COMP 2150 CS 2: Object-Oriented Programming and Data Structure – Fall 2018
Dr. James Yu

Contact Information:

Office: Dunn Hall 320	**Department Office:** Dunn Hall 375
Office Phone: 901.678.3712	**Department Phone:** 901.678.5465
Email: jyu8@memphis.edu	
TA/Graders: Ms Sambriddhi Mainali (smainali@memphis.edu)	

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

Office Hours:

No formal office hours, I am usually around in the morning (M, W, F). You can drop by any time or email me to set up an appointment in advance.

Lecture Meeting Times/Locations:

83483 – COMP 2510 – 001 MW 5:30pm to 7:30 pm Dunn Hall 233
83484 – COMP 2510 – 002 TR 11:20am to 1:20pm Engr Science Bldg 308

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 and performance indicators:

(1) An ability to analyze a problem, identify and define the computing requirements appropriate to its solution. (Performance indicator: Demonstrate an ability to break down a problem into smaller components.)
(2) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices. (Performance indicator: Demonstrate an ability to evaluate the benefits and tradeoffs of different data structures.)

Course Website:

You can find the course materials (lecture notes, assignments, code written in class, grades, etc.) at the University of Memphis eLearn system at https://elearn.memphis.edu.

Required Text:

- Zybooks: Object Oriented programming and the Essentials of Data Structures.
  - Sign in or create an account at up @ http://www.zybooks.com
  - Enter zyBook code: MEMPHIS2150YuFall2018
  - Click Subscribe
    You will need this book for the in-class exercises and assignments

Highly recommended textbook: Introduction to JAVA programming (*Comprehensive Version*) of the 10th/*11th Edition* by Y. Daniel Liang; Pearson, 2018. ISBN-10: 0134694511. This book will be used as a supplementary textbook for the course.

Evaluation:

Items	Points
Assignments (8-10)	225
ZyBooks-Exerc (~10)	125
Proj (1)	100
Quiz (4)	200
Midterm (1)	150
Final	250
Total	1050

Final grade: add up your point total and divide by 1000. Note that the highest possible percentage grade is 105% since the points add up to 1050. *You must pass the final exam (>50%) as well as the overall grade (>= 60%) to pass this course. If you score less than 50% in your final exam, you will receive a "F" grade independent of the resultant grades from the other attributes (Assignments, ZyBooks-exercises, quizzes, project, and Midterm)

Grading Scale: Letter grades will be determined as follows:

A+: 96% and above; **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%

Homework Assignments (225 pts), ZyBooks Exercises (125 pts) and Programming Project: (100 pts)

Comp2150 does not have a dedicated lab section as that of COMP1900. There are ~10 homework assignments given almost weekly to reinforce the concepts discussed in lecture. Some of the assignments may be given from the zyBook exercises. You will be doing most of these exercises after our lectures in class. In addition to the assignments/exercises, there will be one **programming project** provided around the fall break in early October. You should have plenty of time to work on the project before the due date at the end of the term. It is important to start working on the project as early as possible to avoid the last minutes scrambling and produces poor quality software. It is MANDATORY that your project submissions successfully compile and run. A project submission that does not compile/run will receive zero credit.

Attendance / participation: (part of zyBooks exercises)

It is crucial that you attend class regularly. The class will keep building on itself and moves at a fairly brisk pace, so you need to get a good handle on each concept soon after we discuss it. You can accomplish that by completing the assigned zyBook exercise, after almost every lecture. You are required to bring your laptop with the assigned zyBooks textbook installed for this course. You will need to submit a time-limited (closed to the end of the lecture) zyBook question to show your presence in class. If you are missing from the class, you will receive no marks for the zyBooks exercise of that lecture.

Email:

Please check your University of Memphis email (or the email address you provided) regularly (daily), as that is my primary means of communicating with you outside of class.

Late/Makeup Policy:

All assignments (including zyBooks exercises) are expected to be completed and turned in on schedule. Each assignment will have specified due dates. Your TA/GA will not accept late assignments 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 documented proof of your situation.

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 Office of Student Conduct will also receive a copy of the incident 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:

- Talk to me! I'm very willing to sit down and try to provide hints without giving away the solution.
- Contact your lab TA.
- The Computer Science Learning Center (Dunn Hall 208) will be open throughout the semester. Hours will be posted on the door. The hours for this semester should also be posted online soon. You can find them by going to memphis.edu/cs/ and clicking on "Current Students" and then clicking on "Computer Science Learning Center." The lab will be staffed by friendly, knowledgeable 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 the Student Disability Services Office (678-2880) to request such accommodations/services officially.

Course Schedule: (Tentative)

	Date	Lecture Topics	Zybooks+ Text	Quiz	Assignments
M	27-Aug	introduction / COMP 1900 (review)	ZB 2-6+Ch9		HW 1: review COMP190
W	29-Aug	Class and Object: OO thinking	ZB 7. Ch 10		
M	**3-Sep**	**Labor Day (no class)**			
W	5-Sep	Inner Classes; Lamda Expressions	Notes ;Ch 15.4 - 15.6		HW 2: Classes and Obje
M	10-Sep	Inheritance (11.4.- 11.7)	ZB 10 .1 to 10.7; ch11.1 - 11.4	Quiz1	
W	12-Sep	Inheritance (11.7-11.15)	10.7 (Is-a versus has-a)		HW3: Inheritance
M	17-Sep	Polymorphism	ZB10.5		
W	19-Sep	Abstract and interfaces classes	ZB 11; ch 13		HW4 Polymorphism
M	24-Sep	Exception handling and file I/O	ZB13; Ch 11	Quiz2	
W	26-Sep	Binary I/O + working with file	ZB 9; Ch 17		HW5 File I/O
M	1-Oct	Recursion	ZB 12; Ch 18		
W	3-Oct	Generic and Test1 Review	ZB 14		
M	8-Oct	**Test 1**		Test 1	
W	10-Oct	Generics	ZB 14, 15		
M	15-Oct	Lamda expression and Functional Programming	Notes		HW6: Generics, List **Project 1**
W	17-Oct	Fall Break (Oct 13 - 16) (No class)			
M	22-Oct	Collections: Lists, Stacks, Queues	ZB 15; Ch 20		HW7: List, Stack, and Qu
W	24-Oct	Sets and maps	ZB15 (Collections); Ch 21	**Quiz 3**	
M	29-Oct	Implement: Lists, Stacks, Queues	ZB20; Ch 24		
W	5-Nov	Implement: Lists, Stacks, Queues	ZB20; Ch 24		
M	7-Nov	Algorithm efficient: Big-O	ZB19.1 to 19.4; ch 22		HW8: sorting
W	12-Nov	Sorting (Insertion, bubble, Merge, Quick)	ZB19.5 to 19.11; Ch 23	Quiz4	
M	14-Nov	Sorting (Heap, bucket, Radix)	ZB 19.5-19.11: Ch 23		HW9: TBD
W	19-Nov	Tree: Binary search trees (BST)	ZB 22 (tree), Ch 25		
M	**21-Nov**	**Thanksgiving (No-Class) (Nov 21-25)**			
W	26-Nov	Tree: Binary search trees (BST)	ZB 22 (tree), Ch 25		
M	28-Nov	Hashing; Hash tables	ZB 21; Ch29		HW10: TBD
W	3-Dec	Last Day of Class (final review)			
M	5-Dec	Final Exam (Dec 7 -13)		Final	

Final Exam: December 7 – 13. Exact details (date, time, location) will be announced when it is available.

STUFF YOU SHOULD KNOW BY NOW! (you should review your COMP1900 material within the first week of class)

This page summarizes the materials in COMP 1900.

- Installing the Java Development Kit (JDK) and an IDE of your choice (I'll be using BlueJ/Eclipse 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 the 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 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 using 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, 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)