A2: Handling Events and Modeling Systems with Diagrams

You handed in your vending machine--complete with working logic (i.e. how it dispenses change) along with structures to simulate the hardware (e.g. coin chutes, pop chutes, etc.)--right in the nick of time. Unfortunately, just as you do so, your boss informs you that the spec she gave you was wrong, and that they had the entirety of the hardware ready the whole time! What a horrible boss! She suggests you reimplement your logic to work with the hardware she has provided, or you will be out of a job. Oh, and there's a new intern coming in, and so you need to help the intern understand how the vending machine system works. Your job is to:

1. Write the code that drives the logic in the vending machine, and
2. Draw UML diagrams that represent two versions of the vending machine system.

Learning Objectives

By the end of this assignment, you will:

- Implement a component that relies on event callbacks to interact with other components.
- Overcome (again) the fear of opening up a code base that is likely too difficult to understand in its entirety.
- Develop UML diagrams to describe the structural and behavioural function of two small-to-medium sized systems that highlight the differences between the two.

Deliverable / Submission

You will deliver two files into the Dropbox in D2L containing the following:

1. A .zip file containing the source code that drives the logic of the vending machine system using the provided "hardware." As with last time, remember:
    - From the project directory, remove the bin\ and obj\ folders
    - Zip up the solution directory, ensuring that your source files and test scripts are included (if you are including test scripts, include these at test-scripts\)
2. A .pdf file containing a total of six UML diagrams.
    - Two class diagrams -- one for each of your solution to A2, and for *my* solution to A1.
    - Two sequence diagrams -- one for each of your solution to A2, and for *my* solution to A1.
    - Two state diagrams -- one for each of your solution to A2, and for *my* solution to A1.
    - *Optionally:* You may include a page that introduces the diagrams and/or rationale for why the diagrams are abstracted in the way that they are.

As with A1, We will extract your code and run it against a small battery of tests (some should pass, some should fail). These tests are designed against the specification. Your diagrams will be evaluated for clarity and understandability.

Steps

1. Clone the repo from the GitHub site
2. Read through the documentation below
3. Modify only VendingMachineFactory.cs and/or add new classes. You can re-use parts of your A1 or my solution to A1, though bear in mind some of the interface signatures have changed.
4. Draw each of the class, sequence and state diagrams for both systems (your solution to A2, and my solution to A1). *Hint:* It may be useful to do some of this sketching earlier to help your understanding of the code.
5. Submit via D2L.

Description

The Hardware. The vending machine hardware is represented by a set of classes found in the Frontend2.Hardware directory (and namespace). For instance, the vending machine, delivery chute, coin slot, selection buttons, etc. all have their own classes. Coins for change are stored in coin racks; similarly, pop cans for sale are stored in pop can racks. More details about the vending machine are below.

The Logic. Your main programming task is to program the *logic* of the vending machine, so that when someone puts money into the system and/or makes a pop selection, they actually get the pop, and the change that they deserve.

Unloading the Vending Machine. When unloading the vending machine, the system expects this to be done using an VendingMachineStoredContents object rather than that monstrous 3-tuple from A1. Your boss does not understand what the last coder was doing when they came up with that idea for A1. The definition of IVendingMachineFactory has been accordingly modified to account for this.

Scripting Language. The scripting language that drives the system has also been slightly modified. Specifically, the CREATE call now takes three additional arguments representing the capacity of coin racks, the capacity of pop racks, and the capacity of the coin receptacle/storage bin/delivery chute. These must all be positive integers. The parser has been modified to account for this.

Parser. Whereas CHECK_DELIVERY and CHECK_TEAROWN would previously simply write PASS/FAIL to the Console, they now throw exceptions if those commands fail. If they succeed, they do so silently (i.e. there is no output).

UML Diagrams. You need to construct these to illustrate how the vending machine works, specifically, the diagrams should illustrate the relationships and interactions between the script, the Parser, the VendingMachineFactory, and other classes/objects that are present. The state machines should focus on how (and when) monetary change is produced, and the events should be the scripting commands. As you sketch these diagrams, pay particular attention to the *differences* between A1 and A2.

Specification for the Vending Machine

As before, the number of coin racks and pop can racks and valid coin kinds are defined when the vending machine is constructed. Coins entered by a customer are temporarily held in a container called the coin receptacle; after these coins are used to pay for a pop (i.e. after someone has pressed a button), they are moved to one or more coin racks. If a coin rack is too full to accept more coins, the excess coins are stored in the storage bin. You can control *when* coins are moved, but not *where* they are to go (this is already defined in the VendingMachine class). For example, coins in the coin receptacle can be either returned to the customer via the delivery chute or stored in the coin racks and/or the storage bin, while a coin released from a coin rack will automatically be sent to the delivery chute.

The hardware is designed as an event-based system. For example, when a customer enters a coin into the coin slot, this causes the coin slot's CoinAccepted event to fire, which in turn, notifies any listener that has registered for that event. Recall that registering for this event essentially means that you need to add a handler, or a function that matches the delegate signature for the event. Similarly, if the coin that was inserted was not an okay one, the CoinRejected event would fire. If you wanted to register a function for this event from your object (say to print a message about it), you might write some code that looked something like this to register/subscribe to the event:

```
coinSlot.CoinRejected += new EventHandler<CoinEventArgs>(printCoinRejected);
```

In turn, you would probably have event handler code in your class that looked something like this that would run when the event was fired:

```
void printCoinRejected(object sender, CoinEventArgs e) {
    Console.WriteLine("Coin slot just rejected this coin: " + e.Coin);
}
```

To build the logic for the system, you need to figure out which events are interesting to listen to, and work from those events to drive the vending machine system's behaviour.

As with before, you are only to modify VendingMachineFactory.cs and (optionally) add additional classes to make this all happen. Do not modify any other classes (particularly the ones in Frontend2.Hardware) -- your boss is very protective of her code.

Many of the classes and functions are not those that you will need to deal with. So, ignore these. Some functions that you would like are unfortunately not there. Unfortunately, you will just need to cope with the hardware as it is.

Specification for the Test Script Commands

The test script syntax is virtually identical to that of A1. The only difference is that the CREATE command requires an additional three arguments.

Hints

As with A1, this is a lot to look at all at once (even more!). Make sure that you can settle down and concentrate for an extended period of time as you read through the documentation.

1. Take notes based on the requirements and specification above.

2. Look at the test scripts in test-scripts/ (they are new).
3. Write your own test scripts.
4. Take a look at how ICoinAcceptor is used (which classes oject inherit it, and in what ways are those classes used).
5. It may be useful to refresh yourself on how to subscribe to events in C#.
6. It would likely make the most sense to create one or more Logic classes and register these with the appropriate pieces of hardware. When you do your first pass at this (e.g. with DummyVMLogic), you might consider registering for events that you expect to be fired, and then dumping this out with a Console.WriteLine command -- just to understand how the system is behaving.
7. Not all the hardware parts matter to you. Definitely not all the events matter to you. These are not there to confuse you; rather, the point is to illustrate what a "functioning" system would look like.
8. With respect to the diagrams, take multiple passes on each -- your first try on it is likely not to be very good. My suggestion is to expect to do at least three passes on each, refining the sketch as you go (not just "beautifying" it, but actually thinking about what can/should be abstracted away). You are free to use UML drawing software, though my suggestion is to do it by hand (forces you to be more conservative with "ink").

Questions

As with last time, post your questions to the #A2 channel on Slack.

Grading Rubric

You work will be run against a set of test scripts (much like the ones you have in test/) that are designed from the requirements laid out above. The more of these your code passes, the higher your grade.
Again, you may be awarded some discretionary points for submitting interesting test scripts of your own (e.g. that consider boundary conditions).
Your code will not be assessed on its internal structure; however, we can provide verbal feedback on this if you ask.
Your diagrams will be assessed on the understandability, and their match with the system being modeled.

No Late Assignments

Your assignment is due at noon on the specified due date. No late assignments will be accepted.
