Virag Kiss
ITO4133-TP3-23 Introduction to Python TP3
Assessment 2: Vending machine program
Report
12 June 2023

Section 1: High-level Design

1. Classes

The design I chose to implement this program is class-based, with 4 main classes: Item, Coin, Transaction, and VendingMachine.

Item and Coin are objects used exclusively to store data of the vending machine, as the states of these objects are not required to change during program execution. Attributes of these classes are initialized once at the beginning of the program. The numbers of these objects stay constant during program execution as well, and these objects are not reused as they are stored in the respective attributes of the VendingMachine class as stock and inventory of coins. However, the attributes of Item and Coin objects store information about the VendingMachine state, e.g. how many individual items are left of a given product or coin in the machine.

The **Transaction** object is re-initialized at the start of every new transaction and records data of that transaction, e.g. the date, items bought and total price. At the end of a successful transaction, it is appended to the respective attribute of the VendingMachine to permanently record that transaction and persist its data, after which the Transaction object gets reused. Cancelled transactions are not permanently recorded, as the state of the VendingMachine does not change when a transaction is cancelled.

The **VendingMachine** object contains all the real-time data needed for program execution, and its state changes dynamically according to what the user or admin does. Its attributes store Item, Coin and Transaction objects as lists. This class contains the driver functions that consume the functionality of other classes. These driver functions are used in the main() function, which is the main driver of the program.

I attempted to modularize the program as much as possible so that individual components can be updated without having to rewrite large chunks of the program.

2. Data storage and persistence / File handling

Data is stored in the above objects and is persisted throughout the program, except when the vending machine gets restocked or reset. In the latter cases, transaction history is cleared, and the customer (admin) is responsible for exporting transaction data into a file

called **output.csv**, if they want to keep a record of the transactions that happened before reset / restock. The program used a Pandas dataframe to record transactions to be exported. In my design,

Reset: means setting the vending machine to default state, where attributes and object states are hard-coded in the program.

Restock: means setting the state of the vending machine dynamically, where input is collected from the customer. These inputs are a) is the machine operational, b) the number of items to add of each type, c) the number of coins to add of each value, d) serves of sugar.

3. **Usability**

The system has 2 multi-choice menus, the Main Menu and the Edit Menu. The Main Menu navigates all vending machine functions, whereas the Edit Menu is used to manipulate transactions.

```
What would you like to do?
1 Make a new transaction
2 Reset machine (Admin Only)
3 Restock machine (Admin Only)
4 View transaction history (Admin Only)
5 Quit program
```

```
What would you like to do?
1 Confirm order
2 Edit cart
3 Cancel order
```

Left: Main Menu

Right: Edit Menu

These menus provide high functionality and facilitate decisions with many nodes and possible outcomes. They are also simple and straightforward, increasing usability.

Section 2: User Cases

Name: U-001, Start new transaction
Actors: Users
Goal: Users select restart to reset previous transactions.

In my implementation, starting a new transaction can be selected from the Main Menu. The transaction is “reset” in that the new cart is empty, and previous transactions would have been recorded and stored.

```
Welcome


What would you like to do?
1 Make a new transaction
2 Reset machine (Admin Only)
3 Restock machine (Admin Only)
4 View transaction history (Admin Only)
5 Quit program
1
```

Name: U-002, Display list of products
Actors: Users
Goal: Users select a product from the list of products on the display screen.

Users can select a product by entering its item id, then they can specify how many items of the same product they want to add. This facilitates usability and user-friendliness.

```
item id: 1, name: coffee, price: 4.5, stock left: 5
item id: 2, name: tea, price: 3.5, stock left: 5
item id: 3, name: coke, price: 5.0, stock left: 5
item id: 4, name: juice, price: 5.5, stock left: 5

To add a product to your cart, enter its item id:
1
```

```
item id: 1, name: coffee, price: 4.5, stock left: 5
item id: 2, name: tea, price: 3.5, stock left: 5
item id: 3, name: coke, price: 5.0, stock left: 5
item id: 4, name: juice, price: 5.5, stock left: 5

How many coffees would you like to add?
2
```

Users can also select to add more items in one transaction.

```
Items in cart: 2 coffee, Total price (AUD): 9.0

item id: 1, name: coffee, price: 4.5, stock left: 3
item id: 2, name: tea, price: 3.5, stock left: 5
item id: 3, name: coke, price: 5.0, stock left: 5
item id: 4, name: juice, price: 5.5, stock left: 5

Would you like to add more items? y / n:
n
```

Name: U-003, Display cost of selected item(s)
Actors: Users
Goal: Users read and confirm by selecting the continue option.

After the Main Menu, there is what I meant to function like an Edit Menu, where transactions can be confirmed, further edited or cancelled. Cost of selected items are displayed throughout the transaction including in the Edit Menu as seen below,

```
Items in cart: 2 coffee, Total price (AUD): 9.0


What would you like to do?
1 Confirm order
2 Edit cart
3 Cancel order
1
```

and at payment, as seen below.

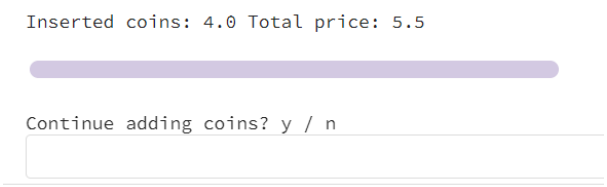
```
Items in cart: 2 coffee, Total price (AUD): 9.0

Your total in AUD is 9.0

To insert coins, type values separated by commas. 200 for $2,
Example: 200, 100, 50, 20, 10
```

Name: U-004, Insert coins to get the product
Actors: Users
Goal: A user can insert coins into the machine based on the indicated price.

In my design, coins are input as a list of integers. There is an appropriate message to instruct the user and the program does not accept any other type of input. However, if the user types in the wrong format, they can still keep adding coins in the right format until total is reached or they choose to cancel the transaction.

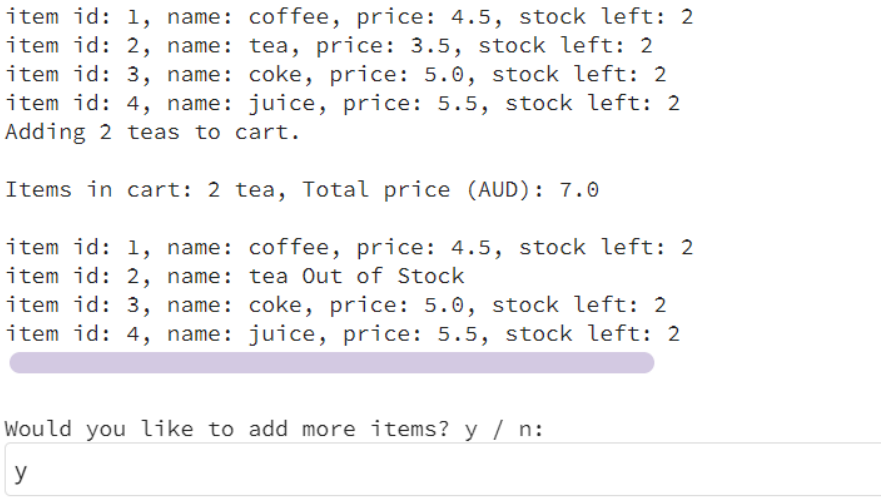
```
Items in cart: 2 tea, Total price (AUD): 7.0

Your total in AUD is 7.0

To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20 cent
Example: 200, 100, 50, 20, 10


Insert coins:
200, 200, 200, 200
```

To keep adding coins (screenshot from another transaction):

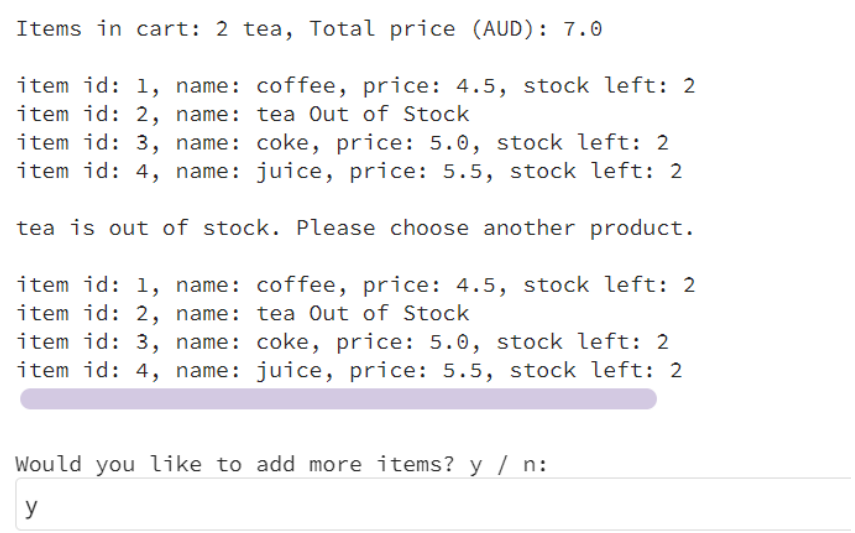


Name: S-002, Display availability of items
Actors: Users
Goal: Display available items, otherwise provide appropriate information to the user if a product is out of stock.

The below screenshot shows the machine dynamically adjusting the stock of selected products, after a number of products have been added to cart.



An error message will be displayed if the user tries to add another tea:



Similarly, when trying to add more products than what is available, the system prints an error message. Below is an example of the user trying to add more cokes than available:

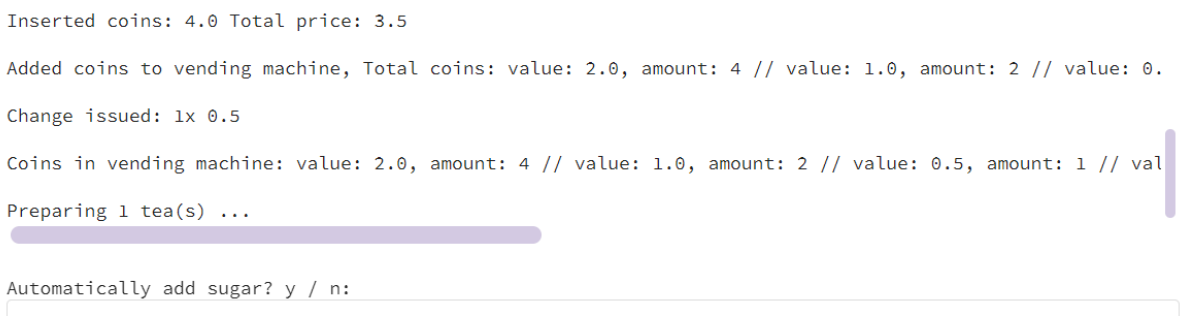
```
item id: 1, name: coffee, price: 4.5, stock left: 2
item id: 2, name: tea Out of Stock
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2

How many cokes would you like to add?
3

Not enough cokes available.

item id: 1, name: coffee, price: 4.5, stock left: 2
item id: 2, name: tea Out of Stock
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2

Would you like to add more items? y / n:
n
```

Name: S-007, Dispense selected item(s) only if correct amount of coins have been inserted
Actors: Users
Goal: Dispense only the items that have been paid for, otherwise cancel the transaction.

When the coins match the price in value exactly, the coins are added to the machine inventory and the item is dispensed.

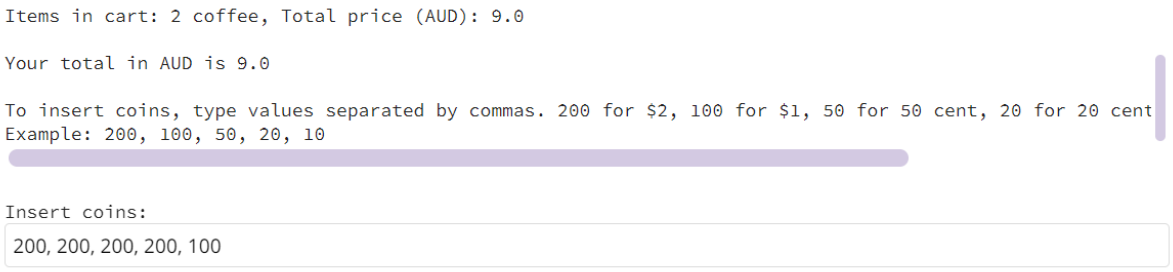
```
Your total in AUD is 3.5

To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20 cent
Example: 200, 100, 50, 20, 10

Inserted coins: 3.5 Total price: 3.5

Added coins to vending machine, Total coins: value: 2.0, amount: 5 // value: 1.0, amount: 3 // value: 0.

Preparing 1 tea(s) ...
```

The machine also issues any change and adjusts the number of coins in the machine accordingly.



Name: S-010, Accept coins of different amounts
Actors: Users and customers
Goal: Accept coins of different amounts: 10 cents, 20 cents, 50 cents, 1 dollar, 2 dollars.

The machine accepts a list of coins of the above values. It prints an appropriate message to help the user. For more, see section on “U-004, Insert coins to get the product”.



Name: S-011, Compare item cost with the total value of entered coins
Actors:
Goal: Compare total price with entered coins and proceed with the purchase if coins are enough.

As seen in “U-004, Insert coins to get the product”, the user can continue adding coins. This extends to the case when not enough coins have been inserted. If pressing “n” for No, the user can go back to the Edit Menu to edit the cart or cancel the transaction. For this, see “U-007, Change order as required”.

```
Items in cart: 1 juice, Total price (AUD): 5.5

Your total in AUD is 5.5

To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20 cent
Example: 200, 100, 50, 20, 10

Inserted coins: 4.0 Total price: 5.5


Continue adding coins? y / n
```

Name: S-012, Check coin validity
Actors:
Goal: Give a warning to the user if the inserted coin is not acceptable, e.g. if the system doesn't accept 5 cent coins.

```
Items in cart: 2 coffee, Total price (AUD): 9.0

Your total in AUD is 9.0

To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20 cent
Example: 200, 100, 50, 20, 10
5 cent coins are not accepted. Returning coin(s)
Inserted coins: 8.0 Total price: 9.0


Continue adding coins? y / n
y
```

Similarly, invalid coin inputs are not accepted, eg 70 would not denote valid coins.

```
Items in cart: 2 tea, Total price (AUD): 7.0

Your total in AUD is 7.0

To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20 cent
Example: 200, 100, 50, 20, 10


Insert coins:
200, 200, 200, 70
```

After inserting an invalid coin, the user can choose to insert more valid coins. Input in the incorrect format classifies as an invalid coin when converting the string input to a list of integers denoting coin values.

```
Your total in AUD is 7.0

To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20 cent
Example: 200, 100, 50, 20, 10

Invalid input: coin 70

Inserted coins: 6.0 Total price: 7.0


Continue adding coins? y / n
y
```

Name: D-005, Persist data of all coins and transactions
Actors:
Goal: Store the coins and record all transactions.

Machine state is displayed at the Main Menu, that is before starting a new transaction, resetting / restocking the machine, viewing the transaction history or quitting.

```
Welcome


Vending machine state:
Stock: item id: 1, name: coffee, price: 4.5, stock left: 1 // item id: 2, name: tea, price: 3.5, stock l
Transactions: date: 12/06/2023 13:54:01, items: {1: ['coffee', 450, 1], 4: ['juice', 550, 1]}, total: 10
Coins: value: 2.0, amount: 9 // value: 1.0, amount: 3 // value: 0.5, amount: 3 // value: 0.2, amount: 2
Serves of sugar: 2
Operational: True
```

The machine also keeps track of stock as they are added to / removed from cart:

```
item id: 1, name: coffee, price: 4.5, stock left: 8
item id: 2, name: tea, price: 3.5, stock left: 10
item id: 3, name: coke, price: 5.0, stock left: 8
item id: 4, name: juice, price: 5.5, stock left: 9
Adding 3 juices to cart.

Items in cart: 3 juice, Total price (AUD): 16.5

item id: 1, name: coffee, price: 4.5, stock left: 8
item id: 2, name: tea, price: 3.5, stock left: 10
item id: 3, name: coke, price: 5.0, stock left: 8
item id: 4, name: juice, price: 5.5, stock left: 6

Items in cart: 3 juice, Total price (AUD): 16.5
```

The machine also keeps track of coins when they are inserted, or change is given out:

```
Inserted coins: 18.0 Total price: 16.5

Added coins to vending machine, Total coins: value: 2.0, amount: 31 // value: 1.0, amount: 11 // value:

Change issued: 1x 1.0  1x 0.5

Coins in vending machine: value: 2.0, amount: 31 // value: 1.0, amount: 10 // value: 0.5, amount: 8 // v
```

Name: U-005, Cancel transaction as required
Actors: Users
Goal: A user can select cancel anytime during the transaction.

The Edit Menu, shown below facilitates cancelling orders and the user will always have to go through this to cancel an order. I designed the decision flow this way in order to navigates as many decision combinations as possible. Thus, there is a way to come back to edit or cancel from anywhere during the transaction.

```
Items in cart: 1 coffee, Total price (AUD): 4.5

What would you like to do?
1 Confirm order
2 Edit cart
3 Cancel order
3
```

The system will ask for confirmation to cancel an order:

```
Items in cart: 1 coffee, Total price (AUD): 4.5

Are you sure you want to cancel this transaction? y / n
y
```

When the order is cancelled, an appropriate message is displayed.

```
Items in cart: 1 coffee, Total price (AUD): 4.5

Transaction cancelled. Goodbye!
```

Cancelling can happen during payment as well:

```
Items in cart: 1 coffee, Total price (AUD): 4.5

Your total in AUD is 4.5

To insert coins, type values separated by commas. 200 for $
Example: 200, 100, 50, 20, 10

Inserted coins: 2.0 Total price: 4.5

Not enough coins? Press Enter to return to edit mode.
```

When returning to the Edit Menu from the payment section, the system gives back previously inserted coins:

```
Items in cart: 1 coffee, Total price (AUD): 4.5

Your total in AUD is 4.5

To insert coins, type values separated by commas. 200
Example: 200, 100, 50, 20, 10

Inserted coins: 2.0 Total price: 4.5

Returning inserted coins:  [200]


Items in cart: 1 coffee, Total price (AUD): 4.5

What would you like to do?
1 Confirm order
2 Edit cart
3 Cancel order
3
```

Name: U-007, Change order as required
Actors: Users
Goal: A user can change order e.g. if s/he doesn't have enough coins to buy the item they choose.

Orders can be changed through the Edit Menu, as seen in previous sections. In my implementation, this allows for dynamic decision-making for the user.

The Edit Menu, shown again:

```
Items in cart: 1 coffee, Total price (AUD): 4.5

What would you like to do?
1 Confirm order
2 Edit cart
3 Cancel order
2
```

Adding more items:

```
Items in cart: 1 coffee, Total price (AUD): 4.5

What would you like to do?
1 Add more items
2 Delete items
1
```

The user can continue editing:

```
Items in cart: 1 coffee, 1 tea, Total price (AUD): 8.0

item id: 1, name: coffee Out of Stock
item id: 2, name: tea, price: 3.5, stock left: 1
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice Out of Stock

Do you want to continue editing? y / n
n
```

If the user doesn't have enough coins, instead of continuing adding coins, they can edit their cart or cancel the transaction. See also previous section "U-005, Cancel transaction as required".

```
Items in cart: 1 coffee, 1 tea, Total price (AUD): 8.0

Your total in AUD is 8.0

To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20 cent
Example: 200, 100, 50, 20, 10

Inserted coins: 6.0 Total price: 8.0

Continue adding coins? y / n
n
```

Returning to edit mode gives back change.

```
Inserted coins: 6.0 Total price: 8.0

Not enough coins? Press Enter to return to edit mode.

```

,

```
Inserted coins: 6.0 Total price: 8.0

Returning inserted coins:  [200, 200, 200]


Items in cart: 1 coffee, 1 tea, Total price (AUD): 8.0


What would you like to do?
1 Confirm order
2 Edit cart
3 Cancel order
2
```

,

```
Items in cart: 1 coffee, 1 tea, Total price (AUD): 8.0


What would you like to do?
1 Add more items
2 Delete items
2
```

Now the user can delete one of the coffees if they can't pay for 2.

When adding / deleting items, stock is automatically updated. Previous stock:

```
Adding 1 teas to cart.

Items in cart: 1 coffee, 1 tea, Total price (AUD): 8.0

item id: 1, name: coffee Out of Stock
item id: 2, name: tea, price: 3.5, stock left: 1
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice Out of Stock
```

New stock displayed with cart modified:

```
Items in cart: 1 coffee, 1 tea, Total price (AUD): 8.0

Items in cart: 1 coffee, Total price (AUD): 4.5

item id: 1, name: coffee Out of Stock
item id: 2, name: tea, price: 3.5, stock left: 2
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice Out of Stock

Would you like to remove more items? y / n:
n
```

Name: S-001, Display status of the status of the machine
Actors: Users and customers
Goal: Display machine status information, whether out of order etc.

When restocking the machine or at first time installation, the admin can select to set the machine as non-operational.

```
You set the mahcine as non-operational. Display message will be adjusted.

This vending machine is under maintenance. Please check back later.

Press Enter to reset machine to default state.
```

The program is designed to continue flowing, therefore the machine can be reset again and the program doesn't have to quit. However, transactions will be cleared after maintenance.

```
Thank you, your machine is set to default mode!

Welcome


Vending machine state:
Stock: item id: 1, name: coffee, price: 4.5, stock left: 10 // item id: 2, name: tea, price: 3.5, stock
Transactions:
Coins: value: 2.0, amount: 10 // value: 1.0, amount: 10 // value: 0.5, amount: 10 // value: 0.2, amount:
Serves of sugar: 20
Operational: True


What would you like to do?
1 Make a new transaction
2 Reset machine (Admin Only)
3 Restock machine (Admin Only)
4 View transaction history (Admin Only)
5 Quit program
```

Name: S-003, Print warning if item ingredients e.g. sugar are unavailable
Actors: Customers
Goal: Alert message is printed automatically by the machine if any of the ingredients are unavailable.

The user can add a maximum of 5 sugars in one hot drink to keep stock under control.

```
Preparing 1 tea(s) ...

Enter amount of sugars to be added (max 5):
3
```

In this case, only 2 sugars were in the machine to begin with so 3 sugars could not be added and an error message was printed.

```
Preparing 1 tea(s) ...
Sorry, not enough sugars in the machine. Dispensing without sugar, please wait 1 minute.

Transaction successful. Goodbye!
```

Name: S-004, Process refund before dispensing item
Actors: Users
Goal: Coin may be refunded when the user cancels the request or the coin they insert is not sufficient.

If a transaction is cancelled, the machine returns the inserted coins. See decision flow:

```
Items in cart: 2 coke, Total price (AUD): 10.0

Your total in AUD is 10.0

To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20
Example: 200, 100, 50, 20, 10

Insert coins:
200, 200, 200
```

,

```
Inserted coins: 6.0 Total price: 10.0

Continue adding coins? y / n
n
```

,

```
Inserted coins: 6.0 Total price: 10.0

Not enough coins? Press Enter to return to edit mode.

```

,

```
Inserted coins: 6.0 Total price: 10.0
```

```
What would you like to do?
1 Confirm order
2 Edit cart
3 Cancel order
3
```

,

The machine returns the coins after the user asked to cancel the transaction:

```
Inserted coins: 6.0 Total price: 10.0

Returning inserted coins:  [200, 200, 200]


Items in cart: 2 coke, Total price (AUD): 10.0
```

```
Are you sure you want to cancel this transaction? y / n
y
```

,

Final message:

```
Items in cart: 2 coke, Total price (AUD): 10.0

Transaction cancelled. Goodbye!
```

A separate case is when the machine runs out of change. In this case, the user can select whether they want to proceed without receiving the change or cancel the transaction:

```
The vending machine ran out of change. Missing change for 150
```

```
What do you want to do? 1 Finish transaction without receiving more change
2 Cancel transaction
```

Name: S-009, Select products
Actors: Users
Goal: Select a range of products: coffee, tea, coke, and juice.

Products can be selected from the list. The user is also aware of prices and available items. Products can be selected after another. More than one product can be selected at once if there are enough items available of the same product.

```
item id: 2, name: tea, price: 3.5, stock left: 2
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2
Adding 1 coffees to cart.

Items in cart: 1 coffee, Total price (AUD): 4.5

item id: 1, name: coffee, price: 4.5, stock left: 1
item id: 2, name: tea, price: 3.5, stock left: 2
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2
Adding 1 juices to cart.

Items in cart: 1 coffee, 1 juice, Total price (AUD): 10.0
```

Name: D-006, Provide statistical data based on the transactions.
Actors: Customers
Goal: Gather and store data of all transactions and their details.

Statistical data can be viewed in 2 ways, during program execution by viewing the transaction history, or after quitting the program by viewing the outpus.csv file. Data is displayed as a Pandas dataframe and exported as a csv.

Viewing transaction history:

```
Welcome


Vending machine state:
Stock: item id: 1, name: coffee, price: 4.5, stock left: 8 // item id: 2, name: tea, price: 3.5, stock l
Transactions: date: 12/06/2023 14:39:13, items: {1: ['coffee', 450, 1], 4: ['juice', 550, 1]}, total: 16
Coins: value: 2.0, amount: 31 // value: 1.0, amount: 10 // value: 0.5, amount: 8 // value: 0.2, amount:
Serves of sugar: 18
Operational: True


What would you like to do?
1 Make a new transaction
2 Reset machine (Admin Only)
3 Restock machine (Admin Only)
4 View transaction history (Admin Only)
5 Quit program
4
```

```
Displaying transaction history...
```

	date	items	totals
0	12/06/2023 14:39:13	[[1, coffee, 450, 1], [4, juice, 550, 1]]	1000
1	12/06/2023 14:39:50	[[1, coffee, 450, 1]]	450
2	12/06/2023 14:40:46	[[3, coke, 500, 2]]	1000
3	12/06/2023 14:41:08	[[4, juice, 550, 3]]	1650

Selecting 5 to quit and exporting to csv file:

```
Exporting transaction history to csv...
                 date                                        items  totals
0  12/06/2023 14:39:13  [[1, coffee, 450, 1], [4, juice, 550, 1]]    1000
1  12/06/2023 14:39:50                      [[1, coffee, 450, 1]]     450
2  12/06/2023 14:40:46                        [[3, coke, 500, 2]]    1000
3  12/06/2023 14:41:08                       [[4, juice, 550, 3]]    1650
```

The csv file:

vending_machine.ip... | output.csv

```
1 12/06/2023 14:39:13,"[[1, 'coffee', 450, 1], [4, 'juice', 550, 1]]",1000
2 12/06/2023 14:39:50,"[[1, 'coffee', 450, 1]]",450
3 12/06/2023 14:40:46,"[[3, 'coke', 500, 2]]",1000
4 12/06/2023 14:41:08,"[[4, 'juice', 550, 3]]",1650
```

Name: U-006, Continue transaction as required
Actors: Users
Goal: There is a continue to buy option to allow for buying multiple items in a single transaction instead of starting a new transaction for every new item.

The user can continue adding as many items as they like, both when adding products for the first time and in Edit mode:

```
item id: 1, name: coffee, price: 4.5, stock left: 2
item id: 2, name: tea, price: 3.5, stock left: 2
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2
Adding 1 coffees to cart.

Items in cart: 1 coffee, Total price (AUD): 4.5

item id: 1, name: coffee, price: 4.5, stock left: 1
item id: 2, name: tea, price: 3.5, stock left: 2
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2

Would you like to add more items? y / n:
y
```

,

```
Items in cart: 1 coffee, Total price (AUD): 4.5

item id: 1, name: coffee, price: 4.5, stock left: 1
item id: 2, name: tea, price: 3.5, stock left: 2
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2

To add a product to your cart, enter its item id:
4
```

```
Adding 1 coffees to cart.

Items in cart: 1 coffee, Total price (AUD): 4.5

item id: 1, name: coffee, price: 4.5, stock left: 1
item id: 2, name: tea, price: 3.5, stock left: 2
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2
Adding 1 juices to cart.

Items in cart: 1 coffee, 1 juice, Total price (AUD): 10.0

item id: 1, name: coffee, price: 4.5, stock left: 1
item id: 2, name: tea, price: 3.5, stock left: 2

Would you like to add more items? y / n:
n
```

Name: S-005, Display welcome and goodbye messages
Actors: Users and customers
Goal: Display appropriate message at the beginning and end of the transaction respectively.

After dispensing an order, print Goodbye message and start the new round with a Welcome.

```
Dispensing 1 juice(s).

Transaction successful. Goodbye!

Welcome


Vending machine state:

What would you like to do?
1 Make a new transaction
2 Reset machine (Admin Only)
3 Restock machine (Admin Only)
4 View transaction history (Admin Only)
5 Quit program
```

Name: S-006, Display waiting time when preparing hot items
Actors: Users
Goal: Hot items can be dispensed in a realistic manner, where it takes time to make them.

Waiting time is displayed as coffees and teas are made. Each coffee and tea is made separately.

```
Preparing 2 coffee(s) ...
Adding 2 sugars and dispensing. Please wait 1 minute.
```

```
Automatically add sugar? y / n:
y
```

Name: S-008, Resetting operation for the vending machine supplier
Actors: Customers
Goal: Provide a restocking option for the vendor, e.g. at first-time installation, or whenever they might want to update the stock in the machine with specific amounts of items.

A custom reset looks like this:

```
Setting up vending machine once...

Is the machine operational? y / n
```

adding items:

```
[ ] Setting up vending machine once...

How many coffees would you like to add to the machine?
```

adding coins:

```
Setting up vending machine once...

How many 2.0 coins would you like to add to the machine?
```

adding sugars:

```
Setting up vending machine once...

How many serves of sugar would you like to add to the machine?
20
```

finishing message:

```
Setting up vending machine once...
Thank you, your machine is set up and ready to use!
```

When first starting the program, the admin is required to custom set the machine to demonstrate first-time installation. Then the custom restock function can be selected from the Main Menu at any time during operation as well.

Another option is an automatic reset, by selecting reset machine from the main menu, as shown below. Automatic reset is used after maintenance as well.

```
Welcome


Vending machine state:
Stock: item id: 1, name: coffee, price: 4.5, stock le
Transactions: date: 12/06/2023 13:54:01, items: {1: [
Coins: value: 2.0, amount: 9 // value: 1.0, amount: 3
Serves of sugar: 2
Operational: True


What would you like to do?
1 Make a new transaction
2 Reset machine (Admin Only)
3 Restock machine (Admin Only)
4 View transaction history (Admin Only)
5 Quit program
2
```

Finishing message and vending machine default state:

```
Thank you, your machine is set to default mode!

Welcome


Vending machine state:
Stock: item id: 1, name: coffee, price: 4.5, stock left: 10 // item id: 2, name: tea, price: 3.5, stock
Transactions:
Coins: value: 2.0, amount: 10 // value: 1.0, amount: 10 // value: 0.5, amount: 10 // value: 0.2, amount:
Serves of sugar: 20
Operational: True
```

Name: D-003, Mix sugar manually or automatically by the machine
Actors: Users
Goal: Allow for the user to add sugar to hot drinks such as coffee or tea.

When the user selects to add sugars automatically, the machine will ask for the number of sugars to be entered and compares it with the serves available in stock. If not enough sugars are available, an error message is printed and the item will be dispensed without sugar. See also section “S-003, Print warning if item ingredients e.g. sugar are unavailable”.

```
Preparing 2 coffee(s) ...

Enter amount of sugars to be added (max 5):
2
```

Each coffee has to be prepared individually:

```
Preparing 2 coffee(s) ...
Adding 2 sugars and dispensing. Please wait 1 minute.

Enter amount of sugars to be added (max 5):
6
```

If more than the maximum number of sugars are entered, the system gives an error but dispenses the item with max amount of sugars. (The max amount of sugars is hard-coded.

```
To insert coins, type values separated by commas. 200 for $2, 100 for $1, 50 for 50 cent, 20 for 20 cent
Example: 200, 100, 50, 20, 10
Inserted coins: 9.0 Total price: 9.0

Added coins to vending machine, Total coins: value: 2.0, amount: 9 // value: 1.0, amount: 6 // value: 0.

Preparing 2 coffee(s) ...

Automatically add sugar? y / n:
y

Preparing 2 coffee(s) ...
Adding 2 sugars and dispensing. Please wait 1 minute.
Max 5 sugars can be added per drink. Adding 5 sugars and dispensing, please wait 1 minute.
Transaction successful. Goodbye!
```

Section 3: Errors

This is a section in addition to the requirements that very briefly goes over some of the error handling techniques.

Cannot enter invalid menu entries:

```
Welcome


Vending machine state:
Stock: item id: 1, name: coffee, price: 4.5, stock left:
Transactions:
Coins: value: 2.0, amount: 2 // value: 1.0, amount: 2 //
Serves of sugar: 2
Operational: True

Invalid input. Please enter a number, no spaces.
```

Cannot enter invalid product numbers:

```
item id: 1, name: coffee, price: 4.5, stock left: 2
item id: 2, name: tea, price: 3.5, stock left: 2
item id: 3, name: coke, price: 5.0, stock left: 2
item id: 4, name: juice, price: 5.5, stock left: 2
Invalid input. No products have been added.
```

Cannot add 0 products:

```
Cannot add 0 products.
Items in cart: 1 coffee, Total price (AUD): 4.5
```

Cannot confirm an empty order:

```
Items in cart: Total price (AUD): 0.0

Cannot confirm an empty order. Please add some products to your cart.
```

When inserting coins, only valid coins and format will be processed:

```
Insert coins:
200 200 200

Invalid input. Please enter whole numbers separated by spaces.

Inserted coins: 0.0 Total price: 4.5
```

When setting up the machine, invalid entries are not allowed:

```
Setting up vending machine once...
Invalid input. Assigning default values.
```

When adding any other ingredients, or entries that are numbers, or y / n entries, only valid entries are allowed, otherwise the machine will proceed without carrying out the request:

```
Automatically add sugar? y / n:

Preparing 1 coffee(s) ...
Dispensing coffee without adding sugar. Please wait 1 minute.
```