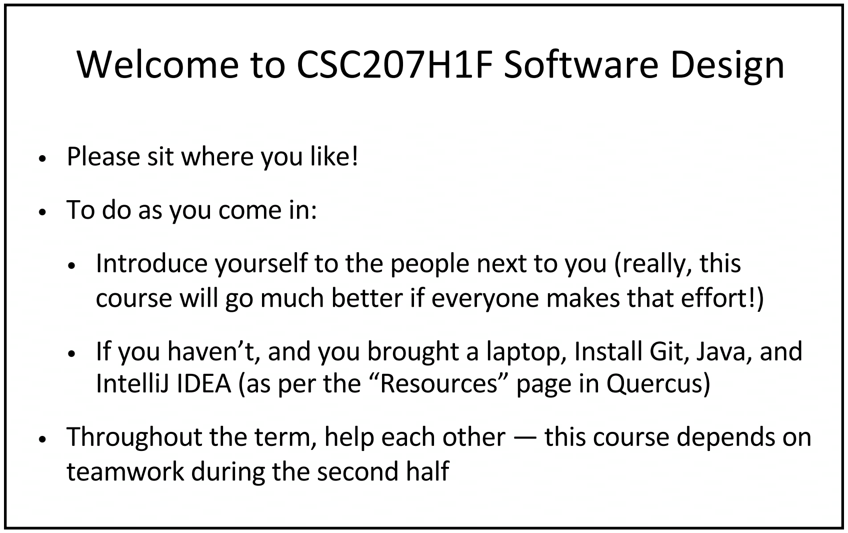
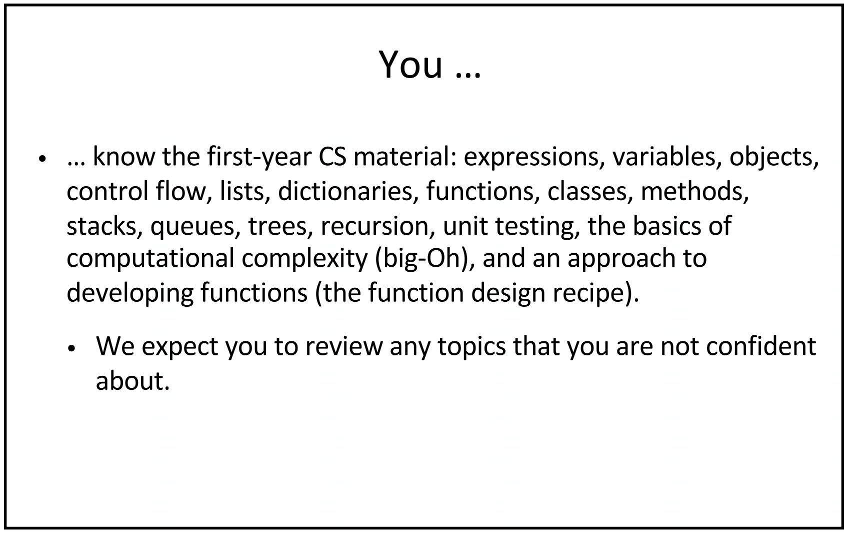


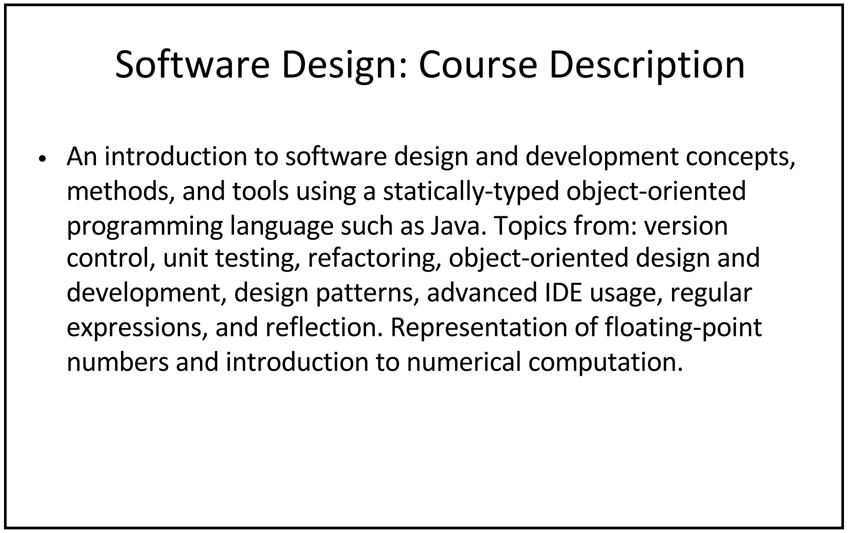
Welcome to CSC207H1F Software Design

- Please sit where you like!
- To do as you come in:
  - Introduce yourself to the people next to you (really, this course will go much better if everyone makes that effort!)
  - If you haven’t, and you brought a laptop, Install Git, Java, and IntelliJ IDEA (as per the “Resources” page in Quercus)
- Throughout the term, help each other — this course depends on teamwork during the second half



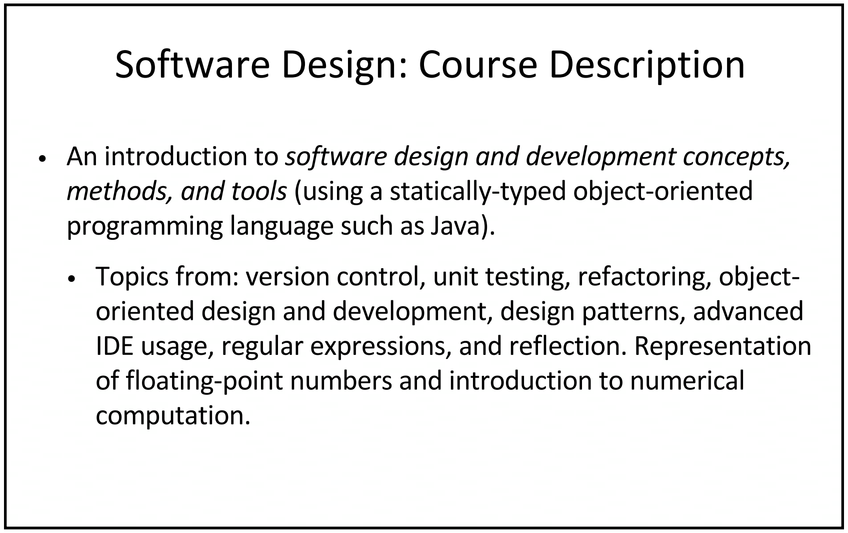
You …

- … know the first-year CS material: expressions, variables, objects, control flow, lists, dictionaries, functions, classes, methods, stacks, queues, trees, recursion, unit testing, the basics of computational complexity (big-Oh), and an approach to developing functions (the function design recipe).
  - We expect you to review any topics that you are not confident about.



Software Design: Course Description

- An introduction to software design and development concepts, methods, and tools using a statically-typed object-oriented programming language such as Java. Topics from: version control, unit testing, refactoring, object-oriented design and development, design patterns, advanced IDE usage, regular expressions, and reflection. Representation of floating-point numbers and introduction to numerical computation.



Software Design: Course Description

- An introduction to *software design and development concepts, methods, and tools* (using a statically-typed object-oriented programming language such as Java).
  - Topics from: version control, unit testing, refactoring, object-oriented design and development, design patterns, advanced IDE usage, regular expressions, and reflection. Representation of floating-point numbers and introduction to numerical computation.

Primary learning objective: understand and apply professional software design techniques

- You will learn to design and write an *easy-to-read*, *hard-to-break*, *maintainable*, *efficient* program in a team environment.
- Software design has you use a set of principles and techniques to achieve this.
- This is the primary learning objective for this course. *Every course topic is motivated by this.*

Subtopic: learn to program in an object-oriented statically-typed programming language (Java)

- Strong typing
- Inheritance
- A memory model for Java (it's a lot like Python's!)
- Information hiding
- Unit testing
- File handling
- Exception handling
- Syntax errors vs. runtime errors

Subtopic: design and development techniques used professionally

- How to think about, design, and implement a large program in a team of programmers
- How to analyze requirements
- How to design code so that it can be safely refactored
- Common implementation choices: design patterns
- How to keep track of changes made to a program (version control using Git)
- An Integrated Development Environment (IntelliJ)
- Unit testing

Working in a team

- Imagine you've graduated and are working in industry. You are working in a team on a large project over many months. All of you are contributing to the same code base, and over time will be editing many different parts of the project.
- What qualities would you want in the code that everyone contributes? (Remember, you will have to edit everyone's code.)
- What behaviours would you want in your team members?

History of Design Principles

- Industry professionals have, over decades, converged on a set of tips and techniques for making programs that are easy-to-read, hard-to-break, maintainable, and efficient program in a team environment.
- We call these *design principles*.
- There is no rulebook, but software that adheres to as many of these design principles as possible is generally considered *better*.
- This is a theme that runs throughout this course.

Object-oriented programming language features: *abstraction*, *encapsulation*, *inheritance*, and *polymorphism*

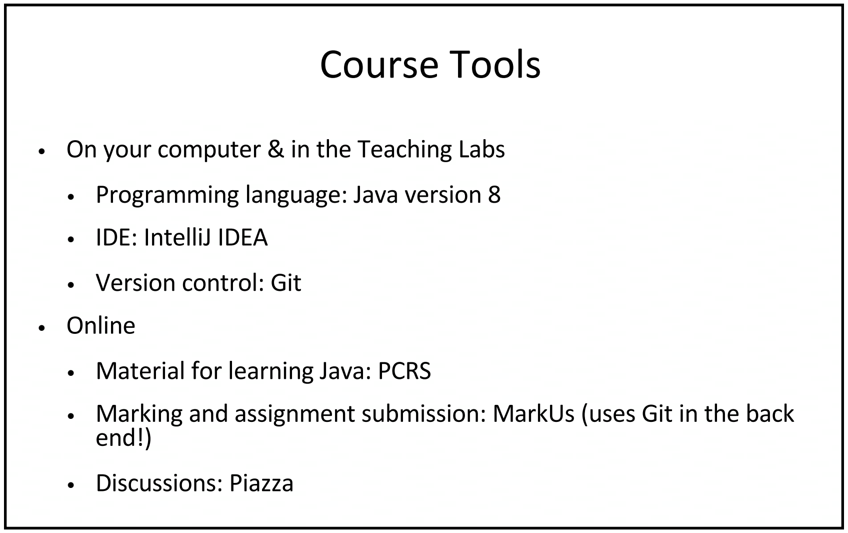
- *Abstraction* — the process of distilling a concept to a set of essential characteristics.
- *Encapsulation* — the process of binding together data with methods that manipulate that data, and hiding the internal representation.
  - The result of applying abstraction and encapsulation is (often) a class with instance variables and methods that together model a concept from the real world. (Further reading: what's the difference between Abstraction, Encapsulation, and Information Hiding?)
- *Inheritance* — the concept that when a subclass is defined in terms another class, the features of that other class are inherited by the subclass.
- *Polymorphism* ("many forms") — the ability of an expression (such as a method call) to be applied to objects of different types.

Object-oriented design concepts: *coupling* and *cohesion*

- *Coupling* — how much a class is directly linked to another class.
  - *High coupling* means that changes to one class may lead to changes in several other classes.
  - *Low coupling* is, therefore a desired goal.
- *Cohesion* — how much the features of a class belong together.
  - *Low cohesion* means that methods in a class operate on unrelated tasks. This means the class does jobs that are unrelated.
  - *High cohesion* means that the methods have strongly-related functionality.

Fundamental object-oriented design principles

- SOLID: five basic principles of object-oriented
  - Developed by Robert C. Martin, affectionately known as "Uncle Bob".
- **S**ingle responsibility principle: A class should have one, and only one, reason to change.
- **O**pen/closed principle: Classes should be open for extension but closed for modification.
- **L**iskov substitution principle: Subclasses should add to a base class's behaviour, not replace it.
- **I**nterface segregation principle: Many client-specific interfaces are better than one general-purpose interface.
- **D**ependency inversion principle: High-level code shouldn't depend on low-level code. Both should depend on abstractions. In addition, abstractions shouldn't depend on details. Details depend on abstractions.



Course Tools

- On your computer & in the Teaching Labs
  - Programming language: Java version 8
  - IDE: IntelliJ IDEA
  - Version control: Git
- Online
  - Material for learning Java: PCRS
  - Marking and assignment submission: MarkUs (uses Git in the back end!)
  - Discussions: Piazza