Welcome!

Who we are:

Kim Bruce
Alexandra Papoutsaki
David Ahia
Bradley Bain
Emily Chen
Victor de Fontnouvelle
Samuel Gearou
Daniel Rosenbaum
Erinna Woo

Why take CS62?

• How to implement algorithms and data structures in Java.
• How to design large programs (in object-oriented style) so that it is easy to modify them.
• How to analyze complexity of alternative implementations of problems.
Sample Problems

• Find the shortest path from Claremont to Chicago on interstate system (and do it efficiently).

Sample Problems

• Schedule final exams so there are no conflicts.

Sample Problems

• Design and implement a scientific calculator.

Sample Problems

• Design and implement a simulator that lets you study traffic flow in a city or airport.
Your responsibilities

• Skim reading in advance of lecture.
• After lectures, review notes and study examples carefully until you understand them.
• Come to labs prepared.
• Don’t remain confused. Faculty and TAs are here to help.
• Follow academic integrity guidelines

Assignments

• Lab work:
  • Learn tools and prepare work for weekly assignments.
  • Lab attendance is mandatory! No lab today!!
• Weekly assignment is separate
  • Programs generally are due on Sunday nights.
  • See late policy on syllabus. 3% penalty per day late.
• Daily homework
  • Not collected, but often on regular Friday quizzes.
  • No quiz this Friday!

Text

• Java Structures, 7th edition, by Duane Bailey
  • Available online for free
  • http://www.cs.williams.edu/~bailey/JavaStructures/Book.html
• Various online resources

Slides

• Will generally be available before class
  • With code, where applicable
• Designed for class presentation, not for complete notes.
• Will need to take notes (perhaps on slides).
• No laptops or other electronics open in class
  • If you have a disability affecting this, come see me.
Prerequisite

• Officially, CS 52 at Pomona
• Knowledge of Java equivalent to CS 51 at Pomona or CMC or the AP Test with 4 or 5.
• not CS 5 from HMC or CS 30 from Pomona
• Come see one of faculty if having any questions
• Assume comfortable with classes & objects, recursion, multi-dimensional arrays, etc. in Java

Heavy Workload

• students spend average of 8+ hours outside of class.
• ... but not “weeder”
• Must both learn practical (programming) skills and more theoretical analysis skills
• Learn about tools to become better programmer
• Be ready to answer “interview questions”

Grading Policy

| Weekly Programming Assignments | 35% |
| Exams: | Total: |
| Midterm: 15% each | 55% |
| Final Exam: 25% | |
| In-lab exercises and quizzes | 10% |
| Total: | 100% |

• We drop the two quizzes with the lowest grade
• Keep this option for real emergencies and unpredictable events

See online syllabus for other important information! Especially academic honesty!!

http://www.cs.pomona.edu/classes/cs062
Object-Oriented Design

• Objects are building blocks.
• Programs are collections of interacting objects.
• Objects cooperate to compute solutions or complete tasks.
• Objects communicate via sending messages.

Objects

• Objects can model objects from world:
  • Physical things
    • e.g., car, student, card, deck of cards
  • Concepts
    • e.g., meeting, date
  • Processes
    • e.g., sorting, simulations

More objects

• Objects have:
  • Properties, e.g., color, model, manufacturer
  • Capabilities, e.g., drive, stop, admit passenger
• Objects responsible for knowing how to perform actions.
  • Commands: change object’s properties, (e.g., set speed)
  • Queries: respond based on object’s properties (e.g., how fast?)
### Even more objects

- Properties typically implemented as “fields” or “instance variables”
  - Affect how objects react to messages
  - Can be:
    - Attributes, e.g., color
    - Components, e.g., door
    - Associations, e.g., driver
- Capabilities as “methods”
  - Invoked by sending messages

### Quick Java Review

#### Primitive vs Object Types

- Objects: String, anything created by a class with “new”
  - respond to messages
- Primitives: `int, double, float, boolean`
  - do not respond to messages
  - cannot be used to instantiate type variables
  - have corresponding object types:
    - `Integer, Double, Float, Boolean`

#### Classes

- Classes are templates for objects
  - Constructors generate new distinct objects
    - `new Car("Toyota")`
  - Specify all fields and methods - public and non-public
  - May be used as basis for more refined classes via inheritance
    - `class Car extends Vehicle`
All classes inherit “Object” class

• Object class has methods:
  • public boolean equals (Object other)
    • Default behavior returns true only if same object
  • public String toString()
    • Returns string representation of object – default is hexadecimal
    • Does not print the string
    • Typically needs to be overwritten to be useful
  • public int hashCode()
    • Unique identifier defined so that if a.equals(b) then a, b have same hashCode

Enum Types

• Example
  • enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}

• Operations:
  • int compareTo(Suit other)
  • String toString()
  • int ordinal() – starts with 0, not 1
  • static Suit valueOf(String name)
  • static Suit[] values() – returns array of all values

Java Keywords

• abstract class – can’t be instantiated
  • usually some methods missing

• Information hiding qualifiers:
  • public
  • private
  • protected

• static – copy associated with class, not objects

• final – only assigned to once
  • in its declaration or constructor

Interfaces & Inheritance

• Provide info on publicly available methods of objects
  • “what not how”

• Class implements interface if it supports all methods of interface

• Try to use interfaces as types for flexibility

• Interface can extend another by adding methods
  • If A extends B and x has type A, then also has type B

• One class can extend another
  • inherits fields and methods
  • can override existing methods, add new ones

• instanceof & casts
  • Ex: in Ratio class later
Card Deck Example

- CardInterface -- interface
- AbsCard
  - abstract class, implements CardInterface
- Card extends AbsCard
- OtherCard extends AbsCard
- Deck
  - Class holding array of Card objects

Extending vs Implementing

- Extending a class allows sharing behavior:
  - Card, OtherCard extend AbsCard
- Implementing an interface provides an implementation
  - Card, OtherCard implement CardInterface
- Either can be associated with variable of type CardInterface
- Makes it easier to replace implementations.