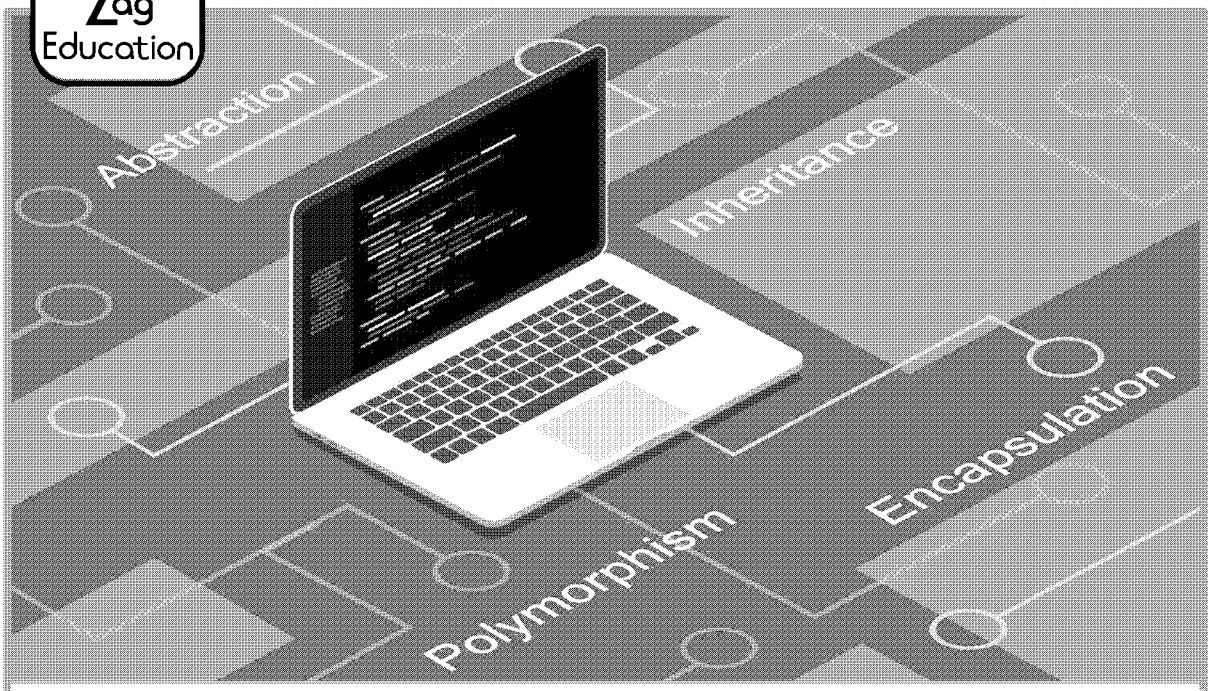
Zig Zag Education

Computer Science



Introduction to Object-Oriented Programming (VB.NET Edition)

Cameron Standring
Richard Lee

DA12/
9760

POD
9760

zigzageducation.co.uk

Contents

TEACHER'S INTRODUCTION

This resource is designed as an introduction to object-oriented programming (OOP), with the aim of taking students with some experience of procedural programming, through to having the required OOP knowledge and skills required for a KS5 course in Computer Science. It is best used by reading and working through the **five topics** in order, as later topics build on the knowledge and skills that students learn in earlier topics.

There are examples of the content and **programming tasks** throughout each topic which provide stretch and challenge for all students through repetition of the topic skills. Review questions have been provided at the end of each topic to test students understanding and application of the theory covered. The answers to these written questions are included in the answer section towards the back of this resource. 'VB Notes' are also included throughout to highlight how VB specifically deals with object-oriented concepts.

Each topic contains pseudocode examples written in these boxes

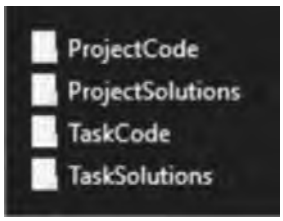
```
These black boxes show where VB.NET code is being used instead of pseudocode
```

There are three **programming projects** for students to test their object-oriented programming skills. These programming projects are independent from each other, can be done in any order (although they are given in order of increasing complexity). For each project, there are two versions of the skeleton code:

1. The *Extended* versions provide only the main method of the program as a starting point
2. The *Basic* versions, in addition to the main method, also include the program's classes and select methods

The basic task provides a version of each project that is less complex; ideal for using when time is limited, or for use with weaker students. Each project also comes with a model solution (with marking guidance).

VB code files are provided electronically on the accompanying CD.



TaskCode contains the skeleton code for the four VB.NET tasks in topics 1-4.
TaskSolutions contains the answer files for each task.

ProjectCode contains the skeleton code for the three VB.NET programming projects.
ProjectSolutions contains the answer files (with marking information written as comments) for each project.

The answer files for both the VB.NET tasks and programming projects provide working programs that contain comments to show where marks should be awarded. In the case of the programming projects, any marks followed by 'ETO' (Extended Task Only) should be awarded only to students attempting the extended version of the project.

In addition to the code files, a **HTML version of the student resources** is also provided. It is recommended that you copy the *IntroToOOP* folder onto your school's secure network, and provide a shortcut to the index.html inside it.

C Standring, July 2019

1. FUNDAMENTALS OF OBJECT-PROGRAMMING

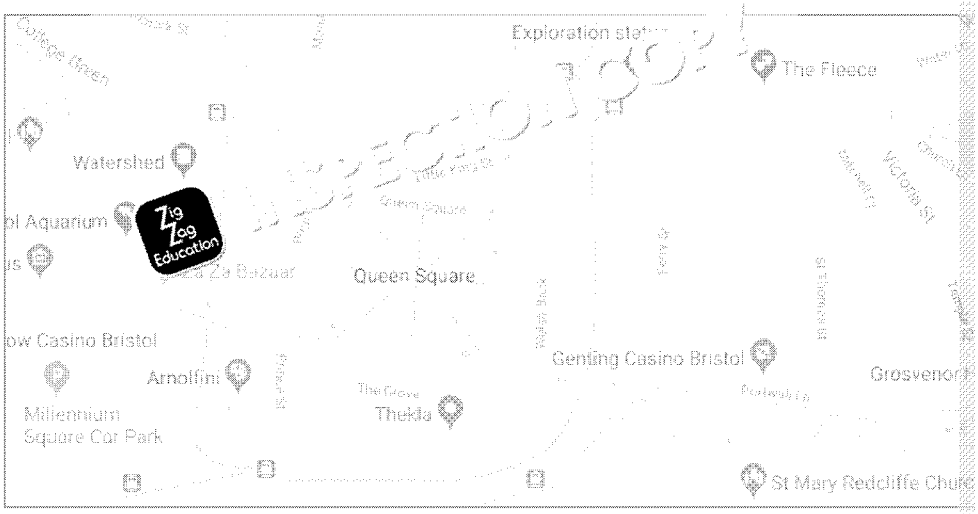
In this chapter you will learn:

- ☑ What object-oriented programming means
- ☑ The differences between procedural programming and object-oriented prog
- ☑ Why the object-oriented programming paradigm is used
- ☑ What classes, objects, attributes and methods (including static attributes and
- ☑ What constructors are and how they are used to instantiate a class
- ☑ How to write basic object-oriented programs

Introduction to object-oriented programming

There are many different programming styles that can be used to create compute (known as **programming paradigms**) that you are likely to be familiar with is **proc** procedural programming, every variable, constant and subroutine is defined sepa relationships between each other.

Another commonly used programming paradigm is **object-oriented programming** define separate **objects** that have their own associated values and subroutines. Th subroutines can be easily grouped together in a logical way. Consider this interac



In an object-oriented program, each location pin would be defined as a different have its own associated values, such as its name, its location and what it is marki restaurant). Each pin object would also have some associated subroutines, such a detailed information if the pin is clicked, or allowing for the pin's information to t

Object-oriented programming is primarily used because of the advantages of thre explained fully in later chapters): **encapsulation**, which allows different parts of th make them easier to understand and work on; **inheritance**, which allows different to still share similar code; and **polymorphism**, which allows subroutines to object is used in a subroutine and what data is passed to it.

Objects and classes

As object-oriented programs can have many different objects, many of which sha
not need to be written to define the properties of each individual object. Instead,
(known as a **class**) is created.

For example, the `Pin` class may look like this:

```
class Pin
    private name //Data belonging to ... in ...ject
    private location
    private markerType

    //Met... cr...ate a new Pin object
    public ...dure new(pinName, pinLocation, pinMarker)
        this.name = pinName
        this.location = pinLocation
        this.markerType = pinMarker
    endprocedure
    ...
endclass
```

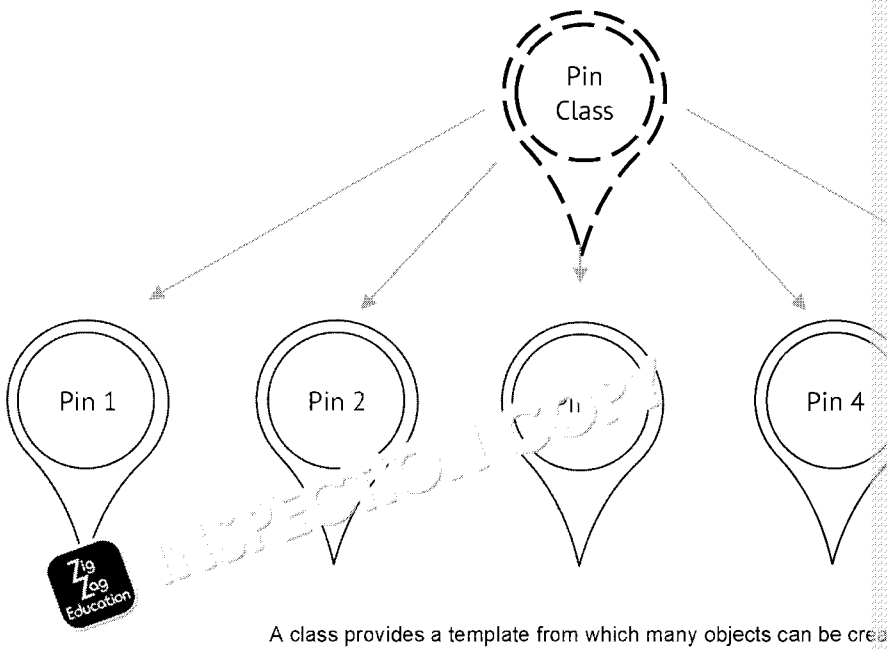
The keyword 'this' is used by an object to refer to itself, so when an object runs the 'new' subroutine, 'th
to the value of 'pinName'

A `Pin` object could then be created using the `Pin` class:

```
templeMeads = new Pin("Bristol Temple Meads", [51.449760, -2.581080], "T
```

The `Pin` class defines the values associated with a `Pin` ...t (in this case, `name`
`markerType`) and the subroutines that a `Pin` ob...ca...perform. The values as
known as **attributes**, and the subroutine...ated with an object are known as

The `new` method is a **con...** ... method that creates an object of a particula
values. The ...ss ...ing an object from a class is known as **instantiation**, a
instance of ... You can think of classes as blueprints, and objects as the indi
blueprints.



A class provides a template from which many objects can be crea

Most attributes and methods are only relevant to a particular object. However, so
attribute or method that is relevant to the class as a whole. These are known as s
methods.

For example, the `Pin` class could include a static attribute to count the number o

```
class Pin
    private name
    private location
    private markerType
    public static noOfPins = 0 //Static attribute that belongs to the class

    public procedure new(pinName, pinLocation, pinMarker)
        this.name = pinName
        this.location = pinLocation
        this.markerType = pinMarker
        Pin.count = Pin.noOfPins + 1
    endprocedure
    ...
endclass
```

Notice that the static attribute is set using 'Pin.count' and not 'this.count' because the attribute belongs t the class. Similarly, static methods are called by 'ClassName.method()' whereas non-static methods ar

If there is an attribute or method in a class that you may want to use, even if ther it should be static.

VB Note

The following pseudocode program:

```
class Pin
    private name
    private location
    private markerType
    public static noOfPins = 0

    public procedure new(pinName, pinLocation, pinMarker)
        this.name = pinName
        this.location = pinLocation
        this.markerType = pinMarker
        Pin.noOfPins = Pin.noOfPins + 1
    endprocedure
    ...
endclass
...
```

... would be written in VB as:

```
Public Class Pin
   Private name As String
   Private location() As Double
   Private markerType As String
   Public Shared noOfPins As Integer = 0

   Public Sub New(ByVal pinName As String, ByVal pinLocation
                  ByVal stringMarker As String)
      name = pinName
      location = pinLocation
      markerType = stringMarker
      noOfPins = noOfPins + 1
   End Sub
End Class
```

In VB, the constructor is always declared with the line 'Public Sub New', and stati the keyword 'shared'.

Questions (Fundamentals of Object-Oriented Programming)

1. Define the term *programming paradigm*.

2. Explain the difference between *object-oriented programming* and *procedural p*

3. Explain the difference between a *class* and an *object*.

4. Identify a situation where a *static method* may be

5. Use ps de to write a *class* with relevant *attributes* and *methods* to repr should represent the time as a 24-hour clock, and include methods to create manually, display the time, and update the time at the end of each minute.

VB Task 1

The *Task 1* skeleton code (`Skeleton`) is part of a program that allows a user to c
their account, check their balance and deposit or withdraw money from their acc
class, that stores information about the individual bank accounts, and a `Bank` cla
accounts and performs operations on individual accounts when asked to by the u

Add the missing attributes and method logic to the *T* ... skeleton code to comp

No changes should be made to M... new methods should be defined, an
methods use should not ... deleted or added to.

Account

```
Public Class Account

    'add integer attribute
    'add string attribute
    Private balance As Double

    'A new bank account should be defined with a given accoun
     'and balance
    Public Sub New(ByVal number As Integer, ByVal password As
                   ByVal balance As Double)
        Me.balance = balance
    End Sub

    'This method should return the account number of this acc
    Public Function getNumber() As Integ

    End Function

    'This method sh... if a given password is equal to
     'this a...
    Publi... ct...n checkPassword(ByVal password As String) A

    End Function

    'This method should return the balance of this account
    Public Function getBalance() As Double

    End Function

    'This method should change the balance of this account to
    Public Sub setBalance(ByVal newBalance As Double)

    End Sub

End Class
```

Bank.vb

```
Public Class Bank

    Private accounts As ArrayList
    Private Shared latestAccount As Integer

    'A new bank is defined with a list of b.. accounts and a
    'track of the account number of th.. lo.. recently added a
    Public Sub New()
        accounts = New Arra..
        latestAccount =
    End Sub

    'This ..od should ask the user to give their account nu
    'retu..ng the account number if they match, or returning
    Public Function login() As Integer

    End Function

    'This method should ask the user how much money they want
    'their account, and correctly update the balance of thei
    Public Sub deposit(ByVal number As Integer)

    End Sub

    'This method should ask the user how much money they want
    'their account, and correctly update the balance of thei
    Public Sub withdraw(ByVal number As Integer)

    End Sub

    'This method should display a m.. ac..elling the user ho
    'their account
    Public Sub checkBal..e(..al number As Integer)

    End Su..

    'This ..od should create a new account with an account
    'account number or the last account created, a password g
    'and a balance of 0. The account should be added to the b
    Public Sub addAccount()

    End Sub

End Class
```

Module1.vb

```
Module Module1

    Sub Main()

        Dim bank As New Bank
        Dim loggedIn As Boolean = False
        Dim quitting As Boolean = False
        Dim accountNo As Integer = -1

        While Not (loggedIn Or quitting)

            Console.WriteLine("Do you have an account? (y/n/quit")
            Dim response As String = Console.ReadLine
            Console.WriteLine()
            If response = "y" Then
                accountNo = bank.login()
                If Not (accountNo = -1) Then
                    loggedIn = True
                End If
            ElseIf response = "n" Then
                bank.addAccount()
            ElseIf response = "quit" Then
                quitting = True
            End If

        End While

        While Not (quitting)
            Console.WriteLine("Press 1 to check your balance")
            console.writeline("Press 2 to deposit money")
            console.WriteLine("Press 3 to withdraw money")
            Console.WriteLine("Press 4 to exit:" & vbCrLf)
            Dim selection As String = Console.ReadLine
            Console.WriteLine()
            If selection = 1 Then
                bank.checkBalance(accountNo)
            ElseIf selection = 2 Then
                bank.deposit(accountNo)
                bank.checkBalance(accountNo)
            ElseIf selection = 3 Then
                bank.withdraw(accountNo)
                bank.checkBalance(accountNo)
            ElseIf selection = 4 Then
                quitting = True
            Else
                Console.WriteLine("Invalid option selected")
                Console.WriteLine()
            End If
        End While

    End Sub

End Module
```

2. ENCAPSULATION

In this chapter you will learn:

☑ What encapsulation is
☑ Why encapsulation is used
☑ How to use private and protected attributes and methods
☑ How to properly encapsulate a program

Encapsulation in object-oriented programming

As mentioned in Chapter 1, **encapsulation** is the idea of grouping data and subro
to work on and understand. In object-oriented programming, encapsulation is ac
should only contain the attributes and methods that it needs, and none of the lo
on the internal processing of another class.

Imagine a company has a system that stores various information about different e
use encapsulation, any data can be used or altered in any part of the program. Th

1. If any errors occur it will be much harder to identify the source of the erro
   anywhere. In a properly encapsulated program, any errors will originate ei
   that isn't working correctly, or from an error in how different parts of the p
2. It means that some parts of the system will have access to attributes and
   have access to. In the example of a company's employee information sys
   employee to be able to update some of their own records (such as their n
   shouldn't have access to change other information (such as their salary) o
   information about other employees (such as their addresses).

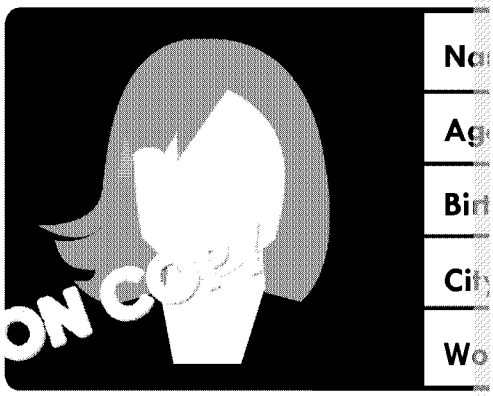
Private attributes and methods

```
class Account
    private accountPassword //This attribute is private
    ...
    public function checkPassword(password) //This method is public
    ...
endclass

class Bank
    private accounts //This attribute is private
    ...
    public procedure withdraw(number) //This method is public
    ...
endclass
```

Take the above program from *Task 1*. Notice how **private** and **public** keywords ar
method is private, it can only be accessed from within the class. If an attribute o
by other classes. In *Task 1*, the attribute `accountPassword` in the `Account` c
`Bank` class cannot access passwords which it shouldn't have access to. Instead, the
method `checkPassword` to check whether a user has entered their password c

This is useful for security purposes (in a real-world system, the more a password i
more vulnerable it is to being stolen) and for encapsulating the program (the Ba
what the password is, it only needs to be able to check that a password is correc
that data).

```
class Account
    public profileImage
    public name
    private age
    public birthday
    public city
    private work
    ...
```



Private attribu e h en from other parts of the system. There are many situations where you stored about a r object.

When an attribute from another class is needed, instead of making that attribute method that returns the value of the attribute. Similarly, to change the value of a public method to change its value rather than directly altering it. Methods that r attribute are known as **accessors** (or 'getters'), and methods that alter the value o as **mutators** (or 'setters'). These may at first appear unnecessary, but can be usefu functionality of the class.

For example, imagine you have the following code:

```
class Clock
    public currentTime //This attribute is public
    ...
endclass

class Display()
    clock = new Clock()
    ...
     public procedure
         print( .ime)//currentTime is public so can be called in Display
    endpr e
endclass
```

The `Display` class directly accesses the clock's `currentTime` attribute to displ if you wanted to make a change to how the clock's time is displayed (e.g. by maki 24-hour clock, or changing whether seconds or milliseconds are displayed) and th `currentTime` in multiple places, then formatting or other checks would need to throughout the `Display` class, which could mean changing a lot of code.

The program could be instead be written as:

```
class Clock
    private currentTime //currentTime is made private
    ...
     public function getTime() //This public metho s ccess to currentTi
         return this.currentTime
    endfunction
endclass

class Displ
    clock = lock()
    ...
     public procedure showTime()
         print(clock.getTime()) //The public method is called in Display
    endprocedure
endclass
```

With this version of the program, the change could be made to the `getTime` acc
does not need to be updated. Accessors and mutators should not just be used to
but to hide information from other classes or limit the ability of other classes to

VB Note

In VB, methods are public by default but attributes are pri e, so:

```
Public Class Pin
   Dim privateAttribute As String

   Sub doSomethi
   End S

End Clas
```

...is equivalent to:

```
Public Class Pin
   Private privateAttribute As String

   Public Sub doSomething()
   End Sub

End Class
```

While it is therefore not necessary to declare an attribute as private or method as
to do so to make it clearer how you intend that attribute or method to be used.

Questions (Encapsulation)

1. Define the term *encapsulation*.

2. Explain the difference between a *private* attribute or method and a *public* attribute or method.

3. Explain one reason why an attribute may be made *private*.

4. Define the terms *accessor* and *mutator*.

5. Identify when *accessors* and *mutators* should be used.

6. Explain why you might make an attribute public instead of using *accessors* and *mutators*.

VB Task 2

The *Task 2* skeleton code is a system that manages a hotel and its staff. Custome
their rooms, and leave feedback depending on how their stay was (if they are su
room is clean they become happier with their stay, and if their room is overbooke
happy with their stay).

Recreate the *Task 2* non-encapsulated code (Non-En[illegible]ted) so that it kee
properly encapsulated. There should be clas[illegible] fo[illegible]el, Room, Customer, Ma
Cleaner. The manager should be [illegible] for processing feedback; the clean
cleaning rooms; the recepti[illegible]ld be responsible for checking customers in
attributes sh[illegible]be [illegible]vate (although you may add any methods that you th

You may use [illegible]ovided *Task 2* encapsulated skeleton code (EncapsulatedSk
converted main method and constructors for each class that do not need to be alte

---

Non-Encapsulated

Cleaner.vb

```
Public Class Cleaner

    Public name As String

    Public Sub New(ByVal name As String)
        Me.name = name
    End Sub

End Class
```

Customer.vb

```
Public Cl[illegible]Cu[illegible]

    Public[illegible]omBooking As Integer
    Public name As String
    Public feedback As Integer

    Public Sub New(ByVal roomBooking As Integer, name As Stri
        Me.roomBooking = roomBooking
        Me.name = name
        feedback = 0
    End Sub

End Class
```

Hotel.vb

```
Public Class Hotel

    Public rooms As ArrayList

    Public Sub New(By[illegible]o[illegible]s As ArrayList)
        Me.rooms [illegible]s
    End S[illegible]

End Class
```

Manager.vb

```
Public Class Manager

    Public name As String

    Public Sub New(ByVal name As String)
        Me.name = name
    End Sub

End Class
```

Module1.vb

```
Module Module1

    Public Sub addOccupant(ByVal room As Room, ByVal occupant
        If room.occupants.Count < room.size Then
            room.occupants.Add(occupantIn)
            occupantIn.feedback += 1
        Else
            occupantIn.feedback -= 1
            Return
        End If
        If room.clean = True Then
            occupantIn.feedback += 1
        Else
            occupantIn.feedback -= 1
        End If
        room.clean = False
    End Sub

    Public Sub removeOccupant(ByVal room As Room, ByVal occup
        Dim index As Integer = -1
        For i = 0 To room.size - 1
            Try
                If room.occupants(i).Equals(occupantOut) Then
                    index = i
                End If
            Catch ex As Exception

            End Try
        Next
        If index > -1 Then
            room.occupants.RemoveAt(index)
        End If
    End Sub

    Public Sub cleanRooms(ByVal cleaner As Cleaner, ByVal hot
        Dim room As Room
        For i = 0 To hotel.Count - 1
            room = hotel(i)
            'Console.WriteLine("Room number " & (i + 1) & " ha
                              room.occupants.Count & " occup
            If room.occupants.Count = 0 Then
                room.clean = True
                Console.WriteLine(cleaner.name & " cleaned roo
            End If
        Next
    End Sub

    Public Sub checkIn(ByVal receptionist As Receptionist, By
                       ByVal customer As Customer)
        addOccupant(hotel(customer.roomBooking - 1), customer)
        Console.WriteLine(receptionist.name & " checked in " &
    End Sub
```

```
    Public Sub checkOut(ByVal receptionist As Receptionist, B
                        ByVal customer As Customer, ByVal man
        removeOccupant(hotel(customer.roomBooking - 1), custo
        Console.WriteLine(receptionist.name & " checked out "
        takeFeedback(manager, customer)
    End Sub

    Public Sub takeFeedback(ByVal manager As Manager, ByVal c
        If customer.feedback > 0 Then
            Console.WriteLine(manager.name & " says: " & custo
                              their stay!")
        ElseIf customer.feedback < 0 Then
            Console.WriteLine(manager.name & " says: " & custo
                              with their stay!")
        Else
            Console.WriteLine(manager.name & " says: " & custo
                              stay ok.")
        End If
    End Sub

    Sub Main()

        Dim room1 As Room = New Room(1, 1, False)
        Dim room2 As Room = New Room(2, 2, False)
        Dim room3 As Room = New Room(3, 1, False)

        Dim hotel As New ArrayList
        hotel.Add(room1)
        hotel.Add(room2)
        hotel.Add(room3)

        Dim customer1 As New Customer(1, "Mr. White")
        Dim customer2 As New Customer(2, "Mr. Green")
        Dim customer3 As New Customer(2, "Miss. Scarlett")
        Dim customer4 As New Customer(3, "Mrs. Peacock")
        Dim customer5 As New Customer(2, "Prof. Plum")
        Dim customer6 As New Customer(3, "Col. Mustard")

        Dim receptionist As New Receptionist("Jane")
        Dim cleaner As New Cleaner("Michael")
        Dim manager As New Manager("Janhavi")

        checkIn(receptionist, hotel, customer1)
        checkIn(receptionist, hotel, customer2)
        checkIn(receptionist, hotel, customer3)
        checkOut(receptionist, hotel, customer1, manager)

        cleanRooms(cleaner, hotel)

        checkIn(receptionist, hotel, customer4)
        checkOut(receptionist, hotel, customer4, manager)
        checkIn(receptionist, hotel, customer5)
        checkOut(receptionist, hotel, customer5, manager)
        checkOut(receptionist, hotel, customer2, manager)
        checkOut(receptionist, hotel, customer3, manager)

        cleanRooms(cleaner, hotel)

        checkIn(receptionist, hotel, customer6)
        checkOut(receptionist, hotel, customer6, manager)
        Console.ReadLine()

    End Sub

End Module
```

Receptionist.vb

```
Public Class Receptionist

    Public name As String

    Public Sub New(ByVal name As String)
        Me.name = name
    End Sub

End Class
```

Room.vb

```
Public C
    Public number As Integer
    Public size As Integer
    Public occupants As ArrayList
    Public clean As Boolean

    Public Sub New(ByVal number As Integer, ByVal size As Int
                   ByVal clean As Boolean)
        Me.number = number
        Me.size = size
        Me.clean = clean
        occupants = New ArrayList
    End Sub

End Class
```

EncapsulatedSkeleton

Cleaner.vb

```
Public C
    Private name As String

    Public Sub New(ByVal name As String)
        Me.name = name
    End Sub

End Class
```

Customer.vb

```
Public Class Customer

    Private roomBooking As Integer
    Private name As String
    Private feedback As Integer

    Public Sub New(ByVal r
        Me.roomBookin
        Me.name =
        f
    End S

End Class
```

Hotel.vb

```
Public Class Hotel

    Private rooms As ArrayList

    Public Sub New(ByVal rooms As ArrayList)
        Me.rooms = rooms
    End Sub

    Public Function checkRooms() As ArrayList
        Return rooms
    End Function

End Class
```

Manager

```
Public Class Manager

    Private name As String

    Public Sub New(ByVal name As String)
        Me.name = name
    End Sub

End Class
```

Module1.vb

```
Module Module1

    Sub Main()

        Dim rooms As New ArrayList

        rooms.Add(New Room(1, 1, False))
        rooms.Add(New Room(2, 2, True))
        rooms.Add(New Room(3, 1, False))

        Dim hotel As New Hotel(rooms)

        Dim customer1 As New Customer(1, "Mrs. White")
        Dim customer2 As New Customer(2, "Mr. Green")
        Dim customer3 As New Customer(2, "Miss. Scarlett")
        Dim customer4 As New Customer(3, "Mrs. Peacock")
        Dim customer5 As New Customer(2, "Prof. Plum")
        Dim customer6 As New Customer(3, "Col. Mustard")

        Dim receptionist As New Receptionist("Jane")
        Dim cleaner As New Cleaner("Michael")
        Dim manager As New Manager("Janhavi")

        receptionist.checkIn(hotel, customer1)
        receptionist.checkIn(hotel, customer2)
        receptionist.checkIn(hotel, customer3)
        receptionist.checkOut(hotel, customer1, manager)

        cleaner.cleanRooms(hotel)

        receptionist.checkIn(hotel, customer4)
        receptionist.checkOut(hotel, customer4, manager)
        receptionist.checkIn(hotel, customer5)
        receptionist.checkOut(hotel, customer5, manager)
        receptionist.checkIn(hotel, customer2, manager)
        receptionist.checkOut(hotel, customer3, manager)

        cleaner.cleanRooms(hotel)

        receptionist.checkIn(hotel, customer6)
        receptionist.checkOut(hotel, customer6, manager)
        Console.ReadLine()

    End Sub

End Module
```

Receptionist.vb

```
Public Class Receptionist

    Private name As String

    Public Sub New(ByVal name As String)
        Me.name = name
    End Sub

End Class
```

Room.vb

```
Public C
    Private number As Integer
    Private size As Integer
    Private occupants As ArrayList
    Private clean As Boolean

    Public Sub New(ByVal number As Integer, ByVal size As Int
                   ByVal clean As Boolean)
        Me.number = number
        Me.size = size
        Me.clean = clean
        occupants = New ArrayList
    End Sub

End Class
```

3. INHERITANCE AND ABSTRAC

In this chapter you will learn:

- ☑ What inheritance is and what parent and child classes are
- ☑ What super methods are
- ☑ What interfaces and abstract methods are and h[illegible]y [illegible]re used
- ☑ How to create object-oriented program[illegible] wit[illegible] inheritance

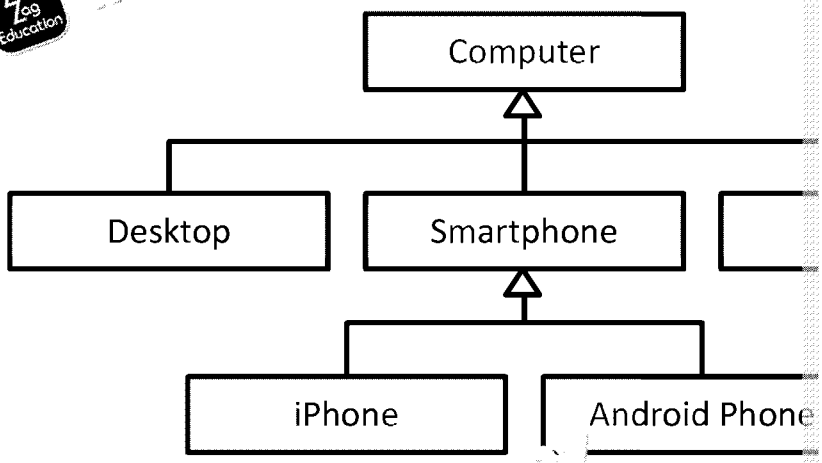
Inheritance

When creat[illegible]sses, you may begin to realise that some classes have certain s[illegible] code being duplicated across classes.

For example, in *Task 2*, the `Manager`, `Receptionist`, `Cleaner` and `Custome` `name` attribute:

```
class Customer
      private roomBooking
    private name //Customer has a 'name' attribute
    ...
endclass

class Receptionist
    private name //Receptionist has a 'name' attribute
    ...
endclass
```

This is not too much of an issue when th[illegible] very few similarities, but as soon multiple identical attributes [illegible] identical methods, a lot of time can be spen one class to another. Object-oriented programming solves this dilemma through i



You can find examples of inheritanc[illegible]e. The above image shows the in different types of computer. Desktops, smartphones and laptops are all specific t[illegible] and Android phones are both specific types of smartphone. Inheritance only work come up w[illegible] fi[illegible]ion for a smartphone, then that definition will also work fo work for eve[illegible]mputer.

Simply put, inheritance refers to when one class copies the characteristics of ano or alter that class's methods or attributes. This means that a generic class can be basic characteristics of a class, as shown on the next page.

```
class Bird
    protected featherColour

    public procedure new(colour)
        featherColour = colour
    endprocedure

    public procedure fly()
        print("This bird is flying")
    endprocedure
endclass
```

Note the use of the protected keyword on the `featherColour` attribute. If an attribute is protected it can only be accessed from within that class or from a class that inherits from it. The `Bird` class defines the attributes and abilities that every bird has. However, we may want a more specific class that builds on this one:

```
class SwimmingBird inherits Bird //Inherits the attributes and methods of Bird
    public procedure swim()
        print("This bird is swimming")
    endprocedure
endclass
```

Note that this class has no defined constructor. It uses the same constructor as its parent class and so

A class that inherits from another class is known as a **child** class, and the class that it inherits from is its **parent** class. In this case, the class `SwimmingBird` is a child of the parent class `Bird`. The `SwimmingBird` class keeps all of the characteristics defined in the `Bird` class,

In the following code:

```
public procedure main()
    bird1 = Bird("Brown")
    bird2 = SwimmingBird("Brown")

    bird1.fly()
    bird2.fly() //SwimmingBirds can use the 'fly' method defined in Bird
    bird2.swim()
    bird1.swim() //Birds cannot use the 'swim' method defined in Swimming
endprocedure
```

`bird1.fly()` and `bird2.fly()` will both print 'This bird is flying' because although `bird2` does not define the `fly` method, it can still use it because it is defined in its parent class.

`bird2.swim()` will print 'This bird is swimming', `bird1.swim()` will cause an error because `bird1` does not have access to the methods defined in the child class `SwimmingBird`.

Inheritance is not just used to add to a parent class, but it can also be used to change how a parent class act in the child class:

```
class Flamingo inherits SwimmingBird
    public procedure new()
        super.new("Pink") //Calls the 'new' method from the SwimmingBird
        print("New flamingo created")
    endprocedure
endclass
```

A class can inherit from a class that inherits from yet another class

The `Flamingo` class inherits from the `SwimmingBird` class, but defines a new co
a `Flamingo` object is created, the inherited `Bird` constructor will be replaced by t

The `Flamingo` constructor uses what is known as a **super method**. Super metho
class's parent class. So, in the previous case, `super.new()` calls the `new` metho
(which in this case is just the `new` method from the `Bird` class).

A parent class can have multiple child classes that inherit from it too, along with the al

```
class FlightlessBird inherits Bird //SwimmingBird and FlightlessBird both inhe
    public procedure fly()
        print("This bird cannot fly")
    endprocedure
endclass
```

Any number of child classes can expand on the same parent class in different way

Abstract classes

While a class can have multiple child classes without causing any problems, if a c
than one class (known as **multiple inheritance**) there can be issues regarding wha
should inherit. For example, if a class tried to inherit from multiple classes as foll

```
class A
    public procedure method()
        print("Do this")
    endprocedure
endclass

class B inherits A //Inherits 'method' from A
endclass

class C inherits A //Inherits 'method' from A
    public procedure method() //Changes 'method'
        print("Do that")
    endprocedure
endclass

class D inherits B, C //Class D inherits 'method' from B and C
endclass
```

It is not clear if the result of `method()` in class `D` should be 'Do this' (as a result
'Do that' (as a result of inheritance from class C). Some programming languages w
multiple classes, while others try to solve this issue by defining a priority for each
the classes are arranged (but this can be confusing or unintuitive to work with).

One way to get around the issues with multiple inheritance is to use **abstract clas**
that declares methods without specifying how they work; these methods are know
example, the following class would be an abstract class:

```
class AbstractClass
    public procedure concreteMethod() //This method is not abstract
        print("Result")
    endprocedure

public abstract procedure abstractMethod(number) //This method is abstra
    endprocedure
endclass
```

An abstract method only defines the method's name and parameters (and the dat parameters in some programming languages) of the method. Any class that conta abstract class, but an abstract class can contain non-abstract methods.

You cannot create an object from an abstract class, and any class that inherits fro all of its parent class' abstract methods using the specified parameters for each n any of the abstract methods undefined, then it will b... ...stract class itself.

```
class ConcreteClass inherits Ab...
    public procedure ab... ...od(number) //This method is no longer ab
        print(...
    endpr... ...e
endclass
```

`ConcreteClass` defines the abstract method in `AbstractClass`, and as it ha methods remaining, `ConcreteClass` objects can be created.

Abstract methods are useful because they tell the programmer what functionality without defining a generic method in the parent class that may not be useful. Fo class `Dog`, with child classes for different breeds of dog, there are some function to implement, although they may all implement it differently.

```
class Dog
    public abstract procedure whatBreed()
    endprocedure
endclass

class Labrador inherits Dog
    public procedure whatBreed()
        print("This dog is a ...")
    endprocedure
endclass
```

A generic version of the `whatBreed` method could be defined in the `Dog` class, t `Dog` would need to replace the method anyway, so declaring it as an abstract me Abstract methods should be used when all child classes require a certain method implementation.

Some languages, including Visual Basic don't permit inheritance from multiple su implementation of multiple interfaces. Interfaces are similar to superclasses, but **not** constructors or attributes.

VB Note

VB does not allow for multiple inheritance, so the following would not be a valid

```
Public Class First
   Inherits Second, Third...
```

In VB, abstract methods cannot be given any function and must be declared
`MustOverride` keyword. The class itself must also be declared using the Mu

```
Public MustInherit Class AbstractClass

   Public MustOverride Sub abstractMethod()

End Class
```

The method `abstractMethod` has been declared as an abstract method with
parameters. This means that any child classes will need to implement a method
requires no parameters and returns no values in order to be instantiated.

In Visual Basic, super methods are implemented by using the keyword 'MyBase'.
declared as follows:

```
Public Class Parent

   Private a As Integer
   Protected b As Integer

   Public Sub New()
      a = 1
      b = 2
   End Sub

End Class

Public Class Child
   Inherits Parent

   Public Sub New()
      MyBase.New
      b += 1
   End Sub

End Class
```

... the `Child` constructor will use `MyBase.New` to run the constructor metho
increases the value of `b` by 1. When a `Child` object is created, it will not have a
attribute and so not inherited, and it will have an attribute `b` with a value of 3.

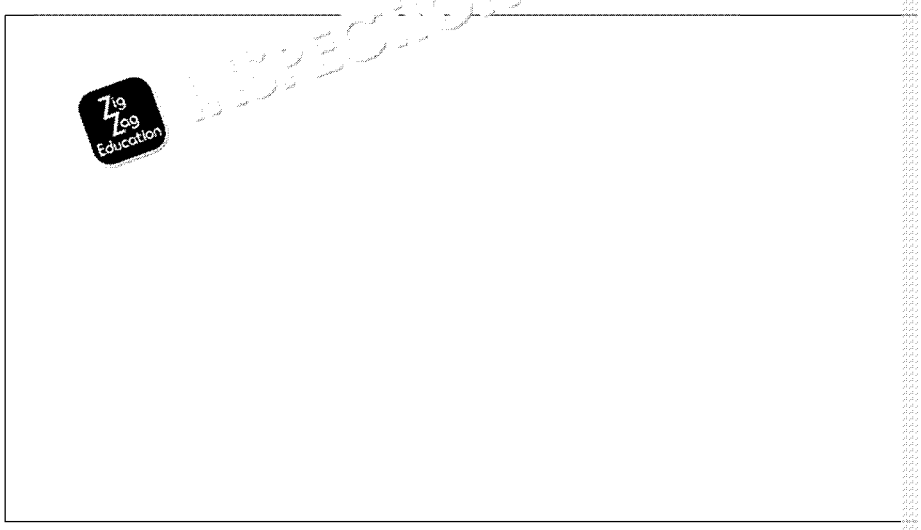
Questions (Inheritance and Abstract Methods)

1. Define the term *inheritance*.

.....

.....

2. Draw a diagram to show inheritance relationships between at least three thi



3. Use the pseudocode below to answer the questions that follow:

```
class Guitar
    private noOfStrings = 6

    public procedure holdFret()
    ...
    endprocedure

    public procedure strum()
    ...
    endprocedure
endclass

class ElectricGuitar inherits Guitar
    public procedure adjustVolume()
        ...
    endprocedure
endclass
```

a) Identify the parent class and the child class.

.....

b) Identify the attributes and methods that ElectricGuitar inherits from Gu

.....

.....

.....

4. Describe what happens when a class calls a *super* method.

.....

.....

5. Explain the issue caused by allowing *multiple inheritance*

.....

.....

.....

.....

6. Define the term *abstract method*, and explain when you might use an *abstract*

.....

.....

.....

.....

VB Task 3

The *Task 3* skeleton code (`Skeleton`) contains classes for various animals, desc
and what actions they can do.

Recreate the *Task 3* skeleton code so that it keeps the same functionality but ad
`Animal`, `Reptile` and `Mammal`. The `Animal` class should include abstract met

Classes should inherit from other classes as appropriate and as much functionali
moved to the three new classes. The main module should not be altered.

Bat.vb

```
Public Class Bat

    Private coldBlooded As Boolean
    Private skinType As String
    Private tail As Boolean
    Private legs As Integer
    Private arms As Integer
    Private wings As Integer

    Public Sub New()
        coldBlooded = False
        skinType = "fur"
        tail = True
        legs = 2
        arms = 0
        wings = 2
    End Sub

    Private Sub move()
        Console.WriteLine("This animal flies")
    End Sub

    Private Sub eat()
        Console.WriteLine("This animal is an omnivore")
    End Sub

    Public Sub birth()
        Console.WriteLine("This animal gives birth to live yo
    End Sub

    Private Sub hibernate()
        Console.WriteLine("This animal hibernates")
    End Sub

    Public Sub getInfo()
        Console.WriteLine("Bat:")
        If coldBlooded Then
            Console.WriteLine("Thi  n  is cold-blooded")
        Else
            Console.Writ ne("This animal is warm-blooded")
        End If
        I  nT  sNot Nothing Then
            sole.WriteLine("The animal is covered in " & sk
        End If
        If tail Then
            Console.WriteLine("This animal has a tail")
        End If
        If legs > 0 Then
            Console.WriteLine("This animal has " & legs & " le
```

```
        End If
        If arms > 0 Then
            Console.WriteLine("This animal has " & arms & " a
        End If
        If wings > 0 Then
            Console.WriteLine("This animal has " & wings & "
        End If
        move()
        eat()
        birth()
        hibernate()
        Console.Write
    End Sub

End Class
```

Gorilla.vb

```
Public Class Gorilla

    Private coldBlooded As Boolean
    Private skinType As String
    Private tail As Boolean
    Private legs As Integer
    Private arms As Integer
    Private wings As Integer

    Public Sub New()
        coldBlooded = False
        skinType = "fur"
        tail = False
        legs = 2
        arms = 2
        wings = 0
    End Sub

    Priva  move()
        Co  e.WriteLine("This animal walks and climbs")
    End Sub

    Private Sub eat()
        Console.WriteLine("This animal is a herbivore")
    End Sub

    Public Sub birth()
        Console.WriteLine("This animal gives birth to live yo
    End Sub

    Public Sub getInfo()
        Console.WriteLine("Gorrilla:")
        If coldBlooded Then
            Console.WriteLine("This an  cold-blooded")
        Else
            Console.WriteL  animal is warm-blooded")
        End If
        If skin  Nothing Then
            sc  .WriteLine("The animal is covered in " & sk
        En
        If tail Then
            Console.WriteLine("This animal has a tail")
        End If
        If legs > 0 Then
            Console.WriteLine("This animal has " & legs & " l
```

```
        End If
        If arms > 0 Then
            Console.WriteLine("This animal has " & arms & " a
        End If
        If wings > 0 Then
            Console.WriteLine("This animal has " & wings & "
        End If
        move()
        eat()
        birth()
        Console.WriteLine()
    End Sub

End Clas
```

Module1.vb

```
Module Module1

    Sub Main()

        Dim tortoise As New Tortoise
        Dim turtle As New Turtle
        Dim snake As New Snake
        Dim otter As New Otter
        Dim gorilla As New Gorilla
        Dim bat As New Bat

        tortoise.getInfo()
        turtle.getInfo()
        snake.getInfo()
        otter.getInfo()
        gorilla.getInfo()
        bat.getInfo()

        Co    e.    Line()
    End Su

End Module
```

Otter.vb

```
Public Class Otter

    Private coldBlooded As Boolean
    Private skinType As String
    Private tail As Boolean
    Private legs As Integer
    Private arms As Integer
    Private wings As Integer

    Public Sub New()
        coldBlooded = False
        skinType = "
        tail = Fa
        l    4
        a    0
        wi   = 0
    End Sub

    Private Sub move()
        Console.WriteLine("This animal walks and swims")
    End Sub
```

```
    Private Sub eat()
        Console.WriteLine("This animal is an omnivore")
    End Sub

    Public Sub birth()
        Console.WriteLine("This animal gives birth to live yo
    End Sub

    Public Sub getInfo()
        Console.WriteLine("
        If coldBlooded Then
            Cons  .WriteLine("This animal is cold-blooded")
        E
            sole.WriteLine("This animal is warm-blooded")
        End If
        If skinType IsNot Nothing Then
            Console.WriteLine("The animal is covered in " & sk
        End If
        If tail Then
            Console.WriteLine("This animal has a tail")
        End If
        If legs > 0 Then
            Console.WriteLine("This animal has " & legs & " le
        End If
        If arms > 0 Then
            Console.WriteLine("This animal has " & arms & " ar
        End If
        If wings > 0 Then
            Console.WriteLine("This animal h  & wings & " w
        End If
        move()
        eat()
        birth()
        Console.W  n()
    End S
End Class
```

Snake.vb

```
Public Class Snake

    Private coldBlooded As Boolean
    Private skinType As String
    Private tail As Boolean
    Private legs As Integer
    Private arms As Integer
    Private wings As Integer

    Public Sub New()
        coldBlooded = True
        skinType = "scales"
        tail = True
        legs = 0
        arms = 0
        wi  =
    End S

    Private Sub move()
        Console.WriteLine("This animal slithers")
    End Sub

    Private Sub eat()
        Console.WriteLine("This animal is a carnivore")
```

```
    End Sub

    Public Sub birth()
        Console.WriteLine("This animal lays eggs")
    End Sub

    Private Sub hibernate()
        Console.WriteLine("This animal hibernates")
    End Sub

    Public Sub getInfo()
        Console.WriteLine("Animal info:")
        If coldBlooded Then
            Console.WriteLine("This animal is cold-blooded")
        Else
            Console.WriteLine("This animal is warm-blooded")
        End If
        If skinType IsNot Nothing Then
            Console.WriteLine("The animal is covered in " & sk
        End If
        If tail Then
            Console.WriteLine("This animal has a tail")
        End If
        If legs > 0 Then
            Console.WriteLine("This animal has " & legs & " le
        End If
        If arms > 0 Then
            Console.WriteLine("This animal has " & arms & " ar
        End If
        If wings > 0 Then
            Console.WriteLine("This animal has " & wings & " w
        End If
        move()
        eat()
        birth()
        hibernate()
        Console.WriteLine()
    End Sub
End Class
```

Tortoise.vb

```
Public Class Tortoise

    Private coldBlooded As Boolean
    Private skinType As String
    Private tail As Boolean
    Private legs As Integer
    Private arms As Integer
    Private wings As Integer

    Public Sub New()
        coldBlooded = True
        skinType = "scales"
        tail = True
        legs = 4
        arms = 0
        wings = 0
    End Sub

    Private Sub move()
        Console.WriteLine("This animal walks")
    End Sub
```

```
    Private Sub eat()
        Console.WriteLine("This animal is a herbivore")
    End Sub

    Public Sub birth()
        Console.WriteLine("This animal lays eggs")
    End Sub

    Private Sub hibernate()
        Console.WriteLine("This animal hibernates")
    End Sub

    Public Sub turtle()
        Console.WriteLine("Tortoise:")
        If coldBlooded Then
            Console.WriteLine("This animal is cold-blooded")
        Else
            Console.WriteLine("This animal is warm-blooded")
        End If
        If skinType IsNot Nothing Then
            Console.WriteLine("The animal is covered in " & sk
        End If
        If tail Then
            Console.WriteLine("This animal has a tail")
        End If
        If legs > 0 Then
            Console.WriteLine("This animal has " & legs & " le
        End If
        If arms > 0 Then
            Console.WriteLine("This animal has " & arms & " ar
        End If
        If wings > 0 Then
            Console.WriteLine("This animal has " & wings & " w
        End If
        move()
        eat()
        birth()
        hibernate()
        Console.WriteLine()
    End Sub

End Class
```

Turtle.vb

```
Public Class Turtle

    Private coldBlooded As Boolean
    Private skinType As String
    Private tail As Boolean
    Private legs As Integer
    Private arms As Integer
    Private wings As Integer

    Public Sub New()
        coldBlooded = True
        skinType = "scales"
        tail = True
        legs = 4
        arms = 0
        wings = 0
    End Sub
```

```
    Private Sub move()
        Console.WriteLine("This animal crawls and swims")
    End Sub

    Private Sub eat()
        Console.WriteLine("This animal is an omnivore")
    End Sub

    Public Sub birth()
        Console.WriteLine("This animal lays eggs")
    End Sub

    Private Sub hibernate()
        Console.WriteLine("This animal hibernates")
    End Sub

    Public Sub getInfo()
        Console.WriteLine("Turtle:")
        If coldBlooded Then
            Console.WriteLine("This animal is cold-blooded")
        Else
            Console.WriteLine("This animal is warm-blooded")
        End If
        If skinType IsNot Nothing Then
            Console.WriteLine("The animal is covered in " & sk
        End If
        If tail Then
            Console.WriteLine("This animal has a tail")
        End If
        If legs > 0 Then
            Console.WriteLine("This an[illegible] & legs & " l
        End If
        If arms > 0 Then
            Console.Wri[illegible]nsole.WriteLine("This animal has " & arms & " a
        End If
        If wings > 0 Then
            Console.WriteLine("This animal has " & wings & " w
        End If
        move()
        eat()
        birth()
        hibernate()
        Console.WriteLine()
    End Sub

End Class
```

4. POLYMORPHISM

In this chapter you will learn:

- ☑ What polymorphism is
- ☑ What the different types of polymorphism are and how they are used
- ☑ What virtual methods are and how they are used
- ☑ How to create object-oriented programs with polymorphism

Polymorphism

Object-oriented languages allow for multiple methods with different implementations but the same name. This is known as **polymorphism** and can come in one of two forms:

1. **Overriding** (which replaces one method with a new method of the same name)
2. **Overloading** (which allows multiple methods with the same name to exist)

Polymorphism is important for object-oriented programming because it means that there can be multiple implementations depending on how it is being used, instead of declaring a different method for each different implementation. This is useful because it allows different parts of a program to be used without needing to know which specific implementation is required, allowing for more flexible program.

Overriding

One of the most common uses for polymorphism is to allow a child class to 'override' a method in the parent class. This is an example of overriding.

Overriding simply replaces one implementation of a method with another. For example, consider the following parent and child classes:

```
class Lizard
    private legs
...
    public procedure new()
        this.legs = 4
        ...
    endclass

class SlowWorm inherits Lizard
...
    public procedure new() //This replaces the 'new' method from Lizard
        this.legs = 0
        ...
endclass
```

The `SlowWorm` class inherits every method and value in the `Lizard` class by default, including the constructor method `new`.

Because a `SlowWorm` doesn't have any legs, it cannot use the same `new` method as a `Lizard`. However, most object-oriented programming languages require constructors to have the same name (in this case 'new'), and also the only way a constructor can be defined for the `SlowWorm` class is to override the `new` method of its parent class, `Lizard`.

The constructor is not the only method that can be overridden. Any method that c
class is known as a **virtual method**. Virtual methods are declared differently depe
language; for example, methods in Java are virtual by default and are made non-
keyword (which prevents child classes from overriding it), whereas in VB the `Ove`
to allow the method to be overridden.

```
class Lizard
...
    public procedure move()
        print("The lizard w... )
    endprocedure
endclass

class Slow... inherits Lizard
...
    final public procedure move() //This method cannot be overridden
        print("The lizard slithers")
    endprocedure
endclass
```

This program overrides the virtual method `move` in the `Lizard` class. The inclus
definition for the `move` method in the `SlowWorm` class means that if another cla
`SlowWorm` it would not be able to override `move` because it is defined as a non-

VB Note

In VB, methods are made virtual using the `Overridabl...` yword:

```
Public Class Lizard

   Public Overridabl... b...ove()
      Console.W...("The lizard walks")
   End S...

End Class
```

The `NotOverridable` keyword, applied to any methods that override the meth
method non-virtual:

```
 Public Class SlowWorm
   Inherits Lizard

   Public NotOverridable Overrides Sub move()
      Console.WriteLine("The lizard Slithers")
   End Sub

End Class
```

A class that inherits from `SlowWorm` wo... b...ble to override the move me
`NotOverridable` keyword wor...re...

Overloading

The other type of polymorphism is overloading. Overloading allows multiple me
combinations of parameters to use the same name. For example, you may want t
when instantiating an object:

```
class Cat
    private age
    private legs

    public procedure ne...() //This 'new' method takes 1 argument
        this.age = ...
        th... = ...
    endpro...e

    public procedure new(age, legs) //A second 'new' method takes 2 argu
        this.age = age
        this.legs = legs
    endprocedure
endclass
...
    mia = new Cat(6)
    percy = new Cat(12, 3) //Both 'new' methods can be called
```

The `Cat` class is defined with two different constructors. The constructor to be u
depending on the arguments that are passed to it; so, when `mia` is instantiated, t
(because one argument is given), but when `percy` is instantiated the second co
arguments are given).

It's possible to have two different constru... w... ne same number of paramet
do not have the same data types ... e order. You could have a string follo
followed by a Boolean. ... even have a string followed by integer for one
followed by ...ng ... nother. The only thing you can't do is create two const
number of p...ters with an identical sequence of data types, even if those par

Unlike overriding, which allows one method to act in place of another, overloadi
methods which simply share a name, so overloading is not considered to be 'true'

VB Note

In VB, the following constructors could exist within the same class:

```
Public Class Cat

   Public Sub New(ByVal name as String, ByVal age as Inte
   End Sub

   Public Sub New(ByVal age as Inte... ByVal Owner as St
   End Sub

End Class
```

This constr... ou... not be included, as its parameters use the same data types
existing co... rs:

```
   Public Sub New(ByVal legs as Integer, ByVal name as St
   End Sub
```

Questions (Polymorphism)

1. Define the term *polymorphism*, and explain why polymorphism is useful.

   .....

   .....

   .....

2. Use the pseudocode below to answer the questions that follow:

```
class Object
    public procedure type()
        print("Type: Object")
    endprocedure

    final public procedure display()
        print(this.value)
    endprocedure
endclass

class Number inherits Object
    ...
    public procedure type()
        print("Type: Number")
    endprocedure

    public function add(num1, num2)
        return(num1 + num2)
    endfunction

    public function add(num1, num2, num3)
        return(num1 + num2 + num3)
    endfunction
endclass
```

   a) State why `Object` cannot override any of the methods in `Number`.

      .....

      .....

   b) Identify the name of an overridden method, and explain why it is an overridden method.

      .....

      .....

   c) Identify the name of an overloaded method, and explain why it is an overloaded method.

      .....

      .....

   d) Identify the name of a virtual method, and explain why it is a virtual method.

      .....

      .....

3. Explain when you would choose to make a method virtual.

   .....

   .....

VB Task 4

The *Task 4* skeleton code (`Skeleton`) provides a series of calls to various metho
and provides expected results for each method call.

Use polymorphism (method overriding/overloading) to implement the methods c
skeleton code so that the expected results are produced. No changes should be m

Shape.vb

```
Public Class Shape

    Priva rToNum As New StringToNumber

End Class
```

StringToNumber.vb

```
Public Class StringToNumber

    Public Function convert(ByVal s As String) As Integer

        If s = "one" Then
            Return 1
        ElseIf s = "two" Then
            Return 2
        ElseIf s = "three" Then
            Return 3
        ElseIf s = "four" Then
            Return 4
        ElseIf s = "five" Then
            Return 5
        ElseIf s = "six" Then
            Return 6
        ElseIf s = "seven" Then
            Return 7
        ElseIf s = "eight" Then
            Return 8
        ElseIf s = "nine" Then
            Return 9
        Else
            Return -1
        End If

    End Function

    Public Function convert(ByVal number As Integer) As Integ
        If number >= 1 And number <= 9 Then
            Return 9
        Else
            Return -1
        End If
    End Function

End Class
```

Module1.vb

```
Module Module1

    Sub Main()

        'Circles have one value: radius
        Dim circle1 As New Shape(2)
        Dim circle2 As New Shape("three")

        'Rectangles have two values: width and height
        Dim rectangle1 As New Shape(5, 3)
        Dim rectangle2 As New Shape("seven", "two")

        'Triangles have three values: the lengths of each side
        Dim triangle1 As New shape("four", "six", "nine")
        Dim triangle2 As New shape(3, 6, 5)

        'You can assume that shapes are either given only int
        'strings with one of the following values:
        '"one", "two", "three", "four", "five", "six", "seven

        'The perimeter of a circle is: 2 x pi x radius
        'The area of a circle is: pi x radius^2
        'You can use 'Math.PI' as the value of pi
        'To square a value, use Math.Pow(a, 2), which would s

        circle1.perimeter() 'Should print "This circle has a p
        circle1.area() 'Should print "This circle has an area

        circle2.perimeter() 'Should print "This circle has a p
        circle2.area() 'Should print "This circle has an area

        'The perimeter of a rectangle is: 2 x (width + height)
        'The area of a rectangle is: width x height

        rectangle1.perimeter() 'Should print "This rectangle h
        rectangle1.area() 'Should print "This rectangle has a

        rectangle2.perimeter() 'Should print "This rectangle h
        rectangle2.area() 'Should print "This rectangle has a

        'The perimeter of a triangle with sides of length a, b
        'The area of a triangle with sides of length a, b and
        'the square root of:
        'p/2 x (p/2-a) x (p/2-b) x (p/2-c)
        'You can use 'Math.Sqrt(value)' to get the square roo

        triangle1.perimeter() 'Should print "This triangle has
        triangle1.area() 'Should print "This triangle has an a

        triangle2.perimeter() 'Should print "This triangle has
        triangle2.area() 'Should print "This triangle has an a

        Console.ReadKey()

    End Sub
End Module
```

5. CLASS RELATIONSHIPS

In this chapter you will learn:

- ☑ What class diagrams are, why they are used, and how to create and understa
- ☑ What composition and aggregation are
- ☑ When composition should be used over inheritan

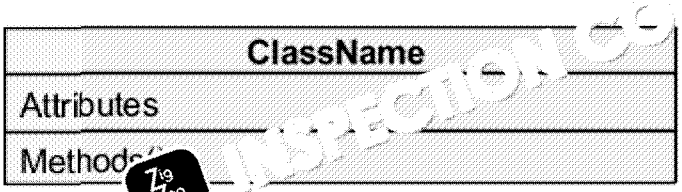
Class diagrams

A very important aspect of object-oriented programs is the relationships between
There are many different types of relationship in object-oriented programs, and a
scope, it can be difficult to understand the relationships between different classe
by using Unified Modelling Language (UML) class diagrams – visualisations that
methods and relationships that form systems.

In a UML class diagram, the following symbols represent the following visibility a
attribute or a method may have:

- Public (+)
- Private (-)
- Protected (#)
- Static (underlined)
- *Abstract* (italics)

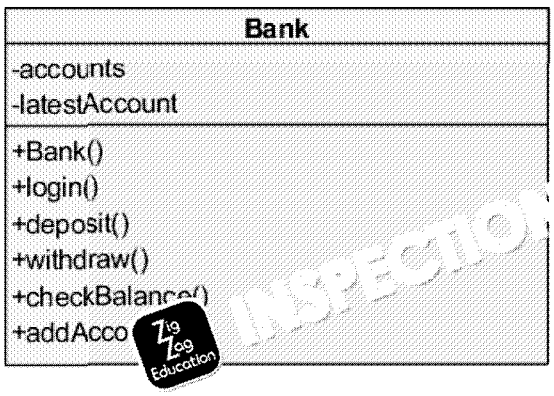
Classes are defined in UML diagrams as follows:



Take, for example, the `Bank` class from *Task 1*:

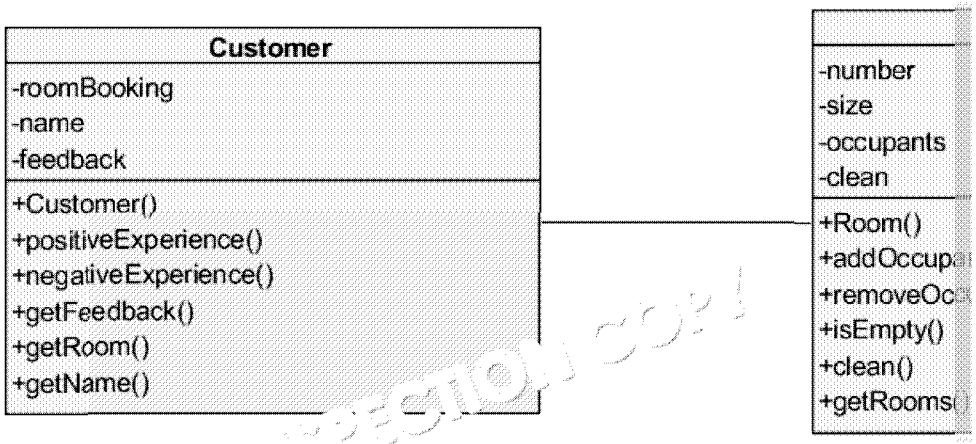
```
class Bank
    private accounts
    private latestAccount

    public procedure new()
        ...
     public function login()
        ...
    public procedure deposit(number)
        ...
    public procedure withdraw(number)
        ...
     public function checkBalance(nu
        ...
     public procedure
        ...
endclass
```

A UML class diagram would represent the `Bank` class as follows. Note that the c
shares the name of the class instead of the keyword 'new':

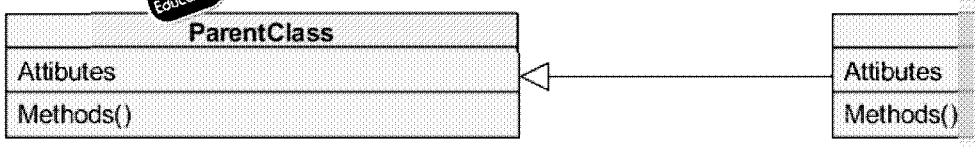


UML class diagrams can sometimes inc
.g. withdraw(number)'), but this is not
choose whether or not to include the pa
are for understanding the system.

UML class diagrams can also show association between classes. For example, in t
`customer` class is associated with the `room` class because there is a relationshi
(i.e. customers have bookings for certain rooms, and each room can contain diffe
his is demonstrated by a line connecting these classes in the diagram as follows:



As well as
a
ations, UML class diagrams can include inheritance relat

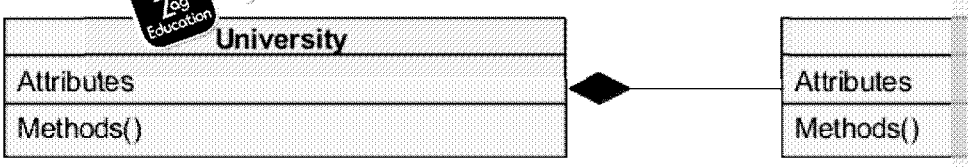


In a UML class diagram, a child class does not need to display the methods and a
parent class, although it can be helpful to display any inherited methods that hav

Composition and aggregation

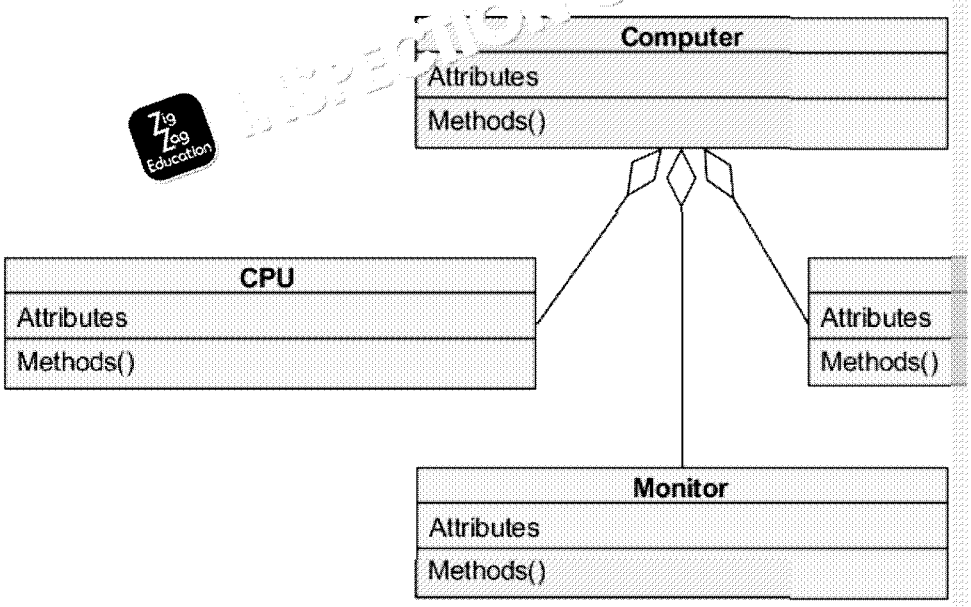
Another type of association between classes that is often seen is called **composit**
composite object is formed from a collection of different component objects, wh
objects can only exist as part of a composite object

For example, a university is formed by a
of different departments. Each d
a department will only exist for as long as the university does. If the university close
This relationship would be shown as follows:



The end of the line with the filled-in diamond is the composite class, and the oth
component class. In this case, `University` is a composite class and `Departme`

There is another type of association whereby several component objects combine
as **aggregation**. Aggregation is very similar to composition, but the component ob
the aggregated object. For example, a computer is made up from a number of sep
component exists as an object in its own right; therefore if you disassemble the c
continue to exist even though the computer no longer exists. This relationship wo



The end of each line with the hollow diamond is the aggregation class, and the o
component classes. In this case, Comput... an aggregated class, while CPU, M
component classes.

There is not always a distinction between composition and aggregation. For
between lecturers and universities a composition, as a lecturer is no longer a lect
or is it an aggregation, because a lecturer still exists as a person without a univer
whatever model is most useful for the system that you are designing.

When designing an object-oriented program, there are some basic principles that

- **Encapsulate what varies** – if the implementation of a particular aspect of a pr
being developed, then it should be encapsulated from the rest of the program
- **Favour composition over inheritance** – inheritance relationships can get comp
created, especially if using multiple inheritance. Instead, it is often better to u
can use parts of many different components while avoiding this complexity.
- **Program to interfaces, not implementation** – use abstract methods wherever p
implementation from a parent class so that you don't have to change child cla
the parent class changes.

Questions (Class Relationships)

1. Explain what *UML class diagrams* are and why they are used.

.....

.....

.....

2. State the similarity between *composition* and *aggregation*.

.....

.....

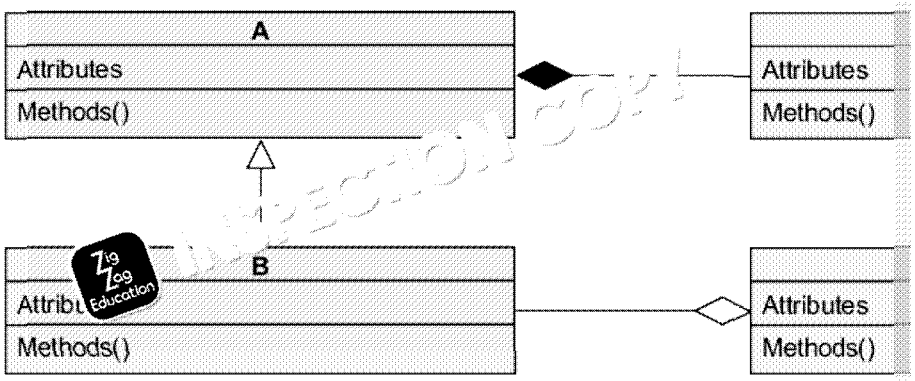
3. Explain the difference between *composition* and *aggregation*.

.....

.....

.....

4. Use the *UML class diagram* below to answer the questions that follow:



a) Describe the relationship between class A and class B.

.....

.....

b) Describe the relationship between class A and class C.

.....

.....

c) Describe the relationship between class B and class D.

.....

.....

UML Class Diagram Tasks

1. Draw a *UML class diagram* for the system created by the '*Task 1 (Answers)*' co
2. Draw a *UML class diagram* for the system created by the '*Task 2 (Answers)*' co
3. Draw a *UML class diagram* for the system created by the '*Task 3 (Answers)*' co
4. Draw a *UML class diagram* for the system created by the '*Task 4 (Answers)*' co

PROJECT 1: FOUR IN A ROW

Introduction

Four in a Row is a game in which players take turns adding tokens to one of the columns on the game board.

Tokens fall to the lowest position in the chosen column that does not already have a token in it. Once one of the players has placed four of their tokens in a straight line (either vertically, horizontally or diagonally), they win the game.

If the board is full and no player has won, then the game ends in a draw.

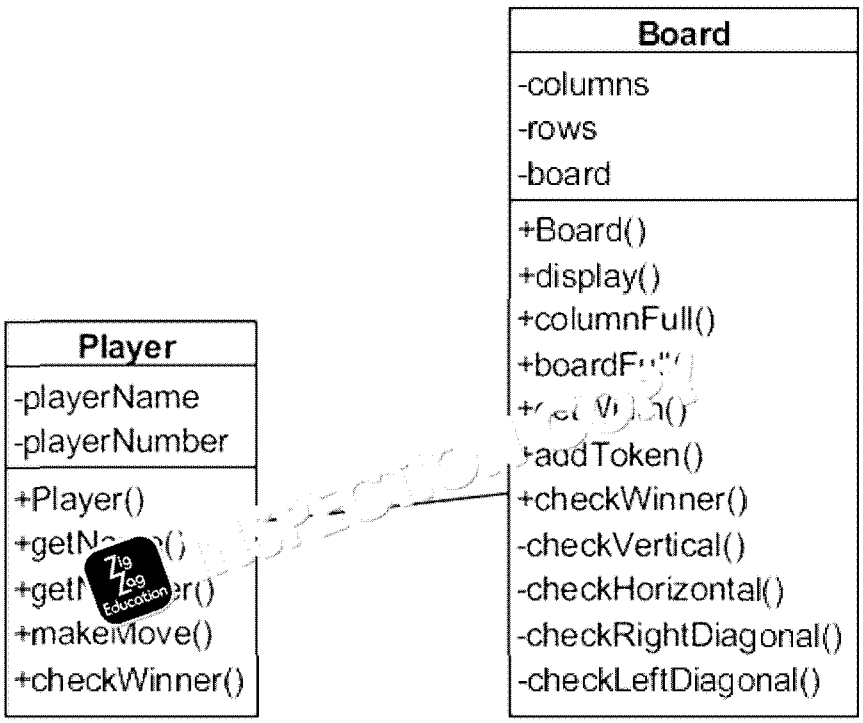
Task

Using the following UML class diagram and class descriptions to help you create Four in a Row.

- The game must allow for a minimum of two and a maximum of four players.
- The game must allow each player to enter their name (duplicate names are accepted)
- The game should give the players the ability to choose how many rows (between four and 10), and how many columns (between four and 10) the game board should have.

You may use the Four in a Row class template code to help you.

UML class diagram



Class Descriptions

Player

Attribute/Method	Description
`playerName`	Specifies this player's name.
`playerNumber`	Specifies the number of this player's token.
`Player()`	Creates a new `Player` object.
`getName()`	Accessor for `playerName`.
`getNumber()`	Accessor for `playerNumber`.
`makeMove()`	Asks the player to pick a column to place their to column is given, and adds their token to the give
`checkWinner()`	Returns the player's name if they have won, or "

Board

Attribute/Method	Description
`columns`	Specifies the number of columns on the game bo
`rows`	Specifies the number of rows on the game board.
`board`	Keeps track of which player's token (if any) is sto board.
`Board()`	Creates a new `Board` object.
`display()`	Displays the current state of the board.
`columnFull()`	Checks whether a given column is full.
`boardFull()`	Checks whether the whole board is full.
`getWidth()`	Accessor for `columns`.
`addToken()`	Adds a given token to a given column.
`checkWinner()`	Checks the board for a winner, returning the win there is no winner.
`checkVertical()`	Checks for vertical lines of four matching tokens of the player who made the line, or `0` if there are
`checkHorizontal()`	Checks for horizontal lines of four matching toke `playerNumber` of the player who made the li lines of four.
`checkRightDiagonal()`	Checks for left-to-right diagonal lines of four ma `playerNumber` of the player who made the li to low-right diagonal lines of four.
`checkLeftDiagonal()`	Checks for right-to-left diagonal lines of four ma `playerNumber` of the player who made the li to low-left lines of four.

PROJECT 2: SINKING SHIPS

Introduction

Sinking Ships is a game in which two players place a number of ships of various length on their own board, which is hidden from the other player.

Players then take turns calling out one of the tiles on their opponent's board. Their opponent then tells them whether the shot hit or missed any of their ships.

Once one player has hit every tile that contains a ship on their opponent

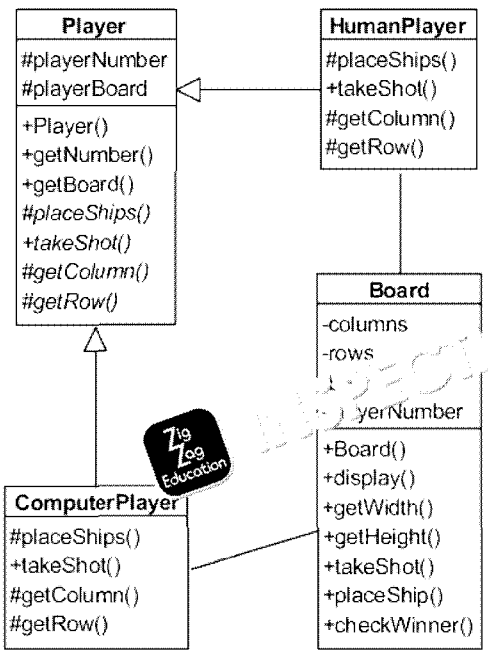
Task

Using the following UML class diagram and class descriptions to help *Sinking Ships.*

- The game must have one human player, and one player control computer player can be made to target random tiles).
- The player should be able to choose how many rows (between 1 columns (between 10 and 26) the game boards should have.
- Players cannot shoot tiles that they have already shot, and befo their shot, they should have the option to look at their own boar their ships and the tiles that the computer player has taken sho shots hit or missed) or the opponent's board showing the tiles t taken shots at, and whether those shots hit or missed, but not s opponent's ships).
- Player input should be given as a number, to indicate the colum and a letter to indicate the row (first row is A, second row is B, e

You may use the Sinking Ships skeleton code to help you.

UML Class Diagram



Class Descriptions

Board

Attribute/Method	Description
`columns`	Specifies the number of columns on the game board.
`rows`	Specifies the number of rows on the game board.
`board`	Keeps track of the ship locations and shot locations on the
`playerNumber`	Specifies the number of the player that the board belongs
`Board()`	Creates a new `Board` object.
`display()`	Displays the current state of the board, only showing ship at their own board.
`getWidth()`	Accessor for `columns`.
`getHeight()`	Accessor for `rows`.
`takeShot()`	Takes a shot at the given location on the board.
`placeShip()`	Asks the player to pick a location on the board and an orie for a ship of a given length until a valid location and orien ship on the board, and then adds the ship to the board as
`checkWinner()`	Checks whether all of the ships on the board have been su

Player

Attribute/Method	Description
`playerNumber`	Specifies this player's number.
`playerBoard`	Specifies this player's board.
`Player()`	Creates a new `Player` object.
`getNumber()`	Accessor for `playerNumber`.
`getBoard()`	Accessor for `playerBoard`.

HumanPlayer

Attribute/Method	Description
`placeShips()`	Places all of this player's ships onto their board, displayin which ship they are placing and to confirm that they have
`takeShot()`	Gets a valid location on a board and takes a shot at it, dis the player another chance to take a shot if they select an
`getColumn()`	Gets a valid column on a board from player input, displayi player another chance to select a column if they select an
`getRow()`	Gets a valid row on a board from player input, displaying another chance to select a row if they select an invalid ro

ComputerPlayer

Attribute/Method	Description
`placeShips()`	Places all of this player's ships onto their board, displayin computer is placing its ships and another message to say have been placed.
`takeShot()`	Gets a valid location on a board and takes a shot at it.
`getColumn()`	Gets a valid column on a board.
`getRow()`	Gets a valid row on a board.

PROJECT 3: CHESS

Your Task

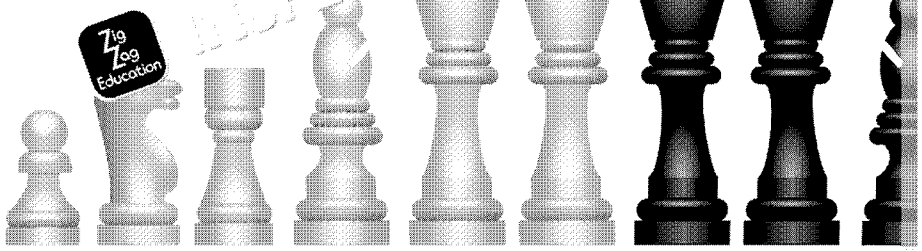
Chess is a game played on an 8 × 8 game board whereby two players h
including one King. Players take turns to move one of their pieces.

Each piece has different rules for how it ca[illegible] around the board, a
onto a tile containing one of the op[illegible]'s pieces, the opponent's pie

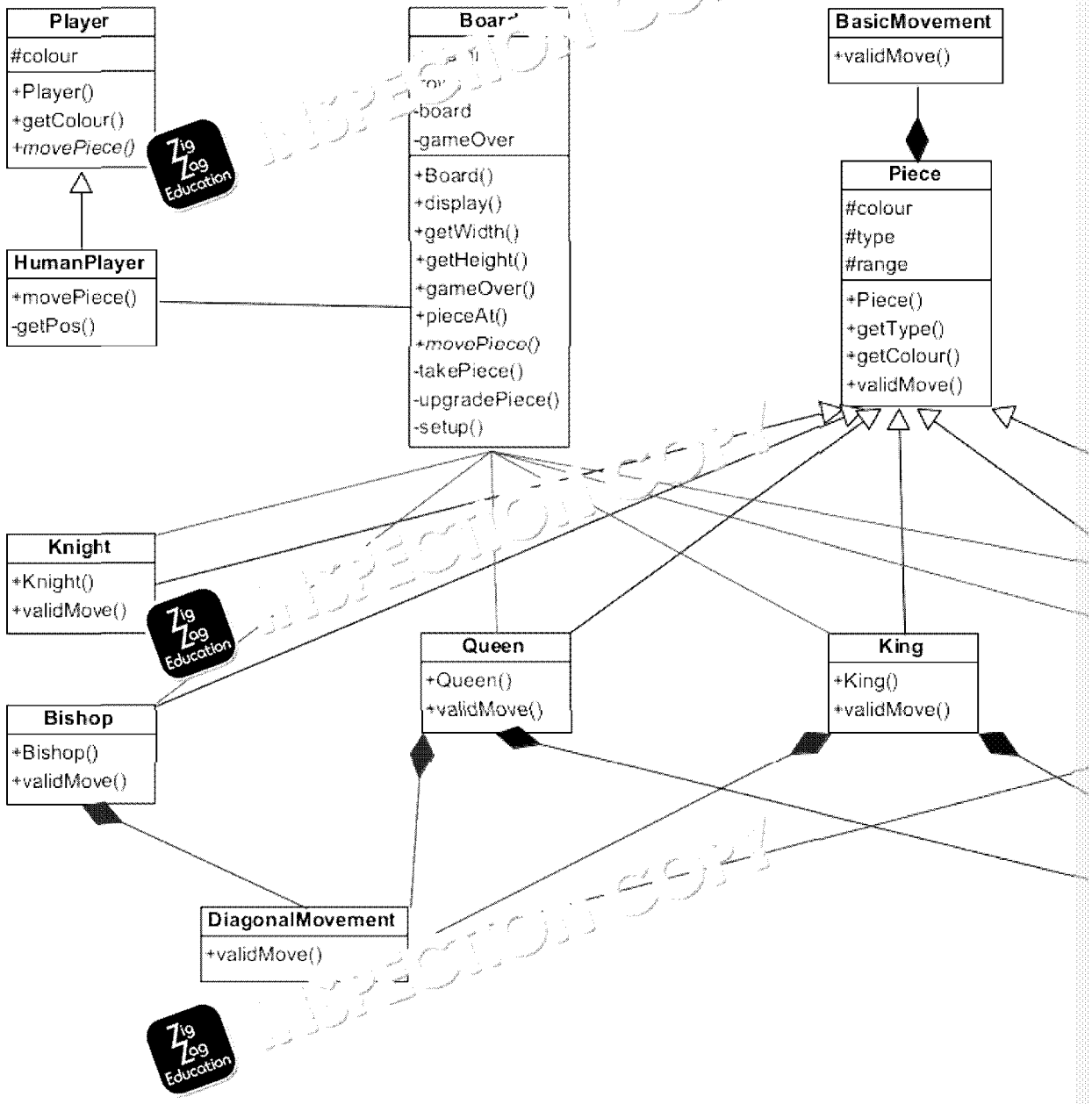
The winner is the play[illegible] manages to take their opponent's King.

Using the [illegible] UML class diagram and class descriptions to help
Chess.

- When it is a player's turn, they must select the tile with the pie
move (if this tile doesn't contain one of their pieces, a message
the player that they don't have a piece on that tile and they sho
selection again) and then select the tile that they would like to
not a valid move, a message should be displayed to tell the play
and they should be asked for their move again).
- Player input should be given as a number, to indicate the colum
and a letter, to indicate the row (first row is A, second row is B,
- You may use the Chess skeleton code to help you.



UML Class Diagram



Class Descriptions

Board

Attribute/Method	Description
`columns`	Specifies the number of columns on the game board.
`rows`	Specifies the number of rows on the game board.
`board`	Keeps track of the piece locations on the game board.
`gameOver`	Specifies whether or not the game has been won.
`Board()`	Creates a new `Board` object.
`display()`	Displays the current state of the board.
`getWidth()`	Accessor for `columns`.
`getHeight()`	Accessor for `rows`.
`getGameOver()`	Accessor for `gameOver`.
`pieceAt()`	Returns the piece at a given location, or none if that location
`movePiece()`	Tries to make a given move, displaying a message saying whether message saying that the move is invalid.
`takePiece()`	Displays a message to say which piece has been taken by which `gameOver = True` if the piece which has been taken is a king
`upgradePiece()`	Replaces a pawn with a queen if it reaches the end of the board
`setUp()`	Sets up the board with all pieces in their starting positions

Player

Attribute/Method	Description
`colour`	Specifies this player's colour.
`Player()`	Creates a new `Player` object.
`getColour()`	Accessor for `colour`.

HumanPlayer

Attribute/Method	Description
`movePiece()`	Asks a player for the start and end locations of their move until locations, then makes the move and returns whether or not it was successful.
`getPos()`	Checks that the player has given a valid location, returning two numbers if one location has been passed to `getPos` or making sure two locations have been passed.

Piece

Attribute/Method	Description
`colour`	Specifies the colour of this piece.
`type`	Specifies the name of this type of piece.
`range`	Specifies the number of tiles this piece can move in a turn.
`Piece()`	Creates a new `Piece` object.
`getPieceType()`	Accessor for `type`.
`getColour()`	Accessor for `colour`.
`validMove()`	When moving, a piece must end on a tile on the board, can't be started on, and cannot land on a tile containing a friendly piece.

Pawn

Attribute/Method	Description
`Pawn()`	Creates a new `Pawn` object, with `type = "pawn"`.
`validMove()`	A pawn can move one tile straight forward if there is no pie diagonally forward if there is an enemy piece in that locatic straight forward if it is in its startin[illegible]sition and there are tiles in front of it.

Knight

Attribute/Method	Description
`Knight()`	[illegible]reates a new `Knight` object, with `type = "knight"`
`validMo[illegible]`	A knight can move two tiles vertically and one tile horizont one tile vertically. A knight can jump over pieces.

Rook

Attribute/Method	Description
`Rook()`	Creates a new `Rook` object, with `type = "rook"`.
`validMove()`	A rook can move any number of tiles in a straight line as lo between it and the end tile.

Bishop

Attribute/Method	Description
`Bishop()`	Creates a new `Bishop` object, with `type = "bishop"`
`validMove()`	A bishop can move any number of ti[illegible]in a diagonal line as between it and the end tile.

Queen

Attribute/Method	Description
`Queen()`	[illegible]reates a new `Queen` object, with `type = "Queen"`.
`validMo[illegible]`	A Queen can make any move that a rook or bishop can mak

King

Attribute/Method	Description
`King()`	Creates a new `King` object, with `type = "King"`.
`validMove()`	A King can move one tile in any direction.

BasicMovement

Attribute/Method	Description
`validMove()`	Makes sure that the given move meets the basic criteria fo

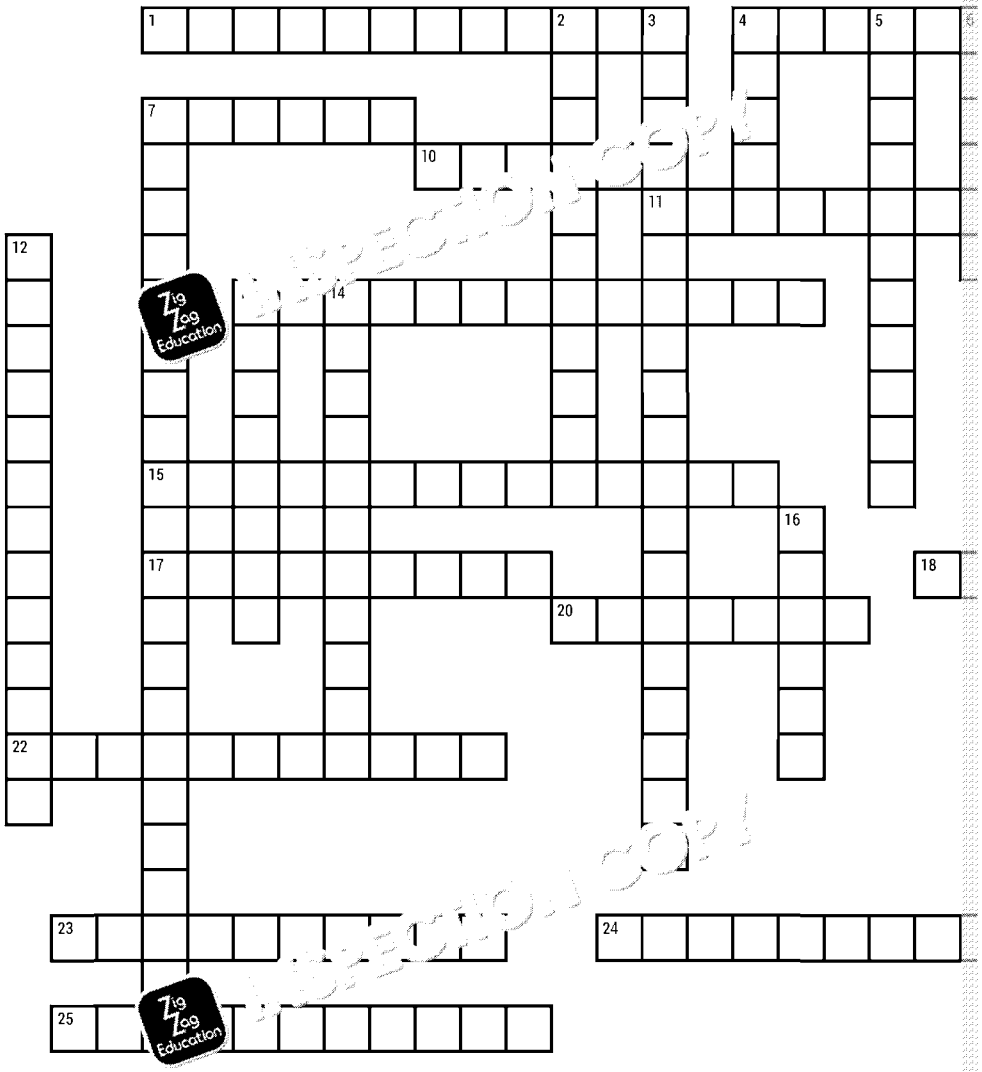
StraightMovement

Attribute/Method	Description
`validMove()`	Make[illegible]e th[illegible]e given move meets the criteria for a vali [illegible]ne.

DiagonalM[illegible]ent

Attribute/Method	Description
`validMove()`	Makes sure that the given move meets the criteria for a vali diagonal line.

CROSSWORD (OOP CONCEPTS)



Across

1 Allowing different implementations of a method to use the same method name (12)

4 Forming one larger object from multiple smaller objects, where the smaller objects cannot exist separately to the larger object (11)

7 A method or attribute of a particular class that can be called by any other class (6)

10 An instance of a data structure that has its own attribute values and associated methods (6)

11 The process of creating an object from a particular class (13)

13 A class that contains abstract methods and cannot be instantiated (8,5)

15 A method whose name and parameters are defined but that d[oes not] have any implementation (8,6)

17 A method or attribute of a particular class tha[t can] be called within that class or its child classes (

18 A public method that chang[es] a private attribute (7)

20 A method or attribute of a particular class that can only be called within th (7)

22 Creating implementations of the same method that take different a[rgu]ment types (11)

23 A class that inherits from another class (5,5)

24 Grouping together related data and subroutines into classes, and providing controlled access to that class's private attributes (13)

25 Forming one larger object from multiple smaller objects, where the smaller objects can exist separately to the larger object (11)

Down

2 When one type of o[bject] a different type of o[bject]

3 When a child class [inherits] (8,11)

4 A template defining [the] object from which [an]

5 A class that is inher[ited]

6 Class method that [can] class have been in[voked]

7 A programming par[adigm] series of steps that

8 A variable or consta[nt] object (9)

9 A method that creat[es]

12 Any method that ca[n be]

13 A public method tha[t] attribute (8)

14 When a child class [can] a method (5,6)

16 A subroutine belong[ing]

19 Superseding the im[plementation] in its child class (10)

21 A particular approa[ch] (8)

ANSWERS

Questions (Chapters 1 – 5)

1 – Fundamentals of Object-Oriented Programming

1. A programming paradigm is a particular style of programming. **(1)**

2. Object-oriented programming ar[illegible] to [illegible]r values and subroutines as objects programming runs through a series of subroutines in sequence. **(1)**

3. A class [illegible] defines what attributes and methods an object should of a cla[illegible]

4. A static method may be used to perform an operation that corresponds to the cl particular object of that class, **(1)** or when the method may be used even if the c **mark**

5. **1 mark** for suitable attributes; **1 mark** for including a constructor method; **1 mar** the time; **1 mark** for including a method to display the time; **1 mark** for displayin **1 mark** for including a method that updates the display time each minute.
Accept any sensible approach that meets the requirements of the question. For e

```
class DigitalClock
    private hours
    private minutes

    public procedure new(currentHour, current[illegible]e)
        this.hours = currentHour
        this.minutes = currentMi[illegible]
    endprocedure

    publ[illegible]rc[illegible]etHour(currentHour)
        h[illegible]rs = currentHour
    en[illegible]edure

    public procedure setMinute(currentMinute)
        this.minutes = currentMinute
    endprocedure

    public procedure displayTime(currentMinute)
        print(this.hours + ":" + this.minutes)
    endprocedure

    public procedure newMinute()
        if this.minutes < 60 then
            this.minutes = this.minutes + 1
        else
            this.minutes = 0
            this.hours = this.hours + 1
    endprocedure

    public pro[illegible]e.Hour()
        [illegible]is [illegible]urs < 24 then
            this.hours = this.hours + 1
        else
            this.hours = 0
    endprocedure
endclass
```

2 – Encapsulation

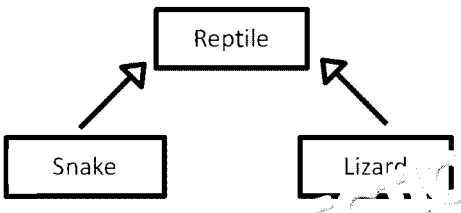
1. Encapsulation is the grouping of data and subroutines that relate to each other
2. If an attribute or method is public, it will be available to any part of the program private, it can only be accessed from within the class in which it is defined. **(1)**
3. An attribute may be made private to prevent it from bein[illegible] incorrectly altered el
4. An accessor is a method that returns the va[illegible] of a private attribute. **(1)** A muta value of a private attribute. **(1)**
5. Accessors are used to [illegible] value of a private attribute outside of its class. value of a private attribute outside of its class. **(1)**
6. If *full* a[illegible] the attribute is needed outside of the class, the attribute should accessors and mutators, **(1)** because accessors and mutators should be used to attributes. **(1)**

3 – Inheritance and Abstract Methods

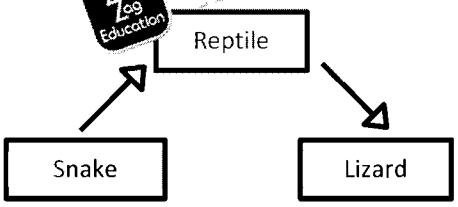
1. Inheritance is when one class uses another class as a base which can then be e
2. 1 mark for showing a correct inheritance relationship; 1 mark if all inheritance

   Accept any diagram that shows a clear inheritance structure between at least th

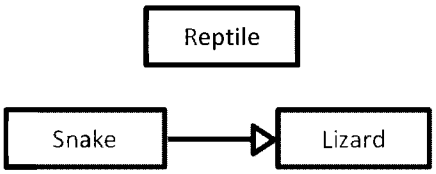
   2 marks: all inheritance relationships are correct between at least three things,



   1 mark: [illegible] inheritance relationship shown, e.g.:



   0 marks: no correct inheritance relationships shown, e.g.:



3.
   a) The parent class is `Guit[illegible]` child class is `ElectricGuitar`. **(1 m the parent class [illegible] child class)**
   b) `El[illegible]` inherits the attribute `noOfStrings` **(1)** and the meth fr[illegible]tar. **(Max. 2 marks if `adjustVolume` is given as an inherited m**
4. When a super method is called, the version of the method in the current class's
5. Multiple inheritance can cause conflicts if a particular method has one implem different implementation in another parent class, **(1)** as the program may be un the method should be inherited. **(1)**

6. An abstract method is a method that has been left undefined in a particular clas
any child class that inherits it. **(1)** You might use an abstract method to highlig
implemented in child classes but that will have a different implementation in ea

4 – Polymorphism

1. Polymorphism is a way of allowing a particular method to have multiple differe
Polymorphism is useful for allowing methods to be called on different data type
the specific implementation of the meth[illegible] **(1**

2. a) `Object` cannot overr[illegible] any [illegible] the methods in `Number` because `Object`
   b) `type` is an [illegible]er method **(1)** because the implementation in the chil
   ir[illegible]nta[illegible]on in the parent class `Object`. **(1)**
   c) `ty[illegible]dd` is an overloaded method **(1)** because it has multiple implemen
   arguments. **(1)**
   d) `type/add` is a virtual method **(1)** because it can be overridden in a child c

3. A method would be made final to prevent child classes that inherit the method f

5 – Class Relationships

1. UML class diagrams are visual representations of object-oriented systems **(1)** th
and understanding object-oriented systems. **(1)**

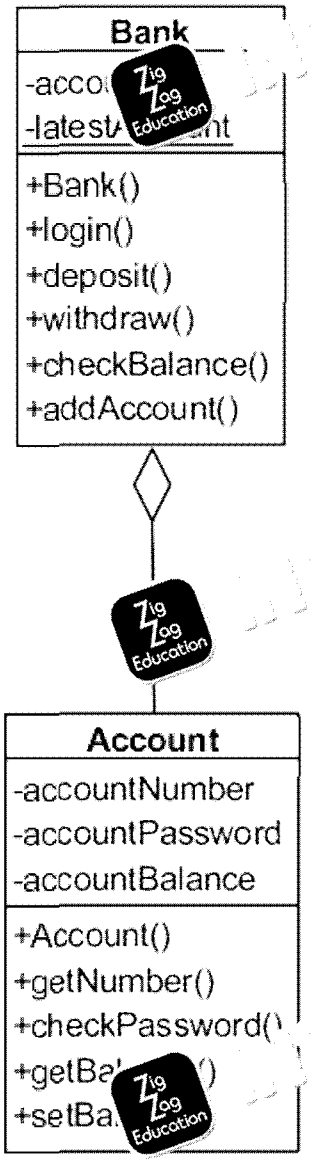
2. Composition and aggregation both form an object from multiple smaller objects

3. The component objects that form a composite object only exist for as long as th
whereas the component objects that form an aggregated [illegible]ject can exist as sep
aggregated object is destroyed. **(1)**

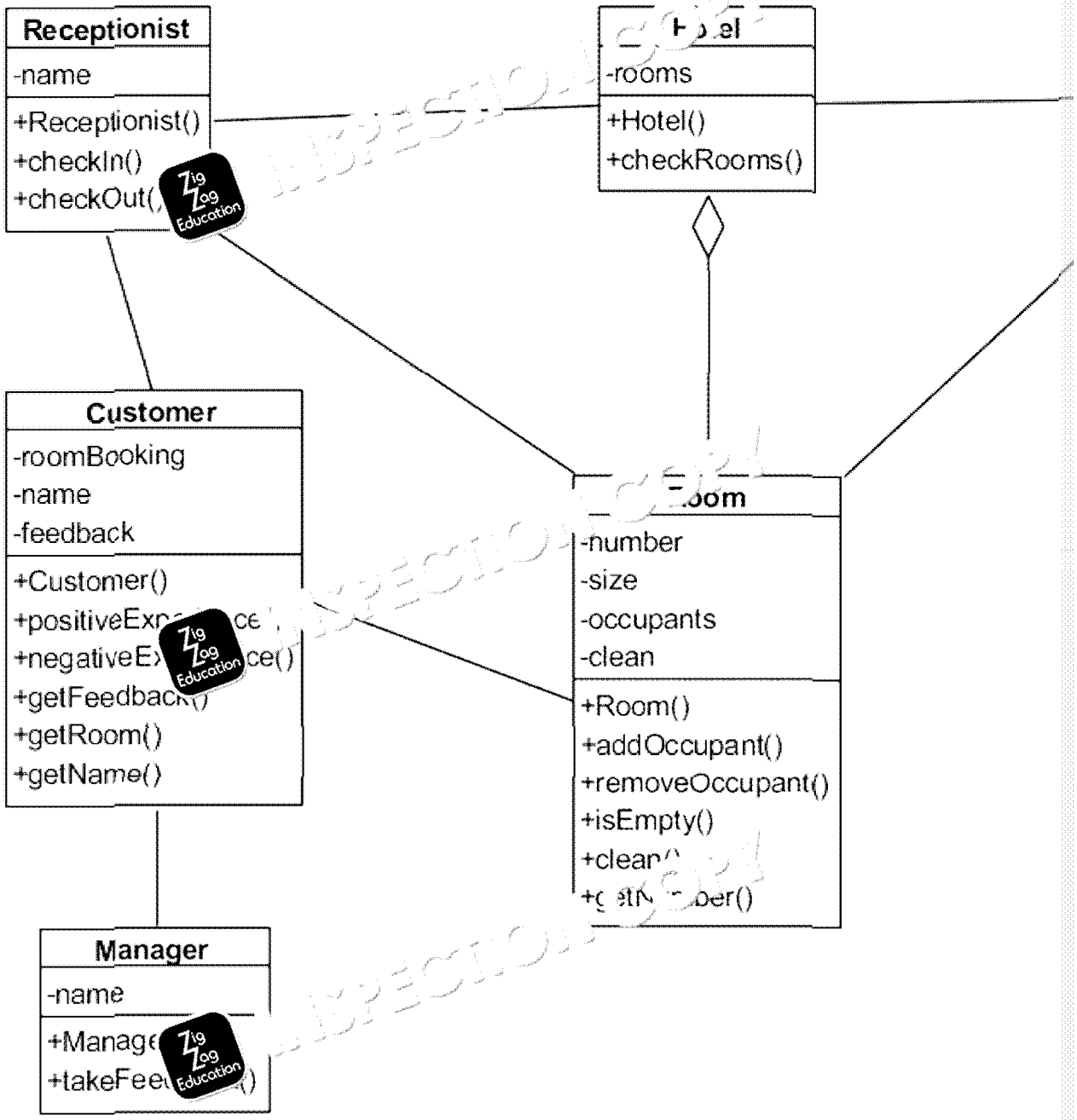
4. a) Class B inherits from Class A (Class [illegible]. the parent class, Class B is the chil
   relationship, i.e. 'inhe[illegible]nce'; 1 mark for getting classes the correct way ro
   b) Class A is a [illegible]e class formed with Class C component objects. **(1 ma**
   'c[illegible]tio[illegible]; 1 mark for getting classes the correct way round, i.e. 'C form
   c) Cl[illegible] an aggregated class formed with Class B component objects. **(1 m**
   'aggregation'; 1 mark for getting classes the correct way round, i.e. 'B form

UML Class Diagram Solutions

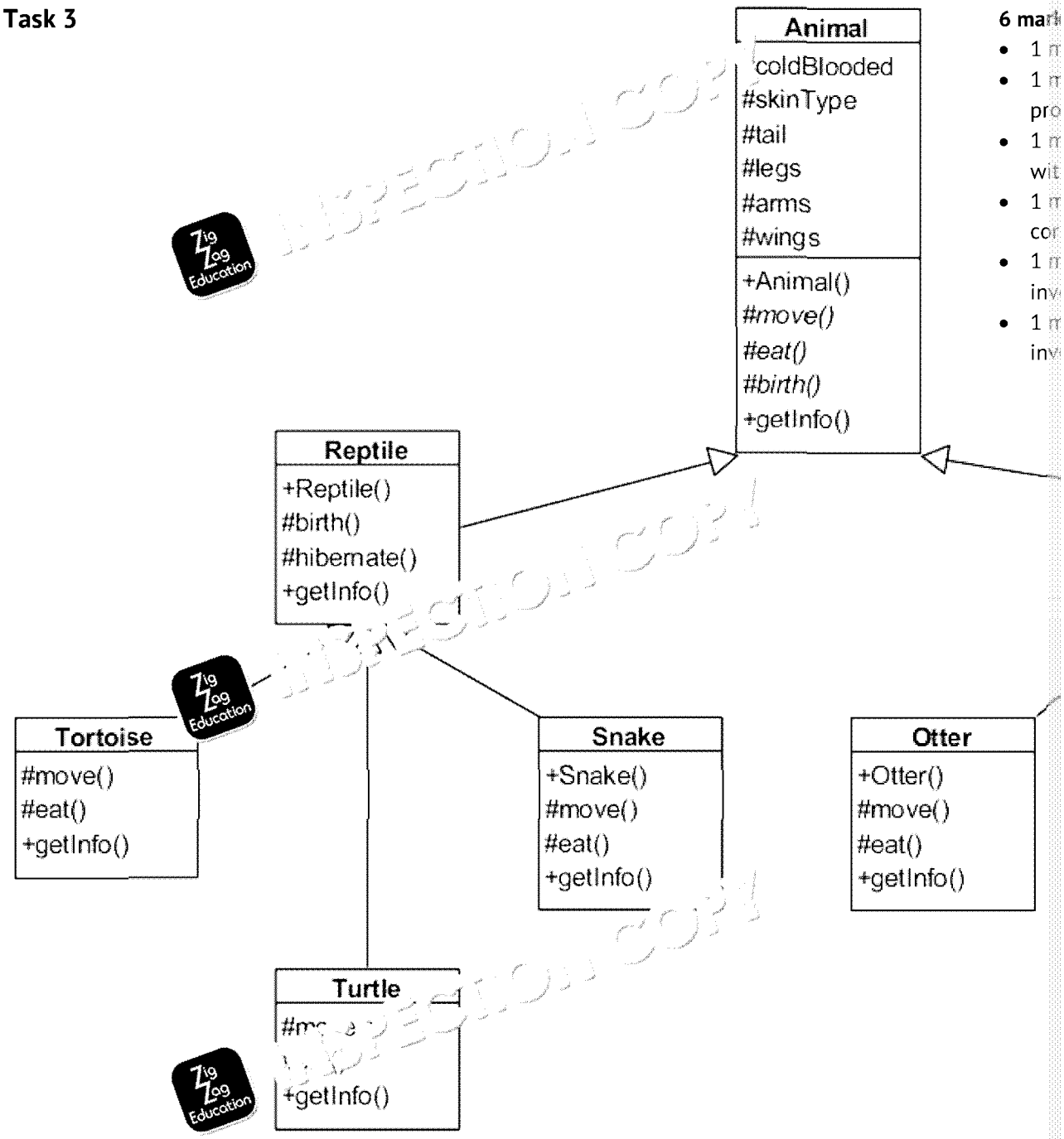
Task 1



Task 2



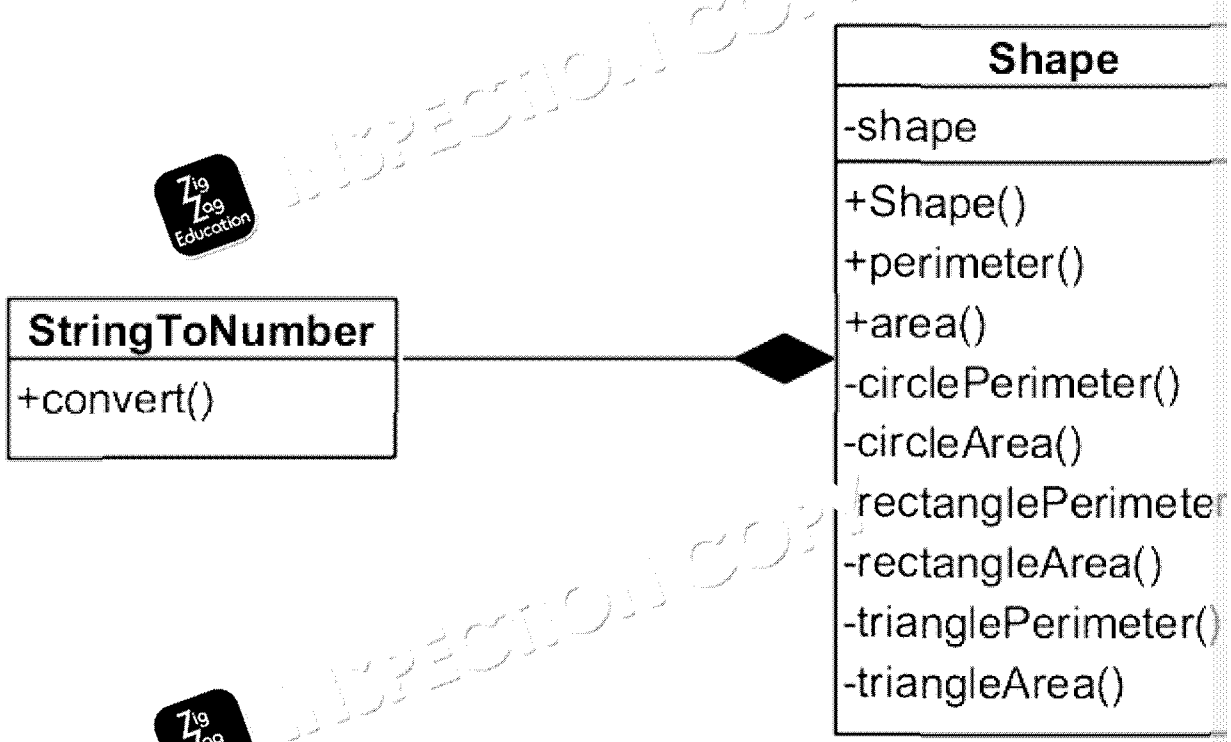
Task 3



6 mar

- 1 m
- 1 m pro
- 1 m wit
- 1 m cor
- 1 m inv
- 1 m inv

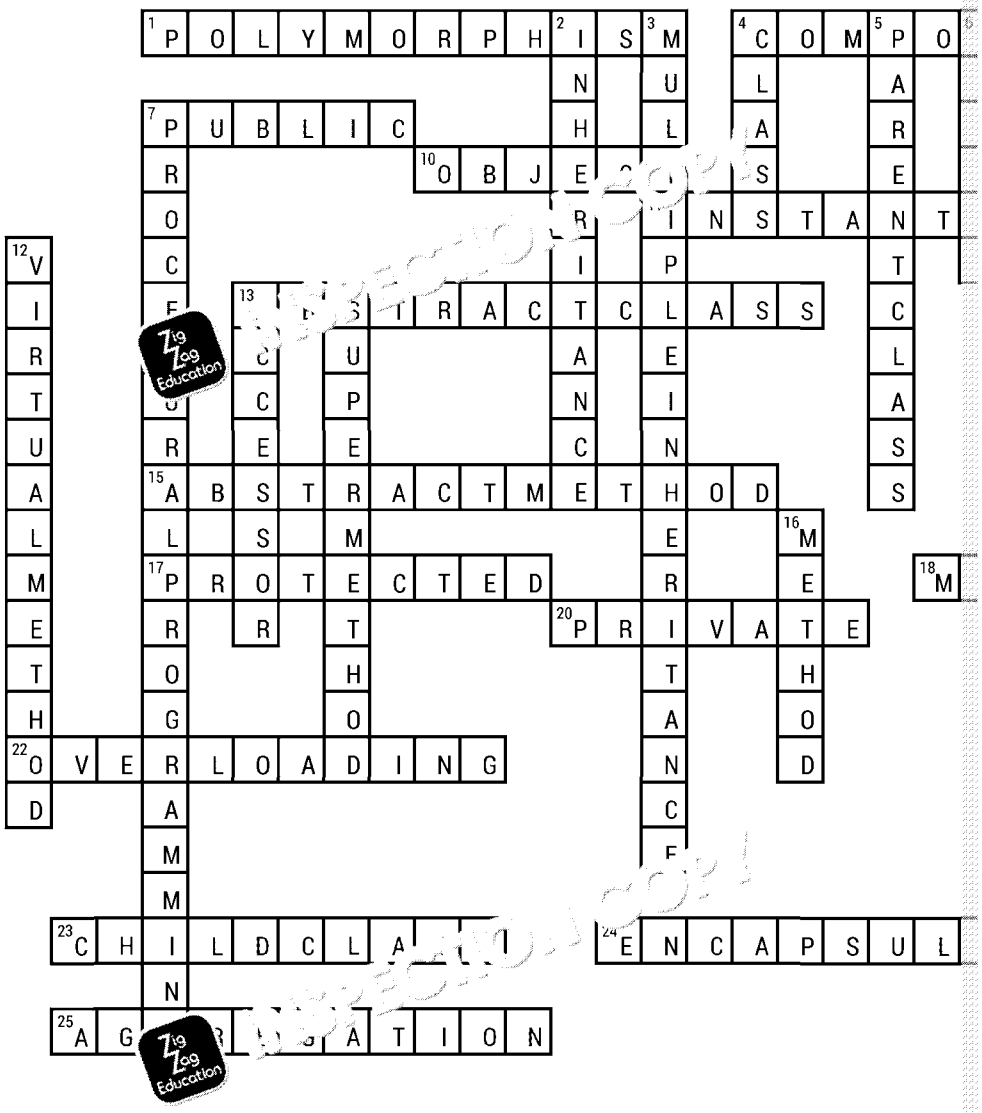
Task 4



4 marks:

- 1 Mark for showing all classes
- 1 Mark for showing all attribute
- 1 Mark for showing all attribute
- 1 Mark for showing the compos

Crossword (OOP Concepts)



Across	Down
1 Allowing different implementations of a method to use the same method name (12)	**2** When one type of o a different type of o
4 Forming one larger object from multiple smaller objects, where the smaller objects cannot exist separately to the larger object (11)	**3** When a child class (8,11)
7 A method or attribute of a particular class that can be called by any other class (6)	**4** A template defining object from which o
10 An instance of a data structure that has its own attribute values and associated methods (6)	**5** A class that is inher
11 The process of creating an object from a particular class (13)	**6** Class method that c class have been ins
13 A class that contains abstract methods and cannot be instantiated (8,5)	**7** A programming par series of steps that
15 A method whose name and parameters are defined but that doesn't have any implementation (8,6)	**8** A variable or const object (9)
17 A method or attribute of a particular class that can only be call within that class or its child classes (9)	**9** A method that creat
18 A public method that changes the value of a private attribute (7)	**12** Any method that ca
20 A method or attribute of a class that can only be called within that class (7)	**13** A public method tha attribute (8)
22 Creating multiple implementations of the same method that take different parameter types (11)	**14** When a child class a method (5,6)
23 A class that inherits from another class (5,5)	**16** A subroutine belon
24 Grouping together related data and subroutines into classes, and providing controlled access to that class's private attributes (13)	**19** Superseding the im in its child class (10
25 Forming one larger object from multiple smaller objects, where the smaller objects can exist separately to the larger object (11)	**21** A particular approa (8)

GLOSSARY

Abstract Class	A class that contains abstract methods and canno
Abstract Method	A method whose name and parameters are define implementation
Accessor	A public method that retu[illegible] data relating to a pri
Aggregation	Forming one la[illegible] object from multiple smaller o objects [illegible] separately to the larger object
Attribute	[illegible] variable or constant belonging to a particular cl
Child Clas	A class that inherits from another class
Class	A template defining the attributes and methods o objects can be created
Composition	Forming one larger object from multiple smaller o objects cannot exist separately to the larger objec
Constructor	A method that creates an object of a particular cla
Encapsulation	Grouping together related data and subroutines in controlled access to that class's private attributes
Inheritance	When one type of object or class adopts functiona object or class
Instantiation	The process of creating an object from a particula
Method	A subroutine belonging to a par[illegible]ular class or obj
Multiple Inheritance	When a child class inh[illegible]s from multiple parent c
Mutator	A pub[illegible] method that changes the value of a priva
Object	[illegible] instance of a data structure that has its own at associated methods
Overloading	Creating different implementations of the same m argument types
Overriding	Superseding the implementation of a parent class
Parent Class	A class that is inherited by another class
Polymorphism	Allowing different implementations of a method t name
Private	A method or attribute of a particular class that ca class
Procedural Programming	A programming paradigm that structures a progra are followed in sequence
Paradigm	A particular approach [illegible] designing and creating p
Protected	A method [illegible] attribute of a particular class that ca class [illegible] its child classes
Public	A method or attribute of a particular class that ca class
Static	Class method that can be called even if no objects instantiated
Super Method	When a child class calls its parent class' implemen
Virtual Method	Any method that can be overridden by a child clas