3. Control Structures

Exercise 3.1 — Use decimal to represent money values in the vending machine

1. Hey, since decimal is a better type to use to represent money, we should go back and modify purchase price to use `decimal` to represent purchase price instead of `int`.
2. However, do not remove the members of the PurchasePrice class that allow its users to create and manipulate PurchasePrice objects using `int`. The interface to a class does not mandate its implementation.
3. Make this change and rebuild your vending machine. Your vending machine shouldn't have to change. However, consider these two possibilities: either you can change vend to take advantage of the change in PurchasePrice, or leave vend as it is. Which makes the most sense to you?
4. How should you change the interface to your PurchasePrice class to best reflect this change in implementation?

Exercise 3.2 — Add a class to represent the cans of soda

1. Create a new enumerated type Flavor to represent possible flavors of soda (i.e., regular, orange, and lemon). If you put this definition inside of a separate file, note that there is no "Add Type" menu item, so select "Add Class" when you right-click on the project you are adding this type to. Alternatively, you can place this definition inside of the same source file as the class Can (the creation of which is the next step in this exercise).

```
public enum Flavor {REGULAR, ORANGE, LEMON}
```

2. Create a new class Can.

```
class Can
{
      public readonly Flavor _flavor = Flavor.REGULAR;

      public Can()
      {
      }


      public Can (Flavor AFlavor)
      {
            _flavor = AFlavor;
      }
}
```

3. You will need to modify your existing class CanRack to allow users to manipulate the can rack based on values of your new enumeration type (i.e., create new overloads of methods like AddACanOf() that take Flavor as a parameter). For example, create a method with the following signature:

```
public void AddACanOf(Flavor FlavorOfCanToAdd)
```

4. Should you just add the new functionality, or replace the old with the new? Note that for simplicity we will still be representing the cans in the rack with simple integer values and not actually storing can objects in the rack.
5. Modify your vending machine to use this new class. Use the new version of any CanRack methods that take Flavor as a parameter rather than a **`String`**.

----OPTIONAL-----

Exercise 3.3 — Create a class to represent coins.

1. Create a new console application and add the new Coin class to it. Eventually we will integrate this class into our vending machine, but for now we will develop Coin separately.
2. Implement class Coin given the information below
3. `ValueOf` gives the `decimal` value for the given `Coin.Denomination` parameter.
4. `CoinEnumeral` gives the value of the coin as a Denomination (e.g., Nickel).
5. We aren't ready to integrate the class Coin into our vending machine yet, so use the Main() method in the new console application can be a simple driver method that will exercise the class Coin.

```
class Coin
{
    public enum Denomination
        { SLUG = 0, NICKEL = 5, DIME = 10, QUARTER = 25, HALFDOLLAR = 50 }

    // parameterless constructor – coin will be a slug
    public Coin()

    // parametered constructor – coin will be of appropriate value
    // if value passed is outside normal set (e.g. 5 cents = Nickel)
    // coin is a slug
    public Coin(Denomination CoinEnumeral)

    // This constructor will take a string and return the appropriate enumeral
    public Coin(string CoinName)


    // parametered constructor – coin will be of appropriate value
    public Coin(decimal CoinValue)

    //  This property will get the monetary value of the coin.
    public decimal ValueOf

    //  This property will get the coin name as an enumeral
    public Denomination CoinEnumeral


    // use Enum.GetName() with a private Denomination instance variable
    // to produce a string
    public override string ToString()
}  // end Coin
```