



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Examination Answer Books

12 Pages

Instructions to Candidates

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.**

Complete, fold and seal

(see note 2 below)

SURNAME: PUGH

FORENAME(S): JAMES

1	Student Number	145295
2	Title of Examination Paper	APPLICATION, DESIGN & DEVELOPMENT
3	Module Code	CSAC 5001M
4	Date and Time of Examination	14 th MAY 2018 9.00am - 13.00pm
5	Examination Venue	PEMBROKESHIRE COLLEGE Room: FI.1
6	Seat Number	

7	Number of Answer Books handed in	1
---	----------------------------------	---

List the numbers of the questions you have attempted	Office Use
1a.	8
1b	2
1c	2
1d	4
2a	2
2b	4
2c	8
3a	6
3b	2
3c	6
Office Use	44

1.

(i) Attributes are basically variables that belong to the class they are created in. Their main purpose is to help us provide information about each class in our program. Although it is possible to declare attributes as public it is considered best practice to declare them as private. Therefore in order to access or change the values of class attributes in our main program we use what are known as properties. Unlike attributes, properties are set to public. ^{methods} Encapsulation is the idea that we should bundle data and the methods that work on that data into one unit. One example of encapsulation is the creation of classes within our program. One example of an attribute can be seen if we imagine that we have a class called User and inside that class we would have attributes that store information about each user such as their name, gender and age. These attributes would look like this in C# code.

```
private string _name;
```

```
private string _gender;
```

```
private int _age;
```

*forcing access to attributes via methods
protects attributes from
accidental change* (2)

(ii) A class basically acts as a blueprint from which instances of the class known as objects can be created. While it is possible to create multiple objects of the same class type, it is not possible to create multiple versions of the same class. Trying to create multiple versions of one class will simply create entirely unique and separate classes. Another aspect of the relationship between classes and objects is that when a class is created it isn't stored within memory. However, each time an object of a class type is created, it is stored in its own location in memory. The final key aspect of the class object relationship is how they differ in terms of their type of existence. While objects have a physical existence in our applications, classes have a purely logical existence. One example highlighting the class/object relationship would be if we had a class called car we would be able to make multiple objects of the data type car. However we could not make another class called car as this would result in an error due to a class called car already existing.

(iii) This concept of object orientated programming centres around one class gaining access to the attributes and methods of another class known as its parent or base-class. It is a rule in most OOP languages that a class can only inherit from one base/parent class. A core benefit of inheritance is that it makes programs more efficient as a class can gain all the functionality present in another class without having to rewrite any of the code that created the inherited functionality. Inheritance allows developers to take advantage of method overriding. This allows a child class to have a method with the same name as one in its parent class but change how the method is implemented to suit the unique needs of the child class. Inheritance assists in creation of what are known as class libraries. These are collections of prewritten classes that developers can use while developing applications. The main drawback of inheritance is that the base class and the class or classes that inherit from it become tightly coupled. This means if a change is made to the functionality of the base class such as a method being removed or altered, this will cause errors anywhere said method is used by objects of the child class's type which in the case of large programs could greatly increase the time needed for effective debugging. One example of inheritance would be if we had a class called dog a class called cat and a class called animal, we could place functionality such as breathe and eat

(3)

(3)

methods in the animal class and have the dog and cat classes inherit the animal class's functionality.

1 (B) First the code creates a class called Dog which will allow instances of that class to be created in our main program. Next, the public method Bark simply results in the word "Bark" being displayed to the console window whenever it is called.

(2)

(C)

- One of the mistakes is that in the Cat constructor that is passed the string theCatName, the constructor is trying to set the value of the private string variable `_name` to the value of `name` which does not exist. for this mistake to be corrected, `_name` should be set to equal the value of the `theCatName` string being passed to the constructor.
- The owner class is attempting to have an attribute of the Cat type. However because there is not an inheritance relationship between the owner and cat classes, it is not possible for the owner class to contain attributes of the Cat type without an error occurring. To fix this, we must create an inheritance relationship between them like so, `class Owner : Cat`.
- In the `console.WriteLine` statement at the end of the owner class, the cat attribute is trying to use dot notation to display the value of its name attribute. This is incorrect because the `_name` attribute which is originally from the Cat class has its access modifier set to private. This prevents it from being accessed via dot notation. Instead there should be a public property within the Cat class called `name` that as well as containing a get method, also has a set method that sets the value of the `_name` attribute equal to its value. If this was done then that property would be accessible via dot notation similarly to the attempt previously discussed.

(2)

(D)

namespace dog_house

{

class Program

{

static void Main(string[] args)

{

Dog dog1= new Dog(); - Creates new dog object

House house1 = new House(dog1); - Creates new house object and passes the previously made dog to the constructor.

house1.BangDoor(); -Calls the BangDoor method for the house1 object which should display the word Bark to the console window

Console.ReadKey(); - Keeps the program running once all previous lines have run untill the user presses a key on their keyboard.

}

}

}

2.

(A) The Window and Door classes are child classes to their parent which in this case is the Part class. Because of this, one of the Part class's intended uses is to give the Window and Door classes access to methods they both share such as the Close and Open methods originally found in the Part class.

(B)

Abstraction is important because it is the process of hiding unnecessary details about a program from the user. This enables the developer to implement more complex logic on top of the provided abstraction without needing to be concerned about the user understanding or even thinking about all the hidden complexity. Abstraction has been used in this example through the use of the strategy design pattern. This pattern is an example of a behaviour design pattern and involves defining a family of algorithms, encapsulating each one and making them interchangeable. The pattern lets the algorithm vary independently from the clients that use it.

(C)

Answer booklet

3.

(A)

The Single responsibility principle is an example of an OOP design principle. The principle states that each class in a piece of software should only be concerned with one specific task. In essence there should only be one reason to change a class at any point during development. If a class is found to be responsible for multiple separate tasks, the principle states that a new class should be created for each of these additional tasks and the code for each task should be moved to the appropriate class. The main advantages of the SRP are the following.

- It becomes easier to extend the functionality of classes due to irrelevant code/functionality not being present.
- Changes to the code are more isolated and less likely to impact other areas of the system in unforeseen ways.
- Individual classes will be smaller thus making the general structure and flow of the program easier to understand.

(6)

(B)

The intended use of this application is to allow a user to purchase products via either a website or mobile phone app. In addition, the application has the purpose of confirming, displaying and taking payment for a customer's chosen orders. One of the key components of this diagram is the user which is an example of what known as an actor. This is because the user both acts upon the other components in the system and is acted upon by the system. Regarding the relationship between the components of this diagram, the majority are tasks which occur in a sequential order as seen in how early tasks such as making an order are positioned higher on the diagram than later tasks such as confirming an order. It is because of this sequential relationship between the tasks in this diagram, that this style of diagram is often referred to as a sequence diagram. Between the confirm order and take payment tasks on this diagram, is what is known as an association link, this is because there exists a strong connection between the process of confirming an order and taking the required payment.

(2)

Dependency

(C)

Black box and white box testing are both highly commonly used testing methods for effective application development. Where these two approaches differ is in the areas of an application to which they focus. For example, white box testing is concerned with testing the internal code of the software and is referred to by the name white box due to how the internal code is both visible and well known by testers. Some alternate names for white box testing include clear and glass box testing respectively. There are 3 core aspects to white box testing, known as statement coverage, branch coverage and path coverage.

Statement coverage involves testing all the statements in a software program and is measured as a percentage. So if we have a program that consists of 20 statements and we have tested 10, then we can say that the level of statement coverage is 50%. This helps testers determine whether more tests need to be conducted.

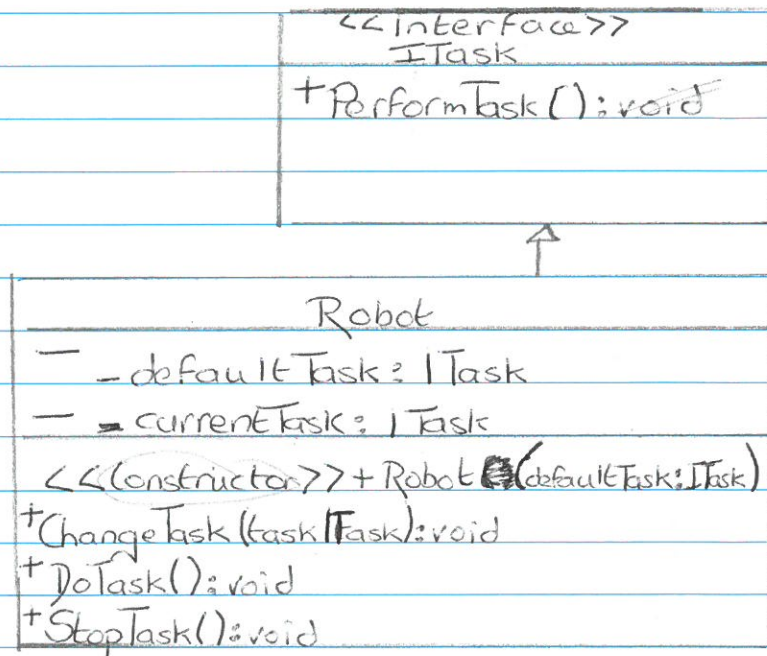
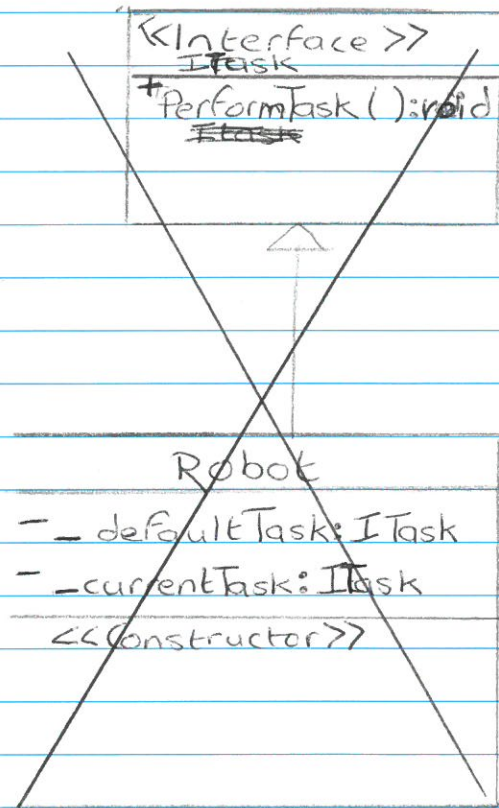
Branch coverage involves the testing of all the branches in a program. Branches are the offshoots that arise from a decision point such as an if statement, switch statement or while loop. Similarly to statement coverage, the mathematical representation of the extent of branch coverage is a percentage of the number of tested branches against the number of total branches in a program.

Path coverage is the process of developing tests that execute ideally every possible path through the code. However in the case of loops it is virtually impossible to archive full path coverage as there is virtually an infinite number of paths that can arise.

In contrast, black box testing is not concerned with the tester knowing or having access to the internal code of the software being tested. This method focuses on the testing of a piece of software's user interface (UI) in order to ensure that it is both intuitive and contains all the features specified in the software requirements.

Both these testing methods should be used when undertaking a software project. This is because, while both may have drawbacks such as white box requiring direct access to the code and blackbox possibly being insufficient due to a reduced knowledge of the application by testers, they both carry advantages that compliment eachother. For example while white box testing can help identify the presence of redundant code to be removed, thus improving application performance, black box testing due to its focus on user experience can help identify gaps in functionality that need need to be coded in.

R (1)



(8)

Question No.

Examiner Only

Rhify Cwestiwn

Arholwyr yn unig