C S 1M: INTERMEDIATE ALGORITHM & DATA STRUCTURE METHODOLOGIES IN JAVA

Foothill College Course Outline of Record

Heading	Value
Effective Term:	Summer 2021
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Prerequisite:	C S 1A.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Non-GE
Transferable:	CSU/UC
Grade Type:	Letter Grade (Request for Pass/No Pass)
Repeatability:	Not Repeatable

Student Learning Outcomes

- A successful student will be able to use the Java environment to define the basic abstract data types (stacks, queues, lists) and iterators of those types to effectively manipulate the data in his or her program.
- The successful student will be able to analyze the time complexity of a variety of algorithms and data structure access techniques and choose the best algorithm and/or data structure for the project at hand.

Description

Systematic treatment of intermediate data structures, algorithm analysis and abstract data types in the Java programming language intended for Computer Science transfer majors. Coding topics include large program software engineering design, multi-dimensional arrays, string processing, primitives, compound types, and allocation of instance and static data. Concept topics include dynamic memory, inheritance, polymorphism, hierarchies, recursion, linked-lists, stacks, queues, trees and hash tables.

Course Objectives

The student will be able to:
A. Use instance members, static members, and dynamic data structure allocation as appropriate in object-oriented Java class design.
B. Analyze and demonstrate the use of multi-dimensional arrays in Java.
C. Design, implement, and test Java programs that use object-orientation and class inheritance as a key ingredient to good software design, and explain why sub-classing is an example of the "is-a" relationship.
D. Describe the difference between deep copies and shallow copies in Java, and write programs that effectively handle deep memory.
E. Demonstrate a working knowledge of data abstraction through various data types, data structures and their Java-based API classes, and choose the appropriate data structure to model a given problem.
F. Give examples of the proper use of recursion and describe when iterative solution is more efficient.
G. Explain what abstract classes and Java interfaces are and how they are used.
H. Describe declaration models for runtime storage allocation and garbage collection.
I. Use the Java Collections Framework to write efficient and portable application programs.
J. Define various types of Java generics and show how each is specialized to a class by the client program.
K. Design, implement, test, and debug intermediate-level Java programs that use each of the following fundamental programming constructs: user interaction and communication, string processing, numeric computation, simple I/O, arrays and the Java API.
L. Analyze the basic algorithms of a general tree ADT.
M. Use object-oriented programming (OOP) to create alternative implementations of binary search trees in Java, and verify or compare the logN behavior of each.
N. Analyze, classify and measure the main non-NlogN sorts and write a clear report of the results.
O. Write applications that solve problems in one or more application area: mathematics, physics, chemistry, cellular automata, 3-D simulation, astronomy, biology, business, internet.

Course Content

A. The proper use of class members and methods
1. Primitive vs. compound types
2. Types as collection of values (data members) and operations (method members)
3. Type checking and incompatibility in classes
4. When to use instance members and methods
5. When to use static members and methods
6. Implicit and explicit use of the "this" object
B. Multi-dimensional arrays
1. 2-D arrays
2. Ragged 2-D arrays
3. Instantiation of objects in multi-dimensional arrays
C. Inheritance in software design
1. The "is a" relationship
2. Base classes
3. Derived classes (subclasses) and class hierarchy
4. Derived class constructors
5. Member method overriding vs. simple overloading
6. Private, protected, public and default members
7. Encapsulation and polymorphism
8. Separation of behavior and implementation and other attributes of good software design
D. Deep vs. shallow copies of objects
1. Instantiation of member objects in constructors
2. Cloning objects
3. Deep copy cloning and its uses
4. Shallow copy cloning and its uses
E. Topics in Abstract Data Types (ADTs)
1. The vector ADT and Java's ArrayList
2. The linked-list ADT and Java's LinkedList
3. The Stack and Queue ADT
4. Implementing ADTs through inheritance
5. Using existing ADTs from java.util
F. Recursion
1. Base case vs. general case
2. Divide-and-conquer strategies
3. Analysis of recursive solutions which are not appropriate due to exponential time complexity vs. iterative polynomial complexity

4. Recursive backtracking
G. Abstract classes and interfaces
1. Defining and using abstract classes
2. Defining and using interfaces
3. Implementing interfaces and abstract classes
H. Storage allocation methods
1. Run time binding and storage management of activation records
2. Consequences of reference declarations and parameter-passing in Java and the relationship between Java references and the values/ pointer mechanism of other languages
3. Strong type-checking and run-time vs. compile time error detection
4. Effect of declaration strategy on binding, visibility and lifetime of variables
5. Effect of declaration strategy on scope and persistence of variables
I. Java Collections Frameworks
1. Java ArrayLists and ListIterators
2. Java LinkedLists and ListIterators
3. Java PriorityQueues
J. Java generics
1. Generic classes
2. Type parameters
3. The occasional need for wrapper classes
4. Autoboxing
5. Static generic methods
6. Type bounds
7. Wildcards
K. Essential examples and assignment areas
1. String/text processing
2. Numeric computation
3. User interaction
4. Multi-class projects and compound data types
5. Inheritance-based projects
L. General trees
1. Tree nodes, roots, leaves, children and siblings
2. Binary node implementation of a general tree
3. Insertion and deletion in general trees
4. Traversal with recursion
M. Searching and Binary Search Trees (BSTs)
1. Ordering condition and structure condition
2. OOP (object-oriented-programming) implementation
3. Time complexity consequence of the divide-and-conquer algorithm in of BSTs
N. NlogN sorts
1. Merge sort
2. Heap sort
3. Quicksort
O. Applications used throughout course in selected areas
1. Math
2. Physics
3. Chemistry
4. Biology
5. Astronomy
6. Business and finance
7. Internet

Lab Content

A. Exploring advanced array constructs in class design
1. Gain experience in effectively using single and multi-dimensional arrays as class members
2. Apply nested loops to process multi-dimensional arrays
3. Use the IDE to debug errors in multi-dimensional arrays
4. Solve problems using fixed-size and dynamic sized arrays, as appropriate
B. Building a program that uses class inheritance to demonstrate how re-use is handled in OOP
1. Create a project that contains at least one class intended to be used as a base class
2. Derive (sub-class) one or more classes from the base class
3. Use function chaining to avoid code duplication between base classes and derived classes
4. Differentiate between, and document in your lab, the distinct use of method overloading and method overriding
C. Incorporating basic queue and stack abstract ions in programming projects
1. Implement a fundamental queue and stack abstract data type (ADT) in a programming lab
2. Use a previously written ADT from the programming language's application programmer interface (API)
3. Incorporate inheritance in a project that uses ADTs
4. Provide a client program that tests and demonstrates the correct behavior of the ADT
D. Building projects that use generics (AKA templates)
1. Demonstrate the difference in a lab project between deriving from a base class and specializing a generic
2. Practice writing a generic as well as using a language-defined generic (template) in a project
3. Use generics to exercise some aspect of ADTs such as specializing a generic ADT to make its behavior specific to an assigned project specification
4. Employ debugging techniques to solve problems that arise when designing with generic (template) classes
E. Using linked-list ADTs to optimize for size-varying or space-sensitive data types
1. Demonstrate the ability to use programming language-supplied linked-list structures in a problem that is not easily solved using fixed-size ADTs such as arrays
2. Incorporate generics so as to allow the algorithm to work on various underlying data types
3. Try different sized data for the linked-list and demonstrate that it handles growth properly
F. Demonstrating competence with binary search trees
1. Implement a binary search tree (BST) from scratch, or make significant assigned adjustments to an existing BST data structure supplied by your instructor
2. Use recursion as appropriate for some of the BST methods
3. Demonstrate that the class works on various underlying base type by use of generic specialization
G. Incorporating hash tables into programs
1. Produce a lab that creates or modifies a hash table and hashing function
2. Write a client that tests out the hash table on various data
3. Using a large data set, demonstrate that near-constant time access is produced by the hashing function and hash table
4. Supply runs and report results with varying sized data sets
H. Analysis of a single sort algorithm
1. Implement a single sort algorithm as directed by the instructor
2. Experiment with coding adjustments to try to improve the performance
3. Compare the known time complexity of that algorithm with what you observe using increasingly larger data sets
4. Attempt to explain any discrepancies in the expected vs. observed growth rate of the sort algorithm
I. Analysis of multiple sort algorithms

1. Implement multiple sort algorithms, at least two of which involve Shell sort and quicksort
2. Experiment with coding adjustments to try to improve the performance on any one of them to see if you can beat the fastest of the algorithms
3. Time the algorithms on very small to very large data sets
4. Report on which algorithms work better on small sets, and which on large sets

Special Facilities and/or Equipment

A. Access to a computer laboratory with Java compilers.
B. Website or course management system with an assignment posting component (through which all lab assignments are to be submitted) and a forum component (where students can discuss course material and receive help from the instructor). This applies to all sections, including on-campus (i.e., face-to-face) offerings.
C. When taught via Foothill Global Access on the Internet, the college will provide a fully functional and maintained course management system through which the instructor and students can interact.
D. When taught via Foothill Global Access on the Internet, students must have currently existing email accounts and ongoing access to computers with internet capabilities.

Method(s) of Evaluation

Tests and quizzes
Written laboratory assignments which include source code, sample runs and documentation
Final examination

Method(s) of Instruction

Lectures which include motivation for syntax and use of the Java language and OOP concepts, example programs, and analysis of these programs
Online labs (for all sections, including those meeting face-to-face/on-campus), consisting of:
1. A programming assignment webpage located on a college-hosted course management system or other department-approved internet environment. Here, the students will review the specification of each programming assignment and submit their completed lab work
2. A discussion webpage located on a college-hosted course management system or other department-approved internet environment. Here, students can request assistance from the instructor and interact publicly with other class members
Detailed review of programming assignments which includes model solutions and specific comments on the student submissions
In-person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing and analyzing programs
When course is taught fully online:
1. Instructor-authored lecture materials, handouts, syllabus, assignments, tests, and other relevant course material will be delivered through a college-hosted course management system or other department-approved internet environment
2. Additional instructional guidelines for this course are listed in the attached addendum of CS department online practices

Representative Text(s) and Other Materials

Weiss, A. Mark Allen. Data Structures and Algorithm Analysis in Java, 4th ed.. 2013.

Herbert, Schildt. Java: A Beginner's Guide, 8th ed.. 2018.

The Weiss text is a classic text that is not out of date for the material taught.

Types and/or Examples of Required Reading, Writing, and Outside of Class Assignments

A. Reading
1. Textbook assigned reading averaging 30 pages per week.
2. Reading the supplied handouts and modules averaging 10 pages per week.
3. Reading online resources as directed by instructor though links pertinent to programming.
4. Reading library and reference material directed by instructor through course handouts.
B. Writing
1. Writing technical prose documentation that supports and describes the programs that are submitted for grades.

Discipline(s)

Computer Science