Course Wrap-up
CSC207 Fall 2015
package basics;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}

10 weeks ago...
Learning a new language

• A new memory model.
• Static typing rather than dynamic typing.
• Compiled rather than interpreted.
• Primitives vs. only objects.
• Everything belongs to a class.
• Generics.
• Interfaces and abstract classes.
• A philosophy oriented towards safety.
• A huge API.

You also did more of the learning yourself than before.
More languages

You’ll see more languages and the underlying language design principles and approaches:

- csc209: Software Tools and Systems Programming
- csc324: Principles of Programming Languages
- csc343: Introduction to Databases
- csc309: Programming on the Web

Value:

- knowing which tool to apply in a given situation
- deeper appreciation of a language’s design and how best to use it
Safety

Much of how Java does things (types, casting, etc.) is intended to make programs safer.

Safety was a key principle behind the design of Java.

Safety came up again when we learned about floating point.

Learn more about floating point and numerical algorithms in csc336: Numerical Methods.
Tools and Technologies

- A new technology: Android development
  - A new API
  - Extending Java knowledge
  - Emulator
- New software development tools:
  - Two (!) fully-featured IDEs
  - Version control
Design

We stepped up several levels from designing functions, methods and classes, to design whole software systems.

CRC cards provided a way to brainstorm and communicate about design at a very high level.

UML provided a way to specify a design, still at a high level and independent of language.
Design Patterns

Language-independent.

Generic solutions to recurring design problems.

Solutions that follow the object-oriented design principles and best practices.

You’ll see more design patterns in csc301: Introduction to Software Engineering.

The algorithmic analog: algorithmic design techniques e.g., divide and conquer, and greedy algorithms. You’ll learn these in csc373: Algorithm Design, Analysis, and Complexity.
Regular expressions

These have many practical applications:

- Parsing input in a Java program (or another language).
- Many Unix commands, such as `grep`, use regular expressions.
- Regular expressions and related concepts are important in language design and compilers.

**csc488: Compilers and Interpreters**

Learn more about the underlying theory in **csc236: Introduction to the Theory of Computation**, and **csc448: Formal Languages and Automata**.
Software Development Process

Working in teams.

Iterative and incremental software development.

Changing requirements.

Elements of state of the art software development practices, such as Scrum and XP.
Software Development Process

You will learn much more through:

csc301: Introduction to Software Engineering

csc302: Engineering Large Software Systems

project courses (csc494/495 or csc490)

UCOSP: undergraduate capstone open-source project

Learn the formal underpinnings of testing in csc410: Software Testing and Verification.
End of Term Stuff
Please complete your course evals!

I will post a new schedule for instructor office hours next week.

I have not forgotten the midterm remarks. I will be in touch.

Don’t forget to do your team evals right after Phase III.
CDF

Due to a power shut-down, CDF will be completely down from 4pm Thu 10 Dec to 10 am Fri 11 Dec.

Includes MarkUs (that means your repos too) and many course websites.

I will mirror the csc207 website elsewhere, and post a link on Piazza.

Piazza will not be affected.
The exam

- 3 hours
- closed book, but there is an API at the end
  - it will be posted before the exam
- there is also some scrap paper at the end
- Javadoc and internal comments not required
- import statements not required
- helper methods always welcome
- assume all input is valid
Possible kinds of questions

Code writing

Tracing and understanding code where the memory model helps you predict results

Multiple choice and short answer questions

Testing: choosing test cases, implementing in JUnit, unit testing concepts

Design patterns: recognize, explain, write code with singleton, observer, iterator (but not strategy)

Regular expressions: write and read them

svn
Possible kinds of questions

Javadoc: write, read, critique it

CRC: read and understand it, solve a design problem

UML: read and understand it, perhaps write a little

Development process: answer questions concerning
• agile concepts such as test-driven development
• some perspective on your team project experience
• awareness of how/where the tools and processes are beneficial
Ideas for how to prepare

Solve old tests and exams.

Revisit the assignment and exercises. Learn from what you did previously; improve it.

Write code that tries to poke holes in your understanding of Java concepts (name lookup, exceptions, etc.).

Write regular expressions and test them in Java.

Write code that uses the design patterns we studied.

Write code on paper.

Think about how you used scrum, TDD, version control and code review on the project. What did you learn?
All the best!