Java Enum

Lab Setup

1. Create a new Java project called "JavaEnums"
2. Add the Bicycle class to the project
3. Create and add a main class that we can use as a driver

Enum Types

Enum types are special data types which enable a variable to be a set of predefined values. An enum variable's value must be one of the predefined values. We can define enums in their own java file, or nested within a class.

1. We will create a BicycleListing class which represents a listing on a marketplace website. A listing has the Bicycle, the original price, and the condition of the bicycle. The condition will be implemented as a nested enum. The described class can be seen in Figure 1.

```java
public class BicycleListing {
    enum Condition{
        NEW (1.0),
        USED (0.5),
        DAMAGED (0.25);

        double price_mod;

        Condition(double mod){
            price_mod = mod;
        }

        double calcPrice(double price) {
            return price * price_mod;
        }
    }

    private Bicycle bicycle;
    private Condition condition;
    private double price;

    public BicycleListing(Bicycle bicycle, Condition condition, double price) {
        this.bicycle = bicycle;
        this.condition = condition;
        this.price = price;
    }

    void setCondition(Condition condition) {
        this.condition = condition;
    }

    Condition getCondition() {
        return condition;
    }

    double getOrigPrice() {
        return price;
    }

    void setOrigPrice(double price) {
        this.price = price;
    }

    double getAdjustedPrice() {
        return condition.calcPrice(price);
    }
}
```

Figure 1 BicycleListing class with Condition enum.

The Condition of a bicycle can be either `NEW`, `USED`, or `DAMAGED`. The price of the bicycle is adjusted based on the condition. This could be done a couple of ways.

The first is shown above. The Condition enum type has a field for a price modifier. Each Condition value has a price modifier associated with it, and that can be used to calculate the adjusted price.

The second would be to use if-statements to adjust the price based on the Condition. Which method do you think is better?

Graded Submission Task 1:
Write a driver to test the BicycleListing class.

Create a listing for each condition and verify the adjusted price of each listing is correct. To create a BicycleListing object, the constructor needs a Condition as a parameter. You can access the Condition enum the same way you can access static nested classes e.g. `BicycleListing.Condition.NEW`.

Print the condition and adjusted price of each BicycleListing. Enums implement toString() to print a string representation of its value.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

Drone Application

We will implement enums into our Drone application by introducing status flags for drones.

Graded Submission Task 2:

Add an enum to the Drone class which represents the drone's job status. The possible job statuses are `ACTIVE`, `IDLE`, and `RESERVED`.

- **ACTIVE**
  All drones are active as they move around the system. At the beginning of each time step, every drone that is not reserved is set to active.
- **IDLE**
  An idle drone will not move for the current time step.
- **RESERVED**
  A drone can become reserved only if it was previously idle. Drones can be reserved for a number of time steps during which the DroneController cannot change its job status and it cannot be commanded by the DroneController. Control of the drone is

handed back to the DroneController at the end of the last time step the Drone is reserved for.

Write a driver to model how these job statuses function. Let the DroneController choose some Drones to be IDLE from the ACTIVE drones. Then choose some of the IDLE drones to be RESERVED for other jobs. Print the status of each Drone at the end of each time step. For RESERVED drones, print the time step they should be returned to the DroneController.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

Lab wrap-up

Enums are special datatypes with a set of predefined values. Enums can be defined in their own .java file, or nested within a class. In our example, we nested the Condition enum within BicycleListing.

Enums can have fields like classes. In our example, the Condition enum had a field for a price modifier. We also defined a constructor in Condition. We use this constructor when defining the set of values the enum can take. Our Condition enum also defined a method which applied the price modifier to a value we gave it. We can omit any fields or constructors from the enum like we do in the Drone application example.

Enums have a toString() method which returns a String representation of the predefined value.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder “Lab9_firstName_lastName”.
- Submit the zipped file to the respective Canvas link.

Rubrics	
Task 1 (allotted points) - Compilation (2) - Listing created for each condition (4) - Adjusted price and Condition shown to be correct in Driver (4)	Task 2 (allotted points) - Compilation (2) - Enum for job status defined (2) - Statuses printed at end of time step (2) - Driver demonstrates logic for each job status (4)

Appendix

This section will describe some of the methods which belong to the Enum base class from which all enums inherit. Assume E is the enum type.

- `compareTo(E)`: Compares an instance of E to another instance.
- `ordinal()`: Returns the position of the enumeration constant in the enum declaration.
- `E.valueOf(String)`: Returns the enum constant with the specified name. The provided String needs to exactly match the enum constant.
- `E.values()`: Returns an array of the enumeration constants.