Ch 05 Polymorphism

What is 'polymorphism'
The word 'polymorphism' is derived from a Greek word ('polumorphos') with two roots:

Polus (many) + **Morphe (shape/form)** = **Polumorphos**

It therefore means 'many-shaped' or 'having many forms'.

When used in the context of object-oriented programming, polymorphism means that it is possible to 'overload' (use to mean more than one thing) the symbols or methods that we use in a program, so that the same symbol can have different meanings in different contexts. What do we mean by 'symbol'? In fact it encompasses two fundamental components of our program source code:

Operators
a + symbol is already polymorphic (overloaded)
it is used in arithmetic (to add numeric values) & with string in concatenation

Method names
in a wages program a CalcPay is a typical method that can apply equally to all employees but a salaried worker and an hourly-paid worker will have their wage payment calculated very differently.

In essence this means that the same symbol or function name can be used to apply to different processes. In this chapter, we will introduce the various types of polymorphism and attempt to categorise them into types applicable to the available facilities in OOP.

Example
As a general example of polymorphic responses to a single message, we might ask the same question of a number of people of different religious faiths. The question (i.e. the message we send) is the following:

What year is it?'

If you ask this question of, for example, a Christian, a Muslim and a Jew, you may get three different answers, because their calendars started counting at different points in history. No doubt there are many variations on calendars in the world, giving different answers to the same question depending on the cultural 'class' of the answering.

Something similar happens when a method is called in an object-orientated program. That method may have the **same name** in different classes, but the response to objects in each class may well be different.

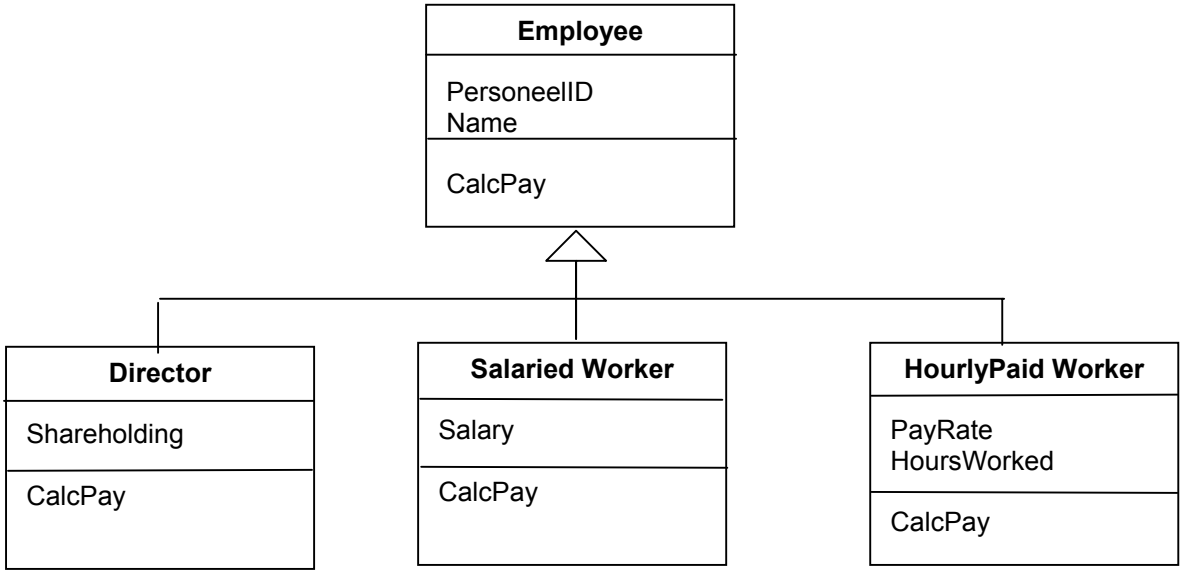
The important point is that we do not need different message to send to every different

type of object in our system. A message is enough; each object is capable of supplying its own response, defined by the class to which the object belongs.

In a program we might define a method called 'Print' in a number of classes, but each version of 'Print' would be specifically tailored to the class of the object being printed. An object of class 'Cheque' will respond to the message 'Print' in one way, an object of class 'Report' in another, a 'Photograph' object in yet another.

We do not have to define methods with entirely different names such as 'PrintCheque', 'PrintReport' 'PrintPhotograph' which would make our programs much less flexible. Because objects themselves take responsibility for their different responses to the same message, we are free to send generic messages to any set of objects with similar but class specific behaviours.

Example –Payroll System



CalcPay is polymorphic
It is defined in each of the three sub classes.
In each class the method used for calculating will be different:
Directors pay will be based on shareholding.
Salaried worker pay is based on Salary.
HourlyPaid Worker pay is based on PayRate & HoursWorked.

Exercise

1) Create the program based on the class diagram for the wage calculations.

The Director method for Calc Pay might be:

```
Function CalcPay() as decimal
        Dim Payment as Decimal
         Payment = Shareholding * Dividend
         Return Payment
End Function
```

The Salaried Worker method for CalcPay might be:

```
Function CalcPay() as Decimal
        Dim Payment as Decimal
         Payment = Salary / 12
         Return Payment
End Function
```

The Hourly-Paid Worker method for CalcPay might be:

```
Function CalcPay() as Decimal
        Dim Payment as Decimal
         Payment = HoursWorked * PayRate
         Return Payment
End Function
```