COMP 2150 CS 2: Object-Oriented Programming and Data Structures – Spring 2016
Mr. Kriangsiri (“Top”) Malasri

Contact Information:

<table>
<thead>
<tr>
<th>Office:</th>
<th>Dunn Hall 396</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone:</td>
<td>901.678.5689</td>
</tr>
<tr>
<td>Email:</td>
<td><a href="mailto:kmalasri@memphis.edu">kmalasri@memphis.edu</a></td>
</tr>
</tbody>
</table>

Office: Dunn Hall 396  
Department Office: Dunn Hall 375  
Phone: 901.678.5689  
Department Phone: 901.678.5465

TAs/Graders: Hari Charan Chekati, hchekati@memphis.edu; Ruthvik Gopidi, rgopidi@memphis.edu (for honors section)

The best way to get in touch with me is through email – I will almost always respond within 24 hours.

Office Hours:
No formal hours, but I’m usually around on weekday afternoons. Feel free to email or call to set up an appointment!

Lecture Meeting Times/Locations:
MW 5:30-7:30 pm Dunn Hall 351

Catalog Description:
COMP 2150 – CS 2: Object-Oriented Programming and Data Structures (4) Principles of object-oriented programming and software development; problem solving with recursion and abstract data types, including linked lists, stacks, queues, binary search trees, hash tables; basic GUIs. PREREQUISITE: MATH 1910 or MATH 1421 (or MATH 1830 for COMP minors) and COMP 1900. COREQUISITE: COMP 2700.

Student Learning Outcomes:
This course focuses on the following ABET student outcomes:
(b) An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution.
(c) An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
(i) An ability to use current techniques, skills, and tools necessary for computing practice.
(k) An ability to apply design and development principles in the construction of software systems of varying complexity.

Course Website:
Lecture notes, code that we write in class, assignments, and grades will be posted to the eCourseware system: https://elearn.memphis.edu

Required Text:

Evaluation:

<table>
<thead>
<tr>
<th>Component</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>300 pts. (best 12 of 13 @ 25 pts. each)</td>
</tr>
<tr>
<td>Programming Projects</td>
<td>225 pts. (3 @ 75 pts. each)</td>
</tr>
<tr>
<td>Quizzes</td>
<td>100 pts. (2 @ 50 pts. each)</td>
</tr>
<tr>
<td>Midterm Exam</td>
<td>100 pts.</td>
</tr>
<tr>
<td>Final Exam (Comprehensive)</td>
<td>300 pts.</td>
</tr>
</tbody>
</table>

Final grade: add up your point total and divide by 1000. Note that the highest possible percentage grade is 102.5% since the points add up to 1025.

Grading Scale: Letter grades will be determined as follows:

- A+: 96-100%; A: 90-95%
- B+: 87-89%; B: 81-86%; B-: 79-80%
- C+: 77-78%; C: 71-76%; C-: 69-70%
- D+: 67-68%; D: 60-66%
- F: Below 60%
**Attendance:**
Attendance doesn’t officially count towards your final grade, but it’s crucial that you attend class regularly to keep up with the material!

**Homework and Programming Projects:**
Although COMP 2150 does not have a dedicated lab time like COMP 1900, homework will be given most weeks to reinforce the concepts covered in lecture. In addition to the homework, there will be three larger programming projects throughout the semester. For both homework and programming assignments, it is **MANDATORY** that your submission successfully compiles and runs. A submission that does not compile/run will receive zero credit.

**Late/Makeup Policy:**
All assignments are expected to be completed and turned in on schedule. Due dates will be clearly indicated for each assignment. Late assignments are **NOT** accepted except in extreme circumstances. Likewise, makeup quizzes and exams will be given only under extreme circumstances. If you feel that your circumstances warrant a late work submission or a makeup quiz/exam, get in touch with me as soon as possible. Be prepared to show some kind of documented proof of your situation.

**eCourseware Dropbox Policy:**
All code submissions should be made through the dropbox on eCourseware unless specifically indicated otherwise. The dropbox will automatically cut off submissions precisely at the deadline. It is your responsibility to submit your work with time to spare, and to double check that your submission made it into the dropbox. “I accidentally submitted the wrong file,” “The dropbox was having technical issues at the last minute,” “I submitted the file but somehow it never made it to the dropbox,” “The dropbox wouldn’t accept my submission because it was 3 seconds late,” and similar statements are **NOT** valid excuses.

**Email:**
Please check your [University of Memphis](http://www.memphis.edu) email account at least once a day, as that is my primary means of communicating with you outside of class.

**Plagiarism/Cheating Policy:**
An essential part of learning how to program is getting plenty of practice with it yourself. As such, all assignments for this class (unless specifically indicated otherwise) are expected to be individual efforts. If I determine that you have copied something directly from a book, the Internet, or some other source, you will receive a failing grade on the assignment and (at my discretion) a failing grade in the course. If I determine that you have copied another student’s assignment, this will happen to both you and the person from whom you copied. The incident may also be forwarded to the University Judicial Affairs Office for further disciplinary action. Please don’t put me in this situation.

**Getting Help:**
Although I expect your work for this class to be done individually, I encourage you to seek help if you get stuck:

- **Come talk to me!** I’m very willing to sit down and try to provide hints without giving away the solution.
- **The Computer Science Learning Center (Dunn Hall 208)** will be open throughout the semester. Hours will be posted on the door, as well as online at [http://www.memphis.edu/cs/current_students/cslc.php](http://www.memphis.edu/cs/current_students/cslc.php). The lab will be staffed by junior- and senior-level computer science students whom you can ask for help.

**Student Disabilities:**
If you have a disability that may require assistance or accommodations, or if you have any questions related to any accommodation for testing, note taking, reading, etc., please speak with me as soon as possible. You must contact Disability Resources for Students ([http://www.memphis.edu/drs](http://www.memphis.edu/drs)) to officially request such accommodations / services.
## Tentative Course Schedule:

<table>
<thead>
<tr>
<th>Date</th>
<th>Lecture Material</th>
<th>Text</th>
<th>Quizzes/Exams</th>
<th>Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/20</td>
<td>Course introduction / review of COMP 1900</td>
<td>Notes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/25</td>
<td>Built-in Java classes and the Java API</td>
<td>Notes</td>
<td></td>
<td>HW 1: Built-in classes</td>
</tr>
<tr>
<td>1/27</td>
<td>Writing custom classes</td>
<td>Notes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2/01</td>
<td>Writing custom classes (cont’d.)</td>
<td>Notes</td>
<td></td>
<td>HW 2: Writing custom classes</td>
</tr>
<tr>
<td>2/03</td>
<td></td>
<td></td>
<td></td>
<td>Project 1</td>
</tr>
<tr>
<td>2/08</td>
<td>Inheritance</td>
<td>Ch. 1</td>
<td>Quiz 1 (2/08)</td>
<td>HW 3: Inheritance</td>
</tr>
<tr>
<td>2/10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2/15</td>
<td>Interfaces and polymorphism</td>
<td>Ch. 1</td>
<td></td>
<td>HW 4: Polymorphism</td>
</tr>
<tr>
<td>2/17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2/22</td>
<td>Exception handling and file I/O</td>
<td>Ch. 1</td>
<td></td>
<td>HW 5: File I/O</td>
</tr>
<tr>
<td>2/24</td>
<td>Introduction to Big-O</td>
<td>Ch. 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2/29</td>
<td>Lists and iterators</td>
<td>Ch. 2</td>
<td>Midterm (3/02)</td>
<td>HW 6: Array lists</td>
</tr>
<tr>
<td>3/02</td>
<td></td>
<td></td>
<td></td>
<td>Project 2</td>
</tr>
<tr>
<td>3/07</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/09</td>
<td><strong>NO CLASS – Spring Break</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/14</td>
<td>Lists and iterators</td>
<td>Ch. 2</td>
<td></td>
<td>HW 7: Linked lists</td>
</tr>
<tr>
<td>3/16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/21</td>
<td>Stacks and queues</td>
<td>Ch. 3-4</td>
<td></td>
<td>HW 8: Stacks and queues</td>
</tr>
<tr>
<td>3/23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/28</td>
<td>Recursion</td>
<td>Ch. 5</td>
<td>Quiz 2 (3/28)</td>
<td>HW 9: Recursion</td>
</tr>
<tr>
<td>3/30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4/04</td>
<td>Introduction to GUIs</td>
<td>Notes</td>
<td></td>
<td>HW 10: GUIs</td>
</tr>
<tr>
<td>4/06</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4/11</td>
<td>Binary search trees</td>
<td>Ch. 6</td>
<td></td>
<td>HW 11: BSTs</td>
</tr>
<tr>
<td>4/13</td>
<td></td>
<td></td>
<td></td>
<td>Project 3</td>
</tr>
<tr>
<td>4/18</td>
<td>Heaps and priority queues</td>
<td>Ch. 6</td>
<td></td>
<td>HW 12: Priority queues</td>
</tr>
<tr>
<td>4/20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4/25</td>
<td>Sets, maps, hash tables</td>
<td>Ch. 7</td>
<td></td>
<td>HW 13: Hash tables</td>
</tr>
<tr>
<td>4/27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FINAL EXAM: Wednesday, May 4, 5:30-7:30 pm**
(same classroom as lecture)

Tentative Quiz and Exam Topics:

- Quiz 1: String manipulation, classes and objects
- Midterm Exam: Object-oriented programming concepts
- Quiz 2: Linear data structures (lists, stacks, queues)
- Final Exam: Everything!
STUFF YOU SHOULD KNOW BY NOW! (or should learn immediately if you don’t 😊)

This page summarizes what was covered in COMP 1900.

- Installing the Java Development Kit (JDK) and an IDE of your choice (I’ll be using BlueJ in class) onto your computer
- Structure of a basic Java program
  - Always starts with the class (program) name, followed by a main method that contains all the steps that you want the program to perform. You can have other methods besides main too, but whatever’s inside main is what will be executed when the program runs.
  - Standard Java conventions for ClassNames, variableAndMethodNames, CONSTANT_NAMES
  - Single and multi-line comments (//, /* */)  
- Java expressions and how they’re evaluated
  - Order of operations in numerical expressions (parentheses, multiplication/division/modulo, addition/subtraction)
  - Order of operations in Boolean expressions (parentheses, !, &&, ||)
  - Constructing Boolean expressions using relational operators (<, >, <=, >=, ==, !=)
  - Integer division and modulo
  - How expressions work when different data types are mixed (such as int and double, or String and anything else)
- Variables
  - Java’s primitive data types (byte, short, int, long, float, double, boolean, char) and what each can hold
  - Declaring a variable
  - Assigning a value to a variable
  - Declaring and working with constants
  - Shorthand notations for changing variable values (+=, -=, *=, /=, %=, ++, --)
  - Implicit and explicit casting
  - Variable scope: a variable exists and is usable only within the block where it’s declared. Examples: a variable declared within a method body exists only within that method; a variable declared within a loop exists only within that loop.
- Program input/output
  - Using a Scanner object to read information from the user as your program is running
  - Remember that before you can use Scanner, you need to 1) include the import statement at the top of your program, and 2) create a Scanner object within the method where you want to read information
  - Displaying stuff on the screen using System.out.println and its variants
- Conditionals
  - if: execute a segment of code if a condition is true
  - if-else: execute one of two possible branches depending on whether a condition is true
  - if-else if: execute one of multiple possible branches depending on a set of conditions. At most one branch can execute. An optional else may be added to the end to provide code to execute if none of the provided conditions are true.
  - switch: allows you to test the value of an expression and execute code based on different case values. Case values must be integers. Remember that more than one case may execute if you don't have a break statement. An optional default case may be added to the end to provide code to execute if none of the provided cases apply.
- Loops
  - while: repeatedly execute a segment of code as long as the provided condition is true. Condition is checked at the beginning of the loop, so the body of a while loop may not execute at all if the condition is initially false.
  - do-while: repeatedly execute a segment of code as long as the provided condition is true. Condition is checked at the end of the loop, so the body of a do-while loop is guaranteed to execute at least once.
  - for: consists of initialization, termination, and increment parts. Remember that this is really just a concise way of writing a while loop – you can rewrite one type as the other very easily! You usually use for loops when working with arrays, and in other situations where you know exactly how many iterations the loop needs to go through.
  - Infinite loops
- Methods – a method is a block of code that performs some specific task. Useful for organizing a complex program into more manageable “chunks.” Once you define a method, you can use (a.k.a. call, invoke) it as many times as you want!
  - Terminology: parameters/arguments (method inputs), return value (method output)
  - Parameter passing: Java uses pass-by-value. Basically this means that the parameters specified in the method header (formal parameters) and the actual values you use when you call the method (arguments or actual parameters) are stored in two different memory locations. When you call a method, the values of the actual parameters are copied over to the formal parameters, and the method performs its actions using the formal parameters. Hence, the actual parameters themselves can never be altered by a method!
  - Using built-in Math methods such as Math.random, Math.sqrt, Math.pow
  - Writing and calling your own methods to perform specific actions
- **Method overloading** – defining two or more methods with the same name. The methods MUST differ in the number and/or type of parameters. (Why can’t they differ only in return type?)
- **Recursive methods** – these are methods that call themselves. Get evaluated by means of your computer’s call stack. Simple problems that lend themselves well to recursive solutions: computing the factorial of a number, computing powers, finding Fibonacci numbers.

- **Arrays** – an array is just a collection of data of the same type. Each element in an array is associated with a numerical index. Indices start counting from 0 and go up to 1 less than the total number of elements in the array. In other words, an array of length $n$ has indices from 0 to $n-1$, inclusive.
  - Declaring and instantiating an array
  - Using `.length` to get an array’s length (number of elements)
  - Remember that array variables are *references*, which are fundamentally different from primitive variables. Primitive variables store information directly. References store memory addresses where information is kept (you can think of this as “pointing to” a memory address). For you C/C++ folks – a Java reference is more or less the same idea as a C/C++ pointer.
  - Using `=` and `==` with array variables – what do they mean?
- **Doing things to individual array elements** – this usually involves a loop that performs the same action(s) at each array index
- **How arrays work with methods** – arrays as parameters, arrays as return values. Remember that what gets passed to/from methods are actually references to the arrays and not the array elements themselves.
- **Working with 2-D arrays** – these are really just arrays of arrays (or more precisely, arrays of references to arrays)
- **Basic array algorithms**: linear search, binary search, insertion sort

- **Fundamentals of object-oriented programming (OOP)**
  - In OOP, software is developed as a collection of software *objects* that interact with one another, as opposed to a single step-by-step sequence of instructions
  - Every object is created from a description called a *class*. A single class can be used to create as many objects as desired. Objects created from a class are known as *instances* of that class, and creating an object from a class is known as *instantiating* the class.
  - A class consists of *attributes* (a.k.a. *instance variables*), which are characteristics or qualities of the object, and *methods*, which are actions or behaviors. Different objects created from the same class can (and often do) have different values for their instance variables, but they all share access to the same methods.
  - Syntax for creating an instance of a class, and calling methods using that instance (think about how you use the *Scanner* class)