Lecture Contact Hours: 48-54; Homework Hours: 96-108; Total Student Learning Hours: 144-162
Laboratory Contact Hours: 48-54; Homework Hours: 0; Total Student Learning Hours: 48-54

CUYAMACA COLLEGE
COURSE OUTLINE OF RECORD

Computer Science 282 – Intermediate Java Programming and Fundamental Data Structures

3 hours lecture, 3 unit
3 hours laboratory, 1 unit
Total units: 4

Catalog Description
Continuation of CS 182. Implement and analyze a variety of data structures and the algorithms used with those data structures, and create abstract data types and learn how and when to utilize them. Fundamental data structures include multidimensional arrays, linked lists, stacks, queues, heaps, trees, and hash tables; learn when to use which of the available dynamic memory data structures. Tools for analyzing and predicting run time and memory usage are introduced, as is Big-O notation. A variety of sort algorithms are reviewed and analyzed for best, worst, and average case performance, and are compared with tree traversal algorithms. Develop increased sophistication in object-oriented basics such as inheritance, encapsulation, design of abstract data types and polymorphism, and will gain experience by working on larger programs and managing large, multi-programmer projects. Laboratory instruction includes program development and execution. Mobile and database applications will be introduced.

Prerequisite
"C" grade or higher or "Pass" in CS 182 or equivalent

Entrance Skills
Without the following skills, competencies and/or knowledge, students entering this course will be highly unlikely to succeed:
1) Write well-designed Java code to solve a variety of scientific problems as applets or standalone programs.
2) Select appropriate storage class characteristics for variables as well as methods.
3) Solve a variety of programming problems using sound structured, top-down, object-oriented design principles.
4) Effectively use inheritance to minimize reinvention of similar objects.
5) Effectively access standard Java library modules.
6) Recognize the difference between (and implications of) reference vs. primitive data types.
7) Design, create and utilize complex classes and objects.
8) Input and output data to and from standard devices and files.
9) Visualize array subscripting as memory offset calculations and recognize some of the implications.
10) Understand and effectively utilize fundamental program control structures.
11) Utilize the try-throw-catch approach to exception handling.
12) Understand the relative efficiency of the various sort and search algorithms.
13) Untangle poorly designed code.
14) Define the requirements for and program a functional mobile application.
15) Utilize basic database SQL commands to access and retrieve data.

Course Content

1) Tools for algorithm analysis and order (Big-O) notation – How to determine the most efficient algorithm for any given solution.
2) Dynamic vs. static memory allocation/management -The differences between ROM and RAM memory types and utilization of data storage
3) Abstract data types – Utilizing text, numeric, data structures and BLOB (Binary Large Object Binary files)
4) Stacks and queues – Defining the two types of data flow within a computer and they are used.
5) Ordered and linked lists – The definitions of Ordered and linked list and how they used to order data flow between data objects.
6) Trees – Data structures that having branching components for data relationships and various algorithms that utilized to navigate these structures
7) Priority queues and heaps – Determine data priority and how to efficiently move the data (Queue or Heap).
8) Recursion and divide and conquer – Methods for searching for data within a structure.

Course Objectives

Students will be able to:

1) Write, test and debug a recursive method and explain how the implementation is being managed using a stack. Determine where the program halts, the base case, and the general case.
2) Describe the logical differences between a stack and a queue; the behaviors and uses of hash tables, heaps and graphs; and the strengths and drawbacks of both the store “by reference” and store “by copy” approaches to implementing data structures.
3) Design and implement a program using standard linked list memory management, stack or queue data structures, tree data structures (including binary trees, binary search trees, and AVL trees), and coding references.
4) Utilizing a variety of existing search and sort algorithms, analyze their best, worst and average case performances.
5) Utilize algorithmic analysis to correctly identify the appropriate dynamic memory data structure for different programming applications.
6) Utilizing the “Big-O” notation method, evaluate an algorithm’s execution effectiveness.
7) Analyze the data structure code for a mobile application.

Method of Evaluation

A grading system will be established by the instructor and implemented uniformly. Grades will be based on demonstrated proficiency in subject matter determined by multiple measurements for evaluation, one of which must be essay exams, skills demonstration or, where appropriate, the symbol system.

1) Quizzes and exams that measure students’ ability to use programming terminology and explain technical concepts related to data structures and their use in Java.
2) Practical exams requiring students to trace programming code to predict outcomes.
3) Projects requiring students to write complete application programs demonstrating effective use of object-oriented principles, methodology and data structures.

Special Materials Required of Student

USB flash drive

Minimum Instructional Facilities

Computer lab with networked computers and software to create Java development environment

Method of Instruction

1) Lecture and demonstration

2) Hands-on practice
3) Assignments
4) Readings

Out-of-Class Assignments

1) Design, code and debug multiple Java programs and objects that demonstrate class objectives
2) Analyze assigned pre-coded Java programs and post analysis comments on class discussion board

Texts and References

1) Required (representative example): Goodrich, Tamassia & Goldwasser. *Data Structures and Algorithms*. 6th Edition, Wiley, 2020.
2) Supplemental: None

Exit Skills

Students having successfully completed this course exit with the following skills, competencies and/or knowledge:

1) Write, test and debug recursive methods.
2) Effectively utilize inheritance and polymorphism to minimize reinvention of code.
3) Design and create a program using standard linked list memory management.
4) Design and create a program implementing a stack or queue data structure.
5) Design and create a program implementing a tree data structure.
6) Understand the behaviors and uses of hash tables, heaps and graphs.
7) Utilize a variety of searching and sorting algorithms; analyze their best, worst and average case performances.
8) Identify which dynamic memory data structures are appropriate for a variety of different applications based on algorithmic analyses.
9) Write larger, more complex programs.
10) Develop sound software engineering skills.
11) Develop mobile applications.

Student Learning Outcomes

Upon successful completion of this course and given a scientific problem-based scenario, students will be able to:

1) Decompose problems and design program solutions using flowcharts, pseudocode, models, or other tools.
2) Properly code applications using the fundamental coding structures: sequence, selection, and loops.
3) Test and debug applications using debugging tools such as trace execution.