Lecture 1: Overview & Java

CS 62
Spring 2012
Kim Bruce & Kevin Coogan
TAs: Ben Fish, Sam Konowich, Eli Omernick, Matthew Stock Matthews, and Will Tachau
Course web page: http://www.cs.pomona.edu/classes/cs062

Why CS 62?

- How to implement algorithms and data structures in Java & C++.
- How to design large programs (in object-oriented way) 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)
- Schedule final exams so there are no conflicts.
- Design and implement a scientific calculator.
- Design and implement a simulator that lets you study traffic flow in a city or airport.
- Design parallel algorithms to run on multicore computers

Your Responsibilities

- Skim reading in advance of lecture.
- After lecture, review lecture notes, sample code, and text until well understood.
- Come to labs prepared.
- Don't remain confused. Ask faculty or TAs.
- Follow academic integrity guidelines.

Assignments

- Lab work:
  - Learn tools & prep work for weekly assignments
  - Lab attendance mandatory!
- Weekly assignment is separate
  - Programs generally due on Monday nights.
  - See late policy on syllabus.
- Daily homework
  - Not collected, but often on regular Friday quizzes
  - No quiz this Friday.

Texts

- Java Structures, 4th edition, by Duane Bailey
  - available on-line for free
- C++ for Java Programmers, by Mark Allen Weiss
  - highly recommended for last 1/3 of course
Prerequisite

- One of:
  - CS 51 at Pomona or CMC
  - AP CSA exam with score of 4 or 5
  - Fluent in Java and object-oriented programming & permission of instructors
- Come see one of faculty if any questions
- Assume comfortable with classes & objects, recursion, multi-dimensional arrays, etc. in Java

Object-Oriented Design

- Objects are building blocks.
- Program is collection of interacting objects.
- Objects cooperate to compute solution.
- Objects communicate via sending messages.

Objects

- Model physical and conceptual world, as well as processes.
- Objects have:
  - Properties, e.g. color, size, manufacturer, ...
  - Capabilities, e.g. drive, stop, admit passenger
- Objects responsible for knowing how to perform actions.
  - Commands: change state
  - Queries: response based on properties

More Objects

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

Quick Java Review
Classes & Interfaces

- Interfaces
  - Provide info on publicly available methods of objects
- 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

All Classes Specialize Object

- 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
    - Typically want to override to be more useful
  - `public int hashCode()`
    - Unique identifier defined s.t. if `a.equals(b)` then `a, b` have same `hashCode`
    - Cover in later chapter of text.

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()`

Card Deck Examples

- `CardInterface` — interface
- `AbsCard`
  - abstract class, implements `CardInterface`
  - `Card extends AbsCard`
  - `OtherCard extends AbsCard`
- `Deck`
  - `Class using cards`

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

- Class implements interface if supports all methods defined in interface
- 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`