CS302 Course Policies
Spring 2016

Last modified on:
January 15, 2016

Contents

This document has been created from years of information finding its way to the course website. Original content provided by Jim Skrentny, Beck Hasti, Deb Deppeler, Laura "Hobbes" LeGault, Gary Dahl and Jim Williams. Other instructors may have also contributed to the content found in this document.

Use this handbook to find CS302 course policies that all students must know and follow.

We have made every effort to have this be the most accurate information for the current semester. However, we do make changes to continuously improve the course.

Please be sure to follow your instructor's web page and download a new copies of this document, should it become necessary to make policy revisions during the semester.

Chapter 1

Introduction

1.1 What is CS302? Course Descriptions

1.1.1 Undergraduate Catalog Description

Instruction and experience in the use of an object-oriented programming language. Program design; development of good programming style; preparation for other Computer Science courses.

1.1.2 Our Description

This course introduces you to fundamental computer programming concepts as you learn to program in the Java language. Algorithm development, structured programming, code organization (methods), data organization (arrays), basic object-oriented programming, file access, and exception handling are covered.

Note we do not consider CS302 an "introductory" course in the sense that it means the course is easy. This is an introduction to programming that, for some students, may be very challenging.

1.2 Prerequisites

Problem solving skills such as those acquired in a statistics, logic, or advanced high school algebra course; or consent of instructor. Open to Freshmen. Note the course does not require that you have prior programming experience.

1.3 Course Summary

Students will solve a variety of problems using fundamental programming constructs and data structures. Students also learn about and how to use advanced constructs like object-oriented programming using classes and interfaces, as well as how to use exception handling, and testing and debugging to improve the robustness of the program solution.

1.4 Learning Objectives

Students who successfully complete CS302 are able to write computer programs in a high-level programming language. Currently, we use the high-level language Java to teach our CS majors fundamental programming concepts and object-oriented programming. Once you know Java,

you will find it much easier to go on an learn any programming language that interests you and is useful in your field of study or employment.

Students successfully completing this course will be able to analyze problems and formulate algorithms; create robust, user-friendly, well-structured and well-documented Java programs; read basic Java programs to determine their purpose; and have a basic understanding of how computers work.

1.5 Personal Computers

Students are not required to purchase or use their own personal computer. Students who do wish to work from home should review the "At Home" page for more information about choosing and installing the development tools (IDE and API software) used to develop programs in the labs for the course. Caution: NetBooks, tablets, and smartphones will not be suitable for most program development work. Academic Conduct: Students are required to do their own work and not seek, view, use, or submit work created by anyone other than their pair-programming partner (if allowed and they choose to work in a pair). Code provided by us as part of assignment may be used without citing source.

1.6 Programming Language

There are many programming languages available for programmers to choose from. Each language came about to solve a particular type of programming problem and typically has features that make solving those types of problems easier than other languages.

Machine Languages

Language understood by the actual hard-ware of your computer. Often described as 1's and 0's and very hard for humans to interpret and write functional code.

Assembly Languages

One step better than machine language for people. Uses mnemonics for basic commands and can be translated (compiled) into machine language by a computer. Used infrequently now that we have sophisticated and highly optimized compilers and efficient interpreters.

High-Level Languages

High-level languages imply program code that is much more human-readable and that other programs are then used to translate (compile) and run (execute machine binaries or interpret results).

Here are just a handful of higher level languages that you may have the opportunity to experience in your studies or work life.

- Compiled
  - Pascal
  - ForTran
  - COBOL
  - c/C++
  - Java
  - C#

- Interpreted
  - Basic and QBasic
  - Java
  - Perl
  - MATLAB
  - PYTHON

As there are many stakeholders to consider and very limited resources, it is very difficult (maybe not possible) to pick a language that meets all students individual objectives. Technical colleges attempt to meet many different skills that are needed in the work place, by having a course or several for each programming language based on current demand. As many UW-Madison students will find themselves learning and using many different programming languages throughout their careers, and we know that the languages that our students may need may not even exist yet or our students will invent the language they need, we teach a more theoretical curriculum that will give students the skills to learn many different programming languages. Thus, Java mostly hits a sweet spot between many different goals for our intro programming language course.

Additionally, many students report difficulty in learning to program in c/C++/Java than learning some current favorites like Perl or Python, or mark up language like HTML. The fact that it is in some ways more difficult to learn Java and object-oriented programming, is only one good reason for us to teach it to our CS majors.

We also chose Java because it is:

1. strongly typed
2. object-oriented
3. available for free on major platforms, Windows/Mac/Linux
4. has a substantial built-in API or function library

The faculty and course instructors frequently discuss the pros and cons of each course's language and editor choice. If you're curious about other programming languages and how to get started learning them, discuss your interests with your instructor during office hours and expect lots of discussion about the pros and cons of various language choices. There is rarely a one size fits all programming language.

1.7 Program Development Environment

There are many parts to the puzzle of developing a working program and there are many tools (programs) to help program developers. We have chosen an Integrated Development Environment called Eclipse for the instructional labs and most students will find it best to use the system installed and used during lab meetings.

An integrated development environment (IDE) is a fancy name for a suite of several programs combined into one program that the student downloads and installs. Most IDEs provide the programmer with a file editor (source code), a compiler (translates source code), a runtime environment (a way to run the program), and a built-in debugger tools (provides ability to step through code one line at a time and view current state of variable).

Eclipse is the IDE that students will use in the CS lab meetings and on the CS instructional machines. Students are encouraged to download and install Eclipse (and the JDK) on their

home computers to be able to practice programming and work on programming assignments at home.

We chose Eclipse because it:

- is free to lab and students
- has syntax-highlighting
- has a built-in compiler that provides help while editing
- is available for free on major platforms, Windows/Mac/Linux
- has intuitive and built-in debugger tools
- it also is fully integrated with JUnit testing (though we do not cover that use in CS302).

1.8 Textbook

1.8.1 Why do we require a textbook?

We choose a textbook for CS302 because there is a vast array of information about programming available to all students, but it is not always organized. Thus, many students will not find all relevant information and many will find much irrelevant information. A required textbook allows us to chose an organization and structure for the many concepts we wish to cover. The text, allows us all to find and use the same reference information in one convenient location. Additionally, multiple terms may be used for the same concept, and in those cases, having a textbook helps us all learn and use a standard vocabulary for those concepts.

A textbook also facilitates students learning about topics that are not covered in the course, but may be of interest and importance to that individual student. We will not cover all chapters in the textbook, but many students will be able to expand on their own knowledge based on the extra content that is found in the textbook but not required by the course itself.

Be sure to read assigned readings before lectures on that chapter for the most benefit.

Tips to make the most out of your textbook purchase

1. skim the assigned readings to get general terms and concepts before lecture
2. read the entire chapter for real after lab to get as much out of it as you can
3. review the chapter for missed details before the exams
4. reference the chapters as needed while working on programming assignments

1.8.2 Printed Textbooks

Many students (and instructors) prefer to have a printed text for ease of use as a reference while working on their computer. Being able to mark pages, add notes and bookmarks, and draw diagrams and arrows is very easy on a printed copy and avoids having to change which window is being viewed while working from our computer screens.

1.8.3 eText access

eText versions of textbook are also appealing to many students as they can carry a copy with them on their computer, tablet, and even their phone. We try to have both printed and eText available for students to chose from.

1.8.4 Online Programming exercises

Learning to program is best done by writing code, lots of code, small code fragments, medium code fragments, small programs, large programs, and more. One way we give students practice writing small bits of code at their own pace with immediate feedback is to have them complete online exercises that get "graded" immediately.

Students can retry those exercises as often as necessary until they get the correct result. They can even repeat with different solutions to the same problem. This type of experiential learning is invaluable when learning to program. Since we can't be with every student during all coding practice, the online tools give the student and us feedback about how things are going. Be sure to purchase the online programming tool and get registered and start on the assigned exercises as soon as possible. You'll get some credit toward your final grade, but mostly you'll get lots of repetition and practice writing code that will help you learn many deeper programming concepts.

We assign exercises to assist students with each chapter's topics. Be sure to complete the online programming exercises as soon as you can to get the most benefit from the small exercises.

1.8.5 Using other textbooks instead of or in addition to the required textbook

Occasionally students ask about the possibility of saving some money and purchasing other editions of other textbooks instead of the required texts. In short, this a possibility and may work for students who are willing to go to the extra trouble of learning the differences between the textbook versions. Be sure to learn the vocabulary used by the required textbook should it differ in any way from another textbook.

Caution, there are many editions that are comprehensive texts for introducing programming fundamentals using the Java Programming Language. However, Students are encouraged to purchase the required text as they will spend less time ensuring that the topics, vocabulary, and reference chapter numbers are the the same and verifying that they have found all relevant content.

Students who intend to use other textbooks should check out a reserved copy of the required textbook as soon as one is available and make notes regarding the differences in Chapter numbering, titles, etc. Students are responsible for the content in the required textbook.

Chapter 2

Course Work

2.1 Grading

Final letter grades are determined from your final weighted percentage score that is computed using the following breakdown:

* Exams 45%
  There will be three (3) exams, 1st midterm (10%), 2nd midterms (15%) and a final (20%). CS exams are cumulative, but each exam will focus on new material. Letter grades are assigned at the end of the semester. We do not curve individual assignments or exams. At the end of the semester after all of the scores are recorded, the scores of the current semester are compared against scores from previous semesters. Thresholds are set given the relative difficulty of the course work assigned and graded during the semester. Thresholds are raised if it appears the course work was easier and they are lowered if our analysis shows it was harder. By adjusting and setting thresholds at the end of the semester we can account for varying difficulty among semesters to ensure consistency of grading across semesters. If you have a question about your grades, please meet with your instructor.
* Programs 40%
  There will be four (4) programming assignments each worth 10% of your final grade.
* Labs 15% (75 total lab points possible)
  - Team Lab: There are fourteen (14) Team Labs each worth 5 lab points. You can get at most 70 of the 75 lab points by attending and fully participating in all 14 Team Labs. You earn the remaining 5 of 75 lab points by completing 100 CodeLab exercises.

    If you miss a Team Lab, you should still complete the exercises to ensure that you do not miss material. Credit for missing a team lab can be earned by completing an additional 100 CodeLab exercises.
  - CodeLab: CodeLab by TuringsCraft.com is a set of online coding exercises we've chosen to help you practice the material we'll be covering in class. Each 20 exercises that you complete successfully before its due date (usually Monday evenings) will count as 1 lab point.

    You may attempt CodeLab exercises as many times as you need to get them correct, without affecting your grade. Students will have the opportunity to complete more than 500 exercises over the course of the semester. Most are offered at the start of the semester for maximum benefit to students.

Solutions to CodeLab problems are available via CodeLab on the date after their due date so you can review a solution to any exercise that you did not solve before its due date. Do not ask for solutions or post solutions to exercises on Piazza or any other public forums, as having a solution available means you don't gain the same benefit from the exercise (ie. someone else clearing a hurdle doesn't make you any more likely to clear the same hurdle).

- Example 1: Complete 14 Team Labs for 70 lab points and complete 100 CodeLab exercises for 5 lab points. 70 + 5 lab points is 75 total lab points out of 75 maximum.
- Example 2: Complete 12 Team Labs for 60 lab points and complete 500 CodeLab exercises for 25 lab points. 60 + 25 lab points is 85, but you can earn at most 75 lab points.

Scores for labs, programs, and exams are reported through the Learn@UW System. Students are responsible for checking their scores in Learn@UW each week. Contact your Lab TAs to correct any lab attendance scores. Contact your program grader to correct any program scores. Contact your Instructors to correct exam scores. CodeLab scores will not be entered until the end of the semester. Until then, students can find the number of exercises completed in CodeLab.

2.2 Final Grades

Final Grades are determined by computing a weighted average of these categories and assigning a letter grade based on a grading curve scale. There is no curve computed for individual exam and program scores. It is the final overall percentage that is used for final grades. If particular assignments and or exams are more difficult than other semesters, the final grade cutoffs may be adjusted to reflect the relative difficulty when compared with other semesters. This scale reflects the standard curve used before any adjustments are made.

- A 93 and above
- AB 88 and above
- B 80 and above
- BC 75 and above
- C 70 and above
- D 60 and above

2.3 Studying Tips

* Attend lectures and take notes.

  Taking your own notes on the material covered in class requires you to organize your thoughts and think about the topics in ways that simply listening does not. Reread your notes soon after lecture to find points that are not clear.

* Participate in class.

Do in-class exercises and answer questions posed. These exercises give you additional practice on skills that you will need for labs, programming assignments and exams. Even more importantly, they help you identify which topics you do not understand. This will help you efficiently use your study time.

* Do the labs and programming assignments.

  This is obvious, but some students mistakenly believe that the purpose of labs and programming assignments is to evaluate student's skills. That is not the case, their purpose is to provide opportunities for you to practice the programming concepts taught in lecture. Such practice is helps students learn to program, discover common mistakes (and their solutions), and discover details about how the most common commands work. Such in depth understanding is vital for performing well on the midterm and final exams.

* Complete CodeLab exercises.

  CodeLab exercises are designed to give you practice with programming terminology and lots of immediate feedback on your coding which has been shown to greatly improve outcomes for students new to programming. This practice is vital for learning how to program and for performing well on the midterm and final exams.

* Prioritize your studying.

  First focus on your lecture notes, review the labs, then review the assignments, and finally go over the readings in the textbook.

* Study by re-organizing your notes.

  Successful studying requires you to be actively thinking about the material. An effective way to study is to re-organize and rewrite notes succinctly in terms that you understand.

* Avoid getting bogged down on specific points.

  If you can't figure something out, move on to other material and wait to ask your instructor or TA for clarification.

* You'll need to do some memorizing.

  Memorizing is a practical way to learn the definitions of terminology and concepts. A good way to memorize information is through repeated exposure to the material.

* Study by Scientific Method: EXPERIMENT! (ADDED 11/17/15)

  An effective way to learn how the keywords and concepts in programming work is to write code fragments and new method and class definitions using those keywords and then experiment with the results. Be scientific, make hypotheses (predictions), record them,

design an experiment (code sample) to test your hypotheses. When you see the actual results, carefully compare your expected results with your actual results. Identify areas that you thought you understood but do not. Be sure to try things that know as well as things you don't yet know.

* Form a study group.

  Often it is easier and more motivating to work with others when studying for an exam. You can distribute the work by having each group member come up with a few questions on a topic and then going over the solutions collectively. If you run into a concept that causes confusion often others in the group will feel similarly while some will have explanations. Working together will help you learn things by providing multiple perspectives and insights into the material. Try to find a study group that provides you with ample opportunity to communicate your understanding. The effort you make to express information that will be of the most benefit to your learning.

* Finally, avoid cramming!

  We all know this, but often we find ourselves waiting until it is too late. This habit is worth breaking! Research has shown that studying in half hour to hour intervals followed by half hour breaks is far more effective than non-stop cramming.

2.4 At Home

To work effectively from a home computer, students must know or learn several things:

What a file system is? and how to use their computer's file system.

How to download and install software on their computer.

What an API is? and how to download and install the correct one.

What an IDE is? and how to download and install the correct one.

See the Programmer's Handbook for more information about each of these items. It is not uncommon for students to spend several hours learning how and doing all of these steps. Don't let that discourage you, you will learn so much and feel very empowered knowing how to perform these steps on your computer.

Chapter 3

Labs

3.1 Online Programming Exercises

CodeLab is a online interactive programming application. Students complete exercises and get feedback regarding the correctness of the solution code. Students may retry the exercises as many times as they like. As long as they solve the problem before the due date and time, they will get credit.

Solutions to exercises are available online the day after the exercises are due. See Grading for information about how the exercises are used in your final grade calculation.

3.2 CS Instructional Labs

3.2.1 General Information

The instructional Windows computer labs have Java and Eclipse installed for CS302 students to do their lab work, their programming assignments, and to practice Java programming. Students are automatically issued a Windows account to use the CS lab computer, which is done by the Computer Systems Lab (CSL) after you register for the course. This account is completely separate from the email account supplied to you by the university. You may also choose to work on your own machine but you'll need to follow instructions to correctly download, install and configure the your program development environment. This typically includes installing a development kit for the desired language (JDK) and an integrated development environment (Eclipse). See course web site for details on which ones are recommended for your section.

3.2.2 Locations & Hours

The instructional computer labs are located on the first floor of the Computer Sciences building. These labs are open from 7 AM - 1 AM every day. The following Windows 7 labs available for the use by CS302 students.

Operating System	Room	Workstation Type	Notes
Windows 7	1350 CS	core i7 2600 16GB RAM	Used for labs on Tues and Wed (see schedule)
Windows 7	1370 CS	core i7 2600 16GB RAM	Used for labs on Tues and Wed (see schedule)

3.2.3 CS Login and Account Activation

CS302 Students have Computer Science (CS) accounts created for them to log into the Workstations that are located in the instructional labs on first floor, 1350cs and 1370cs. This means that you will have a separate login username and password for accessing the computers in the CS building. Please activate your account before your lab session.

If a new account was created for you this semester, you should have received instructions for activating your CS account via your WISC email.

Windows Users

You can now activate, get your CS login name, or reset your password directly from one of the CS Windows machines.

Login with the following credentials:

Username: AD
accthelp No password.

An application lauches that goes directly to the CSL account activation website. The application can sometimes take a small amount of time to load, so please be patient if it seems like nothing is loading.

The instructions are also written on the whiteboard in each Windows instructional lab and signs will be posted shortly.

Existing CS Accounts

If you took a CS class last spring or summer (including 302 on Windows), you already have an account, and do not need to activate it. If you don't remember their username/password, use the activation process to reset your password.

To activate or reset the password on a CS Instructional Account:

Use any web browser to visit csl.cs.wisc.edu/account_management/computer-sciences-account-activation You will need to authenticate to the UW-Madison NetID Login Service.

If you are not already authenticated via NetID, you will be redirected to the NetID Login page when you continue to the account activation page. Follow the instructions to activate your account or reset your password.

Login

Once you have activated your account, you are able to login to any of the CS instructional machines (1st floor labs).

3.3 Lab Sections

3.3.1 Description

CS302 students are registered for one 75-minute lab, which meets once per week in the CS Instruction Lab rooms. Attend the lab for which you enrolled. Labs give you valuable experience using a computer while you work with an assigned partner to solve lab problems. Lab instructors (TAs) guide labs and are there to help you succeed on the lab problems. Labs give students the opportunity to learn concepts with a partner and in a small group. lab exercises will also help you learn vocabulary and achieve the desired programming skills.

3.3.2 Collaboration

Students have an assigned lab partner and a machine to use. You'll work with that partner for the first three weeks. You be assigned a new lab partner for weeks 4, 8 and 12 of the semester.

3.4 Grades

Labs grades are based on prompt attendance, preparation, cooperation with your partner, and progress completing the lab. Cell phones and other personal electronic devices must be turned off and put away during lab. Students may not use personal computers during lab.

3.4.1 Missed Team Lab points

Make up missed team lab points by completing more than the required 100 CodeLab exercises. You earn 1 pt for every 20 CodeLab exercises and there are over 500 CodeLab exercises. Caution: most CodeLab exercises are due in the first half of the semester. So don't wait until the end of the semester to complete CodeLab exercises.

Chapter 4

Programs

Programs help you achieve the desired programming skills.

4.1 Collaboration

If a programming assignment is designated as a team assignment (in pairs) then you must follow the Pair Programming rules especially registering via FORMS tool BEFORE you begin work with another student.

4.2 Pair-Programming

Pair Programming is where two programmers work together to write one program solution. It is critical that both programmers work on all aspects of the solution. Students are not permitted to divide the work up and have each student complete only some part.

Both team members must register the partnership using the CS302 Forms Page BEFORE you begin the programming assignment assignment together. REMINDER: Both teammates must log in to Forms and complete the Team Information section. One teammate will create their team using the forms tool and the other teammate will join the team via Forms tool. Only when both teammates have completed their part of the process is the team considered registered.

Only one of the team members hands in the program. It can be either team member, but it must be one of you. Either member may submit the program and download a backup of the team's work. Use List Files to download a backup copy. The grade and grade report are assigned to both students.

- You may have only one partner for each programming assignment.
- You may have a different partner on different programming assignments.
- You may not pair program with multiple partners on the same assignment.
- Your partner must be currently enrolled in a CS302 lecture.
- You may partner with someone from a lecture other than your own.
- You must list your partner as a collaborator in the header comments of each of your source files (see Commenting Guide).
- You must follow the principles of pair programming summarized below.

- You may cancel your team membershipup until the team due date, but you will not be permitted to join a new team for that programming assignment. To disband your team, log in to Forms, enter Team Information page, and click button labelled Leave Team to cancel your membership in that assignment's team.

Submitting someone else's work as your own is academic misconduct, which will result in a zero on the assignment, as well as possible additional penalties in accordance with University Academic Misconduct procedures. It is also Academic Misconduct to help another student commit Academic Misconduct. Do not show your solution or share your files with anyone other than your registered programming pair partner.

4.2.1 Principles of Pair Programming

The following is a summary of successful pair programming principles taken from a paper by Williams and Kessler:

Pair programming involves two people working together at one computer, continuously collaborating on design, coding, and testing. One person types; the other either provides directions ("Now we need to write a new method that does ..., Now we need a loop to do ...") or constantly reviews what the typer is doing, and provides comments.

Pair programming has proved successful both in classes and in industry. Programmers usually report having more confidence in their solutions and enjoying programming more when working in pairs.

It is important to switch roles often (slide the keyboard back and forth). Because pair programming can be quite intense, it is also a good idea to take breaks (to check e-mail, go for a walk, have a snack).

It is important to provide honest but friendly feedback. To be effective, there needs to be some healthy disagreement and debate, but pairs also need to be respectful of each other, and try to avoid becoming defensive when receiving criticism.

Inevitably, programmers do some independent thinking/working. For best results, that work should be reviewed by both partners (and perhaps revised) when they start working together again.

To be successful, pair programmers must realize that the benefits of working together outweigh their usual preference for working alone, they must confidently share their work, accepting instruction and suggestions for improvement in order to improve their own skills and the code they are writing, and they must accept ownership of their partner's work and thus be willing to constructively express criticism and suggest improvements.

4.3 Late Programs

Late programs are not accepted for a grade without prior approval from your instructor. Contact your instructor at least three (3) days in advance of the due date if circumstances beyond your control will prevent you from completing an assignment by the due date/time. We will ask you to submit the work you've completed, which we will review and then may give you an extension to the due date.

4.4 Regrades

Students may request that their program be regraded, if you believe there was an error in the grading of your program. You may request a regrade once per programming assignment.

Regrades are for when you believe your program was incorrectly graded or for circumstances when a minor error results in a disproportionate deduction. Regrades are not for making a few changes to get a few points back or for re-writing your code. You should verify that your code works in advance of the due date by comparing your results to the posted results. To make a regrade request, send an email to the TA listed in your Grade Report within one week after the graded program is returned.

4.5 Grade Reports

A grade report with comments about the grading of your work is accessed through the CS302 Forms Page (introduced in your first lab). To see your grade report: log into the CS302 Forms Page, click on the particular program, and then click on "Grade Report".

Chapter 5

Exams

Exams help us determine if you have achieved the desired programming skills. There are three exams worth a significant portion of your final grade. See "Grading" for details. Exams are multiple choice unless directed otherwise, and they serve as the primary tool we use to evaluate your performance in this course.

In general, students are responsible for:

- lectures (including topics covered in lecture that are not covered in the textbook)
- readings (including topics covered in the textbook that are not covered in lecture)
- labs and online lab exercises
- Programming Assignments.

5.1 Content

Each exam has an Exam Topics section where we outline major chapters and topics for that particular exam. However, as exams are necessarily a sample and not complete coverage of course material, it would be counter productive to indicate only those parts that will get asked about on any given exam. Students must prepare for questions on any topics covered and not limit themselves to those topics found on a previous exam.

Any and all information available regarding exams is available here (the Exams sections of the Policy handbook), the Exams section on the course web page, and the samples posted via Learn@UW.

Students are cautioned that we strive to ask different questions regarding the content of the current semester, so we do not advise studying previous exams to determine which topics will be on your exam. Old exams are provided to illustrate some types of questions.

Please develop your own ways of determining if you have correctly answered a question. For example: Code your answer and the question code, or write your own test methods. Typically, each instructor will plan an exam review session to answer any remaining questions during the last lecture before the exam.

5.2 Campus ID

Be sure to have your UW Campus ID and arrive at the correct exam room for your lecture.

5.3 Exam Room Assignments

For the location of the buildings, click on the building name or check out the campus map. We will direct you to the correct exam room should you arrive at the wrong room and this would most likely delay your start on the exam. For the location of the buildings, click on the building name below or check out the campus map.

5.4 Exam Materials

No textbooks, electronic devices, or help from neighbors are allowed during the exams. Reference material is provided as part of each exam. See sample on Learn@UW.

5.5 Exam Conflicts

Enter exam conflict information for ALL exams via the CS302 Forms Page. Conflicts for all three exams must be submitted during the first three weeks of class.

5.6 McBurney Exam Accomodation Requests

McBurney students must complete an "exam conflict" submission via the CS302 Forms page during the first three weeks of class. McBurney students must also give a hardcopy of your VISA to the CS302 Faculty Associate.

5.7 Alternate "Makeup" Exams

Make-Up Exams are given only with the Faculty Associate's permission when you are unable to take the regular exam due to extenuating circumstances. Requests for make-ups after an exam are rarely approved and only for verifiable emergencies.

5.8 Regrade Requests

Be sure to review the correct answers (if available) before asking for a regrade. Original scantron forms can be shown to students, but there has never been a scanning error. It is more common that students mistinterpret their scantron report. However, if you believe there was an error in the grading of your exam, submit your request for a regrade in writing and with your exam to your instructor within one week of the return of the exam results to the class.

Note: It is possible to lose points in a regrade. For example, if we notice some other mistake that was not indicated on the original grading of your exam. Also, we sometimes photocopy written exams to ensure that students cannot benefit from a re-grade by making changes to their answer (such attempts are academic misconduct and will result in a zero on the exam and possible additional consequences).

5.9 Exam Topics

Students do not need to memorize the operator precedence table or the methods for the Java classes covered. We will provide a reference in the exam for operator precedence and for select methods for these, and other, Java classes: `Math`, `Scanner`, `String`, `Random`, and `Arrays`.

See the course website for links to any available sample question sets or old exams. These topic lists represent the most common topics for each exam over most recent semesters.

Please be sure to follow the course website and discussion forums for the most up-to-date information on what is covered for each exam.

Note: Unless specified, Random Fact" sections are not included.

Exam 1 topics

Exam 2 topics

Exam 3 topics

5.9.1 Midterm Exam 1 topics

- Lectures
- Textbook readings, see topics below
- Labs: 1 through 5
- Programming assignment 1 and static methods part of assignment 2

Introduction

algorithm and pseudocode, IDE, source code → compiler, class files (bytecode) → virtual machine, class and main method, use of braces, statements, parameters, string of characters, `System.out.print` vs. `System.out.println`, compile-time (syntax) vs. run-time (logic) errors, exceptions, debugging and code tracing.

Fundamental Data Types

declaring and initializing variables, numeric data types (`int, double`) and overflow, naming rules and conventions, reserved words, comments, assignment operator (`=`) and assignment statement, program user input and `Scanner` class, `import` statement, defining constants with `final`, expression, operator, operand, arithmetic operators (`* / + - %`), integer division, operator precedence, unary increment operator (`++`) and unary decrement operator (`--`), `Math` class, compound assignment (`+= -= *= /= %=`), cast operator, `String` class, concatentation, string input with next and nextLine, escape sequences (`\n \t \\`), `String` methods (such as length, charAt, substring), `char` type, calling instance methods vs. calling `static` methods; NOT INCLUDED: Special Topic 2.2

Decisions

conditions, `boolean` type and `true false` values, relational operators (`== != < <= > >=`), `if` statements (`if`, `if-else`, nested (`if-else-if`) forms and flowcharts), `switch` statement (including `break` statement and `default` case), boolean variables and boolean operators (`! && ||`), truth tables, `==` vs. equals, nested branches, dangling else problem, short-circuit evaluation, DeMorgan's Law, input validation (using `Scanner` instance and hasNext), choosing test cases (normal, abnormal, boundaries) NOT INCLUDED: Special Topics 3.1-3.2, 3.4-5

Loops

repetition idea, repeat *while* true, infinite loops, off-by-one errors, `while` loop, sentinel or event-controlled loop, count-controlled loop, definite vs. indefinite loop, `for` loop, scope of `for` loop

counter variable, `do` loop, post- vs. pre-test loop, user input validation, sentinel value termination, common repetition algorithms (such as summing, average, counting matches, comparing adjacent values), nested loops

Methods

call-execute-return sequence, declaring and calling (`static`) methods, parameter passing and *pass-by-value* (argument values stored in parameter variables), returning values and `return` statement, `void` methods, call stack tracing, stepwise refinement and incremental coding, stubs, local variables and variable scope; NOT INCLUDED: 5.9 Recursive Methods

Arrays and Array Lists

declaring and initializing arrays, initial values list and filling arrays, accessing elements and indexing from 0, length property, bounds checking and `ArrayIndexOutOfBoundsException`, array reference and memory diagrams for arrays, partially-filled arrays, array algorithms (such as linear search, removing/inserting, printing with `Arrays.toString(<type []>)`, copying with `Arrays.copyOf()`, passing arrays to and returning arrays from methods; NOT INCLUDED: 6.2 The Enhanced for Loop, 6.7 Two-Dimensional Arrays, 6.8 Array Lists, and Special Topics 6.1-6.3 `Random` class seed, nextInt(), next(), generating random integers for a specified range, class (`static`) methods, calling object, Java API.

5.9.2 Midterm Exam 2 topics

You are responsible for material covered in first exam as well as this additional material:

- Lectures
- Textbook readings, see topics below
- Labs: 1 through 10
- Programming assignment 1, 2, and 3 (though we won't expect that you've finished 3)

The reference pages will includes `ArrayList` class methods.

Arrays and Array Lists

two-dimensional arrays, `ArrayList` class, type parameter for generic class, wrapper classes and auto-boxing, methods to add/remove/find/size/etc., copying with the constructor, passing/returning, plus: arrays of objects, 2D arrays with variable row (second dimension) lengths, partially filled arrays of objects and null, initiallizing the array vs. the objects in the array; NOT INCLUDED: 6.8.2 Using the Enhanced for Loop

Objects and Classes

object-oriented programming (OOP), class diagram, Program 3's object diagram, flow charts, (instantiable) class vs. object (aka instance), creating objects using `new` operator, defining instance methods and declaring instance fields, default values for instance fields, interface vs. implementation, `public` vs. `private`, encapsulation, accessors and mutators, boolean methods, constructors, overloading, main (or driver or tester) class, object references and `null`, shared references (aka aliases), passing/returning references, `this` reference, calling instance methods,

tracing OOP method calls and memory diagrams, equals vs. `==`, toString() method, instance constants, class (`static`) constants, class (`static`) variables, class (`static`) vs instance (non *static*) methods, wrapper classes for each primitive (ie. `Character` and `Integer` and other wrapper classes) and named constants.

Other Topics

- fields and variables (local, instance, class)
- constants (local, instance, class)
- chaining method calls (using the dot operator)
- garbage collection
- this() constructor call

Here's a link to a tutorial by Oracle about static vs instance members of Java classes: http://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html.

5.9.3 Midterm Exam 3 topics

The final exam is cumulative in that new topics build on previous ones. The focus (maybe 2/3 to 3/4) will be specifically on these additional topics. Review the topics listed for both midterm exams in addition to those listed below. You are also responsible for the new material covered in:

- Lectures
- Textbook readings, see topics below
- Labs: 11 through 14
- Programming assignment 3 and 4

Input/Output and Exception Handling

text files, file names - paths (absolute vs. relative) - directories (root vs. current, "." vs. ".."), import java.io.*, file output using PrintWriter, file input using Scanner with File class, FileNotFoundException and IOException, close methods, InputMismatchException, command line arguments, exception mechanism and messages, exception classes and objects, ArrayIndexOutOfBoundsException and IndexOutOfBoundsException, NullPointerException, ArithmeticException, throwing exceptions (throw statement), catching exceptions (try-catch statement), exception handlers, finally clause, exception class hierarchy, checked vs. unchecked exceptions, checked exceptions and throws clause, defining your own exceptions with extends clause, NOT INCLUDED: 7.2.9 Formatting Output and Special Topics 7.1 to 7.5

Inheritance and Interfaces

Inheritance, superclass, subclass, extends, protected, overriding (redefining) inherited methods, Special Topic 9.1: calling the super class constructor, the "Cosmic" superclass: Object class, instanceof operator and casting, toString and equals ,defining and implementing an interface, implements clause, Comparable interface, compareTo method NOT INCLUDED: Special Topics: 9.3,9.4,9.8,9.8

Chapter 6

Resources

6.1 Getting Help

There are several resources available to help students learn to program and succeed in the course.

6.1.1 Instructor

Getting help from your instructor by participating during class and visiting them during office hours is a great way to get help directly. To facilitate getting help from your instructor during office hours, please be sure to write your questions down and have your work and code available for review. While we can look up work that you've handed in, it takes time and thus increases the wait for other students and reduces the number of students we can assist during scheduled office hours.

6.1.2 Lab Teaching Assistant (TA)

Your Lab TA's primary goal during lab is to help all students understand the material and to help students learn to solve problems on their own. This can be difficult as simply giving the correct answer is counter-productive. We strive to help students figure out the answer or even how to tell if they have solved the problem. It is common for TAs and Instructors to ask "What do you think?", "How would you check to see if this was the correct result?", etc.

6.1.3 Consulting Hours in the labs

Lab TAs will also be available during scheduled consulting hours in the labs. As with office hours, all students benefit when each student writes down their question and is prepared to show all work tried and explain exactly where their program does not behave as expected. Attempt to identify the few (less than 10) lines of code that are not working as you intend.

6.1.4 Piazza

The course's primary discussion forum. Registration is required and the tool is free to students. All program specific announcements are posted via Piazza. Be sure to register to recieve emails notifications so that you don't miss critical information.

6.1.5 Google

The use of Google is not forbidden, but it is discouraged. If you must search via Google for something, please let us know so that we may include such reference information on course pages

for the use of all students.

6.1.6 Tutoring

Many students believe that they can only learn to program if they have a personal tutor. Personal tutors can help by providing immediate and individual feedback in a timely manner. But, there are many ways that a personal tutor can actually inhibit your learning the material.

If you do decide to hire a tutor, your tutor may help you interpret your notes from lecture, textbook, labs, etc.

Tutors may not under any circumstances write any code (not one line) for your programming or lab assignments. If they can not help you without writing code for you, they are not an effective tutor for you and their help may even harm your learning.

An effective tutor will provide new problems for you to solve. They will require you to actually write and work through problems, giving feedback on your work without doing the work themselves.

An effective tutor will know and promote good programming practices, like carefully reading program specifications, using comments and style that promote program readability and easy maintenance.

An effective tutor will not promote the use of techniques and constructs that have not yet been introduced in the course. This is sometimes suggested to students as an "easier" or "better" or "more efficient" way, but can obscure the real objective of an assignment.

Not finding a tutor should not mean that you do not recieve timely and effective feedback and help as you learn to write programs. We have Piazza, lab consultants, office hours, and answer questions during lectures.

If it is after hours and you're stuck, please consider asking your question publicly via the course discussion tools. Just be sure to make your question general enough that an answer will not be code that solves the assigned problem. Focus your question to the smallest piece that you don't understand and ask about that with a made up sample code that illustrates just the concept that you're having difficulty with. Then, work on a different part of the problem until you have a new idea or another student or TA offers a suggestion to your post.

Is is common that the work you do to ask your question will in fact give you a new idea to try. So don't wait to write your question down, even if you don't post it publicly.

DO NOT PROVIDE YOUR CODE TO ANYONE OTHER THAN YOUR REGISTERED PAIR-PROGRAMMING PARTNER (NOT EVEN YOUR TUTOR). We have recently investigated and prosecuted a case where a "tutor" shared one student's code with another student they were "helping". THIS IS "ACADEMIC MISCONDUCT".

Chapter 7

Academic Misconduct: ALL STUDENTS MUST READ

It's very simple, asking anyone else to solve YOUR programming problems or using anyone else's work is Academic Misconduct. Getting caught will subject you to zeros on assigments and a letter to the Dean of Students. For example: Students caught posting links to assignments to code forums and or code for pay websites, will fail the course and receive a letter to the Dean of Students. Students caught submitting work not wholly created by them or their confirmed teammate will receive a zero on the assigment and a letter to the Dean of Students. Students caught cheating on any exam will receive a zero on ALL exams and a letter to the Dean of Students. Students caught allowing their work to be used by other students in the course will receive a zero on that assignment or exam and a letter to the Dean of Students. Students caught cheating twice will fail the course and may be expelled from the University.

Don't cheat on programs.

Don't cheat on CodeLab.

Don't cheat on exams.

Keep your work private and password protected. On exams, keep your answers covered. Do not take photos or communicate with anyone during the exam. Do not let your eyes wander. Students who get caught cheating on an exam get a zero on the exam and other penalties as determined by the instructor and the Dean of Students.