Department of Computer Science
Franklin College of Arts and Sciences
UNIVERSITY OF GEORGIA

Course Information Sheet

CSCI 1302

Software Development

Brief Course Description
(50-words or less)

Software development techniques in an object-oriented computer language. An intermediate programming course in Java emphasizing systems methods, top-down design, testing, modularity, and structured techniques. Applications from areas of numeric and non-numeric processing and data structures.

Extended Course Description / Comments

This course is the 2nd in a 2-part series of courses introducing students to the Java programming language. This course includes group work and/or pair programming.

Pre-Requisites and/or Co-Requisites

Prerequisite: CSCI 1301
Co-requisite: CSCI 1730

Required, Elective or Selected Elective

Required Course

Approved Textbook

The following open educational resources are available for this course:
Michael E. Cotterell and Bradley J. Barnes. CSCI 1302 Class Exercises. GitHub and Zenodo, Department of Computer Science, 415 Boyd GSRC, University of Georgia, Athens, GA, 2019sp edition, April 2019. DOI: 10.5281/zenodo.2652510.

The following textbook may be used in addition to or instead of the open educational resources above:
Author: John Lewis, Peter DePasquale, Joseph Chase
Title: Java Foundations: Introduction to Program Design & Data Structures
Edition: 4th Edition
ISBN-13: 978-0983507901

Specific Learning Outcomes (Performance Indicators)

1. Plan, design, implement, compile and execute a complete object-oriented software solution on a target Unix environment.
2. Describe and apply fundamental programming techniques, including exception handling, generics, recursion, and code reuse.
3. Apply accepted programming practices, including documentation and proper code style.
4. Describe and apply object-oriented programming techniques, including interfaces, inheritance, and polymorphism, and apply combinations of each in a software solution.
5. Utilize software development tools, including tools for version control, build management, debugging, and unit testing.
6. Design, analyze, and implement algorithms for searching, sorting, and other real-world problems.
7. Reinforce course concepts using integrative examples, including

graphical user interfaces, stream-like operations, and abstract data types.

ABET Learning Outcomes

A. Graduates of the program will have an ability to: Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions.
B. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline.
C. Communicate effectively in a variety of professional contexts.
D. Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.
E. Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.
F. Apply computer science theory and software development fundamentals to produce computing-based solutions.

NOTE: In the construction of the student learning outcomes for this course, the instructors interpreted "computing requirements" in (B) as the functional requirements for a software solution and not as specific hardware requirements for the target platform; likewise, the phrase "[a]pply computer science theory" in (F) was interpreted as using computer science principles.

Relationship Between Student Outcomes and Learning Outcomes

		ABET Learning Outcomes					
Specific Learning Outcomes		a	b	c	d	e	f
	1	●	●				●
	2	●	●				●
	3			●		●	●
	4	●	●				●
	5						●
	6	●	●				●
	7	●	●				●

Major Topics Covered

1. Unix Fundamentals (Knowledge level: Usage)
   a) Navigate and modify files, directories, and permissions in a multi-user Unix-like environment.
   b) Execute, redirect, pipe, and manage programs/processes in a multi-user Unix-like environment.
   c) Create and modify text files and source code using a powerful terminal-based text editor such as Emacs or Vi. NOTE: Extremely simple programs like nano, pico, etc. are not sufficient.
   d) Use shell commands to compile new and existing software solutions

that are organized into multi-level packages and have external dependencies.

2. Programming Fundamentals (Knowledge level: Usage)
   a) Identify redundancy in a set of classes and interfaces, then refactor using inheritance and polymorphism to emphasize code reuse.
   b) Define, throw, and propagate exceptions appropriately in a software solution.
   c) Use recursion to solve a non-trivial problem in a software solution.
   d) Implement new generic methods, interfaces, and classes in a software solution.
   e) Utilize existing generic methods, interfaces, and classes in a software solution.
3. Accepted Programming Practices (Knowledge level: Usage)
   a) Create and update source code that adheres to established style guidelines.
   b) Create class, interface, method, and inline documentation that satisfies a set of requirements.
   c) Generate user-facing API documentation for a software solution.
   d) Apply pair-programming principles in a software-based project.
4. Object-Oriented Programming (Knowledge level: Usage)
   a) Design, create and use interfaces in a software solution.
   b) Utilize interface-based polymorphism in a software solution.
   c) Design, create and use inheritance relationships in a software solution.
   d) Utilize inheritance-based polymorphism in a software solution.
   e) Use visibility modifiers to provide inheritance-based and package-based access protection in a software solution.
5. Software Development Tools (Knowledge level: Usage)
   a) Utilize a version control tool such as Git or Subversion to store and update source code in a multi-programmer software solution.
   b) Utilize a build tool such as Maven or Ant to create and manage a complex software solution involving external dependencies.
   c) Utilize a debugger to trace and identify logical errors in a software solution.
   d) Create unit tests for classes, interfaces, and methods using a unit testing framework like JUnit.
6. Analysis (Knowledge level: varies by topic)
   a) Given an algorithm, perform an analysis that classifies the algorithm according to its best Big-O class for a given unit of measurement (e.g., comparisons vs. swaps). (Usage)
   b) Design an algorithm that solves a given problem with a particular Big-O class given as a constraint. (Assessment)
   c) Implement, analyze, and assess combinations of searching/sorting algorithms such as linear search, binary search, quadratic sorts, and linearithmic sorts. (Assessment)
7. Integrated Examples
   a) Design and implement a graphical user interface in a software project. (Usage)
   b) Use stream-like operations (e.g., map, reduce, and filter in the Java Stream API) as an alternative to iteration in solving problems. Observe the difference between resulting implementations. (Usage)
   c) Use common abstract data types and structures, including lists,

queues, arrays, and stacks in solving typical problems. (Usage)

Knowledge Levels

The following is the ACM's categorization of different levels of mastery: Assessment, Usage, and Familiarity. Note that Assessment encompasses both Usage and Familiarity, and Usage encompasses Familiarity.

Familiarity: The student understands what a concept is or what it means. This level of mastery concerns a basic awareness of a concept as opposed to expecting real facility with its application. It provides an answer to the question "What do you know about this?"

Usage: The student is able to use or apply a concept in a concrete way. Using a concept may include, for example, appropriately using a specific concept in a program, using a particular proof technique, or performing a particular analysis. It provides an answer to the question "What do you know how to do?"

Assessment: The student is able to consider a concept from multiple viewpoints and/or justify the selection of a particular approach to solve a problem. This level of mastery implies more than using a concept; it involves the ability to select an appropriate approach from understood alternatives. It provides an answer to the question "Why would you do that?"

Modified	6/14/2019 by Dr. Cotterell and Dr. Barnes
Approved	No