Polymorphism Exercises

In order to practice applying polymorphism, your task is to create interface definitions and class implementations for the exercises defined below. In any of the cases, you may add attributes (i.e. properties) and other supporting methods to the classes in order to fully implement the interface.

Each implementation class requires unit tests to verify its correctness.

Vehicle

IVechicle Interface

Method	Return Type
CalculateToll(int distance)	double

Car

Create a Car class that implements the IVechicle interface. Additionally it has the following properties and constructors.

Property	Type	Get	Set
HasTrailer	bool	X	

Constructor	Description
Car(bool hasTrailer)	Creates a new car indicating whether or not it is pulling a trailer.

The logic for CalculateToll(int distance) based on distance is:

> toll = distance * 0.020
>
> if pulling a trailer then toll = toll + 1.00

Truck

Create a Truck class that implements the IVechicle interface. Additionally it has the following properties and constructors.

Property	Type	Get	Set
NumberOfAxles	int	X	

Constructor	Description
Truck(int numberofAxles)	Creates a new truck indicating how many axles it has.

The logic for CalculateToll(int distance) based on distance is:

> | Axles | Per Mile |
> | --- | --- |
> | 4 | 0.040 |
> | 6 | 0.045 |
> | 8+ | 0.048 |
>
> toll = rate per mile * distance

Tank

Create a Tank class that implements the IVechicle interface.

All military vehicles travel free on the toll roads. The logic for CalculateToll(int distance) based on distance is:

```
toll = 0;
```

Void Main in Program.cs

Following the approach that the morning's examples has led, create a List<IVehicle> that represents all of the vehicles that travel through a particular tollbooth. Using a random number for distance (10 to 240) calculate the tolls for each vehicle so that you can:

- indicate each vehicle type, its distance traveled, and toll
- indicate the sum of all miles traveled and total tollbooth revenue

Sample Output using random distances

```
Vehicle          Distance Traveled      Toll $
-----------------------------------------------
Car              100                    $2.00
Car              75                     $2.50
Tank             240                    $0.00
Truck            150                    $6.75

Total Miles Traveled: 5.65
Total Toolbooth Revenue: $11.75
```

DIFFICULT - Postage

IDeliveryDriver Interface

Method	Return Type	Description
CalculateRate(int distance, double weight)	double	CalculateRate takes the distance being traveled (in miles) and the weight in ounces to calculate the rate.

Postal Service

Create PostalService classes that implements the IDeliveryDriver interface. The Postal services deals with pounds and ounces.

The rate calculation depending on which postal service class being used is below.

```
|--------|-----------|-----------|-----------|
|        | 1st Class | 2nd Class | 3rd Class |
| Ounces | Per Mile  | Per Mile  | Per Mile  |
|--------|-----------|-----------|-----------|
| 0 - 2  |   0.035   |   0.0035  |   0.0020  |
| 3 - 8  |   0.040   |   0.0040  |   0.0022  |
| 9 - 15 |   0.047   |   0.0047  |   0.0024  |
| Pounds |           |           |           |
| 1 - 3  |   0.195   |   0.0195  |   0.0150  |
| 4 - 8  |   0.450   |   0.0450  |   0.0160  |
| 9+     |   0.500   |   0.0500  |   0.0170  |
|--------|-----------|-----------|-----------|

rate = per mile rate * distance
```

HINT Consider how multiple classes per delivery mechanism (1st class, 2nd class, 3rd class) could help.

FexEd

FexEd is a really expensive IDeliveryDriver that charges a flat rate for most packages. They may apply extra charges if the distance or weight exceeds a specific threshold.

FexEd "is-a" DeliveryDriver and charges a flat rate for all packages, but may apply extra charges based upon weight and distance

```
rate = 20.00
if distance > 500 miles then rate = rate + 5.00
if weight > 48 ounces then rate = rate + 3.00
```

SPU

Create SPU classes that implement the IDeliveryDriver interface. They each have their own formula based on class, weight (in lbs), and distance., SPU "is-a" DeliveryDriver and follows a simple formula based upon class, weight (in lbs) and distance.

```
If four-day ground then rate = (weight * 0.0050) * distance
If two-day business then rate = (weight * 0.050) * distance
if next day then rate = (weight * 0.075) * distance
```

void Main in Program.cs

Following the approach that the morning's examples has led, create a List<IDeliveryDriver> that represents a distribution hub and can calculate the shipping values for all of the various delivery methods so that the customer can make a safe and informed decision.

Sample Output

```
Please enter the weight of the package? 15
(P)ounds or (O)unces? O
What distance will it be traveling to? 340


Delivery Method                $ cost
--------------------------------------
Postal Service (1st Class)     $15.98
Postal Service (2nd Class)     $1.65
Postal Service (3rd Class)     $0.84
FexEd                          $20.00
SPU (4-day ground)             $1.75
SPU (2-day business)           $17.50
SPU (next-day)                 $26.25
```