CCPS 209 Computer Science II, W2020

Ryerson Chang School of Continuing Education, Toronto, Canada

General

Lecturer: Ilkka Kokkarinen, email *ilkka.kokkarinen@gmail.com*. I do not currently have an office at Ryerson, so there are no office hours. Most problems can be solved by email, and for those that can not, we can set up a private meeting in the Chang School student lobby before the class.

A version of these lectures is available as a YouTube playlist "CCPS 209 Computer Science II".

Environment: This course uses BlueJ as the compulsory environment to submit your labs. Downloading and installing BlueJ will also conveniently install the latest version of Java for you.

Material: There is no official textbook assigned in this course. Instead, the official course material comprises the selected parts of the Oracle Java Tutorial listed below (available online for free), and the course notes prepared by the instructor. Both **midterm and final exams are open book** so that you can bring any non-electronic material (such as these notes) into the exam.

For a more lighthearted illustration of many topics of this course for all my fellow kids, check out the compilation of CCPS 209 memes. (If you think up and create a genuinely new meme that isn't already in any other programmer humour site, please send it to me for inclusion on that page with proper credit to you.)

All example programs that we go through in these lectures, and some example programs from the earlier Java version of CCPS 109, are available in the GitHub repository JavaExamples.

Grading (CHANGED MARCH 16): Your grade is determined by the straightforward formula $M + L + P$, where M is your midterm mark (out of possible 40), L is your lab mark (out of a possible 20), and P is your take-home Bulls and Cows Programming Project mark (out of possible 40).

Labs: The lab specifications and all automated JUnit testers are available in the GitHub repository CCPS209Labs. Attendance in the lab sessions late at night is not mandatory, but the lab problems can be worked on on your own time whenever convenient. The labs are due as one bunch at the end of the course. (The instructor will open up the submission mechanism on D2L about one month before the end of the course.)

Outline

The course meets weekly on Thursdays. Each class starts with a lecture that takes between three and four hours in the room **ENG102**, followed by a freeform lab session in the **ENG201** computer lab, for a total of five hours.

Lecture	***Date***	***Enables lab***
Imperative Java for Pythonistas (PythonToJava, DataDemo, ConditionsAndLoops, ArrayDemo, and you can also check out the instructor's old course notes for Java CCPS 109 to fill in some gaps)	Jan 16	0A, 0B, 0C, 0F
Classes and objects (Length, Fraction)	Jan 23	1, 2
Inheritance and polymorphism (Barnyard, InheritanceDemo, Overload)	Jan 30	4
Java Collection Framework (CollectionsDemo)	Feb 6	0D, 0E, 3, 11, 12
Swing GUI framework, graphics, event handling (ShapePanel, Counter, ImageDemo)	Feb 13	5, 9, 13, 14
Winter Study Week Feb 15 to Feb 21, no school		
Midterm Exam	**Feb 27**	
Exception handling machinery, pre- and postconditions, Liskov Substitution Principle, reflection (ExceptionDemo, ReflectionDemo)	Mar 5	
Stream I/O, text processing, association maps (GZip, WordFrequency, SomeUnixCommands, DissociatedPress)	Mar 12	6, 8
Some Java 5 tidbits, generics in Java (Java5Demo, Pair, GenericsDemo)	Mar 19	
Concurrency fundamentals (ConcurrencyDemo)	Mar 26	7
Concurrency control with semaphores and others (BigPrimes, MySemaphore, Sliders)	Apr 2	
Lambda expressions, computation streams (MapReduceDemo, ImageLoops, TakeUntil)	Apr 9	10
Final exam	**Apr 16**	

Material

The course material comprises the course notes and the following listed trails and lessons of the Oracle Java Tutorial for the Java language. Everything necessary for this course is already included in my handouts and example programs, so you don't actually need to get yourself a textbook, unless you actually want to get yourself a book such as *Big Java*.

Midterm exam topics: Object-Oriented Programming Concepts, Interfaces and inheritance, Collections, 2D Graphics.

Final exam topics: Exceptions, Basic I/O, Reflection, Concurrency, Generics, Lambda expressions, Annotations.

Another excellent additional learning resource are the clear and concise Jenkov's tutorials. For this particular course, especially the series "java.util.concurrent", "Java Exception Handling", "Java Generics" and "Java Reflection". (In fact, it actually might not be a bad idea to check out Jenkov's stuff before diving into the official Java course material.)

Additional material for the interested students after this course: I have done my best to steal into my notes and examples all the little gold nuggets from the good material that I have come across over the years, so that you don't have to waste your time to uncover all these precious little bits of value. However, there are a couple of websites and printed books that I would without hesitation recommend to anybody as additional reading material about both Java and object oriented programming in general.

To get the ball rolling, your instructor certainly enjoys reading on his iPad various course materials from MIT OpenCourseware. Most relevant for our course, you might want to check out their 6.005 Elements of Software Construction that expands on many issues that were merely touched here in CCPS 109 and 209. There are two versions of the lecture notes, a lighter and better formatted set from 2016 versus a deeper set from 2008 for you to take your pick.

Many issues and topics of this course are humorously illustrated in the short essay collection The Codeless Code, written in the style of zen koans.

If you are going to read only one Java book in your life, just skip all the hucksters and middlemen by making that book to be "Effective Java" by Joshua Bloch, now updated to its third edition to cover the features of Java 8 and 9. This fantastic book is basically the goal that I am trying to steer this course to converge towards, but this book still says pretty much everything much better than I could ever say.

"Java Puzzlers" by the same author illustrates many issues in language design in a more lighthearted form. The book website has some sample chapters and all the puzzles freely available for download. Each puzzle works the exact same way: without compiling and running the given piece of code, simply read it through and try to reason what it will actually do when run. Then compile and run it for real to see if you were right. The correct answers will both shock and educate you.

In general, I would venture to proclaim that you can measure your understanding of programming and computer science pretty accurately just by counting the number of "Epigrams" whose meaning and purpose you can explain and illustrate with examples while standing on one foot. (Note how all these observations date from the seventies and early eighties, and yet are still as timeless as ever.)

Object-oriented design is a great way to model reality, but unfortunately all models of reality are fundamentally imperfect, and trying to coax the complexity of human existence into bits and bytes and two-way decisions might be far trickier than it initially seems, as illustrated in the Curated List of Falsehoods Programmers Believe In, originally kick-started by the list of Falsehoods Programmers Believe About Time.

Another great blast from the past is the original and therefore pretty austere WikiWikiWeb whose expert discussions and insights on various topics of software engineering and computer science you could casually browse with a glass of wine in the same spirit as you might browse TV Tropes. Unlike in Wikipedia and Stack Overflow, it is not quite as easy to find what you are looking for, but a good starting point would be CategoryCategory. Scroll down to pick a topic and go on randomly from there by clicking on whatever looks interesting. (To access the actual contents for each category, you have to click the title of the small page: this might be a bit confusing first.) For the needs of our course CCPS 209, of course a good first starting point would be CategoryJava, but some more interesting starting points might be CategoryPattern or CategoryAlgorithm.