TPRG 2131 Project 1: Vending Machine

Release 1.0: October 8, 2019

A vending machine is a state machine controller that accepts coins then waits for a selection. If the sum of the value of the coins is sufficient for the selection, the machine activates motors or solenoids to deliver the product. Finally, the machine determines the change that must be returned to the user, if any.

Develop a program that simulates a vending machine controller as a state machine using object-oriented techniques: `StateMachine` class and `State` derived classes. The simulation runs in PowerShell or CMD (command line mode). The inputs are received from the Windows (PC) keyboard and the actions are simulated by messages to the screen.

Deliverables

This assignment is to be submitted in three parts:

1. The statechart in the simplified UML statechart scheme introduced in week 4 as a Visio or PowerPoint document, or you may also use an online drawing tool like lucidchart.com (sign up for a free trial account). Hand drawn is acceptable if drawn with straight lines and neat lettering, then scanned as a PDF (no cell phone photos).
2. The Python source code file `vending_machine.py` in a sub-folder `vending_machine`, committed to your personal Fossil repository on server `tprg.set.org`. Grading will be based on correct operation, generally well-organized layout and intelligent use of the SCM.
3. The Pytest testing script file `test_vending_machine.py` in the same sub-folder that tests the operation of the vending machine.

There is an optional set of requirements for bonus marks. Please see the **Optional challenges** section below.

Requirements

Program requirements

- The file name shall be: `vending_machine.py` in sub-folder `vending_machine`.
- Coins are entered from the keyboard as single keystrokes coded as:
  - 1 = 5 cents (nickel)
  - 2 = 10 cents (dime)
  - 3 = 25 cents (quarter)
  - 4 = 1 dollar (loonie)
  - 5 = 2 dollars (twonie)

- The selection is entered from the keyboard as a single keystroke coded as letters A—E (5 possible selections). Assign names to each choice – name them whatever names you want (e.g. chocolate bars, 74LS logic chips, whatever…)
- The "Cancel – Money back" button is the single keystroke 'Q' (for quit). When the user hits 'Q', the coins are returned, and the operation is ready to restart.
- The state machine runs until the CTRL-C is typed, and the program exits cleanly with a message like "shutting down…".

Grading scheme (15 points)

4	Functionality	Does it fulfill its functional requirements?
2	Style	Conformance to rules 1.4 to 1.9 of the course style guide.
2	Technique to be used	Does the program use the specified technique? The state machine is implemented with the class `StateMachine` and `state` classes derived from `State`.
1	Use of Fossil	Committed to your Fossil repository, with reasonable commit messages. The final commit message should indicate that the files are ready to mark.
2	heading & comments	Conform to style guide 1.1 to 1.3
4	Statechart	Statechart on Visio, PowerPoint or other drawing tool, or neatly hand drawn (straight edge for lines, etc.) and scanned (no cell phone photos).

If the program crashes on start-up, the maximum program mark is 4.4/11 (statechart is separate).

Program file structure

1. File heading
2. Import statement(s)
3. Utility function definitions (as needed)
4. Class definitions for StateMachine, State and derived state classes
5. Main program

Testing the state machine

The file `test_vending_machine.py` includes the PyTest script to test the state machine. You may have any number of test cases in the function `test_VendingMachine` to run a number of scenarios.

Use the example files `watercooler.py` and `test_watercooler.py` to see how the testing script can feed events to the state machine then run assertions against the current state name, or properties of the state machine (e.g. total of coins, selection, change due, etc.).

Each event must be processed after it is received. For example if vending is the name of the state machine object, the two lines:

```
vending. event='5'
vending. update ()
```

would cause the vending machine to receive the '5' event, a "townie" ($2 coin) entered, and to process the event. The assertion following might check `machine.state.name` or for total amount entered.

Hints

Statechart first: Draw the statechart before you code. Understand the events and transitions, and the update, enter and exit actions..

Money in cents: Do all the money calculations with cents (for example a $2 coin is 200 cents) to keep the arithmetic simple with integers only.

Optional challenge (+2 points each)

1: Event objects

In Python, as the saying goes, “everything’s an object”. Instead of using single character strings like ‘5’ and ‘A’ for events, create a class `Event` that encapsulates the type of event and its value. `CoinEvent`, `SelectionEvent` and `CancelEvent` events are subclasses derived from the main `Event` class. For example, a `CoinEvent` event has property type= '`coin`' and value 200. Modify the `get_event()` function to return an `Event` object, either `CoinEvent`, `SelectionEvent` or `CancelEvent`, instead of just a single character string. Instead of returning the empty string "" when there is no event, return the None object. Modify the `test _vendingmachine.py pytest` script accordingly.

2: Item objects

In Python, as the saying goes, “everything’s an object”. Instead of using individual attributes (variables) in class `VendingMachine` for the selections (name, price), define a class `item` that holds the name and price of the item.

3: Item objects

The machine keeps track of the quantity of each item (perhaps in an `Item` object, see above) and issues the “Sold out” message when the quantity goes to zero. When that happens, the machine does not return coins in case the user makes another selection. The coins are only returned when the user cancels the transaction (‘Q’ cancel event). For this assignment, there is no need to re-stock the vending machine.