C S 1B: INTERMEDIATE SOFTWARE DESIGN 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.
Advisory:	Demonstrated proficiency in English by placement via multiple measures OR through an equivalent placement process OR completion of ESLL 125 & ESLL 249.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Area V: Communication & Analytical Thinking
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.
- A successful student will be able to define and use Java generics to make their data and algorithms work with a variety of data types.
- A successful student will be able to write and debug Java programs which make use of inheritance, i.e., the "is a" relationship, common to all OOP languages. Specifically, the student will define base and derived classes and use common techniques such as method chaining in his or her programs.

Description

Systematic treatment of intermediate concepts in computer science through the study of Java object-oriented programming (OOP). Coding topics include Java interfaces, class extension, generics, the Java collections framework, multi-dimensional arrays and file I/O. Concept topics include OOP project design, inheritance, polymorphism, method chaining, functional programming, linked-lists, FIFOs, LIFOs, event-driven programming and guarded code.

Course Objectives

The student will be able to:
A. Configure a Java Development Kit (JDK) and Integrated Development Environment (IDE) for advanced Java programming.
B. Use both instance members and static members, as appropriate, in class design.
C. Analyze and demonstrate the use of multi-dimensional arrays in Java.
D. Design, implement, and test Java programs that use class inheritance, and explain why this is an example of the "is-a" relationship.
E. Demonstrate the use of function chaining between derived class and base class methods.
F. Describe the difference between deep copies and shallow copies in Java, and write programs that effectively handle deep memory.
G. Explain how guarded code is implemented in Java through exceptions.
H. Express numbers in decimal, binary and hexadecimal representations and use bitwise logical operators to process data at the bit and byte level.
I. Identify an inner class and note its use in event-driven Graphical User Interface (GUI) code.
J. Demonstrate a working knowledge of basic abstract data types and their Java-based API classes.
K. Produce end-user programs which feature event-driven techniques that provide a sensible and easy-to-use GUI.
L. Explain what abstract classes and Java interfaces are and how they are used.
M. Describe declaration models for run-time storage allocation, garbage collection and type checking.
N. Use some of the Java Collections Framework collections to write efficient and portable application programs.
O. Define various types of Java generics and show how each is specialized to a class by the client program.
P. Write to and read from files using intermediate file I/O operations in a Java program.
Q. Design, implement, test, and debug intermediate-level Java programs that use each of the following fundamental programming constructs: string processing, numeric computation, simple I/O, arrays and the Java API.
R. 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. Setting up a complete Java environment
1. The JDK
2. Eclipse
3. Configuring the IDE for advanced Java programming
B. The proper use of class members and methods
1. When to use instance members and methods
2. When to use static members and methods
3. Implicit and explicit use of the "this" object
C. Multi-dimensional arrays
1. 2-D arrays
2. Ragged 2-D arrays
3. Instantiation of objects in multi-dimensional arrays
D. Inheritance
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
E. Function chaining
1. Constructor chaining
2. Member method chaining
F. 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
G. Exception handling and event-driven programming

1. Built-in Java exception classes
2. User-defined exceptions
3. When to re-throw and when to handle an exception
4. Alternatives to exceptions
5. Event-handling methods

H. Non-decimal arithmetic
1. Bitwise numeric operators
2. Bitwise logical operators
3. Binary and hexadecimal constants

I. Inner classes
1. When to use an inner class
2. Inner classes in event-driven GUI programs

J. 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

K. Topics in Graphical User Interface (GUI) design
1. GUI through applets
2. GUI through applications
3. The font class
4. Swing
5. Multi-threaded GUIs

L. Abstract classes and interfaces
1. Defining and using abstract classes
2. Defining and using interfaces
3. Implementing interfaces and abstract classes

M. Storage allocation methods
1. Run time binding and storage management of activation records
2. Declaration consequences of pointers, references and value parameters
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

N. Java Collections Frameworks
1. Java ArrayLists and ListIterators
2. Java LinkedLists and ListIterators
3. Java PriorityQueues

O. 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

P. Topics in Java File I/O
1. Binary I/O and FileInput/OutputStream
2. Handling file exceptions
3. Buffered input readers and writers and BufferedInput/OutputStream
4. Primitive data and DataInput/OutputStream
5. Print writers and PrintWriter

Q. 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
6. Representative GUI project with event-driven design
7. Multi-threading GUI project

R. 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. Familiarization with the intermediate-level online lab environment
1. Modify and customize project-specific and global settings of an Integrated Development Environment (IDE)
2. Use the IDE to create multi-file programming projects
3. Organize projects within an IDE so as to support easy project-switching and orderly submission of labs
4. Gain experience with the steps needed to edit a complex program
5. Modify IDE settings to produce an industry standard code style

B. Organizing and debugging multi-class projects
1. Demonstrate the ability to debug programs that contain multiple classes
2. Distinguish between interface and implementation in projects by creating classes that are independent of I/O modality
3. Write individual component classes that are independent of client use and can serve several client programs
4. Incorporate symbolic constants, statics and instance members into classes in a way that demonstrates a mature understanding of object-oriented programming (OOP) in a lab project
5. Debug a multi-class project to produce a working program

C. 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

D. Demonstrating competence in intermediate level algorithm design using classes within the IDE
1. Use the IDE to implement a multi-faceted algorithm and/or simulation that makes effective use of OOP
2. Evaluate and comment on other students' algorithms
3. Utilize a combination of string processing and numeric processing to address various aspects of the algorithm implementation
4. Produce clear program runs which demonstrate that the algorithm addresses a variety of cases and/or input states
5. Incorporate bitwise and logical operations to address binary logic tasks within an algorithm

E. 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

F. Incorporating basic abstract data types in programming projects
1. Implement a fundamental abstract data type (ADT) such as a queue or stack 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
G. The proper use of deep and shallow copies in conjunction with inheritance and other advanced techniques learned in previous labs
1. Create the proper set of methods within a class that enables an object to be cloned (copied deeply) correctly
2. Utilize inheritance to reinforce the segregation of data into base- and derived-level behavior
3. Utilize at least one other lab concept previously, such as binary logic or multi-dimensional arrays to further improve integration of intermediate concepts
4. Separate I/O and implementation in advanced programs
H. 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
I. Exceptions and file I/O in programming
1. Write a class that has methods which use exception handling to report errors to the client
2. Write a test client that uses try/catch blocks to detect exceptions
3. Implement an algorithm or simulation that reads from and/or writes to files rather than the user screen
4. Demonstrate various ways errors are handled and reported besides exceptions

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

Liang, Y. Daniel. Introduction to Java Programming, 10th ed.. 2014.

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

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