CSE332: Data Abstractions, Summer 2012

General Information

Lectures: Mondays & Wednesdays, 10:50am–12:20pm, Electrical Engineering (EEB), Room 026
Section AA: Thursdays, 9:40am–10:40am, CSE, Room 203
Section AB: Thursdays, 10:50am–11:50am, CSE, Room 203
Staff: Katherine Deibel (instructor)
David Swanson (teaching assistant)
Website: http://www.cs.washington.edu/education/courses/cse332/12su/
Course E-mail List: cse332a_su12 AT uw DOT edu
Discussion Board: https://catalyst.uw.edu/gopost/board/deibel/28528/

Catalog Description

This course covers abstract data types and structures including dictionaries, balanced trees, hash tables, priority queues, and graphs. Various related concepts are also covered: sorting; asymptotic analysis; fundamental graph algorithms (graph search, shortest path, and minimum spanning trees); concurrency and synchronization; and parallelism.

Contact Information

Course staff can be reached individually at the e-mail addresses below. However, we generally recommend that you send e-mail instead to **cse332-staff@cs.washington.edu**. Email sent to that address will reach all of the course staff, thereby helping ensure a quicker reply. If your message is specific to one staff member, do feel free to directly contact that person. When e-mailing, be sure to include your UW NetID so that we can look at your dropbox, grades, etc.

Course Staff E-mails

Kate Deibel (instructor): deibel AT cs.washington.edu
David Swanson (TA) swansond AT cs.washington.edu

Course announcements will also be made regularly on the course mailing list **cse332a_su12@uw.edu**. Be sure to read all messages sent out on said list. Announcements will include items such as corrections, advice, and schedule changes. Your UW e-mail address (not your CSE e-mail) should be automatically subscribed within 24 hours of registering for the course. You can manage your subscription, change your e-mail address, view archives, et cetera by visiting the list's mailman page.

You can also send Anonymous Feedback to the instructor using the form at the provided link.

Course Discussion Board

The course also has a discussion board for you to use. Participation is completely optional but it is provided as a means to discuss course topics, offer peer help on assignments, and post pictures of cats (the inevitable fate of most online message boards). Course staff will monitor the board. When posting to the board, keep in mind the course's collaboration and academic integrity policies.

Office Hours

Kate Deibel: Tuesdays, 2:30-4:00pm CSE 210
Also available whenever my office door is open (please do visit!)
And by appointment

David Swanson: Mondays and Wednesdays, 2:30-3:30pm in CSE220
Sundays, 1:30-3:30pm in Allen Library Research Commons, Green C
And by appointment

Calendar: Lecture/Section Materials and Assignments

Topics and readings are subject to change. Be sure to check with this calendar regularly. Lecture/section materials will be posted usually by the start of lecture/section if not earlier.

Week 1: Jun 18 – Jun 22

Mon Lecture: Introduction and Abstract Data Types
Read *Weiss* Chapter 3

Wed Lecture: Algorithmic/Asymptoptic Analysis excel file
Read *Weiss* Chapter 2

Thu Section: Big O notation

Assignments:

Week 2: Jun 25 – Jun 29

Mon Lecture: Priority Queues and Binary Heaps
Read *Weiss* Chapter 6-6.5

Wed Lecture: Dictionary ADT: Arrays, Lists and Trees
Read *Weiss* Chapter 4-4.3

Thu Section: Simulating recursion using a stack

Assignments:

Week 3: Jul 2 – Jul 6

Mon Lecture: Balanced Trees: AVL, Splay, and B-Trees
Read *Weiss* Chapter 4.4-4.8

Wed Lecture: Holiday

Thu Section: More on Balanced Trees

Assignments:

Week 4: Jul 9 – Jul 13

Mon Lecture: Hashing
Read *Weiss* 5-5.5

Wed Lecture: More Hashing and some Sorting
Read *Weiss* Chapter 7 (excluding 7.4, 7.10, 7.12)

Thu Section: Midterm Review

Assignments:

Week 5: Jul 16 – Jul 20

Mon Lecture: Sorting
Read *Weiss* Chapter 7 (excluding 7.4, 7.10, 7.12)

Wed Lecture: Midterm Exam

Thu Section: Post-Exam Review

Assignments:

Week 6: Jul 23 – Jul 27

Mon Lecture: Introduction to Graphs, Graph Traversals, and Topological Sort
Read *Weiss* Chapter 9.1-9.2

Wed Lecture: Introduction to Fork-Join Parallelism
Read *Goldman* 1-3.3

Thu Section: More on Fork-Join

Assignments:

Week 7: Jul 30 – Aug 3

Mon Lecture: Fork-Join Framework and Analyses
Read *Goldman* 3.4-5

Wed Lecture: Parallel Algorithm Examples: Prefix, Pack, and Sort
Read *Goldman* 5-9

Thu Section: Sample Fork-Join code: FindMin and Histogram

Assignments:

Week 8: Aug 6 – Aug 10

Mon Lecture: Unweighted Shortest Path Analysis and Dijkstra's Algorithm
Read *Weiss* 9.3

Wed Lecture: More Graph Algorithms
Read *Weiss* 9.4-9.7

Thu Section: Final Exam Review

Assignments:

Week 9: Aug 13 – Aug 17

Mon Lecture: Disjoing Set Union Find and Minimum Spanning Trees
Read *Weiss* Chapter 8

Wed Lecture: Final Exam

Readings

Course Texts

The textbook is *Data Structures and Algorithm Analysis in Java, Mark Allen Weiss, 3rd Edition, 2011*, ISBN: 0132576279. You may purchase the second edition (ISBN: 0321370139), but we do recommend the third edition. We will the the textbook to provide a second explanation for material covered in class, and we will also assign some homework problems from the text.

A Java reference is also strongly recommended but not required. While there are a variety of sufficient references, we recommend *Core Java(TM), Volume I--Fundamentals 8th Edition, Cay S. Horstmann and Gary Cornell, 2007*, ISBN: 0132354764. Note this book is also recommended for CSE 331.

Other Readings

For the parallelism and concurrency material, these reading notes (written by UW CSE's own Dan Grossman) will be helpful and cover the lecture material: A Sophomoric Introduction to Shared-Memory Parallelism and Concurrency (pdf).

Here are some interesting, usful, and accessible articles related to the course material. These are optional reading that you may find helpful and enriching.

- The Data-Structure Canon, George V. Neville-Neil
- Bubble Sort: An Archaeological Algorithmic Analysis, Owen Astrachan
- Wikipedia: Comparison Sort (follow links to different sorting algorithms to see animations of the algorithms in action)
- You Don't Know Jack About Shared Variables or Memory Models, Hans-J. Boehm & Sarita V. Adve

Software Installation and Use

As needed, section will provide basic orientation to the software used in the course.

We will have three programming projects using Java. The third project will require **Java 7**, the most recent version of the language. So if installing Java on your own machine (rather than using the department lab machines), then you may as well get Java 7 now from http://jdk7.java.net/.

How you go about writing your Java code is up to you. You may choose to work in a standard text editor or you may use an IDE. Course staff will do our best to support either the **Eclipse** or **jGRASP** IDEs. You can download Eclipse for your own machine from http://eclipse.org/ and jGRASP from http://jgrasp.org/. Both are installed on all departmental lab machines.

Java's support for generics will be very useful in our projects, but its support for *generic arrays* requires a few common workarounds. These workarounds are discussed here in a to-the-point description.

Grading

Overall course grade

Your overall grade in this course will depend on your performance in four areas: written homework, programming projects, and two exams. Your scores in these areas will be weighted as as shown in the table. These proportions may change if necessary, but such change is unlikely.

Weightings for Final Grade	
Written Homework:	25%
Programming Projects:	25%
Midterm Exam:	20%
Final Exam:	30%

We will have approximately three programming assignments (with phases) and eight written homework assignments. Expect each written homework to contribute equally to the course grade. We will drop your lowest homework grade. The first project will contribute *half* as much to the course grade as the subsequent two projects.

You will be able to follow your performance on the course gradebook through Catalyst.

We will track any extra features you implement (the "Above and Beyond" or Extra Credit parts). You will not see these affecting your grades for individual projects, but they will be accumulated over all projects and used to bump up borderline grades at the end of the quarter.

Submitting Your Work On Time

Written assignments are due promptly by the end of lecture the day the assignment is due. Late assignments will **not** be accepted. If you cannot attend lecture, please arrange to turn in your homework earlier to the instructor or have a classmate turn it in for you during lecture.

Programming projects will be submitted electronically to the course dropbox by the deadline announced for each assignment. The dropbox for each assignment will close shortly after the deadline. After it closes, no further submissions will be accepted.

Lateness Exemptions

As can be seen in the above paragraphs, this course is very strict about its turn-in deadlines. Because we understand that occasional exceptional circumstances can and do occur, we provide the following opportunities/exceptions:

- **Twice** per quarter you may use a *late day waiver* to obtain an extra 24 hours. You will need to contact the course staff to state that you are using your waiver. You can apply your two late day waivers to the same assignment if so desired.
- The course staff are also caring, emotional beings and are willing to discuss some flexibility in some cases. If you have a timing conflict, please contact us in advance to discuss some options. Similarly, if a sudden illness or accident strikes, contact us as soon as you are able to. We will discuss your turn-in options from there.

Course Policies

Academic Accommodations

If at any point you are experiencing difficulties with some aspect of the course, please contact the course staff. Our goal is for you to learn the course material, and we will do what we can to help. In general, you can address course difficulties through any of the following options:

- To request personal academic accommodations due to a disability, please contact Disability Resources for Students: 448 Schmitz, 206-543-8924 (or 206-543-8925 for TTY). If you have a letter from DR Sindicating that you have a disability which requires academic accommodations, please present the letter to me so we can discuss how to meet your needs for this course.
- Contact the instructor to arrange a meeting. I will discuss with and work with you to address your access issues. Together, we will identify ways to address these issues as best as we can.
- Provide feedback and/or suggestions through the course's anonymous feedback form.

Written Homework Guidelines

For written homework assignments, typed assignments are preferred. Handwritten is also fine, but it is often more difficult to organize your work clearly and legibly so that the TA can give a good grade.

Some problems on the written assignments ask you to give an algorithm to solve a problem. Unless the assignment specifically tells you to implement the code and run it, pseudocode is acceptable. Pseudocode means that you do not need to write every line in Java with correct syntax. English explanations of operations are more than acceptable. The general rule you should follow is that you can substitute English for any *O(1)* operation, but not for more complex steps.

Thus, the following would *NOT* be acceptable:

```
scan the list and count all elements greater than x
```

while the following would be acceptable:

```
while list has more elements
    increment counter if current element is greater than x
    move to next element of list
```

The idea is that you do not need to give all the nitty-gritty coding details (that is the purpose of the programming assignments), but you should demonstrate a clear understanding of what your algorithm does and where those nitty-gritty details would need to be.

Programming Guidelines

For the programming projects, your primary goal is to write code that runs correctly on legal input and gracefully rejects flawed input. Your code will also need to follow the provided design specifications. This will mean achieving the indicated time and space efficiencies as well as implementing everything required. This might require you to code a data structure that is already available in a Java Library. Finally, your submitted code should reflect clarity in its formatting, naming conventions, and documentation.

These guidelines for programming are formally described on the Programming Guidelines page. Every student should familiarize themselves with the contents of that page.

Regarding Policy

If you have a question about an assignment or exam that was returned to you, please do not hesitate to ask a TA or the instructor about it. Course staff are humble and will admit their own mistakes just as long as you respect your own errors. Furthermore, in talking about any mistakes you might have made in an assignment, you will have the opportunity to further understand the concepts. Learning from one's errors is often a very memorable way to learn.

Collaboration and Academic Integrity

Your instructor and your fellow students expect and deserve a basic respect for the integrity of this course and an environment where we can all focus on learning. The following paragraphs establish a clear understanding of what we all will do, with the expectation that it will never be an issue.

We want you to learn from your fellow students and discuss the course material, but the work you complete must be your own. You should NEVER share complete solutions with other students for any assignment. Unless we specifically state otherwise, we encourage collaboration on "individual" homework, provided you follow the *Gilligan's Island* rule:

1. You spend at least 30 minutes on each and every problem alone, before *discussing* it with others.
2. Cooperation is limited to group discussion and brainstorming. No written or electronic material may be exchanged or leave the brainstorming session.
3. After collaborating, you do something mind-numbing or otherwise non-technical for at least 30 minutes (e.g., watch an episode of Gilligan's Island or pretty much any reality TV show).
4. You write up each and every problem in your own writing, using your own words, and fully understanding your solution.
5. You identify each person that you collaborated with at the top of your written homework or in your README file.

If you are ever unclear about how to represent what work you have done, please ask the instructor or a TA. Furthermore, just makse sure to describe clearly what you have done. If you do, the worst that will happen is you that may lose some credit on an assignment. This is much better than being caught cheating.

Cheating and other forms of academic misconduct are a very serious offense. Copying someone else's homework is cheating, as is copying the homework from another source (the web, other classes, previous course offerings, etc.). If you are caught cheating, you can expect a failing grade and initiation of a cheating case in the University system. Cheating is an insult to other students, to the instructor, to the department and major program, and most importantly, to yourself. If you feel that you are having a problem with the material, or do not have time to finish an assignment, or have any number of other reasons to cheat, then talk with the instructor. We want you to learn the material and will help you do just that. However, the caveat is that you **respect** our efforts, your classmates, and yourself. Copying the work of others is not the solution and violates that respect.

To avoid creating situations where copying can arise, never e-mail or post your solution files. You can post general questions about interpretation and tools but limit your comments to these categories. If in doubt about what might constitute cheating, send the instructor email describing the situation. For more details, you should familiarize yourself with the CSE deparment's Academic Misconduct web page and the University's Statement on Student Academic Responsibility (pdf).

Programming Guidelines, CSE332 Summer 2012

Grading

For each project the, approximate and subject-to-change grade breakdown is:

Program correctness and Compilation:	40%
Architecture/Design, Style, Commenting, and Documentation:	30%
Writeup/README:	30%

The reason why "so few" points are allocated toward program correctness and error-free compilation is because CSE332 students are accomplished enough to know how to get their code to compile and run against the general input (although testing "boundary conditions" is a skill that students should aim for). Program correctness and error-free compilation is neither a fair nor discriminating measurement of project quality.

The two biggest discriminating factors among CSE332 students are program design (such as style and architecture) and analysis (the README/writeup), which is why these factors are heavily weighted. CSE332 is a course about data structures and the tradeoffs made during algorithm/data structure/abstraction design, so putting additional weight on program design, and questions about algorithm analysis and weighing tradeoffs, is more in keeping with the course goals.

Platform

You should use Java 7, making appropriate use of generics and JUnit. You may choose to work in a standard text editor or you may use an IDE. Course staff will do our best to support either the **Eclipse** or **jGRASP** IDEs. How you code is up to you. All the software you need for the course is available for free and is pre-installed on the machines in the department's undergraduate computing labs (Windows or Linux).

Robustness

Your primary task is to create a working, robust program. This means that your program should produce correct answers on all legal input and produce comprehensible error messages on invalid input. Keep in mind that unreasonably long running time is probably an error (unless otherwise mentioned). How should you ensure robustness? At the very least, try to complete (without problems) the following steps before submitting your code:

- Compile without warnings or errors. (Notify us of any errors in the code we provide.)
- Run correctly on all test data we give you. (Tell us if you find an error in the provided test data. We are not infallible.)
- Run correctly on test data of your own which has:
  - Difficult cases
  - Boundary cases
  - Minorly incorrect input
  - Egregiously incorrect input
  - An empty input file
- Run correctly on test data from another student. The discussion board is a great place to share test data. If you use someone else's test data, you **must** credit this group in your writeup.

This doesn't guarantee robustness, but it's a good start! Turn in your test cases; they may help us grade your work better (read: give you the maximum possible amount of credit).

Coding Clearly

You should *always* make an effort to write code that is easy for other people to read. In the real world, this is an

important skill that is valued higher than cleverness or using the least number of lines/characters possible. In this class, it is important so that we can understand your program and grade it fairly. By reading through your code and comments, we should be able to figure out how your code is organized and why it works (or fails). If we cannot, we will likely deduct points.

Commenting: One aspect of writing clear code involves commenting your code so that it is clear what it does without reading it line by line. Comments should be used:

- at the top of each file to tell what code it contains
- at the top of each method to tell what it does
- at the declaration point of important variables
- on or before any lines whose purpose is non-obvious.

Variable names: Variable names should conform with Java conventions (e.g., myVariableName, MyClassName, MY_CONSTANT_NAME) and should give a good idea of what purpose that variable serves in your program.

Coding simply: Although Java allows it, there is no benefit to writing overly complicated statements such as:

```
if ((i = myfunc(counter++)) < num_iterations) { ... }
```

when they could easily be rewritten in a clearer manner:

```
i = myfunc(counter);
counter++;
if (i < num_iterations) { ... }
```

Coding should not involve grandstanding or distracting cleverness. It should be like writing English sentences that you truly want someone to understand, including yourself. Try to keep in mind the **Three-Three Rule**:

> *You should always write code such that you should be capable of understanding your code when reading it three months later at 3AM in the morning.*

Speed

As long as your program takes a reasonable amount of time and your data structures and algorithms have the correct time and space complexity, we are not interested in small constant-factor differences. **Robustness and clarity come first!**

Using Java Class Libraries

Unless specifically approved or directed to, you should **not** use Java class libraries in your programming assignments to implement data structures and algorithms that are your responsibility. This rule is admittedly in contrast to many situations in the real world, but a large goal of this course is to fully understand core data structures that are so widely useful that Java provides them in its standard library. Often such understanding comes from implementing them yourself.