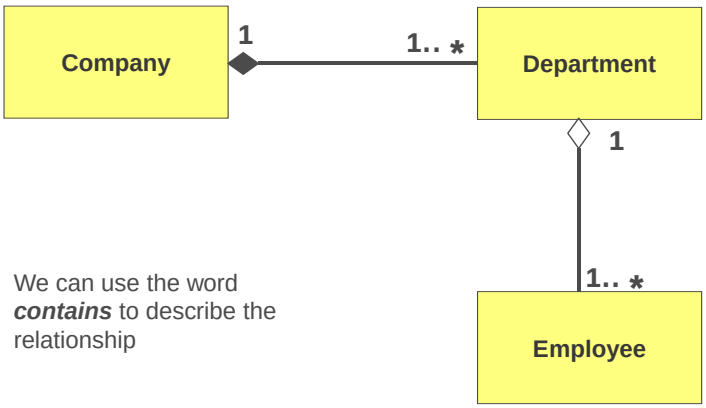
1. A small electrical company has engaged a number of departments. Each department contains a number of employees.

   We have 3 classes Company, Department and Employee.

   Draw a class diagram to show how the three are **related** (remember to use the correct symbols for **aggregation**).

AQA

Solution a)



We can use the word ***contains*** to describe the relationship

AQA

Task 2

2. The **Company** Class has the following properties
   (data fields) ***Name and Status***
   It also has the method ***MakeJob***

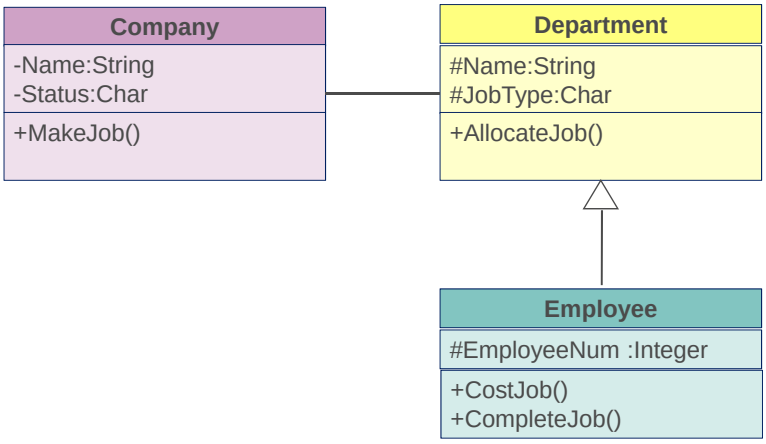
   The **Department** class and it has its own properties
   Name and ***JobType***
   It also has its own method ***AllocateJob***

   The **Employee** class inherits from **Department** and has its own property ***EmployeeNo***
   and its own methods ***CostJob*** and ***CompleteJob***

   Draw a class diagram showing inheritance and the access modifiers you think will be most appropriate for this scenario.

AQA

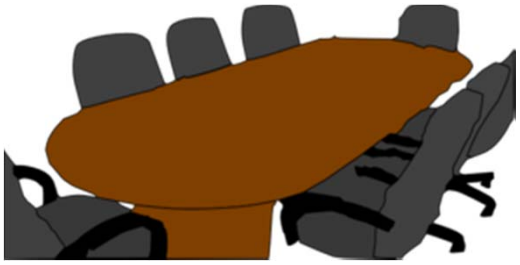
Solution



AQA

Aggregation – past paper question

- Find in your pack the past paper question on aggregation.

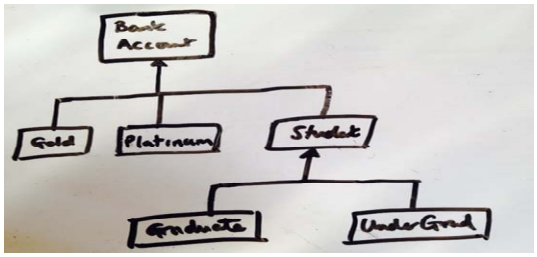


- Useful Link

AQA

Advanced Concepts - encapsulate what varies

- This is the idea that where there is a variance, then there should be a class for it.
- This is based on encapsulation and information hiding, so that the properties and methods are appropriate to the real world scenario they reflect.
- Looking at the Bank Account class for example, we could have Student, Gold and Platinum sub-classes.



AQA

Encapsulate what varies

- However, we could drill down further into say Student and have a **Graduate** account and **UnderGrad** account.
- These accounts may have their own unique properties and methods pertinent to them.
- This specialisation would continue until you were certain that a class had been created for each set of unique properties and methods.

AQA

Favour composition over inheritance

- We have looked at inheritance and aggregation and each has their merits.
- Inheritance allows objects to be created in the superclass and subclasses can inherit them.
- This allows code reuse and saves time.
- We do need to be very clear on relationships and classes when using inheritance however, as sometimes it can be prone to unintentional side effects when new classes are created and methods in the superclass are altered.

Useful Link - Composition

AQA

Favour composition over inheritance

- Aggregation allows us to use existing objects to combine new ones. In composition we can use relationships between classes rather than inheritance.
- Favour composition over inheritance- uses composition as classes are **easier to maintain and test,** they are often easier to understand in this format and this approach can also be less prone to errors.
- If inheritance is used, a base class cannot be tested separately from the derived class but composition allows each class to be tested separately.
- Look at the **Monster Specimen Skeleton Program** – the **Game** class combines existing objects in this way. (It instantiates Grid and Enemy but the classes can be dealt with individually).

AQA

Interface

- In object oriented programming, we define software objects that mimic "real world" objects.
- This makes programs easier to think about and more reliable. However, in the real world, we often think about an object in several different ways.
- An **interface** is a programming structure that allows the methods we want to use on our objects (classes) to be defined.
- This can be a list of properties and method signatures. There is no implementation of the methods (there is no method body).
- A class that implements an interface must implement each of the methods listed in the interface but can do this in each way that suits each class (object).
- Useful link

AQA

Interface

- A class can use **inheritance** to inherit the methods and properties of a superclass.
- A class can also **implement** an interface to gain additional methods and constants. However, the additional methods must be explicitly written as part of the class definition (as they have no body in the interface).
- The interface is a list of requirements that the class definition must explicitly meet (using code, not through inheritance).
- It sometimes suggested that an interface is used where there is a **"has-a"** relationship as opposed to the **"is-a"** relationship used in inheritance.

AQA

Program to interfaces, not implementation

- This principle allows programs to be written based on the interface as opposed to each individual implementation of a class.
- This allows us to alter individual classes, perhaps updating a method or adding additional functionality. However, this is done with reference to the interface and therefore there is little or no effect on the other classes in the program.
- For example, if we have a Car class it may inherit from the Vehicle class. Inheritance then gives it all the methods and variables of Vehicle.

AQA

Program to interfaces, not implementation

- Cars however need to be insured and we could create an interface with properties and methods to cater for this.
- If car then implements the **Insurance** interface, then its definition should also include all the code for the methods in **Insurance**.

AQA

Practical Activity

Open the Practical Activity Handout and try

Topics 8 – Interfaces

AQA

Consolidation and plenary

New specification questions

- CS Paper 1 specimen questions based on the Monster Game
- You should now understand the specimen skeleton program and be confident to complete all of the questions on this that relate to OOP.

Any questions?

AQA

Consolidation and plenary

New specification additional questions

- Look at Q3 about the Estate Agents
- This will test your knowledge of inheritance and aggregation including association and composition.

Any questions?

Learning review

Plenary

- Have your learning needs been met?
- Reflect on the most significant learning from today
- Think about how you will teach these concepts to your students
- Please complete the evaluation form

Thank you

Contact points

Contact us

aqa.org.uk/contact-us

Customer Support Team

0161 957 3980

computerscience@aqa.org.uk

Events Team

0161 696 5994

events@aqa.org.uk

aqa.org.uk/professional-development