AQA Computer Science A-Level

4.1.2 Programming paradigms

Concise Notes

Specification:

4.1.2.1 Programming paradigms:

Understand the characteristics of the procedural and object-oriented programming paradigms, and have experience of programming in each.

4.1.2.2 Procedural-oriented programming:

Understand the structured approach to program design and construction.

Be able to construct and use hierarchy charts when designing programs.

Be able to explain the advantages of the structured approach.

4.1.2.3 Object-oriented programming:

Be familiar with the concepts of:

- class
- object
- instantiation
- encapsulation
- inheritance
- aggregation
- composition
- polymorphism
- overriding

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

Be able to write object-oriented programs

Be able to draw and interpret class diagrams

The procedural programming paradigm

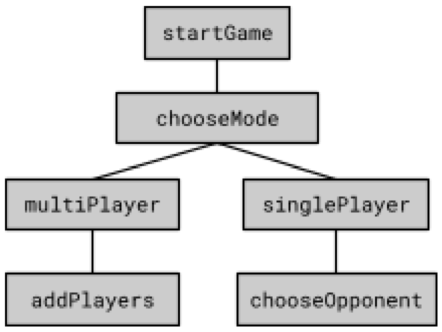
- Procedural programs are formed from sequences of instructions
- Instructions are executed in the order in which they appear
- Procedures form parts of the program and can be called from anywhere within the program, by other procedures or recursively
- Data is stored in procedural programs by constants and variables
- A data structure is said to have a global scope if it can be accessed from all parts of the program
- A data structure is said to have a local scope if it is only accessible from the structure within which it is declared

The structured approach to program design and construction

- Keeps programs easy to understand and manage
- Four basic structures are used:
  - Assignment
  - Sequence
  - Selection
  - Iteration
- Structured programs are said to be designed from the top down
- The most important elements of a problem are broken down into smaller tasks, each of which can be solved in a block of code such as a procedure or module which goes on to form part of the overall solution
- Makes maintaining the program easier as navigation of different elements of the overall solution is improved
- Testing can be carried out on the individual modules before they are combined to form the overall solution
- Development can be split over a team of developers each of which is assigned a different module to work on

Hierarchy charts

- Graphically represent the structure of a structured program
- Each procedure is displayed as a rectangle which is connected to any other procedures that are used within it
- Lines between the rectangles show the relationships that exist between the different parts of the program



The object-oriented programming paradigm

Objects
- Containers of both data and instructions
- Created from classes in a process called instantiation
- Defined as an instance of a class

Classes
- Blueprints for objects
- Specify what properties (data) and methods (instructions) objects of their type will have
- Can be expressed on paper as class definitions

Class definitions
- List a class' name, properties and methods in text form
- Independent of any particular programming language
- A method or property that is listed as private can only be accessed from within an object
- Public methods allow an interface for accessing and modifying a class' private properties

```
Car = Class {
  Private:
    Manufacturer: String
    Model: String
    EngineCapacity: Float
    IsTaxed: Boolean
  Public:
    Function GetManufacturer
    Function GetModel
    Function GetEngineCapacity
    Function GetIsTaxed
    Procedure SetDetails
}
```

Encapsulation
- The name given to the process of combining methods and procedures to form an object in object-oriented programming
- An object is said to encapsulate its contents, forming a single entity which encompasses all of the object's properties and methods
- Allows the development of large programs to be split across a team of developers

Inheritance

- Allows one class to share the properties and methods of another class
- Classes which use inheritance can have their own properties and methods too
- Can be described as an “is a” relationship
- Shown in class definitions by the name of the inherited class featuring in brackets as highlighted below

```
DeLorean = Class (Car) {
  Private:
    ReactorOutput: Integer
    FluxCapacitorInput: Integer
  Public:
    Function GetReactorOutput
    Function GetFluxCapacitorInput
    Procedure SetDetails (Override)
}
```

Polymorphism

- Comes from the Greek for “many forms”
- Occurs when objects are processed differently depending on their class

Overriding

- An overridden method has the same name as a method in an inherited class but different implementation
- The word override is used in class descriptions (including the one above) to indicate that the implementation of a method in one class differs from the implementation of the method with the same name in the inherited class

Association

- Two objects that are associated can be described as having a “has a” relationship
- For example, objects of the classes `Car` and `Driver` could be associated as a car **has a** driver
- An associated object forms part of its container object as a property
- There are two specific types of association:
    - Aggregation
    - Composition
- **Aggregation** is the weaker of the two kinds of association
- When an object is associated with another by aggregation, it will still exist if its containing object is destroyed
- **Composition** is a stronger relationship between classes
- If two objects are associated by composition and the containing object is destroyed, the associated object is also destroyed

Why is object-oriented programming used?

- Object-oriented programming provides programs with a clear structure
- This makes developing and testing programs easier for developers
- Using the paradigm allows for large projects to be divided among a team of developers
- The use of classes allows code to be reused throughout the same program and even in other programs
- This improves the space efficiency of code

Object-oriented design principles

- There are three design principles used in object-oriented programming that you need to be aware of:
  - Encapsulate what varies
  - Favour composition over inheritance
  - Program to interfaces, not implementation

Encapsulate what varies

- Any requirements which are likely to change in the future should be encapsulated in a class
- This way, any future changes can be easily made when required

Favour composition over inheritance

- Wherever possible, composition should be used over inheritance
- Composition is seen as a more flexible relationship between objects
- Composition isn't always appropriate and inheritance should still be used in any such situations

Program to interfaces, not implementation

- Allows unrelated classes to make use of similar methods
- An interface is defined as a collection of abstract procedures that can be implemented by unrelated classes
- When a new object is created, it can implement an interface which provides it with the correct properties and methods

Writing object-oriented programs

You should have experience of writing object-oriented programs in your chosen programming language for the exam.

Abstract, virtual and static methods

Abstract methods are declared in abstract classes and do not have an implementation. They serve as a blueprint for methods that must be implemented by subclasses. Abstract methods enforce a contract for subclasses to provide specific method implementations, ensuring consistency across different subclasses.

Virtual methods are methods in a base class that can be overridden by derived classes. This allows subclasses to provide their specific implementation while maintaining a consistent method interface. Virtual methods facilitate polymorphism, enabling method behaviour to vary based on the object that invokes the method.

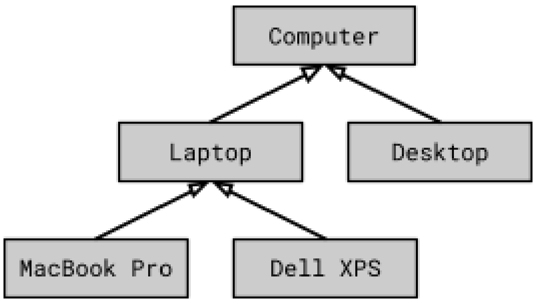
Virtual methods belong to the class itself rather than any instance of the class. They can be called on the class directly and do not require an instance to be invoked.

Class diagrams

- Visually represent the relationships that exist between classes
- Classes are represented by boxes
- Different connectors represent different kinds of relationship between classes

Inheritance diagrams

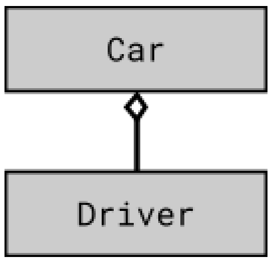
- Show the different inheritance relationships that exist between classes
- Inheritance is shown with unfilled arrows which point from an inherited class towards the class which inherits it
- Inheritance arrows should always point upwards



Association

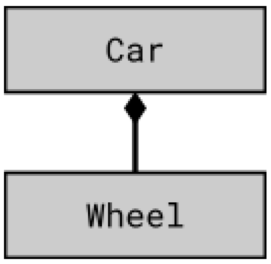
- Shown in class diagrams with diamond headed arrows

Aggregation



Shown with an unfilled diamond headed arrow

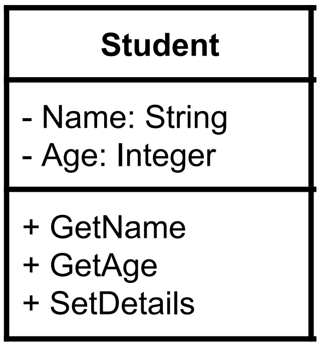
Composition



Shown with a filled diamond headed arrow

Class diagrams with properties and methods

- Class diagrams can contain more information about classes than just their name
- A class can be represented by three boxes:
  - The uppermost box contains the class name
  - The middle box contains the class' properties
  - The bottom box contains the class' methods
- A plus sign indicates that a property or method is public
- A minus indicates that the property or method is private
- A pound symbol (#) indicates that a property or method is protected
- Protected properties and methods are accessible from within the object that declares them and also from any inherited objects


Student
- Name: String
- Age: Integer
+ GetName
+ GetAge
+ SetDetails