Project

In this lesson, the complete Vending Machine project is included.

WE'LL COVER THE FOLLOWING

The Aim

The aim is to design a console-based application of a vending machine with the following specifications:

- Provides the options to the customers to:
  - Feed Money
  - Select a Product
  - Get Change
  - Quit
- Reads the customer inputs and acts accordingly.
- Accepts $ bills of 1, 2, 5, 10, 20, 50 and 100 denomination.
- Contains three shelves, one each for snacks, drinks, and chocolates. Each shelf can contain 4 items.

- Validates the inputs and displays the appropriate messages at each stage of the transaction.
- Returns the selected product and remaining change if any.
- Keeps track of the sold products and displays empty shelf slots accordingly.
- Says goodbye to the customers when they quit.

The Model

The first thing we need to figure out when designing this machine is the required classes and the relationship between these classes. For this, think of a real-world vending machine and its components.

A basic vending machine will have the following components:

- A keypad to get the customer input.
- A display screen to display messages.
- A shelf to store products.
- A calculator to handle the transactions.

> The shelf of the vending machine contains products, but these products are not a component of the vending machine itself.

Required Classes

According to the above-discussed components of a vending machine, we need to implement these classes followed by their respective functionalities:

- `Display`: This class is responsible for displaying all the messages to the customers, i.e., it replicates the functionality of a display screen.
- `KeyPad`: This class is responsible for reading and validating the input given by the customers, i.e., it replicates the functionality of a keypad.
- `Transactor`: This class is responsible for the whole transaction process of the customers by validating the $ bills, keeping track of the total money collected and returning the change to a customer if any, i.e, it replicates the functionality of the cash handling module of a vending machine.

- `Item`: This is an abstract base class which several types of food item classes can be derived from.
- `VendingMachine`: This is the most important class of our application where all the interaction between the objects of aforementioned classes takes place.

Identifying the Relationships

Inheritance

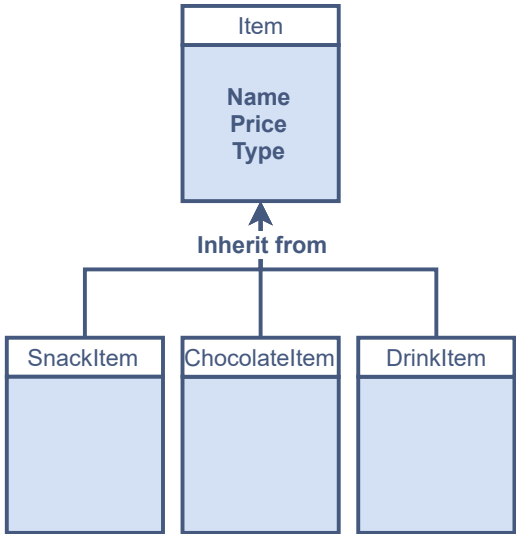
According to our goal, we need three types of items to be placed in the machine. Instead of implementing the chocolate, snack, and drink classes separately, we can save a lot of time and effort by implementing a base class `Item` that has all the commonalities of all types of food items and then derive the specific item classes, i.e., `SnackItem`, `DrinkItem`, and `ChocolateItem` classes, from it.

Composition

The components listed above work together to build a fully functional vending machine. The idea here is the same. The `VendingMachine` class will be the owner class containing the objects of the `KeyPad`, `Display`, and `Transactor` classes. When the vending machine ceases to exist, so do the components.

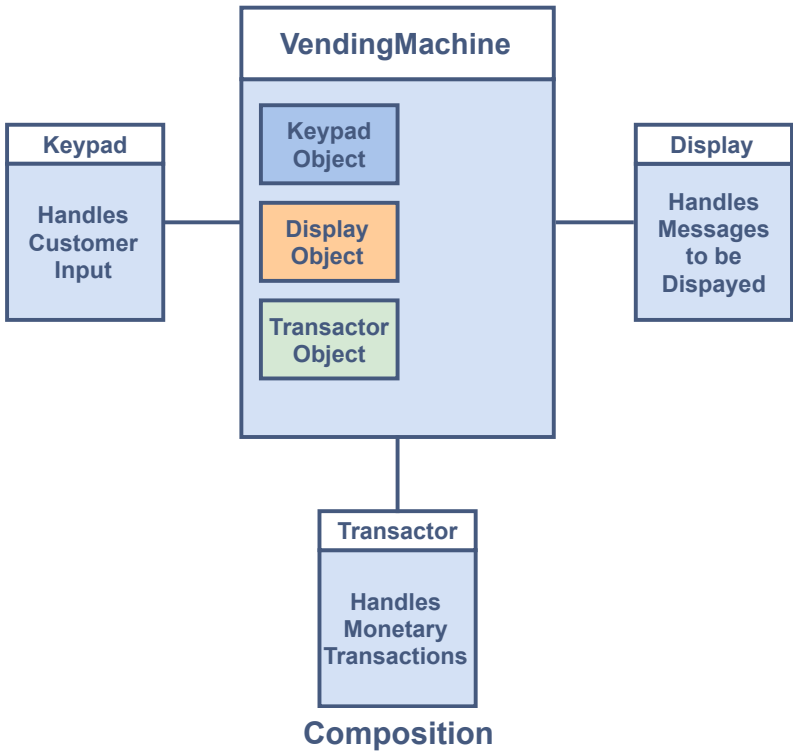
Aggregation

The items placed in the vending machine are not the components of the vending machine. They can exist independently even if the vending machine ceases to exist. Hence, the `VendingMachine` will just contain the reference to the objects of these items which is the very idea of aggregation.



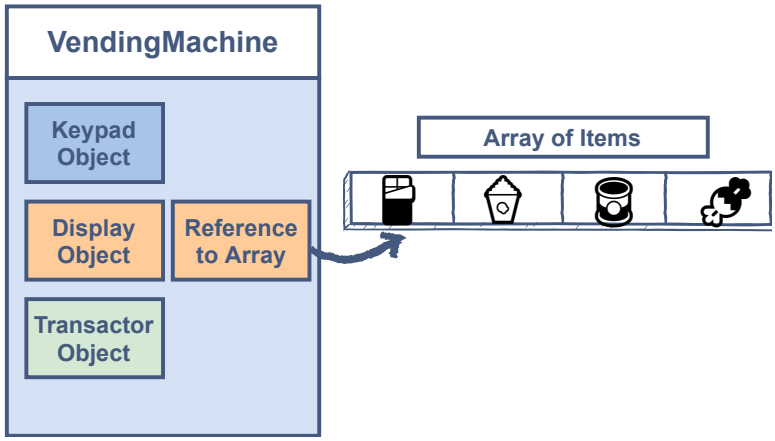
Inheritance

1 of 3



Composition

2 of 3



Aggregation

Application

The complete console-based vending machine application is given below:

```
using System;

namespace VendingMachineProject
{
    class Program
    {
        static void Main(string[] args)
        {
            // 3 x 4 array of Items
            Item[,] products = new Item[3,4];

            products[0, 0] = new SnackItem("Munchy", 7);
            products[0, 1] = new SnackItem("Chetoo", 5);
            products[0, 2] = new SnackItem("Dorito", 3);
            products[0, 3] = new SnackItem("Fritos", 5);

            products[1, 0] = new ChocolateItem("KitKat", 5);
            products[1, 1] = new ChocolateItem("Bounty", 6);
            products[1, 2] = new ChocolateItem("Snicks", 8);
            products[1, 3] = new ChocolateItem("Chocly", 7);

            products[2, 0] = new DrinkItem("CoCola", 2);
            products[2, 1] = new DrinkItem("Fantay", 3);
```

```
            products[2, 1] = new DrinkItem("Fantay", 3);
            products[2, 2] = new DrinkItem("Sprite", 4);
            products[2, 3] = new DrinkItem("Stingy", 4);


            // Passing the products array to the vending machine
            // Aggregation relationship
            VendingMachine myvendy = new VendingMachine(products);
            // Turning on the vending machine
            myvendy.TurnOn();
        }
    }
}
```

You can play around with the code and make any changes to it to understand things deeper.

The next lesson contains the concluding remarks for this course.