Linfield College, Computer Science Department
COMP 160: Introduction to Programming
Syllabus fall 2014
Prof. Daniel Ford

COURSE DESCRIPTION:
Introduces the basic concepts of programming: reading and writing unambiguous descriptions of sequential processes. Emphasizes introductory algorithmic strategies and corresponding structures. (QR). COMP 160 is required for a major in electronic arts and is a prerequisite for COMP 161 and all other required courses for a major or minor in computer science.

PREREQUISITES:
Prerequisite: MATH 105 or equivalent. No prior programming experience is required. You must be willing to practice regularly – teaching yourself to recognize patterns, and to organize, simplify, and solve problems.

CREDITS: 3

LECTURES: Renshaw 211
Section 1: MWF 12:45 – 1:35 PM

WORKSHOPS: Renshaw 211
Workshop 1: Tuesday & Wednesday, 7—8 PM (tentatively)
Additional help and tutoring is available by request and through the learning center.

PROFESSOR: Daniel Ford
Office: Renshaw 210 Phone: x2706 E-mail: ford@linfield.edu
Office Hours: MF 10:30—11:30 AM, TTh 12:30—1:30 AM, and by appointment.

RECOMMENDED BOOKS:
- **Cay S. Horstmann,** Big Java, http://www.horstmann.com/bigjava.html. Recommended for COMP 160 & 161. Wiley 5th Edition (2012 Early Objects) ~ $90, 4th Edition, 3rd Edition, …, 2nd Edition (2005) ~ $10.
- **Y. Daniel Liang**, Introduction to Java Programming, Comprehensive Version, http://cs.armstrong.edu/liang/intro9e/. Recommended for COMP 160 & 161. Faster, more advanced, but with fewer tips than Big Java. 9th Edition (2012) ~ $90, 8th Edition, …, 6th Edition (2006) ~$10.
- **Cay S. Horstmann & Gary Cornell,** Core Java™, *Volume I – Fundamentals* and *Volume II – Advanced Features* (9th Edition), 2012 http://www.horstmann.com/corejava.html. The best reference manuals for Java. Recommended for CS majors and professional Java programmers. $37 each.

RECOMMENDED (FREE) WEB RESOURCES:
Java
- **CodingBat.com**. http://www.codingbat.com/
- **Java 6 API**. http://java.sun.com/javase/6/docs/api/
- **Liang's Java Debug Common Errors and Answers** http://cs.armstrong.edu/liang/intro9e/debug.html
- **Horstmann's Worked Examples** http://bcs.wiley.com/he-bcs/Books?action=resource&bcsId=7872&itemId=1118431111&resourceId=30501
- **Horstmann's Sample Programs** http://horstmann.com/bigj5/bigjava.zip
- **Horstmann's Animations** http://bcs.wiley.com/he-bcs/Books?action=resource&bcsId=7872&itemId=1118431111&resourceId=31273
- **The Java Tutorials** http://java.sun.com/docs/books/tutorial/index.html
- **Liang's Algorithms and Data Structure Animations** in JavaScript and Processing http://cs.armstrong.edu/liang/animation/animation.html

JavaScript and Processing
- **Kahn Academy Programming** https://www.khanacademy.org/cs/programming
- **Processing.org (Reference** http://processing.org/reference/, **Tutorials** http://processing.org/tutorials/**, and Examples** http://processing.org/examples/.)
- **Sketchpad.cc** http://sketchpad.cc/ sketchpad is built on processing.js and etherpad http://etherpad.org/ a handy tool for collaboration.)
- **Has Canvas** http://hascanvas.com/

- **Open Processing** http://www.openprocessing.org/
- **Fun Programming** http://www.funprogramming.org/ (not a fan of the videos, but the organized list of 128 simple examples on how to do various things is excellent!!)
- **Proclipsing (processing in Eclipse)** https://code.google.com/p/proclipsing/
- **Codecademy** http://www.codecademy.com/
- **AppendTo JavaScript Video Lessons** http://learn.appendto.com/
- **LearnStreet Beginner JavaScript Course V3** http://www.learnstreet.com/lessons/study/javascript
- **W3Schools JavaScript Introduction** http://www.w3schools.com/js/js_intro.asp
- **Mozilla Development Network JavaScript Guide** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
- **Cheatography JavaScript Cheat Sheet** http://www.cheatography.com/davechild/cheat-sheets/javascript/

COURSE OBJECTIVES:

- Practice decomposing complex algorithmic problems into smaller simpler problems.
- Practice solving problems via iterating over *strings* and *arrays*.
- Practice *tracing* and *debugging* programs.
- Practice testing programs, e.g. via JUnit tests.
- Practice writing programs with *style*.
- Practice researching programming solutions.

GRADES:

Grading for this course will be based on: (I thought I talked to Jacob about weekly mini quizzes?)

6 Chapter Quizzes	24%
11 ¼ Java Programming Assignments	33.75%
3 Exams	36%
JavaScript Project	6.25%

Make-up exams and quizzes will not be granted without prior arrangement or for reasons stated in the college catalog.

MAKE UP QUIZZES AND EXAMS

Make-up quizzes and exams will not be granted without prior arrangement or for reasons stated in the college catalog.

LABS/WORKSHOPS:

This class will have weekly workshops run by students, tentatively Tuesday and Wednesday night from 7-8pm. These are not mandatory, but are highly recommended especially if you find it challenging to finish all of the weekly programming problems. Extra-credit will be given for attendance.
Many students have claimed that the workshops were even more helpful then the lectures.

CODINGBAT.COM

CodingBat.com/home/kermit@linfield.edu provides an online Java practice environment. I recommend that you create an account with codingbat and share your account with me, dford@linfield.edu, via the *prefs* link in the Teacher Share -- enter the email address of the teacher account ... section. Do not rely on CodingBat to save your work: keep your own copy in your CS account on Nova.

ACADEMIC CONDUCT:

In this course we will adhere to the college policy on academic honesty, as published in the Linfield College Course Catalog. Please review this policy if you are not already familiar with it.

- Obey the *30 second long term memory rule*. We all get stuck now and again. If you get help on a homework problem in the form of looking at someone else's code, take a minute to think about their code why it works or doesn't work and how you might improve on their idea before starting to write your own solution.
- Do not use any code that you do not understand.

In computer science you will earn credit by combining basic building blocks, e.g. the vocabulary of a specific language, common patterns, well known data structures, and algorithms, into expressions, segments of code, procedures, functions, and entire programs that solve a particular problem or accomplish a specific task. The basic building blocks are common knowledge and thus represent ideas that do not need to be cited. You are expected to generate, evaluate, and illustrate your own ideas on how these building blocks can be put together to solve particular

problems. The building block combinations that you submit for credit will illustrate a combination of ideas that is similar to the combinations used by others in some respects and unique to your combination in others.

PRACTICE, DEBUGGING, AND LEARNING ON YOUR OWN:

Learning programming is a bit like learning a sport: I will guide your efforts to help them to be as productive and enjoyable as possible, but most of your progress will come from practice and learning on your own.

A major part of this process is debugging, i.e. generating and implementing ideas to investigate and determine the cause of problems, a.k.a. bugs, as well as ideas to generate, test, evaluate and select solutions to fix these problems. These debugging ideas are implicit in any solution you submit for credit.

The purpose of having credit for the course based on problem solving and debugging skills is that repeated practice in generating and evaluating solutions is the best way to increase your programming and problem solving skills.

CLASS/GROUP PARTICIPATION:

Computer science requires tolerance, individual contributions, teamwork and the ability to learn from others. For the academic endeavor to succeed, we must treat each other with civility, courtesy and respect. You will frequently be encouraged to work in. pairs. Every student will be expected to make pertinent and substantive contributions to every group they are a member of.

One learns more when working in groups when the others help them to remain productive: to generate better ideas and evaluate them faster and to spend less time on technical difficulties with little pedagogical value. If, however, the group is doing the work for you, supplying you with ideas and evaluating and choosing which ideas to use to solve your problems, then they are doing you a disservice by reducing your opportunity to generate and evaluate your own ideas and become a better programmer.

COPYRIGHT, PATENTS, PLAGIARISM, AND COLLABORATION IN COMPUTER SCIENCE

There is no universally accepted definition of Plagiarism. Plagiarism is an ethical not a legal issue. Copyright and patents have legal definitions and are financial issues. Plagiarism is a type of lying, pretending that you deserve **credit** for an idea when the credit should go to someone else. Copyright and patent infringement is gaining financial benefit from someone else's idea or its expression.

Plagiarism encompasses the following guidelines:

1) You should give credit where credit is deserved and desired; if you want to present someone else's idea and they want credit for it than you should give it to them.
2) You shouldn't accept or receive credit for others ideas; when submitting something for credit, give credit to others for their ideas, even if they don't care, so that no one inadvertently gives you credit for them.
3) You shouldn't try to gain credit for others ideas: attempting to get credit for someone else's idea is unethical even if you didn't benefit from it, e.g. because the idea was incorrect or useless.

Any student submission for credit, any publication, any broadcast speech, any piece of art, and many other works emphasizing original creativity should satisfy all three of the plagiarism guidelines above. However, any work by a student, faculty, or other individual where they are neither attempting to gain nor gaining credit for being the **original source** of the ideas presented are by definition satisfying guidelines 2 and 3 and thus only need concern themselves with following the first guideline. For example, if the person presenting an idea neither intended to take credit for it nor received credit of any value for it, and if the person who came up with the idea didn't desire to take credit for it in that particular presentation, then there is no plagiarism even if the person presenting doesn't acknowledge the original source of the idea.

Academics obsess over the concept of originality and associated credit of ideas: it is often the primary component in how we are judged and in how the colleges we work at are ranked. Taking credit for someone else's academic idea is akin to stealing part of their reputation. Moreover, ideas are not created in a vacuum. They are preceded by a progression of other ideas. It is considered the moral responsibility of the person claiming credit for an idea to accurately communicate how much credit they deserve for the idea and how much credit others deserve for the ideas that led up to it. Not giving credit for someone else's idea that led up to your own is implying that you also deserve credit for their work, and hence akin to stealing part of their reputation.

Ideas, however, can be somewhat nebulous. Multiple people can seem to come up with the same idea independently, and people often disagree about how similar one idea is to another and who deserves credit for what. Some people debate that it is impossible to accurately assign credit to an idea. In contrast, the expression of an idea in writing is much more concrete than the idea itself. This is why plagiarism usually focuses on the copying of the written expression of an idea.

ILLNESS/MEDICAL POLICY
In the unfortunate event that you must miss class you are encouraged to contact your workshop partner to find out what we worked on in class and to pick up any materials that were handed out. In general, homework extensions and make-up exams will be granted as needed on an individual basis.

DISABILITY SUPPORT:
Students with disabilities are protected by the Americans with Disabilities Act and Section 504 of the Rehabilitation Act. If you are a student with a disability and feel you may require academic accommodations contact Cheri White, Assistant Director of Learning Support Services (LSS), within the first two weeks of the semester to request accommodations. LSS is located in Walker 126 (503-883-2444). We also recommend students communicate with their faculty about their accommodations and any special needs an instructor should be aware of.

LINFIELD CURRICULUM:
For students who entered Linfield Fall 2010 or later, in order to earn a QR for this course, you must submit relevant exemplars of your work to TaskStream by the last day of finals week, as discussed in the Linfield College Course Catalog, pages 6-8.

This course contributes to the Linfield Curriculum in the area of *Quantitative Reasoning* (QR). Courses with this designation develop the student's ability to:

1. *Frame contextual questions using mathematical representation.*

When programming, we often ask the following question, "What sequence of instructions would accomplish our goal?"
Programming frames real world questions about achieving a goal first into a desired quantifiable outcome and then creates a hypothesis, i.e. a program or sequence of logical instructions, on how to achieve that outcome. Programming involves repeatedly testing these hypotheses and observing the result. Any difference between the result and the desired outcome is generally the result of faulty deductive reasoning.

2. *Apply models to deduce consequences and make predictions.*

When programming, the best question is often, "What would be the result of executing the following sequence of instructions? "
In order to better understand our deductive reasoning errors, improve our deductive reasoning skill, and debug/improve our program/hypothesis, programming involves repeatedly making predictions about the consequence of executing a sequence of logical instructions and then testing these predictions/hypotheses by writing a program consisting of these instructions, executing the programs and observing the results. During this process the programming language, e.g. Java, and related abstractions, e.g. data structures and algorithms, act as a model. We use our understanding of this model to predict the consequences of executing specific sequences of instructions. Note the similarity between the scientific method, which explores questions by forming, testing, and modifying hypotheses, and programming.

3. *Verbally communicate and critique quantitative arguments.*

It is often much easier for others to spot our mistakes than it is for us to spot them, and sometimes simply explaining something to someone else will help us realize errors in our logic. As social and fallible creatures we humans often find it enjoyable, enlightening, and better to program with others, and this benefit increases with practice. Moreover, real world projects are almost always so large that working in groups is unavoidable.