CPSC-24500: Object-Oriented Programming

Instructor: Eric Pogue, Adjunct Instructor, Computer and Mathematical Sciences (CAMS)
- Discussion: Tuesdays 3 PM (optional)
- Office Hours: W 3-5 PM and by appointment
- Materials: Blackboard plus course site at http://www.epogue.info/CPSC-24500
- Email: epogue@lewisu.edu
- Phone: (630) 613-7088

Course Description:

Students will learn to design and develop software using the object-oriented approach. Topics include encapsulation, inheritance, polymorphism, abstraction, and patterns. Students will learn how to use an SDK to develop desktop and web applications that provide data processing and visualization services. Students will also learn how to manage threads and networking connections in software they write.

Credits: 3

Prerequisites:

CPSC 21000 Programming Fundamentals or consent of instructor

Course Objectives:

Students will be use basic object-oriented constructs to write complex programs in multiple object-oriented languages. Additionally, students will be able to design and implement efficient data-intensive multi-threaded applications that interface with each other over networks.

Student Learning Outcomes:

On the successful completion of this course you will be able to:

- Solve problems by writing programs using standard language elements such as data declarations, arithmetic operations, conditional statements, loops, and functions
- List and explain the key concepts of object-oriented development: inheritance, abstraction, information hiding, and polymorphism.
- Describe problems that typically plague software: rigidity, fragility, immobility
- define the following object-oriented patterns: Factory, Singleton, Delegation, and Model-View-Controller
- Define and provide examples for object-oriented design principles: Liskov Substitution Principle, Dependency Inversion Principle, Interface Segregation Principle, Open-Close Principle, Single-Responsibility Principle
- Write class definitions and create objects from them
- Declare and use special types of functions for classes, including constructors, accessors, and mutators, and properties
- Create hierarchies of classes that start with abstract base classes and add functionality in descendant classes.
- Design an object-oriented program in UML (Unified Modeling Language) that is organized around a set of classes whose objects interact
- Describe what exceptions are and write programs that deal with them
- Perform screen-scraping by retrieving data from a website.
- Write programs that use various collections
- Use generic data types in programs
- Work with collections of objects from related classes polymorphically

- Explain the difference between classes and interfaces
- Define interfaces that specify behaviors that certain objects must have
- Perform input and output with text file streams
- Perform input and output with xml file streams and serialization
- Use an API as a reference when writing programs
- Build attractive, intuitive graphical user interfaces
- Write programs that use a graphical interface and manage user events using event-handling
- Describe and use the client-server computing model
- Write a program that stores and retrieves data with a relational database
- Create a user interface in Python and Java
- Describe how Java achieves cross-platform compatibility
- Distinguish among heavyweight and lightweight components
- Define callback function as it relates to event handling
- Respond to user events in Java and Python
- Describe how layout managers arrange components
- Write unit tests to verify the correctness of software modules

Course Outline (with weekly student learning outcomes):

Week 1

- Define object-oriented programming
- Distinguish between an object and a class
- Describe how object-oriented programming is fundamentally different from the programming you have done before.
- Justify the choice to use an object-oriented approach to developing software.
- Identify and define six object-oriented concepts: abstraction, encapsulation, information hiding, ownership, inheritance, and polymorphism
- Distinguish between aggregation and composition as two different kinds of ownership.
- Identify the superclass and the subclass in an inheritance relationship.
- Demonstrate inheritance, ownership, and polymorphism in snippets of Java code
- Depict classes and their relationships using UML class diagrams.
- Define and demonstrate the software patterns delegation, singleton, factory, and model-view-controller
- Explain through examples the following software design principles: Open-Closed, Dependency-Inversion, Interface Segregation, Liskov Substitution, and Single Responsibility.
- Identify and describe characteristics of bad software: rigidity, immobility, fragility

Week 2

- Write definitions for classes that comprise the model of a model-view-controller application.
- Use inheritance to build classes that expand upon the capabilities and features of classes that came before.
- Declare an abstract class and explain why creating abstract classes is useful.
- Implement information-hiding by adding private data members to your model classes and including public get and set functions fo accessing their values.
- Write default and non-default constructors for a model class.
- Redefine the toString function a class inherits from the base java.lang.Object class to represent the object as a String.
- Distinguish between function overloading and function overriding.
- Create objects that are instances of a model class.
- Store multiple objects in a Java collection called an ArrayList.
- Define what generic means as applied to a data type.

- Distinguish between using an array and an ArrayList to store a collection of objects.
- Use Javadocs to provide attractive and comprehensive documentation for your code.
- Work with a collection of related objects polymorphically.
- Explain how polymorphism is implemented behind the scenes.

Week 3

- Explain the difference between lightweight and heavyweight components.
- Identify lightweight and heavyweight components.
- Describe how lightweight and heavyweight components render themselves.
- Explain what a layout manager does and identify and describe three of them.
- Specify layout managers for heavyweight and lightweight components.
- Use paint and paintComponent's Graphics object to draw a variety of shapes.
- Explain how to achieve true Model-View-Controller architecture by removing any reference from the view to the model and from the model to the view.
- Distinguish between extending a class and implementing an interface.
- Create an event handler that implements the ActionListener interface to respond to the user clicking on a button.
- Explain multiple ways to implement an event handler for a particular object and event (anonymous inner classes vs. named classes vs. having the frame itself serve as the handler).

Week 4

- Create interactive applications that adhere to the Model-View-Controller pattern.
- Write code that responds to a variety of events, including clicking the mouse on a frame or panel, moving or dragging the mouse, and typing a key on the keyboard.
- Implement animation using a timer and a corresponding event handler.
- Build a menu system that enables the user to trigger a variety of actions in a familiar way.
- Create multiple, intuitive ways for a user to perform a particular task.
- Design and implement a controller class that outputs data to a text file.
- Design and implement a controller class that inputs data from a text file and builds a collection of objects from the read data.
- Describe how Java achieves speedier input and output through its hierarchy of input and output classes.

Week 5

- Explain why software testing is important for modern applications.
- Define various terms related to the process of software testing.
- Distinguish between the various kinds of testing: unit testing, integration testing, and performance testing.
- Compare testing state with testing behavior.
- Explain the purpose of annotations in Java source code.
- Explain the purpose and syntax of the various assert statements junit supports.
- Install junit onto your machine and make it available in a Java project you write.
- Write and execute a junit test for a simple application you write.
- Write javadoc comments that describe the purpose, inputs, and outputs of classes and their components.
- Build javadoc documentation using Java's javadoc command-line tool.
- Identify reasons to use a Java package.
- Explain what a Java package is in terms of both how it is used and how it is stored.
- Compile a class from the command line so that it belongs to a particular package.
- Import a class you write that belongs to a particular package.
- Identify reasons for using jar files to group together related java classes.

- Create a jar file that stores the contents of a particular package.
- Provide access to the classes you've added to a jar file in an application you write.

Week 6

- Identify characteristics of C# (possibly Python) that distinguish it from other languages.
- Work with C# lists.
- Write non-object-oriented programs that use sequence, selection, and repetition to accomplish tasks.
- Write C# functions and place those functions in separate libraries.
- Define a C# class, complete with data members, functions / methods, and an initializer (constructor).
- Use C# built-in text file objects to create and read text files.
- Use inheritance to create a hierarchy of classes that are related to each other.
- Create objects of classes and use them to carry out the work of your program.
- Deal with a list of related objects polymorphically.

Week 7

- Design and implement a Model-View-Controller application in C#.
- Perform basic drawing operations using a popular C# drawing package.
- Implement input and output with plain text files using a controller class designed for that purpose.
- Separate code among various libraries that you can reuse in other applications by importing them.
- Implement information hiding in C# using both accessor and mutator functions and properties.
- Download documents from remote servers from your application.
- Parse data expressed in XML format.
- Separate an application's functionality among classes having well-defined function sets and purposes.

Week 8

- Design, build, and query a simple database using SQL
- Write a class to query a SQL database in C#.
- Encapsulate the data returned by an SQL query in lists of objects so that your program can process them.
- Program the interaction of a client and server at the socket level.
- Describe what a thread is and why it can be useful to distribute tasks among multiple threads
- Write a multi-threaded application in C#.
- Explain why it is important to synchronize threads that need to share access to particular data sources.

Textbook:

No textbook is required for this course. The course notes will give serve as the text.

Recordings:

Because there is no required textbook, each session is recorded and posted online.

Course Materials:

Class notes, assignments, video recordings, and other course materials will be posted on the course's Blackboard site [link] or provided at the course content site [link]. All assignments must be submitted using the Blackboard assignment area. You must make sure that your email address is recorded on Blackboard, because Blackboard's email facility will be utilized to send messages to the class.

Course Requirements:

Homework will be assigned weekly to help you keep your skills sharp. Each homework will include a requirement to post to the Discussion Board and respond to another student's post. Your performance on the homework will determine your grade in the course.

Weekly Schedule:

- Each week begins on a Monday. The weekly lecture video(s) will be posted no later than Monday morning at 8 AM. Please review the video(s) and weekly assignments by noon on Tuesday.
- A live weekly discussion will take place on Tuesdays at 3 PM. It will last no more than 90 minutes. This is an optional discussion and will be focused on questions related to the lecture video(s) and the assignments for the week.
- Write an original post on the Discussion Board by Thursday night at 11:59pm.
- Respond to another student's post on the Discussion Board by Friday night at 11:59pm.
- Submit the week's homework by Sunday night at 11:59pm.

Grading Policies:

Final course letter grade will be determined using the following approximate scale:

A	>= 90	C-	70-72.99
B+	87-89.99	D+	67-69.99
B	83-86.99	D	63-66.99
B-	80-82.99	D-	60-62.99
C+	77-79.99	F	< 60
C	73-76.99		

Withdrawal from Course:

Important dates:

- 100% - March 26th
- 50% - March 26th
- Withdrawal – April 18

Participation:

You must participate consistently throughout the course by reading the notes, working on the examples from class, attending the class session or watching the class recordings, and working on the homework gradually rather than all at once. If you don't, you won't succeed in this course, because you won't be building a deep understanding of the course material.

Late Homework:

Students must turn in all assignments on time. No late homework will be accepted except under serious and documented circumstances.

Plagiarism:

Copying work from each other or from the Internet will be punished harshly and appropriately. I intend to use a system called MOSS (Measure of Software Similarity) [link] which was created at Stanford and is used at CS departments throughout the world. If it is concluded (through MOSS or otherwise) that you have copied someone else's work, you will receive negative the number of points the assignment or test is worth. For example, if the assignment is worth 20 points, you will receive -20 out of 20 points on that assignment. If you do this twice, you will fail the course. There is no excuse for dishonesty, and it will not be tolerated.

Connection to Mission:

The Mission of Lewis University is to prepare lifelong learners who will use their knowledge, faith, wisdom, and talents to improve the lives of others. In our modern, digital world, there is no field that can as profoundly improve the lot of others than Computer Science. By taking this third course in the major, you will be moving much closer to your goal of becoming a world-improving computer scientist, one who can write efficient data-aware applications that inform.

Additional Assistance:

If you have a disability that may require consideration by your instructor and you have not previously submitted documentation to the staff in the Leckrone Academic Resource Center (LARC), please make an appointment with Denise Rich, Director of Academic Support Services in LARC (x5593). It is recommended that you address this prior to the start of class or within the first week of class. If you need accommodations for successful participation in class activities prior to your appointment in LARC, you should provide information in writing to your instructor that includes suggestions for assistance in participating in and completing class assignments. It is not necessary to disclose the nature of your disability to your instructor. For more information about academic support services, visit the LARC website at www.lewisu.edu/larc.

Closing Words:

Most modern, sophisticated software is created in an object-oriented way. Thanks to this course, you'll get good at programming in that way, which means your programming is going to improve significantly over the next several weeks. Just make sure you let me know when you're struggling so that I can help you be successful. Finally, be sure to take a moment and enjoy the journey.

This learning space is an extension of Lewis University's Sanctified Zone, a place where people are committed to working to end racism, bias and prejudice by valuing diversity in a safe and nurturing environment. This active promotion of diversity and the opposition to all forms of prejudice and bias are a powerful and healing e*xpression of our desire to be Signum Fidei, "Signs of Faith," in accordance with the Lewis Mission Statement. To learn more about the Sanctified Zone, please* visit: http://www.lewisu.edu/sanctifiedzone.