

Syllabus Content:

20.1 Programming paradigms



Show understanding of programming paradigm



Show understanding of the characteristics of a number of programming paradigms (low-level, imperative (procedural), object-oriented, declarative) –



low-level programming

Notes and guidance

- understanding of and ability to write low-level code that uses various addressing modes: immediate, direct, indirect, indexed and relative



Imperative programming- (procedural programming)

Notes and guidance

- Assumed knowledge and understanding of Structural Programming (see details in AS content section 11.3)
- understanding of and ability to write imperative (procedural) programming code that uses variables, constructs, procedures and functions. See details in AS Content



Object-oriented programming (OOP)

Notes and guidance

- Terminology associated with OOP (including objects, properties, methods, classes, inheritance, polymorphism, containment (aggregation), encapsulation, getters, setters, instances)
- Solve a problem by designing appropriate classes
- Ability to write code that demonstrates the use of OOP



Declarative programming

- understanding of and ability to solve a problem by writing appropriate facts and rules based on supplied information
- understanding of and ability to write code that can satisfy a goal using facts and rules

Programming paradigm – a set of programming concepts.

Low-level programming – programming instructions that use the computer's basic instruction set.

Imperative programming – programming paradigm in which the steps required to execute a program are set out in the order they need to be carried out.

Object-oriented programming (OOP) – a programming methodology that uses self-contained objects, which contain programming statements (methods) and data, and which communicate with each other.

Class – a template defining the methods and data of a certain type of object.

Attributes (class) – the data items in a class.

Method – a programmed procedure that is defined as part of a class.

Encapsulation – process of putting data and methods together as a single unit, a class.

Object – an instance of a class that is self-contained and includes data and methods.

Property – data and methods within an object that perform a named action.

Instance – An occurrence of an object during the execution of a program.

Data hiding – technique which protects the integrity of an object by restricting access to the data and methods within that object.

Inheritance – process in which the methods and data from one class, a superclass or base class, are copied to another class, a derived class.

Polymorphism – feature of object-oriented programming that allows methods to be redefined for derived classes.

Overloading – feature of object-oriented programming that allows a method to be defined more than once in a class, so it can be used in different situations.

Containment (aggregation) – process by which one class can contain other classes.

Getter – a method that gets the value of a property.

Setter – a method used to control changes to a variable.

Constructor – a method used to initialise a new object.

Programming paradigms

Programming paradigm:

A programming paradigm is a set of programming concepts and is a fundamental style of programming. Each paradigm will support a different way of thinking and problem solving. Paradigms are supported by programming language features. Some programming languages support more than one paradigm. There are many different paradigms, not all mutually exclusive. Here are just a few different paradigms.


Low-level programming paradigm




The features of Low-level programming languages give us the ability to manipulate the contents of memory addresses and registers directly and exploit the architecture of a given processor.





We solve problems in a very different way when we use the low-level programming paradigm than if we use a high-level paradigm.


 Note that each different type of processor has its own programming language. There are 'families' of processors that are designed with similar architectures and therefore use similar programming languages.

 For example, the Intel processor family (present in many PC-type computers) uses the x86 instruction set.


Imperative programming paradigm


 Imperative programming involves writing a program as a sequence of explicit steps that are executed by the processor. Most of the programs use imperative programming (Chapters 11 to 15 and Chapters 23 to 26).


 An imperative program tells the computer how to get a desired result, in contrast to declarative programming where a program describes what the desired result should be.

 Note that the procedural programming paradigm belongs to the imperative programming paradigm. There are many imperative programming languages, Pascal, C and Basic to name just a few.

Object-oriented programming paradigm

 The object-oriented paradigm is based on objects interacting with one another. These objects are data structures with associated methods.

 Many programming languages that were originally imperative have been developed further to support the object-oriented paradigm.

 Examples include Pascal (under the name Delphi or Object Pascal) and Visual Basic (the .NET version being the first fully object-oriented version). Newer languages, such as Python and Java, were designed to be object-oriented from the beginning

Object Oriented Programming

Object-oriented programming (OOP) is a programming methodology that uses self-contained objects, which contain programming statements (methods) and data, and which communicate with each other.

This programming paradigm is often used to solve more complex problems as it enables programmers to work with real life things.

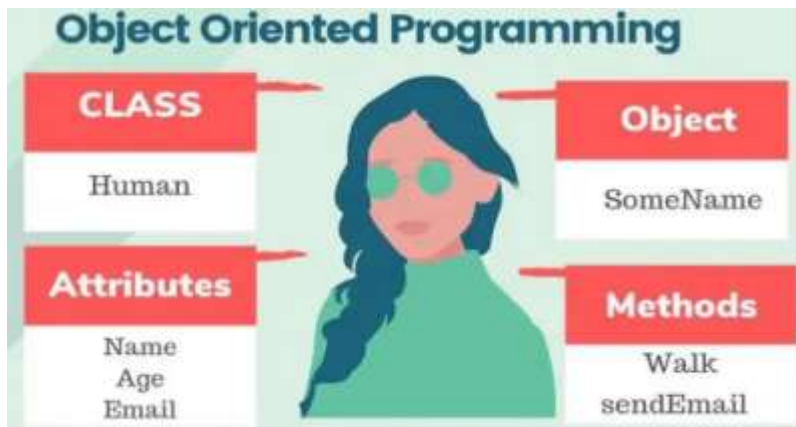
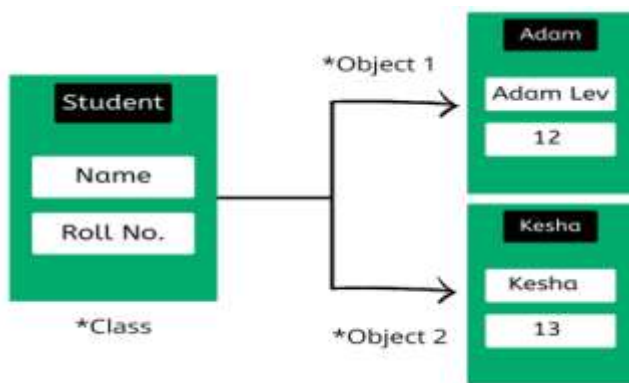
Many procedural programming languages have been developed to support OOP. For example, Java, Python and Visual Basic all allow programmers to use either procedural programming or OOP. Object-oriented programming uses its own terminology, which we will explore here.

Class

A class is a template defining the **methods** and **data** of a certain type of **object**.

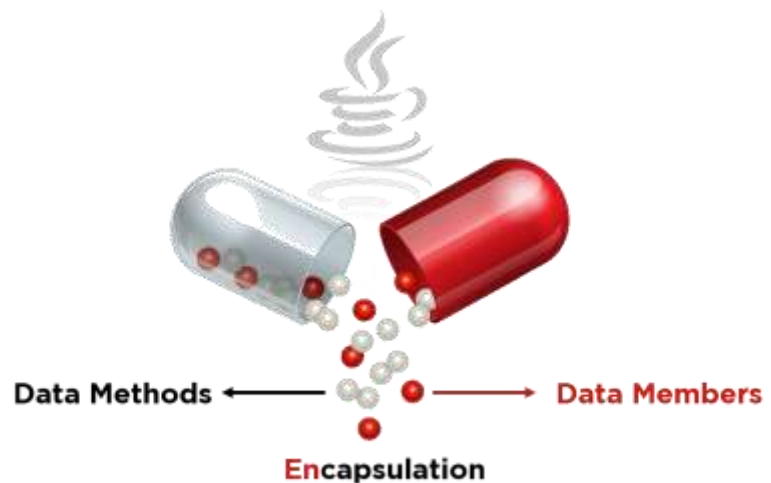
The **attributes** are the **data items** in a class.

A **method** is a programmed procedure that is defined as part of a class.



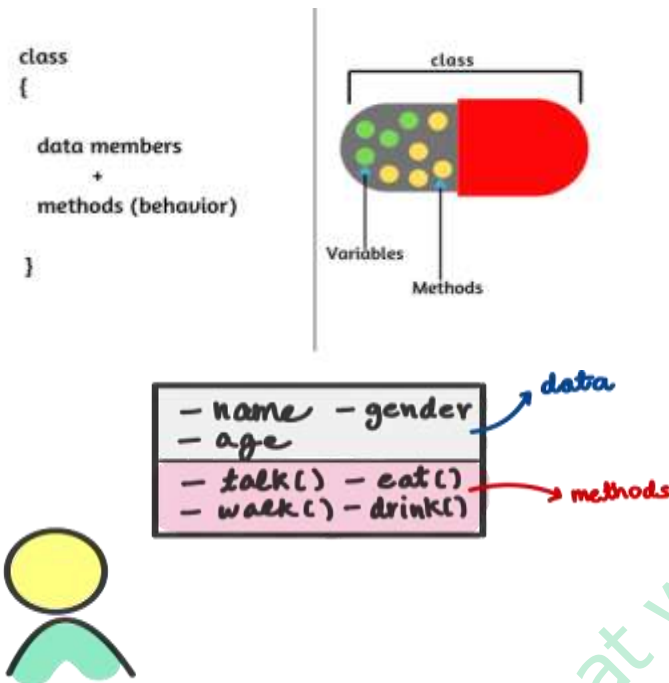
Encapsulation:

Putting **data** and **methods** together as a single **unit (class)**, is called **encapsulation**.



Encapsulation:

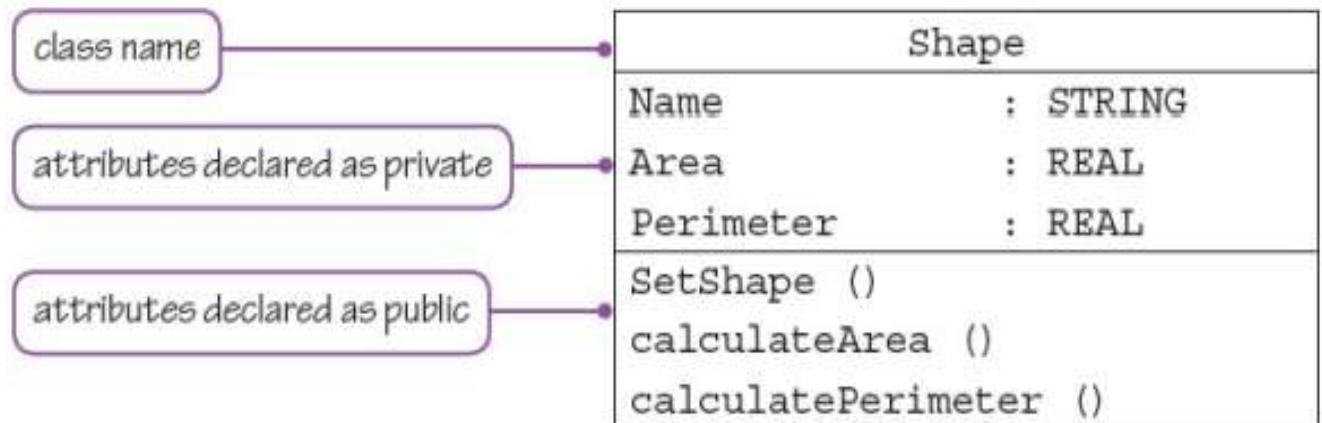
Putting **data** and **methods** together as a **single unit** in a class, is called encapsulation.



Private attributes & Public methods:

To ensure that only the methods declared can be used to access the data within a class, attributes need to be declared as **private** and the methods need to be declared as **public**.

For example a Class Shape can be defined by this **Class Diagram**

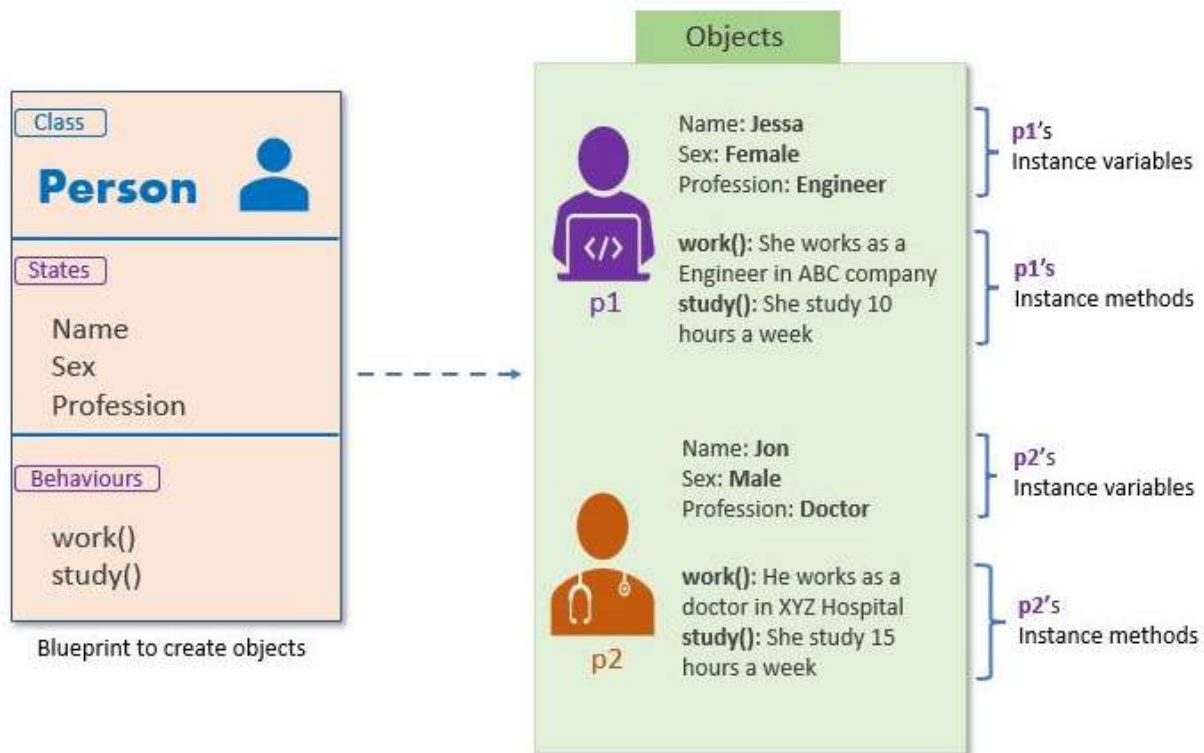


Object:

When writing a program, an **object** needs to be declared using a **class type** that has already been defined.

An **object** is an instance of a class that is self-contained and includes **data** and **methods**.

Properties of an object are the data and methods within an object that perform named actions. An occurrence of an object during the execution of a program is called an instance.



For example, a **Class employee** is defined and the object **myStaff** is instantiated in these programs using Pseudocode , VB and PYTHON are as follows:

```

CLASS Employee
  DECLARE name : STRING
  DECLARE StaffNo : INTEGER

  PUBLIC PROCEDURE New (ByValue n : STRING, ByValue s : INTEGER)
    Name ← n
    StaffNo ← s
  END PROCEDURE

  PUBLIC PROCEDURE ShowDetails ( )
    PRINT ("Employee name is " , Name )
    PRINT ("Employee Staff number is " , StaffNo )
  END PROCEDURE
END CLASS
BEGIN
DECLARE MyStaff : Employee
  CALL New( "Eric" , 721)
  CALL myStaff.ShowDetail ( )
END
  
```

VB

```

Module Module1
  Public Sub Main()
    Dim myStaff As New employee("Eric Jones", 72)
    myStaff.showDetails()
  End Sub

  class employee:
    Dim name As String
    Dim staffno As Integer
    Public Sub New (ByVal n As String, ByVal s As Integer)
      name = n
      staffno = s
    End Sub
    Public Sub showDetails()
      Console.WriteLine("Employee Name " & name)
      Console.WriteLine("Employee Number " & staffno)
      Console.ReadKey()
    End Sub
  End Class
End Module
  
```

PYTHON Code for Class:

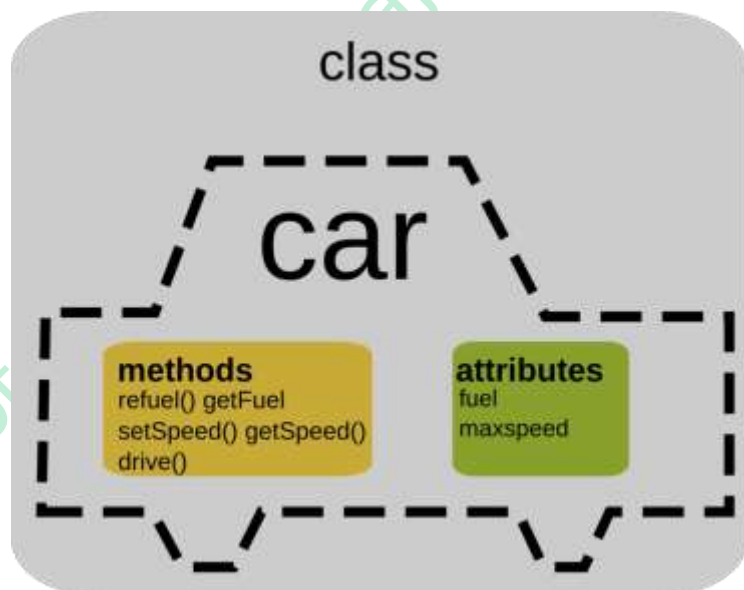
```

Python
class employee:
    def __init__ (self, name, staffno):
        self.name = name
        self.staffno = staffno
    def showDetails(self):
        print("Employee Name " + self.name)
        print("Employee Number " , self.staffno)
myStaff = employee("Eric Jones", 72)
myStaff.showDetails()
    
```

class definition

object

Another example of **Class Car** has **attributes** and **methods** given below:



Concept of OOP:

Previous chapters covered programming using the procedural aspect of our programming languages. Procedural programming groups related programming statements into subroutines. Related data items are grouped together into record data structures. To use a record variable, we first define a record type. Then we declare variables of that record type.

OOP goes one step further and groups together the record data structure and the subroutines that operate on the data items in this data structure. Such a group is called an 'object'.

Encapsulation: combining data and subroutines into a class

Class: a type that combines a record with the methods that operate on the properties in the record

REMEMER: Example of RECORD Data Type

A car manufacturer and seller wants to store details about cars. These details can be stored in a record structure

```
TYPE CarRecord
    DECLARE VehicleID      : STRING
    DECLARE Registration   : STRING
    DECLARE DateOfRegistration: DATE
    DECLARE EngineSize     : INTEGER
    DECLARE PurchasePrice  : CURRENCY
END TYPE
```

We can write program code to access and assign values to the fields of this **record**. For example:

```
PROCEDURE UpdateRegistration(BYREF ThisCar : CarRecord, BYVAL
NewRegistration)
    ThisCar.Registration ← NewRegistration
END PROCEDURE
```

We can call this procedure from anywhere in our program. This seems a well-regulated way of operating on the data record. However, we can also access the record fields directly from anywhere within the scope of **ThisCar**:

```
ThisCar.EngineSize ← 2500
```

Classes in OOP

The idea behind classes in OOP is that attributes can only be accessed through methods written as part of the class definition and validation can be part of these methods. The direct path to the data is unavailable. Attributes are referred to as 'private'. The methods to access the data are made available to programmers, so these are 'public'. Classes are templates for objects. When a class type has been defined it can be used to create one or more objects of this class type.

Attributes: the data items of a class
Methods: the subroutines of a class
Object: an instance of a class

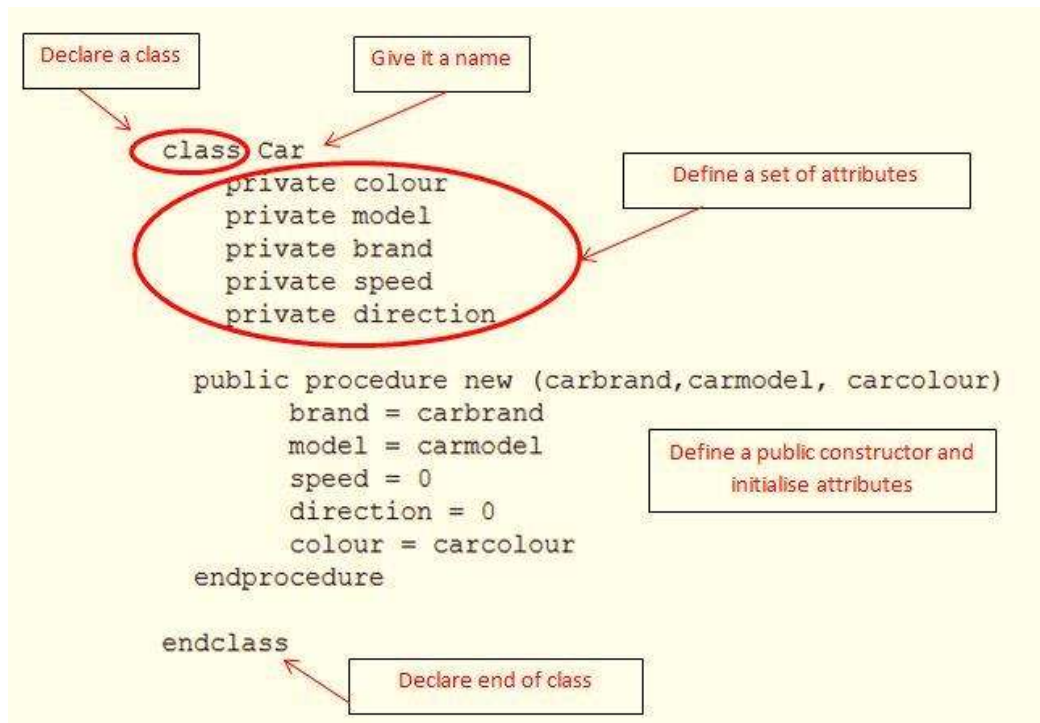


The first stage of writing an object-oriented program to solve a problem is to design the classes. This is part of object-oriented design. From this design, a program can be written using an object-oriented programming (OOP) language.

The programming languages the syllabus prescribes can be used for OOP: Python 3, VB.NET and Delphi/ObjectPascal.

Class parts

The parts that make up a class is shown below



Data Protection PRIVATE attributes PUBLIC Functions in Pseudocode:

```

CLASS Car          // Each attribute must be preceded by PRIVATE
  PRIVATE VehicleID :String
  PRIVATE Registration:String = " " 'String will be in " "
  PRIVATE DateOfRegistration:Date = #1/1/1900# 'Date between # #
  PRIVATE EngineSize :Integer
  PRIVATE PurchasePrice :Decimal = 0.0
  //Every public method header must start with Public
  //The constructor always has identifier New
  PUBLIC PROCEDURE New(ByVal n : String, ByVal e : String)
    VehicleID = n
    EngineSize = e
  END PROCEDURE

  PUBLIC PROCEDURE SetPurchasePrice(ByVal p: Decimal)
    PurchasePrice = p
  END PROCEDURE

  PUBLIC PROCEDURE SetRegistration(ByVal r: String)
    Registration = r
  END PROCEDURE

  PUBLIC PROCEDURE SetDateOfRegistration(ByVal d: Date)
    DateOfRegistration = d
  END PROCEDURE

  PUBLIC FUNCTION GetVehicleID() RETURNS String
    Return(VehicleID)
  END FUNCTION

  PUBLIC FUNCTION GetRegistration() RETURNS String
    Return (Registration)
  END FUNCTION

  PUBLIC FUNCTION GetDateOfRegistration()RETURNS Date
    Return (DateOfRegistration)
  END FUNCTION

  PUBLIC FUNCTION GetEngineSize() RETURNS Integer
    Return (EngineSize)
  END FUNCTION

  PUBLIC FUNCTION GetPurchasePrice() RETURNS Decimal
    Return (PurchasePrice)
  END FUNCTION
END CLASS

```



An example of VB and PYTHON CLASS code with its Procedures and Functions:

Visual Basic (Console mode)

```
Class student
    Dim name As String
    Dim dateOfBirth As Date
    Dim examMark As Integer

    Public Sub New(ByVal n As String, ByVal d As Date, ByVal e As Integer)
        name = n
        dateOfBirth = d
        examMark = e
    End Sub

    Public Sub displayExamMark()
        Console.WriteLine("Student Name " & name)
        Console.WriteLine("Exam Mark " & examMark)
        Console.ReadKey()
    End Sub
End Class

Sub Main()
    Dim myStudent As New student ("Mary Wu", #10/12/2012#, 67)
    myStudent.displayExamMark()
End Sub
```





PYTHON (Code)

```
class student:
    def __init__(self, name, dateOfBirth, examMark):
        self.name = name
        self.dateOfBirth = dateOfBirth
        self.examMark = examMark

    def displayExamMark(self):
        print("Student Name " + self.name)
        print("Exam Mark " , self.examMark)





myStudent = student("Mary Wu", 12/10/2012, 67)
myStudent.displayExamMark()
```

Advantages of OOP over procedural languages

-  The advantage of OOP is that it produces robust code.
-  The attributes can only be manipulated using methods provided by the class definition.
-  This means the attributes are protected from accidental changes.
-  Classes provided in module libraries are thoroughly tested. If you use tried and tested building blocks to construct your program, you are less likely to introduce bugs than when you write code from scratch.






Designing classes and objects

When designing a class:

-  We need to think about the **attributes** we want to store.
-  We also need to think about the **methods we need to access the data and assign values to the data of an object.**
-  A data type is a blueprint when declaring a variable of that data type.
-  A class definition is a blueprint when declaring an object of that class.



Attributes

The Car class has a number of attributes that can be altered.

-  In the above example, the attributes are "colour", "model", "brand", "speed" and "direction".
-  If the value of an attribute can be altered, then it is stored as a variable.
-  If you don't want to allow the value to be altered, then it is stored as a constant.
-  Attributes are normally 'private' which means only methods within the class can alter their value.
-  It is possible to have 'public' attributes i.e. variables that can be altered directly by external code, but that kind of loses the point of object orientated programming.

Instantiation

Creating a new object is known as '**instantiation**'.

-  Any data that is held about an object must be **accessible**, otherwise there is no point in storing it.
-  We therefore need methods to access each one of these attributes.

Constructor:

A constructor **instantiates** the object and **assigns initial values to the attributes.**

Constructor: a special type of method that is called to create a new object and initialise its attributes



A class can have many methods i.e. functions, that use the methods' attributes.



The most important of these methods is the **constructor**.



This is the method that creates an instance of the class i.e. it creates an object.



When an object is to be created, the constructor is called.

Getters:

These methods are usually referred to as **getters**.



They get an attribute of the object.



When we first set up an object of a particular class, we use a constructor.

Setters:

Any properties that might be updated after instantiation will need subroutines to update their values.



These are referred to as setters.



Some properties **get set** only at instantiation. These don't need setters.



This makes an object more robust, because you cannot change properties that were not designed to be changed.

WORKED EXAMPLE:

Consider the car data from above section

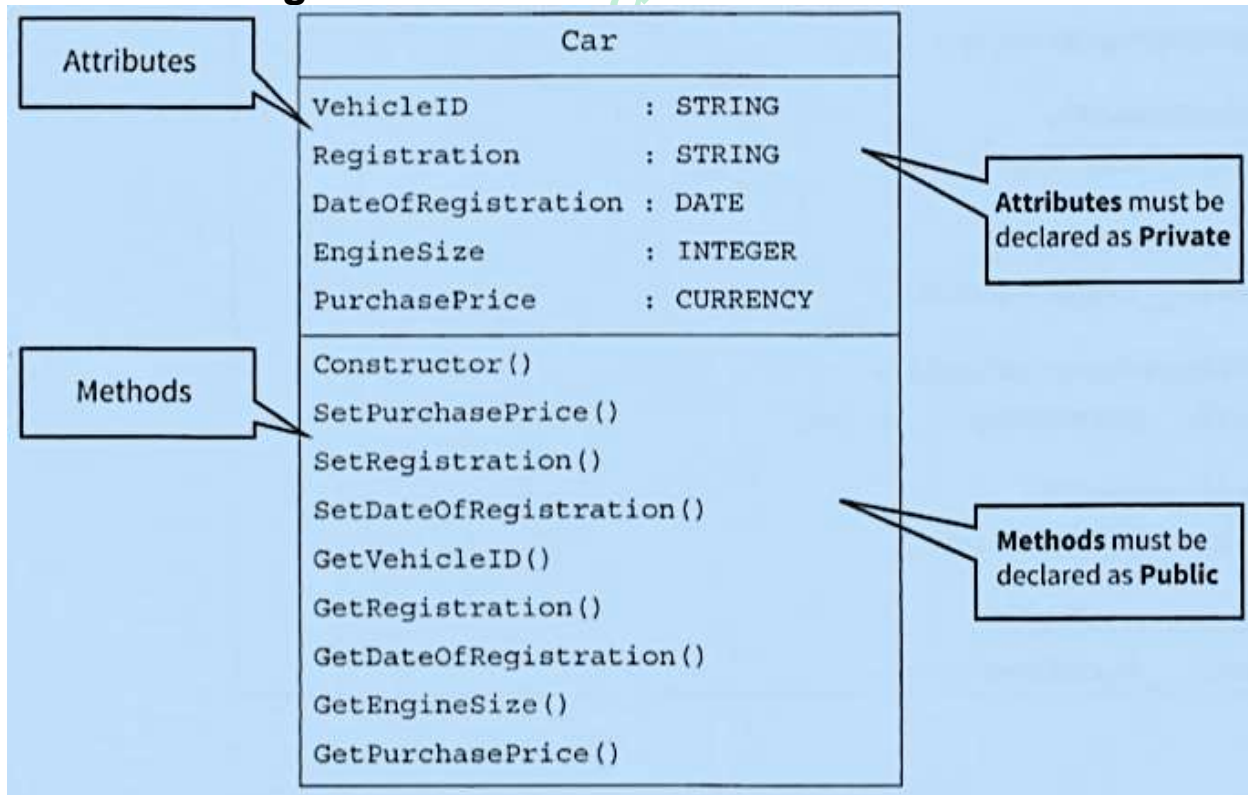
When a car is manufactured it is given a unique vehicle ID that will remain the same throughout the car's existence. The engine size of the car is fixed at the time of manufacture. The registration ID will be given to the car when the car is sold.

In our program, when a car is manufactured, we want to create a new car object. We need to instantiate it using the constructor. Any attributes that are already known at the time of instantiation can be set with the constructor. In our example, **vehicle ID** and **Engine size** can be set by the constructor. The other attributes are assigned values at the time of purchase and registration. So we need setters for them. The identifier table for the car class is shown in Table below:

Identifier	Description
Car	Class identifier
VehicleID	Unique ID assigned at time of manufacture
Registration	Unique ID assigned after time of purchase
DateOfRegistration	Date of registration
EngineSize	Engine size assigned at time of manufacture
PurchasePrice	Purchase price assigned at time of purchase
Constructor()	Method to create a Car object and set properties assigned at manufacture
SetPurchasePrice()	Method to assign purchase price at time of purchase
SetRegistration()	Method to assign registration ID
SetDateOfRegistration()	Method to assign date of registration
GetVehicleID()	Method to access vehicle ID
GetRegistration()	Method to access registration ID
GetDateOfRegistration()	Method to access date of registration
GetEngineSize()	Method to access engine size
GetPurchasePrice()	Method to access purchase price







We can represent this information as a class diagram in Figure

Car Class Diagram





Writing object-oriented code

Declaring a class in:

-  Attributes should always be declared as **'Private'**.
-  This means they can only be accessed through the class methods.
-  Methods can be called from the main program, so they have to be declared as **'Public'**.
-  There are other modifiers (such as **'Protected'**), but they are beyond the scope of this book.
-  The **syntax** for declaring classes is quite different for the different programming languages.
-  We will look at the three chosen languages. You are expected to write programs in one of these.

TASK27.01

-  Copy the car class definition into your program editor and write a simple program to test that each method works.
-  A business wants to store data about companies they supply. The data to be stored includes: company name, email address, date of last contact.
 - Design a class Company and draw a **class diagram**.
 - Write program code to declare the class. Company name and email address are to be set by the constructor and will never be changed.
 - Instantiate one object of this class and test your class code works.

Declaring a class in Visual Basic:

Module Module1

Class Car

```

' Each attribute must be preceded by PRIVATE
Private VehicleID As String
Private Registration As String = " " 'String will be in " "
Private DateOfRegistration As Date = #1/1/1900# 'Date between # #
Private EngineSize As Integer
Private PurchasePrice As Decimal = 0.0

'Every public method header must start with Public

'The constructor always has identifier New

Public Sub New(ByVal n As String, ByVal e As String)
    VehicleID = n
    EngineSize = e
End Sub

```



```
Public Sub SetPurchasePrice(ByVal p As Decimal)
    PurchasePrice = p
End Sub

Public Sub SetRegistration(ByVal r As String)
    Registration = r
End Sub

Public Sub SetDateOfRegistration(ByVal d As Date)
    DateOfRegistration = d
End Sub

Public Function GetVehicleID() As String
    Return(VehicleID)
End Function

Public Function GetRegistration() As String
    Return (Registration)
End Function

Public Function GetDateOfRegistration() As Date
    Return (DateOfRegistration)
End Function

Public Function GetEngineSize() As Integer
    Return (EngineSize)
End Function

Public Function GetPurchasePrice() As Decimal
    Return (PurchasePrice)
End Function
End Class

Sub Main()
    Dim ThisCar As New Car("ABC1234", 2500)
    ThisCar.SetPurchasePrice(12000)
    Console.WriteLine(ThisCar.GetVehicleID())
    Console.ReadLine()

    ThisCar = Nothing ' garbage collection
End Sub
End Module
```



Declaring a class in PYTHON:

Two underscore characters are required before and after `init` to define the constructor

`Self` is the first parameter in the parameter list for every method

```
class Car:
    def __init__(self, n, e):
        self.__VehicleID = n
        self.__Registration = ""
        self.__DateOfRegistration = None
        self.__EngineSize = e
        self.__PurchasePrice = 0.00

        # constructor
    def SetPurchasePrice(self, p):
        self.__PurchasePrice = p

    def SetRegistration(self, r):
        self.__Registration = r

    def SetDateOfRegistration(self, d):
        self.__DateOfRegistration = d

    def GetVehicleID ID(self) :
        return(self.__VehicleID)

    def GetRegistration(self):
        return(self.__Registration)

    def GetDateOfRegistration(self):
        return(self.__DateOfRegistration)

    def GetEngineSize(self):
        return(self.__EngineSize)

    def GetPurchasePrice(self):
        return(self.__PurchasePrice)
```

Two underscore characters before an attribute name signify it is private

Instantiating a class

To use an object of a class type in a program the object must first be instantiated. This means the memory space must be reserved to store the attributes.

The following code instantiates an object **ThisCar** of class **Car**.

Python

```
ThisCar = Car("ABC1234", 2500)
```

VB

```
Dim ThisCar As New Car("ABC1234", 2500)
```

Instantiating a class:

To use an object of a class type in a program the object must first be **instantiated**. This means the memory space must be reserved to store the attributes.

The following code instantiates an object **Thiscar** of **class car**.

VB.NET

```
Dim ThisCar As New Car("ABC1234", 2500)
```

Using a method

To call a method in program code, the object identifier is followed by the method identifier and the parameter list.

The following code sets the purchase price for an object **ThisCar** of class **Car**.

VB.NET

```
ThisCar.SetPurchasePrice(12000)
```

The following code gets and prints the vehicle ID for an object **ThisCar** of class **Car**.

VB.NET

```
Console.WriteLine(ThisCar.GetVehicleID())
```

The following code gets and prints the vehicle ID for an object **ThisCar** of class **Car** in Python.

```
print(ThisCar.GetVehicleID())
```

```
print(ThisCar.VehicleID) # using properties
```

TASK27.01

A business wants to store data about companies they supply. The data to be stored includes: company name, email address, date of last contact.

- Design a class Company and draw a **class diagram**.
- Write program code to declare the class. Company name and email address are to be set by the constructor and will never be changed.
- Instantiate one object of this class and test your class code works.

Module Module1

Class Company

```
Private CompanyName As String
Private EmailAddress As String
Private DateOfLastContact As Date

Public Sub New(ByVal n, ByVal e) 'constructor
    CompanyName = n
    EmailAddress = e
    DateOfLastContact = #1/1/1900#
End Sub
```

```
Public Sub SetDateOfLastContact(ByVal d)
    DateOfLastContact = d
End Sub
```

```
Public Function GetCompanyName()
    Return (CompanyName)
End Function
```

```
Public Function GetEmailAddress()
    Return (EmailAddress)
End Function
```

```
Public Function GetDateOfLastContact()
    Return (DateOfLastContact)
End Function
```

End Class

Sub Main()

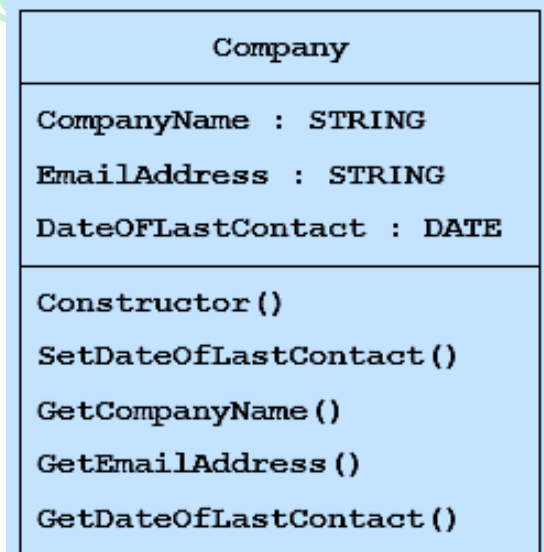
```
Dim ThisCompany As New Company("SLimited", "abc@slimited.cie")
ThisCompany.SetDateOfLastContact(#1/2/2016#)
Console.WriteLine(ThisCompany.GetDateOfLastContact())
```

```
Console.ReadLine()
```

End Sub

End Module

Class diagram



Class Company in Python

```
class Company:
    def __init__(self, n, e): # constructor
        self.__CompanyName = n
        self.__EmailAddress = e
        self.__DateOfLastContact = None

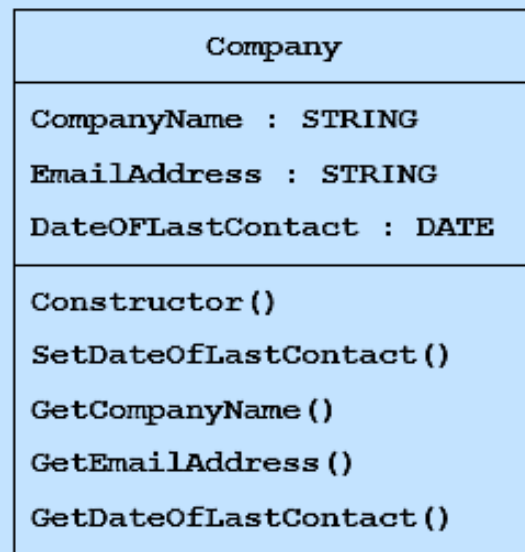
    def SetDateOfLastContact(self, d):
        self.__DateOfLastContact = d

    def GetCompanyName(self):
        return(self.__CompanyName)

    def GetEmailAddress(self):
        return(self.__EmailAddress)

    def GetDateOfLastContact(self):
        return(self.__DateOfLastContact)
```

Class diagram



```
ThisCompany = Company("SLimited", "abc@slimited.cie")
ThisCompany.SetDateOfLastContact(date(2016,2,1))
print(ThisCompany.GetDateOfLastContact())
```

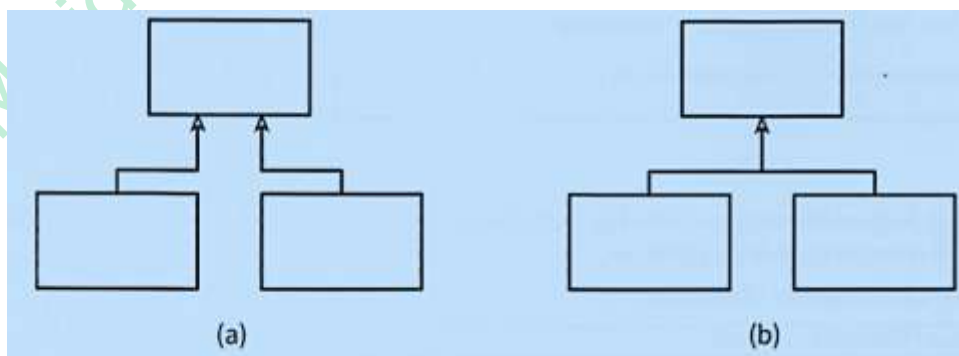
Inheritance:

The advantage of OOP is that we can design a class (a **base class** or a **superclass**) and then derive further classes (**subclasses**) from this **base class**.

Inheritance is the process by which the methods and data from one class, a **superclass** or **base class**, are copied to another class, a **derived class**.

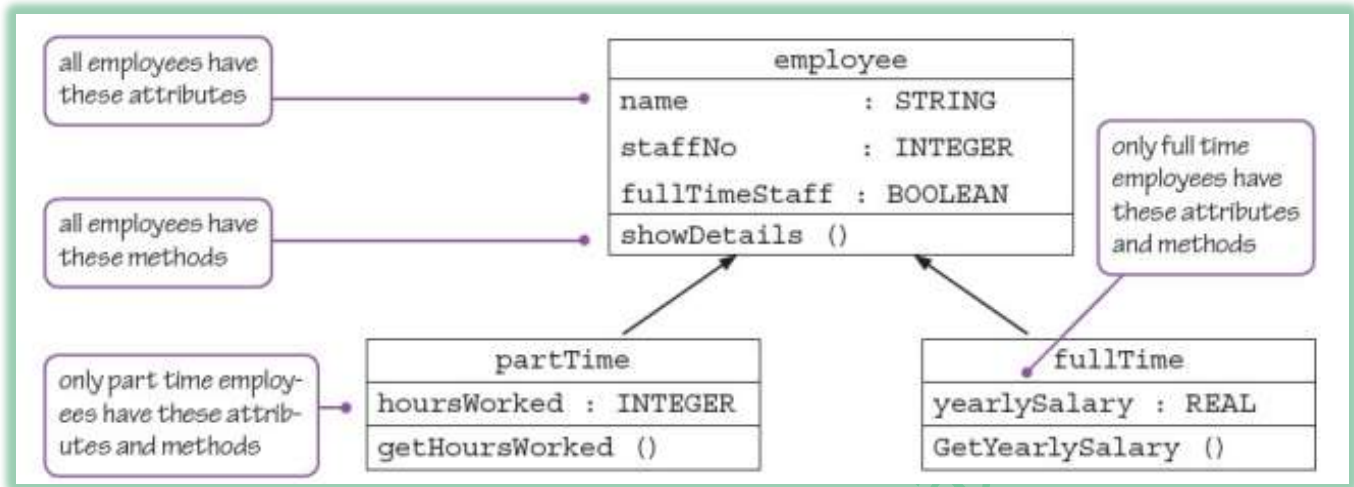
This means that we write the code for the base class only once and the subclasses make use of the attributes and methods of the base class, as well as having their own attributes and methods.

This is known as inheritance and can be represented by an inheritance diagram



Inheritance: all attributes and methods of the base class are copied to the subclass

Inheritance diagram for **parTime**, **fullTime** and **employee** class



WORKED EXAMPLE

Implementing a library system

Consider the following problem:





-  A college library has items for loan.
-  The items are currently books and CDs.
-  Items can be borrowed for three weeks.
-  If a book is on loan, it can be requested by another borrower.

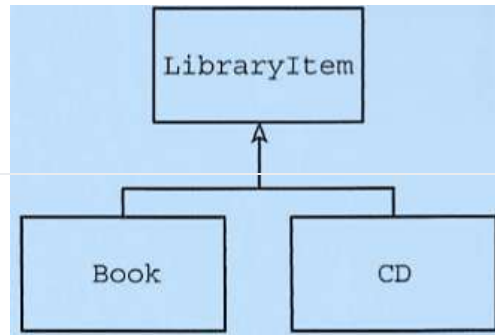
Table below shows the information to be stored.

Library item	
Book	CD
Title of book*	Title of CD*
Author of book*	Artist of CD*
Unique library reference number*	Unique library reference number*
Whether it is on loan*	Whether it is on loan*
The date the book is due for return*	The date the CD is due for return*
Whether the book is requested by another borrower	The type of music on the CD (genre)

The information to be stored about books and CDs needs further analysis. Note that we could have a variable Title, which stores the book title or the CD title, depending on which type of library item we are working with. There are further similarities (shown in above table).

There are some items of data that are different for books and CDs. Books can be requested by a borrower. For CDs, the genre is to be stored.

We can define a class **Libraryitem** and derive a **Book** class and a **CD** class from it. We can draw the inheritance diagrams for the **Libraryitem**, **Book** and **CD** classes as in Figure below



Inheritance diagram for Library Item, Book and CD classes

Analysing the attributes and methods required for all library items and those only required for books and only for CDs, we arrive at the class diagram in Figure below

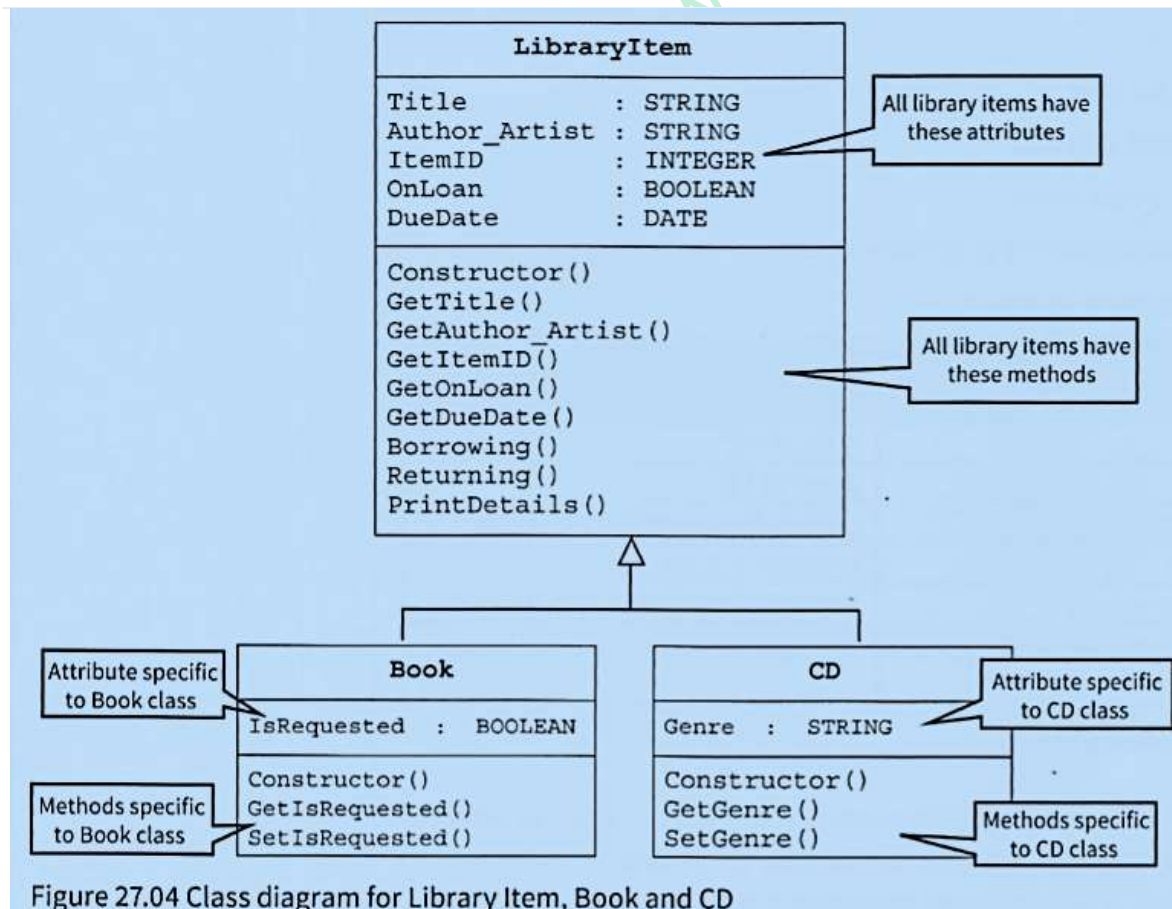


Figure 27.04 Class diagram for Library Item, Book and CD

A base class that is never used to create objects directly is known as an abstract class. `LibraryItem` is an abstract class.

Abstract class: a base class that is never used to create objects directly

Inheritance of CLASS in Pseudocodes:

Declaring a base class and derived classes (subclasses)

The code below shows how a base class and its subclasses are declared.

```

CLASS LibraryItem
    DECLARE PRIVATE Title : STRING
    DECLARE PRIVATE Author_Artist : STRING
    DECLARE PRIVATE ItemID : INTEGER
    DECLARE PRIVATE OnLoan : BOOLEAN = False
    DECLARE PRIVATE DueDate : DATE
    PUBLIC PROCEDURE Create(ByVal t : String, ByVal a :String, ByVal i : Integer)
        Title = t
        Author_Artist = a
        ItemID = i
    END PROCEDURE
    PUBLIC FUNCTION
        GetTitle() As String.Return (Title)
    END FUNCTION ' other Get methods go here
    PUBLIC PROCEDURE Borrowing()
        OnLoan = True
        DueDate = DateAdd(DateInterval.Day, 21, Today()) '3 wee ks from today
    END PROCEDURE
    PUBLIC PROCEDURE Returning()
        OnLoan = False
    END PROCEDURE
    PUBLIC PROCEDURE PrintDetails()
        PRINT(Title & """, """, ItemID & """, """, OnLoan & """, """, DueDate)
    END PROCEDURE
END CLASS
CLASS Book 'A subclass definition
INHERITS LibraryItem 'The Inherits statement is first statementof subClass definition
PRIVATE Isrequested : Boolean = False
    PUBLIC FUNCTION GetisRequested() RETURNS BOOLEAN
        RETURN (Isrequested)
    ENDFUNCTION
    PUBLIC PROCEDURE SetisRequested()
        Isrequested = True
    END PROCEDURE
END CLASS
CLASS CD
INHERITS LibraryItem 'The Inherits statement is first statementof subClass definition
    DECLARE PRIVATE Genre : String
    PUBLIC FUNCTION GetGenre() RETURNS STRING
        Return (Genre)
    END FUNCTION
    PUBLIC PROCEDURE SetGenre(ByVal g : String)
        Genre = g
    END PROCEDURE
END CLASS

```



Inheritance in VB:**Declaring a base class and derived classes (subclasses) in VB.NET**

The code below shows how a base class and its subclasses are declared in VB.NET.

```

Class LibraryItem
    Private Title As String
    Private Author_Artist As String
    Private ItemID As Integer
    Private OnLoan As Boolean = False
    Private DueDate As Date = Today
    Sub Create(ByVal t As String, ByVal a As String, ByVal i As Integer)
        Title = t
        Author_Artist = a
        ItemID = i
    End Sub
    Public Function
        GetTitle() As String.Return (Title)
    End Function ' other Get methods go here
    Public Sub Borrowing()
        OnLoan = True
        DueDate = DateAdd(DateInterval.Day, 21, Today()) '3 weeks from today
    End Sub
    Public Sub Returning()
        OnLoan = False
    End Sub
    Sub PrintDetails()
        Console.WriteLine(Title & " ", " " & ItemID & " ", " " & OnLoan & " ", " " & DueDate)
    End Sub
End Class
Class Book 'A subclass definition
Inherits LibraryItem 'The Inherits statement is first statement of subClass definition
Private IsRequested As Boolean = False
    Public Function GetIsRequested() As Boolean
        Return (IsRequested)
    End Function
    Public Sub SetIsRequested()
        IsRequested = True
    End Sub
End Class
Class CD
Inherits LibraryItem 'The Inherits statement is first statement of subClass definition
Private Genre As String
    Public Function GetGenre() As String
        Return (Genre)
    End Function
    Public Sub SetGenre(ByVal g As String)
        Genre = g
    End Sub
End Class

```

Instantiating a subclass

Creating an object of a subclass is done in the same way as with any class (See Section 27.03).

VB.NET	<pre>Dim ThisBook As New Book() Dim ThisCD As New CD() ThisBook.Create(Title, Author, ItemID) ThisCD.Create(Title, Artist, ItemID)</pre>
---------------	---

Inheritance in Python:

```
import datetime
class LibraryItem:
    def __init__(self, t, a, i): # initialiser method
        self.__Title = t
        self.__Author_Artist = a
        self.__ItemID = i
        self.__OnLoan = False
        self.__DueDate = datetime.date.today()
    def GetTitle(self):
        return(self.__Title)
    def GetAuthor_Artist(self):
        return(self.__Author_Artist)
    def GetItemID(self):
        return(self.__ItemID)
    def GetOnLoan(self):
        return(self.__OnLoan)
    def GetDueDate(self):
        return(self.__DueDate)
    def Borrowing(self):
        self.__OnLoan = True
        self.__DueDate = self.__DueDate +datetime.timedelta(weeks=3)
        def Returning(self):
            self.__OnLoan = False
    def PrintDetails(self):
        print(self.__Title, ' ; ', self.__Author_Artist, end='')
        print(' ; ', self.__ItemID, ' ; ', self.__OnLoan, end='')
        print(' ; ', self.__DueDate)

class Book(LibraryItem):
    def __init__(self, t, a, i): # initialiser method
        LibraryItem.__init__(self, t, a, i)
        self.__IsRequested = False
```

```

def GetIsRequested(self):
    return(self.__IsRequested)

def SetIsRequested(self):
    self.__IsRequested = True

class CD(LibraryItem):
    def __init__(self, t, a, i): # initialiser method
        LibraryItem.__init__(self, t, a, i)
        self.__Genre = ""

    def GetGenre(self):
        return(self.__Genre)
    def SetGenre(self, g):
        self.__Genre = g

def main():
    ThisCD = CD("Let it be", "Beatles", 2345)
    ThisBook.PrintDetails()
    ThisCD.PrintDetails()

main()

```

Instantiating a Class:

```

ThisBook = Book(Title, Author, ItemID)
ThisCD = CD(Title, Artist, ItemID)

```

TASK 27.02

Copy the class definitions for Libraryitem, Book and CD into your program editor. Write the additional get methods. Write a simple program to test that each method work

```

Module Module1
    Class LibraryItem
        Private Title As String
        Private Author_Artist As String
        Private ItemID As Integer
        Private OnLoan As Boolean = False
        Private DueDate As Date = Today

        Sub Create(ByVal t As String, ByVal a As String, ByVal i As Integer)
            Title = t
            Author_Artist = a
            ItemID = i
        End Sub
        Public Function GetTitle() As String
            Return (Title)
        End Function
        Public Function GetAuthor_Artist() As String
            Return (Author_Artist)
        End Function
        Public Function GetItemID() As Integer

```

```
        Return (ItemID)
    End Function
    Public Function GetOnLoan() As Boolean
        Return (OnLoan)
    End Function
    Public Function GetDueDate() As Date
        Return (DueDate)
    End Function
    Public Sub Borrowing()
        OnLoan = True
        DueDate = DateAdd(DateInterval.Day, 21, Today()) '3 weeks from today
    End Sub

    Public Sub Returning()
        OnLoan = False
    End Sub

    Public Sub PrintDetails()
        Console.Write(Title & "; " & ItemID & "; " & OnLoan & "; ")
        Console.WriteLine(DueDate)
    End Sub
End Class

Class Book
    Inherits LibraryItem
    Private IsRequested As Boolean = False
    Public Function GetIsRequested() As Boolean
        Return (IsRequested)
    End Function
    Public Sub SetIsRequested()
        IsRequested = True
    End Sub
End Class

Class CD
    Inherits LibraryItem
    Private Genre As String = ""
    Public Function GetGenre() As String
        Return (Genre)
    End Function
    Public Sub SetGenre(ByVal g As String)
        Genre = g
    End Sub
End Class

Sub Main()
    Dim ThisBook As New Book()
    Dim ThisCD As New CD()
    ThisBook.Create("Computing", "Sylvia", 1234)
    ThisCD.Create("Let it be", "Beatles", 2345)
    ThisBook.PrintDetails()
    ThisCD.PrintDetails()
    Console.ReadLine()
End Sub
End Module
```



TASK27.03

Write code to define a Borrower class as shown in the class diagram in Figure 27.05

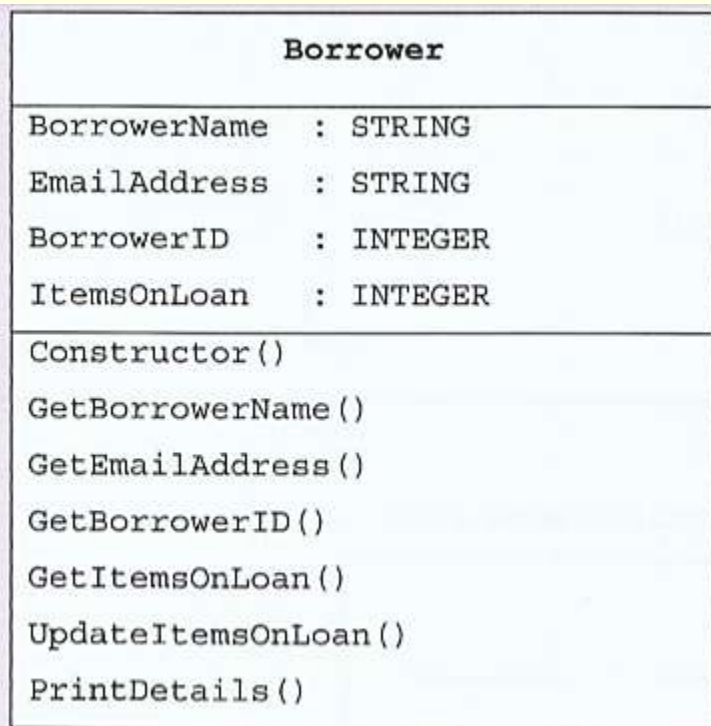


Figure 27.05 Borrower class diagram

The constructor should initialise ItemsOnLoan too. UpdateitemsOnLoa-no should increment ItemsOnLoan by an integer passed as parameter.

Write a simple program to test the methods

Module Module1

Class Borrower

```

Private BorrowerName As String
Private EmailAddress As String
Private BorrowerID As Integer
Private ItemsOnLoan As Integer
Public Sub Create(ByVal n As String, ByVal e As String, ByVal b As Integer)
    BorrowerName = n
    EmailAddress = e
    BorrowerID = b
    ItemsOnLoan = 0
End Sub

Public Function GetBorrowerName() As String
    GetBorrowerName = BorrowerName
End Function

Public Function GetEmailAddress() As String
    GetEmailAddress = EmailAddress
End Function

```









```

Public Function GetBorrowerID() As Integer
    GetBorrowerID = BorrowerID
End Function
Public Function GetItemsOnLoan() As Integer
    GetItemsOnLoan = ItemsOnLoan
End Function
Public Sub UpdateItemsOnLoan(ByVal n As Integer)
    ItemsOnLoan += n
End Sub
Public Sub PrintDetails()
    Console.WriteLine("Borrower      : " & BorrowerName)
    Console.WriteLine("ID          : " & BorrowerID)
    Console.WriteLine("email       : " & EmailAddress)
    Console.WriteLine("Items on loan: " & ItemsOnLoan)
End Sub
End Class

Sub Main()
    Dim NewBorrower As New Borrower()
    NewBorrower.Create("Sylvia", "adc@cie", 123)
    NewBorrower.UpdateItemsOnLoan(3)
    NewBorrower.PrintDetails()
    NewBorrower.UpdateItemsOnLoan(-1)
    NewBorrower.PrintDetails()
    Console.ReadLine()
End Sub
End Module

```

Polymorphism:

-  The constructor method of the base class is redefined in the subclasses.
-  The constructor for the Book class calls the constructor of the Libraryitem class and also initialises the IsRequested attribute.
-  The constructor for the CD class calls the constructor of the Libraryitem class and also initialises the Genre attribute.
-  The PrintDetails method is currently only defined in the base class.
-  This means we can only get information on the attributes that are part of the base class.
-  To include the additional attributes from the subclass, we need to declare the method again.
-  Although the method in the subclass will have the same identifier as in the base class, the method will actually behave differently.
-  This is known as polymorphism.

Polymorphism is when methods are redefined for derived classes. Overloading is when a method is defined more than once in a class so it can be used in different situations.

Example of polymorphism: A base class shape is defined, and the derived classes rectangle and circle are defined. The method area is redefined for both the rectangle class and the circle class. The objects myRectangle and myCircle are instanced in these programs.

```
Module Program
  Class shape
    Protected areaValue As Decimal
    Protected perimeterValue As Decimal
    Overridable Sub area()
      Console.WriteLine("Area is: " & areaValue)
    End Sub

    Overridable Sub perimeter()
      Console.WriteLine("Perimeter is : " & perimeterValue)
    End Sub
  End Class
  Class rectangle : Inherits shape
    Private length As Decimal
    Private breadth As Decimal

    Public Sub New(ByVal l As Decimal, ByVal b As Decimal)
      length = l
      breadth = b
    End Sub
    Overrides Sub Area()
      areaValue = length * breadth
      Console.WriteLine("Area = " & areaValue)
    End Sub
  End Class
  Class circle : Inherits shape
    Private radius As Decimal
    Public Sub New(ByVal r As Decimal)
      radius = r
    End Sub
    Overrides Sub Area()
      areaValue = radius * radius * 3.142
      Console.WriteLine("Area = " & areaValue)
    End Sub
  End Class
  Sub main()
    Dim myCircle As New circle(20)
    myCircle.Area()
    Dim myRectangle As New rectangle(10, 17)
    myRectangle.Area()
    Console.ReadKey()
  End Sub
End Module
```



Polymorphism in python:

```
class shape:
    def __init__(self):
        self.__areaValue = 0
        self.__perimeterValue = 0
    def area(self):
        print("Area ", self.__areaValue)
    def perimeter(self):
        print("Perimeter ", self.__areaValue)
class rectangle(shape):
    def __init__(self, length, breadth):
        shape.__init__(self)
        self.__length = length
        self.__breadth = breadth
    def area (self):
        self.__areaValue = self.__length * self.__breadth
        print("Area ", self.__areaValue)
class circle(shape):
    def __init__(self, radius):
        shape.__init__(self)
        self.__radius = radius
    def area (self):
        self.__areaValue = self.__radius * self.__radius * 3.142
        print("Area ", self.__areaValue)
myCircle = circle(20)
myCircle.area()
myRectangle = rectangle (10,17)
myRectangle.area()
```

original method in shape class

redefined method in rectangle class

redefined method in circle class

Polymorphism in python:




```
class shape:
    def __init__(self):
        self.__areaValue = 0
        self.__perimeterValue = 0

    def area(self):
        print("Area ", self.__areaValue)

class square(shape):
    def __init__(self, side):
        shape.__init__(self)
        self.__side = side

    def area (self):
        self.__areaValue = self.__side * self.__side
        print("Area ", self.__areaValue)

class rectangle(shape):
    def __init__(self, length, breadth):
        shape.__init__(self)
        self.__length = length
        self.__breadth = breadth

    def area (self):
        self.__areaValue = self.__length * self.__breadth
        print("Area ", self.__areaValue)

class circle(shape):
    def __init__(self, radius):
        shape.__init__(self)
        self.__radius = radius

    def area (self):
        self.__areaValue = self.__radius * self.__radius * 3.142
        print("Area ", self.__areaValue)

mySquare = square(10)
mySquare.area()
myCircle = circle(20)
myCircle.area()

myRectangle = rectangle (10,17)
myRectangle.area()
```

Garbage collection:



When objects are created they occupy memory. When they are no longer needed, they should be made to release that memory, so it can be re-used. If objects do not let go of memory, we eventually end up with no free memory when we try and run a program.

This is known as 'memory leakage'.

How do our programming languages handle this?

VB.NET	A garbage collector automatically reclaims memory from objects that are no longer referred to by the running program.
---------------	---

In VB.NET we used Class Car earlier

When we want to reclaim memory we use following code for Garbage collection:

```
ThisCar = Nothing ' garbage collection
```

Python INTERPRETER automatically does Garbage Collection.

Exception handling

Run-time errors can occur for many reasons.



Some examples are division by zero, invalid array index or trying to open a non-existent file.



Run-time errors are called '**exceptions**'.



They can be handled (resolved) with an error subroutine (known as an '**exception handler**'), rather than let the program crash.

Using pseudocode, the error-handling structure is:

```
TRY  
    <statementsA>  
EXCEPT  
    <statementsB>  
END TRY
```

Any run-time error that occurs during the execution of **<statementsA>** is caught and handled by executing **<statementsB>**. There can be more than one EXCEPT block, each handling a different type of exception. Sometimes a FINALLY block follows the exception handlers. The statements in this block will be executed regardless of whether there was an exception or not.

VB.NET is designed to treat exceptions as abnormal and unpredictable erroneous situations. You may find you need to include exception handling in the code for Worked Example 26.02. Otherwise the end of file is encountered and the program crashes.

VB.NET	<pre> NumberString = Console.ReadLine() Try n = Int(NumberString) Console.WriteLine(n) Catch Console.WriteLine("this was not an integer") End Try </pre>
---------------	--

TASK 26.03

Add exception-handling code to your programs for Task 26.01 or Task 26.02. Test your code handles exceptions without the program crashing

Solution code is written below:

```

Module Module1
    Class Borrower
        Private BorrowerName As String
        Private EmailAddress As String
        Private BorrowerID As Integer
        Private ItemsOnLoan As Integer
        Public Sub Create(ByVal n As String, ByVal e As String, ByVal b As Integer)
            BorrowerName = n
            EmailAddress = e
            BorrowerID = b
            ItemsOnLoan = 0
        End Sub
        Public Function GetBorrowerName() As String
            GetBorrowerName = BorrowerName
        End Function
        Public Function GetEmailAddress() As String
            GetEmailAddress = EmailAddress
        End Function
        Public Function GetBorrowerID() As Integer
            GetBorrowerID = BorrowerID
        End Function
        Public Function GetItemsOnLoan() As Integer
            GetItemsOnLoan = ItemsOnLoan
        End Function
        Public Sub UpdateItemsOnLoan(ByVal n As Integer)
            ItemsOnLoan += n
        End Sub
        Public Sub PrintDetails()
            Console.WriteLine("Borrower      : " & BorrowerName)
            Console.WriteLine("ID          : " & BorrowerID)
            Console.WriteLine("email       : " & EmailAddress)
            Console.WriteLine("Items on loan: " & ItemsOnLoan)
        End Sub
    End Class

    Sub Main()
        Dim NewBorrower As New Borrower()
        NewBorrower.Create("Sylvia", "adc@cie", 123)
    End Sub
End Module
    
```

```
NewBorrower.UpdateItemsOnLoan(3)
NewBorrower.PrintDetails()
NewBorrower.UpdateItemsOnLoan(-1)
NewBorrower.PrintDetails()
Console.ReadLine()
End Sub
End Module
```

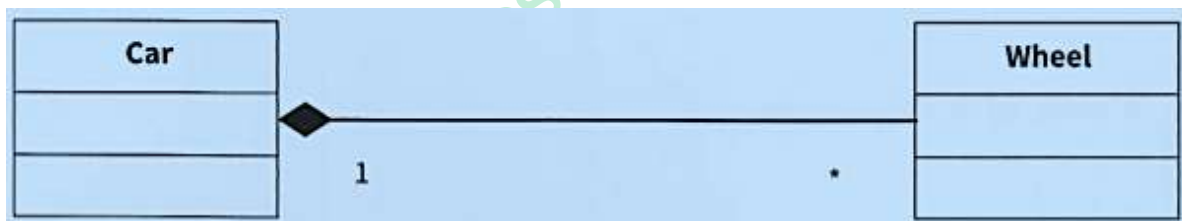
Containment, or aggregation:

Containment is the process by which one class can contain other classes. This can be presented in a class diagram. When the class 'aeroplane' is defined, and the definition contains references to the classes – seat, fuselage, wing, cockpit – this is an example of containment.

Containment {aggregation}

- Containment means that one class contains other classes.
- For example, a car is made up of different parts and each part will be an object based on a class.
- The wheels are objects of a different class to the engine object.
- The engine is also made up of different parts.
- Together, all these parts make up one big object.

Containment Diagram

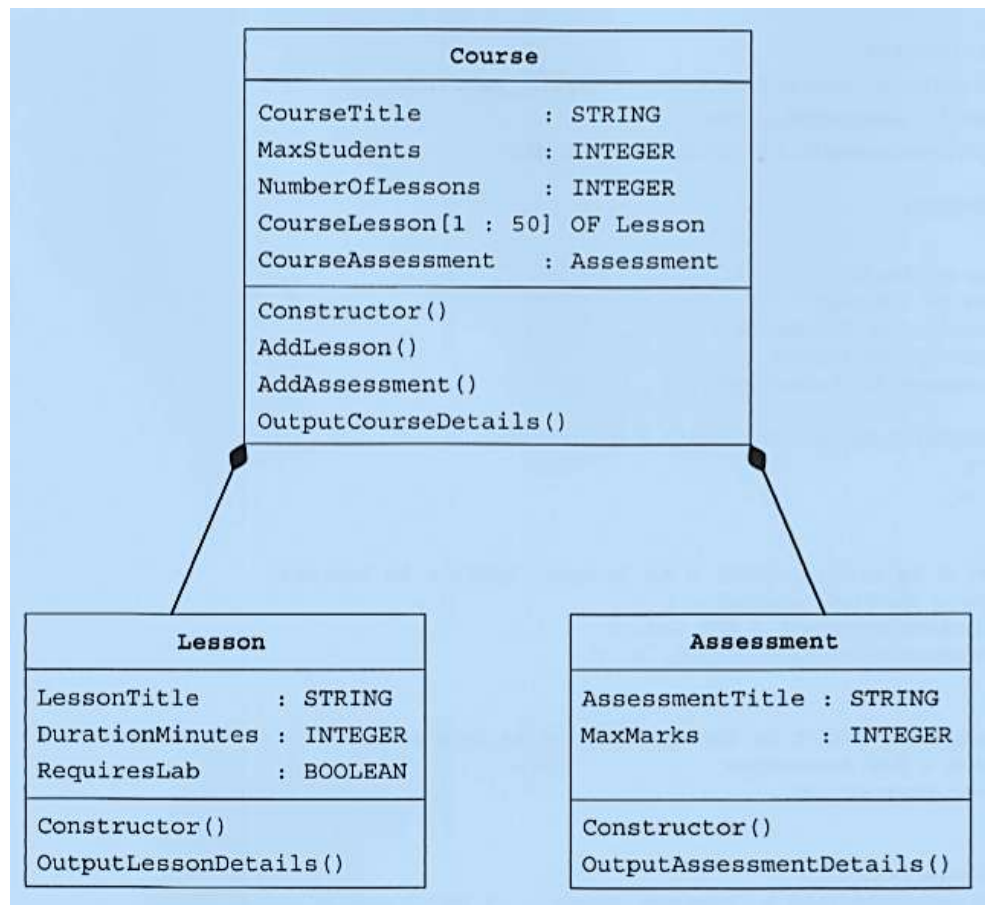


Containment: a relationship in which one class has a component that is of another class type.

WORKED EXAMPLE:

Using containment:

A college runs courses of up to 50 lessons. A course may end with an assessment. Object-oriented programming is to be used to set up courses. The classes required are shown in figure:



CONTAINMENT PSEUDOCODE:

CLASS Assessment

DECLAE PRIVATE AssessmentTitle : String
DECLAE PRIVATE MaxMarks As Integer

PUBLIC PROCEDURE Create(ByVal t : String, ByVal m : Integer)
 AssessmentTitle = t
 MaxMarks = m
END PROCEDURE

PUBLIC PROCEDURE OutputAssessmentDetails()
 PRINT(AssessmentTitle , "Marks: " , MaxMarks)
END PROCEDURE

END CLASS

CLASS Lesson

DECLAE PRIVATE LessonTitle : String
DECLAE PRIVATE DurationMinutes : Integer
DECLAE PRIVATE RequiresLab : Boolean

PUBLIC PROCEDURE Create(ByVal t: String, ByVal d: Integer, ByVal r: Boolean)
 LessonTitle = t
 DurationMinutes = d
 RequiresLab = r
END PROCEDURE

```
PUBLIC PROCEDURE OutputLessonDetails()  
    Console.WriteLine(LessonTitle & " " & DurationMinutes)  
END PROCEDURE  
END CLASS  
  
CLASS Course  
    DECLAE PRIVATE CourseTitle : String  
    DECLAE PRIVATE MaxStudents : Integer  
    DECLAE PRIVATE NumberOfLessons : Integer = 0  
    DECLAE PRIVATE CourseLesson(50) : Lesson  
    DECLAE PRIVATE CourseAssessment : Assessment  
  
    PUBLIC PROCEDURE Create(ByVal t As String, ByVal m As Integer)  
        CourseTitle = t  
        MaxStudents = m  
    END PROCEDURE  
  
    PUBLIC PROCEDURE AddLesson(ByVal t : String, ByVal d: Integer, ByVal r: Boolean)  
        NumberOfLessons = NumberOfLessons + 1  
        CourseLesson(NumberOfLessons) = New Lesson  
        CourseLesson(NumberOfLessons).Create(t, d, r)  
    END PROCEDURE  
  
    PUBLIC PROCEDURE AddAssessment(ByVal t As String, ByVal m As Integer)  
        CourseAssessment = New Assessment  
        CourseAssessment.Create(t, m)  
    END PROCEDURE  
  
    PUBLIC PROCEDURE OutputCourseDetails()  
        PRINT (CourseTitle)  
        PRINT ("Maximum number of students: " & MaxStudents)  
        FOR i = 1 To NumberOfLessons  
            CourseLesson(i).OutputLessonDetails()  
        NEXT i  
    END PROCEDURE  
End Class  
  
BEGIN  
    DECLARE MyCourse As New Course  
        MyCourse.Create("Computing", 10) //sets up a new course  
        MyCourse.AddAssessment("Programming", 100) //adds an assessment  
        //add 3 lessons  
        MyCourse.AddLesson("Problem Solving", 60, False)  
        MyCourse.AddLesson("Programming", 120, True)  
        MyCourse.AddLesson("Theory", 60, False)  
        MyCourse.OutputCourseDetails() //check it all works  
END
```

WORKED EXAMPLE solution VB.NET :

```
Module Module1 'CONTAINMENT
    Class Assessment
        Private AssessmentTitle As String
        Private MaxMarks As Integer

        Public Sub Create(ByVal t As String, ByVal m As Integer)
            AssessmentTitle = t
            MaxMarks = m
        End Sub

        Public Sub OutputAssessmentDetails()
            Console.WriteLine(AssessmentTitle & "Marks: " & MaxMarks)
        End Sub
    End Class

    Class Lesson
        Private LessonTitle As String
        Private DurationMinutes As Integer
        Private RequiresLab As Boolean

        Public Sub Create(ByVal t As String, ByVal d As Integer, ByVal r As Boolean)
            LessonTitle = t
            DurationMinutes = d
            RequiresLab = r
        End Sub

        Public Sub OutputLessonDetails()
            Console.WriteLine(LessonTitle & " " & DurationMinutes)
        End Sub
    End Class

    Class Course
        Private CourseTitle As String
        Private MaxStudents As Integer
        Private NumberOfLessons As Integer = 0
        Private CourseLesson(50) As Lesson
        Private CourseAssessment As Assessment

        Public Sub Create(ByVal t As String, ByVal m As Integer)
            CourseTitle = t
            MaxStudents = m
        End Sub

        Sub AddLesson(ByVal t As String, ByVal d As Integer, ByVal r As Boolean)
            NumberOfLessons = NumberOfLessons + 1
            CourseLesson(NumberOfLessons) = New Lesson
            CourseLesson(NumberOfLessons).Create(t, d, r)
        End Sub

        Public Sub AddAssessment(ByVal t As String, ByVal m As Integer)
            CourseAssessment = New Assessment
            CourseAssessment.Create(t, m)
        End Sub
    End Class
End Module
```

```

Public Sub OutputCourseDetails()
    Console.Write(CourseTitle)
    Console.WriteLine("Maximum number of students: " & MaxStudents)
    For i = 1 To NumberOfLessons
        CourseLesson(i).OutputLessonDetails()
    Next
End Sub
End Class

Sub Main()
    Dim MyCourse As New Course
    MyCourse.Create("Computing", 10) ' sets up a new course
    MyCourse.AddAssessment("Programming", 100) ' adds an assessment
    ' add 3 lessons
    MyCourse.AddLesson("Problem Solving", 60, False)
    MyCourse.AddLesson("Programming", 120, True)
    MyCourse.AddLesson("Theory", 60, False)
    MyCourse.OutputCourseDetails()'check it all works
    Console.ReadLine()
End Sub
End Module

```

WORKED EXAMPLE solution Python :

```

class Assessment:
    def __init__(self, t, m):
        self.__AssessmentTitle = t
        self.__MaxMarks = m

    def OutputAssessmentDetails(self):
        print(self.__AssessmentTitle, " Marks: ", self.__MaxMarks)

class Course:
    def __init__(self, t, m): # sets up a new course
        self.__CourseTitle = t
        self.__MaxStudents = m
        self.__NumberOfLessons = 0
        self.__CourseLesson = []
        self.__CourseAssessment = Assessment

    def AddLesson(self, t, d, r):
        self.__NumberOfLessons = self.__NumberOfLessons + 1
        self.__CourseLesson.append(Lesson(t, d, r))

    def AddAssessment(self, t, m):
        CourseAssessment = Assessment(t, m)

    def OutputCourseDetails(self):
        print(self.__CourseTitle, end=' ')
        print("Maximum number of students: ", self.__MaxStudents)
        for i in range(self.__NumberOfLessons):
            print(self.__CourseLesson[i].OutputLessonDetails())

```




```
class Lesson:
    def __init__(self, t, d, r):
        self.__LessonTitle = t
        self.__DurationMinutes = d
        self.__requiresLab = r
    def OutputLessonDetails(self):
        print(self.__LessonTitle, self.__DurationMinutes)
    def Main():
        MyCourse = Course("Computing", 10) # sets up a new course
        MyCourse.AddAssessment("Programming", 100) #adds assessment
        MyCourse.AddLesson("Problem Solving", 60, False)
        MyCourse.AddLesson("Programming", 120, True)
        MyCourse.AddLesson("Theory", 60, False) # check it all works
        MyCourse.OutputCourseDetails()

Main()
```

Refrecences:



Cambridge AS & A level Coursebook



Cambridge AS & A level Teacher's Resource



AS & A level Computer Science by HODDER EDUCATION