CS4227 Assignment 1 – Interceptor

Quinn Painter
19234201

Following Steps 1-7

1 - Model the Internal Behaviour of the Framework

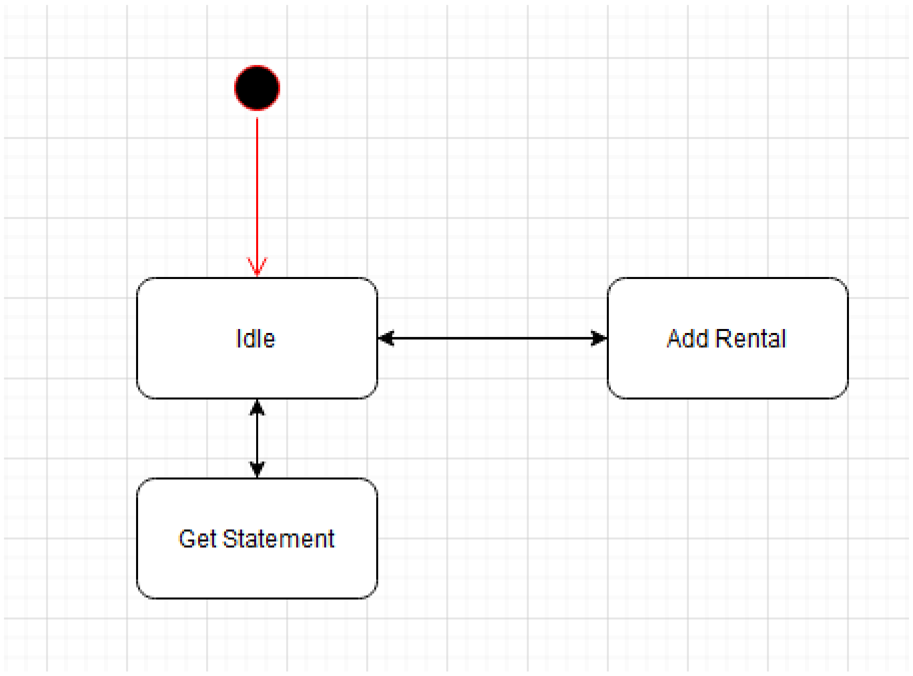


Figure 1 - State machine for the Movie System

2 - Identify and Model Interception Points

As seen in the state diagram, the movie rental code doesn't have significant internal state transitions that would be subject to interception, so we'll just use an externally visible state transition as an interception point – AddRental.

The interception will provide information about the rental addition for logging etc., so it only needs to be a Reader.

As there is only one interception point, there will only be one interception group.

3 - Specify the Context Objects

As there is only one interception point, there only needs to be one context object.

The AddRental interception will need information about the rental that was added, so the context object can have the following functions:

public string getRentalTitle()
public double getRentalCharge()
public string getCustomerName()

As these pieces of information change for each new rental that is added, the context object should be passed per-event.

4 - Specify the Interceptors

See figure 4.

5 - Specify the Dispatchers

See figure 7.

6 - Implement the Callback Mechanisms in the Concrete Framework

See figure 7.

7 - Implement the Concrete Interceptors

See figures 5 and 6.

Sequence Diagram

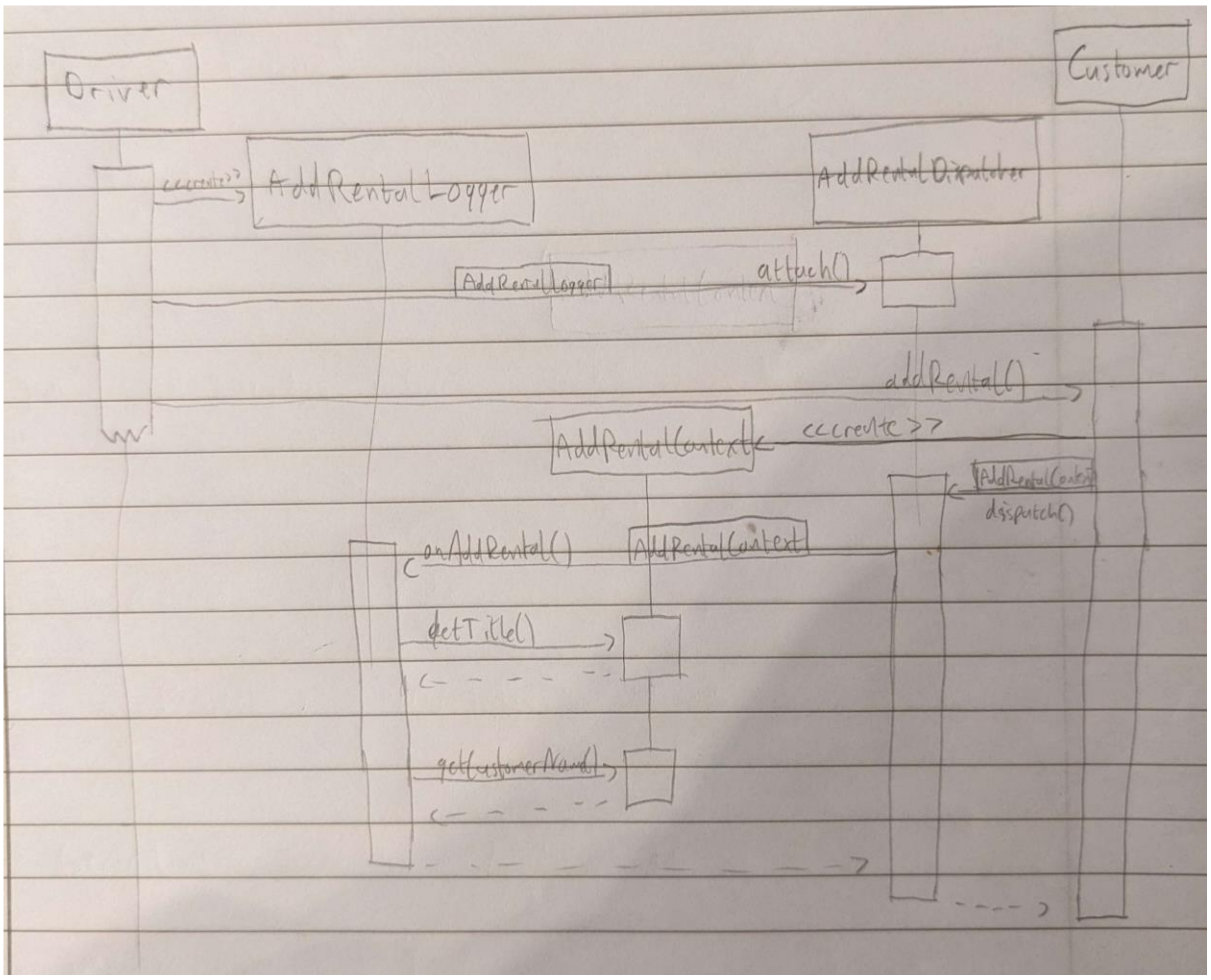
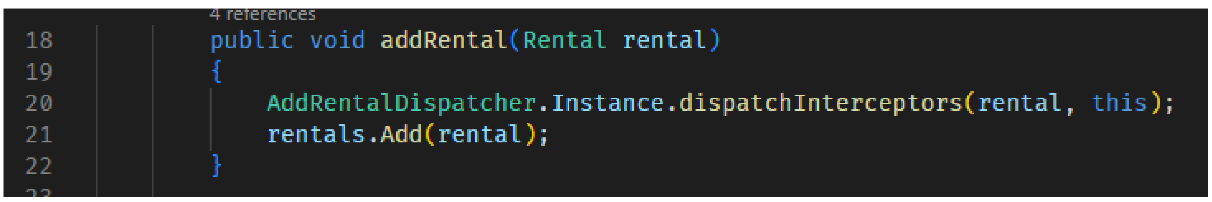


Figure 2 - Sequence diagram of program execution

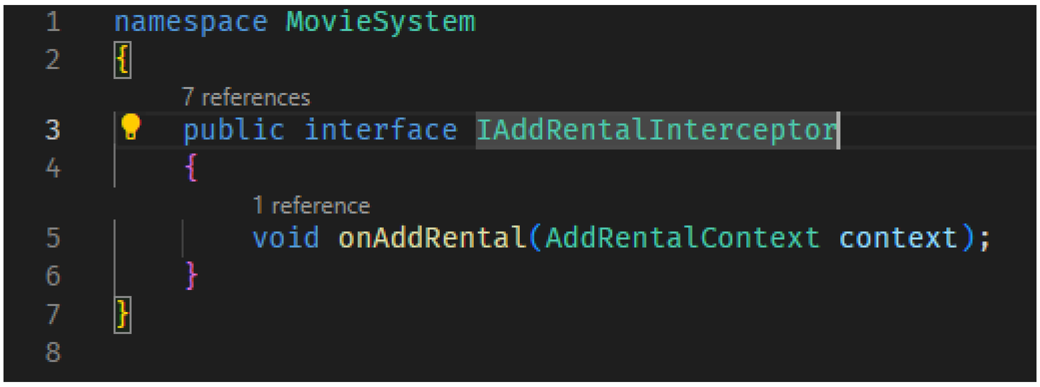
Code

Implementation is based on the Movie System example from *Refactoring* (Fowler et al., 2012)



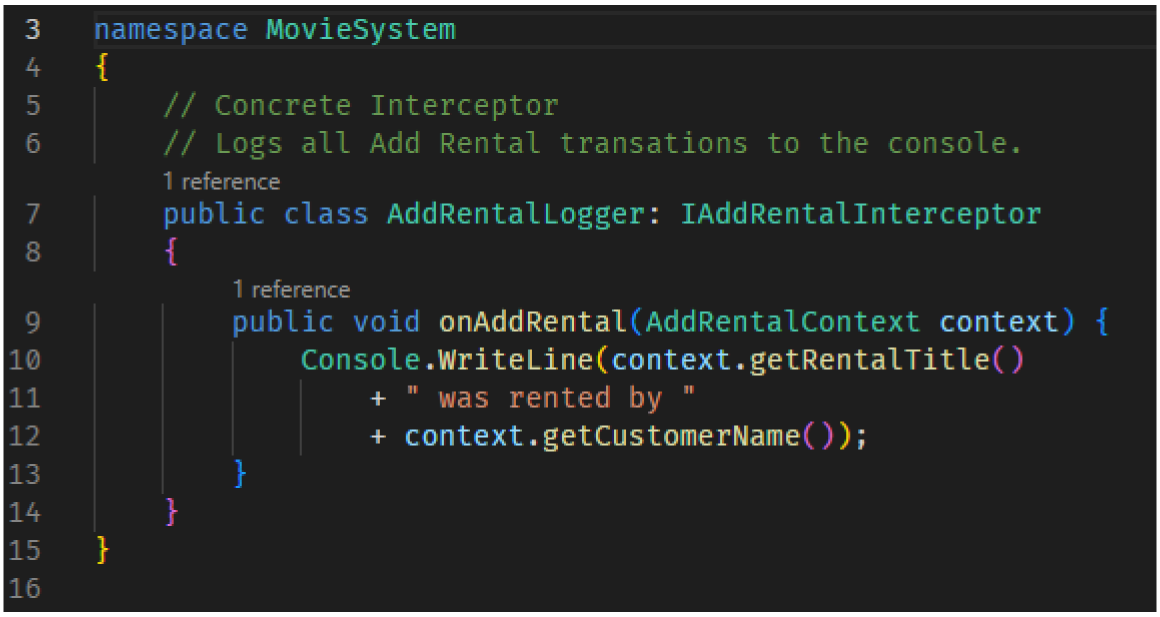
```
        4 references
        public void addRental(Rental rental)
        {
            AddRentalDispatcher.Instance.dispatchInterceptors(rental, this);
            rentals.Add(rental);
        }
```

Figure 3 - Add Rental function in the Customer class



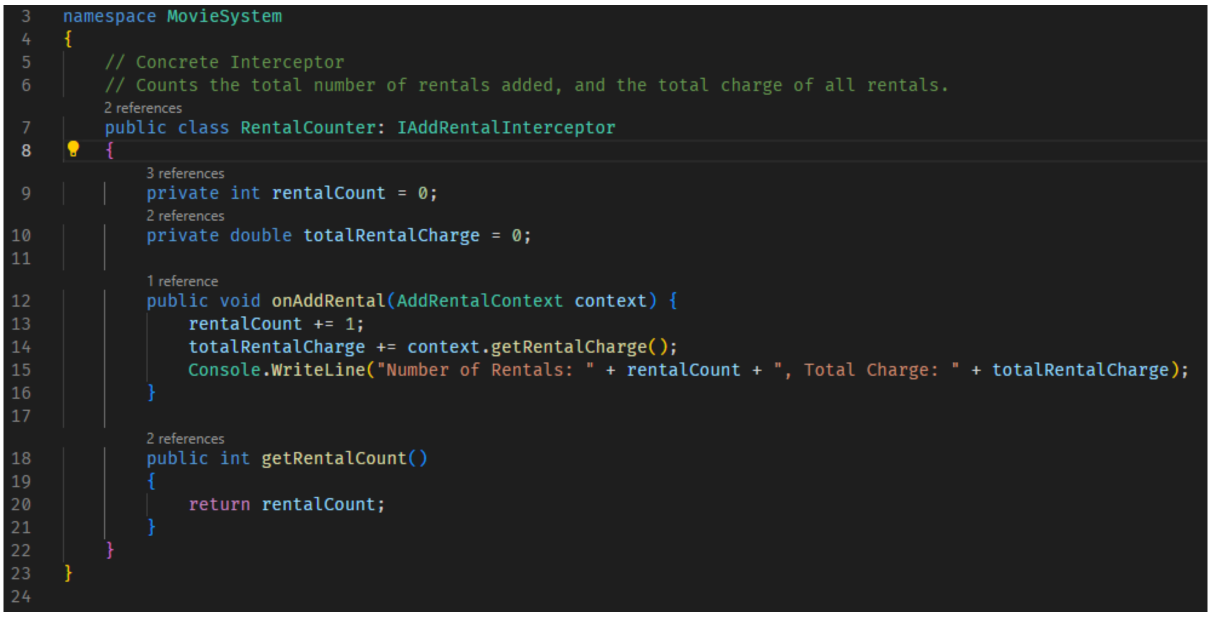
```
namespace MovieSystem
{
    7 references
    public interface IAddRentalInterceptor
    {
        1 reference
        void onAddRental(AddRentalContext context);
    }
}
```

Figure 4 - Interceptor



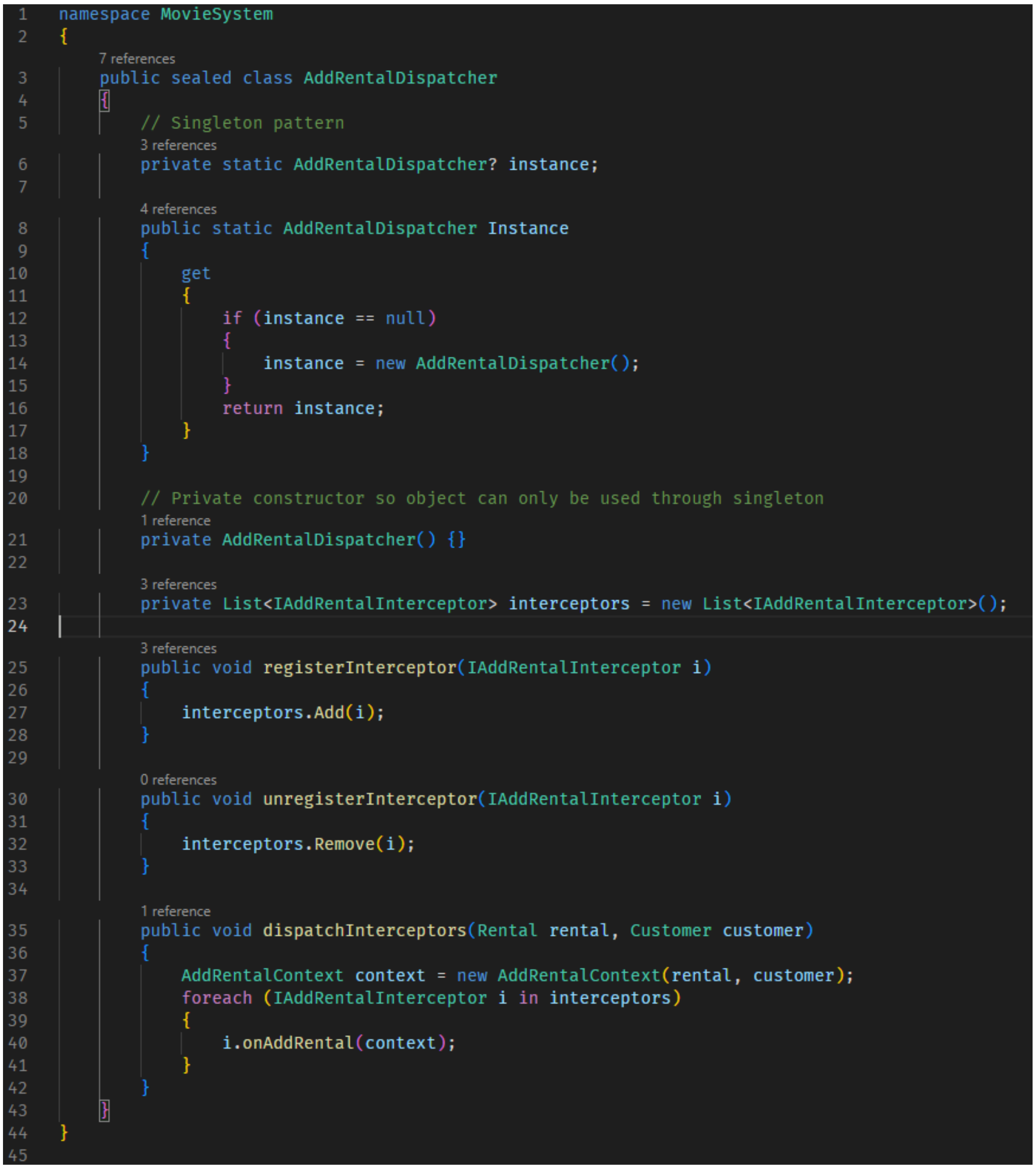
```
namespace MovieSystem
{
    // Concrete Interceptor
    // Logs all Add Rental transations to the console.
    1 reference
    public class AddRentalLogger: IAddRentalInterceptor
    {
        1 reference
        public void onAddRental(AddRentalContext context) {
            Console.WriteLine(context.getRentalTitle()
                + " was rented by "
                + context.getCustomerName());
        }
    }
}
```

Figure 5 - Concrete Interceptor 1



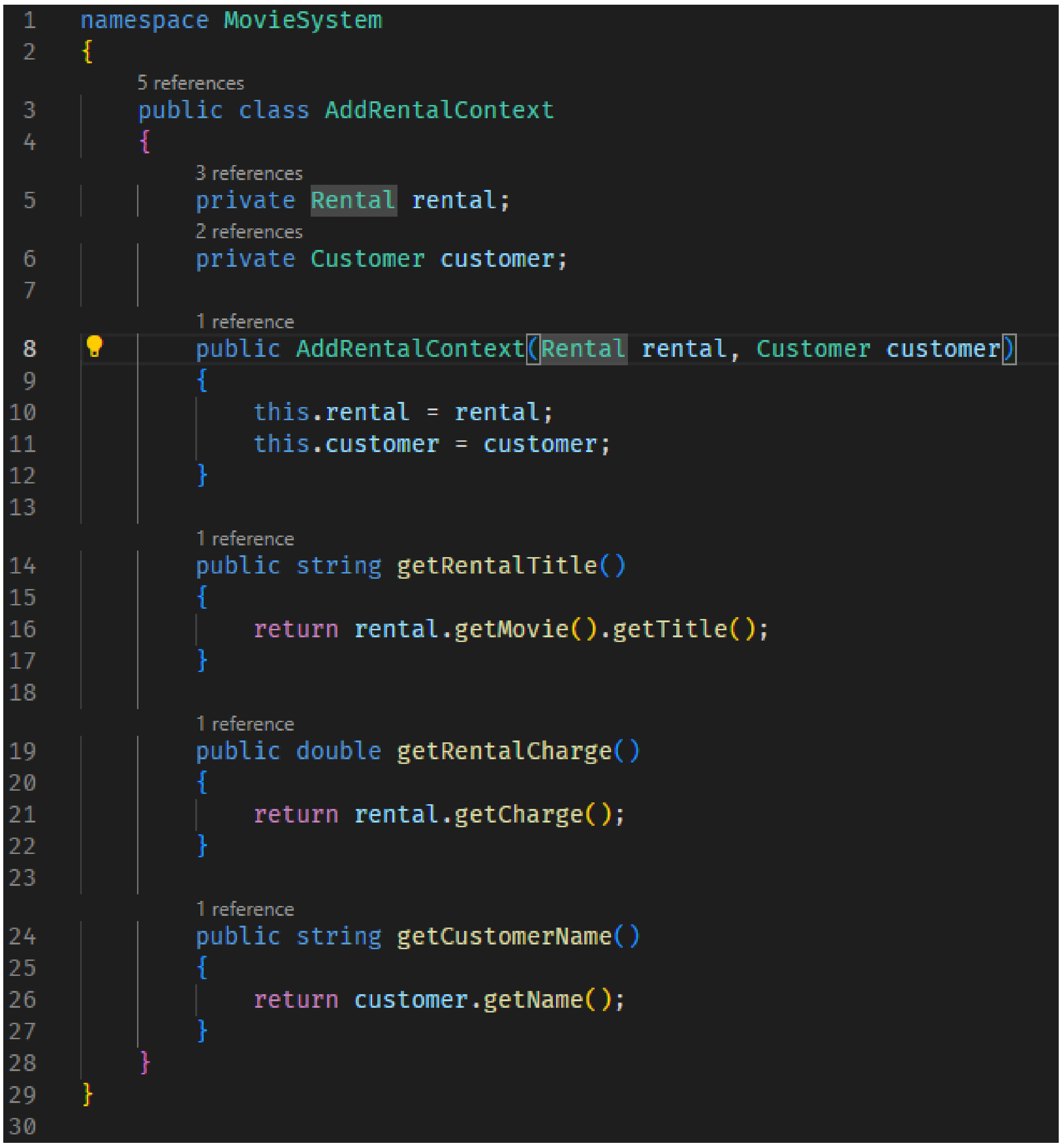
```
namespace MovieSystem
{
    // Concrete Interceptor
    // Counts the total number of rentals added, and the total charge of all rentals.
    2 references
    public class RentalCounter: IAddRentalInterceptor
    {
        3 references
        private int rentalCount = 0;
        2 references
        private double totalRentalCharge = 0;

        1 reference
        public void onAddRental(AddRentalContext context) {
            rentalCount += 1;
            totalRentalCharge += context.getRentalCharge();
            Console.WriteLine("Number of Rentals: " + rentalCount + ", Total Charge: " + totalRentalCharge);
        }

        2 references
        public int getRentalCount()
        {
            return rentalCount;
        }
    }
}

```

Figure 6 - Concrete Interceptor 2



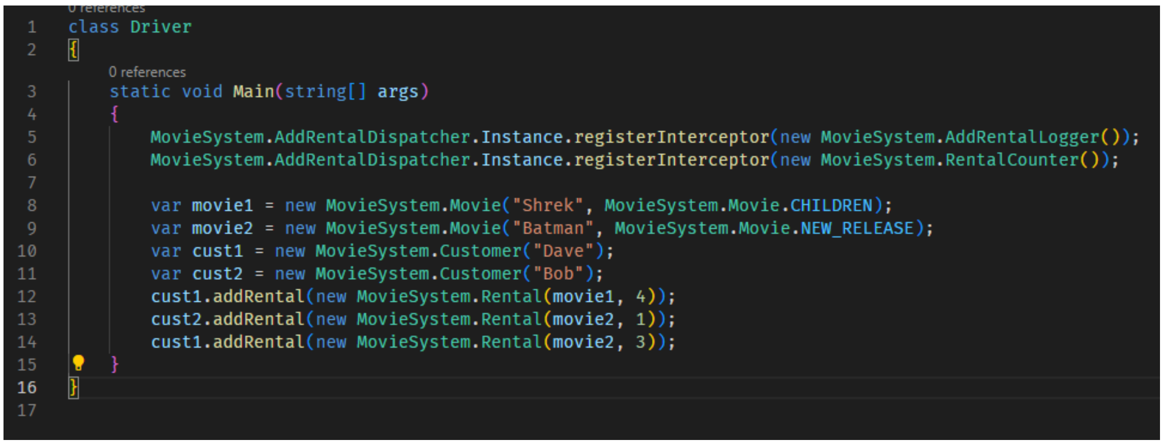
```
namespace MovieSystem
{
    7 references
    public sealed class AddRentalDispatcher
    {
        // Singleton pattern
        3 references
        private static AddRentalDispatcher? instance;

        4 references
        public static AddRentalDispatcher Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new AddRentalDispatcher();
                }
                return instance;
            }
        }

        // Private constructor so object can only be used through singleton
        1 reference
        private AddRentalDispatcher() {}

        3 references
        private List<IAddRentalInterceptor> interceptors = new List<IAddRentalInterceptor>();

        3 references
        public void registerInterceptor(IAddRentalInterceptor i)
        {
            interceptors.Add(i);
        }

        0 references
        public void unregisterInterceptor(IAddRentalInterceptor i)
        {
            interceptors.Remove(i);
        }

        1 reference
        public void dispatchInterceptors(Rental rental, Customer customer)
        {
            AddRentalContext context = new AddRentalContext(rental, customer);
            foreach (IAddRentalInterceptor i in interceptors)
            {
                i.onAddRental(context);
            }
        }
    }
}

```

Figure 7 - Dispatcher



```
namespace MovieSystem
{
    5 references
    public class AddRentalContext
    {
        3 references
        private Rental rental;
        2 references
        private Customer customer;

        1 reference
        public AddRentalContext(Rental rental, Customer customer)
        {
            this.rental = rental;
            this.customer = customer;
        }

        1 reference
        public string getRentalTitle()
        {
            return rental.getMovie().getTitle();
        }

        1 reference
        public double getRentalCharge()
        {
            return rental.getCharge();
        }

        1 reference
        public string getCustomerName()
        {
            return customer.getName();
        }
    }
}
```

Figure 8 - Context Object



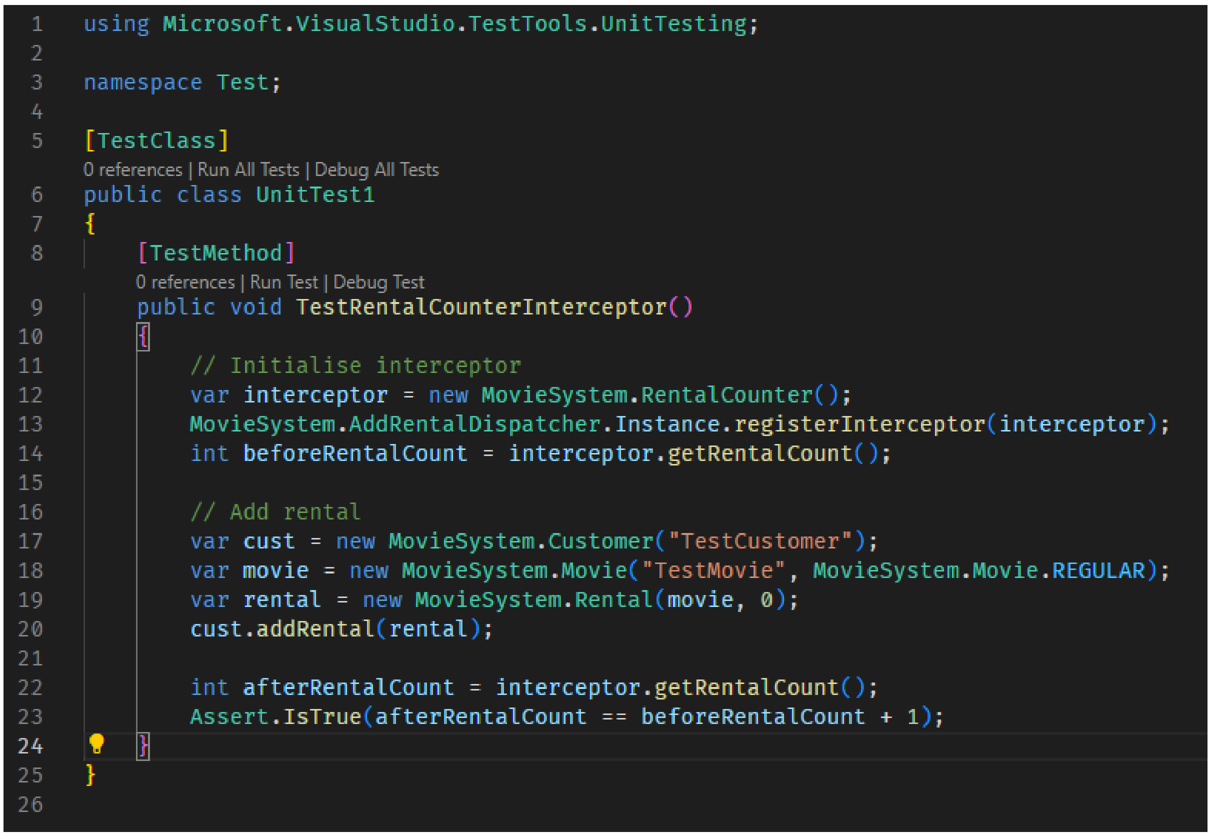
```
0 references
class Driver
{
    0 references
    static void Main(string[] args)
    {
        MovieSystem.AddRentalDispatcher.Instance.registerInterceptor(new MovieSystem.AddRentalLogger());
        MovieSystem.AddRentalDispatcher.Instance.registerInterceptor(new MovieSystem.RentalCounter());

        var movie1 = new MovieSystem.Movie("Shrek", MovieSystem.Movie.CHILDREN);
        var movie2 = new MovieSystem.Movie("Batman", MovieSystem.Movie.NEW_RELEASE);
        var cust1 = new MovieSystem.Customer("Dave");
        var cust2 = new MovieSystem.Customer("Bob");
        cust1.addRental(new MovieSystem.Rental(movie1, 4));
        cust2.addRental(new MovieSystem.Rental(movie2, 1));
        cust1.addRental(new MovieSystem.Rental(movie2, 3));
    }
}
```

Figure 9 - Driver

All other code (Rental, Price, RegularPrice, NewReleasePrice, ChildrensPrice, rest of Customer) is unchanged from the Movie System example.

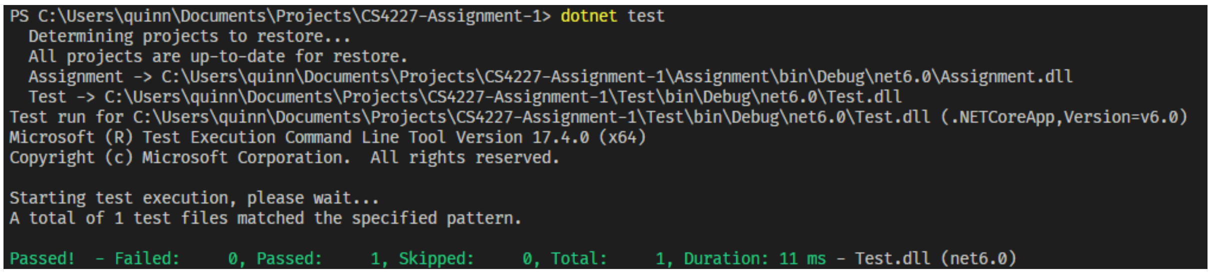
Git Repo: https://github.com/QuinnPainter/CS4227-Assignment-1

Test Case



```csharp
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Test;

[TestClass]
public class UnitTest1
{
    [TestMethod]
    public void TestRentalCounterInterceptor()
    {
        // Initialise interceptor
        var interceptor = new MovieSystem.RentalCounter();
        MovieSystem.AddRentalDispatcher.Instance.registerInterceptor(interceptor);
        int beforeRentalCount = interceptor.getRentalCount();

        // Add rental
        var cust = new MovieSystem.Customer("TestCustomer");
        var movie = new MovieSystem.Movie("TestMovie", MovieSystem.Movie.REGULAR);
        var rental = new MovieSystem.Rental(movie, 0);
        cust.addRental(rental);

        int afterRentalCount = interceptor.getRentalCount();
        Assert.IsTrue(afterRentalCount == beforeRentalCount + 1);
    }
}
```

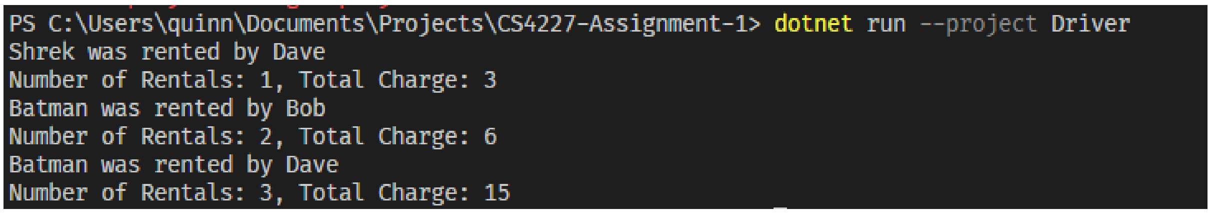
Figure 10 - Test Case Code



```
PS C:\Users\quinn\Documents\Projects\CS4227-Assignment-1> dotnet test
  Determining projects to restore...
  All projects are up-to-date for restore.
  Assignment -> C:\Users\quinn\Documents\Projects\CS4227-Assignment-1\Assignment\bin\Debug\net6.0\Assignment.dll
  Test -> C:\Users\quinn\Documents\Projects\CS4227-Assignment-1\Test\bin\Debug\net6.0\Test.dll
Test run for C:\Users\quinn\Documents\Projects\CS4227-Assignment-1\Test\bin\Debug\net6.0\Test.dll (.NETCoreApp,Version=v6.0)
Microsoft (R) Test Execution Command Line Tool Version 17.4.0 (x64)
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:     1, Skipped:     0, Total:     1, Duration: 11 ms - Test.dll (net6.0)
```

Figure 11 - Test case running and passing

Code compiling and running



```
PS C:\Users\quinn\Documents\Projects\CS4227-Assignment-1> dotnet run --project Driver
Shrek was rented by Dave
Number of Rentals: 1, Total Charge: 3
Batman was rented by Bob
Number of Rentals: 2, Total Charge: 6
Batman was rented by Dave
Number of Rentals: 3, Total Charge: 15
```

Figure 12 - Running the code

Evaluation

The interceptor pattern is useful to add functionality to a framework without having to modify the framework itself. It can help to decouple concerns, and promote usability of interceptors. However, it has the downside that it can add significant complexity to a system and make it harder to maintain.

References

Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. (2012). *Refactoring*. Addison-Wesley.