COP 3330, Spring 2013

Basic Enums

Instructor : Arup Ghosh
03-13-13

School of Electrical Engineering and Computer Science
University of Central Florida

Today

- Enums

Enums

- Enums (short for Enumeration) are used to keep track of non-standard data that only takes on a limited number of states.

- Example:
  - Days of the week
  - Seasons
  - Suits of cards

Without enums

- In the absence of enums, this would be done using named constants (usually integers).

- Example:
  - `static final int SUNDAY = 1;`
  - `static final int MONDAY = 2;`
  - `static final int TUESDAY = 3;`
  - `static final int WEDNESDAY = 4;`
  - `static final int THURSDAY= 5;`
  - `static final int  FRIDAY = 6;`
  - `static final int SATURDAY = 7;`

Why is this bad?

- While this is workable, we're using a type that is simply too broad for the problem at hand.
- Say we have following method:
- `void myViewOnDays(int day)`
- Someone using your code can use the following call:
  - `myViewOnDays(20);`

- This is invalid – 20 doesn't refer to a season.

Exceptions?

- Why not just throw an exception if an invalid value is passed?
- Nothing wrong with that, but it feels like bad design.
- Your attitude should be: Why allow a design to ***potentially*** cause an error, when I can just make the error ***impossible***?
- Make it so that passing 20 causes a compile error.
  - Infinitely better than the annoyance of exception handling.
- Enums will give you this ability.

The enum solution

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

- This creates a little data type called Season which can only take four values.
- They're constants, but not ints or floats or any preexisting type.
- Each value can be accessed as: `Day.SUNDAY, Day.MONDAY,` etc.

Now…

- We change the method to:

```
void myViewOnDays(Day day) {
   …
}
```

- Now someone who uses your code cannot possibly pass an incorrect value.

- If he passes 20, it'll refuse to compile and demand to know why an `int` was passed when a `Day` was expected.

- Season only has 7 values to use. A perfect solution.

Additional benefits

- Every enum has a values() method that allows us to iterate over all valid values.
  - Can iterate over all valid seasons with perfectly safe code.

```
for(Day s: Day.values()) {
    …
}
```

- This will not need changing, even if seasons are added, removed or modified.
  - Constant ints cannot achieve such elegance.

Additional benefits

- We can also add extra information to enums, and even give them methods (and constructors).
- For example, we might initialize each Day with some other information.
- Adding a `toString()` method to an enum means we can get it to print nicely formatted information, instead of just the dumb looking shouty text `Day`.

Examples

- myViewOnDays

- Adding fields, methods and constructors to an enum.
  - Apple Price Example

Summary

- Enums are essentially type-safe constants.
- The values() method allows iteration over all valid values of an enum.
- We can add associated information (fields) to an enum, as well as some functionality (methods).
- See http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html for more details