ICS C250: OBJECT-ORIENTED PROGRAMMING

Item	Value
Curriculum Committee Approval Date	11/17/2023
Top Code	070700 - Computer Software Development
Units	3 Total Units
Hours	72 Total Hours (Lecture Hours 54; Lab Hours 18)
Total Outside of Class Hours	0
Course Credit Status	Credit: Degree Applicable (D)
Material Fee	No
Basic Skills	Not Basic Skills (N)
Repeatable	No
Open Entry/Open Exit	No
Grading Policy	Standard Letter (S), • Pass/No Pass (B)

Course Description

This course serves as an introduction to the fundamental principles and practices of object-oriented programming (OOP). Object-oriented programming is a key paradigm in software development, and this course provides students with the knowledge and skills to design, implement, and maintain software using OOP concepts. Students will learn to create classes, objects, and methods, and apply encapsulation, inheritance, and polymorphism to solve real-world programming problems. Hands-on coding exercises and projects will reinforce the theoretical concepts taught in the course. ADVISORY: ICS C120 and ICS C123. Transfer Credit: CSU.

Course Level Student Learning Outcome(s)

1. Create classes and objects in an object-oriented programming language.
2. Write programming code to implement encapsulation, inheritance, and polymorphism to model real-world entities and relationships.
3. Analyze and refactor given source code to improve software maintainability and readability.

Course Objectives

- 1. Describe the core principles and concepts of object-oriented programming.
- 2. Provide examples on how to create classes and objects in an object-oriented programming language.
- 3. Explain how to implement encapsulation, inheritance, and polymorphism to model real-world entities and relationships.
- 4. Provide real-world examples of methods to develop software applications using object-oriented design principles.
- 5. Show examples of how to analyze and refactor code to improve software maintainability and readability.
- 6. Explain the use of design patterns to solve common software design problems.
- 7. Provide methods and techniques to effectively debug and test object-oriented code.
- 8. Facilitate collaboration with peers to design and implement object-oriented projects.

Lecture Content

Introduction to Object-Oriented Programming (OOP) Overview of OOP and its significance in software development. OOP principles and terminology. Classes and Objects Creating classes and objects in an object-oriented language. Constructors and destructors. Encapsulation Access control and data hiding. Getters and setters for data protection. Inheritance Extending classes and inheriting attributes and methods. Method overriding and base classes. Polymorphism Polymorphic behavior and method overloading. Interfaces and abstract classes. Object-Oriented Design Principles SOLID principles (Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion). The importance of modularity and design patterns. Exception Handling Handling exceptions in object-oriented code. Try-catch blocks and custom exception classes. Unit Testing in OOP Writing unit tests for object-oriented code. Test-driven development (TDD). Object-Oriented Programming Language Features Language-specific features in popular OOP languages (e.g., Java, C++, Python, C#). Design Patterns Common design patterns (e.g., Singleton, Factory, Observer) and their applications. Object-Oriented Software Development Applying OOP principles to design and develop complete software applications. Collaborative group projects that apply OOP concepts to solve real-world problems.

Lab Content

Creating Classes and Objects: Write a Python program that defines a class with attributes and methods, and create instances (objects) of that class. Encapsulation and Access Control: Create a Python class with private attributes and use getter and setter methods to access and modify those attributes. Inheritance and Method Overriding: Define a base class and a derived class that inherits from the base class. Override methods in the derived class. Polymorphism and Duck Typing: Create multiple classes that implement a common interface, and demonstrate polymorphic behavior using these classes. Class Composition and Aggregation: Define classes that have relationships with other classes through composition and aggregation. Abstract Classes and Interfaces: Create an abstract class (using the abc module) and implement concrete classes that inherit from it. Exception Handling in OOP. Write Python classes that raise and handle exceptions, demonstrating how exception handling can be used in OOP. File I/O and Serialization: Create classes that handle file I/O and data serialization (e.g., JSON or Pickle) in Python.

Method(s) of Instruction

- Lecture (02)
- DE Live Online Lecture (02S)
- DE Online Lecture (02X)
- Lab (04)
- DE Live Online Lab (04S)
- DE Online Lab (04X)

Instructional Techniques

This course will utilize a combination of lecture, remote virtual machine assignments, classroom/discussion student interactions, problem-solving, quizzes, tests, and troubleshooting assignments to achieve the goals and objectives of this course. All instructional methods are consistent across all modalities.

Reading Assignments

Students will be asked to read and review academic concepts in the textbook and other materials while experiencing how working code behaves under different conditions. Thus, practical programming techniques are taught through personal understanding, hands-on discovery, active learning, and discussing concepts and their results with others.

Writing Assignments

Written assignments will focus on demonstrations of programming skills. Written assignments will include opportunities to compare and contrast code with student examples and real-world examples.

Out-of-class Assignments

The problem-solving exercises will include analysis of programming language for common coding practices. Programming demonstrations are included in the hands-on projects.

Demonstration of Critical Thinking

Students will be asked to read and review academic concepts in the textbook while experiencing how working code behaves under different conditions. Thus, practical programming techniques are taught through personal understanding, hands-on discovery, active learning, and discussing concepts and their results with others.

Required Writing, Problem Solving, Skills Demonstration

The problem-solving exercises will include programming language analysis and comparisons.Programming skills demonstrations are included in the Projects, Midterm and Final examinations.

Eligible Disciplines

Computer information systems (computer network installation, microcomputer ...: Any bachelor's degree and two years of professional experience, or any associate degree and six years of professional experience. Computer science: Master's degree in computer science or computer engineering OR bachelor's degree in either of the above AND master's degree in mathematics, cybernetics, business administration, accounting or engineering OR bachelor's degree in engineering AND master's degree in cybernetics, engineering mathematics, or business administration OR bachelor's degree in mathematics AND master's degree in cybernetics, engineering mathematics, or business administration OR bachelor's degree in any of the above AND a master's degree in information science, computer information systems, or information systems OR the equivalent. Note: Courses in the use of computer programs for application to a particular discipline may be classified, for the minimum qualification purposes, under the discipline of the application. Master's degree required.

Textbooks Resources

1. Required Lott, S.; Phillips, D.. Python Object-Oriented Programming, 4th ed. Packt Publishing, 2021

Other Resources

1. Technology related white papers and articles are available at no charge to all students at multiple sites as recommended by the instructor. 2. Coastline Library