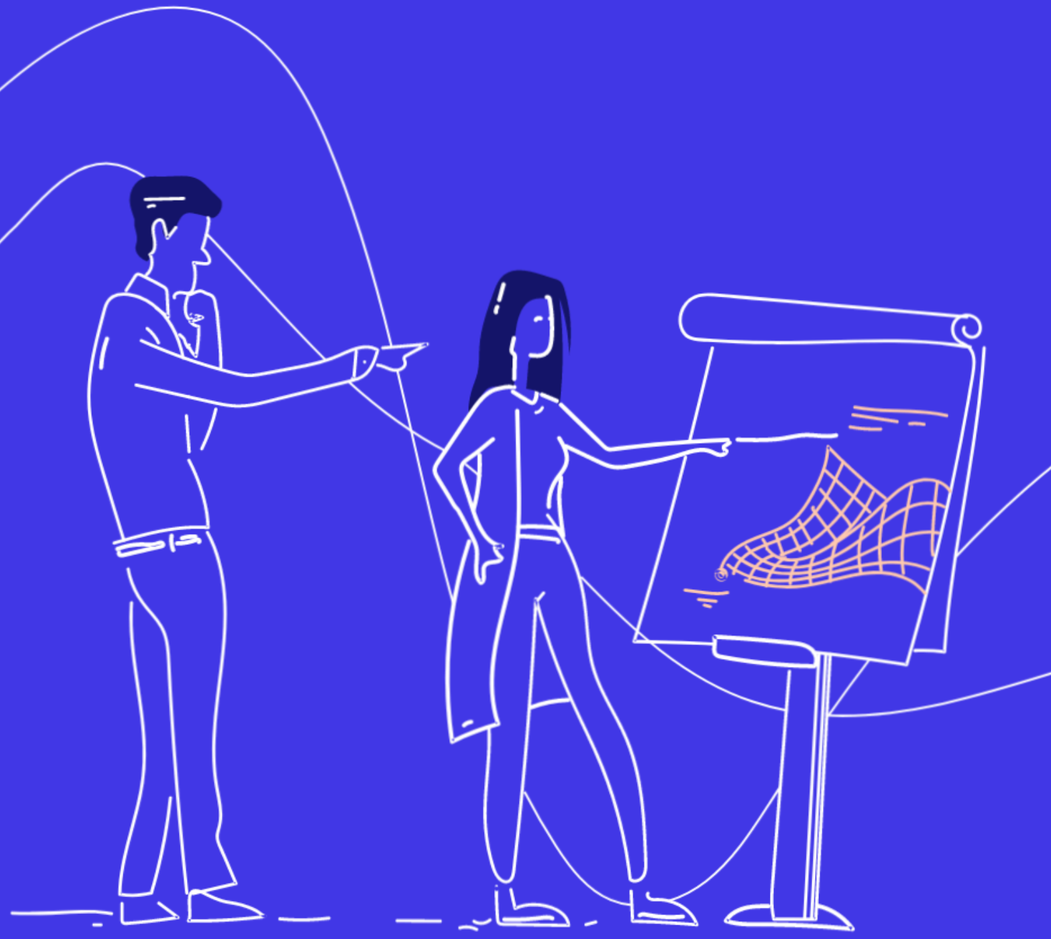
sciengineer

Object-Oriented Design with MATLAB



SciEngineer's training courses are designed to help organizations and individuals close skills gaps, keep up-to-date with the industry-accepted best practices and achieve the greatest value from MathWorks® and COMSOL® Products.

Object-Oriented Design with MATLAB

A code base is difficult to manage over time due to evolving requirements and increasing complexity. Learn how to improve the maintainability and extensibility of your object-oriented code by restructuring it into a set of classes with clear responsibilities and minimized inter-dependency. Understand the root causes of hard-to-manage code projects and how to prevent this by following generic design principles. Solve common design problems and avoid pitfalls by employing specific design patterns. Throughout the training, concepts are explained through use-cases, practical examples, and hands-on exercises. UML class diagrams are employed to visualize the newly introduced concepts and design ideas.

Prerequisites

Object-Oriented Programming with MATLAB or existing knowledge of object-oriented programming in MATLAB.

TOPICS

Day 1

- Object-Oriented Design Principles

Day 2

- Design Patterns

Object-Oriented Design Principles

OBJECTIVE: Follow proven guidelines to make your applications maintainable, flexible, and easy to extend.

- Learn to encapsulate what varies by extracting code that is likely to change into a separate class
- Experience why it's better to favour composition over inheritance, as well as over adding more methods to the class
- Determine whether to use handle or value classes to store data
- Single Responsibility Principle, or why to avoid having a single God Object
- Open-Closed Principle, or how to use hierarchies in order to make your code open for extension, but closed for modification
- Liskov Substitution Principle, or why you should use subclasses to add new functionality, but not to remove unnecessary one
- Interface Segregation Principle, or how several small abstract classes can help you reuse code
- Dependency Inversion Principle, or how depending on abstract instead of concrete methods or properties can make the code more flexible

Design Patterns

OBJECTIVE: Efficiently solve common software design problems by using an established set of classes and class relationships defined by design patterns

- Public Private Property Pair – a pattern that helps ensure two properties always have the same length
- Observer – a pattern enabling a class to automatically react to modifications in another class, ideal for real-time data display in apps
- Template Method – a pattern that ensures different versions of an algorithm always follow the same set of steps, where the steps themselves can differ
- Strategy – a pattern that helps you plug in different versions of an algorithm into another algorithm
- Simple Factory – a pattern for creating an object of a specific subclass, based on a set of rules
- Singleton – a pattern that guarantees there is only one object of a specific class, ideal for ensuring that only one instance of an app is open
- Adapter – a pattern for modifying the user interface of an algorithm, to enable, for example, using legacy code
- Facade – a patter that encapsulates a complicated algorithm requiring several objects and steps

sciengineer

Expand your knowledge