Syllabus, revised May 5, 2014

CS3331: Advanced Object-Oriented Programming

Spring 2014

Instructor: Nigel Ward, nigel@utep.edu
Computer Science 3.0408, 747-6827
Office Hours: Wednesdays 2:30-3:30, or by appointment,
and generally whenever the door is open

TA: TBD
Office Hours: TBD in CCS G.0512B, and by appointment

Class Time: MW 10:30-11:50 in CCS G.0208

Textbook *Object-Oriented Software Development Using Java*. Xiaoping Jia.
Addison Wesley, 2003, ISBN 0-201-73733-7. (required; bring to class every day)

Recommended Books

Head First Object-Oriented Analysis and Design. Brett D. McLaughlin, Gary Pollice, and Dave West. O'Reilly 2006.

Head First Design Patterns. Eric Freeman and Elizabeth Freeman. O'Reilly 2004.
The Object-Oriented Thought Process, 3rd edition Matt Weisfeld. Addison Wesley 2009.
The Elements of Java Style. Allan Vermeulen, et al. Cambridge University Press, 2000.

Course Description: Advanced design and programming techniques in the object-oriented programming paradigm.

Goals: Learn the knowledge and skills needed to develop reusable, quality programs; increase knowledge of object-oriented design concepts; learn the use of object-oriented design tools such as UML for modeling problem solutions and complex systems; increase proficiency in programming in object-oriented and procedural environments.

Course Policies

The prerequisite for this class is CS2302 with a C or better.

Assigned readings are to be done before class.

Assignments are to be submitted in hardcopy in class, unless otherwise specified. After a 1 minute grace period at the start of class, assignments will be considered late. Late assignments will be accepted at the end of class or before or after any subsequent class session, and will be penalized at least 10% per day or partial day of lateness, for up to five days. Depending on the circumstances the penalty may be higher, for example, if an assignment is received after the solution has been discussed in class. For some assignments the code will also need to be uploaded to the class folder. Email submissions of assignments are not accepted.

Assignments are to be done individually unless specifically designated as group assignments. While you may discuss assignments with others, your solutions should be designed, written, and tested by

you alone. If you need help, consult the TA or the instructor.

All code should be your own, unless the assignment explicitly permits the use of borrowed code or found code; in such cases you must acknowledge your sources and state specifically what you used.

Programming assignments will be graded primarily on functionally, design quality, thoroughness of testing, and readability. In addition style, interface usability and other factors will be considered when appropriate. Some of these factors inevitably involve subjective judgments; if you have questions about these or any other aspect of the grading, please see the TA or the instructor

Tests will be closed-book, except that one single-sided page of hand-written notes may be brought in for the first test, two for the second test, and three for the final. If you leave the classroom for any reason, your test will be graded on only what you did up until that time.

Grades will be based on two components: the assignments and everything else, with good performance on both components needed for a good grade. Specifically, the grade will depend on a weighted average of the two component scores, where the weight for the stronger score is half that of the weaker score. The "everything else" score will come 40% from the final, 25% from each of the two tests, and 10% from other factors, including quizzes, in-class exercises, and general participation.

Assignments and tests will be challenging. Grading will be on a points-earned basis (points above zero), rather than a points-off basis. Letter grades will be assigned accordingly; in the past, the A/B break has been around 80% and the B/C break around 70%.

General Policies

Students are expected to be punctual, and, as always, to conduct themselves professionally and courteously.

If you have or suspect a disability and need accommodation you should contact CASS at 747-5148 or dss@utep.edu or visit Room 106 Union East Building.

No make-up exams or assignments will be given except under the conditions set forth in the Catalog.

Students are free to attend class or not, bearing in mind that absence may annoy other students, interfere with learning, and result in a lower grade.

Important Dates

January 22: Class begins
February 20: Test 1
March 10-14: Spring Break
April 3: Test 2
May 15,1:00-3:45: Final Exam

Course Website: http://www.cs.utep.edu/nigel/oo/

Likely Topics, Readings and Assignments (chosen to provide interwoven coverage of four themes: modeling and design, development methods, advanced Java, hands-on learning.)

Object-Oriented Modeling with UML (Chapters 1 & 2) (5 days)
- Assignment A: Java Warm-Up (adjacency checker) [2]
- Assignment B: code reuse, enumerations, UML class diagrams [1]
- Assignment C: Debugging [1]

Advanced Java Features (Chapters 3, 4 & 5) (2 days)
- Packages, Exceptions, Javadoc
- Assignment D: Code Reuse, Raising and handling exceptions []

Unit Testing with JUnit (Chapter 6) (1 day)
- Assignment E: Writing Unit Tests [6]
- Assignment F: Test-Driven Development

Design Principles and Class-Level Design (Chapters 5 & 6) (3 days)

Learning from Experience (4 days)
- Assignment G1: Play Battleships [1]
- Assignment G2: Initial Battleships Player [1]
- Assignment H: Revised battleships player, 1st tournament [2-20]
- Assignment I: Improved player, 2nd tournament [2-20]

Applets (Chapters 3, 4, 5, 7, 8) (2 days)
- Assignment K: An Applet (pairs) [2]

Design Patterns (Chapter 7 & 10) (5 days)

Personal Software Process (1 day)
- Assignment P: Monitoring Time Spend and Bugs Created [2]

GUI Programming (Chapter 8) (1 day)
- Assignment L: A Graphical User Interface (pairs) [5]

Concurrent Programming (Chapter 11)
- Assignment M: Queuing Simulation [2] (1 day)

Network Programming (Chapter 12) (1 day?)

Review
- Assignment O: Design Principles Revisited [2]
- Assignment N: A Question for the Final Exam [1] (1 day)
- Assignment Q: Evaluate the Course [0]

Learning Outcomes

Level 1: Knowledge and Comprehension (Level 1 outcomes are those in which the student has been exposed to the terms and concepts at a basic level and can supply basic definitions.) The material has been presented only at a superficial level. Upon successful completion of this course, students will be able to:

1a. Explain the differences between an object-oriented approach and a procedural approach.

Level 2: Application and Analysis (Level 2 outcomes are those in which the student can apply the material in familiar situations, e.g., can work a problem of familiar structure with minor changes in the details.) Upon successful completion of this course, students will be able to:

2a. Formulate and specify user requirements of a software system using use case diagrams and scenarios.
2b. Use object-oriented design tools such as CRC and UML class diagrams to model problem solutions.
2c. Use basic object-oriented design patterns to structure solutions to software design problems.
2d. Implement association relationships and multiplicities.
2e. Use frameworks, classes, and methods from standard libraries in problem solutions.

Level 3: Synthesis and Evaluation (Level 3 outcomes are those in which the students can apply the material in new situations. This is the highest level of mastery.) Upon successful completion of this course, students will be able to:

3a. Design and implement software employing the principles of modularity, encapsulation, information hiding, abstraction, and polymorphism.
3b. Design, implement, and use classes and methods in an object-oriented programming language, employing standard naming conventions and making appropriate use of advanced features such as inheritance, exception handling, I/O, references, and simple GUIs.
3c. Evaluate existing classes and software for the purposes of extension through inheritance.
3d. Use and create standard API documents to understand and document the use of classes and methods.
3e. Design and implement test suites for unit testing.
3f. Re-factor existing software to improve its design or efficiency.