CS 211: Enumerations

Chris Kauffman

Week 9-1
Logistics

Exam 1
Back Wednesday (probably)

Today
- Top-level Kinds in Java
- Enumerations
- P4 discussion

Lab 8: Exercises
Enumerations

P4 Circuits
- Due in 3 weeks
- Big-ish
- Abstract classes
- Enumerations (today)
- Interfaces (next)
The Continuum of Java’s Top-Level Entities

- Regular classes are more concrete
- Abstract classes are more ethereal
- **Enumerations** are as concrete as possible
- Interfaces are as ethereal as possible
Java has 4 Top-Level Kinds

**class**
- Run of the mill concrete objects
- Child classes extend

**enum**
- Like a class (fields methods) except...
- All instances declared up front, automatically static final
- Good for modeling fixed collections
- Cannot extend

**abstract class**
- Can’t instantiate but good for single inheritance hierarchies,
- Child classes extend

**interface**
- Can’t instantiate
- Good for capabilities cutting across class hierarchies: savable, accessible, observable, comparable
- Child classes implement
enum: An Enumeration

Like saying class
  ➤ Can have fields
  ➤ Can have methods
  ➤ Can have constructors
  ➤ **BUT** all possible instances are declared up front
  ➤ **NO** public constructors allowed
    ➤ You’ll never get to `new` one

**enum** will be a fixed set
Typical Uses

Create a fixed set of objects for modeling

States of a baby: no properties

- BState used by Baby
- BabyWithState has an inner enum
- Latter indicates enum isn’t meant for public use
- Irritating need to include BState or State for all enum values
- Note weird .class files after compiling BabyWithState
Exercise: Cards in a Deck

- Create an `enum Card` for the `value` of a playing card
- Values: two, three, four, ... ten, jack, king, queen, ace
- Should take you 2 minutes
Enums are functional

- Can have fields, Can implement methods

**Cards: beats(c) method**

```java
Card x = Card.two;
Card y = Card.ten;
boolean wins = x.beats(y); // false
Card z = Card.king;
wins = z.beats(y); // true
```

**Exercise:** Discuss implementation

How can one easily implement the `beats(c)` method?
Answer: Include fields for face value

- Each card has an integer `faceValue` field
- Method `beats(c)` uses that value

```java
public enum Card {
    two(2), three(3), four(4), five(5), six(6), seven(7), eight(8),
    nine(9), ten(10), jack(11), queen(12), king(13), ace(14);

    public int faceValue;
    private Card(int v) {
        this.faceValue = v;
    }
    public boolean beats(Card c) {
        return this.faceValue > c.faceValue;
    }
}
```
Enumerations in P4: Signal

Values: HI, LO, X

public Signal invert()
    HI -> LO
    LO -> HI
    X  -> X

public static Signal fromString(char c)
    '1' -> HI
    '0' -> LO
    'X' -> X
    'x' -> X

public static List<Signal> fromString(String inps)
    List<Signal> sigs = Signal.fromString("1001x1X0");
    sigs  -> [HI, LO, LO, HI, X, HI, X, LO]

@Override public String toString()
    HI -> "1"
    LO -> "0"
    X  -> "X"