# CompSci 230
# Software Construction

Course Revision: Themes A, B & C

S1 2015

# Overview

- In Stage 1, you learned how to write programs to solve small problems.
  - In CompSci 230, we teach programming "in the large".
- Large software systems have many stakeholders.
  - What will its users want?
  - Can we describe user requirements, accurately and succinctly?
- Large software systems are very complex.
  - Can we describe the design of a complex software system, accurately and succinctly?
  - Can we be sure that a complex system will do what it is designed to do, and that it will not do anything unintended?
- In CompSci 230, you will learn some incomplete answers to these difficult questions.
  - I will also attempt to teach you how to "learn how to learn" the technical skills you will need in the future – as a competent computer professional.

# Syllabus

- Four Themes:
  - A. The object-oriented programming paradigm
    - Object-orientation, object-oriented programming concepts and programming language constructs – because, for many important problems, OO design is a convenient way to express the problem and its solution in software.
  - B. Frameworks
    - Inversion of control, AWT/Swing and JUnit – because many important "sub-problems" have already been solved: these solutions should be re-used!
  - C. Software quality
    - Testing, inspection, documentation – because large teams are designing, implementing, debugging, maintaining, revising, and supporting complex software.
  - D. Application-level concurrent programming
    - Multithreading concepts, language primitives and abstractions – because even our laptops have multiple CPUs. Dual-core smartphones are now available...

# Theme A: The OO Design Paradigm

- Object-orientation, object-oriented programming concepts and programming language constructs – because, for many important problems, OO design is a convenient way to express the problem and its solution in software.
- Topics (by lecture slides):
  - 01: Intro to Java
  - 02: Hello World!
  - 03: Intro to OOD
  - 04: Use cases
  - 05: OOD concepts: abstraction, …
  - 06-08: Java Implementation

# Software Construction

- Review (or learn for the first time?)
  - What is Object-Oriented Programming?
    - Related objects in classes.  State + behaviour.  Instantiation.  Comparison with procedural and data-architectural styles of programming.
  - Classes & Objects
    - Message passing by calls, returns, and exceptions
  - Variables & Methods (for instances and classes)
- Introduction to OO Design
  - A process:
    1. determining what the stakeholders require,
    2. designing a set of classes with objects which will meet these requirements,
    3. implementing, and
    4. delivering.
  - You learned two new languages:
    - Use-case diagram, for requirements
    - Class diagram, for design
    - Object diagram, to explain "what's happening" in an implementation
      - □ not emphasised, but may be very helpful for your understanding

# Use Case Diagrams

- Learning goals for this unit:
  - **Interpretative:** Any student who passes CompSci 230 can accurately interpret the information presented in a use-case diagram or description.
  - **Productive**: Any student with a B or better in CompSci 230 can draw up an accurate set of use cases from an informal specification.
  - **Creative**: Excellent CompSci 230 students are able to apply their course-specific knowledge in novel situations. For example, they could discuss the strengths & weaknesses of use case analysis as a methodology for requirements capture.
- Note: I cannot test a students performance on all topics, at all levels, in an hour.
  - The final exam has some questions that are focused at A-level, some at B-level, and some at C-level. I won't reveal the levels at which topics are tested.
  - Some topics won't be tested at all, but I won't reveal which ones.
  - Such incomplete (and secret) coverage allows a limited range of quality-assurances e.g.
    - Any student who knows all important topics "at B level" will get a B.
    - Some B/C-level students will "get lucky" – they'll also get a B.
    - Students who have only C-level knowledge will get a C.
      - It is impossible to write in a language if you can't read it. You must be able to read & write in order to express novel thoughts.

# OOD & Class Diagrams

- Abstraction:
  - The ability of a language (and a designer) to take a concept and create an abstract representation of that concept within a program
- Information Hiding:
  - How well does this language, designer, and programmer hide an object's internal implementation?
- Polymorphism:
  - How does this language let us treat related objects in a similar fashion?
- Inheritance:
  - The "is-a" relation: important for code reuse.
- Composition, Aggregation, Association:
  - Types of "has-a" relations: ways to build complex classes from simpler ones. (I'm emphasising only the most general case: the "association".)

# Java Implementation

- Interfaces and Abstract Classes
  - Important in practice, but not emphasised this semester.
- Java's type system: Static & dynamic typing, conversions.
  - Very important in practice, rather difficult in theory.
- Visibility
  - Important in practice, but not emphasised this semester.
- Overriding, hiding (this is usually evil ;-), shadowing, overloading
  - Java syntax: `super`, `this`, `final`. (Static vs instance methods; name conflicts)
- Type conversions
- Enums
- Java's runtime system
  - A very "deep" topic. We skimmed over memory allocation.
- Object identity, assignment, equality, copying,
  - Very important in practice, with a straightforward theory after you understand instantiation (which is moderately complex: object diagrams might help).

# Theme B: Frameworks

  - Inversion of control, AWT/Swing and JUnit – because many important "sub-problems" have already been solved: these solutions should be re-used!
- Topics (by lecture):
  - 09: Collections
  - 10: Introduction to Swing
  - 11: Applets and AWT
  - 12: Swing and MVC
  - 13: Custom widgets and drawing

# Collections

- (Why use a framework? What is a framework?)
- The `Collection` interface
- Sub-interfaces:
  - `List`
  - `Set`
  - You know a little about `Map`; there are others, but you haven't used them
- I don't expect you to remember the details but you should know what operations "make sense" for the interfaces.
  - You should know how to implement a traversal using a for-loop (but we didn't explore Iterators)
- Implementations: `ArrayList`, `LinkedList`
- Generic types, e.g. `ArrayList<Integer>`

# Swing and AWT

- Only a few concepts here:
  - windows, components, containers,
  - Model-View-Controller; Swing's separable model-view.
- I don't expect you to implement a Swing app "from scratch", but you should be able to interpret a simple code and modify it.

# Exam Format

- 30% short-answer:
  - Allow 45 minutes for this part.
- 70% defined response (multiple-choice, true-false).
  - About 55 questions, allow 60 minutes for this.
  - There is one correct answer.
  - If it seems ambiguous, please write a note on the overflow page.

# Best wishes, and please keep in touch!

- I have enjoyed teaching this course.
- I'd enjoy hearing from you in the future.
  - Please don't hesitate to "volunteer yourself" to give a guest lecture to a future CompSci 230 class!