Class Design Part 3

Enumerated Data Types (enums)

enum

- A way to provide a restricted set of values
- You can declare variables of this type.
- Examples
  - Sizes: Small, medium, large
  - Suits: Diamonds, hearts, spades, clubs
  - Semesters: Fall, summer, spring

```
enum Size {SMALL, MEDIUM, LARGE};

Size s = Size.LARGE;
// s can only hold the values SMALL, MEDIUM, LARGE, null
```

Can’t we just use constants?

```
public static final int SMALL = 0;
public static final int MEDIUM = 1;
public static final int LARGE = 2;

public int size = SMALL;
```

- No type safety
  - `public void setSize(int size) {` // someone could send in -9!
- Allows for illogical results
  - If you had `public static final int FALL = 2;` then `FALL == LARGE`. Huh?!
- Brittle
  - If a constant value is changed, the code must be recompiled, and so must any other code that uses that value.
- No easy way to translate to String output
- No way to iterate over all of the choices

Constants vs. enums

- Bottom line: Constants are good things! You should use them in your code. But enums are better when they make sense.
- Constants are good for single values like min, max, default values, etc.
- enums are good when something has a predefined, finite set of possible values.

Practice

- Add an enums to our classes to represent a *status* that describes all employees.
- Use that enum in the class- only active, full time employees receive benefits.

enums are actually classes!

- An enum is actually a class with exactly X number of instances declared. And it's not possible to construct any more instances.
- Can add constructors, methods, and fields.
  - Constructors are invoked when the enum constants are constructed.
  - Methods and fields are used when you want to associate data or behavior with a constant
- All enums are subclasses of `Enum`.
- You can use == to compare enum types.

Example (from Core I)

```
enum Size {
	SMALL("S"), MEDIUM("M"), LARGE("L"), EXTRA_LARGE("XL");

	private String abbreviation;

	private Size(String abbreviation) {
		this.abbreviation = abbreviation;
	}
	public String getAbbreviation() {
		return abbreviation;
	}
}
```

The Enum Class- Inherited Methods

- `toString`
  - returns the name of the constant
  - invoke on an enum's value
  - example: Size.SMALL.toString() returns "SMALL"
- `valueOf`
  - send in the class and the name and get back the value
  - static method invoked on Enum class
  - example: Size s = Enum.valueOf(Size.class, "SMALL");

The Enum Class- Inherited Methods

- `values`
  - returns an array of all possible values
  - static method invoked on the actual enum
  - example: Size[] values = Size.values();
- `ordinal`
  - returns the position of the constant in the declaration (starting from 0)
  - invoke on an enum’s value
  - example: Size.LARGE.ordinal() returns 2

enums- Better than Constants!

- Type safety
  - `public void setSize(Size s) {`
    `// can only accept a variable of type Size`
- No illogical results
  - Size.SMALL is not equal to Semester.FALL
- Flexible
  - You can add onto your enum values- in any order- and other code that uses the enum will continue to work. No recompiling needed!
- Easy String output through enum methods.
- Easy iteration over all possible choices through values() method.

Practice

- Revise the Status enum to add more information using a constructor and method.

Practice

- Write an enum to represent ice cream flavors.
  - Some flavors are nut-free, others are not.
- Write a class to represent an ice cream order (described by flavor and number of scoops).
- Write a driver program to ask the user to create orders by entering the flavors (taking nuts into consideration!) and number of scoops.

Practice

- Write code related to a CustomerAgent (perhaps who works on the phone doing customer service) and a FeedbackReport that is made about an agent (perhaps by the customer after he/she is helped).
- Write an enum to describe the range of the score (negative, neutral, positive).
  - Include variables for the range of scores.
  - Add a method to return an enum value based on the score.
- Write a FeedbackReport class:
  - Described by the score, range, and text feedback
  - Each report has a unique ID generated when the report is created
  - Keep track of the number of all negative feedback reports created
- Write a CustomerAgent class:
  - Described by name and a list of FeedbackReports
  - Write a method to determine if an agent is eligible for a bonus- they are eligible if they have had at least 100 feedback reports and 75% or more of them are positive
  - Write a method to clear out an agent's feedback report (perhaps at the start of a new year)

COMPARATOR

The Comparable Interface

- Provides a way to order objects.
  - The *natural ordering*
- Used by Arrays.sort and Collections.sort.

```
public MyClass implements
        Comparable<MyClass> {…

public int compareTo(MyClass obj)
```

There's more than one way to sort!

- Sometimes I want to sort based on ID. Sometimes I want to sort based on Name.
- I can only have one compareTo method!

The Comparator Interface

- Allows you to specify an ordering of two objects.

```
public int compare(T o1, T o2)
```

- API: https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/util/Comparator.html

Writing a Comparator Class

- Comparators are often written as static inner classes.
- Common also to create constants of type Comparator<T> that are instances of the comparator.

```
public final static Comparator<Employee> ID_ORDER
    = new EmployeeIDComparator();

public final static Comparator<Employee> NAME_ORDER
    = new EmployeeNameComparator();
```

Writing a Comparator Class

```
private static class MyClassComparator implements Comparator<MyClass> {
        public int compare(MyClass m1, MyClass m2) {
                // compare m1 and m2 using their instance data;
                // common to invoke compareTo on instance data variable

                // return negative number if m1 < m2, positive if
                // m1 > m2, and 0 otherwise
        }
}

public static final MyClassOrder MY_SORT = new MyClassComparator();
```

Using a Comparator Object

- Collections.sort(myCollection, MY_COMPARATOR_OBJECT)
- Arrays.sort(myArray, new MyCompartor())

Practice

- Add a comparator to the Employee class.
- Create a list of employees and test out different orderings.

Comparable vs. Comparator

	Comparable	Comparator
What is represents	The *natural ordering* of an object Only one per class	Any ordering Can have as many as you want
Where it is written	Implemented by the object's class: `public MyClass implements Comparable<MyClass>`	Implemented by a new (often nested) class: `public MyComparatorClass implements Comparator<MyClass>`
The method	`public int compareTo (MyClass otherObj)`	`public int compare (MyClass object1, MyClass object2)`
How method is invoked	`myObject.compareTo( otherObject)`	`myComparator.compare( thisObject, thatObject)`

Used in sorting	By default:
	
	`Collections.sort(myCollection );`
	
	`Arrays.sort(myArray);`
		By specifying:
		
		`Collections.sort(myCollection, new MyComparatorClass());`
		
		`Arrays.sort(myArray, new MyComparatorClass());`

Practice

- Review the user examples.
- Implement the various comparators.

On Your Own Practice

- Use the Store Inventory classes from Modules 01 and 02.

- Add at least one enum with at least one field and method. Use this enum somewhere in the classes.
- Create two comparator classes for your Store Inventory. Practice sorting by different characteristics.