CSC140 Foundations of Computer Science

Professor. Leon Tabak

Table of Contents

Calendar

	MON	TUE	WED	THU	FRI
Week 1	08	09	10	11	12
Week 2	15	16	17	18	19
Week 3	22	23	24	25	26
Week 4	29	01	02	03	04

Our meeting times and places

My office is in Law 206C.

You may call me in my office at (319) 895 4294.

You may send me electronic mail at `<l.tabak@ieee.org>`.

I will be in my office and available to meet with you Monday through Friday from 3:00 p.m. until 3:30 p.m.

We will all meet together in the classroom in the mornings and in the laboratory in the afternoons.

	Where	**When**
Classroom	Law Hall 121	1 p.m. to 3 p.m.
Laboratory	Law Hall 113	9 a.m. to 11 a.m.

Textbooks

Introduction to Programming in Java: An Interdisciplinary Approach [http://www.cs.princeton.edu/IntroProgramming] . Robert Sedgewich and Kevin Wayne. Addison-Wesley. Copyright © 2008. 978-0-321-49805-2.

AP Computer Science A Picture Lab Student Guide [http://media.collegeboard.com/digitalServices/pdf/ap/picture-lab-studentguide.pdf] . Barbara Ericson. The College Board. New York .

Software Engineering Code of Ethics . Institute of Electrical and Electronics Engineers. Association for Computing Machinery. Copyright © 1999.

Etiquette for the Classroom

Please show respect to your classmates, to me, and to the seriousness of our enterprise by exercising the following courtesies:

- Please give your attention to whomever is speaking. You cannot view unrelated pages on the Web and be part of our class' discussion at the same time.
- You learn from your classmates. Be generous in offering help to classmates in the laboratory. Take interest in your classmates' work. Encourage them. Compliment them for work that is well done. Give them a good audience when they stand at the front of the room to present their work. Show these courtesies to all of your classmates.
- Please do not interrupt the class by late entries or early departures. If you anticipate a need to be absent from all or part of one of our meetings, please notify me in advance of your anticipated absence.
- You may listen to music while working in the laboratory so long as you are still able to hear your name when called and you do not disturb neighbors.
- Please refrain from bringing food or drink into the classroom or laboratory. We can make reasonable exceptions for eating that is not noisy and foods that do not have strong smells. Acceptable beverages and foods include water, tea, and granola bars. Bringing breakfast to class is not courteous.
- Please dress as you might for an employer in the software engineering industry. Please keep your shoes on. Wearing hoods, hats, or sunglasses (except when there is a medical reason for shielding the eyes) that hide your face is not courteous.

- Imagine that you are seeking employment. How will you present yourself to your prospective employer?

  Imagine that you are now employed in a software engineering firm. How will you speak to your teammates, the head of your team, and your company's clients?

  Imagine that your grandmother has purchased the company for which you work. She has joined you in the company's conference room to hear and see you walk through the code that you have written for the company (her company). Are there some words that you will keep out of your vocabulary during this hour?

Policies

Cornell College is committed to providing equal educational opportunities to all students. If you have a documented learning disability and will need any accommodation in this course, you *must* request the accommodation(s) from the instructor of the course and no later than the third day of the term. Additional information about the policies and procedures for accommodation of learning disabilities is available on Cornell College's Web site [http://www.cornellcollege.edu/academic-support-and-advising/disabilities/academic-accommodation/index.shtml].

Please also familiarize yourself with the college's statement on academic honesty [http://www.cornellcollege.edu/registrar/pdf/Academic%20Honesty.pdf] and its policies for dropping courses [http://www.cornellcollege.edu/registrar/gb-resources-student/add-drop.shtml]

Description of the course

This course introduces students to problems that engage the interests of computer scientists and define the field. The course introduces students to object-oriented design, a principal discipline that computer scientists use to solve problems. Students learn to divide large problems into small problems, bundle related data with methods that operate on that data, and incorporate into new designs elements of previously completed designs. The course emphasizes creative expression using an abstract notation. Students practice designing, writing, testing, and presenting programs. Success in the course does not require previous programming experience.

This course will provide students with hands-on experience during two hours in the laboratory on each day of the term. The instructor will provide guidance and assistance. The instructor will encourage classmates to help one another. In total, students work together in the laboratory for more than 30 hours.

Objectives of the course

Upon the successful completion of this course, students will be able to translate simple algorithms into working code. These will include iterative and recursive algorithms. They will include five algorithms for searching and sorting.

- sequential search
- binary search
- selection sort
- insertion sort
- merge sort

Students will also be able to:

- Identify important problems in computer science and important contributors to the field.
- Analyze and compare alternative methods for the solution of a given problem (by, for example, counting the number of instructions that the computer must execute using each method).
- Design, write, and test a class that models a number that has two parts. Examples of numbers that have two parts include fractions (numerator and denominator), lengths (feet and inches), weights (pounds and ounces), times (hours and minutes), vectors (x and y components), and complex numbers (real and imaginary components).
- Contribute to the development of a program that includes several cooperating classes, including classes that are related by inheritance.
- Present programs that they have written to their peers using the Javadoc tool to extract comments from code and UML class diagrams.
- Describe the attitudes and habits that define professional practice in software engineering.

This course gives special attention to three of Cornell College's Educational Priorities & Outcomes. You learn how to apply reason in the design, development and testing of software. You will learn how to communicate with clients and teammates. You will earn how ethical conduct helps define professional practice.

- Reasoning
- Communication
- Ethical behavior

Grades

Written work will be due on each day of the term except for the first day and the last day. Printed copies and electronic copies of your papers will be due at 9 a.m.

Experience presenting work to peers will be a central part of the course. Practice asking your teammates questions during their presentations, critiquing their decisions, and suggesting improvements to their code will also be an important part of your education during this term. We will schedule one day in each week of the term for you to present your work.

	Activity	**Points**
	Daily problems and writing	16
	Code reviews (4 presentations)	20
	Examination 1 (Friday, February 12)	16
	Examination 2 (Friday, February 19)	16
	Examination 3 (Friday, February 26)	16
+	Examination 4 (Wednesday, March 2)	16
		100

Daily schedule

Monday, February 08

Read.

Read Section 1.1 on pages 3–13 (11 pages) in *Introduction to Programming in Java*.

Discuss.

What you can learn in this course that will help you in other courses and in your career, even if you never study computer science again.

A brief history of programming languages:

- imperative programming languages
- structured programming languages
- abstract data type languages
- object-oriented languages

How to write a program, prepare a program for execution, and run a program.

- What is computer science?
- The role of programming in the study of computer science.
- Why the Java programming language?
- The kinds of tools that programmers use in their work.
- Kinds of errors in computer programs.
- Knowledge, skills, habits, and attitudes that lead to success.

Church-Turing Thesis—what all computers and all programming languages have in common.

Components of a computer:

- microprocessors
- memory
- busses
- input and output devices

Software tools:

- operating systems
- compilers

- interpreters and virtual machines
- editors
- Integrated Development Environments (IDEs)

Write.

Transcribe and customize the Sequential Search program (use the handout).

Write solutions to Exercises 1.1.1, 1.1.2 (deliberate introduction of errors into the Hello, World program), 1.1.3, 1.1.4, and 1.1.5 on page 13 of *Introduction to Programming in Java*.

These exercises will familiarize you with tools we will use throughout the term and with the form of programs written in the Java language.

Return to the top.

Tuesday, February 09

Read.

Read the *Software Engineering Code of Ethics*.

Read pages 14–34 (21 pages) in Section 1.2 in *Introduction to Programming in Java*.

Discuss.

A software engineer's professional obligations (including ethical obligations) to teammates.

How (and why) to associate types with data, how to combine data arithmetically and logically, how to convert data from one type to another.

- Literals, identifiers, and variables.
- Integers, floating pointing numbers, characters, strings, and Boolean values,
- Precedence and associativity of operators.
- Conversions between data types.

We will look more closely at the parts of a class.

- Comments.
- Constructors, accessors, and mutators.
- Instance variables, class variables, and parameters.

Programmers always write for two audiences. They write for the machine—they must write code that the compiler or interpreter will accept. They also write for their human collaborators and clients. Other people will build upon the code, use the code in other projects, test the code and correct errors in the code. Programmers have an obligation to help their teammates by making their code as easy to understand as possible.

We will review conventions for naming classes, methods, variables, and constants. Conformity to these conventions will make it easier for other people to understand what we have written.

Write.

Transcribe and customize the Weight program (use the handout). This program defines a class that models a weight (pounds and ounces). The class includes a method for adding one weight to another.

Write solutions to Exercises 1.2.4, 1.2.10 (experiment that shows how arithmetic on a computer can produce surprising results), 1.2.11, and 1.2.22 on pages 39–44 of *Introduction to Programming in Java*.

You will learn how arithmetic on a computer differs from the arithmetic done in a mathematics classroom.

Return to the top.

Wednesday, February 10

Read.

Read pages 35–45 (11 pages) in Section 1.2 in *Introduction to Programming in Java*.

Discuss.

- The “software crisis”.
- Re-usable components in computer programs.
- Application Program Interfaces (APIs)

Practice with declarations of variables, initializations of variables, expressions that include variables, literals, operators, and calls to methods. We will get some practice using the methods found in the Math and String classes.

A variable is a named location in the computer's memory. A variable has six attributes:

1. name
2. type
3. value
4. address
5. scope
6. lifetime

We will introduce types. We will define the differences between integers and floating point numbers.

- What is a type?
- Why did the designers of Java include type-checking in the language?

The interpreter can produce values by evaluating expressions. We will learn how to write expressions that include numeric literals, calls to methods, and arithmetic operators.

Write.

Write a program that models a length (feet and inches). Imitate the Weight program (given to you).

Write solutions to Exercises 1.2.31 and 1.2.33 (compute distance between two points on the surface of the earth) on pages 43 and 44 of *Introduction to Programming in Java*.

Your programs will compute functions found in Geographic Information Systems. A Geographic Information System is a tool for creating, editing, sharing, interpreting, and analyzing maps and other geographic data.

Return to the top.

Thursday, February 11

Read.

Read pages 46–73 (28 pages) in Section 1.3 in *Introduction to Programming in Java*.

Discuss.

How to direct the computer to choose between two alternative sequences of statements. How to direct the computer to execute a sequence of statements repeatedly.

- computing square roots
- producing the binary representation of an integer
- simulating gambler's ruin
- if statements
- while statements
- for statements
- break statements
- do-while statements
- switch statements
- compound statements
- examples to illustrate the use of loops:

We will learn how to instruct the computer to make decisions. We will learn how to describe the condition under which the computer should take a prescribed action by writing expressions that evaluate to true or false. This will require the introduction of Boolean operators.

Write.

Transcribe and customize the Binary Search program (use the handout).

Write solutions to Exercise 1.3.3, 1.3.4, 1.3.5, and 1.3.6 (experiments with if statements and for loops) on page 77 of *Introduction to Programming in Java*.

You will begin to learn how to guard against errors and make your programs more robust.

Return to the top.

Friday, February 12

Read.

Read pages 74–85 (31 pages) in Section 1.3 in *Introduction to Programming in Java*.

Discuss.

Practice with if statements, for loops, and while loops.

Learn about several famous and curious mathematical problems. Discover that your new knowledge of programming enables you to solve them!

Write.

Write solutions to Exercises 1.3.43 (a program that models the growth of a population of, for example, fish or bacteria) and 1.3.44 on page 85 of *Introduction to Programming in Java*.

In these exercises you will see a little of how computer science has provided new methods for discovery and experimentation in the natural sciences and mathematics.

Examination 1. The examination include a question that will ask you to define a class that models a time (hours and minutes). The solution to that exercise will look very much like the Weight and Length programs (seen in exercises).

Return to the top.

Monday, February 15

Read.

Read pages 86–111 (26 pages) in Section 1.4 in *Introduction to Programming in Java*.

Discuss.

How to create and use variables that hold multiple values. How to declare and initialize arrays, assign a value to an element of an array, retrieve a value, and exchange two values.

We will establish the distinctions between constructors and methods.

We will establish the distinctions among instance variables, constants, and local variables.

We will learn how to use methods to describe the behaviors of an object and how to use instance variables to describe the state of an object.

We will outline five steps for the design of a method:

1. Describe in a single, plainly worded sentence the purpose of the method.

2. Give the method a descriptive name.

3. Identify the kinds of data that the method needs to receive from its caller. Give these parameters types and descriptive names.

4. Identify the type of result that the method will return to its caller.

5. Write a sequence of logical and arithmetic operations by which the method will produce the effect described in the first step.

We will see how and why we can overload methods and constructors—we will define methods within a class that have the same name, but different numbers and types of parameters.

Write.

Transcribe and customize the Find Minimum program (use the handout).

Write solutions to Exercises 1.4.18 (compute average length of a path in a self-avoiding random walk) and 1.4.19 on page 115 of *Introduction to Programming in Java*.

Return to the top.

Tuesday, February 16

Read.

Read the *Software Engineering Code of Ethics*.

Read pages 112–119 (8 pages) in *Introduction to Programming in Java*.

Discuss.

A software engineer's professional obligations (including ethical obligations) to clients.

Practice interpreting and writing loops that manipulate arrays. How to recognize and avoid easy to make mistakes.

We will look more closely at looping and decision-making constructs. We will complete our list of Java's control statements. We will practice rewriting Boolean expressions, loops, and branches to make our programs easier to read and understand. Once again, we will emphasize that we are writing for both the machine and other human beings—we must make our intentions clear to both!

Write.

Transcribe and customize the second and third versions of the Find Minimum program (use the handout). The second version finds the position (rather than the value) of the smallest element in a list. The third version find the position of the smallest element in that part of the list that begins at a specified position.

Write solutions to Exercises 1.4.12, 1.4.15 (compute product of boolean matrices), and 1.4.17 (on pages 114 and 115 of *Introduction to Programming in Java*).

In these exercises, you will learn more about how to work with matrices. Matrices have important applications in computer graphics.

Return to the top.

Wednesday, February 17

Read.

Read pages 120–151 in Section 1.5 (32 pages) in *Introduction to Programming in Java*.

Discuss.

How to read, write, and format data. How to create drawings and sounds.

Write.

Transcribe and customize the Selection Sort program (use the handout).

Write solutions to Exercises 1.5.5, 1.5.6 (filter a sequence of integers by removing consecutive, repeated values), and 1.5.7 (on page 154 of *Introduction to Programming in Java*).

In these exercises, you will see some simple ideas for the design of error detection and correction codes and data compression codes.

Return to the top.

Thursday, February 18

Read.

Read pages 152–161 (10 pages) in Section 1.5 in *Introduction to Programming in Java*.

Discuss.

Write.

Transcribe and customize the Insertion Sort program (use the handout).

Write solutions to Exercises 1.5.19 and 1.5.21 (plot a function of polar coordinates) on page 157 of *Introduction to Programming in Java*.

You will write programs that draw geometric patterns.

Return to the top.

Friday, February 19

Read.

Read Section 1.6 on pages 162–180 (9 pages) in *Introduction to Programming in Java*.

Discuss.

Java is more than a programming language. It is a family of technologies, tools, and libraries of software. Within Java's libraries, the classes that model buttons, labels, menu items, scrollbars, and other elements of graphical user interfaces rank among the most important. These classes are components (designed, written,

and tested by other people) that we can plug into our own programs. We do not have to write everything from scratch!

We will examine a program that simulates the behavior of people on the World Wide Web. The program contains several cooperating classes. We will see that sometimes simple assumptions and a simple program can yield powerful insights.

Write.

Write a solution to Exercise 1.6.13 (compute powers of a matrix) or 1.6.17 on page 180 of *Introduction to Programming in Java*.

Examination 2.

Return to the top.

Monday, February 22

Read.

Read Section 2.1 on pages 183–217 (35 pages) in *Introduction to Programming in Java*.

Discuss.

- How mathematicians define functions and how computer scientists define methods.
- How (and why) to define our own static methods.
- How to give a method a parameter that is an array.
- How to define a method that returns an array to its caller.
- How to represent harmonic tones in a musical note.

A brief introduction to “big-Oh” notation.

Your teachers in elementary school might have asked you to define words. When they did, they insisted that you give the meaning of a word without using the same word in the definition. Computer scientists do not accept this restriction!

A recursive definition refers to itself. A recursive method invokes itself.

We will see how recursion sometimes enables us to describe the solution of a problem more simply and directly, and how it is sometimes the key to writing more efficient solutions. We will see as well that a naive use of recursion can greatly increase the amount of working required to solve a problem.

Write.

Transcribe and customize the second version of the Binary Search program (use the handout). This version is recursive.

Execute the Instrumented Sorts program (given to you). Vary the parameters. Record the outputs. Use a spreadsheet program to tabulate and graph the number of comparisons as a function of the length of the list for the selection and insertion sorts.

Write solutions to Exercises 2.1.7, 2.1.8, 2.1.19 (compute a histogram), and 2.1.21 on pages 209 and 212 of *Introduction to Programming in Java*.

Return to the top.

Tuesday, February 23

Read.

Read the *Software Engineering Code of Ethics*.

Read Section 2.2 on pages 218–253 (36 pages) in *Introduction to Programming in Java*.

Discuss.

A software engineer's professional obligations (including ethical obligations) to society.

- How and why programmers share code.
- How to create our own libraries (collections of related methods).
- The special importance of testing and documenting code that is to be shared.

Choices about how to organize data within a program greatly influence the simplicity (or complexity) of a program. We will begin our study of data structures by experimenting with various applications of arrays. We will discover a close relationship between arrays and loops (because we can use loops to initialize arrays, update the values of the elements of an array, determine whether an array contains a particular value, and so on).

The library that comes with Java includes classes that model many kinds of data structures. We will look closely at just one of these classes. Instances of the ArrayList class, like arrays, hold elements that each have an index. Unlike arrays, ArrayLists can grow—in this and other ways ArrayLists represent improved arrays.

Write.

Transcribe and customize the Merge 2 Lists program (use the handout).

Write solutions to Exercises 2.2.1, 2.2.3, and 2.2.11 (design, write, and test a class that models a matrix) on pages 248–249 of *Introduction to Programming in Java*.

Return to the top.

Wednesday, February 24

Read.

Read Section 2.3 on pages 254–285 (32 pages) in *Introduction to Programming in Java*.

Discuss.

- How to write recursive methods.
- Reasons for sometimes prefering recursion.
- Cases in which recursion does not provide an advantage.

- Classic examples of recursive algorithms.

We will review and relate the concepts that define object-oriented design. These concepts include information hiding, inheritance, and polymorphism.

We will learn how to refer to methods and constructors of a parent (super) class. We will learn how to limit access to variables from instances of other classes, and how to restrict access to classes that extend a given class.

Write.

Transcribe and customize the Merge Sort program (use the handout).

Write solutions to Exercises 2.3.1, 2.3.2 (recursively compute logarithm of N!), 2.3.7, and 2.3.30 on pages 278, 279, and 284 of *Introduction to Programming in Java*.

Return to the top.

Thursday, February 25

Read.

Read Section 2.4 on pages 286–313 (28 pages) in *Introduction to Programming in Java*.

Discuss.

Computer scientists study algorithms for searching and sorting because:

1. a large fraction of all software contains searching or sorting functions

2. lessons learned through the study of the complexity of searching and sorting algorithms can be applied in the study of the complexity of other kinds of algorithms

We will define the complexity of an algorithm by measuring how the number of instructions executed varies with the size of the data on which the algorithm operates. For example, the number of operations executed in a sequential search is proportional to the number of items in the array in which we look for a key value.

We will write, modify, and measure the performance of algorithms that search for a key value in data. We will evaluate sequential and binary searches.

We will write, modify, and measure the performance of algorithms that sort data. We will evaluate the insertion, selection, and merge sorts.

We will introduce a program that simulates percolation. Examples of percolation include the movement of water or petroleum through porous rock. Again, we will see that simple assumptions and a simple program can yield powerful insights. This program includes several cooperating classes.

We will talk about how classes can be related through inheritance and aggregation. We will talk about the problems that multiple inheritance could cause and show Java's interfaces as a solution.

Write.

Write solutions to Exercises 2.4.1, 2.4.4, 2.4.9 (add code to Percolation program that will record and report depth of recursion), and 2.4.20 on pages 308, 309, and 312 of *Introduction to Programming in Java*.

Return to the top.

Friday, February 26

Read.

There is no reading assignment for today.

Discuss.

The Waterfall Model, although flawed and outdated in important ways, is a good starting point for a discussion of the ways in which software engineers create new products. An examination of the Waterfall Model (and better models that followed the Waterfall Model) will also give us a clearer picture of where opportunities lie within the software engineering profession.

- Requirements. (Determine what the clients need. No Java here! The clients probably do not know very much about computer science and we probably have a lot to learn about our clients' work.)
- Specification. (Quantify the required functionality, cost, and performance. Establish priorities. Negotiate with clients.)
- Design. (Recruit a team, choose a programming language and other tools, develop a budget and schedule, divide the problem into pieces, and define the relationships among components.)
- Code. (Translate the design into a working program that meets the specifications.)
- Test. (There are many kinds of tests: alpha, beta, white box, black box, unit, modular, and system. We will need testers who have a deep understanding of software and testers who have a deep understanding of how our clients work.)
- Release. (Make the case to potential customers that this is a product that can make their work easier. Help customers learn how to use the software effectively.)
- Maintenance. (Correct errors as they are discovered and reported. Add new features. Adapt the program to run on other platforms.)
- Retirement. (Give notice that support for the product will end. Meet and conclude all contractual obligations. Maintain and build good will so that customers will want to buy the next product.)

Which of these roles most appeals to you? How do you think that this sequential approach might fail in practice? How could we improve upon this formula for the development of software?

Write.

Examination 3.

Return to the top.

Monday, February 29

Read.

Read *AP Computer Science A: Picture Lab Student Guide*.

Discuss.

In our examination of the Picture Lab, we will bring together and review several of the big ideas that we encountered during the term:

- primitive data types: int, double, boolean
- 1-D and 2-D arrays
- ArrayList class and List interface
- classes: instance variables, methods, parameters, local variables
- relationships among classes: inheritance, aggregation
- selecting data structures and developing algorithms: measuring complexity

We will consider ways in which software engineering differs from other kinds of engineering. How does software differ from other kinds of products (or services)?

The world needs computer scientists with many different talents and interests. We will review some of the talents and interests that could be the foundation for satisfying and successful study of computer science.

We will survey the variety of special fields within computer science, other academic disciplines, and careers in which students might apply their knowledge of computer science.

Write.

Complete the exercises 3 and 4 (create a color negative image and create a black and white image) in the Picture Lab (page 13). Optionally experiment with algorithms for posterization and for creating bas relief images and vignettes.

Return to the top.

Tuesday, March 01

Read.

There is no reading assignment for today.

Discuss.

Identify the special responsibilities of a professional software engineer to customers/clients, teammates, professional peers, and to other citizens.

Outline opportunities for further study of computer science.

Prepare for the final examination by reviewing what we have learned during the term.

Write.

Use of any of the programs that you have developed during the term to generate an image to take home. Experiment with color, scale, or other parameters to get the most pleasing effect. This can be the picture that your mother proudly posts on her refrigerator.

Evaluate the course.

Return to the top.

Wednesday, March 02

Read.

Review previously assigned readings, your notes, and your programs.

Discuss.

There is no discussion scheduled for today.

Write.

Final examination.

Return to the top.

Thursday, March 03

Read.

There is no reading assigned for today.

Discuss.

We will not meet today.

Write.

There is no writing assignment for today.

Return to the top.

Friday, March 04

Read.

There is no reading assignment for today.

Discuss.

We will not meet today.

Write.

There is no writing assignment for today.

Return to the top.