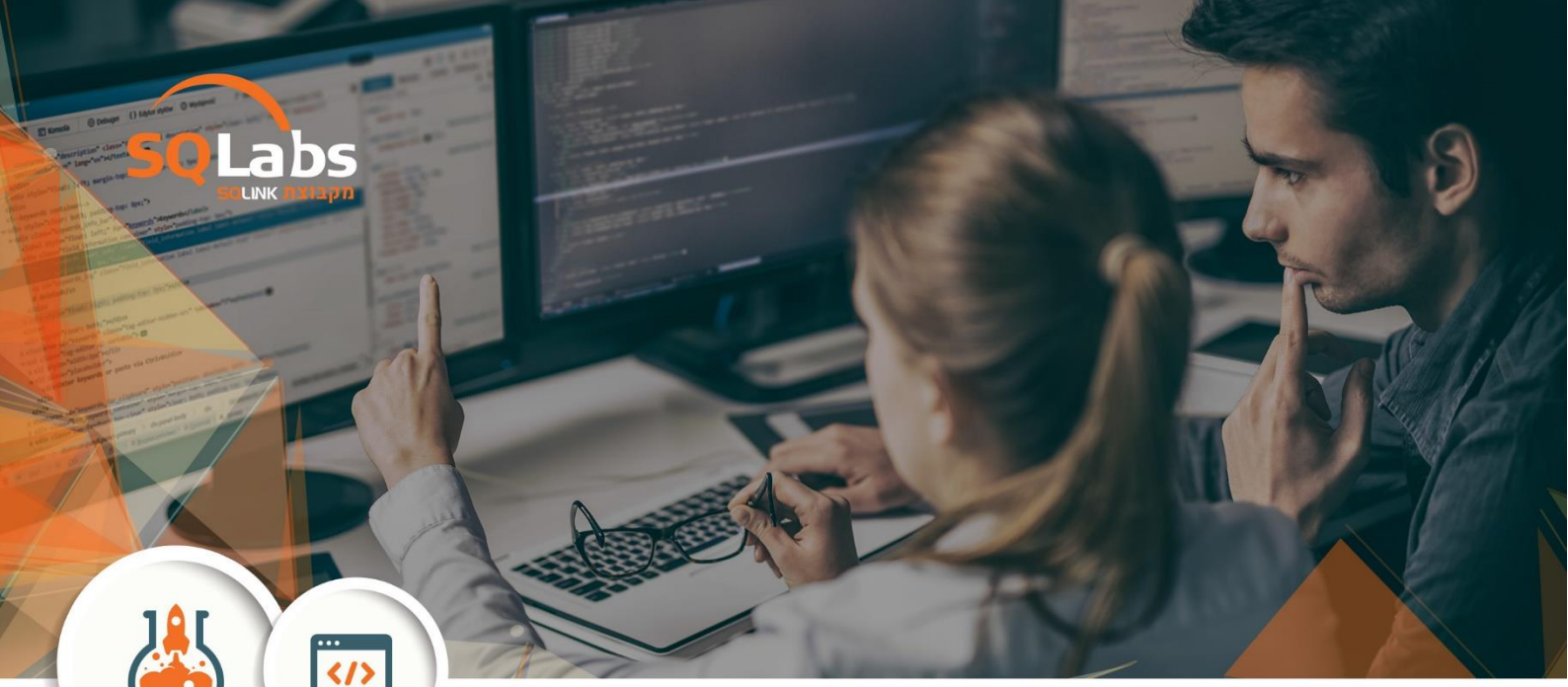

Design Patterns – 40 Hours

Overview

Design patterns describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”

Christopher Alexander

In this course you will get a deep understanding of the most common object oriented design patterns. You will understand how to use them in different projects.

You will know how using design patterns help you to receive higher maintainability, reusability and extensibility.

Motivation

- you get better and more ideas on how to model a particular technical problem
- you can reuse and apply design ideas which are proven to be effective
- you can get to a more appropriate granularity of your classes that model a particular domain

Prerequisites

Understanding Object Oriented Programming basics (C++)

- abstract data type
- encapsulation
- overriding
- overloading
- inheritance
- polymorphism
- abstract classes
- virtual methods

Course Outline Module

Creational Design Patterns

In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.

- **Singleton**
  A class of which only a single instance can exist
- **Factory Method**
  Creates an instance of several derived classes
- **Prototype**
  A fully initialized instance to be copied or cloned
- **Abstract Factory**
  Creates an instance of several families of classes
- **Builder**
  Separates object construction from its representation

Structural Design Patterns

These design patterns are all about Class and Object composition. Structural patterns define ways to compose objects to obtain new functionality.

- **Adapter**
  Match interfaces of different classes
- **Bridge**
  Separates an object's interface from its implementation

- **Composite**
  A tree structure of simple and composite objects
- **Decorator**
  Add responsibilities to objects dynamically
- **Facade**
  A single class that represents an entire subsystem
- **Flyweight**
  A fine-grained instance used for efficient sharing
- **Proxy**
  An object representing another object

Behavioral Design Patterns

These design patterns are about objects communication.

- **Chain of responsibility**
  A way of passing a request between a chain of objects
- **Command**
  Encapsulate a command request as an object
- **Interpreter**
  A way to include language elements in a program
- **Iterator**
  Sequentially access the elements of a collection
- **Mediator**
  Defines simplified communication between classes
- **Memento**
  Capture and restore an object's internal state
- **Observer**
  A way of notifying change to a number of classes
- **State**
  Alter an object's behavior when its state changes
- **Strategy**
  Encapsulates an algorithm inside a class
- **Template method**
  Defer the exact steps of an algorithm to a subclass
- **Visitor**
  Defines a new operation to a class without change