Task 1

Разработать класс для хранения объектов двух классов (из Inheritance Task 1):

– базовый класс, описывающий покупку Purchase,

– производный класс, описывающий покупку со скидкой PricePurchase.

Входной файл в *.csv* формате содержит последовательность строк. Каждая строка содержит информацию об одной покупке в виде значений, разделённых «;». Тип покупки (со скидкой или без скидки) определяется количеством значений в строке. Строки с ошибочными данными программа должна пропускать.

Поля класса:

– коллекция (ссылка на List<Purchase>);

Конструкторы:

– по умолчанию,

– с параметром – имя *.csv* файла, из которого данные должные загрузиться в коллекцию. Коллекция должна остаться пустой, если файл не удалось прочитать.

Методы:

– доступ к элементам коллекции и самой коллекции;

– private Purchase createPurchase(string csvString) – создание покупки по строке в формате csv;

– void Insert(int index, Purchase p) – вставка покупки в позицию index, если index неверное значение, то добавить покупку в конец коллекции;

– int Delete(int index) – удаление покупки по индексу index; если ошибочный index , вернуть −1 (*минус один*), иначе – index;

– decimal или int TotalCost( ) – вычисление общей стоимости всех покупок в коллекции;

– void Print( ) – вывод коллекции в виде таблицы + итоговая стоимость в последнюю строку (см. пример ниже);

– public void Sort(IComparer<Purchase> cmp) – сортировка коллекции выбранным способом (параметр cmp);

Разработать класс для сравнения (нужен в сортировке Sort()), реализующий интерфейс IComparer<Purchase>.

Метод:

– public int Compare(Purchase x, Purchase y) – сравнение покупок по возрастанию названия, если две покупки одинаковые, то тогда покупка базового класса всегда меньше покупки производного, если классы покупок равны, то тогда сравнить по возрастанию стоимости.

В классе программы в методе Main(string[] args) имена файлов с данными брать из параметров командной строки `args`. Установка значений параметров в Visual Studio выполняется через меню Project | Properties… вкладка Debug поле Command line arguments. В русской версии Проект | Свойства… вкладка Отладка поле Аргументы командной строки. В командной строке имена файлов должны указываться *без расширения* .csv.

В командной строке приложению передаются имена двух файлов:

– основной файл данных в csv формате;

– дополнительный файл данных в csv формате.

В основном файле могут быть некорректные строки, которые программа должна пропускать.

Дополнительный файл будет корректным и содержит пять строк с данными (см. ниже).

Алгоритм приложения (функция Main( )):

– создать объект разработанного класса и загрузить в него данные из основного файла;

– вывести коллекцию покупок на консоль (метод Print());

– создать второй объект разработанного класса и загрузить в него данные из дополнительного файла;

– вставить *последний* элемент второй коллекции в позицию 0 первой коллекции покупок;

– вставить *первый* элемент второй коллекции в позицию 1000 первой коллекции покупок;

– вставить элемент с позицией 2 из второй коллекции в позицию 2 первой коллекции покупок;

– последовательно удалить элементы через метод Delete(…) с индексами 3, 10 и –5;

– вывести первую коллекцию на экран;

– отсортировать первую коллекцию методом Sort(…);

– вывести первую коллекцию на экран;

– найти через метод binarySearch( ) коллекции List<…> элементы с индексами 1 и 3 из второй коллекции в первой коллекции и вывести результаты поиска.

Пример. Аргументы командной строки:

in addon

Пример файла in.csv:

```
bread;1550;1;20
milk;1310;2
bread;1540;3
candy
;1000;2
candy;1000;2;5000
water;150;4;0.1;cold
bread;1450;5
potato;1800;2;100
butter;3700;1
water;ok;4
water;700;4;0.5
water;700.5;1
butter;3410;1;10
meat;18000;2;800
```

Пример файла addon.csv:

```
choco;7200;2;200
choco;14000;1
meat;21250;2;50
choco;7300;2;300
potato;1750;2;50
```

Вывод:

<Описания ошибок в данных>

После создания

Name	Price	Number	Cost	Discount

Bread	1 550	1	1 530	20
Milk	1 310	2	2 620	
Bread	1 540	3	4 620	
Bread	1 450	5	7 250	
Potato	1 800	2	3 400	100
Butter	3 700	1	3 700	
Butter	3 410	1	3 400	10
Meat	18 000	2	34 400	800
Total cost			60 920	

До сортировки

Name	Price	Number	Cost	Discount
potato	1 750	2	3 400	50
bread	1 550	1	1 530	20
meat	21 250	2	42 400	50
bread	1 540	3	4 620	
bread	1 450	5	7 250	
potato	1 800	2	3 400	100
butter	3 700	1	3 700	
butter	3 410	1	3 400	10
meat	18 000	2	34 400	800
choco	7 200	2	14 000	200
Total cost			118 100	

После сортировки

Name	Price	Number	Cost	Discount
bread	1540	3	4620	
bread	1450	5	7250	
bread	1550	1	1530	20
butter	3700	1	3700	
butter	3410	1	3400	10
choco	7 200	2	14 000	200
meat	18 000	2	34 400	800
meat	21 250	2	42 400	50
potato	1 750	2	3 400	50
potato	1 800	2	3 400	100
Total cost			118 100	

Результаты поиска:

Элемент

choco;14000;1

не найден

Элемент

choco;7300;2;300

найден в позиции 5

Замечания к задаче 1

– В методе createPurchase( ) при обнаружении ошибки в аргументе csvString выбрасывать **собственное** обрабатываемое исключение CsvLineException, которое должно содержать причину ошибки во входной строке. Данное исключение необходимо обработать в конструкторе класса вектора со строковым аргументом, выведя на консоль саму *строку* и *причину* ошибки.

Схематически метод createPurchase() будет выглядеть:

```
private Purchase createPurchase(string csvString)
{
    …
```

```
        if (someProblem) throw new CsvLineException(csvString, … );
        …
        if (otherProblem) throw new CsvLineException(csvString, … );
        …
}
```

Дополнительным аргументом конструктора исключения не обязательно будет строка. Допускаются любые типы в любом количестве, а также наследование исключений.

В конструкторе вектора(коллекции) должна быть обработка исключения примерно так:

```
public PurchaseCollection(string inFile)
{
      try
      {
        StreamReader sr = new StreamReader(inFile + ".csv");
        string s;
        while ((s = sr.ReadLine()) != null)
        {
           try
           {
             listPurchases.Add(createPurchase(s));
           }
           catch (CsvLineException e)
           {
             Console.WriteLine("Ошибка " + e.Message);
           }
        }
      }
      catch (FileNotFoundException)
      {
        Console.WriteLine("Файл " + inFile + " не найден.");
      }
}
```

– В классы покупок можно добавлять методы.

– Принадлежность к классу определять методом GetType(), операторами is и typeof().

– Для вывода покупок в табличном виде удобно использовать вывод с форматированием (string.Format).

– Не злоупотребляйте блоками try-catch. Если метод элементарно реализуется через if без блока try-catch, то и надо делать через if. Например, методы `Insert`( ) и `Delete`( ).

– Magic numbers – это огромный минус. Заменяйте константами или перечислениями.

Task 2

Файл in.txt содержит корректные вещественные координаты отрезков **на плоскости** (по одному отрезку в строке) в формате:
$(x_1;y_1)$ $(x_2;y_2)$

1 Создать список вида (len; num) где len – длина отрезка, округлённая до целого значения, а num – количество отрезков длины len.

2) Отсортировать список по убыванию num.

Пример файла in.txt:

```
(0 ; 0) ( 2,5; 0)
( 0;1) ( 0; 2 )
(-3,0; 20,1e-1) (-2;2)
```

Вывод:

```
1;2
3;1
```

Замечания к задаче 2

– Информационный класс, как обычно, описать в отдельном файле. Считывание файла данных и необходимые действия реализовать в методе Main() приложения.

– Строку разобрать методом split( ) через регулярное выражение. Формат строки (неформально):
\s*(\s*вещ.число\s*;\s*вещ.число\s*)\s*(\s*вещ.число\s*;\s*вещ.число\s*)\s*
Вещественное число во входной строке **правильное**. Пары скобок и точка с запятой между числами гарантированно имеются. Учитывать данные условия при составлении регулярного выражения. Не нужно подбирать выражение под вещественное число. Оно далеко не тривиальное.

– Для поиска элемента в коллекции использовать метод BinarySearch( ) коллекции. Цикл с линейным поиском элемента **запрещается**, т.к. очень неэффективный.

– Для корректной работы BinarySearch() нужно, чтобы список всегда был отсортирован. Возвращаемое значение в случае неудачного поиска может быть использовано для получения индекса, куда нужно вставить новый элемент с сохранением упорядоченности массива. Об этом подробнее есть в ссылке в задании.

– BynarySearch() использует IComparable< >, а Sort() – IComparer. Это позволяет в одном классе иметь два метода сравнения: один по длине отрезка (нужен для поиска BynarySearch()), а второй – по количеству (для сортировки перед выводом).

– Длина отрезка = $\text{sqrt}((x_1 - x_2)^2 + (y_1 - y_2)^2)$ – это на всякий случай ☺;

– чтобы половинные значения округлялись к большему, нужно вызывать Math.Round(…,MidpointRounding.AwayFromZero).