

Examination Answer Books

12 Pages

Complete, fold and seal
(see note 2 below)

SURNAME:

Archer

FORENAME(S):

Ben

Instructions to Candidates

- Please give the information requested in the boxes below.
- Enter your name in the space provided to the right, fold over and seal down (using the label provided) to hide your name.
- 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.
- Write your answers in ink.
- Begin each answer on a fresh page. Do all rough work in the answer book and cross it through.
- No part of this book is to be torn out; it must be handed in complete.
- If more books than one are used, fasten them together using the tags available from the invigilator(s).
- Every book must be handed in. In no circumstances may answer books, used or unused, be taken out of the examination venue.**

1	Student Number	131842
2	Title of Examination Paper	Application Design and Development
3	Module Code	CSA CS001M
4	Date and Time of Examination	14/05/2018 9:00 AM
5	Examination Venue	Pembrokeshire College
6	Seat Number	1
7	Number of Answer Books handed in	

List the numbers of the questions you have attempted	Office Use
1	22
2	19
3	18
Office Use	59



Q1

a. Attributes represent the state of an object, they are usually prefixed with an underscore, and are usually manipulated after instantiation using object methods. Encapsulation of attribute values is used to reduce the amount of ways attributes can be manipulated in code. Encapsulation provides a way of 'data-hiding' and can often be seen in practice through the use of getters and setters. *

(3)

b. A class is the logical expression of code and an object is the physical existence of the data the code uses. An object can be seen as an instance of a class. Objects can be 'instantiated' in code through the use of the 'new' keyword. *

(3)

c. Inheritance of classes creates parent-child relationships within classes, also known as Superclasses and Subclasses. Inheritance is often used when several classes share common attributes. Specialisation of classes occurs as you move down the class hierarchy, adding more specific attributes and behaviors. Generalisation occurs as you move up the class hierarchy. *

(3)

*
class {
 int

 private attribute {get; set; }
}

*2

```
Class() c = new Class();
Class c = new Class();
```

*3

```
public ClassA() :-
public ClassA : ClassB
```

↑ colon shows inheritance

b) A class Dog exists that contains the method Bark() that writes to console. A class House exists that contains a Dog object called -guardDog as an attribute. The House class contains a constructor that accepts a Dog object as a parameter. Inside the constructor, the dog parameter is passed to a private method ChangeDog that sets the -guardDog attribute to the dog object passed into it. The House class also contains a method called BarkDoor() which calls the Bark() method inside the Dog class inside the -guardDog object.

c.

- Incorrect ~~variable~~ parameter name called in Cat constructor. Should be : _name = theCatName.
- Incorrect object inside Owner constructor, should be Cat. ✓
- Incorrect method call inside Console.WriteLine inside greetCat method. Should be -cat.GetName()
- Constructor Cat should not have return type. Remove void. ✓

d)

```
Dog theDog = new Dog();
```

```
House theHouse = new House(theDog);
```

```
theHouse.BangDoor();
```

(4)

Q2

a)

House and part have a 0...^{*} relationship (a House can contain many parts). House and owner have a relationship wherein a house can only have 1 owner. Owner and car have transient relationship. Part has an aggregated relationship with window and Door. House ^{Interface} is the main class that would be instantiated first, you would then either instantiate an owner to assign to the house object or instantiate a part object to assign to house. You ~~created~~ ^{then}

(3)

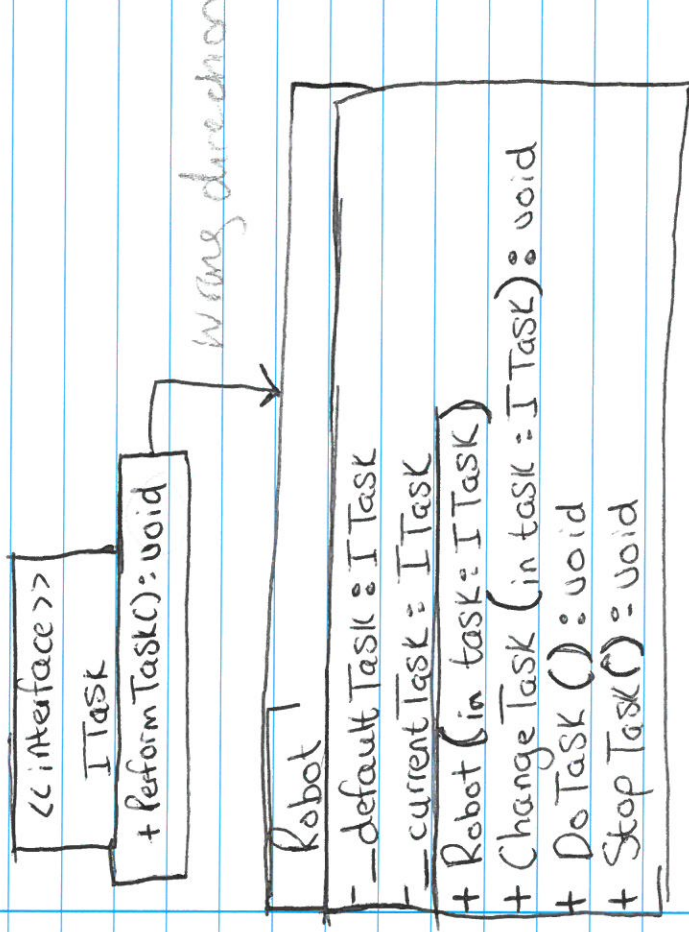
b)

The code in appendix 4 appears to be using the Strategy design pattern as it is clear that the code has been written in a way that identifies the aspects of the application that vary, and has been abstracted around that. In the example we can see that the interface ITask has been created. An interface acts like a contract to classes, ensuring that certain methods are carried out. Interfaces are often used when the type of data is different, but the functionality required is identical. In this case the performTask() method. There is usually no implementation of the functionality of the method inside the interface method as it is the classes responsibility.

to implement it. Moreover interfaces are very important in C# code as they allow for C# classes to perform more than one role. Using the Strategy design pattern allows for the scope of the application to be increased along with its ability to be maintained and tested.

8

C.



8

Q3.

a) Introduced in 1979, the Single Responsibility Principle stated that a class should only fulfill one responsibility, it should do it well, and that each new responsibility is an axis of change. Some of the benefits of the SRP is that it avoids the problem of monolithic class design, improves readability, reduces code complexity, reduces coupling, increased cohesion within classes as each class has a specific function, and improves testability. (b)

b) The application proposed in appendix 5 is an ordering system that allows users to place an order through either a mobile app order form, or a web order form. Users are also able to display orders, along with confirming orders through a payment system.

The state order diagram node has an aggregated relationship with both the mobile app order node and the web order node, ^{instance} suggesting that both these nodes can ^{have} ~~outline~~ the make order node within the application.

The display orders node is a singular node that has no relationships with other nodes. (2)

~~Suggesting~~ it will be a contained process.
 The confirm order node contains a

transient ~~includes~~ relationship with the Take Payment node suggesting the confirm ^{other} order node will supply ^{something} to the take ^{way} payment node, also suggesting that the ~~Take Payment~~ node will use the Take confirm order.

Payment node internally.

c) Black Box testing, also known as data-driven or input/output testing is a testing method whose goal is to be completely unconcerned about the internal behaviour and structure of the program in which its testing. Black Box testing concentrates on finding circumstances in which a program does not behave according to its Spec. Black Box testing is made up of multiple disciplines including Functional Testing, in which a specific feature is tested, Non-Functional Testing, in which the operation of the system rather than features are tested.

Moreover, some of the best strategies adopted by black box testing include Testing To Pass and Error Forcing (Testing to fail) as well as Equivalence Partitioning Strategies such as Exhaustive Input Testing. Boundary Conditions are also tested to simulate situations at the edge of the planned operational limits of the software.

'White Box testing on the other hand does permit testers to examine the internal structure of programs as this method of testing is Logic Driven. White Box testing derives test data from an examination of the program's logic. Some of the testing strategies involved with white box testing include Exhaustive Path Testing wherein all possible paths of control flows are tested through the program. However, the most popular White Box testing strategy is Unit Tests. Unit tests are small pieces of code that are used to test some aspect of a program programmatically.

Unit Tests use Mocks, Fakes, Dummies, and Stubs to Simulate certain conditions within Programs. Unit tests are used to test Sub-Boundary Conditions and formulae.

It Should be argued that using a Combination of Black and White box testing within Software development is a best practice as when used in conjunction with each other, the two testing methods provide a large scope of testing that provides testing data not only from a Program view, but from a Users perspective. It also could be argued that using the two methods decreases the chance for bugs to appear later on in an applications development cycle, decreasing the cost of fixing bugs.

(16)

Arholwr yn unig

Rhify Cwestiwn

Question No.

Examiner Only