# OCR A Level Computer Science

Your notes

## 2.5 Object Oriented Languages

### Contents

Your notes

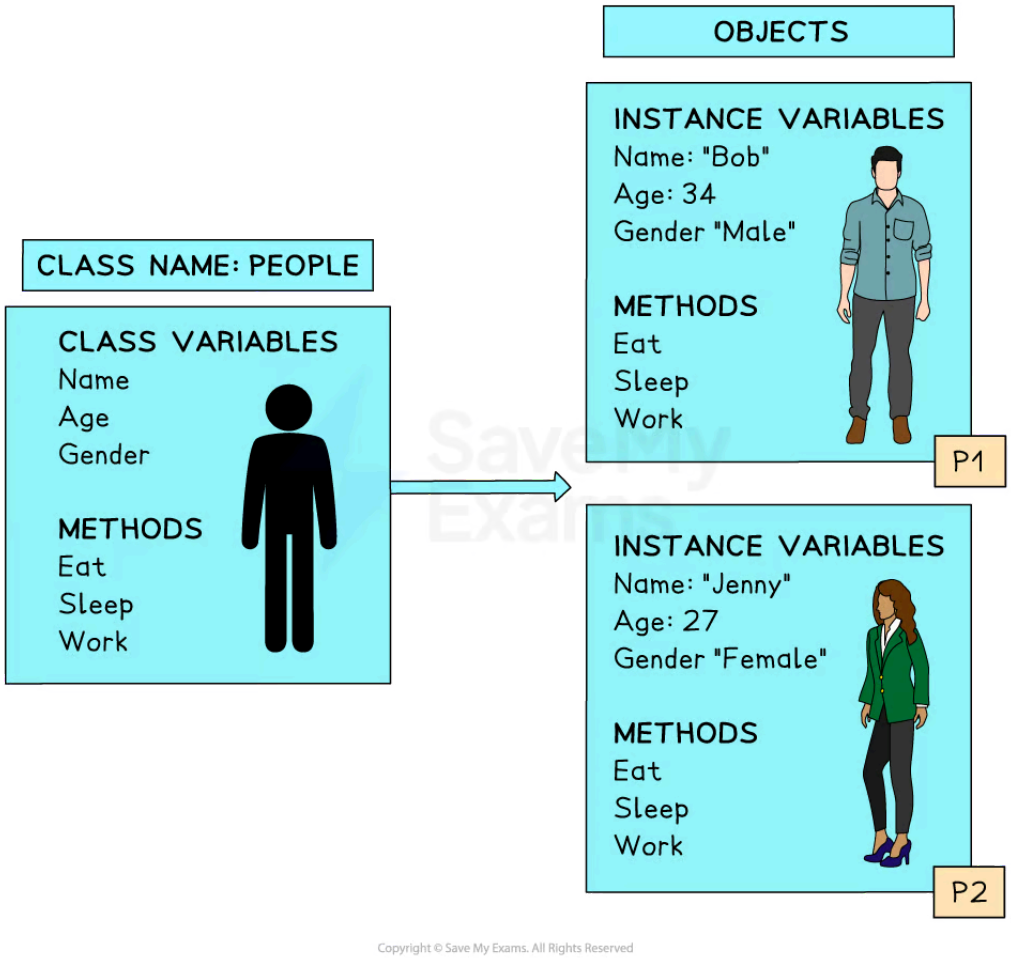
# Classes (OOP)

## What is a Class?

- **Classes** are used as blueprints or templates that can be used to create **objects** within Object Oriented Programming or OOP
- An object is created from a specific instance of a class and has its own **state** and **behaviours**
- Using this method allows for reusable and organised code in a modular way
- Consider a **class** of students
  - Each student has a name, a date of birth and gender
- Therefore we can create a blueprint or template for all students by making a class which contains these three **attributes**
- As each student (or object) is created they will each have their own name, date of birth and gender attributes along with their own state for example
  - Name "John"
  - Date of birth "06/10/2015"
  - Gender "Male"
- Some **classes** are already prebuilt into a programming language saving the developer from having to write them from scratch and often provide common functionality
- Examples from Java include:
  - Date and calendar when working with dates
  - String when working with strings of text
  - Random when generating random numbers
  - Scanner when reading input from a user or file
- Custom classes are created by the programmer to define new data types
- For example, a class for animals does not exist and so the programmer must define a custom class
- **Instantiation** is the term used for creating an **object** from a class

- Each class contains **attributes** which are essentially variables within a class and are also known as Class Variables
- **Objects** that are created from a **class** contain attributes which are also known as instance variables
- Classes can also contain **methods**/functions/procedures
- **Methods** are actions or behaviours that can be performed
- The name that is used to refer to an **object** is known as the **identifier**
- Below is a visual representation of both a **class** and **objects** that have been **instantiated**
- In the image below, the identifiers are P1 and P2

Your notes






*Example of a class and objects*

Your notes

## Examiner Tips and Tricks

- Although you may see the terms **methods**/functions/ procedures, a method can be either a function or a procedure
- A function is a method that must return a value
- A procedure is a method that does not need to return a value

## Worked Example

**What is a class in object-oriented programming (OOP) ?**

[2]

**How to answer this question:**

- Classes provide a way to organize code in a modular way [1]
- A blueprint or template for creating objects with specific attributes [1]

**Answer:**

**Example answer to get full marks:**

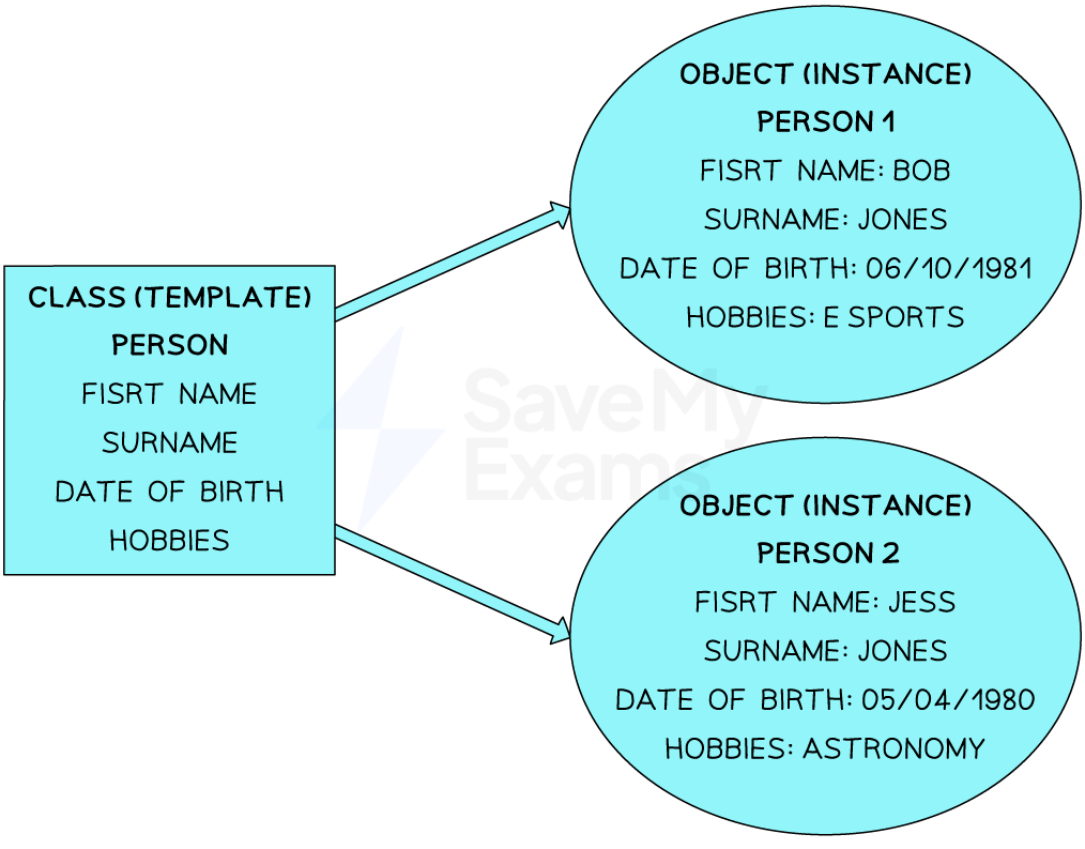
A class is acts as a blueprint or template for creating objects with specific attributes [1 mark], while also providing a way to organize code in a modular fashion. [1 mark]

Your notes

# Objects (OOP)

## What is an Object?

- An **object** is a representation of a real-world entity eg teacher, aeroplane, mobile phone, cat etc
- A **class** is like a blueprint that describes the properties and behaviours of objects, while an **object** is a specific instance created based on that blueprint with its own unique values for the properties
- A constructor is a special **method** within a class that is automatically called when an **object** of that class is created (**instantiated**)
- **Constructors** typically define the initial values of **instance variables** and perform any necessary setup to prepare the **object** for use



*Example of 2 objects belonging to a class*

Your notes

## Worked Example

**A supermarket uses an object-oriented approach to organise items that it offers for sale. Part of the class definition for the ItemForSale class is shown below:**

```
class ItemForSale
  public itemName
  public price
  public discount
```

**Write a line of code to create an object of type** ItemForSale **called** mushypeas **that has a name of "mushy peas" and a price of £0.89**

[3]

**How to answer this question:**

- Creating object with identifier
  - mushypeas =
- creating object as type
  - ItemForSale
- mark for parameters passed in as needed ("mushy peas",0.89) [1 mark]

**Answer:**

**Example answer 1 to get full marks:**

```
mushypeas=new ItemForSale("mushy peas", 0.89)
```

**Example answer 2 to get full marks:**

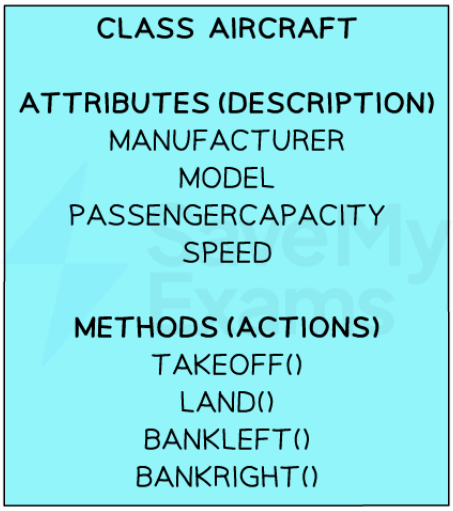
```
ItemForSale mushypeas = ItemForSale("mushy peas",0.89);
```

Your notes

# Methods (OOP)

## What is a Method?

- **Methods** are fundamental in **object-oriented programming (OOP)** and are functions associated with **objects** or **classes** that define the behaviour and actions that objects can perform
- There are two main types of methods:
  - **A function**
    - A function performs a task and returns a value
  - **A procedure**
    - A procedure performs a task buts does nor return a value
- For example, there are many different types of aircraft which all differ in design, but all require a 'take off' and 'landing' method
- An example of an aircraft **class** with both **attributes** and **methods** is shown below:






***Example class for "aircraft" containing several methods***

- Each **object** that is created from the aircraft class will have:

- A manufacturer value to determine the company that created the aircraft
- A model name for the type of aircraft
- A value for passenger capacity to determine how many people it can carry
- A speed value to determine its maximum speed

- Any **objects** that are created for the aircraft class also have access to the following Methods:
  - A take off method to get the plane airborne
  - A land method to land the plane safely
  - A bank left method to allow the plane to turn to the left
  - A bank right method to allow the plane to tur to the right
- As you can imagine, for an aircraft there would be many more methods that could be implemented such emergency landing and altitude cruising actions
- Once **objects** have been created, they can use the **methods** from within their **class** by using the dot (.) notation
- As these methods are associated with objects, they are called instance methods
- For example if an **object** has been created as below:
  - jumboJet = new aircraft ("Boeing", "747", 416, 547)
- It can use methods by doing the following:
  - Objectname.Methodname:
  - For example: jumboJet.Takeoff()

Your notes

## Examiner Tips and Tricks

- In some OOP languages there are both instance methods and static methods
  - Instance methods
    - Instance methods are associated with individual instances (objects) of a class. They operate on the specific data and properties of an object (see above on the Jumbo Jet object example)
  - Static methods
    - Static methods are associated with a class itself and can be called without creating an instance (object) of the class

# Public and private methods

Your notes

- When declaring **methods**, it is important to determine how they can be accessed:

| Public Methods | Private Methods |
|---|---|
| Accessible and can be **invoked** by any code within the same class or from any external classes | Private methods are only accessible within the same class and cannot be invoked by external code or other classes |
| Changes to public methods may have an impact on other parts of the codebase, so they should be carefully designed, documented, and backward compatible whenever possible | Changes to private methods have a localized impact since they are only used internally within the class, providing flexibility to modify or refactor them without affecting other parts of the program |
| Public methods are used when you want to provide access to certain functionalities or behaviours of an object or class to other parts of your program | Private methods are used when you have internal implementation details that should not be accessed or used by external code. They are meant to be used only within the class itself for organizing and managing the code internally |

## Examiner Tips and Tricks

- If a method does not specify the keyword public or private then its default value is set to public.

## Worked Example

**An object oriented system is implemented to organise further information about each worker's attendance. Classes, objects, methods and attributes are used in this system.**

**State the meaning of a method**

[1]

**How to answer this question:**

- A clear definition of a method

- The purpose of methods in OOP

**Answer:**

**Example answer to get full marks:**

A method is a function that belongs to a class and operates on its instance data, allowing objects to perform actions and behaviours. **[1]**

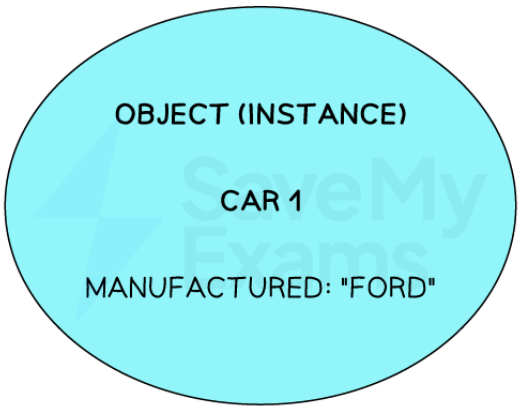

Your notes

Your notes

# Attributes (OOP)

## What is an Attribute?

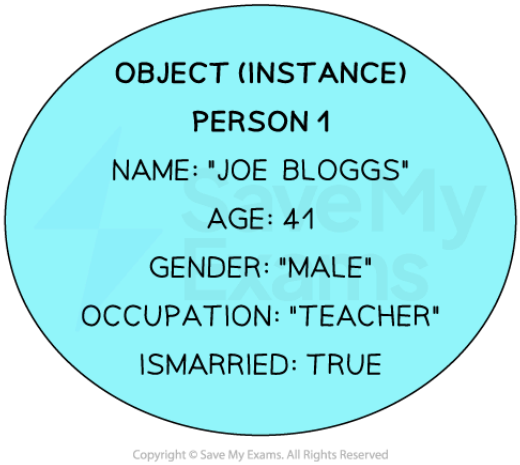
- In object-oriented programming (OOP), an attribute refers to a data member or a property associated with an **object** or a **class**
- They define the state of an object and can have different values for different instances of the same class
- Attributes can be of various data types, such as integers, strings, Booleans, or even other objects
- Attributes can have different access rights
- The example below shows a Car class object with an attribute called manufacturer
- It has a private access meaning that it can be accessed only by instances of the Car class
- The data that this attribute will hold must be of the String data type
- The image below gives a visual representation of an object of this class being instantiated with a data value of “Ford” :






***An example instance of an object***

- In most cases each class has many different attributes
- Below is an example of an object of class "person":





*Example of an object of class "person"*

Your notes

## Examiner Tips and Tricks

- Attributes declared within methods (local variables) cannot have access modifiers because they are local to the method and have a limited scope
- Local variables are only accessible within the block or method in which they are declared. They are not part of the class's state and cannot be accessed from other methods or classes

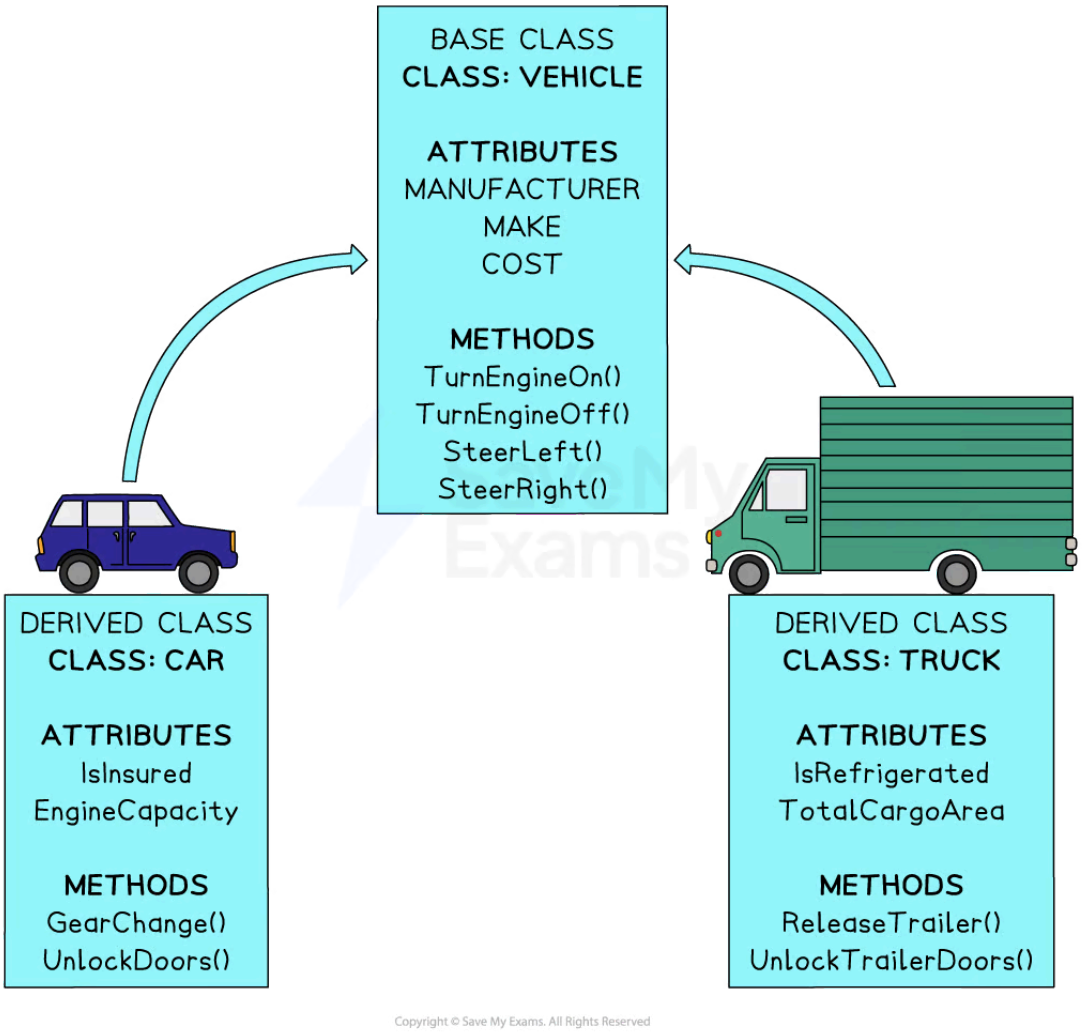
Your notes

# Inheritance (OOP)

## What is Inheritance?

- **Inheritance** is a key concept in **object-oriented programming (OOP)** that allows a **class** to inherit the properties and behaviours (methods and attributes) of another class
- **Inheritance** promotes code reuse by allowing **derived classes** to inherit and utilise the existing code from the **base class**. This avoids duplicating code and promotes better organization and maintainability
- **Inheritance** establishes an **"IS-A" relationship** between the base class and the derived class
- For example, if you have a base class called Vehicle and a derived class called Car, you can say that "a Car is a Vehicle."
- The car class **inherits** the properties and behaviours associated with being a vehicle
- Inheritance involves two main entities:
  - The **base class** (also known as the parent class or superclass) and the derived class (also known as the child class or subclass)
  - The **derived class** inherits the characteristics of the base class, meaning it can access and use the methods and attributes defined in the base class

Your notes





***Example of a base class and derived classes***

- **Base Class**: The base class serves as the blueprint or template from which the derived class inherits
  - It defines common properties and behaviours that can be shared among multiple derived classes
- **Derived Class**: The derived class inherits the attributes and methods of the base class
  - It can add additional attributes and methods
- If a car object was to be created, it may have the following attributes:
  - Manufacturer - The company that makes the car

- Make -The model of the car
- Cost – The price of the car to purchase
- IsInsured – Whether or not the car is insured
- EngineCapacity – The size of the engine for the car

- It may also have access to the following **methods**:
  - TurnEngineOn() – To start the car engine
  - TurnEngineOff() – To turn off the car engine
  - SteerLeft() – To turn the car to the left
  - SteerRight() – To steer the car to the left
  - GearChange() – To change the gear of the car
  - UnlockDoors() – To unlock the doors to the car
- The above **methods** are only a select few and there could be many more added for extra functionality
- In the following code, the **super keyword** is used in inheritance to refer to the superclass (Base class: Vehicles) and access its members (methods, attributes, or constructors) from within the subclass (Derived Class: Cars)

Your notes

## Worked Example

**The classes office and house inherit from building.**
**Describe what is meant by inheritance with reference to these classes.**

[2]

**How to answer this question**:

- 1 mark per bullet up to a maximum of 2 marks, e.g:
  - When the child/derived/subclass class office/house takes on attributes/methods…
  - … from building / parent/base/superclass/ class

**Answer:**

**Example answer to get full marks:**

When the derived classes "office" and "house" inherit attributes/methods **[1]** from the "building" base class, they gain access to the properties and behaviours defined in the "building" class. **[1]**

Your notes

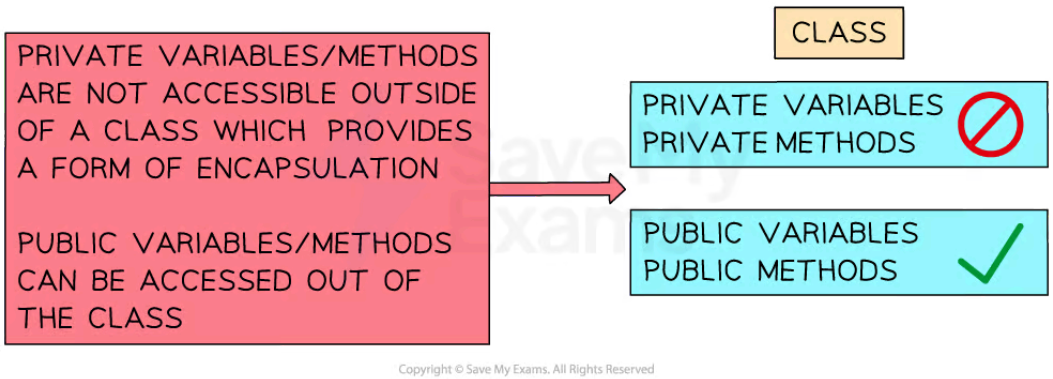
# Encapsulation (OOP)

## What is Encapsulation?

- Encapsulation in object-oriented programming is the practice of bundling data (attributes) and methods (functions) together within a **class**
- Using encapsulation ensures that data remains secure and is not accidentally modified or misused by controlling access to them using access modifiers (e.g., public, private)
- It also helps to organize code by keeping related data and methods together within an **object**
- Encapsulation promotes code reusability, which means the same object or class can be used in different parts of a program without rewriting the code
- **Encapsulation** uses a concept called **"Abstraction"** which reduces complexity by hiding the implementation details of the object, making it easier to understand and work with
- Programmers can use **methods** and **classes** from other parts of the program without having to understand how that it has been constructed internally

## Encapsulation in Classes

- Private variables are only accessible within the class itself, and external code cannot access them directly.
- **Encapsulation** hides how things work inside a class from the outside. External code can interact with the class using public methods without needing to understand its internal details

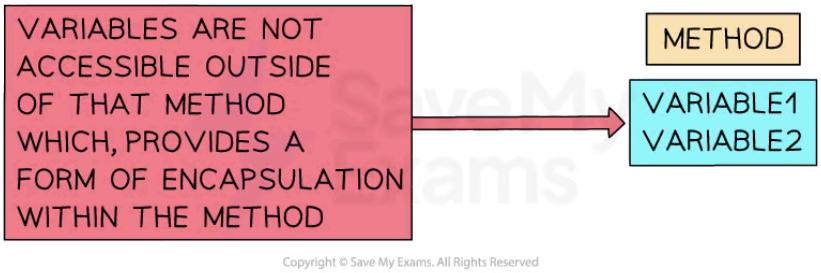




*Encapsulation in classes*

# Encapsulation in Methods

Your notes





*Encapsulation in methods*

## Examiner Tips and Tricks

- When determining whether a method or attribute is public or private, if neither keyword appears, then assume it is public

## Worked Example

**A taxi firm is investigating replacing its drivers with self-driving cars.**

**The code for the self-driving system has been written using an object-oriented programming language.**

**It recognises obstacles in the road and then classifies them.**

**The class for** Obstacle **is shown below.**

```
public class Obstacle

  private moving //Boolean value

  private distance //Real number given in metres

  private direction //Integer given as between 1 and 360 degrees


  public procedure new(givenMoving, givenDistance, givenDirection)
```

```
 moving=givenMoving
   distance=givenDistance
  direction=givenDirection
  endprocedure

public procedure updateDistance(givenDistance)
   distance=givenDistance
endprocedure
endclass
```

**Describe an example of encapsulation in the class definition code above.**

[2]

**How to answer the question:**

- Stating that distance is set to private [1 mark]
- To update the value the method updateDistance must be used [1 mark]

**Answer:**

**Example answer that gets full marks:**

An example of encapsulation in the class definition code above is demonstrated through the attribute "distance" being declared as private. **[1]**.

The "updateDistance()" method serves as a public interface to modify the "distance" attribute. **[1]**


Your notes


Your notes

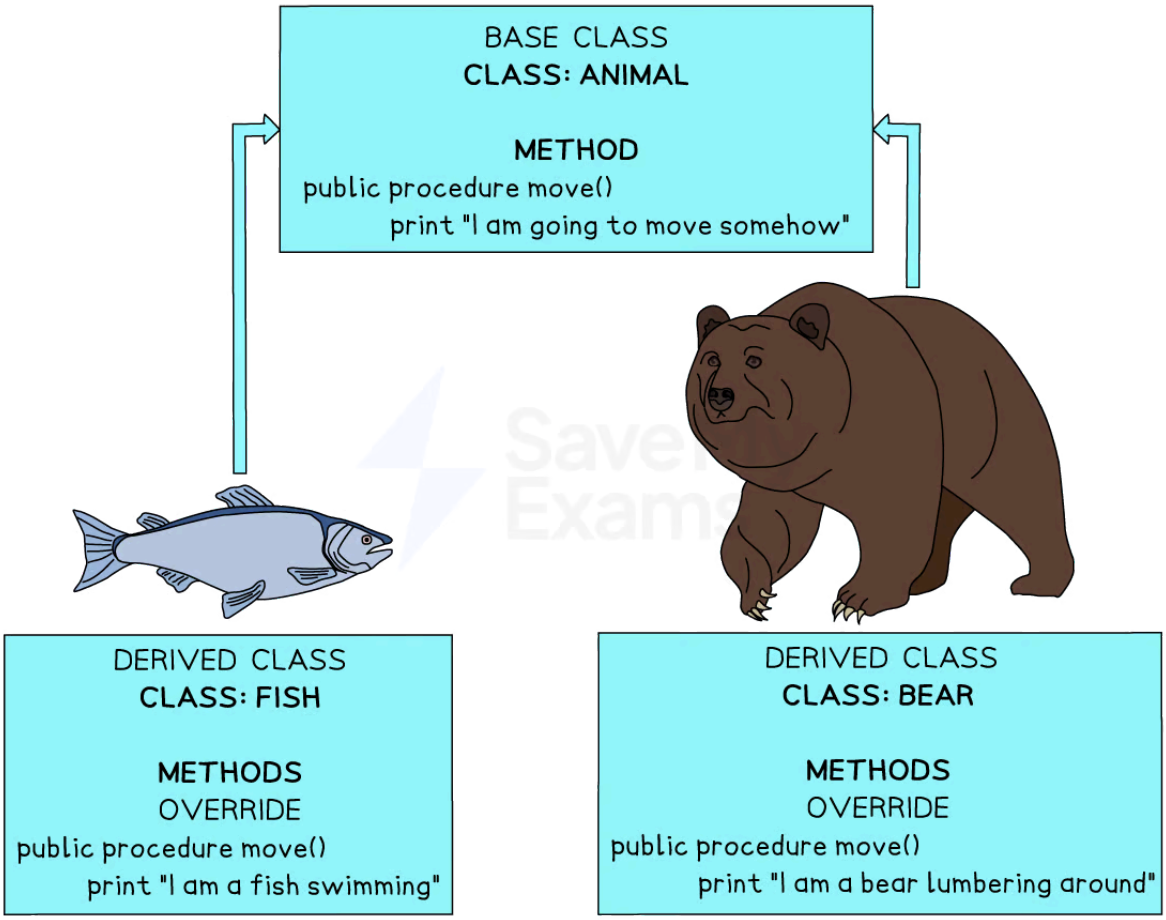
# Polymorphism (OOP)

## What is Polymorphism?

- **Polymorphism** is a concept in programming that allows **objects** to take on different forms or behaviours.
- Different **objects** can share the same name or behaviour but can work in different ways
- It helps make code more flexible, reusable, and easier to maintain
- It allows flexibility and reusability in programming, making it easier to write and manage code
- **Objects** can be treated as belonging to a common group, even if they belong to different **classes**, making your code more versatile and adaptable to changes

## Example 1 – Method Overloading

Your notes



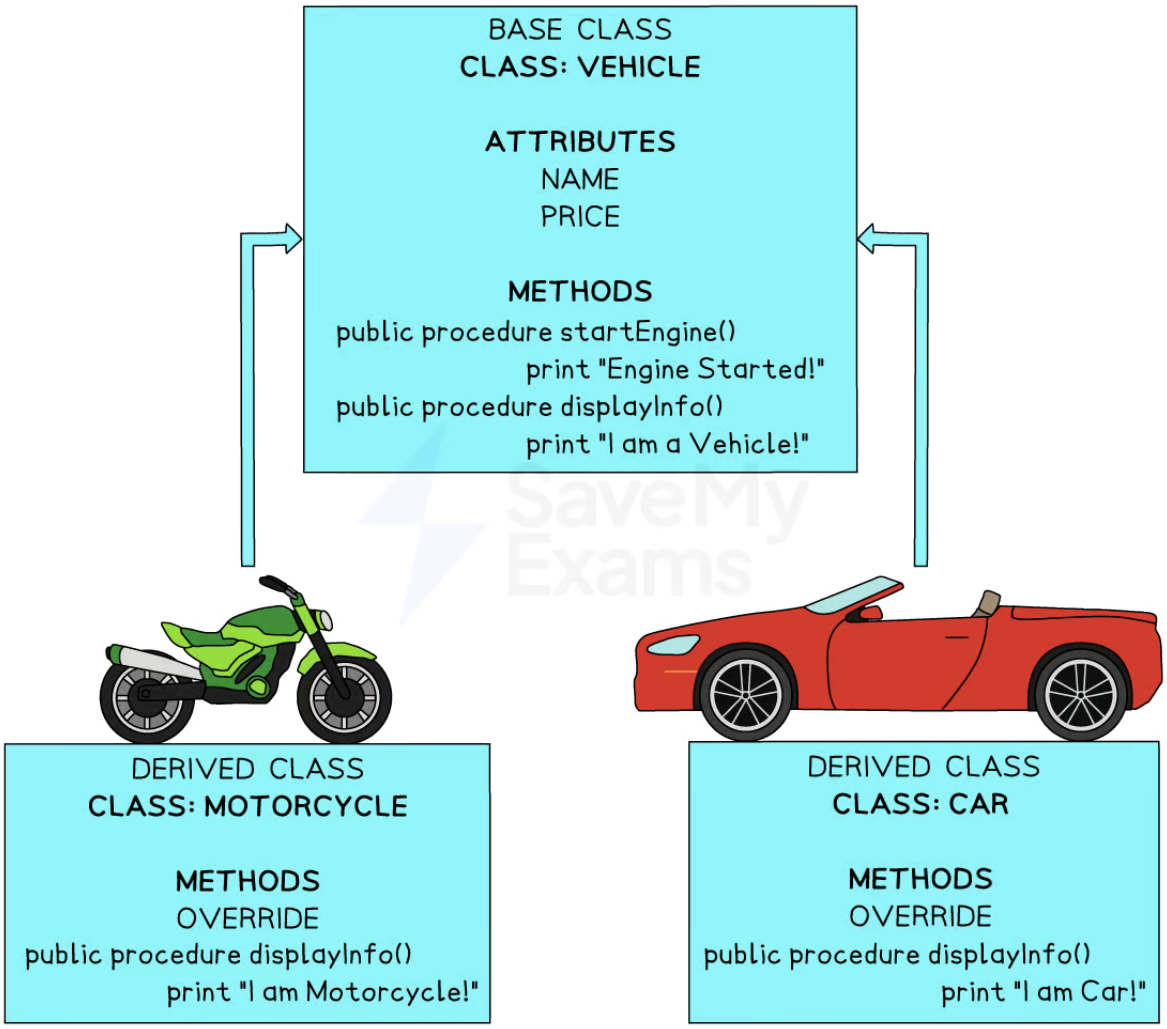


*Method Overloading Example 1*

- In the example above, all three **classes** all have a **method** named move(). **Polymorphism** allows methods to be declared with the same name but execute different code (in this case printing different messages)
- The **override** keyword Is used to provide a new **implementation** for a method that is already defined in the parent class (**base class**)

## Example 2 - Method Overloading

Your notes






***Method Overloading Example 2***

- In the above example both the Motorcycle class and the Car class both **inherit** from the base class 'Cars'
- **Objects** from the Motorcycle Class and the Car class can call the startEngines() **method** which will output "Engines Started!"
- If either of the object types call the displayInfo() method, the program will execute the method from the objects class as it **overrides** the Vehicle class method
- For example

- If a motorcycle object calls the displayInfo() method, "I am a Motorcycle!" will be output
- If a Car object calls the displayInfo() method, "I am a Car!" will be output


Your notes

## Treating objects as common groups

- **Polymorphism** also allows objects of different classes to be treated as objects of a common **superclass** or **base class**
- For example:
  - Vehicle vehicle1 = new Car()
  - Vehicle vehicle2 = new Motorcycle()
- This allows an array of type Vehicle to store both Motorcycle and Car objects rather than in separate data structures
  - If the vehicle1.displayInfo() method is called, it will still output "I am a Car!"
  - If the vehicle2.displayInfo() method is called, it will still output "I am a Motorcycle!"
- This flexibility provided by **polymorphism** are essential for creating more maintainable and modular code