|  |  |
| --- | --- |
| **Learner Name** | Husnain Ahmed |
| **Assessor Name** | Chris Livesey |
| **Qualification Title** | BTEC - L3 IT Extended Diploma in Computing |
| **Unit Title** | Unit 16 – Object Oriented Programming |
| **Assignment No./Title** | 14.2 – Design and Develop a Computer Game to Meet Client Requirements  16.2 – Design and Develop Object Oriented Solutions to Identified Problems  17.2 – Design and Develop a Mobile App |
| **Learning Aim(s)** | A: Understand the principles of object-oriented programming |
|  |  |
| **Issue Date** | 22/02/21 |
| **Planned Submission Date** | 19/03/21 |
| **Re-submission Date (if approved)** | 02/04/21 |

|  |  |
| --- | --- |
| **Feedback** | Provided online in the form of rubric, comments, and general feedback. |
| **Reflection** | Once you receive feedback, you should reflect on your performance using the reflection document. This should be recorded on your Pro Portal.  --> ILP --> 3. My Learner Reflections |
| **SMART Actions/Targets** | You should regularly set SMART Actions/Targets on your Pro Portal.  --> ILP --> 4. My SMART Actions |

|  |  |  |
| --- | --- | --- |
| **Evidence of proof-reading and improvements to quality of communication using Microsoft Editor.** | | |
| Print screen of Microsoft Editor before improvements | Print screen of Microsoft Editor after improvements (1) | Print screen of Microsoft Editor after improvements (2)  Re-sub if required. |
|  |  |  |

The Principles of OOP

Contents

[The Features of Object-Oriented Programming 2](#_Toc63757522)

[The Importance and Effectiveness of Object-Oriented Programming 2](#_Toc63757523)

# The Features of Object-Oriented Programming

**What is OOP?**

Object oriented programming (OOP) is a programming paradigm based on objects (anything that contains data, usually a variable) in the form of procedures (subroutines), procedures within an object can modify the value of the objects themselves. It is coded by making different objects interact with each other. (taken from my 22.1 work)

**Distinguishing features of OOP**

The distinguishing features of OOP encapsulation, inheritance and polymorphism as these features are not common in many other programming paradigms.

**4 pillars of OOP**

The 4 pillars of OOP are abstraction, encapsulation, inheritance and polymorphism.

**Abstraction** is a method to reduce the complexity of data to make it more readable and to hide some redundant details from the user who is viewing them as they do not need to see them.

An example of abstraction is making coffee or tea with a vending machine, this is because the user only needs to select their drink and press a button to have it made without knowing of any of the background details about how their drink is made and what processes it goes through.

**Inheritance** is a method to derive a subclass called a ‘child’ class from a ‘super’ class / ‘parent’ class. This subclass will share a set of attributes and methods with the class that it was derived from. There is a strict hierarchy with parent classes being at the top and child classes beneath them, child classes can derive their own children that are even lower on the hierarchy.

An example of inheritance is when a variable set in a parent class is called in a child class and used there. As it is a public variable it can made use of.

**Polymorphism** is a method where different classes can be used with the same interface, each of the classes can have their own implementation of the interface. This means that a single method could have multiple actions.

An example of polymorphism is how a person can have multiple roles. Such as being both a child, brother, father, son, worker and much more.

**Encapsulation** is a method where a bunch of data is grouped together with the methods that are used to operate on them or where you restrict access to some of the components of an object (similar to abstraction in this regard).

An example of encapsulation is the initialisation of private variables within a class as they can only be used within that class. Protected variables are also encapsulated to only be accessed by the class that they are in and in subclasses of that class.

**The features of OOP**

**Objects**: this is an instance of a class or the materialisation of it. An object contains many methods and data on things such as colour and names. Calling an object executes one of its many functions.

**Classes**: these are like the blueprints of an object or the concept behind it. If you code a person object you will name the class ‘person’. An object is coded inside of a class.

**Methods**: these are actions or subroutines within an object that can be executed later on when they are called.

**Modularity**: this is the process of decomposing a problem in the program into a smaller set of modules to reduce the overall complexity of the program. These smaller modules can be sorted out and remedied separately which would be easier than doing it all together.

**Overloading**: this is when 2 or more methods have the same name but different parameters, if called the class that they are in will need to be specified.

**Overriding**: this is when 2 methods have both the same name and parameters, this changes the value of the method. When a function in a subclass changes the value of a variable in a superclass.

**The benefits of OOP with examples**

Reusability is one of the benefits of OOP, with the classes we can reuse the code an unlimited amount of time by calling the methods and functions in them. Previous objects that have been inherited can be reused in different ways.

Reliability is another benefit of OOP; it is reliable because it is remarkably effective in solving problems and because its structure does not allow for redundant data. Common functions can be written in a parent class and then inherited into multiple child classes instead of being rewritten repeatedly.

Flexibility is another benefit of OOP; this is done through the principle of polymorphism as different objects can react differently in different circumstances such as a young man acting as a student at school and as a customer in a shop.

Multiplatform is another benefit of OOP, as software coded through object-oriented programming languages are eligible for multiple different operating systems and platforms such as ‘google chrome’.

**The use of mathematical and logical processes in OOP**

**Algorithms**: Algorithms tell the program what to do, describe how to solve problems in order and how to implement features. Structure lays out the algorithm in source code.

**Application of predefined functions**: Predefined functions can be inherited into subclasses and reused repeatedly for different purposes.

**Graphical user interface (GUI) resolutions and element position**: GUI provides a visual interface for the user to interact with to navigate the program. There can be different GUIs for different classes to allow for better suitability for the classes with different intents behind them.

**Creation of 2D shapes**: A class can set the number of sides that a shape has as well as its dimensions which will be set as the value of variable (object). A method will use these values to create a shape, there may also be the option to add values for the area and perimeter of the shape to have it be more accurate.

**Use of programming logic and conditional statements**: This allows for the program to be more readable and easily understood. By following programming logic there is less chance for errors and bugs to appear in your code as they can be easily spotted and remedied. The use of conditional statements lets you use conditions in your code to allow for different decisions to be made, not restricting you to a single path of the program and letting you branch out.

**Use of Boolean algebra in conditional statements**: This allows for the implementation of Booleans (and, or, nor, xor) in your conditional statements to be more specific with your conditions. It also allows you to make branching paths in your program that allow for many different decisions to be made by the user.

**The factors that affect performance, safety and security**

**Platform**: The performance of a program will change depending on the platform they run on since different platforms may have different hardware requirements to run on or may not be optimised properly which would affect how the program runs.

An example of this is a computer game being ported to a mobile platform such as android and not being optimised as well as having a large file size that is not capable of running on mobile, as the hardware requirements are for PC they will not run on weaker mobile phones without optimisation.

The safety / security of your device will not change on different platforms as there is not much change between them unless they come with a built-in firewall or antivirus software but for the platforms that do not come with this they can be downloaded personally.

**Garbage collection**: This will help improve performance as there will be less clutter and useless data taking up space on your hard drive, this will allow the freed-up space to be allocated to improving the performance of the program.

An example of this is the application ‘CCleaner’ that removes useless clutter and unwanted files from your device.

The safety / security of your device will be increased as there will be less files with data that are not being actively used and taking up memory which will make it harder for third parties to access them without you being notified.

**Interpreters**: This will help improve performance as program will be opened faster since they will already be compiled, this will allow you to save time.

An example of this is ‘Java virtual machine’ that allows you to run programs that have been compiled to Java bytecode.

The safety / security of your device will be increased as the source code is more difficult to affect and infect with any form of malware such as viruses. The security is also boosted by the fact that hackers cannot view the source code of the software and learn its inner workings to hack into it easily.

# The Importance and Effectiveness of Object-Oriented Programming

**OOP Quiz**

This is a small program that asks the user some questions about object-oriented programming, it asks the user 5 questions which they answer, it then outputs their score as well as their average score over the total amount of games that they have played. It also asks whether the user would like to retry the quiz or end the program.

**Use of principles of OOP**

**Abstraction** has been used in the player class of this program where your average score is calculated by adding up your total score from previous games and dividing it by the number of games you have played. As the scores and number of games played are private variables, they make use of data abstraction.

**Encapsulation** has been used is also used in the player class as the private variables of score and number of times the game has been played cannot be used within other classes, this prevents them from being called in other classes and used for other purposes.

**Inheritance** has not been used as all the classes have been separate without any subclasses to inherit data.

**Polymorphism** has been used as the program makes use of multiple classes to create objects, this is mainly used to create the quiz’s questions. It allows for reusability of code and increased efficiency as it helps save time.

**Objects** have been used throughout the program to code functions such as ‘NumOfPlays’ which keeps track of the amount of time a certain person has played the game for and outputs it at the end of the game along with their score and average score over all of their games played.

**Classes** have been used in this program, there are 2 main classes named ‘player’ and ‘questions’. these classes are both loaded into the main program to be used as the basic framework of the game, they consist of empty variables that will be filled out in the main program.

**Methods** have been used in this program to make it function, one of these methods that have been used are the ‘CreateQuestions’ method that contains all 5 of the questions that the user will answer when they play the quiz game. This method uses the question class as its outline as and its contents are filled in later on in the main program.

**Modularity** has been used to break up the initialisation of questions into two separate areas, one is in the question class where they are initialised but not given any string values and the other is in the main program where the actual question is entered through the method ‘CreateQuestions’.

**Overloading** has not been used in this program.

**Overriding** has not been used in this program.

**Use of mathematics and logical processes in OOP**

**Algorithms** have been used in this program to dictate the direction it will go in, all of the objects and functions in this program are algorithms as they tell the program what to do by giving it instructions that have been laid out in the code.

**Predefined functions** have been used by making use of the prebuilt features that come with the IDE ‘Visual Studio’. These functions were used to code the entire program with commands like ‘Console.WriteLine()’ and ‘int score = 0;’.

**GUI’s** have not been used, instead this program makes use of the command line interface (CLI) and does not show any images, instead being solely made up of text.

**Programming logic and conditional statements** have been used in the ‘IF statements’ that validate the user’s answers to the questions and check if they are correct. After checking if the answer is correct it will add to the score counter if it is and do nothing if it isn't. At the end of the program it is used again to create a branch in the code that will allow the user to choose if they would like to replay the quiz or end the program.

**Evaluation**

Overall, the OOP quiz is very effective at achieving its purpose and has shown quality in terms of the following aspects:

It shows **robustness** by having multiple choices to the questions that are already written and laid out for the user to pick from by typing in a letter from ‘a’ to ‘d’, this shows robustness as it helps prevent spelling errors that could arise if the user had to manually type out the entire answer which could be multiple words.

It is not robust however, in the way that the program would simply continue if erroneous input is entered such as an integer or a letter that is not ‘a’, ‘b’, ‘c’ or ‘d’ even though it is not a valid input, the program should instead repeat that question until a valid input is entered.

It shows **maintainability** through its structure that allows it to be easily readable due to its neatness and allows it to be changed without the risk of many errors popping up. Due to following coding conventions such as indentations and naming conventions like PascalCase are used to name the variables, classes and objects which would allow for them to easily be called later if they are needed to be executed.

It does not show maintainability however, in the way that the questions are coded. This is because if the programmer wishes to change the questions and answers they will need to manually do so by rewriting large portions of the program.

It shows **efficiency** by displaying itself on the console and giving short and concise instructions to the user to follow to answer the questions. It is also very efficient due to being an OOP program which means that the code has a high level of reusability which can help save time.

However, it will be very inefficient to change the questions or answers as it requires you to manually rewrite large portions of the program.

It shows **portability** due to its small size of 396 KB which means that it can be run on devices with small amounts of storage space. It can also be run on devices with lower specifications as it is a text based CLI which does not have very high visual requirements like a GUI would have due to the lack of images needing to be rendered.

However, it is not very portable as of right now since it needs to be compiled through ‘Visual Studio’ to be run. This can be solved by publishing the program as an executable file.

It shows a high degree of **ease of use** as there are instructions on how to navigate the program on the screen that are displayed to the user at the start of the program’s runtime, this ensures that they will know exactly how to use it.

It could be further improved upon by adding more colour to the sentences to differentiate the questions and answers better, this will ensure that no one gets confused between the two. It could also be improved by adding validation for the questions to ensure that the user input is correct and does not have any errors with it.

In conclusion, the program is very good and has achieved its purpose to a high degree. However, it could be improved with the aforementioned changes to make the program more robust as well as adding a wider range of questions to make the quiz longer and more engaging. It does its intended purpose and is very easy to use.

**OOP Escape Game**

The aim of the ‘escape game’ is to escape the rooms that you are locked in, the program asks for the user's name and then explains the rules and goals of the game to them. It then proceeds to explain each room that you are in as you enter them and show a series of prompts with decisions for you to make.

The first room that you will navigate through will be the storeroom, after leaving this you will enter the kitchen and living room which both require a key to exit. After leaving the final room you will complete the game.

**Use of principles of OOP and mathematics**

**Abstraction** has been used to clear the console after each section of the game or ‘room’ have been completed, this removes any unnecessary information from previous parts of the game from distracting the user and confusing them.

**Encapsulation** has been used to hide information such as the password after it has been entered into the text field, this is a way to help increase the security of the program and make it so that no one other than the user can access their account.

**Inheritance** has been used in this program in the room class, the attributes of this class have been inherited into its subclasses ‘Kitchen’ and ‘LivingRoom’. The inherited attributes help save time and make the program more efficient as you can reuse the code instead of rewriting it all.

**Polymorphism** has been used for the object called ‘Key’ to change its properties at different times, it does this by changing its colour from red to blue to open different locks. This also makes the program more efficient and saves time by allowing this piece of code to be reused more than once.

**Objects** have been used to code functions and methods in the program, an example of this in the escape game is the ‘Key’ item. This item is created in a subclass of the ‘item’ super class and its purpose is to open locked doors in the game, a red key will open a red door and a blue key will open a blue door.

**Classes** have been used in this program, one of the classes that have been used is the ‘room’ class. It is a parent class and contains the attributes that the other rooms will use once they have derived them from this class. An example of a child class in this program that does this is the ‘Kitchen’ class, this inherits many attributes from the ‘room’ class that it reuses within itself to create objects.

**Methods** have been used in this program to make it function, an example of a method in the escape game program is the ‘PlayGame’ method that runs after the player inputs their details and introduces the player to the game with an introduction and a set of instructions, after this runs the actual game begins.

**Modularity** has been used to create many methods by initialising them in one area then calling them somewhere else. An example of this happening is the ‘password’ variable being initialised in the player class and then being checked later on.

**Overloading** has been used in the room class to change the value of the variables from strings to attributes.

**Overriding** has been used has been used to change the value of the method ‘getRoomDetails()’ in the room class, it updates the value of the light status and the door status as they have been changed from their previous state.

**Use of mathematics and logical processes in OOP**

**Algorithms** have been used in this program to dictate the direction it will go in, all of the objects and functions in this program are algorithms as they tell the program what to do by giving it instructions that have been laid out in the code.

Depending on the users inputs and the choices that they make in the game the program will go down different paths or branches, this order has all been planned out in the algorithms.

**Predefined functions** have been used by making use of the prebuilt features that come with the IDE ‘Visual Studio’. These functions were used to code the entire program.

An example of this is the output to the user of the description of each room. This is done with the predefined ‘Console.WriteLine();’ command that comes with ‘Visual Studio’. Predefined functions such as this can be recommended to the user when they are typing them to save time and be more efficient since they are built into the program.

**GUI’s** have not been used, instead this program makes use of the command line interface (CLI) and does not show any images, instead being solely made up of text. This program does however make use of different colours for the text to distinguish it from others and to make it look more pleasing.

**Programming logic and conditional statements** have been used to allow for the user to make decisions that can go down branching paths in the code. There is also a conditional statement used to loop the current section of the program (room) that the player is on if they choose the wrong answer, and this will continue to be looped until the correct answer is chosen and they have moved on to the next room of finished the game. This is done through the use of do while statements that repeat until a specific input is entered, that input aligns with the number of the correct answer that will allow you to proceed to the next level of the game.

**Evaluation**

Overall, the Escape game is very effective at achieving its purpose and has shown quality in terms of the following aspects:

It shows **robustness** as it repeats questions if the incorrect answer is inputted and will allow you another chance to enter in the right answer. The fact that it does not allow you to go back to previous rooms that you have already completed after finishing them shows robustness as well as it ensures that the user follow the programs intended structure and path.

However, the robustness of the program could be improved upon by adding a validation after every input that checks if the user input is of the correct datatype and format. This is needed because there is an error in the program that causes it to crash if an incorrectly formatted answer is entered.

It shows **maintainability** as new rooms ant item can easily be created; this is because of the principle of inheritance. The new rooms and items will be created of subclasses of the ‘item’ and ‘room’ classes which will allow for them to inherit a lot of attributes that will be useful in creating them and will save a lot of time and effort.

However, the program would not be very maintainable in the case that you decide to create a new parent class for an entirely different purpose such as obstacles, this is because it will not follow a predetermined structure and will need to be coded manually without any derived attributes to help it.

It shows **efficiency** as a lot of code can be reused due to it being inherited from parent classes, an example of this happening is in the room classes where a lot of attributes for the status of the room doors and lights are derived from the parent class of ‘room’. The fact that the console is cleared after each room is passed also shows efficiency as any data that is no longer necessary for the user, this gives a simplistic look that looks very efficient due to it not being confusing and distracting the user with superfluous amounts of information.

However, the program could be more efficient if the colour of the ‘Key’ variable was declared in the parent class named ‘item’. This would be more efficient as it would help save a lot of time declaring the colour of the key over and over again as is done in the original code.

It shows **portability** due to its small size of 132 KB which means that it can be run on devices with small amounts of storage space. It can also be run on devices with lower specifications as it is a text based CLI which does not have very high visual requirements like a GUI would have due to the lack of images needing to be rendered. The only thing that would need time to render would be the coloured texts and even that extra rendering time is negligible due to how low it is.

However, it is not very portable as of right now since it needs to be compiled through ‘Visual Studio’ to be run. This can be solved by publishing the program as an executable file.

It shows a high degree of **ease of use** as there are instructions on how to navigate the program on the screen that are displayed to the user at the start of the program’s runtime, this ensures that they will know exactly how to use it. They are also given prompts of what to do at certain times such as ‘press enter to continue’ which make navigating the program much easier. Due to the differently coloured words there is a clear distinction between the instructions of each room and the game its self, such as the options that you have to choose from.

It could be further improved upon by creating a GUI that would show a visual display of each room according to its description such as showing the doors and a table with an item on it in the rooms that they belong in. This would increase the user’s experience when playing the game as it would be easier to visualise it when there is a picture given rather than going off a description. But this will increase the file size by a large amount which may make it harder to run, the use of a GUI would also make it harder to run as it Is not only rendering a series of text in the CLI but also images which is much more graphically intensive on a device.

In conclusion, the escape game makes good use of the principle of object-oriented programming and has a high level of efficiency and maintainability. It could, however, be improved if it was more robust and didn’t crash due to incorrectly formatted inputs such as integers being inputted where a string was expected.