Module (Course Syllabus) Catalogue

2023-2024

College/ Institute	**Technical Engineering**	
Department	**Information System Engineering**	
Module Name	**Object oriented Design**	
Module Code	**OOD204**	
Degree	**Technical Diploma** ☐ **Bachler** **High Diploma** ☐ **Master** [✓ X] **PhD** ☐	
Semester	**second**	
Qualification	**Phd computer science,complex and networks**	
Scientific Title	**Lecturer**	
ECTS (Credits)		
Module type	**Prerequisite** ☐ **Core** [✓ X] **Assist.** ☐	
Weekly hours		
Weekly hours (Theory)	**(3)hr Class**	**( )Total hrs Workload**
Weekly hours (Practical)	**( )hr Class**	**( )Total hrs Workload**
Number of Weeks	**15**	
Lecturer (Theory)	**Dr bzar khidir hussan**	
E-Mail & Mobile NO.	**Bzar.hussan@epu.edu.iq 07504648672**	
Lecturer (Practical)		
E-Mail & Mobile NO.		
Websites		

Course Book

Course Description	Object-Oriented Design is a software development approach to design and implement software system as a collection of interacting stateful objects with specified structure and behavior and is a course that presents an introduction to the design and construction of software systems using techniques that view a system as a set of objects that work together to realize the system's functionality. This course introduces students to the principles and practices of object-oriented design (OOD). Students will learn the fundamental concepts of OOD and apply them to design and implement software systems.
Course objectives	1. Object-oriented methodology is widely accepted as the best methodology for tackling a wide range of software design problems. From its early days, reusability has been one of its most important promises of this methodology. 2. **Understanding OOP Principles:** • Define and explain the core principles of object-oriented programming (OOP), including encapsulation, inheritance, and polymorphism. • Apply the principles to design and implement software solutions. 3. **Class and Object Concepts:** • Differentiate between classes and objects. • Create and manipulate classes and objects in a programming language (e.g., Java,). 4. **Encapsulation:** • Define encapsulation and understand its role in data hiding and abstraction. • Implement encapsulation in classes and explain its benefits. 5. **Inheritance:** • Understand the concept of inheritance and its role in code reuse. • Implement and use inheritance in designing class hierarchies. 6. **Polymorphism:** • Define polymorphism and understand how it contributes to flexibility in software design. • Implement polymorphic behavior through method overloading and overriding. 7. **Design Patterns:** • Introduce common design patterns (e.g., Singleton, Factory, Observer) and their applications. • Apply design patterns to solve specific design problems. 8. **UML Diagrams:** • Introduce Unified Modeling Language (UML) as a tool for visualizing and documenting OOD. • Create and interpret UML diagrams, including class diagrams, object diagrams, and sequence diagrams. 9. **Software Architecture:** • Discuss the relationship between OOD and software architecture. • Design and analyze the architecture of software systems using OOD principles.

	10. **Testing and Debugging:** • Emphasize the importance of testing in object-oriented development. • Develop strategies for testing and debugging object-oriented code
Student's obligation	**1. Active Participation:** • Actively engage in class discussions, ask questions, and contribute to group activities. • Participate in hands-on coding exercises and design projects. **2. Preparation and Readiness:** • Come to class well-prepared by completing assigned readings and reviewing relevant materials. • Be ready to discuss and apply theoretical concepts in practical scenarios. **3. Independent Learning:** • Take the initiative to explore additional resources to deepen understanding. • Practice coding independently to reinforce concepts covered in class. **4. Critical Thinking:** • Apply critical thinking skills to analyze and solve design problems. • Challenge assumptions and explore alternative solutions **5. Utilization of Resources:** • Make effective use of available resources, including textbooks, online materials, and software tools. **6. Attendance and Punctuality:** • Attend all classes and arrive on time. • Notify instructors in advance if unable to attend a class.
Required Learning Materials	➢ Lecture halls with computers, data show equipment for lecture presentations and whiteboard.

Evaluation	**Task**		**Weight (Marks)**	**Due Week**	**Relevant Learning Outcome**
	Paper Review				
	Assignments	Homework			
		Class Activity			
		Report	5		
		Seminar	10		
		Essay			
		Project			
	Quiz		15		
	Midterm Exam		20		

	Final Exam	50		
	Total	100		

Specific learning outcome:	• Students will learn to understand the use of the Java language for developing applications. • Strategies and Actions used to produce the outcome: ▪ Review of the basic concepts of the Java language ▪ Review of advanced concepts of the Java language • Students will learn to understand how to design object-oriented applications, ▪ Strategies and Actions used to produce the outcome: ▪ Study of the object-oriented techniques for programming applications ▪ Study of the object-oriented characteristics of the Java Programming language • Students will learn to be able to implement object-oriented designs using Java. ▪ Strategies and Actions used to produce the outcome: ▪ Study of implementation techniques using Java ▪ Study of the Java API for implementing programs • Students will use the Unified Modeling Language (UML) for modeling the applications. ▪ Strategies and Actions used to produce the outcome: ▪ Study of the UML diagrams in general, Class diagrams, Sequence diagrams, Collaboration diagrams .
Course References:	1. David J. Barnes and Michael Kölling, Objects First With Java, A Practical Introduction Using BlueJ, Third Edition 2. Martin Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling. 3.Barry Holms, Daniel T. Jouse-Object oriented programming 4.Nell Dale, Chip Weems-Programming and problem solving with Java 5.https://www.javatpoint.com/java-oops-concepts 6.Head First Design Patterns" by Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra. 7. "Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin 8.Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Course topics (Theory)	**Week**	**Learning Outcome**
Introduction to OOP ➢ Overview of Object-Oriented Programming (OOP) ➢ Variable, Data Types in Java ➢ Initializing objects in Java ➢ Constructors: Types of constructors	1	Ability to organize code into modular units called classes and objects. Understanding the role of constructors in initializing objects when they are created. Awareness of default constructors and their automatic invocation when an object

➢ Constructor Overloading. ➢ Usage of Java this keyword		is created. Ability to define constructors that accept parameters during object creation. Mastery of constructor overloading, where a class can have multiple constructors with different parameter lists.
Object Model: ➢ Encapsulation, Inheritance, and Polymorphism ➢ Abstraction and Modularity ➢ Interface. ➢ Static Binding variables, instance variables, static method, overloading methods. ➢ Dynamic binding	2-3	Understanding of encapsulation principles, where the internal details of an object are hidden from the outside world. Proficiency in using inheritance to create a hierarchy of classes with shared properties and behaviors. Mastery of polymorphic concepts, allowing objects to take on multiple forms through method overriding and interfaces
Introduction to UML ➢ **Static Modeling:** Class diagrams, Object Diagram ➢ **Dynamic Modeling**: Interaction, Activity, Sequence Diagram ➢ Modeling relationships and associations ➢ **Functional Model** - DFD, Constraints, Relation of Functional to Object and Dynamic	4-5	UML provides a standardized way to visually represent the structure and behavior of a system. UML enables the creation of models that capture different aspects of a system, including its structure, behavior, and interactions. Use case diagrams illustrate the interactions between system components and external entities, representing the functional requirements of a system. Class diagrams depict the static structure of a system, including classes, their attributes, and relationships
Graphical User Interface (GUI) ➢ Introduction to GUI Programming ➢ Event-driven programming, Principles of GUI Design, GUI Programming Issues, Parts of a GUI Program. ➢ Building a GUI ➢ Using a GUI Component Components and Events	6-7	The GUI should be intuitive, allowing users to easily understand and navigate the application. GUI design aims to provide a user-friendly interaction that aligns with users' expectations and mental models. Maintain consistency in design elements, such as colors, fonts, and layout, across the entire application. Ensure that the GUI is responsive and adapts to different screen sizes and resolutions
Design Principles: ➢ SOLID principles (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) ➢ DRY (Don't Repeat Yourself) and KISS (Keep It Simple, Stupid) principles ➢ Design by Contract	8	**(SRP):** Each class should have only one reason to change. **(OCP):** Software entities (classes, modules, functions) should be open for extension but closed for modification. **(LSP):** Subtypes must be substitutable for their base types

		without altering the correctness of the program. **(ISP):** Clients should not be forced to depend on interfaces they do not use. **(DIP):** High-level modules should not depend on low-level modules, but both should depend on abstractions.
Design Patterns : ➢ Introduction to design patterns ➢ Creational patterns (e.g., Singleton, Factory, Builder) ➢ Structural patterns (e.g., Adapter, Decorator, Composite) ➢ Behavioural patterns (e.g., Observer, Strategy, Command)	9	Reusable solutions to common design problems that help guide the overall architecture of the system. Benefits: Encourages best practices, provides a common vocabulary for developers, and accelerates the design process
Software Architecture: ➢ Overview of software architecture ➢ Layered architecture, client-server architecture, MVC ➢ Microservices and SOA (Service-Oriented Architecture)	10	Capability to understand and decompose complex systems into manageable and modular components. Recognition and application of common architectural styles (e.g., client-server, microservices) and design patterns. Understanding of principles for designing scalable and high-performance systems.
Testing in OOD: ➢ Unit testing and test-driven development (TDD) ➢ Integration testing and mocking ➢ Code coverage and continuous integration	11	Design classes and modules with testability in mind, allowing for easy and effective unit testing. Benefits: Improves code quality, identifies and prevents bugs early in the development process, and supports continuous integration and delivery practices.
Practical Topics	**Week**	**Learning Outcome**
• Short answer questions: This includes for example, multiple choice, True/False and fill-in-the-blank type questions. • Code analysis questions: We will give a short segment of code and you may be asked to identify syntax and logical errors, generate code output, etc. • Code Writing: Write a program/code snippets to solve a given problem. You should be prepared to give a complete program, including full class heading, import, package statement, and main methods, but we may		

also ask you to provide just a single method or a code fragment.		
Extra notes:		
External Evaluator **Shahab Wahhab Kareem** Shahab		