



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.**

FORENAME(S): PETER NEIL

1	Student Number	P079868
2	Title of Examination Paper	APPLICATION DESIGN + DEV
3	Module Code	MSX01025
4	Date and Time of Examination	14/05/18
5	Examination Venue	GYM
6	Seat Number	GOQO
7	Number of Answer Books handed in	1

List the numbers of the questions you have attempted	Office Use
1	19 1/2
2	17 1/2
3	14
Total	68%

(Gweler yr ochr arall am y fersiwn Cymraeg)

$$\frac{51}{75}$$

68% GT
1/6/18

Q1

A) ENCAPSULATION IS AN OBJECTS ABILITY TO HIDE DATA THAT ISN'T ~~NE~~ A NECESSITY TO THE USER. ENCAPSULATION IS THE MAIN REASON WE USE CLASSES; TO AVOID OTHER PARTS OF OUR CODE FROM CHANGING THE SAME VARIABLES, FOR EXAMPLE, IF THERE WERE ~~8~~ GLOBAL ATTRIBUTES AND ONE CAUSED AN ERROR, TRACKING THE TROUBLESOME ATTRIBUTE WOULD BE DIFFICULT.

IN OOP THE ATTRIBUTES EXIST IN THE SAME CLASS AS THEM AND THEY ARE RESTRICTED USING THE 'PRIVATE' KEYWORD TO PREVENT OTHER CODE IN SEPRATE CLASSES FROM HAVING ACCESS TO THE VARIABLES

EXAMPLE

```

CLASS POINT
{
    PRIVATE INT - X;
    PRIVATE INT - Y;
}

```

PRIVATE ACCESS SPECIFIER DEFING THE SCOPE + VISIBILITY OF ATTRIBUTES

// PROPERTIES

```

PUBLIC INT X { GET { RETURN - X } }
PUBLIC INT Y { GET { RETURN - Y } }
}

```

JUST USE GET TO MAKE PROPERTIES READ ONLY.

✓ DON'T PROVIDE GET IF YOU WANT WRITE ONLY PROPERTIES

* PROVIDE BOTH GET + SET IF YOU WANT READ + WRITE PROPERTIES

- a ii) CLASSES CAN BE BEST THOUGHT OF AS A TABLE WITHIN A DATABASE. THE PROGRAM REPRESENTS THE DATABASE, AND THE CLASS ~~IS~~ THE DB TABLE. THE COLUMNS OF THE TABLE ARE MADE UP OF THE ATTRIBUTES AND THE OBJECTS ARE THE DATA WHICH FILLS THE TABLE'S ROWS + COLUMNS.

THE CLASS SIMPLY DEFINES THE STRUCTURE FOR CREATING OBJECTS, WITH OBJECTS BEING VARIABLES OF THAT CLASS TYPE.

CLASS OBJECT VISUALISATION EXAMPLE

CAR	MANUFACTURER	MODEL	ENGINE	COLOR
SMALL	VW	UP!	800cc	BLACK
MED	VAUXHALL	ASTRA	1800cc	RED
LARGE	RENAULT	LAGUNA	2000cc	PINK

TABLE IS THE 'CAR' CLASS DEFINING THE DATA.

CAR OBJECTS, INSTANCES OF THE CAR CLASS

ATTRIBUTES DEFINING THE PROPERTIES OF DIFFERENT TYPES OF 'CAR' OBJECTS

- a.iii) INHERITANCE CAN BE THOUGHT OF AS DATABASE NORMALISATION WHERE WE TRY TO AVOID REPEATING FIELDS IN DIFFERENT TABLES, IN OOP WE WILL TRY TO AVOID REPEATING ~~CODE~~ THE SAME CODE IN DIFFERENT CLASSES.

INHERITANCE ALLOWS US TO CREATE PARENT OR BASE CLASSES WHICH CHILD CLASSES INHERIT FROM, WE DO THIS AS IT IS COMMON FOR OBJECTS TO SHARE SIMILAR PROPERTIES, FOR EXAMPLE, A SHIP, PLANE OR CAR ARE ALL TYPES OF VEHICLE AND THEREFORE ALL WILL SHARE SIMILAR PROPERTIES LIKE A MANUFACTURER, A MODEL, YEAR OF MANUFACTURE ETC. ALL THE COMMON ATTRIBUTES CAN BE PLACED IN A 'BASE' VEHICLE CLASS AND THEN INHERITED DOWN TO THE CHILD CLASSES USING THE FOLLOWING SYNTAX IN THE CLASS DEFINITION:

```
Public Class SHIP : VEHICLE
{
}
```

BY USING INHERITANCE WE CAN TIDY UP CODE, ORGANISE IT BETTER, MAKE IT MORE READABLE + RE-USE EXISTING CODE EFFICIENTLY.

```
Public Class SHIP VEHICLE
{
    - MANUFACTURER;
    - MODEL;
}
```

```
Public Class SHIP : VEHICLE
{
    Public SHIP A()
    {
        : BASE ("SHIP A")
    }
}
```

* INHERITING PROPERTIES + ATTRIBUTES FROM VEHICLE CLASS
↑ BASE.

- b) THE CODE PROVIDES TWO CLASSES, DOG + HOUSE, THE DOG CLASS CONSISTS OF A METHOD PRINTING OUT "BARK" TO THE CONSOLE, IN CLASS HOUSE AN ATTRIBUTE OF TYPE 'DOG' IS CREATED AS A GUARD DOG THE ~~PROPERTY OF THE~~ CONSTRUCTOR ~~THEN IS MADE UP~~ THEN ~~SEES~~ CALLS THE SET VALUE PROPERTY COPIED IN THE SET CHANGEDOG() FUNCTION, WE ARE COPYING DOG INTO ATTRIBUTE - GUARDDOG. A FUNCTION BANGDOOR() IS THEN ~~ETA~~ CALLED WHICH CALLS OUR INSTANCE OF DOG - GUARDDOG TO "BARK" IN THE CONSOLE.

- c) ① CONSTRUCTOR "CAT" SHOULD HAVE NO RETURN TYPE SPECIFIED, NO "VOID" SHOULD BE PRESENT. ✓
- ② CONSTRUCTOR "CAT" IS PASSING IN VARIABLE "THECATNAME" BUT COPYING VARIABLE "NAME" INTO THE ATTRIBUTE "NAME". ✓
- ③ IN FUNCTION "GREETCAT()" THE ATTRIBUTES PASSED THE PLACEHOLDER SHOULD BE "_CAT.NAME" WITH NO UNDERSCORE. *Get Name()*
- ④ IN CONSTRUCTOR "OWNER" IN CLASS "OWNER" THE TYPE OF PARAMETER PASSED IN SHOULD BE OF TYPE "CAT" NOT TYPE "OWNER". ✓
- 3 1/2

d) PUBLIC STATIC VOID MAIN (STRING[] ARGS)
{

// CREATE DOG

DOG GUARDDOG = NEW DOG()

// CREATE HOUSE

HOUSE NEWHOUSE = NEW HOUSE (GUARDDOG)

// MAKE DOG IN HOUSE BARK.

NEWHOUSE (GUARDDOG). BARK();

} GUARDDOG.BARK();

QUESTION 2

a) IN ~~THE~~ APPENDIX 3 THE FIRST RELATIONSHIP IS ASSOCIATION, DEMONSTRATED FROM THE



ARROW. IN CLASS HOUSE IT CAN BE SEEN THAT AN ASSOCIATION TO CLASS OWNER + PART ARE PRESENT IN A UNI-LATERAL ~~Direction~~ FASHION IT ALSO DENOTES THAT THE HOUSE HAS ONE OWNER AND ~~IS CAPABLE~~ ONE HOUSE CAN HAVE MANY PARTS $\bullet \rightarrow 1$ AND $0 \dots \infty$ RESPECTIVELY. OWNER AND PART ARE DERIVED CLASSES OF PARENT HOUSE AND INHERIT PROPERTIES OWNER TO OWNER AND PART TO PART.

IT CAN ALSO BE SEEN THAT PART HAS TWO CHILD CLASSES WINDOW AND DOOR AND INHERITANCE IS SHOWN ON THE DIAGRAM WITH THE FOLLOWING ARROW.



THE THIRD RELATIONSHIP IS SHOWN WITH THE FOLLOWING ARROW



- b) IN C# INHERITANCE FROM MORE THAN ONE CLASS IS PROHIBITED BUT IT IS POSSIBLE TO IMPLEMENT MULTIPLE INTERFACES.

INTERFACES ALLOW PROGRAMMERS TO DEFINE BEHAVIOUR OF AN OBJECT WITHOUT DECIDING HOW THE OBJECT IS IMPLEMENTED.

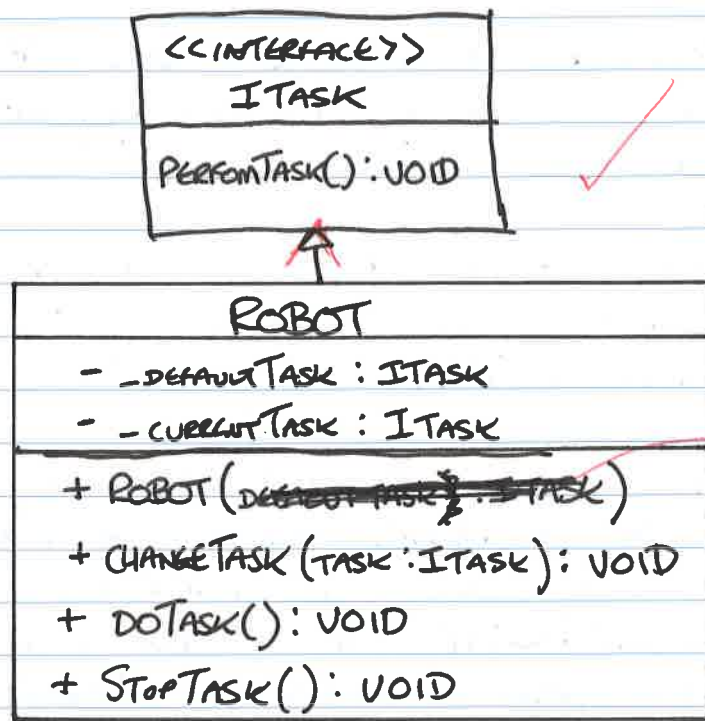
WHEN A CLASS IMPLEMENTS AN INTERFACE, IT MUST USE METHODS OR PROPERTIES WHICH THE INTERFACE NEEDS BUT THE CLASS CAN IMPLEMENT THESE METHODS IN DIFFERENT WAYS, THE INTERFACE DEFINES THE CAPABILITIES OF A CLASS.

IN THE APPENDIX CODE THIS CAN BE SEEN WHERE CLASS ROBOT HAS A FUNCTION ^{DO TASK()} WHICH IS IMPLEMENTING THE FUNCTION PerformTask() FROM THE INTERFACE 'ITASK'.

THIS TYPE OF ABSTRACTION IS IMPORTANT BECAUSE IT ALLOWS NEW FEATURES OR IN THE APPENDIX 4 CODE CASE, NEW OBJECTS APART FROM ROBOTS, EG HUMANS, DOGS, CIGARS ETC TO BE EASILY ADDED TO THE PROGRAM WITHOUT CHANGING ANY CODE TO IMPLEMENT THESE OBJECTS TO PERFORM TASK();

THE PATTERN IN USE IN THE APPENDIX CODE IS THAT OF THE FACTORY PATTERN, THIS IS RECOGNISABLE AS ~~THE~~ A LARGE NUMBER OF OBJECTS COULD BE CREATED WITHOUT ~~MODIFY~~ MODIFYING ANY CODE IN THE INTERFACE, ALL NEW CLASSES + INSTANCES COULD SHARE THE SAME CODE.

c)



QUESTION 3

- a) THE SINGLE RESPONSIBILITY PRINCIPLE OR SRP WAS FIRST INTRODUCED BY ROBERT C MARTIN AND IT MAKES UP THE 'S' IN THE 'SOLID' DESIGN PRINCIPLES.

HE ARGUES THAT EVERY MODULE OR CLASS SHOULD HAVE ONE RESPONSIBILITY. A RESPONSIBILITY OF A SINGLE PART OF THE FUNCTIONALITY PROVIDED BY THE PROGRAM, AND THE RESPONSIBILITY SHOULD BE ENCAPSULATED BY THE CLASS.

THE BENEFITS OF THIS INCLUDE EASIER TO MAINTAIN CODE AS IT IS BETTER ORGANISED AND LESS CLUTTERED. SECONDLY, THE CLASS IS EASIER TO UNDERSTAND AS LESS CODE IS IMPLEMENTED. THIRDLY, THE CLASS BECOMES MORE REUSABLE AS IT ONLY HAS ONE RESPONSIBILITY IT CAN BE REUSED WITHOUT MODIFICATION.

- b) APPENDIX FIVE SHOWS A DIAGRAM OF A USER CASE PATH TESTING FLOW CHART, THIS TYPE OF CHART EXPLORES THE POSSIBLE PATHS A USER MAY TAKE WHEN OR WISH TO SEE WHEN USING AN APPLICATION.

ALL OF THE REQUIREMENTS CAN BE SEEN. EACH REQUIREMENT ~~WO~~ REPRESENTS CODE WHICH WILL PERFORM THE TASK THE USER IS TRYING TO ACHIEVE.

MAKE ORDER TASK WOULD INHERIT FROM ONE CLASS REPRESENTING MOBILE PHONE ORDERS OR WEB FORM ORDERS, ~~FROM~~

THE USER THEN MAY WISH TO DISPLAY ORDERS, FOLLOWING ON WOULD BE THE USER DISPLAYING ORDERS WHICH WOULD INCLUDE AN INTERFACE OF PAYMENT DETAILS TO TAKE ~~THE~~ PAYMENT FROM THE CUSTOMER FOR THE ORDER.

3



- c) WHITE BOX TESTING IS USED FOR VERIFICATION AND BLACK BOX TESTING FOR VALIDATION, IT IS SAID THAT FOR COMPLETE EXAMINATION BOTH SETS OF TESTS SHOULD BE PERFORMED.

BLACK BOX TESTING IS A METHOD OF SOFTWARE TESTING WHERE FOCUS IS MADE ON THE INPUTS + OUTPUTS AND CONFIRMING RESULTS AGAINST ANTICIPATED OUTPUTS, IT IGNORES THE INTERNAL MECHANISMS OF A PROGRAM. BLACK BOX IS MORE END-USER EXPERIENCE FOCUSED FORM OF TESTING.

WHITE BOX TESTING TESTS BEYOND THE USER INTERFACE, IT USES KNOWLEDGE OF THE INTERNAL WORKINGS OF A PROGRAM TO SELECT DIFFERENT TESTS. IT REQUIRES DETAILED KNOWLEDGE OF THE SOFTWARE CODE TO SCRUTINISE THE CUT-POINTS FROM THE PROGRAM, WHICH MAKES THE TESTS USELESS UNLESS TESTER IS AWARE OF WHAT PROGRAM IS INTENDED TO DO, HOWEVER, A SIDE EFFECT OF THIS CAN BE TESTERS TAILORING TESTS TO MATCH THE CODES OPERATION, RATHER THAN WHAT IT DOES MEANING TESTS CAN BECOME BIASED.

BLACK AND WHITE BOX TESTS COVER A RANGE OF TESTING TECHNIQUES, FOR BLACKBOX; UNIT BOUNDARY ARE TWO EXAMPLES AND FOR WHITEBOX, PATH TESTING IS AN EXAMPLE.