

Object Oriented Programming Report

Author: Tomos Leggett · **Student Code:** 1901852 · **Module Code:** ACCA5032

Object Oriented Programming Report

Link to project

The Importance of Object Oriented Programming

The main concepts

Encapsulation

Abstraction

Inheritance

Polymorphism

References

Link to project

Mega link to both solutions (Snakes and Ladders + RPG board game) along with documentation of concepts:

<https://mega.nz/folder/56ghyChJ#-2uKmxCdfHo9PXNgWolHbw>

The Importance of Object Oriented Programming

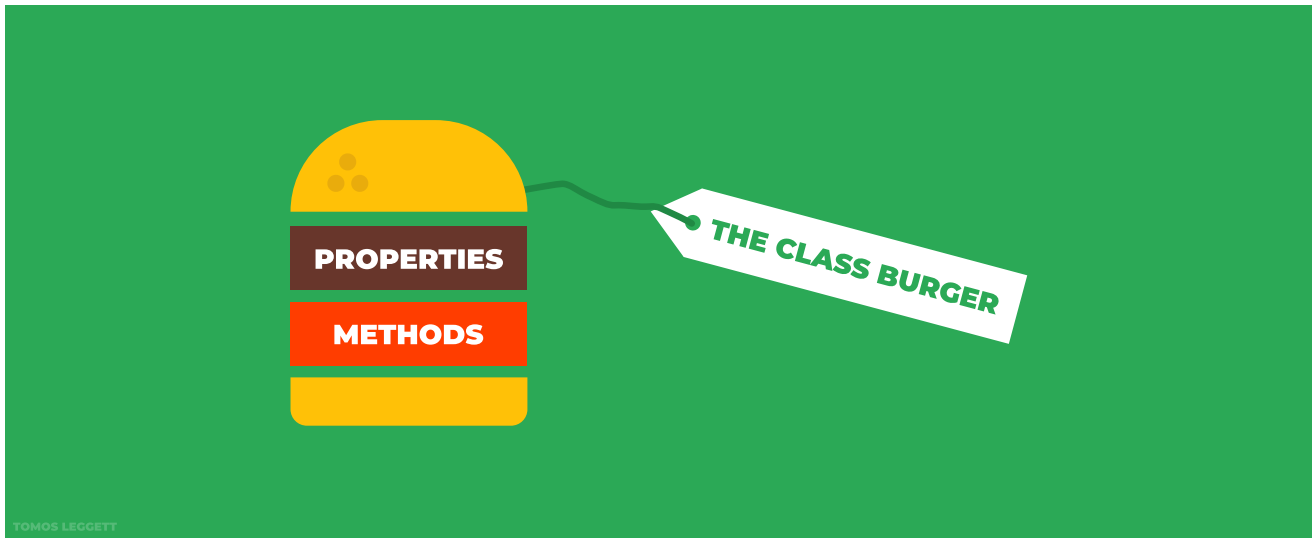
Object Oriented Programming brings various advantages to traditional programming methods.

One major advantage of OOP is ease of troubleshooting your program. Object Oriented programming gives the added advantage of being able to pin point the exact class and/or method that is causing issues and trouble throughout your code as all the data and functions in OOP are encapsulated into one class. This means developers are able to pinpoint the exact class that is causing the issue and solve the problem without having to re-write large chunks of code as the main program and other classes often adapt dynamically to the changes saving a lot of corporate time and programmer energy. [1]

Object Oriented Programming also enables effective problem solving techniques for the developers, as OOP languages allow developers to break down programs into 'bite sized problems', as most OOP based languages allow developers to approach their programs in a more pragmatic approach that's easier to digest and understand once you're fully aware of its capabilities. [2]

The main concepts

Encapsulation



Encapsulation is a fundamental core component of Object Oriented Programming. It is the concept of encapsulating together various properties and methods respectfully into one component, often referred to as a class. [3] Classes are often ‘template like’ functions that allow developers to create objects. Objects contain both variables for storing data about the object or around the object, and methods (or sometimes referred to as functions) which are sequences of code that add functionality to the variables.

A class in C# would be a perfect example of encapsulation, as by definition, it encapsulates variables and the methods that work together with these variables into a single item. [4]

Encapsulation is also sometimes referred to the act of hiding the values of a variable, such as the state of an object, by making them inaccessible without a “get” method. This is a security benefit as it prevents unauthorised access to these current variable states as they are often encapsulated in separate classes that aren’t as easy to access, especially if they are set with attributes such as ‘private’ or ‘read-only’ etc. [5]

The author uses encapsulation in their C# snakes and ladders game in various situations, for example, a class for players is used to store both properties about the players and the methods that can be used to adjust that data:

The properties

```
1 private readonly string _playerName;
2     private int _playerPosition;
3     private readonly char _playerSymbol;
4     private int _playerCurrentScore = 0;
5     private int diceValue;
```

The methods

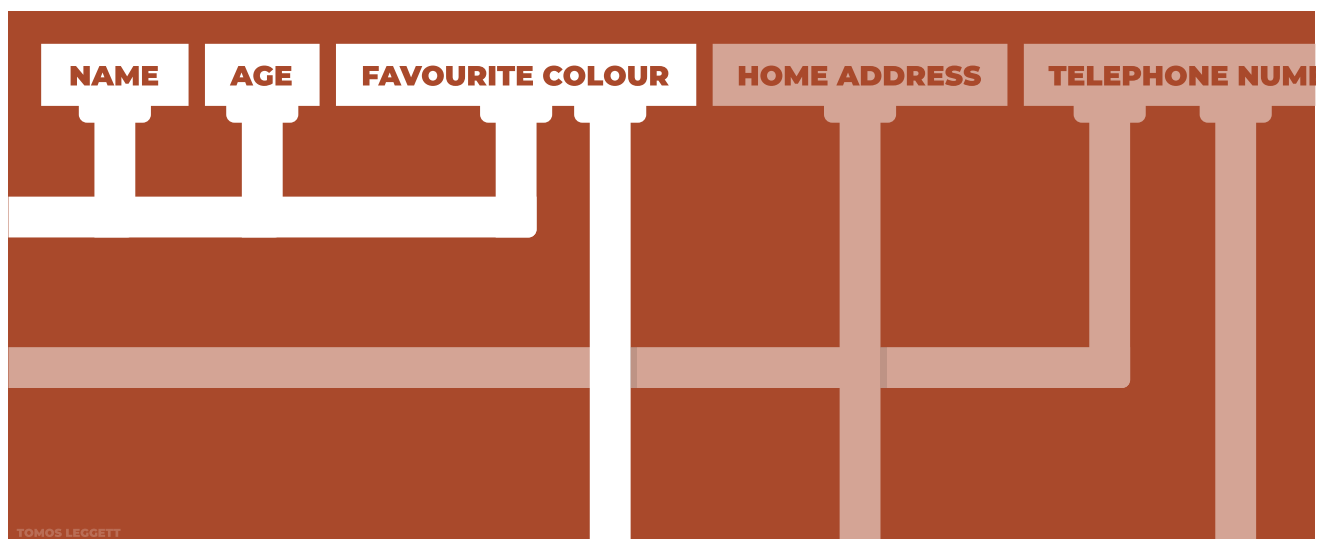
```

1 public void ThrowDice()
2     {
3         // Random die value between 1 and 6
4         diceValue = dice.RollDice();
5         dice.RenderDice();
6         if ((_playerPosition + diceValue) < 100)
7         {
8             _playerPosition = _playerPosition + diceValue;
9         }
10    }

```

This ThrowDice() method is used to adjust the value of the player position property based on the value of the rolled dice, and is encapsulated within this class along with the properties themselves. Both the properties and the methods are not on the main program class themselves, and are instead encapsulated as one object that can be called with one line.

Abstraction



As we've discussed above in the definition of encapsulation, hiding certain information from the user provides security benefits, but abstraction in OOP is another fundamental element of OOP. Designed to only show information that is necessary, and therefore, hiding information that is **not** necessary (only showing information that is relevant to the subject at hand) reduces the complexity of the program and the efforts of the programmer. [6]

A good example of abstraction would be real world examples, for example, the coffee machine example.

Quoting Thorben Janssen in their stackify article "OOP Concept for Beginners: What is Abstraction?"

“You need to know how to use your coffee machine to make coffee. You need to provide water and coffee beans, switch it on and select the kind of coffee you want to get. The thing you don’t need to know is how the coffee machine is working internally to brew a fresh cup of delicious coffee. You don’t need to know the ideal temperature of the water or the amount of ground coffee you need to use.”

The coffee machine example is a very good example in OOP, as it encapsulates the core concept of abstraction with a real world example. The article mentions the details the coffee machine keeps from the end user, as those particular details don’t necessarily concern the end user, and would otherwise overcomplicate things needlessly. [7]

The author uses abstraction as a concept in their own C# Snakes and Ladders game, for example, in their Players class, they’ve created a series of private and read only properties for the players:

```
1 internal class Player
2 {
3     private readonly string _playerName;
4     private int _playerPosition;
5     private readonly char _playerSymbol;
6     private int _playerCurrentScore = 0;
7     private int diceValue;
8
9     private readonly Dice dice = new Dice();
```

This information is kept private and in some cases read only, a constructor is then used to modify these values and said values cannot be accessed on the main game class.

```
1 public int GetPlayerPosition => _playerPosition;
2
3     public int GetY()
4     {
5         return _playerPosition / 10;
6     }
7
8     public int GetX()
9     {
10         // Odd rows are traversed right to left
11         return GetY() % 2 == 1 ? ((10 - 1) - _playerPosition % 10)
12 : (_playerPosition % 10);
13     }
14
15     public int SetPlayerScore
16     {
17         set { _playerCurrentScore = value; }
```

```
17     }
18
19     public int SetPlayerPosition
20     {
21         set { _playerPosition = value; }
22     }
```

Inheritance



Inheritance is a powerful feature in OOP that allows developers to create entirely new classes with very little modification of an already existing class in that namespace. These new classes are referred to as derived or child classes and are usually inherited by parent classes in which they are children of. (Also referred as a base class). [8]

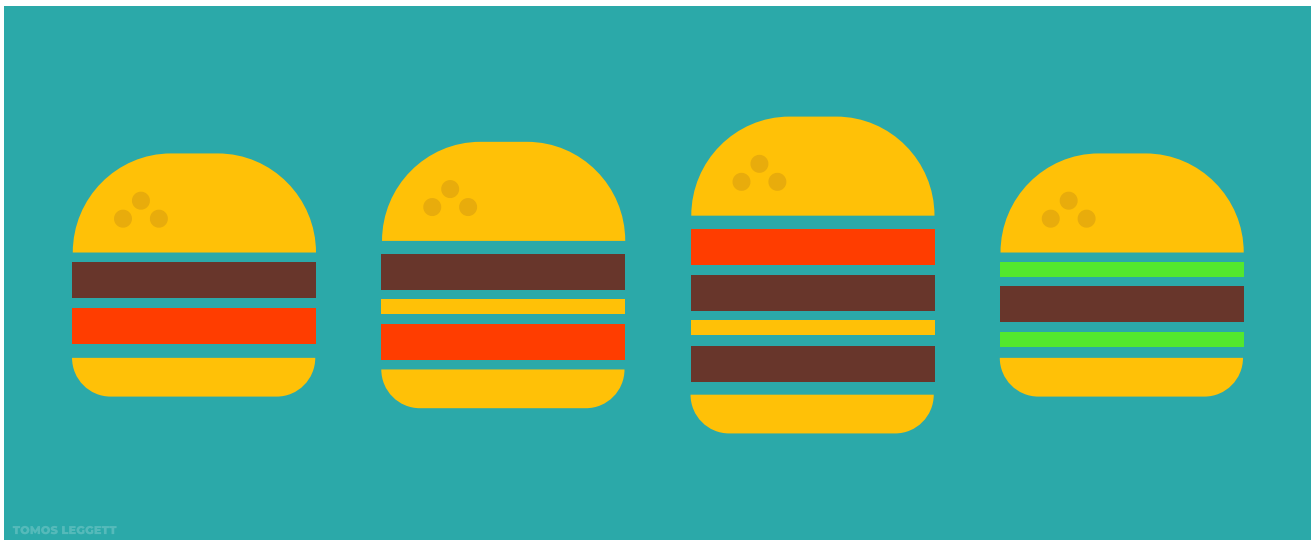
In the author's C# program, they've exemplified the use of inheritance by using classes in amongst other classes, for example:

```
1 public int RollDice() // Generate a random number to represent the dice
2     {
3         Random rnd = new Random();
4         _diceValue = rnd.Next(0, 6) + 1;
5         return _diceValue;
6     }
```

In this snippet, the author has developed a function named `RollDice()` for their dice class. The random number itself uses another class named `Random()` and creates an entirely new `Random` object called `rnd`.

The `rnd` object has a method with 2 input values that is used to generate a dice value.

Polymorphism



Polymorphism in Object Oriented Programming is a program's ability to parse an object's functionality with different inputs and methods rather than just one singular input and output. It's an extremely powerful tool that allows developers to be flexible with objects in their program and enables objects to parse information in different ways and output information in different formats despite being the exact same object. [9]

A good example of this would be the difference between `Console.Beep()` and `Console.Beep(300, 100)`. The first command immediately executes a method from the Console class called beep with the default properties of that object, whilst the second command specifies the exact tone and length in milliseconds of the beep. [10]

The author also utilised polymorphism in their development of their C# Snakes and Ladders program

```
1  public Player(string playerName, char playerSymbol, int playerScore)
   // For creating a new player but specifying their score
2      {
3          _playerName = playerName;
4          _playerSymbol = playerSymbol;
5          _playerCurrentScore = playerScore;
6      }
7
8      public Player(string playerName, char playerSymbol) // For
   creating a new player with a default score of 0
9      {
10         _playerName = playerName;
11         _playerSymbol = playerSymbol;
12         _playerCurrentScore = 0;
13     }
```

References

- [1] P. Pedamkar, "Advantages of OOP | Explore the Top 9 Advantages OF OOP", *EDUCBA*. [Online]. Available: <https://www.educba.com/advantages-of-oop/>. [Accessed: Apr- 2021]
- [2] R. Half, "4 Advantages of Object-Oriented Programming", *Roberthalf.com*, 2017. [Online]. Available: <https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming>. [Accessed: Apr- 2021]
- [3] T. Janssen, "OOP Concept for Beginners: What is Encapsulation", *Stackify*, 2017. [Online]. Available: <https://stackify.com/oop-concept-for-beginners-what-is-encapsulation/>. [Accessed: Apr- 2021]
- [4] "What is Encapsulation in OOP? | Sumo Logic", *Sumo Logic*. [Online]. Available: <https://www.sumologic.com/glossary/encapsulation/>. [Accessed: 09- Apr- 2021]
- [5] D. Braunschweig, "Encapsulation", *Press.rebus.community*. [Online]. Available: <https://press.rebus.community/programmingfundamentals/chapter/encapsulation/>. [Accessed: Apr- 2021]
- [6] "What is Abstraction in OOPs? Java Abstract Class & Method", *Guru99.com*. [Online]. Available: <https://www.guru99.com/java-data-abstraction.html>. [Accessed: Apr- 2021]
- [7] T. Janssen, "OOP Concept for Beginners: What is Abstraction?", *Stackify*, 2017. [Online]. Available: <https://stackify.com/oop-concept-abstraction/>. [Accessed: Apr- 2021]
- [8] "Python Inheritance (With Examples)", *Programiz.com*, 2021. [Online]. Available: <https://www.programiz.com/python-programming/inheritance>. [Accessed: Apr- 2021]
- [9] "What is Polymorphism in Computer Science? | Sumo Logic", *Sumo Logic*. [Online]. Available: <https://www.sumologic.com/glossary/polymorphism/>. [Accessed: Apr- 2021]
- [10] "Console.Beep Method (System)", *Docs.microsoft.com*. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/api/system.console.beep?view=net-5.0>. [Accessed: 09- Apr- 2021]