C# Code Review Task

Code Review:

The systematic examination of code is intended to identify and address any mistakes that may have been overlooked during the initial development phase, improving both the overall quality of software and well as the developers' skills.

Deliverables: a single .cs file from task 1 and a solution containing two projects from task 2

Task 1

The purpose of this task is to examine the code (below) that a work colleague has written.

- You must critically analyse and report on the quality and review it for any mistakes, bugs or issues that you feel are present.
- You should make any comments or suggestions that you feel are appropriate about style, design and logic etc.
- You are encouraged to be bold.

Copy the source into an editor of your choice. Make your comments and save as a single .cs file

There is no time or word limit but try to not spend too long completing the task – a good guideline is about 30-40 minutes.

This example is massively contrived and intentionally badly coded; we don't expect any coding you do to look anything like this!

General comments on the existing code and design:

Task 2

Create a solution with two projects. The first, a refactored version of the code as how you would write it. The second a test project containing some suitable unit tests

```
public class OrderManager
{
    private readonly IOrderStore orderStore;

    public OrderManager(IOrderStore orderStore)
    {
        this.orderStore = orderStore;
    }

    public void WriteOutSmallOrders()
    {
        var orders = orderStore.GetOrders();
        SmallOrderFilter filter = new SmallOrderFilter(new OrderWriter(),
orders);
        filter.WriteOutFiltrdAndPriceSortedOrders(new OrderWriter());
    }

    public void WriteOutLargeOrders()
    {
        var orders = orderStore.GetOrders();
        LargeOrderFilter filter = new LargeOrderFilter(new OrderWriter(),
orders);
        filter.WriteOutFiltrdAndPriceSortedOrders(new OrderWriter());
    }
}


public class LargeOrderFilter
{
    private IOrderWriter orderWriter;
    private List<Order> orders;

    public LargeOrderFilter(IOrderWriter orderWriter, List<Order> orders)
    {
        filterSize = "100";
        this.orderWriter = orderWriter;
        this.orders = orders;
    }

    protected string filterSize;

    public void WriteOutFiltrdAndPriceSortedOrders(IOrderWriter writer)
    {
        List<Order> filteredOrders = this.FilterOrdersSmallerThan(orders,
filterSize);
        Enumerable.OrderBy(filteredOrders, o => o.Price);

        ObservableCollection<Order> observableCollection =
            new ObservableCollection<Order>();

        foreach (Order o in filteredOrders)
        {
            observableCollection.Add(o);
        }

        writer.WriteOrders(observableCollection);
    }
    protected List<Order> FilterOrdersSmallerThan(List<Order> allOrders,
string size)
    {
        List<Order> filtered = new List<Order>();
        for (int i = 0; i <= allOrders.Count; i++)
        {
            int number = orders[i].toNumber(size);

            if (allOrders[i].Size <= number)
            {
                continue;
            }
            else
            {
                filtered.Add(orders[i]);
            }
        }

        return filtered;
    }
}

public class SmallOrderFilter : LargeOrderFilter
{
    public SmallOrderFilter(IOrderWriter orderWriter, List<Order> orders)
        : base(orderWriter, orders)
    {
        filterSize = "10";
    }
}


public class Order
{
    public double Price
    {
        get { return this.dPrice; }
        set { this.dPrice = value; }
    }

    public int Size
    {
        get { return this.iSize; }
        set { this.iSize = value; }
    }

    public string Symbol
    {
        get { return this.sSymbol; }
        set { this.sSymbol = value; }
    }

    private double dPrice;
    private int iSize;
    private string sSymbol;

    public int toNumber(String Input)
    {
        bool canBeConverted = false;
        int n = 0;
        try
        {
            n = Convert.ToInt32(Input);
            if (n != 0) canBeConverted = true;
        }
        catch (Exception ex)
        {
        }

        if (canBeConverted == true)
        {
            return n;
        }
        else
        {
            return 0;
        }
    }
}


// These are stub interfaces that already exist in the system
// They're out of scope of the code review
public interface IOrderWriter
{
    void WriteOrders(IEnumerable<Order> orders);
}

public class OrderWriter : IOrderWriter
{
    public void WriteOrders(IEnumerable<Order> orders)
    {
    }
}

public interface IOrderStore
{
    List<Order> GetOrders();
}
```