Object Oriented Design & Programming II

Computer Science 262

Instructor:	Erik Krohn
E-mail:	krohne@uwosh.edu
Text Message:	920-644-3745
Class Time:	Tuesday, Wednesday & Thursday: 11:30am - 12:30pm
Classroom:	Halsey 237
Lab Time:	Monday: 11:30am - 12:30pm
Lab Location:	Halsey 101C
Office Location:	Halsey 216
Office Hours:	Monday: 12:30pm - 1:30pm Tuesday: 12:30pm - 1:30pm Wednesday: 9:20am - 10:20am Thursday: 9:20am - 10:20am
Prerequisites:	Math-108 or equivalent with a grade of C of better, or qualifying for a higher level mathematics via the Mathematics Placement Test, and CompSci-221 or equivalent with a grade of C or better.
Course Website:	`http://www.uwosh.edu/d2l`
Recommended Textbook:	Introduction to Java Programming, by Y. Daniel Liang Any Edition (10th recommended)

Course Information

A second course in problem solving, software design, and computer programming using an object-oriented language. Problem solving/software design topics include: abstract data types, universal modeling language (UML), simple recursion, unit testing, event-handling, simple concurrency. Data structures and algorithms include: binary search, simple sorting algorithms, use of collection classes and their iteration protocols, sequential file processing. Additional topics include: inheritance, polymorphism, graphical user interfaces, simple use of threads.

Course Website

You should check d2l on a regular basis - it will contain lecture notes, handouts, assignments, announcements, and grades. Ill do my best to let you know when something new and important comes up, but it is your responsibility to check the web site frequently for information that you might not get otherwise.

Mini Assignments

You will have daily mini assignments. Mini assignments are generally short and should take less than fifteen minutes to complete. You will be assigned a mini assignment every lecture to ensure you are staying current with the material. I will drop your 2 lowest mini assignments. Not all mini assignments will be graded. **No late mini assignments will be accepted.**

Projects & Labs

Most projects and labs will consist of short programming projects. One of your goals (during this class and beyond, in Java or any programming language) should be to write understandable, readable code. You should be making every effort to comment anything that might be confusing to a reader unfamiliar with your program, to name variables intelligently, to use indentation that reflects the code's organization, and so on. All of this will be taken into account during grading: poorly organized or written code may have a negative impact on your grade, even if the resulting program works fine.

One of the goals of this class is to teach you to write functioning programs in Java - thus, your code must compile in order for you to receive any credit. Code that does not compile will not be tested and your score will be a 0. Keep this in mind when writing programs: write your code in small pieces, making sure each piece works before moving on to the next one. It is much better to turn in a project that is not finished but has many working pieces than to turn in one that doesn't work at all, even though most of the code is written.

All assignments must be submitted electronically via d2l. It is your responsibility to ensure that your submission was submitted correctly. You must double check to ensure your program was uploaded correctly. Each assignment must be submitted by 11:00pm on the night of the due date. **There are no late submissions**.

Project Resubmissions

Each project must pass **all** of my test cases. A program that fails just one test case will receive at most 70%. That said, you will be allowed to resubmit assignments *once* for regrading within 10 days of the initial program being graded. Complete test cases will be posted after the initial grading so you can go back and test your code to see what went wrong. Regraded programs can receive a maximum score of 85% if the modifications made were very minor. Major modifications or additions will receive very little, if any, additional points.

Exams

Exam material will come from the lecture notes, labs, mini assignments, book and assignments. There will be more information about each exam as it approaches. The *tentative* exam dates are listed below. All exams will be taken during the regular class period. These may change, so as the date approaches make sure you've got the most recent information.

- **Exam One** - Wednesday, March 2^{nd}, 2016
- **Exam Two** - Wednesday, April 6^{th}, 2016
- **Exam Three** - Wednesday, May 11^{th}, 2016

If you are unable to take a scheduled exam, it may be possible to take a make-up exam provided that you do both of the following, which are then subject to my approval:

1. Make arrangements prior to the scheduled exam. For last minute emergencies, telephone me at 424-7080 or leave a message at the computer science office, 424-2068 or send me a text message No after-the-fact notifications will be accepted.

2. Have a written medical excuse signed by the attending physician OR have a note of justification from the Dean of Students Office.

If allowed, only one make-up exam will be given. It will be a comprehensive exam given at an arranged time during the last week of the semester.

Grading

Course grades will be based on assignments, mini assignments and exams. Your final grade will be computed with the following percentages:

- 40% - projects
- 15% - mini assignments/labs
- 45% - exams

If you believe anything was graded incorrectly or unfairly and would like to have it regraded, you must let me know about it within *one week* of having the item graded. I will regrade the entire assignment or exam and you may gain or lose points.
Grading will be on a plus/minus system. Grading may be done on a curve depending on the overall performance of the class. If no curve is used, your grade will be computed based on the following:

Percentage	Grade	Percentage	Grade
≥ 92	A	72 - 78	C
90 - 92	A-	70 - 72	C-
88 - 90	B+	68 - 70	D+
82 - 88	B	62 - 68	D
80 - 82	B-	60 - 62	D-
78 - 80	C+	< 60	F

Learning Outcomes

- Debugging Java programs with BlueJ Students will be expected to:
  - analyze a program on its correctness
  - identify software bugs with the debugger in BlueJ
- Objects and Classes Students will be expected to:
  - specify a class with the UML graphical notation
  - use the UML graphical notation to describe classes
  - distinguish between object reference and primitive data type variables
  - apply classes in the Java API (Application Programming Interface)
  - differentiate between instance and static variables

  - develop methods in classes
  - store and process objects in arrays
  - apply class abstraction to develop software

- Inheritance and Polymorphism Students will be expected to:
  - develop a subclass from a superclass through inheritance
  - apply the polymorphism concept to handle different data types using a uniform interface

- Abstract Classes and Interfaces Students will be expected to:
  - identify the similarities and differences between an abstract class and an interface
  - model weak inheritance relationships with interfaces
  - specify a natural order using the Comparable interface
  - to wrap primitive data values into objects
  - create a generic sort method
  - simplify programming using JDK 1.5 automatic conversion between primitive types and wrapper class types

- Exceptions and Assertions Students will be expected to:
  - distinguish exception types: Error versus Exception in Java
  - throw an exception in a method
  - write an exception handler using a try-catch-finally block
  - explain the propagation of an exception
  - apply assertions to help ensure program correctness

- Text I/O Students will be expected to:
  - Read and write characters using the InputStreamReader, FileReader, BufferedReader, OutputStreamWriter, FileWriter, PrintWriter, BufferedWriter classes
  - Be able to apply the appropriate class in text I/O operations based on the requirements and performance needs
  - Distinguish between text I/O and binary I/O

- Object-Oriented Design Students will be expected to:
  - become familiar with the software development process
  - model a system with the appropriate relationships: association, aggregation, composition, dependency, strong inheritance, and weak inheritance

  - declare classes to represent the relationships among them.
  - design systems by identifying the classes and discovering the relationships among these classes

- Unit testing with JUnit Students will be expected to:
  - Create test classes, test methods, and run tests with JUnit
  - Create and use test fixtures in JUnit
  - Interpret test results with JUnit
  - Correlate the test fixtures with assertions
  - Verify that a software unit performs as specified
  - GUI and Graphics Students will be expected to:
  - Describe the Java GUI hierarchy
  - Create user interfaces using frames, panels, and simple GUI components
  - Apply layout managers
  - Use JPanel as subcontainers
  - Draw figures using the methods in the Graphics class
  - Override the paintComponent method to draw figures on a GUI component
  - Introduction to Threads creation, simple usage

- Event Driven Programming Students will be expected to:
  - declare listener classes and write event handlers to handle events
  - apply the Observer Pattern to decoupled programs
  - register listener objects in the source object
  - create inner classes and anonymous inner classes
  - write programs to handle ActionEvent, MouseEvent, KeyEvent, and Timer event

- Recursion Students will be expected to:
  - solve problems with recursion
  - write programs using recursion
  - explain the difference between iteration and recursion

- Generic Types Students will be expected to:
  - improve reliability and readability of Java programs by using generic types

- Java Collections Framework Students will be expected to:

- describe the Java Collections Framework hierarchy
- utilize the common methods in the Collection interface for operating sets and lists
- use the Iterator interface to traverse a collection
- examine the Set interface and be capable of deciding when to use HashSet, LinkedHashSet,or TreeSet to store elements
- compare elements using the Comparator interface
- examine the List interface, and be capable of deciding how and when to use ArrayList or LinkedList to store elements
- examine the Collection and Map and be capable of deciding how and when to use HashMap, LinkedHashMap, and TreeMap to store values associated with keys.

Academic Dishonesty

Academic dishonesty of any kind will not be tolerated. All assignments, mini-assignments and exams are to be completed individually unless otherwise specified. While high-level discussion of ideas and problems with fellow students is encouraged, all work must be done individually. In certain circumstances, code fragments from the instructor may be provided to eliminate tedious coding or to provide a common framework for all students. **All other code must be original**. Online resources may be used to help you understand the material, but you may not copy online code nor can you "borrow" code from other students, past or present. If you use a book, website or any reference to help you solve a problem, you must cite the reference in your assignment.

Any suspected academic dishonesty will be dealt with on a case-by-case basis. Any clarification of what does or does not constitute academic dishonesty must take place **before** you turn in questionable work. For clarification on what constitutes academic dishonesty, contact me or consult the printed policy in the UWO Student Discipline Code, Chapter UWS 14.