CSE 8B discussion: week 6

HW4 review, inheritance, overriding (or is it overloading?)
Agenda

1. HW4 review
2. Inheritance (motivation, aspects of, when to use)
3. Object, and other core aspects of Java OO
HW4 review
Inheritance

- Reduces repetition in code (Don’t Repeat Yourself principle)
  - e.g., a single place we define types of Coins, 2048 tile values, etc.

- An opportunity to give organization to your code
  - UML diagrams convey structure of a program succinctly (no code, just high-level description)

- ...also restrictive; see the diamond problem of multiple inheritance
  - note: do BufferedInputStream example

- An identifying characteristic of an object-oriented language
  - other languages with inheritance: JavaScript, Python, Ruby, C++, C#, CSS (in a way)
  - some have inheritance features similar to Java, others do it differently
Java’s inheritance model

- The concepts:
  - **superclasses, subclasses**, and the `extends` keyword
  - **interfaces**, and the `implements` keyword
  - **abstract** methods and classes
  - **polymorphism**, a set of features that allow for more flexibility in your code
Interfaces: what they can contain

- "can contain only constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces."
- all method declarations are implicitly public (so you may omit the "public")
- more on interfaces
Interview question: abstract class vs. interface

- Compare and contrast (I’ve been asked this twice)
- Impress the heck out of the interviewer: demonstrate strong grasp of Java fundamentals if you **describe how to make design choices using both** (i.e., when to use which)
- What interfaces have you worked with? What abstract classes?
“Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.”

source: Oracle abstract class/methods tutorial
The more important thing: when to use them

- Taken from Oracle’s tutorial on abstract methods and classes:

- Consider using abstract classes if any of these statements apply to your situation:
  - You want to share code among several closely related classes.
  - You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
  - You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.

- Consider using interfaces if any of these statements apply to your situation:
  - You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.
  - You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
  - You want to take advantage of multiple inheritance of type.
Interfaces and subclasses together: pest example

- Stolen from here
Polymorphism

- A collection of features that give you flexibility in your code, letting you keep certain things separate
  - `SortedMap dictionary = new TreeMap();`
  - object can be passed to methods that take a Map, `SortedMap` (both of which are interfaces)
  - choosing the type to be `SortedMap` rather than `TreeMap` conveys that the identifying feature of the object is that it's a `SortedMap`; being a `TreeMap` is a coincidence
- Java figures out at runtime (while the program is running) what version of a method to call
- this aspect of polymorphism is called **dynamic dispatch**
- example
Agenda

1. HW4 review
2. Inheritance (motivation, aspects of, when to use)
3. *Next week*: Object, and other core aspects of Java OO