OODP Design Patterns Assignment

Author: *Alan O'Regan*
Student Number: *B00133474*

Application Overview:

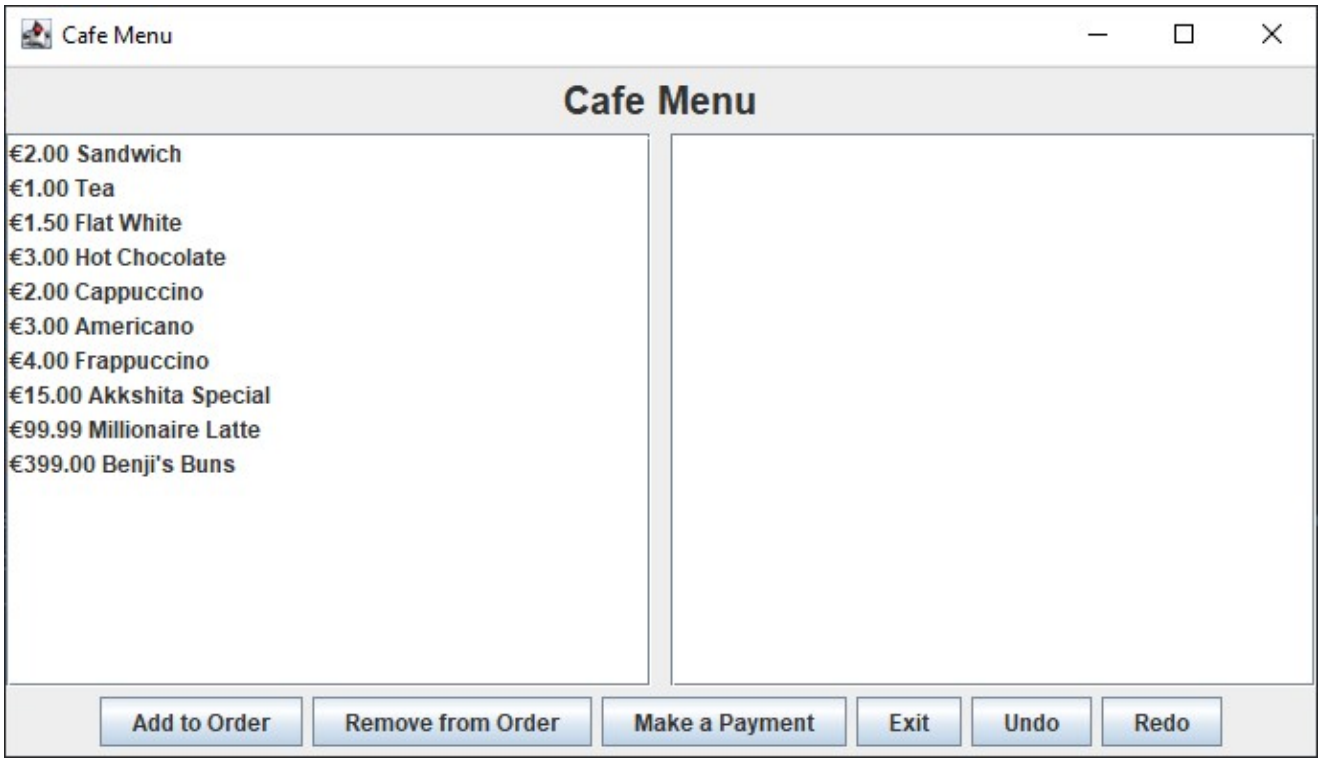
This is a point of sale computer program for a coffee shop.
The program displays the menu from an inventory file,allows the user to process orders and then save them to a transaction file.

Functional Requirements

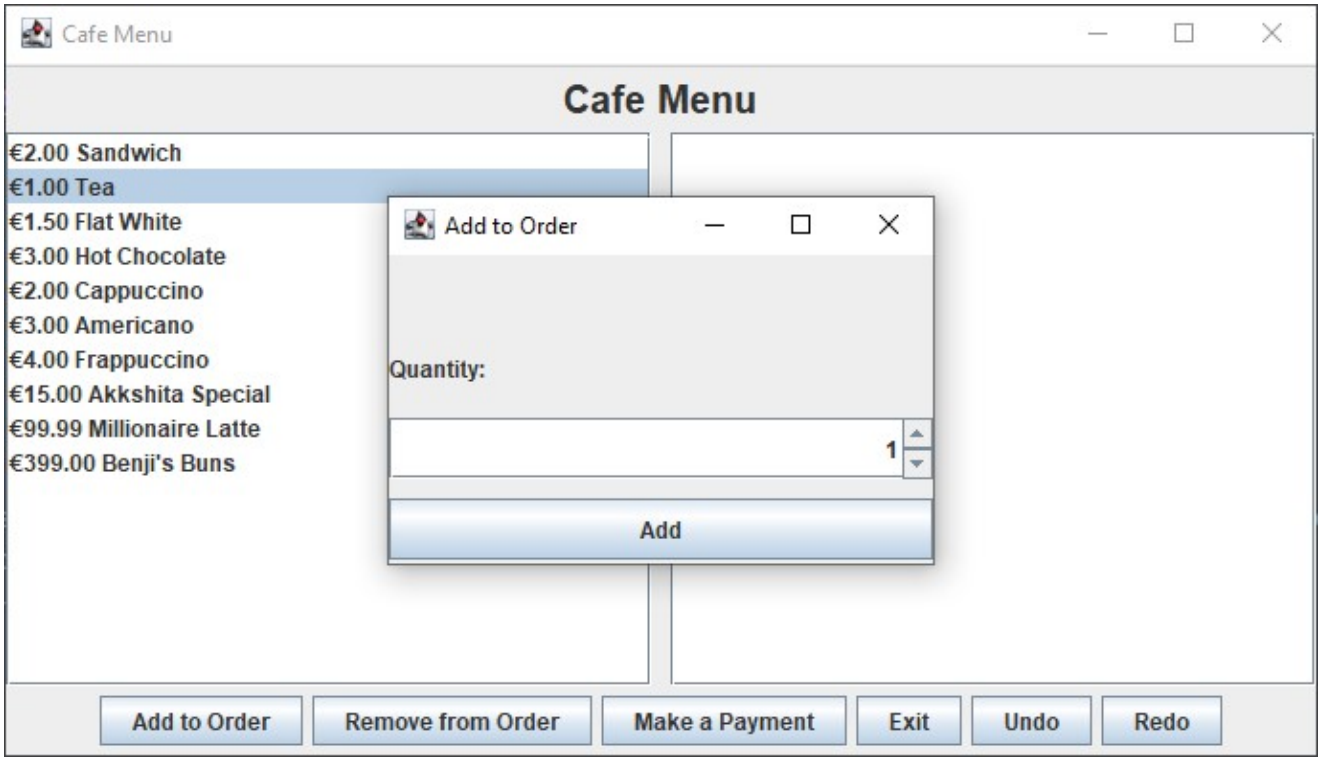
- [x] ~~Display Menu from Inventory File~~
- [x] ~~Add Items to Order~~
- [x] ~~Remove Items from Order~~
- [x] ~~Display Order~~
- [x] ~~Display Total~~
- [x] ~~Take Payment~~
- [x] ~~Display Receipt~~
- [x] ~~Save Order to Transaction File~~

User Requirements and Interface

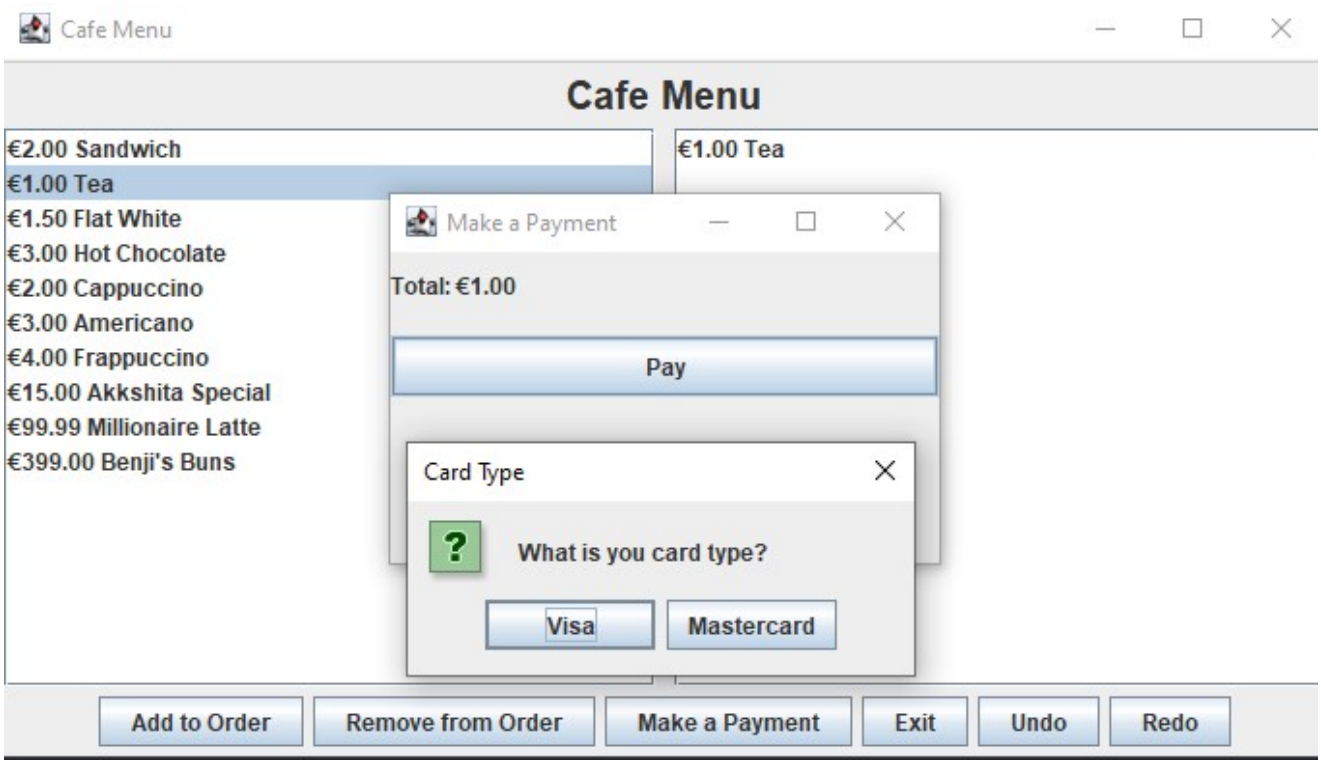
Home Screen



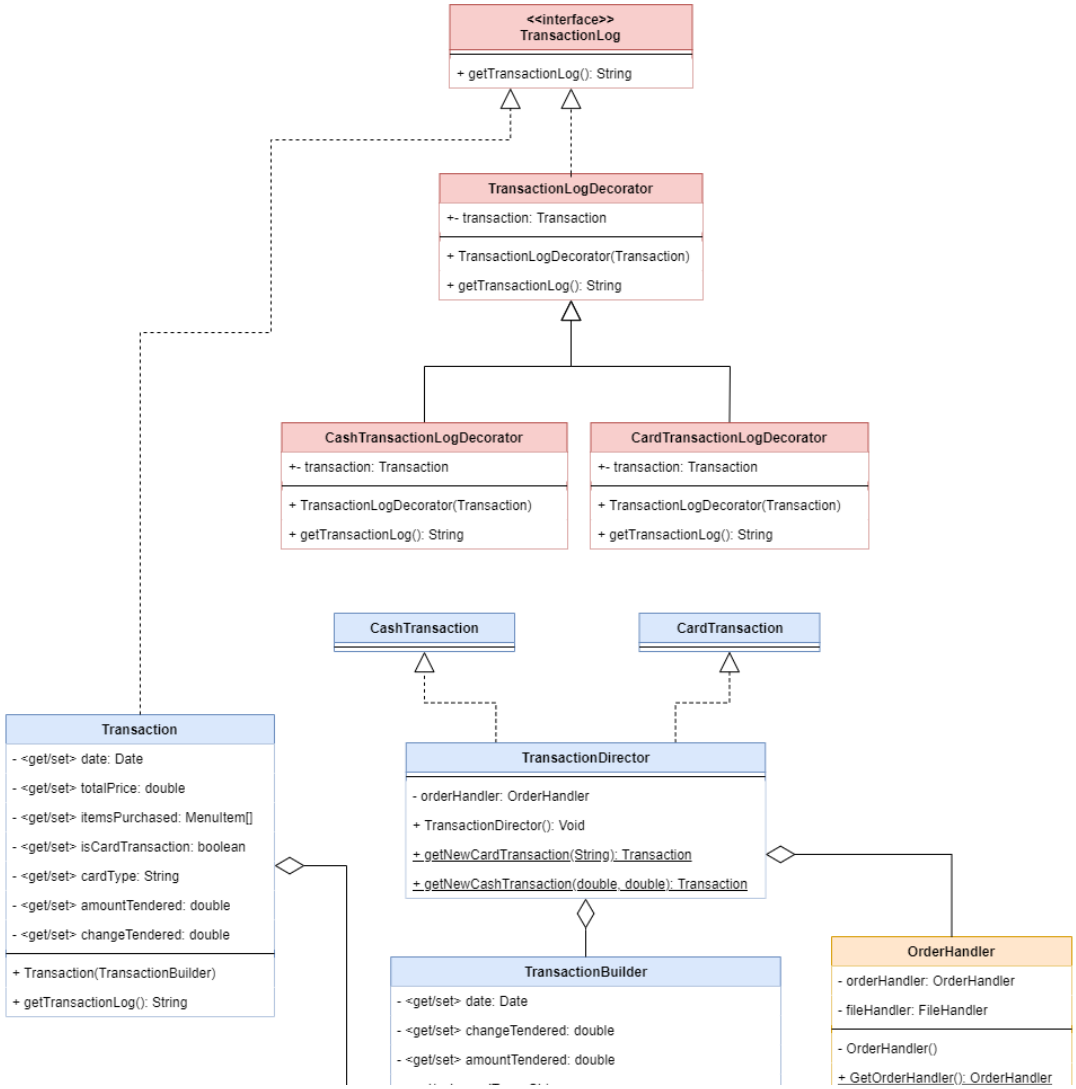
Add item to the order

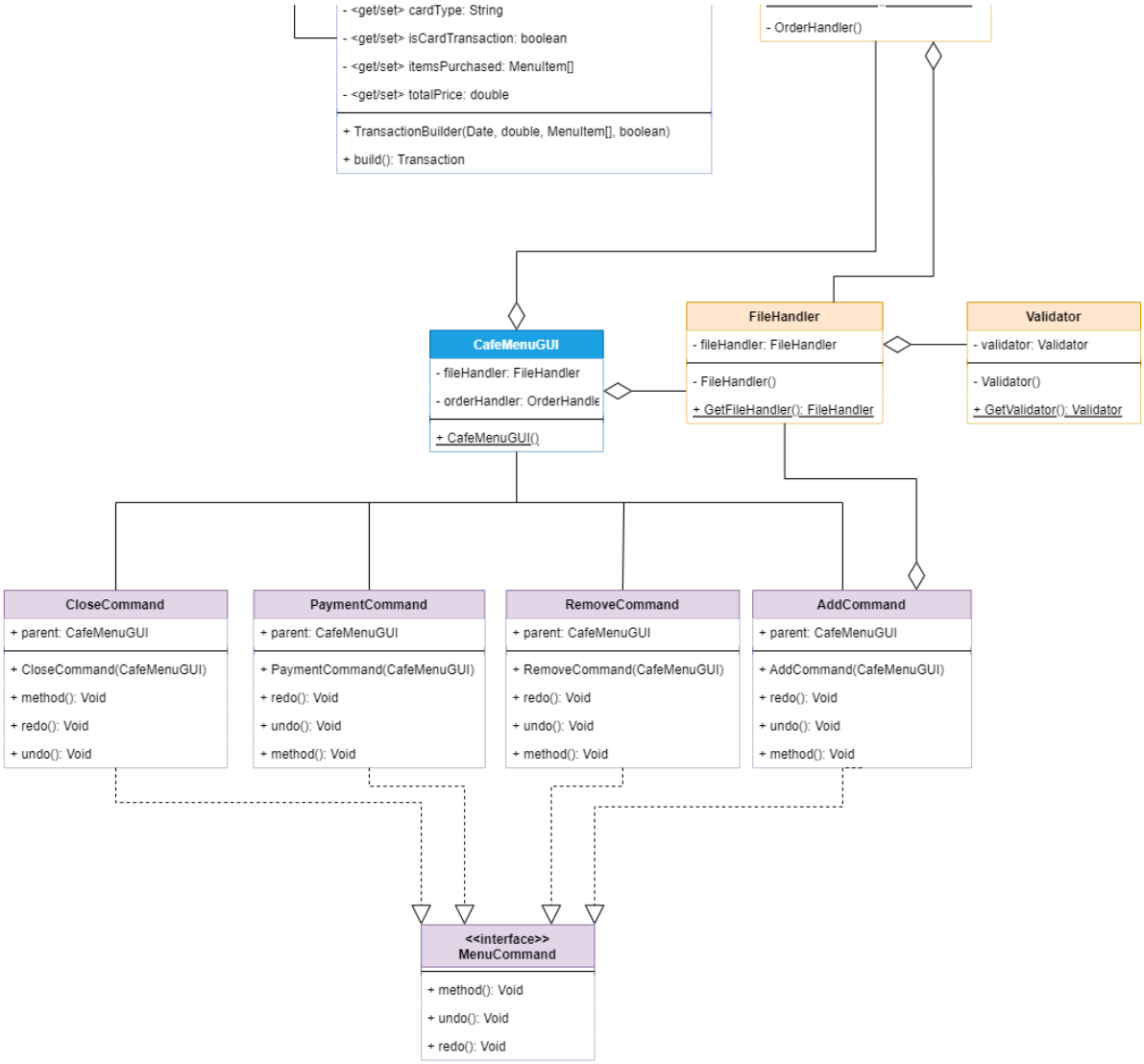


Pay for the order



UML Class Diagram





Design Patterns

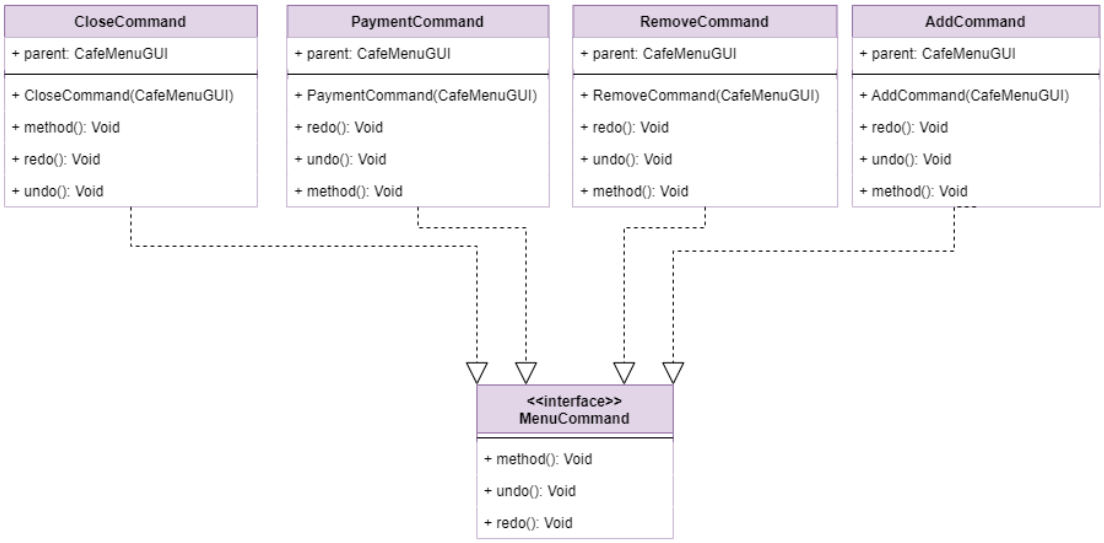
The design patterns I chose to implement in this project are as follows:

Behavioural

Command

For the Behavioural pattern I implemented the command pattern as it allows the user to undo and redo actions.
This is useful for a point of sale system as it allows the user to undo an order if they make a mistake or change their mind.

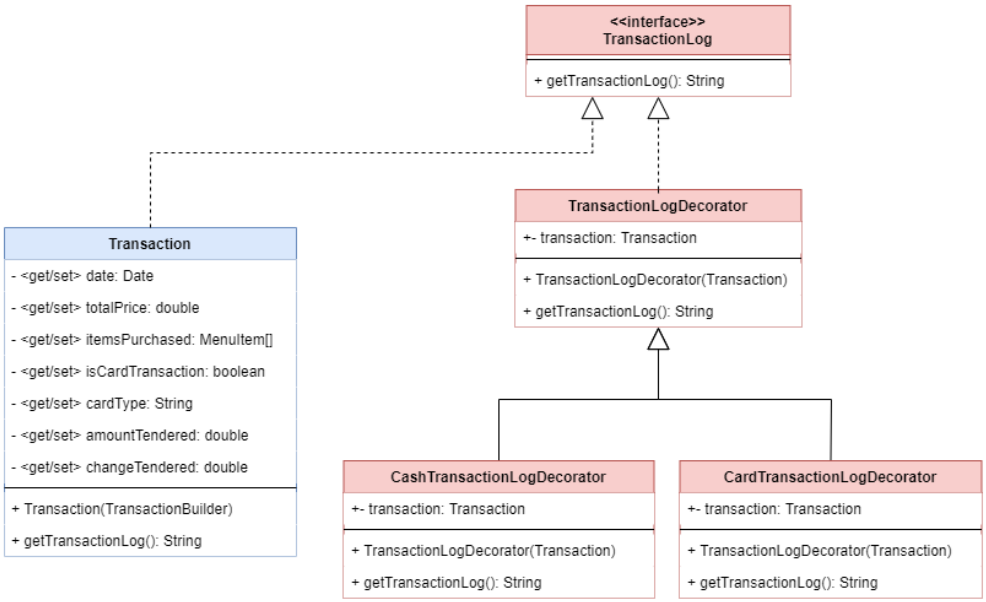
It also allows the user to redo an order if they accidentally undo an order.



Structural

Decorator

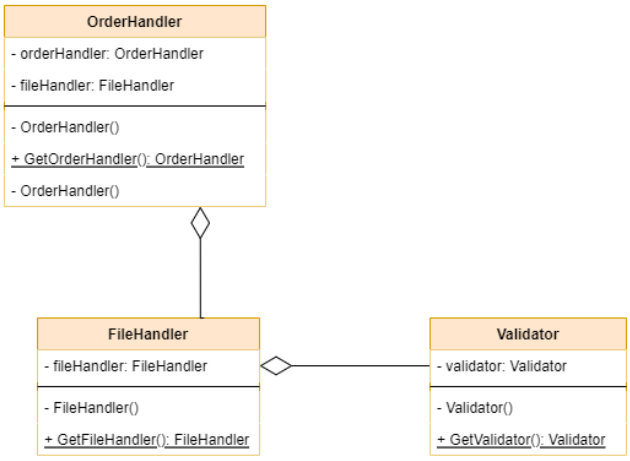
For the structural pattern I implemented the decorator pattern as it makes it easier to modify the transaction log depending on the transaction type.



Creational

Singletons

For the creational pattern I implemented the singleton pattern as it allows for only one instance of FileHandler to be created.
Because it is synchronised it is thread safe and can be used in a multi-threaded environment.



As I was implementing the singleton pattern I thought why not make it a static class, and so I looked into the idea and found this:

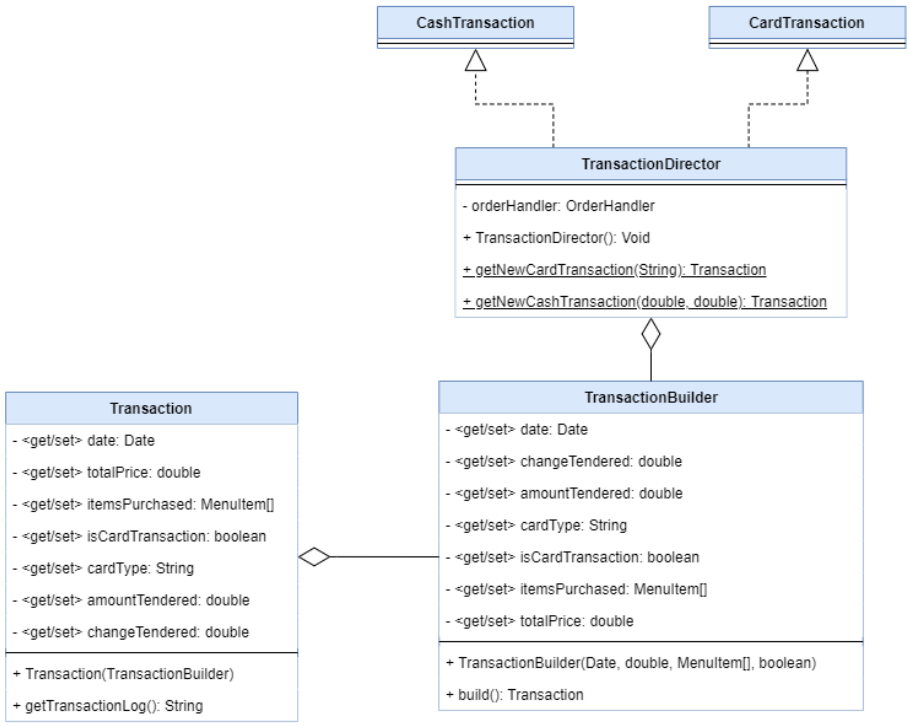
> A Singletons can implement interfaces, inherit from other classes and allow inheritance. While a static class cannot inherit their instance members. So Singletons is more flexible than static classes and can maintain state.

_source https://net-informations.com/faq/netfaq/singlestatic.htm

Builder

I also implemented the builder pattern as it allows for the creation of a transaction object with different attributes depending on the transaction

type.


CashTransaction
CardTransaction
TransactionDirector
- orderHandler: OrderHandler
+ TransactionDirector(): Void
+ getNewCardTransaction(String): Transaction
+ getNewCashTransaction(double, double): Transaction
Transaction
- <get/set> date: Date
- <get/set> totalPrice: double
- <get/set> itemsPurchased: MenuItem[]
- <get/set> isCardTransaction: boolean
- <get/set> cardType: String
- <get/set> amountTendered: double
- <get/set> changeTendered: double
+ Transaction(TransactionBuilder)
+ getTransactionLog(): String
TransactionBuilder
- <get/set> date: Date
- <get/set> changeTendered: double
- <get/set> amountTendered: double
- <get/set> cardType: String
- <get/set> isCardTransaction: boolean
- <get/set> itemsPurchased: MenuItem[]
- <get/set> totalPrice: double
+ TransactionBuilder(Date, double, MenuItem[], boolean)
+ build(): Transaction