



Complete, fold and seal
(see note 2 below)

FORENAME(S): HENRY

12 Pages

1. Please give the information requested in the boxes below.
2. Enter your name in the space provided to the right, fold over and seal down (using the label provided) to hide your name.
3. The answers must be legibly written. The Examiners can either ignore illegible script or have it typed and the expense of typing charged to the candidate.
4. Write your answers in ink.
5. Begin each answer on a fresh page. Do all rough work in the answer book and cross it through.
6. No part of this book is to be torn out; it must be handed in complete.
7. If more books than one are used, fasten them together using the tags available from the invigilator(s).
8. **Every book must be handed in. In no circumstances may answer books, used or unused, be taken out of the examination venue.**

List the numbers of the questions you have attempted	Office Use
1	~~12 1/2~~
2	~~0~~
3	12 1/2
Total	41 1/2

(Gweler yr ochr arall am y fersiwn Cymraeg)

$$\frac{30\frac{1}{2}}{75}$$

41%

GT 1/6/18

Q1

a)

- i) An 'objects' attributes are used describe the 'objects' in the 'class' framework to give the individual instance of the 'object' in the 'class' a context.

Though they are initially set they can be altered in accordance to the 'properties' set for objects in this class.

'Attributes', when set are given a suitable protection level to define them. Typically they are set to private to ensure they are not accessible outside the classes boundary.

It is this ~~member~~ protection that ensures 'object' attributes are not globally accessible and a potential security risk.

- ii) 'Objects' are single instances that are placed in a 'class', a structure designed to group items that contain similar attributes.

A class contains four headings to provide a 'table' like view objects are placed within, as seen below for a range of vehicles:

Vehicle ←			
Make	Model	Colour	Age
"VW"	"Golf"	"Red"	3
"Audi"	"A3"	"Blue"	4
"VW"	"Polo"	"Green"	1
"Saab"	"95"	"orange"	2

← an attribute
the class
← an object

The attributes are the ~~extra~~ columns and describe the individual objects.

A class comprises, as well as attributes, constructors to handle the objects attributes, properties to

define the encapsulation or access modifiers for each attribute and then methods. Methods allow actions to be performed for each object, commonly used to alter data contained in an objects attribute(s). As shown below

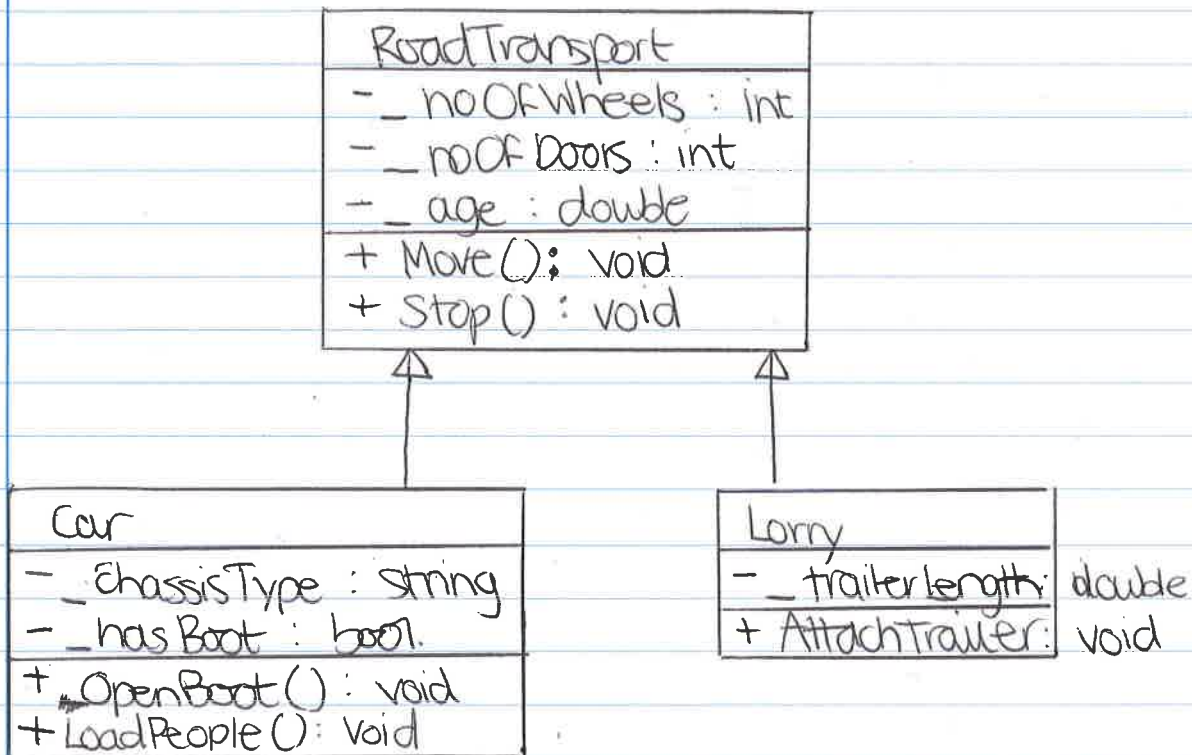
```
// Public class Car
{
    // Attributes
    private string _make;
    private string _model;
    private string _colour;
    private double _age;
    // Constructors
    public Car(string make, string model, string
                colour, string double age)
    {
        _make = make;
        _model = model;
        _colour = colour;
        _age = age;
    }
    // Properties
    public string Make
    {
        get { return _make; }
        set { _make = value; }
    }
    // Methods
    suitable methods would appear here such as
    Move(), Stop() or SoundHorn();
}
```

} for each attribute.

- iii) Inheritance describes the process of implementing a class hierarchy. This framework offers the most generic classes at the top and the most specific classes towards the bottom. To reflect this in code classes must explicitly say what they are

Inheriting and from where.

e.g. Cars and Lorries are both types of road transport, so they can inherit all of the highly generic road vehicle attributes and methods from this class and only add specific attributes to them in their own class. As seen below.



This is done in the class by adding the inherited class to the class name.

"public class Car : RoadTransport"

After the constructor arguments are added "base" is used to specify inherited attributes that have been defined in a different class.

Methods that will be overwritten by a derived class add the keyword "virtual" ~~to the base class~~ ~~to the derived class~~ the base class and "override" in the derived class.

Multiple inheritance is not accepted, so an interface can be used to provide multiple derivatives.

- b). The code is designed to create a house object. ~~When~~ During creation of the house it must be assigned a guard dog as one of its initialisation arguments.

The class 'Dog' contains a single method, designed to print "Bark" to the console when the method "BangDoor" is called on a house object. Another method on the house object allows the changing of guard dog by adding in the dog to the methods arguments on the Method House();

- c) There are several errors in the provided code;
- 1) In class 'Cat' the ~~private~~ constructor should not include 'void', and instead read "public Cat"
 - 2) The 'constructor' line "_name = name" should instead read "_name = theCatName" as this is how the variable is written in the 'constructor' blocks argument list.
 - 3)

- d) The 'main' file for code shown in appendix 1 should look like this:

This creates a house and with it a dog,

```
"House House1 = new House(Dog1);"
```

Then the method 'BangDoor' can be called on the

house ~~just~~ just created, this prints "Bark."

"House1.BangDoor();" ✓

This code can be used to ~~add there dogs to the~~ perform the private method. ChangeDog(); cannot be called from the main, hence the ~~House1.ChangeDog();~~ public method calling this ~~private~~ private method.

~~House1.ChangeDog(Dog2);~~ it is possible to change the guard dog, as shown by simply calling the method 'House();' on the ~~House1.ChangeDog(Dog2);~~ house object.

"House1.House(Dog2);"

```
public static void Main(String[] args)
{
    House House1 = new House(Dog1);
    House1 House1.BangDoor();
    House1.House(Dog2);
    House1.BangDoor();

    // Returns
    // "Bark"
    // "Bark"
}
```

2 1/2

18

Q3

- a) The single responsibility principle states the individual aspects of program responsibility should be broken down into their individual aspects. This class should contain minimal methods that allow the classes purpose to remain independant. E.g. An XML printer must handle both the formatting aspect and content of its output. These

are different aspects of program functionality and to prevent changes influencing other factors should be separated. Changes made to formatting ~~may~~ could create errors in the correct code managing the content to be printed.

This principle has the following advantages:

- 1) Increases class or member usability as it is managing a single aspect of functionality.
- 2) Helps promote good design principles, those used to identify application requirements.
- 3) Positive effect on maintainability as functionality is separated.

c). Blackbox testing is designed to measure program, or function outputs with no understanding of how the internal processes work or is implemented. It can find errors with functions, interfaces, database integration, and the runtime of internal components, but doesn't consider how it occurred. A typical example of this is boundary testing, a process of testing program output around its extremities (boundary). The accuracy of values output can then be evaluated to check if the desired values were received.

Whitebox testing worries less about the value outcomes as it concentrates on the internal processes used. Unit testing allows a tester to monitor a block, or selection of code to analyse/review its processes and composition.

White-box testing comes at testing from the developers perspective whilst black-box tests from the users point of view, this therefore requires far less system familiarity or experience. Both are critical and should be deployed to complement each other during post-developmental testing.