How to use Enums

What is an Enum?

`Enums` are a way of creating a custom data type in Java that allows you to restrict to a specific set of values. Think of it like a String or a number, but you are listing out the only acceptable Strings or numbers allowed.

This is helpful for a number of reasons. One, it makes data that gets stored in databases much more accurate. You're using only business-approved values and spelling. Two, it makes validations a whole lot easier.

For example, an `Enum` for days of the week might look like:

```
enum DaysOfWeek {
    MON,
    TUE,
    WED,
    THR,
    FRI,
    SAT,
    SUN
}

DaysOfWeek days = DaysOfWeek.SUN; // days is now equal to the String "SUN"
```

You could just as easily make a String called `DaysOfWeek`, but then the client could type in something that wasn't allowed, like `"Bob"`. This way, there's free validation, as nothing else can be stored in `DaysOfWeek` other than the codes listed.

You can even add `constructors` to Enum values, to translate between codes and more readable values.

```
enum DaysOfWeek {
    MON("Monday"),
    TUE("Tuesday"),
    WED("Wednesday"),
    THR("Thursday"),
    FRI("Friday"),
    SAT("Saturday"),
    SUN("Sunday");

   public String day;

   DaysOfWeek(String day) {
      this.day = day;
  }

  public String getDay() {
     return this.day;
  }
```

```
}

DaysOfWeek day = DaysOfWeek.SUN; // day is equal to SUN

day.getDay() // Returns "Sunday"
day.toString() // Returns "SUN"
```

Now you can create a DaysOfWeek as `SUN` and have it equal `"Sunday"` and `"SUN"` depending how you reference it. Oftentimes systems have more coded values, and clients see more human readable values.

Enum built-in methods

You may have noticed we used `toString()` but I didn't have to create it myself. Enums have a number of handy built-in methods.

Another handy one is the static `valueOf(Enum, String)` which allows you to pass in an Enum class and a String. It will return the Enum constant if it finds a match, or throws an exception if it doesn't. You can use this for quick one-liner validation or to translate between a String value and a code value in one line.

See and read more below in the official Javadocs. Official Javadocs for Enum with more built-in methods and information

Validation with Enums

Beyond limiting what you can store in it, Enums also allow you to easily validate plain text Strings against your list of acceptable values.

For example, you could validate a String input for day of the week against your known list of acceptable days of the week:

```
for (DaysOfWeek day: DaysOfWeek .values()) {
            if(userInput.equalsIgnoreCase(String.valueOf(day)) ||
                    userInput.equalsIgnoreCase(day.getDay())) {
                return true;
            }
}
```

This works because an `Enum` can easily be converted to a `List` using the built-in `.values()` method.

Here's the same example using Streams and functional programming, a topic we don't yet cover but I think is cool and I hope we do someday 😀

```
return Arrays.stream(DaysOfWeek.values()).anyMatch(day ->
(userInput.equalsIgnoreCase(String.valueOf(day)) ||
                userInput.equalsIgnoreCase(day.getDay())));
```

You can also do this using the built-in `valueOf()` method mentioned earlier, which would return a match or throw an exception if no match was found. Using this in combination with a try/catch makes a pretty nice one-liner validation. (Note this only works to validate codes, like "MON" and not the String values of "Monday")

```
try {
    DaysOfWeek.valueOf(DaysOfWeek.class, userInput);
} catch (IllegalArgumentException e) {
    // no match found!
}
```

If you're dealing with an Enum already created and want to do something depending on it's value, you can utilize a switch statement.

```
DaysOfWeek day = DaysOfWeek.MON;

switch(day) {
    case MON:
        // Do something
        break;
    case TUE:
        // Do something else
        break;
    etc etc
}
```