Module Code & Module Title

CS6004 – Application Development

Week-4 Assignment

Semester

2025 Autumn

Student Name: Sharvik Khadka

London Met ID: 23050002

College ID: np05cp4a230249

Submitted to: Mr. Khilendra Chaudhary

Assignment Due Date: 2025-11-19

Assignment Submission Date: 2025-11-19

Word Count (Where Required): 1600

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be award

Table of Contents

Table of Figures

Research Report 1: Role of Constructors in Software Development

1. Introduction

Constructors are special methods in object-oriented programming (OOP) that are employed to initialize the objects when they are being created. They are automatically executed when creating objects, are used to initialize initial values, resource allocation, and valid states of objects.

Constructors are specialized methods in object-oriented programming (OOP) crucial for creating and initializing objects, ensuring they are set up with appropriate initial values, thus enhancing software integrity and reliability. They are implemented differently across programming languages like Java, C++, and JavaScript. Constructors are invoked automatically during object creation, which allows immediate setup of an object's state, significantly reducing the risk of runtime errors due to uninitialized attributes. (Microsoft, 2025)

There are two main types of constructors: default constructors, which have no parameters and provide default values, and parameterized constructors, which allow customization during object instantiation. Additionally, constructor overloading and chaining enhance their utility, promoting efficient code management. However, improper use can result in complex logic within constructors and memory management issues, especially in languages requiring manual memory handling like C++. (Microsoft, 2025)

Best practices for constructors include focusing on initialization rather than business logic, which fosters code clarity and maintainability. They hold significant importance in design patterns like Singleton and Builder, contributing to better architectural practices and scalability. (Microsoft, 2025)

Furthermore, constructors provide various advantages such as ensuring data members are initialized at creation, improving code organization by keeping related operations together, enforcing the necessity for complete initialization, and enhancing overall code readability and maintainability. The flexibility afforded by parameterized constructors and constructor chaining further elevates their role in robust software development. (Microsoft, 2025)

2. How Constructors Help in Software Development

2.1 Object Initialization

Constructors enable developers of the object to set default or custom values of the object fields immediately the object is created. This makes sure that the object is prepared to be used.

Example from task 4 : Player class

```
namespace App_Workshop4
{
    6 references
    internal class Player
    {
        public string PlayerName;
        public int Level;
        public int Health;

        // Default constructor
        1 reference
        public Player()
        {
            Console.WriteLine("Default constructor has been called");
        }

        // Parameterized constructor
        1 reference
        public Player(string playerName, int level, int health)
        {
            PlayerName = playerName;
            Level = level;
            Health = health;
        }

        2 references
        public void Display()
        {
            Console.WriteLine($"Player: {PlayerName}, Level: {Level}, Health: {Health}");
        }
    }
}
```

Figure 1: Example from task 4: Player class

This guarantees that any time the parameterized version is called, the player has a legitimate name, level and health.

2.2 Code Reliability

Constructors assist in avoiding the formation of objects in an invalid, incomplete or inconsistent condition. In situations whereby a class needs some values so that it can be used properly, a parameterized constructor can be used to guarantee that the values are supplied when the object is being created. This introduces a sense of rightness at the very outset and minimizes the possibility of mistakes in the course of program execution.

As an example, the creation of a Calculator class can be considered. To be able to perform operation reliably, a calculator might need initial values like the current result, the mode of calculation or particular settings. The class ensures that all Calculator objects are well prepared and have been properly initialized by ensuring that these values are forced through an overloaded parameterized constructor. This will remove the chances of developing a calculator without vital information which would otherwise cause errors, unforeseen action or even incomplete performance.

By doing so, constructors do not only create objects, but also serve as protective measures. They restrict the manner in which an object is to be created and all the necessary parts, dependencies, or settings are given. This renders the entire program more predictable, stable and maintains it more easily.

2.3 Maintainability

The programmers put together initializing code in a single spot.

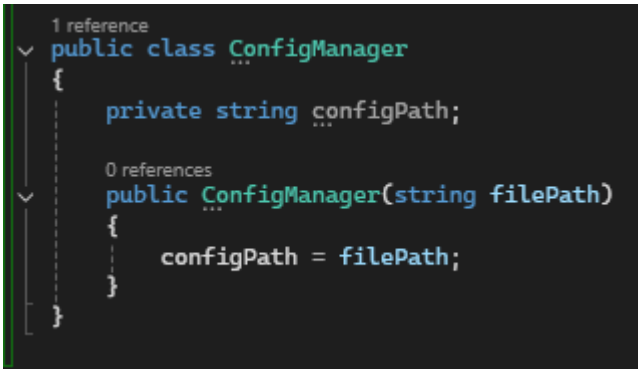
When the rules are adjusted, the developers do not need to adjust all the code everywhere that the objects are created but simply change the constructor.

3. Real-World Use Cases of Constructors

Use Case 1: Database Connection Setup

Connection strings are intoxicated by objects that create database connections (e.g., SQL, MongoDB).

Example Code:



Use Case 2: Game Development (Unity / Unreal)

Constructors are used to determine starting position, health and difficulty of game objects like players, enemies and levels.

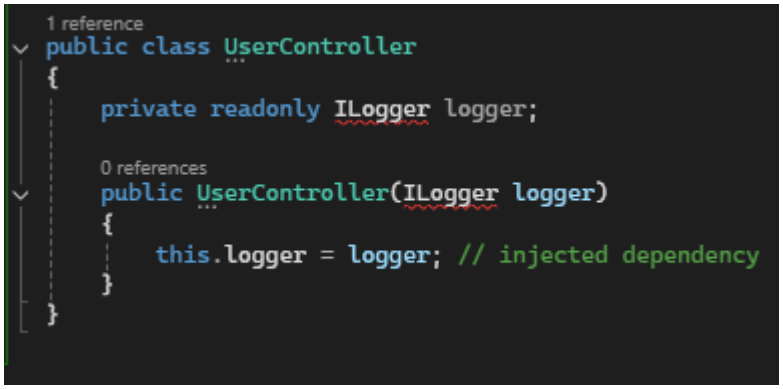
Related to the previous example of Player constructor.

Use Case 3: Dependency Injection

Constructors are used to inject modern frameworks (ASP.NET Core) using services like:

- Logging
- Authentication handlers
- Repository classes

Example Code:



Research Report 2: OOP Principle — Encapsulation

1. Introduction

Encapsulation is a vital concept of object-oriented programming which centers on the integration of data and operations which operate on such data into one unit, which is usually a class. It controls the access to the inner world of an object by concealing its fields and implementations. Rather, the object is interacted with by means of well-defined, controlled public means or properties. This prevents accidental or unauthorized alteration of data, a valid state of an object, and the establishment of a sharp distinction between the use and the implementation of an object. The effects of encapsulation are safer, more modular and more maintainable code.. (Oracle, 2024)

2. What Are Classes and Objects?

Class

A blueprint or a template that defines:

- Data (fields)
- Behaviors (methods)

This is an example on your previous tasks:

```
namespace App_Workshop4
{
    5 references
    internal class Student
    {
        public string name;
        public int age;
        public string address;

        public static string college = "IIC";
```

Figure 2: Example of Class

Object

An instance that is a product of a class is an object:

```
Console.WriteLine("======== Task 1: Student Class ========");
Student s1 = new Student { name = "Shar", age = 20, address = "Kathmandu" };
Student s2 = new Student { name = "Ram", age = 22, address = "Pokhara" };
s1.PrintDetails();
```

Figure 3: Objects Example

3. Understanding Encapsulation

Encapsulation implies that internal data of a class are hidden and only controlled access to them is provided.

A class, which achieves this, makes use of:

- data to be stored in private variables.
- public getters/setters (properties) to regulate the data access or modification.
- public operations to do things without revealing internal information.

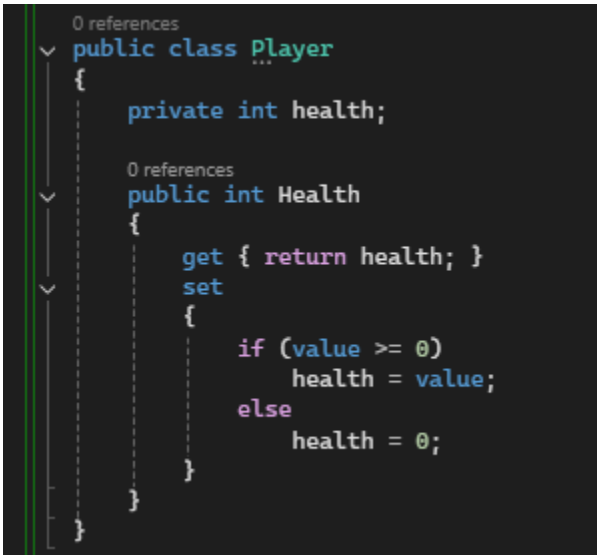
Benefits include:

- Securing information against undesirable modifications.
- Breaking down the class into something simpler.
- The ability to update or to make improvements without interfering with other sections of the program.

4. Practical Examples of Encapsulation

We make simple fields into private fields and access them via getters/setters.

Example 1: Using Properties to Protect Data



```
0 references
public class Player
{
    private int health;

    0 references
    public int Health
    {
        get { return health; }
        set
        {
            if (value >= 0)
                health = value;
            else
                health = 0;
        }
    }
}
```

Advantages

- prevents inaccurate health values
- keeps game logic safe.

Example 2: Calculator Class from Previous task 2

The class only exposes safe methods and conceals internal operations:

```
public class Calculator
{
    public void PrintWelcome()
    {
        Console.WriteLine("Welcome to the Calculator");
    }

    // Creating method Add
    public int Add(int num1, int num2)
    {
        return num1 + num2;
    }

    // Creating method Multiply
    public int Multiply(int num1, int num2 = 1)
    {
        return num1 * num2;
    }
}
```

Figure 4: Calculator Class from Previous task 2

The user can only call permitted functions; they cannot alter internal operations.

5. Why Encapsulation Is Important ?

5.1 Data Security

Encapsulation is used to secure sensitive or important data that is not to be exposed externally but rather hidden within a class and only the necessary information is exposed. External code is not allowed to access private fields and this prevents accidental modification and unauthorized access. The class guarantees that valid and safe operations are only carried out on its own internal state by regulating data using the methods or properties which are public. (Microsoft, 2025)

5.2 Reduced Complexity

Encapsulation conceals the implementation details of a class, and the only interaction with a class is through a simple and clear external interface. This implies that programmers do not have to know the internal workings of the class and only how to use it. This means that programs are simpler to comprehend, operate and debug, as unwarranted internal complexity remains hidden.

5.3 Easier Maintenance

As the internal logic and data structure of a class is not visible, it is possible to make changes within the class without impacting other sections of the program. The rest of the system will still be functioning as usual as long as the public interface does not change. This segregation simplifies the process of updates and fixes and allows them to add to the codebase with fewer chances of creating errors in different parts of the code.

6. Task 6: Debugging

```
Console.WriteLine("\n======== Task 6: Debugging ========");
Console.Write("Enter marks obtained: ");
bool marksOk = int.TryParse(Console.ReadLine(), out int marks);

Console.Write("Enter total marks: ");
bool totalOk = int.TryParse(Console.ReadLine(), out int total);

if (!marksOk || !totalOk || total <= 0)
{
    Console.WriteLine("Invalid input!");
    return;
}

double percentage = marks / total * 100;

Console.WriteLine($"Percentage: {percentage}%");
```

Figure 5: Task 6 Incorrect Code

```
if (!marksOk || !totalOk || total <= 0)
```

```
Console.WriteLine($"Percentage: {percentage}%");
```

6.1 Why is the Output Incorrect?

```
======== Task 6: Debugging ========
Enter marks obtained: 80
Enter total marks: 100
Percentage: 0%
```

Figure 6: Task 6 Incorrect Output

6.2 How to Correct the Program

Root Cause: Integer Division Both marks and total are int. When you divide two integers in C#, the result is an integer (decimal part is discarded). So, 80 / 100 → 0, not 0.80.

Then 0 * 100 → 0

Corrected calculation:

Fixed: Force floating-point division

```
// ← BUG: integer division happens first!
//double percentage = marks / total * 100;

// Fixed: Force floating-point division
double percentage = (double)marks / total * 100;
```

Figure 7: Task 6 Corrected Code

6.3 Final Correct Output Example

```
======== Task 6: Debugging ========
Enter marks obtained: 80
Enter total marks: 100
Percentage: 80%
```

Figure 8: task 6 Correct Output

7. References

Microsoft. (2025). *Constructors (C# programming guide)*. Retrieved Nov 19, 2025, from https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constructors

Microsoft. (2025). *Overview of object oriented techniques in C#* . Retrieved Nov 19, 2025, from https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/object-oriented/

Oracle. (2024). *Lesson: Object-Oriented Programming Concepts* . Retrieved Nov 19, 2025, from https://docs.oracle.com/javase/tutorial/java/concepts/