Task 1

Описать класс, хранящий информацию о командировочных расходах работников.

Поля:

- суточные в бел. рублях (константа),
- ФИ работника,
- транспортные расходы в бел. рублях,
- количество дней.

Конструкторы:

- по умолчанию;
- с параметрами.

Методы:

- getters/setters;
- GetTotal( ) – расчёт общей величины расходов (=транспортные + кол-во дней * суточные);
- Show( ) – вывод всех полей на консоль по одному в строке в формате name=value;
  Пример:
  rate = 25000
  account = Anton Slutsky
  transport = 50000
  days = 5
  total = 175000
- ToString( ) – переопределить метод ToString() для вывода информации в csv формате (все поля в одну строку через точку с запятой).
  Пример:
  25000;Anton Slutsky;50000;5;175000

В методе Main(…) консольного приложения сделать следующее:

1 Создать массив из пяти объектов, элемент с индексом 2 должен быть пустым, последний элемент должен быть создан с использованием конструктора по умолчанию, остальные – с использованием конструктора с параметрами.

2 Вывести все объекты массива на консоль с использованием метода Show().

3 Изменить транспортные расходы в последнем объекте в массиве.

4 Вывести общую продолжительность двух первых командировок.

Пример:
Duration = 9

5 Вывести массив на консоль с использованием метода ToString( ).

Task 2

Разработать класс, описывающий некоторый физический материал(вещество).

Поля:

- имя,
- плотность.

Конструкторы:

- по умолчанию;
- с параметрами.

Методы:

- getters/setters;
- ToString( ) – значения полей объекта в csv формат.
  Пример:
  steel;7850

Разработать класс, описывающий однородный предмет, состоящий из одного материала.

Поля:

- имя,
- материал (класс, разработанный выше),
- объём.

Конструкторы:

- по умолчанию;
- с параметрами.

Методы:

- getters/setters;
- GetMass( ) – вычисление массы предмета (= плотность * объём);
- ToString( ) – объект в строку в csv формате.
  Пример:
  wire;steel;7850;0.03;235.5

В методе Main(…) консольного приложения сделать следующее:

1 Создать объект Стальной_Провод из стали в объёме 0.03м^3.

2 Вывести объект на консоль с использованием ToString( ).

3 Изменить материал провода на медь (плотность = 8500) и вывести на консоль его новую массу.

Task 3

Разработать класс Purchase, описывающий покупку некоторого товара по некоторой цене, класс должен реализовать интерфейс IComparable.

Поля:

- название товара,
- цена в бел. рублях,
- кол-во единиц товара,
- день недели (создать перечисление (enum).

Конструкторы:

- по умолчанию,
- с параметрами.

Методы:

- getters/setters;
- GetCost( ) – стоимость покупки;
- ToString( ) – перевод объекта в csv формат в строку;
- CompareTo(Object purchase) – сравнение покупок по *стоимости покупки* (возвращает int <0, =0, >0 если исходный объект меньше, равен или больше того, с которым сравнивают (параметр метода).

В методе Main(…) консольного приложения сделать следующее:

1 Создать массив с пятью непустыми объектами.

2 Вывести объекты из массива на консоль с использованием ToString( ).

3 Вычислить среднюю стоимость всех покупок, найти день, в который была осуществлена покупка с максимальной стоимостью.

4 Отсортировать массив с использованием метода Sort( ) класса Array.

5 Вывести объекты из массива на консоль с использованием ToString( ).

Замечания

– При создании объекта в конструктор передавать константы (т.е. не нужно вводить с клавиатуры или из файла поля класса):

BusinessTrip anton = new BusinessTrip("Anton Slutsky", 50000, 5);

– Если поле является константой класса, то оно должно иметь спецификаторы:

```
public const
```

Пример:

```
class Purchase
{
  public const int MaxPrice = 45;
 …
}
```

– Если ссылка на объект встречается в контексте строки, то по умолчанию вызывается у объекта вызывается метод ToString( ).

Пример. Такой код:

```
Console.WriteLine("Expenses of anton >"+anton);
```

предпочтительнее, чем эквивалентный ему:

```
Console.WriteLine("Expenses of anton >"+anton.ToString());
```

– В задаче 2 **нет** отношения **наследования** между сущностями (материал и предмет). Наследование возникает, когда одна сущность является частным случаем другой. Например, жидкость (или твердое вещество) и материал. Другими словами, жидкость (как и твердое вещество) является материалом. Предмет **состоит** из материала, а **не является** материалом.

– В задаче 3 структура класса Purchase:

```
class Purchase : IComparable
{
      public int CompareTo(Object o)
      {
            Purchase p = (Purchase)o;
            if (price < p.price)
            {
                  return -1;
            }
            //и так далее
            //можно возвращать любое отрицательное число,
            //так что можно обойтись без if
}
```

При реализации механизма сравнения объектов класса через метод compareTo( ) интерфейса Comparable массив объектов этого класса сортируется следующим образом:

```
//объявление и инициализация массива
Purchase[] purchases = new Purchase[] {
```

```
        new Purchase(…),
        …
    };
    //сортировка
    Arrays.sort(purchases);
```

Еще более эффективно использовать параметризованный интерфейс IComparable:

```
    class Purchase : IComparable<Purchase>
    {
    //тип аргумента Purchase, а не Object
    //избавляемся от преобразования.
    //а значит ошибка, если и выскочит, то на этапе компиляции, а не в рантайме!
        public int CompareTo(Purchase p)
        {

        }
```

– В задаче 3 для дня недели создать перечисление.

– В задаче 3 вычисление средней стоимости покупок и поиск дня макс. покупки сделать циклом(ами), без использования LINQ.

– При использовании ООП сущности и их атрибуты именуют не абы как, а осмысленно. Часто это помогает правильно реализовать задачу.
Например, в задаче 2 типичная ошибка – замена не материала целиком, а по отдельности замена имени и плотности.
Создадим для сравнения два экземпляра класса Material, оба соответствующих стали. Первый назовем абстрактно m, а второй осмысленно steel. Следующей операцией заменим имя на медь.

```
    Material m = new Material("steel", 7850);
    Material steel = new Material("steel", 7850);
    m.setName("copper");
    steel.setName("copper");
    //аналогично плотность
```

В первом случае как бы все и гладко. А во втором случае выглядит формально правильно, но по своей сути абсурдно. Правильный выбор имени материала steel наводит на мысль, что нужно создать еще один экземпляр класса Material для меди и заменить у предмета один материал на другой.
Вывод: абстрактное именование – скрытый источник проблем реализации проекта.
Логично, что после создания материала, его плотность и название не меняются, для того, чтобы запретить возможность изменения значений полей класса кроме как в конструкторе при создании используется модификатор readonly.
Например:

```
public readonly int MaxPrice;
```