# 2. Classes

## Exercise 2.1 — Create a class to represent the purchase price of a can of soda.

1. This class will be worked into the vending machine application built in the first exercise set.
2. Create the code for this class in the vending machine application project.
3. The details for this class appear below.

```
//  This class represents the purchase price of something.
//  In our software project, we will use it to represent the price of
//  one can of soda.
class PurchasePrice
{
      // This constructor sets the purchase price to zero
      public PurchasePrice()

      // This constructor allows a new purchase price to be set by the user
      public PurchasePrice(int initialPrice)

      //  This property gets and sets the value the purchase price.
      public int Price

} //end PurchasePrice
```

4. Modify your simple vending machine to use the PurchasePrice class.
    a. Create a purchase price object
    b. Use the price to display to the user how much to enter

## Exercise 2.2 — Create a class to represent the cans held in the soda vending machine

1. Add a can rack class to your vending machine project. A can rack will be composed of three can bins (one can bin holds all the cans of a particular flavor and all three can bins together comprise the can rack). Each bin holds a maximum of three cans of a given flavor.

2. For this class, instrument the methods by printing out an informational message to the debug window when they are called. To write to the debug window use the System.Diagnostics.Debug.WriteLine() method. Include a using directive for System.Diagnostics at the top of the code file.

```
using System.Diagnostics;

namespace blahblahblah
{
    class Program
    {
        static void Main(string[] args)
       {

       Debug.WriteLine("This goes to the debug window");

       Debug.WriteLine("To make this window visible use menu item”);

       Debug.WriteLine("Debug\\Windows\\Output");
```

3. The message printed should be a simple, complete statement of fact (e.g., “adding a can of orange to the rack”).

4. For this exercise there are three flavors of soda: Regular, Orange, and Lemon.

```
    //  This class will represent a can storage rack of the vending machine.
    //  A can of soda will simply be represented as a number
    // (e.g., orangeCans = 1 means there is one can of orange soda in the rack).
    class CanRack
    {

        // Constructor for a can rack. The rack starts out full
        public CanRack()

        //  This method adds a can of the specified flavor to the rack.
        public void AddACanOf(string FlavorOfCanToBeAdded)

        //  This method will remove a can of the specified flavor from the rack.
        public void RemoveACanOf(string FlavorOfCanToBeRemoved)

        //  This method will fill the can rack.
        public void FillTheCanRack()

        //  This public void will empty the rack of a given flavor.
        public void EmptyCanRackOf(string FlavorOfBinToBeEmptied)

        // OPTIONAL – returns true if the rack is full of a specified flavor
        // false otherwise
        public Boolean IsFull(string FlavorOfBinToCheck)

        // OPTIONAL – return true if the rack is empty of a specified flavor
        // false otherwise
        public Boolean IsEmpty(string FlavorOfBinToCheck)


    } //end Can_Rack
```

5. Use this class to create a can rack object. Use this object in your vending machine.

6. When the vending machine is ready to dispense a soda, call the appropriate method. For now, have the vending machine choose which flavor can to eject.
7. You should be able to observe the flow of control through the software due to the instrumentation that you did with the methods of can rack.

**OPTIONAL -------**

8. If you wrote the IsFull() and IsEmpty() methods, add code to Main() to attempt to remove more cans of soda of a specified flavor from the machine than it has in it. Consider how AddACanOf() and RemoveACanOf() should behave based on using the information from these methods.