CSE 331

Enumerated types (`enum`)

slides created by Marty Stepp
based on materials by M. Ernst, S. Reges, D. Notkin, R. Mercer, Wikipedia
http://www.cs.washington.edu/331/

Anti-pattern: int constants

```
public class Card {
    public static final int CLUBS = 0;
    public static final int DIAMONDS = 1;
    public static final int HEARTS = 2;
    public static final int SPADES = 3;

    ...
    private int suit;
    ...
    public void setSuit(int suit) {
        this.suit = suit;
    }
}
```

- What's wrong with using `int` constants to represent card suits?
  - variation (also bad): using `Strings` for the same purpose.

Enumerated types

- **enum**: A type of objects with a fixed set of constant values.

```
public enum Name {
    VALUE, VALUE, ..., VALUE
}
```

  - Usually placed into its own .java file.
  - C has `enum`s that are really `int`s; Java's are objects.

```
public enum Suit {
    CLUBS, DIAMONDS, HEARTS, SPADES
}
```

- **Effective Java Tip #30:** Use `enum`s instead of `int` constants.

  "The advantages of `enum` types over `int` constants are compelling. Enums are far more readable, safer, and more powerful."

What is an enum?

- The preceding `enum` is roughly equal to the following short class:

```
public final class Suit extends Enum<Suit> {
    public static final Suit CLUBS    = new Suit();
    public static final Suit DIAMONDS = new Suit();
    public static final Suit HEARTS   = new Suit();
    public static final Suit SPADES   = new Suit();

    private Suit() {}   // no more can be made
}
```

What can you do with an enum?

- use it as the type of a variable, field, parameter, or return

```
public class Card {
    private Suit suit;
    ...
}
```

- compare them with `==` (why don't we need to use `equals`?)

```
if (suit == Suit.CLUBS) { ...
```

- compare them with `compareTo` (by order of declaration)

```
public int compareTo(Card other) {
    if (suit != other.suit) {
        return suit.compareTo(other.suit);
    } ...
}
```

The switch statement

```
switch (boolean test) {
    case value:
        code;
        break;
    case value:
        code;
        break;
    …
    default:  // if it isn't one of the above values
        code;
        break;
}
```

- an alternative to the `if/else` statement
  - must be used on integral types (e.g. `int`, `char`, `long`, **`enum`**)
  - instead of a `break`, a case can end with a `return`, or if neither is present, it will "fall through" into the code for the next case

Enum methods

method	description
`int compareTo(E)`	all `enum` types are Comparable by order of declaration
`boolean equals(o)`	not needed; can just use `==`
`String name()`	equivalent to `toString`
`int ordinal()`	returns an enum's 0-based number by order of declaration (first is 0, then 1, then 2, ...)

method	description
`static E valueOf(s)`	converts a string into an `enum` value
`static E[] values()`	an array of all values of your enumeration

EnumSet

- class `EnumSet` from `java.util` represents a set of `enum` values and has useful methods for manipulating `enum`s:

`static EnumSet<E> allOf(Type)`	a set of all values of the type
`static EnumSet<E> complementOf(set)`	a set of all enum values other than the ones in the given set
`static EnumSet<E> noneOf(Type)`	an empty set of the given type
`static EnumSet<E> of(...)`	a set holding the given values
`static EnumSet<E> range(from, to)`	set of all enum values declared between from and to

```
Set<Coin> coins = EnumSet.range(Coin.NICKEL, Coin.QUARTER);
for (coin c : coins) {
    System.out.println(c);              // see also: EnumMap
}
```

- **Effective Java Tip #32:** Use `EnumSet` instead of bit fields.
- **Effective Java Tip #33:** Use `EnumMap` instead of ordinal indexing.

More complex enums

- An enumerated type can have fields, methods, and constructors:

```
public enum Coin {
    PENNY(1), NICKEL(5), DIME(10), QUARTER(25);

    private int cents;

    private Coin(int cents) {
        this.cents = cents;
    }

    public int getCents()  { return cents; }

    public int perDollar() { return 100 / cents; }

    public String toString() {          // "NICKEL (5c)"
        return super.toString() + " (" + cents + "c)";
    }
}
```