

## COURSE DESCRIPTION

Dept., Number	<b>CSCI 150L</b>	Course Title	<b>Introduction to Algorithmic Design II Lab</b>
Semester hours	1	Course Coordinator	Crystal Edge
		URL (if any):	

### Current Catalog Description

(Prereq: Grade of **C** or above in Computer Science 140 and 140L) (Coreq: Computer Science 150) Laboratory demonstrates the topics and principles presented in the lecture. F, S, SU.

### Textbook

Horstmann, C. (2007). *Java Concepts*, 5th edition. John Wiley and Sons.  
ISBN: 978-0-470-10555-9

### References

### Course Goals

The objective is to provide a growing understanding of how computer programming works and to provide hands-on practice writing more advanced computer programs to solve problems. Building on skills from the prerequisite course, this course presents more advanced topics, larger problems, and introduces data structures. The course begins with a review of classes and methods, and then introduces arrays and vectors. Inheritance, composition and polymorphism are studied in detail, followed by exception handling, recursion, interfaces and abstract types, and generics. An array-based list implementation is studied in detail. This course adds to the foundation for subsequent programming classes, teaching students how to design algorithms to solve more complex problems requiring in-depth object oriented programming techniques and data structures, and how to implement those algorithms in a programming language.

#### **Learning Outcomes:**

Upon completion of this course, students should, at a minimum able to:

1. Discuss the purpose of classes.
2. Describe modifiers, including public, private, protected and static
3. Describe the basic components of a class
4. Construct one or more classes to solve a given problem.
5. Write common and customized methods for a class, including constructors, accessors and mutators, toString, equals, compareTo, and clone.
6. Draw a diagram to represent the state of variables and objects in memory as a program executes.

7. Describe arrays and why they are used.
8. Declare an array of an appropriate data type and manipulate data into and out of the array.
9. Write a loop structure to process an array.
10. Explain the difference between arrays and ArrayList.
11. Write a loop structure to process an ArrayList.
12. Define inheritance and explain why it is useful.
13. Define composition.
14. Define subclass and superclass.
15. Write a class that inherits from another class.
16. Write a class that is composed of an object of another class.
17. Distinguish between method overloading and method overriding.
18. Define and discuss polymorphism.
19. Define abstract classes and their purpose.
20. Define interfaces and their purpose.
21. Write a class that implements an interface.
22. Define exception.
23. Use the language-provided exception handling facilities in a program.
24. Describe recursion.
25. Give an example of a problem that can be solved with recursion.
26. Trace the execution of a recursive method.
27. Implement recursion in a small program.
28. Describe generic methods and classes.
29. Implement operations for array-based lists, including search, insert, remove, and sort.
30. Use and implement linked lists, stacks, and queues.
31. Describe advanced data structures, such as sets, maps, hash tables, heaps, and binary trees.

Prerequisites by Topic

Problem solving  
 Algorithm design  
 Program structure and style  
 Data types  
 Relational, Logical, and Arithmetic operations  
 Input/Output  
 Method design  
 Class Design  
 Control structures (conditional and iterative)  
 Array structures  
 Program walkthroughs

Major Topics Covered in the Course

Class Design and Implementation  
 Inheritance and Composition  
 Polymorphism  
 Abstract classes and Interfaces  
 Exception handling  
 Recursion  
 Data Structures  
 Searching and Sorting

Laboratory projects (specify number of weeks on each)

Class Design and Implementation – 3 weeks  
 Inheritance and Composition – 2 weeks  
 Polymorphism – 1 weeks  
 Abstract classes and Interfaces – 1 weeks  
 Exception handling – 2 weeks  
 Recursion – 1 week  
 Data Structures – 4 weeks  
 Searching and Sorting – 2 weeks

Estimate Curriculum Category Content (Semester hours)

Area	Core	Advanced	Area	Core	Advanced
Algorithms	.5		Data Structures		
Software Design			Prog. Languages	.5	
Comp. Arch.					

## Oral and Written Communications

Every student is required to submit at least   0   written reports (not including exams, tests, quizzes, or commented programs) of typically        pages and to make        oral presentations of typically        minute's duration. Include only material that is graded for grammar, spelling, style, and so forth, as well as for technical content, completeness, and accuracy.

## Social and Ethical Issues

Please list the topics that address the social and ethical implications of computing covered in all course sections. Estimate the class time spent on each topic. In what ways are the students in this course graded on their understanding of these topics (e.g., test questions, essays, oral presentations, and so forth)?

Approximately 10% of class time is spent discussing the importance of correct code and the ramifications of incorrect code in class and in society. Students are introduced to the importance of error-checking and input validation as a means to produce more secure, safe and stable code. They are graded on their ability to implement such features in a programming assignment.

## Theoretical Content

Please list the types of theoretical material covered, and estimate the time devoted to such coverage.

## Problem Analysis

Please describe the analysis experiences common to all course sections.

For each lab, students are given descriptions of problems that can be solved with computer programs. They are required to analyze the problem descriptions to gather requirements, including input and output specifications and necessary formulas or equations.

## Solution Design

Please describe the design experiences common to all course sections.

After analyzing problem descriptions and extracting requirements and specifications for each lab, students design a solution to the problem using a specified programming language. Students must determine what types of variables and data structures to use, what classes and methods are needed, and what levels of abstraction are required to improve the reusability and readability of their design. They must design the classes and algorithms to solve the given problem, and implement the solution in code. Students must consider the levels of cohesion and coupling in their solution.