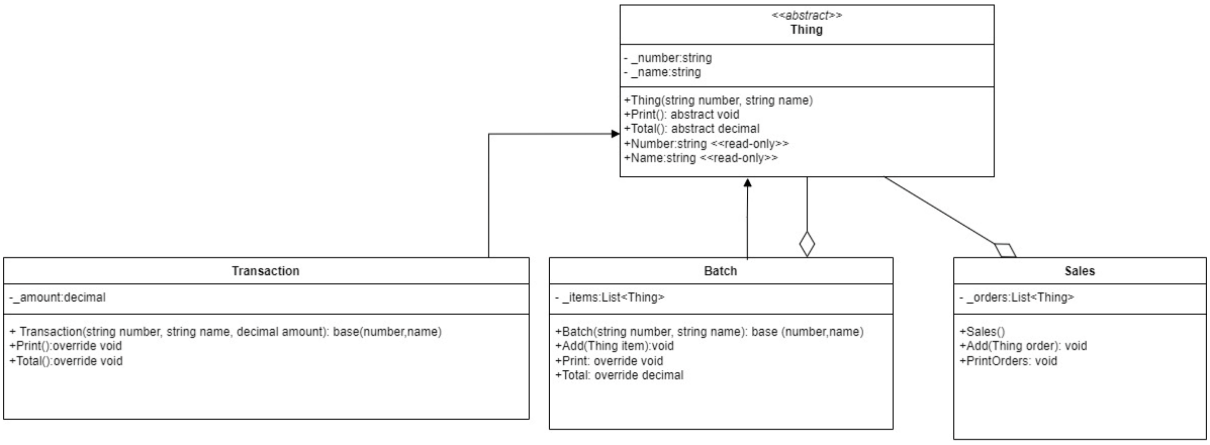
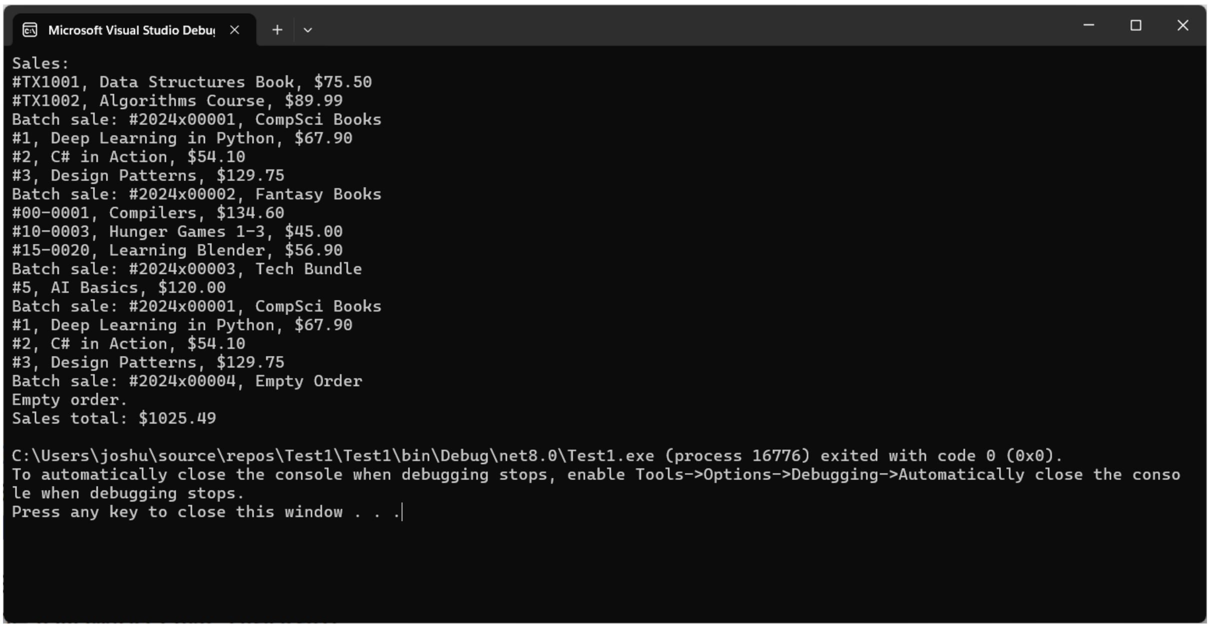
TASK 1

Uml:



Screen output:



```
Sales:
#TX1001, Data Structures Book, $75.50
#TX1002, Algorithms Course, $89.99
Batch sale: #2024x00001, CompSci Books
#1, Deep Learning in Python, $67.90
#2, C# in Action, $54.10
#3, Design Patterns, $129.75
Batch sale: #2024x00002, Fantasy Books
#00-0001, Compilers, $134.60
#10-0003, Hunger Games 1-3, $45.00
#15-0020, Learning Blender, $56.90
Batch sale: #2024x00003, Tech Bundle
#5, AI Basics, $120.00
Batch sale: #2024x00001, CompSci Books
#1, Deep Learning in Python, $67.90
#2, C# in Action, $54.10
#3, Design Patterns, $129.75
Batch sale: #2024x00004, Empty Order
Empty order.
Sales total: $1025.49

C:\Users\joshu\source\repos\Test1\Test1\bin\Debug\net8.0\Test1.exe (process 16776) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

program.cs

```
using System;

using Test1;


class Program

{

    static void Main(string[] args)
```

```
{
    // Create Sales object
    Sales sales = new Sales();

    // Create single transactions and add them to Sales
    Transaction singleTransaction1 = new Transaction("TX1001", "Data Structures Book", 75.50m);
    Transaction singleTransaction2 = new Transaction("TX1002", "Algorithms Course", 89.99m);

    sales.Add(singleTransaction1);
    sales.Add(singleTransaction2);

    // Create batch orders and add items
    Batch batch1 = new Batch("2024x00001", "CompSci Books");
    batch1.Add(new Transaction("1", "Deep Learning in Python", 67.90m));
    batch1.Add(new Transaction("2", "C# in Action", 54.10m));
    batch1.Add(new Transaction("3", "Design Patterns", 129.75m));

    // Create another batch order
    Batch batch2 = new Batch("2024x00002", "Fantasy Books");
    batch2.Add(new Transaction("00-0001", "Compilers", 134.60m));
    batch2.Add(new Transaction("10-0003", "Hunger Games 1-3", 45.00m));
    batch2.Add(new Transaction("15-0020", "Learning Blender", 56.90m));

    // Create a nested batch and add batch1 inside it
    Batch nestedBatch = new Batch("2024x00003", "Tech Bundle");
    nestedBatch.Add(new Transaction("5", "AI Basics", 120.00m));
    nestedBatch.Add(batch1); // Add batch1 as a nested order

    // Add batches and the nested batch to Sales
    sales.Add(batch1);
    sales.Add(batch2);
```

```
        sales.Add(nestedBatch);

        // Add an empty batch
        Batch emptyBatch = new Batch("2024x00004", "Empty Order");
        sales.Add(emptyBatch);

        // Print all orders
        sales.PrintOrders();
    }
}
```

Batch.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Transactions;
using System.Xml.Linq;

namespace Test1
{
    public class Batch : Thing
```

```
{
    private List<Thing> _items;

    public Batch(string number, string name) : base(number, name)
    {
        _items = new List<Thing>();
    }

    public void Add(Thing item)
    {
        _items.Add(item);
    }

    public override void Print()
    {
        Console.WriteLine($"Batch sale: #{_number}, {_name}");
        if (_items.Count == 0)
        {
            Console.WriteLine("Empty order.");
        }
        else
        {
            foreach (Thing item in _items)
            {
                item.Print();
            }
        }
    }

    public override decimal Total()
    {
            decimal total = 0;
            foreach (Thing item in _items)
            {
                total += item.Total();
            }
            return total;
        }
    }

}
```

Sales.cs

```
using Test1;

public class Sales
{
    private List<Thing> _orders;

    public Sales()
    {
        _orders = new List<Thing>();
    }
    public void Add(Thing order)
    {
        _orders.Add(order);
    }

    public void PrintOrders()
    {
        Console.WriteLine("Sales:");
        decimal totalSales = 0;
        foreach (Thing order in _orders)
        {
            order.Print();
            totalSales += order.Total();
        }
        Console.WriteLine($"Sales total: ${totalSales}");
    }
}
```

Thing.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Test1
{
    public abstract class Thing
    {
        protected string _number;
        protected string _name;

        public Thing(string number, string name)
        {
            _number = number;
            _name = name;
        }

        // Abstract methods to be implemented by derived classes
        public abstract void Print();
        public abstract decimal Total();

        // Read-only properties
        public string Number
        {
            get { return _number; }
        }


        public string Name
        {
            get { return _name; }
        }
    }

}
```

Transaction.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Test1
{
  public class Transaction : Thing
  {
    private decimal _amount;

    public Transaction(string number, string name, decimal amount) : base(number, name)
    {
      _amount = amount;
    }

    public override void Print()
    {
      Console.WriteLine($"#{_number}, {_name}, ${_amount}");
    }

    public override decimal Total()
```

```
        {
          return _amount;
        }
    }


}
```

TASK 2

1. Polymorphism is allowing calling the same method on different objects and the different objects will respond with their own implementations. In Task 1, polymorphism is used at *abstract class Thing* which defines methods like Print() and Total() and both Batch and Transaction classes inherit from Thing and provide their own implementation of the method.
2. The name is too general and makes it difficult for someone to understand the purpose in the system. Using the name OrderItem is clearer as the class Thing is focus more on representing individual transactions.
3. Abstraction is hiding complex implementation details and only showing necessary details to user.
   In task 1, abstraction is implemented using the abstract class Thing. This class has common properties such as number and _name, and behaviours like Print() and Total(). These properties and behaviours shared by Batch and Transaction. The specific methods for printing or calculating totals are encapsulated in each subclass, giving flexibility in managing different order types.

   Access modifiers reinforce abstraction by keeping fields like number,_name, and items private to prevent direct access from outside classes. The system exposes only essential behaviours through public methods and read-only properties, supporting encapsulation by protecting each object's internal state while allowing external interaction through a clean, defined interface.

   Abstraction allows the system to manage various order types without needing to know their specifics. This design simplifies Sales class and allow it to work with Thing objects without distinguishing between Batch and Transaction. Abstraction creates flexible, scalable and maintainable system, encouraging clear separation of concerns and reusability.

4. A real world system example would be an inventory management system that handles individual products and bundled group items.

   Transaction class: represents individual item in inventory

   Batch class: represent collection of items like a bundle of products

   Sales class: manage overall inventory by adding both individual items (transaction class) and a bundle of products (batch) and then calculate the total value of inventory and print.