COURSE TITLE: Introduction to Software Development Methods

REQUIRED BACKGROUND: C- or higher in CS 101. Expected prerequisite is a CS1 course using an Java and an Object-Oriented (OO) approach. This course must address the topics of algorithmic problem-solving, variables, streams, objects, classes and types, control constructs (if, switch, loops and break), constructors, arrays and collections.

TEXTBOOKS:
2. Kathy Sierra, Bert Bates: *Head First Java*

COURSE OBJECTIVES:

Students at the end of the course will:

1. Comprehend more **advanced principles of object-oriented programming** and how a programming language supports these, and apply these by developing larger and more complex programs than in their first programming course. (Topics include polymorphism, inheritance, interfaces, etc.)
2. Comprehend and apply **principles of design at the class and object level**. These principles include abstraction, encapsulation, and information hiding. This also includes the ability to define and evaluate class interfaces to solve specified design problems, as well as the ability to understand, apply, and evaluate the use of reusable components to solve such problems.
3. Comprehend and analyze **problems and programming issues** such as dynamic memory management, indirect object references, and recursion. Also, be able to apply this knowledge by implementing software that includes these features.
4. Apply **knowledge of software development practice** to effectively use tools and environments such as interactive development environments, debuggers, unit testing, acceptance testing, version control, etc.
5. Comprehend **important basic concepts of software engineering** and the development of large software systems, including the software lifecycle, requirements, design, and software quality. In their development activities, students will be able to apply basic unit testing and carry out a software inspection.
6. Comprehend the basic **principles of the architecture of larger software systems**, in particular, object-oriented frameworks. Apply this knowledge by developing a GUI using a framework.

For further information on course specifications, see the course specification at the end of this memo.
COURSE WEB PAGE: http://www.cs.virginia.edu/~cs201. Check this webpage daily. Everything about the course, including this syllabus, resides there. News and updates will be posted prominently, and it is assumed that you will read them regularly.

INSTRUCTOR: Christopher W. Milner

OFFICE: Olsson Hall, 228A.

HOURS: (subject to change, check the web page) Likely hours will be: Tuesday 10-11AM, Thursday 11-noon, Friday 1-2PM and by appointment. E-mail read daily.

TAs: TBA
TA Hours: Will appear on class webpage

TA email: cs201@cs.virginia.edu
Please send email to cs201 for all questions about labs, pre- and post-lab exercises, grades, etc.

Instructor Email: Spring2015.cmilner@spamgourmet.com
Please use this ONLY for appointments and emergencies. This account only accepts email from UVA accounts. Email from yahoo, hotmail, gmail, etc. may be bounced!

TEAM PARTICIPATION: Some portion of each lab will be devoted to working in teams. Exercises may include pair programming (and evaluating pair programming), estimating time to complete programming tasks, learning to take specifications from clients (TAs), learning to break specifications into tasks, small experiments (spikes), and group reviewing of code, etc.

Project work will be done as a team and accounts for 15% of your grade. Students failing to participate in teamwork may be removed from the team and will not get credit for team exercises.

EXAMS: 3 exams, a Lab Exam and a Final Exam. Unless otherwise noted, exams are closed-books, notes, etc.

LABS: Labs will be held weekly in Olsson 001. Please print out the lab before coming to class. Exercises designated as Pre-Labs are due at the beginning of your lab. Lab time will be spent in the following activities (in no particular order): programming quizzes to be completed on the computer (10-20 minutes), meeting with your team to inspect pre-lab code and or talk to your TA / coach / customer (10-20 minutes), programming activities (20-60 minutes), small lectures or demonstrations by TAs (10-15 minutes).

EVALUATION: Class participation (including reading quizzes) counts 5%. Exam 1 counts 6%, exam 2 counts 7%, exam 3 counts 8, the lab exam counts 15 and the final exam counts 9%. Labs, including pre- and post-lab programming exercises, in-lab quizzes and in-lab programming and team exercises count 40%. The team-based project counts 10%.
LNEC: if you have been identified as an LNEC student, please let the Center know you are taking this class. If you suspect you should be an LNEC student, please schedule an appointment with them for an evaluation. I happily and discretely provide the recommended accommodations for those students identified by LNEC.

HONOR POLICY:

1) Unless explicitly noted on an assignment, all work is to be done individually.
2) You must take proper precautions to guard your work so that others may not retrieve it from public workstations or printers, alter it and submit it as their own.
3) In-lab quizzes and in-lab exams must be taken without notes or personal files retrieved from websites or your home directory.
4) In-lab quizzes and the in-lab exam must be taken in the lab. Submitting solutions outside of lab, or after the quiz or exam is over is not permitted.
5) If you inadvertently download some else's solutions, you must immediately erase the file and then email the cs201 account to let them know of your mistake.
6) You may not discuss the contents of in-lab quizzes or exams with other students until the Friday following the quiz or exam.

Before the due date, you may talk with others in your team about your homework assignments. During these discussions, you may not display or exchange actual code for the solution. You may write notes, if you wish, but these must be conceptual and not actual code solutions. If you consult with other students on the assignment, you must acknowledge, in a README.txt file, where such consultation helped you. If you find that you need more help than simple discussions, please see a TA during TA office hours. Please start your assignments early so that you can see a TA if you need help.

You must take proper precautions to guard your work. This includes removing your work from public workstations once you have finished. If two people turn in the same code (or code that only varies on trivial ways but that is clearly the work of one person) both parties will receive a zero (0) for the assignment. Any student involved in more than one case of intentional or negligent "sharing" may be subject to referral to the honor system.

PLANNED COURSE OUTLINE:
See the webpage: http://www.cs.virginia.edu/~cs201.

ADDITIONAL NOTES:

The E-School Dean has very explicit rules about missing or rescheduling exams. We will abide by those rules. If you are going to miss an exam, make every effort to notify me before the exam. Generally, I will not accept assignments that were not completed before the established deadline. Assignments late due to good medical reasons or other extraordinary circumstances will be accepted.
## Tentative Schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Days</th>
<th>W reading</th>
<th>W lecture</th>
<th>Fri reading</th>
<th>Friday Lecture</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan 20,22</td>
<td>1.1</td>
<td>intro /life story</td>
<td>1.2</td>
<td>software development</td>
<td>intro/lifecycle</td>
</tr>
<tr>
<td>2</td>
<td>Jan 24,26,28</td>
<td>2.1, slides</td>
<td>user stories, tasks, estimating, prioritize</td>
<td></td>
<td>what is an object?</td>
<td>2.2</td>
</tr>
<tr>
<td>3</td>
<td>Jan 31,Feb 2, Feb 4</td>
<td>2.3, slides</td>
<td>pairing, code ownership, test first!!</td>
<td></td>
<td>finding classes (arrays and vectors)</td>
<td>2.3.2</td>
</tr>
<tr>
<td>4</td>
<td>Feb 7, 9, 11</td>
<td>3.1-3.2</td>
<td>debugging</td>
<td></td>
<td>classes in java, access levels</td>
<td>classes in Java, where things live on the heap and stack, references.</td>
</tr>
<tr>
<td>5</td>
<td>Feb 14,16,18</td>
<td>3.3</td>
<td></td>
<td></td>
<td>inheritance in java</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Feb 21,23,25</td>
<td>Review for exam</td>
<td></td>
<td>EXAM 1 (objects, classes, debugging, inheritance (interface, abstract class), dynamic memory)</td>
<td>exam Lab on debugging and inheritance</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Feb 28, Mar 2,4</td>
<td>4.1-4.4</td>
<td>case study</td>
<td>4.5</td>
<td>case study</td>
<td>4.1-4.7 case study, simulation Lab on GUI and exceptions Drop Day MARCH 1</td>
</tr>
<tr>
<td>Week</td>
<td>Days</td>
<td>W reading</td>
<td>W lecture</td>
<td>Fri reading</td>
<td>Friday Lecture</td>
<td>Notes</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td>-----------</td>
<td>--------------------------------</td>
<td>-------------</td>
<td>-------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>8</td>
<td>Mar 14, 16, 18</td>
<td>6.2</td>
<td>recursion, linked lists</td>
<td>7.2, 7.3</td>
<td>lists // trees</td>
<td>linked lists and recursion Lab on streams, strategy pattern</td>
</tr>
<tr>
<td>9</td>
<td>Mar 21, 23, 25</td>
<td>7.3</td>
<td>trees / review for exam 2</td>
<td>EXAM 2 (linked lists, recursion)</td>
<td>trees Lab on linked lists and recursion Lab on streams, strategy pattern</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Mar 28, 30, April 1</td>
<td>13</td>
<td>networking</td>
<td>13 networking</td>
<td>Lab on exceptions and trees</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>April 4, 6, 8</td>
<td>11</td>
<td>threads</td>
<td>11 threads</td>
<td>threads Lab on networking</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>April 11, 13, 15</td>
<td>slides</td>
<td>code reviews</td>
<td>slides</td>
<td>code reviews, regular expressions Lab: more networking and code reviews</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>April 18, 20, 22</td>
<td>slides</td>
<td>security</td>
<td>slides</td>
<td>exam 3 review                       Lab: threads and formal code review</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>April 25, 27, 29</td>
<td>exam 3</td>
<td>(trees, iterative software development methods, software life cycle)</td>
<td>review, discuss final and in-lab exam</td>
<td>Project demo</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>&quot;May 2, 3&quot;</td>
<td></td>
<td></td>
<td></td>
<td>In-lab Exam</td>
<td></td>
</tr>
<tr>
<td>12-May</td>
<td></td>
<td></td>
<td><strong>FINAL EXAM</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# The UVA CS Curriculum: CS201 Specifications

## Lab Courses - CS201

### Software Development Methods

<table>
<thead>
<tr>
<th>Knowledge</th>
<th>Skills</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Engineering</strong></td>
<td><strong>Mastery</strong></td>
</tr>
</tbody>
</table>
| Mastery | • Information Hiding  
• Software Reuse Via Existing Classes  
• Use Of Class Libraries |
| Familiarity | • Good Programming Practices  
• Object-Oriented Design  
• Separate Compilation  
• Debugging  
• Representation Of Elementary Types  
• Simple Test Coverage Metrics  
• Configuration Management |
| Exposure | • Spiral And Iterative Enhancement Development Processes  
• Requirements Specification  
• Large System Design Recovery  
• Maintenance Costs |

<table>
<thead>
<tr>
<th>Mastery</th>
<th>Familiarity</th>
<th>Exposure</th>
</tr>
</thead>
</table>
| • Identification Of Design Decisions  
• Use Of Existing Classes  
• Retrieval Of Existing Classes |
| • Small System Design  
• Implementation Of Simple Designs  
• Developing Source Code With Good Style  
• Developing Test Cases |
| • Working From Requirements Specifications  
• Working With Large Systems  
• Development Of Suitable Processes |

### OS and Environment

<table>
<thead>
<tr>
<th>Familiarity</th>
<th>Exposure</th>
</tr>
</thead>
</table>
| • Debugging Tool  
• Profiling Tool  
• Source-Code Control Tool  
• Representation Of Basic Types And Instructions |
| • Locating Simple Errors  
• Measuring Statement Coverage |
| • Interpret Binary Representations Of Simple Types And Instructions  
• Recovering Existing Designs |
<table>
<thead>
<tr>
<th>Mastery</th>
<th>Mastery</th>
<th>Familiarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator Precedence</td>
<td>Implementation Of Simple Sequential Algorithms Requiring Up To 1,000 Lines Of Source Code.</td>
<td>References</td>
</tr>
<tr>
<td>All Iteration Mechanisms</td>
<td></td>
<td>Simple Data Structures - Lists, Queues, Binary Trees</td>
</tr>
<tr>
<td>Conditional Statements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streams</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference and Value Parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structures</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Familiarity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple Data Structures - Lists, Queues, Binary Trees</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Object-Oriented Programming

#### Mastery
- Classes, objects, and members.
- Access rules for private and protected members.
- Constructors and relevant knowledge of garbage collection.
- Generics (possible topic).
- Promotion by constructors.
- The standard IO library.
- References as function parameter.

#### Mastery
- Design a simple class with the necessary constructors and member functions.
- Design a simple container class using generics.
- Design constructors as necessary to perform promotions that will be needed for a given class.
- Parameter passing.

#### Familiarity
- Single inheritance.
- Virtual functions.
- Shallow and deep cloning and assignment operators.
- Memory allocation and memory leaks.
- Matching rules for calls to overloaded functions.
- References as function return types.
- Compiler generation of implicit calls to constructors.

#### Familiarity
- Design a hierarchy of classes.
- Derive a new class from an existing class.
- Provide and redefine functions where needed.
- Correctly implement the Clonable interface.
- Determine which of a set of overloaded functions will be invoked for a given function call.
- Design and implement member functions which return references.

#### Exposure
- Restriction in Java on multiple inheritance and use of interfaces to address this.
- Abstract classes and interfaces.
- Private inheritance.
- Static members.
- Implementation of virtual function calls.
- Initialization lists for derived classes.

#### Exposure
- In existing code, recognize the use of single inheritance and multiple interfaces, private inheritance, static members and initialization lists.