Object Oriented Programming with Java

detailed 2-day course contents

Module 1

Introduction
- OO Overview.
- Why use UML to represent OO concepts?
- Quick history & basics of UML.
- Inheritance and Information hiding.
- Java service concepts.
- OO tools overview.
- Round trip engineering
- Code generation.

Abstract Data types (ADT)
- Handling the changing software development.
- Defining basic operations on a type.
- Abstract data type definition.
- Delaying implementation decisions.
- Exercise: creating an ADT using Java.

Information Hiding
- Knowing how to use the abstract data type.
- Hiding the implementation of the ADT.
- Keeping things simple for the user.
- Not relying on implementation details.
- Extending an ADT's interface.

The Object Oriented View
- Sending a message to an object.
- Determining which method will be used.
- Objects sending messages to each other.
- Object collaboration details.
- Design by contract.
- Designing clean interfaces.

Invoking Methods
- Different ways for method invocation.
- Operations and messages.
- Operation contracts.
- Pre-and post conditions.
- Exercise: creating an operation contract.
- The 5 categories for an operation contract.

Modular Design and reuse
- Creating service packages.
- Conceptual classes.
- Elaborating subsystems from class diagrams.
- The architectural view.
- Exercise: creating multiple service packages.
- The layers pattern.

Subsystems and Components
- Adding organization to class diagrams.
- The coupling between packages
- Avoiding multiple dependencies.
- Grouping criteria.
- The singleton pattern.
- Using deployment diagrams.
- Exercise: mapping components to nodes.
-

Composition and Aggregation
- When to use composition.
- Using composition or inheritance?
- Aggregation versus composition.
- Exercise: modeling using composition.

Module 2

Constructor Functions
- What is a constructor?
- Defining constructors for classes.
- Allocating space for an object.
- Initializing the class.
- Constructors that take no arguments.

Interfaces
- Interfaces.
- When to use interfaces?
- Why are interfaces so important?
- Extending an interface.
- Implementing an interface.
- Exercise: making an interface using Java.

Inheritance
- Software reusability.
- Abstract classes.
- Inheriting class members.
- Derived classes or subclasses.
- The Liskov substitution principle.
- Complete and incomplete classes.

Derived Classes
- Inherited members.
- Converting instances.
- Checking casts.
- Inheritance hierarchy.
- Descendants.
- Ancestors.
- Redefining members.

Shadowing versus Overriding
- Redefining inherited data members.
- Shadowing explained.
- Member visibility.
- Subclass compatibility.
- Compatibility in meaning and behavior.
- Exercise: shadowing and overriding examples.

Polymorphism
- Overriding of member functions.
- Examples based on graphical objects.
- Defining different method behavior.
- Exercise: polymorphism using Java.

Overloading
- Determining the method.
- Determining the return type.
- Built-in overloading for arithmetic operators.
- Overloading used in a Java.

Case Study
- A full-blown example using OO concepts.
- UML diagrams to show OO principles.
- Defining conceptual classes.
- Some GOF patterns.
- State, Singleton, Adapter pattern.
- Going towards a design class diagram
- Creating the final example code.
- The future of Object Oriented languages.