Orientación a Objetos 2 – Práctica 4

Ejercicio 1: Vending machine

Considere una máquina vendedora de botellas de agua sin gas. La misma conoce el precio de la botella y tiene un cierto stock (cantidad de botellas). La máquina acepta monedas de 0.50, 1 y 2 pesos y también puede dar vuelto. Su funcionamiento es muy sencillo:

- El cliente indica la cantidad de botellas que desea comprar. Si la cantidad excede al stock la máquina lo informa y termina la venta.
- Caso contrario, la máquina informa el monto total y espera a que el cliente ingrese el dinero.
- El cliente ingresa las monedas hasta igualar o superar el monto total.
- Si el monto supera al monto total, la máquina calcula el vuelto que debe entregar. Si no dispone del cambio necesario, devuelve las monedas al cliente y finaliza la venta.
- Caso contrario, entrega los productos y el vuelto.

Tenga en cuenta que el cliente puede cancelar la compra en todo momento y la máquina devuelve el dinero ingresado. Además, considere las siguientes situaciones:

- La máquina no acepta dinero antes de que se indique la cantidad de botellas que se desean comprar.
- La máquina no acepta indicar la cantidad de botellas a comprar cuando está aceptando dinero.

Considere que la máquina posee un display (para nuestro caso el Transcript) y los anuncios lo hace a través del mismo. Preste atención a los siguientes métodos y respecto de la funcionalidad de entregar las botellas y el vuelto, solo actualice el stock y dinero que posee la máquina.

- `#comprar: cantidadDeBotellas`, define la cantidad de botellas que el cliente desea comprar. Devuelve true si es posible hacer la compra, false caso contrario.
- `#costoTotal`, devuelve el monto total que el cliente debe pagar.
- `#ingresar: unaMoneda`, el cliente ingresa en la máquina una moneda de la denominación indicada. Devuelve true cuando alcanzó o superó el costo de la compra, false en caso contrario.
- `#dineroRestante`, devuelve el dinero que aún resta por pagar.
- `#poseeVuelto`, retorna si la máquina posee el dinero suficiente para dar el vuelto.
- `#confirmarCompra`, una vez que el dinero fue ingresado, el cliente confirma que desea hacer la compra. La máquina actualiza el stock y el dinero.
- `#cancelarCompra`

Tareas:

1. Analice los cambios de estados y en que estado tienen sentido las operaciones indicadas.
2. Diseñe la solución para proveer la funcionalidad descripta.
3. Diseñe casos de prueba.

4. Implemente la clase VendingMachine y los casos de prueba

Ejercicio 2: Calculadora

Sea la clase `Calculadora` con los siguientes mensajes:

- `#resultadoParaMostrar` que retorna el valor acumulado en la operación actual de la calculadora en forma de String (los números entienden el mensaje printString).
- `#borrar` vuelve a cero el valor acumulado.
- `#valor:` asigna el valor acumulado.
- `#mas` provoca que la calculadora espere un nuevo valor. Si a continuación se le envía el mensaje `#valor:` la calculadora sumará el valor recibido como parámetro al valor actual acumulado y guardará el resultado en esta última.

Si la calculadora está esperando un valor y se le envía cualquier otro mensaje, entonces pasará a un estado de error. Sólo saldrá de ahí si se le envía el mensaje `#borrar`. Los mensajes `#menos`, `#dividido` y `#por` actúan de manera similar al mensaje `#mas`.
Cuando la calculadora está en estado de error, el mensaje `#resultadoParaMostrar` retorna el string Error. La calculadora también entra en este estado si se intenta dividir por cero.

Tareas:

1. Analice los cambios de estado y documéntelos mediante un diagrama de estados UML.
2. Diseñe la aplicación calculadora para proveer la funcionalidad antes descripta. Utilice un diagrama de clases completo indicando qué patrón aplica para que el diseño quede extensible, claro y modificable. Indique lo roles de las clases mediante estereotipos.
3. Realice un diagrama de secuencia para el mensaje `#valor:` con la calculadora en estado de espera de un valor para efectuar una suma.
4. Implemente en Smalltalk completamente

Ejercicio 3: Patrón State

Lea el capítulo del patrón State en el libro *Design Patterns* y responda:

1. ¿Cuándo es necesario usar el patrón State?
2. ¿Qué representa el contexto dentro del patrón State?
3. ¿Puede el estado concreto acceder al contexto?
4. ¿Quién define las transiciones entre estados?