ELE 503

Advanced Computer Programming and Statistics

Week#7: Advanced C# Programming and Applications in Engineering

By Kingsley E. Erhabor

Week 07.

Advanced C# Programming and Applications in Engineering

Why Advanced C# for Engineers?

Object-Oriented Programming (OOP) in C#
Classes and Objects, Inheritance, Polymorphism, Encapsulation & Abstraction

Real World Applications of OOP Using C#
Applications in Engineering

Example C# Application and Codes

Q&A
Closing Take away

Efosa's Introduction

Engineer | Programmer | Innovator

Technical Authority

Shell Nigeria

Subject Mater Expert (EMEA) for Process Automation & Control (PACO)-Subsea control systems and Subsea Distribution

Containerization 85 %

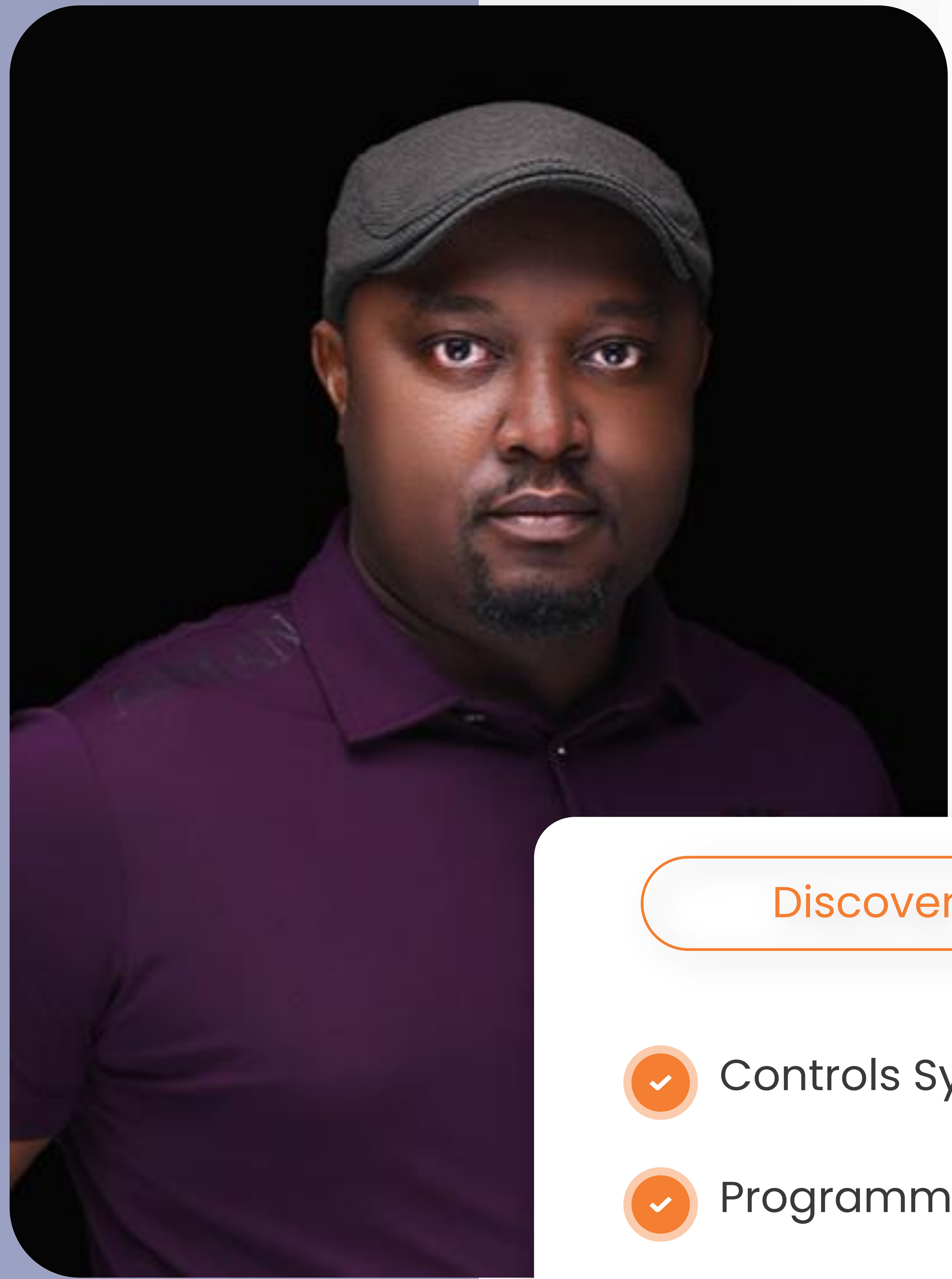
AI & ML 80 %

Renewable Energy 90 %

Innovator, VC

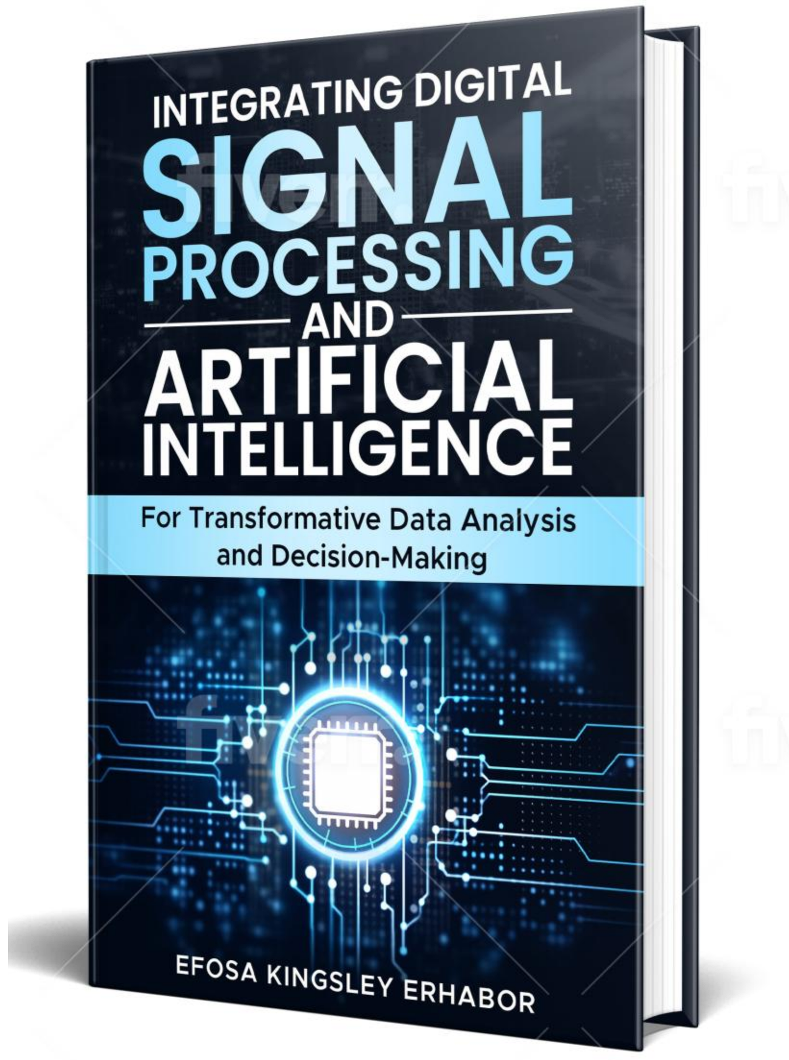
Katharos Technologies

Linux, Devops, AI and Software SME, Innovator and enterprenur



Discover

- Controls System
- Programmer
- Subsea Engineer
- AI and ML



X or Twitter

Linkedin

Part 1:
Why Advanced C# for Engineers?

Learning Objectives

ELE 503: Advanced Computer Programming and Statistics

By the end of this lecture, you will be able to:

1. Dive into object-oriented programming (OOP) concepts using C#.
2. Implement classes, objects, inheritance, and polymorphism in engineering programs.
3. Develop applications that interface with hardware or simulate engineering systems.

Why Advanced C# for Engineers?

Advantages of C#:

- Robust object-oriented features.
- Strong integration with the .NET ecosystem.
- High performance and scalability.
- Extensive libraries for engineering and simulation.
- Versatility in developing desktop, web, and hardware-interfacing application

Part 2:

Object-Oriented Programming (OOP) in C#

Object-Oriented Programming (OOP) in C#

ELE 503: Advanced Computer Programming and Statistics

- **Core OOP Concepts:**
  - Classes and Objects
  - Inheritance
  - Polymorphism
  - Encapsulation
  - Abstraction

- **Benefits of OOP:**
  - Modular and maintainable code.
  - Reusability through inheritance and polymorphism.
  - Improved collaboration in large projects.

Classes and Objects in C#

ELE 503: Advanced Computer Programming and Statistics

- **Classes:**
  - Blueprint for creating objects.
  - Define properties (attributes) and methods (functions).

- **Objects:**
- Instances of classes.
- Represent real-world entities in programs.

- **Example:**
- class Engine { public double Horsepower; public void Start() { } }

Implementing Classes and Objects

ELE 503: Advanced Computer Programming and Statistics

Defining a Class

```csharp
public class Motor
{
    // Properties
    public double Power { get; set; }
    public string Type { get; set; }

    // Constructor
    public Motor(double power, string type)
    {
        Power = power;
        Type = type;
    }

    // Method
    public void Start()
    {
        Console.WriteLine($"{Type} motor with {Power} HP started.");
    }
}
```

Creating an Object

```csharp
Motor electricMotor = new Motor(150, "Electric");
electricMotor.Start();
```

Inheritance in C#

ELE 503: Advanced Computer Programming and Statistics

- **Definition:**
- Mechanism where one class (derived) inherits properties and methods from another (base) class.
- **Benefits:**
- Code reusability.
- Hierarchical classification.
- Simplifies maintenance and scalability.
- **Example:**
- class ElectricMotor : Motor { public double Efficiency { get; set; } }

Implementing Inheritance

ELE 503: Advanced Computer Programming and Statistics

Base Class

```
public class Motor
{
    public double Power { get; set; }
    public string Type { get; set; }

    public Motor(double power, string type)
    {
        Power = power;
        Type = type;
    }

    public void Start()
    {
        Console.WriteLine($"{Type} motor with {Power} HP started.");
    }
}
```

Derived Class

```
public class ElectricMotor : Motor
{
    public double Efficiency { get; set; }

    public ElectricMotor(double power, double efficiency)
        : base(power, "Electric")
    {
        Efficiency = efficiency;
    }

    public void DisplayEfficiency()
    {
        Console.WriteLine($"Efficiency: {Efficiency}%");
    }
}
```

Example

```
ElectricMotor em = new ElectricMotor(150, 95.5);
em.Start();
em.DisplayEfficiency();
```

Polymorphism in C#

ELE 503: Advanced Computer Programming and Statistics

- **Definition:**
  - Ability of different classes to be treated as instances of the same class through inheritance.
- **Types:**
  - Compile-Time (Method Overloading)
  - Run-Time (Method Overriding)
- **Benefits:**
  - Flexibility and scalability.
  - Simplifies code management.
- **Example:**
- Overriding the Start method in derived classes.

Implementing Polymorphism

ELE 503: Advanced Computer Programming and Statistics

Method Overriding (Run-Time Polymorphism)

```
public class Motor
{
    public virtual void Start()
    {
        Console.WriteLine("Motor started.");
    }
}

public class ElectricMotor : Motor
{
    public override void Start()
    {
        Console.WriteLine("Electric motor started silently.");
    }
}

public class DieselMotor : Motor
{
    public override void Start()
    {
        Console.WriteLine("Diesel motor started with a roar.");
    }
}
```

Example

```
Motor motor1 = new ElectricMotor();
Motor motor2 = new DieselMotor();

motor1.Start(); // Output: Electric motor started silently.
motor2.Start(); // Output: Diesel motor started with a roar.
```

Encapsulation in C#

ELE 503: Advanced Computer Programming and Statistics

- **Definition:**
  - Bundling data (properties) and methods that operate on the data within one unit (class).

- **Benefits:**
  - Protects object integrity by restricting access to internal state.
  - Enhances code maintainability and flexibility.

- **Access Modifiers:**
  - **Public:** Accessible from anywhere.
  - **Private:** Accessible only within the class.
  - **Protected:** Accessible within the class and its derived classes.
  - **Internal:** Accessible within the same assembly.

Implementing Encapsulation

ELE 503: Advanced Computer Programming and Statistics

Example

Output

```
Engine engine = new Engine();
engine.UpdateEngine(85.0, 250.0);
Console.WriteLine($"Engine Temperature: {engine.Temperature}°C");
Console.WriteLine($"Engine Pressure: {engine.Pressure} PSI");
```

```
public class Engine
{
    // Private fields
    private double _temperature;
    private double _pressure;

    // Public properties with getters and setters
    public double Temperature
    {
        get { return _temperature; }
        private set
        {
            if (value < -50 || value > 150)
                throw new ArgumentOutOfRangeException("Temperature out of range.");
            _temperature = value;
        }
    }

    public double Pressure
    {
        get { return _pressure; }
        private set
        {
            if (value < 0 || value > 300)
                throw new ArgumentOutOfRangeException("Pressure out of range.");
            _pressure = value;
        }
    }

    // Public method to update temperature and pressure
    public void UpdateEngine(double temp, double pressure)
    {
        Temperature = temp;
        Pressure = pressure;
    }
}
```

Abstraction in C#

ELE 503: Advanced Computer Programming and Statistics

- **Definition:**
  - Hiding complex implementation details and showing only the necessary features of an object.
- **Benefits:**
  - Simplifies interaction with complex systems.
  - Enhances code readability and maintainability.
- **Implementation:**
  - Abstract Classes
  - Interfaces

Implementing Abstraction with Abstract Classes

ELE 503: Advanced Computer Programming and Statistics

```
public abstract class Sensor
{
    public string ID { get; set; }

    public Sensor(string id)
    {
        ID = id;
    }

    // Abstract method
    public abstract double ReadValue();
}

public class TemperatureSensor : Sensor
{
    public TemperatureSensor(string id) : base(id) { }

    public override double ReadValue()
    {
        // Simulate reading temperature
        return 25.0; // Placeholder value
    }
}

public class PressureSensor : Sensor
{
    public PressureSensor(string id) : base(id) { }

    public override double ReadValue()
    {
        // Simulate reading pressure
        return 101.3; // Placeholder value
    }
}
```

Example

```
Sensor tempSensor = new TemperatureSensor("TS-001");
Sensor pressureSensor = new PressureSensor("PS-001");

Console.WriteLine($"Temperature: {tempSensor.ReadValue()}°C");
Console.WriteLine($"Pressure: {pressureSensor.ReadValue()} PSI");
```

Abstract Class Example:

Implementing Abstraction with Interfaces

ELE 503: Advanced Computer Programming and Statistics

```
public interface ICommunicate
{
    void SendData(string data);
    string ReceiveData();
}

public class WirelessModule : ICommunicate
{
    public void SendData(string data)
    {
        Console.WriteLine($"Sending data wirelessly: {data}");
    }

    public string ReceiveData()
    {
        return "Received wireless data.";
    }
}

public class WiredModule : ICommunicate
{
    public void SendData(string data)
    {
        Console.WriteLine($"Sending data via wire: {data}");
    }

    public string ReceiveData()
    {
        return "Received wired data.";
    }
}
```

Example

```
ICommunicate wireless = new WirelessModule();
ICommunicate wired = new WiredModule();

wireless.SendData("Temperature Data");
Console.WriteLine(wireless.ReceiveData());

wired.SendData("Pressure Data");
Console.WriteLine(wired.ReceiveData());
```

Interface Example:

Part 3:

Real World Applications of OOP Using C#

Developing Applications that Interface with Hardware

ELE 503: Advanced Computer Programming and Statistics

- **Key Concepts:**
  - **Serial Communication:** Using COM ports to communicate with hardware devices.
  - **GPIO Control:** Managing General-Purpose Input/Output pins for interfacing with sensors and actuators.
  - **Library Utilization:** Leveraging libraries like System.IO.Ports for serial communication.

- **Example Applications:**
  - Data acquisition systems.
  - Control systems for machinery.
  - Simulation tools for engineering processes.

Simulating Engineering Systems with C#

ELE 503: Advanced Computer Programming and Statistics

- **Benefits of Simulation:**
  - Cost-effective testing of engineering designs.
  - Ability to model complex systems and predict behavior.
  - Enhances understanding of system dynamics.
- **Simulation Tools and Libraries:**
  - **Unity:** For real-time simulations and visualizations.
  - **Mathematical Libraries:** For numerical computations and modeling.
  - **Custom Simulation Frameworks:** Tailored to specific engineering needs.

Building a Simple Engineering Application Step-by-Step

ELE 503: Advanced Computer Programming and Statistics

- **Project Overview:**
  - Develop a Temperature Monitoring System.
- **Steps:**
  1. Define requirements and system architecture.
  2. Create classes for sensors and data processing.
  3. Implement data acquisition from sensors.
  4. Develop a user interface for data visualization.
  5. Test and optimize the application.

Best Practices in Software Development

ELE 503: Advanced Computer Programming and Statistics

- **Code Optimization:**
  - Efficient algorithms and data structures.
  - Minimizing resource consumption.
  - Profiling and performance tuning.
- **Documentation:**
  - Clear and concise code comments.
  - Comprehensive documentation using XML comments.
  - Maintaining up-to-date documentation.
- **Version Control:**
  - Using Git for tracking changes.
  - Collaborating through repositories.
- **Testing:**
  - Unit testing with frameworks like NUnit.
  - Integration and system testing.

Code Optimization Techniques

ELE 503: Advanced Computer Programming and Statistics

- **Efficient Algorithms:**
  - Choosing the right algorithm for the task.
  - Understanding time and space complexity.
- **Data Structures:**
  - Selecting appropriate data structures (e.g., arrays, lists, dictionaries).
  - Leveraging built-in collections in C#.
- **Memory Management:**
  - Minimizing memory leaks.
  - Proper disposal of unmanaged resources.
- **Parallel Programming:**
  - Utilizing multithreading and asynchronous programming for performance.

Documentation and Code Comments

- **Importance of Documentation:**
  - Facilitates code maintenance and updates.
  - Aids in onboarding new team members.
  - Enhances code readability and understanding.
- **Types of Documentation:**
  - **Inline Comments:** Brief explanations within the code.
  - **XML Documentation Comments:** Structured comments for generating documentation.
  - **External Documentation:** Comprehensive guides, user manuals, and API documentation.
- **Best Practices:**
  - Keep comments up-to-date with code changes.
  - Avoid redundant or obvious comments.
  - Use meaningful variable and method names.

Version Control with Git

ELE 503: Advanced Computer Programming and Statistics

- **Benefits of Version Control:**
  - Tracking changes over time.
  - Facilitating collaboration among multiple developers.
  - Managing different versions and branches of the project.
- **Using Git with C#:**
  - Integrating Git with IDEs like Visual Studio.
  - Best practices for commit messages and branching strategies.
- **Basic Git Commands:**
  - git init: Initialize a repository.
  - git add: Stage changes.
  - git commit: Commit changes with a message.
  - git push: Push changes to a remote repository.
  - git pull: Retrieve and merge changes from a remote repository.

Testing in C# Applications

ELE 503: Advanced Computer Programming and Statistics

- **Types of Testing:**
  - **Unit Testing:** Testing individual components or methods.
  - **Integration Testing:** Testing the interaction between different components.
  - **System Testing:** Testing the complete and integrated application.
- **Testing Frameworks:**
  - **NUnit:** A popular unit-testing framework for .NET.
  - **xUnit:** Another widely-used testing framework.
  - **MSTest:** Microsoft's testing framework integrated with Visual Studio.
- **Best Practices:**
  - Write tests alongside development (Test-Driven Development).
  - Aim for high code coverage.
  - Automate testing processes.

Encouraging Mini-Projects

ELE 503: Advanced Computer Programming and Statistics

- **Benefits of Mini-Projects:**
    - Reinforces learning through practical application.
    - Encourages creativity and problem-solving.
    - Provides hands-on experience with C# and engineering concepts.
- **Guidelines:**
    - Define clear objectives and scope.
    - Plan the project with milestones and deadlines.
    - Focus on applying OOP principles and best practices.
- **Project Ideas:**
    1. **Temperature Monitoring System:** Interface with sensors to collect and display temperature data.
    2. **Mechanical Simulation Tool:** Simulate the behavior of mechanical systems under different conditions.
    3. **Data Acquisition Application:** Collect and process data from various engineering instruments.
    4. **Control System Interface:** Develop a user interface to control and monitor machinery.

Conclusion and Further Resources

ELE 503: Advanced Computer Programming and Statistics

- **Recap of Learning Objectives:**
  1. Dive into object-oriented programming concepts using C#.
  2. Implement classes, objects, inheritance, and polymorphism in engineering programs.
  3. Develop applications that interface with hardware or simulate engineering systems.

- **Summary of Key Points:**
  - Advanced OOP concepts enhance C# programming for engineering.
  - Inheritance and polymorphism promote code reuse and flexibility.
  - Best practices in software development ensure efficient, maintainable, and reliable applications

Part 4:
Example C# Application and Codes

Example Slide 1: Building a Temperature Monitoring System

ELE 503: Advanced Computer Programming and Statistics

Title: Building a Temperature Monitoring System with C#

Problem Statement:

- Develop a C# application that interfaces with temperature sensors to collect, process, and display temperature data in real-time.

Execution steps:

Step 1: Define the Sensor Class

Step 2: Implement a TemperatureSensor Class

Step 3: Create the Data Acquisition Module

Step 4: Develop the User Interface

Results and Interpretation:

- The application continuously displays simulated temperature readings from multiple sensors.
- Demonstrates real-time data acquisition and display.
- Showcases the use of OOP principles in structuring the application.

Example Slide 2: Implementing Inheritance and Polymorphism

ELE 503: Advanced Computer Programming and Statistics

Title: Implementing Inheritance and Polymorphism in a Sensor System

Problem Statement:

- Extend the Temperature Monitoring System to include different types of sensors (e.g., HumiditySensor) and demonstrate polymorphism.

Execution steps:

Step 1: Create a HumiditySensor Class

Step 2: Modify DataAcquisition to Handle Different Sensors

Step 3: Update the User Interface

Results and Interpretation:

- The application now supports multiple sensor types, demonstrating polymorphism.
- Each sensor type provides its own implementation of the ReadValue method.
- The user interface dynamically handles different sensor data.

Example Slide 3: Interfacing with Hardware Using Serial Communication

ELE 503: Advanced Computer Programming and Statistics

Title: Interfacing with Hardware Using Serial Communication in C#

Problem Statement:

- Develop a C# application that communicates with an external hardware device (e.g., Arduino) via serial port to receive sensor data.

Execution steps:

Step 1: Setting Up Serial Communication

Step 2: Using the HardwareInterface Class

Results and Interpretation:

- The application establishes a serial connection with the hardware device.
- Receives and displays data sent from the hardware in real-time.
- Demonstrates basic serial communication setup in C#.

Example Slide 4: Developing a Control System Interface

ELE 503: Advanced Computer Programming and Statistics

Title: Developing a Control System Interface with C#

Problem Statement:

- Create a C# application with a graphical user interface (GUI) to control and monitor an engineering system (e.g., a motor controller).

Step 1: Setting Up the Project with Windows Forms

- **Create a New Windows Forms App:**
  - Open Visual Studio.
  - Select Create a new project.
  - Choose Windows Forms App (.NET Framework).

Step 2: Designing the User Interface

- **Add Controls:**

Results and Interpretation:

- The application provides a user-friendly interface to control the motor.
- Buttons trigger start and stop actions, updating the motor status.
- Demonstrates the integration of GUI components with backend logic.

Example Slide 4: Developing a Control System Interface

ELE 503: Advanced Computer Programming and Statistics

Step 1: Setting Up the Project with Windows Forms

- **Create a New Windows Forms App:**
    - Open Visual Studio.
    - Select Create a new project.
    - Choose Windows Forms App (.NET Framework).

Step 2: Designing the User Interface

- **Add Controls:**
    - **Buttons:** Start Motor, Stop Motor.
    - **Labels:** Display Motor Status.
    - **Charts:** Visualize motor performance data.
- **Example Layout:**

Step 3: Implementing Cont

Example Slide 5: Applying Best Practices in C# Development

ELE 503: Advanced Computer Programming and Statistics

Title: Applying Best Practices in C# Development for Engineering Applications

Problem Statement:

- Enhance code quality, maintainability, and performance in C# engineering applications by following best practices.

Results and Interpretation:

- Adherence to SOLID principles enhances code modularity and flexibility.
- Clear separation of concerns makes the codebase easier to maintain and extend.

Example Slide 5: Applying Best Practices in C# Development

ELE 503: Advanced Computer Programming and Statistics

Best Practice 1: Follow SOLID Principles

S: Single Responsibility Principle
O: Open/Closed Principle
L: Liskov Substitution Principle
I: Interface Segregation Principle
D: Dependency Inversion Principle

Best Practice 2: Use Meaningful Naming Conventions

- Use clear and descriptive names for variables, methods, and classes.
- Follow PascalCase for class names and methods, camelCase for variables.

Best Practice 3: Implement Error Handling

- Use try-catch blocks to manage exceptions.
- Provide meaningful error messages.
- Ensure resources are properly disposed using using statements or finally blocks.

Best Practice 4: Optimize Performance

- Minimize memory usage by disposing unmanaged resources.
- Use efficient data structures and algorithms.
- Avoid unnecessary computations within loops.

Best Practice 5: Write Unit Tests

- Ensure code reliability and correctness.
- Facilitate code refactoring and maintenance.
- Use testing frameworks like NUnit or xUnit.