CSC 216
Data Structures

Dr. Melissa Wiggins
MCC 306
(601) 925-3874
http://www.mc.edu/~mwiggins
mwiggins@mc.edu

COURSE CREDIT: 4 hrs. credit **PREREQUISITES:** CSC 116, with a C or better

OFFICE HOURS: MWF 9-10, TR 8-9, TW 1-3, F 11-12, other hours by appointment

TEXT: *Java Software Structures*, John Lewis & Joseph Chase
Course web site: http://cosmo.mc.edu/moodle - Computer Science - CSC216MW

OTHER MATERIALS: You ***MUST*** have a MCnet (Novell and GroupWise) account for this course for email and the ability to use the laboratory computers. You ***MUST*** check this email account (MCnet) ***every*** day. Failure to do so may result in missed assignments, etc. that may result in a lower grade in this course. A USB storage device is necessary for this course.

DESCRIPTION: Three hours of lecture and three hours of laboratory per week. An introduction to the basic concepts of data structures from a practical standpoint with an emphasis on the use of some of the abstractions necessary for structured program development. Topics include, software development tools, top-down design, algorithm analysis, encapsulation, and methods of implementation as well as the common data structures stacks, queues, lists, and trees. Sorting and searching techniques employing these data structures will also be examined.

RATIONALE: This course is required of all majors and minors in Computer Science and Computing and Information Systems as well as various other degree programs on campus.

LEARNING OBJECTIVES: After successfully passing this course, the student will be able to write well-designed object-oriented programs in the Java language that use and implement the common data structures and their extensions. The entire list of course objectives can be found at http://www.mc.edu/campus/users/mwiggins/csc216.pdf.

EVALUATION:

Exams: There will be a three exams worth 150 points. (450 points)
Assignments: There will be 5-8 programming/problem solving assignments worth a total of 150 points.
Lab Assigns: There will be a 12 laboratory assignments worth a total of 200 points. This grade will come from the quizzes that are given at the end of lab each week.
Final Exam: There will be a comprehensive final examination given at the time specified by the college. *Friday, December 7, 2007, 11:00a.m.-1:00p.m.* This examination will be worth 200 points.

Grading Scale:

895 - 1000 points	**A**	595 - 694 points	**D**
795 - 894 points	**B**	0 - 594 points	**F**
695 - 794 points	**C**		

CLASS ATTENDANCE: The student is expected to attend classes. Regulations for class attendance are given in the Class Schedule. Remember in a MWF class 12 absences is an automatic **F**. **Three tardies counts as one absence in this class. (See Mississippi College catalog). Laboratory attendance is very important!!** Four absences in lab will result in a grade of 0 for the lab portion. "For lesser numbers of absences, the student should expect a lowered grade in the course, with the maximum penalty of one letter grade for each week of absences (in a semester) or the equivalent. The calculation of the semester grade, including any penalty for absences, is the responsibility of the professor and may vary according to the nature of the course and the grading scale used. In some classes points will be deducted from the semester grade for unexcused absences; in others, the penalty may be built into the grading scale by means of frequent pop quizzes, grades for class participation and the like." Mississippi College Policy 2.10 Students should expect a penalty for absences as stated above.

MAKE-UP WORK & TESTS: Students are expected to take tests on the day they are assigned. However, it is the student's responsibility to contact the instructor in case of an emergency illness or death in the family **before** the test. ***At that time the student and instructor will agree on a time for the make-up exam.*** This time should be within 2 days of the missed test. Assignments are to be turned in on the day they are due!! All work is due at the beginning of the class period. Any work not turned in will lose 10% credit for each school day ***until the third day***. The due date at the beginning of class is day 1. No work will be accepted after the third day. Under no circumstances will work be accepted after the assignment has been graded and handed back in class. Laboratory work will be due at the end of each week's lab at which time a lab quiz will be administered. Exceptions to this may be made at the instructor's discretion.

ACADEMIC INTEGRITY: This statement on academic honesty in computer science courses is an addendum to the Mississippi College Policy 2.19. In a computer science class individual effort is expected. Student misconduct not only includes cheating on tests, but also extends to copying or collaborating on programming assignments, projects, lab work or research unless otherwise specified by the instructor. Using other people's accounts to do your work or having others do your work is prohibited. Close proximity in lab does not mean collaboration is permitted. NOTE: Discussing logical solutions to problems is acceptable, exchange of code, pseudocode, designs, or procuring solutions from the Web, other texts, the Internet or other resources on or off campus is not acceptable and will be treated as cheating. **REMEMBER: Any work submitted by the student which is not the student's own will be considered cheating.**
First offense: grade of 0 for **all** parties involved unless the "guilty" party can be determined as well as any punishment deemed necessary under policy 2.19
Second offense: grade of F in the course as well as any punishment deemed necessary under policy 2.19

SPECIAL ACCOMMODATIONS: If you need special accommodations due to learning, physical, psychological, or other disabilities please contact Dr. Buddy Wagner in the Counseling and Career Development Center. He may be reached by phone at (601)925-3354 or by mail at P.O. Box 4013, Clinton, MS 39058.

DROPPING A COURSE: **LAST DROP DATE - October 26**

Students cannot withdraw after this date with a W (passing) unless the three following criteria are met:
- Extenuating circumstances (clearly outside the student's control)
- Passing the course at the time of withdrawal
- Does not have excessive absences at the time of withdrawal

NOTE: Dropping after the THIRD (3rd) WEEK will result in a grade of W appearing on your permanent record (transcripts). See http://www.mc.edu/publications/policies/2.13.doc**.**

INCOMPLETE GRADES: An Incomplete may be given to a student who has been providentially hindered from completing work required in a course, provided that:
1. semester attendance requirements have been met;
2. most of the required work has been done;
3. the student is doing passing work; and
4. the student has made prior arrangements with the professor to complete the remaining work at a later date.

The grade of I must be removed promptly or it becomes an F; it cannot be removed by repeating the course.

TENTATIVE COURSE OUTLINE*

Day	Date	Course Topic & Reading Assignment	Other Course Assignments
1	Aug 22	Course Policies - Intro. to Course & Review	
2	24	Ch. 1 - Software Engineering	
3	27	Ch. 1	
4	29	Ch. 2 - Object-Oriented Design	
5	31	Ch. 2	
6	Sept 3	**No Class**	**Labor Day Holiday**
7	5	Ch. 2	
8	7	Ch. 3 - Collections	
9	10	Ch. 3	
10	12	Ch. 4 - Linked Structures	
11	14	Ch. 4	
12	17	Ch.6 - Stacks	
13	19	Ch. 6	
14	21	Ch. 6	
15	24	***Exam 1 - Chapters 1-5***	
16	26	Ch. 7 - Queues	
17	28	Ch. 7	
18	Oct 1	Ch. 7	
19	3	Ch. 8 - Lists	
20	5	Ch. 8	
21	8	**No Class**	**Fall Recess**
22	10	Ch. 8	
23	12	Ch. 10 - Recursion	
24	15	Ch. 10	
25	17	***Exam 2 - Chapters 6-9***	
26	19	Ch. 11 - Searching & Sorting	
27	22	Ch. 11	
28	24	Ch. 11	
29	26	Ch. 12 - Trees	*****Last Drop Date***
30	29	Ch. 12	
31	31	Ch. 13 - Binary Search Trees	
32	Nov 2	Ch. 13	
33	5	Ch. 13	
34	7	Ch. 15 - Heaps	
35	9	Ch. 15	
36	12	Ch. 15 / Ch. 17 - Hashing	

37	14	Ch. 17	
38	16	Ch. 17	
39	19	Ch. 16 - Multi-Way Search Trees	
40	21	**No Class**	**Thanksgiving Recess**
41	23	**No Class**	**Thanksgiving Recess**
42	26	Ch. 16	
43	28	***Exam 3 - Chapters 10-15, 17***	
44	30	Ch. 18 - Graphs	
45	Dec 3	Ch. 18	***Dead Days***
46	5	Wrap-Up & Review	***Dead Days***
47	7	***Comprehensive Final Examination***	***11:00 a.m.-1:00 p.m.***

**Instructor reserves the right to modify as necessary.*

Program Submission Guidelines

All programs must be submitted by email as an attachment. Source code must be submitted as well as all files necessary for the programs execution. The email message should contain the following information:

- ***Author's name***
- ***Date completed***
- ***Brief problem description***
- ***Statement regarding whether the program works or not.***
- ***If the program does not work, a brief but concise description of what is wrong and what it will take to "fix" it.***

CSC 216 LABORATORY SCHEDULE

LAB #	***WEEK OF***	***LAB CONTENT***
1	***Aug 27***	***Chapter 1 - Software Engineering***
2	***Sept 3***	***No Lab - Labor Day***
3	***Sept 10***	***Chapter 2 - Object-Oriented Design***
4	***Sept 17***	***Chapter 3 - Collections***
5	***Sept 24***	***Chapter 4 - Linked Structures***
6	***Oct 1***	***Chapter 6 - Stacks***
7	***Oct 8***	***No Lab - Fall Break***
8	***Oct 15***	***Chapter 7 - Queues***
9	***Oct 22***	***Chapter 8 - Lists***
10	***Oct 29***	***Chapter 10 - Recursion***
11	***Nov 5***	***Chapter 11 - Searching & Sorting***
12	***Nov 12***	***Chapter 12 - Trees***
13	***Nov 19***	***Chapter 13 - Binary Search Trees***
14	***Nov 26***	***Chapter 15 - Heaps***

LEARNING OBJECTIVES: *After successfully passing this course, the student will be able to write well-designed object-oriented programs in the Java language that use and implement the common data structures and their extensions.*

A. The Language of Efficiency

The student will be able to

1. *Know why data giving raw measurements of algorithm performance cannot be used to compare algorithms.*
2. *Understand how equations describing the resource consumption patterns of algorithms stay in the same family, even though computers, compilers, and languages may vary.*
3. *See how the dominant term of the running time equation accounts for most of an algorithm's running time for large problem sizes.*
4. *Develop intuition for the meaning of O-notation by exploring patterns in the same data.*
5. *Understand how O-notation is used by computer scientists to refer to the performance properties of algorithms.*
6. *Understand the formal definition of O-notation.*
7. *Use some useful shortcuts for manipulating O-notation.*
8. *Understand about circumstances in which O-notation analysis does not apply.*
9. *Know what to do to find optimal algorithms under these circumstances.*

B. Software Engineering Concepts

The student will be able to

1. *Design programs using object-oriented design (OOD).*
2. *Understand the difference between top-down design and bottom-up programming.*
3. *Understand the process of stepwise refinement.*
4. *Examine how to refine choices of data structures and algorithms in parallel.*
5. *Understand unit, integration, regression, and acceptance testing.*
6. *Use test drivers and stubs.*
7. *Understand the separate roles of testing and verification.*
8. *Understand the difference between top-down and bottom-up testing.*
9. *Appreciate the value of formatted debugging aids.*
10. *Understand what a test plan is and write test plans.*
11. *Use concepts of good program structuring.*
12. *Discuss some ideas that help produce good documentation.*

C. Introduction to Object-Oriented Programming

The student will be able to

1. *Define a subclass by extending a given class.*
2. *Override applet and interface methods to create a customized new drawing applet.*
3. *Develop programs starting with reusable software components.*
4. *Use abstract classes and abstract methods.*
5. *Understand inheritance and overriding in class hierarchies.*
6. *Use type polymorphism to advantage.*
7. *Draw into off-screen image to eliminate flashing.*
8. *Understand the flavor of OOP that relies on heavy use of prefabricated software components.*
9. *Understand the role of standard applet method invocations.*

D. Linked Data Representations

The student will be able to

1. *Understand what pointers are and how they can be used.*
2. *Appreciate the importance of pointers in real-world systems.*
3. *Understand how Java's reference values can represent pointers and links.*
4. *Know how to allocate object memory dynamically and obtain references to it.*
5. *Manipulate reference values in Java to create linked data representations.*
6. *Understand how to interpret the pointer diagrams used in this book.*
7. *Use a notation for picturing linked representations that is helpful when reasoning about them.*
8. *Understand how to represent and manipulate linked lists using Java objects and reference values.*
9. *Develop skill in designing and implementing linked list algorithms.*
10. *Understand and use a set of useful basic linked list operation.*

11. Understand some possibilities for linked data representations other than simple, one-way linked lists.
12. Illustrate how nodes with two links can be linked into representations such as two-way linked lists, circular lists, and trees.

E. *Recursion*

The student will be able to

1. *Think recursively.*
2. *Understand how strategies for recursion involve both base cases and recursion cases.*
3. *Search for different ways of decomposing a problem into subproblems.*
4. *Understand how to use call trees and traces to reason about how recursive programs work.*
5. *Understand infinite recursion.*
6. *Understand the symptoms caused by the occurrence of infinite recursion.*
7. *Understand that some recursive algorithms have exponential running times.*
8. *Understand why it is impractical to use exponential algorithms to solve problems of large size.*

F. *Modularity and Data Abstraction*

The student will be able to

1. *Identify an abstract data type (ADT) from an example.*
2. *Understand how it is possible to replace the underlying representation for an abstract data type without changing the operations it presents to its external users through its interface.*
3. *Use information hiding features of Java to hide the implementation details of an abstract data type's operations.*
4. *Change underlying data representations implementing an ADT without changing any of the code that uses the ADT.*
5. *Use Java interfaces to make plug-compatible replacement (Strings for integers, etc.) work.*
6. *Understand the role of Java interfaces and classes in implementing modular software.*
7. *Understand the general benefits of modularity and information hiding in the design of large software systems.*
8. *Understand the philosophy and terminology in general use concerning modularity and information hiding.*

G. *Linear Data Structures - Stacks and Queues*

The students will be able to

1. *Understand and use the common terminology of stacks and queues.*
2. *Understand how stacks can be used in dealing with recursion and in other applications.*
3. *Understand the view of ADTs as data structures + operations.*
4. *Understand about the abstract operations and interfaces for stack and queue ADTs.*
5. *Understand how stacks can be used to evaluate postfix expressions.*
6. *Understand how to implement stacks in two different but substitutable ways.*
7. *Understand how to separate implementation details form abstract operations in an ADT interface.*
8. *Understand features of sequential and linked representations.*
9. *Understand how queues can be used in simulations and modeling.*
10. *Understand how to implement queues in two different but substitutable ways.*
11. *Understand about print buffers and print spoolers and synchronization problems.*

H. *Lists, Strings, and Dynamic Memory*

The student will be able to

1. *Understand the advantages and disadvantages of sequential and lined representations of lists.*
2. *Understand one-way, two-way, and circular linked list representations.*
3. *Understand the role of list header nodes.*
4. *Understand that many varieties of list representations are possible.*
5. *Understand how generalized lists differ from simple linear lists.*
6. *Understand how to represent and manipulate generalized lists.*
7. *Understand the difference between shared and copied sublists.*
8. *Understand how the Java String and StringBuffer classes are used to manipulate a string's characters.*
9. *Understand the difference between static and dynamic memory allocation.*
10. *Understand about available space lists and garbage collection.*
11. *Understand how heaps can be used to support dynamic memory allocation.*
12. *Understand fragmentation, coalescing, handles, and heap compaction.*

I. Trees

The students will be able to

1. *Understand the terminology of trees and the relationships among nodes in a tree.*
2. *Understand the definition of binary trees, extended binary trees, and complete binary trees.*
3. *Understand how to represent a heap using a contiguous sequential representation.*
4. *Understand how heaps cam serve as efficient representations for priority queues.*
5. *Understand and be apply to apply the four types of traversal orders commonly applied to nodes of binary trees.*
6. *Understand how stacks and queues can be used to perform nonrecursive traversals of linked binary tree representations.*
7. *Understand how to search for a key in a binary search tree.*
8. *Understand what shapes binary search trees must take to yield the best and worst search performance.*
9. *Understand how binary search trees perform in the average case.*

J. Hashing & Searching

The student will be able to

1. *Understand intuitively hashing concepts using simple examples.*
2. *Be familiar with hash functions, collisions, collision resolution policies, open addressing, probe sequences, chaining, and buckets.*
3. *Understand the performance of hash algorithms.*
4. *Understand how the algorithms for hashing with open addressing work.*
5. *Understand the performance formulas characterizing how well hashing works.*
6. *Understand how to design a good hash function.*
7. *Trace the execution of a serial and binary searches.*

K. Sorting

The student will be able to

1. *Understanding how comparison trees are defined.*
2. *Understand how abstract priority queue sorting.*
3. *Understand how SelectionSort and HeapSort work.*
4. *Know the O-notation for the sorting times for SelectionSort and HeapSort.*
5. *Understand the divide-and-conquer theme, as applied to sorting methods.*
6. *Understand that MergeSort is an O(n log n) sorting method.*
7. *Understand how QuickSort works and that it runs in average time O(n log n) and worst case time* $O(n^2)$.
8. *Understand how the InsertionSort and InterchangeSort work and what their efficiencies are.*
9. *Know about the O(n) RadixSort.*
10. *Understand how the ShellSort and BubbleSort work and why the BubbleSort should be avoided in general.*
11. *Measure the performance of several of the sorting methods and be able to know when and when not to use various sorting methods.*