Code Exercise

Create the package by.epam.lab and define the class Purchase inside which represents a wholesale purchase of the **same** product of the **same** price during a week. The class Purchase must implement the interface Comparable parameterized by this class.

Class fields:

- product name,
- **price** (in BYN),
- **number** of purchased units,
- discount **percent**,
- week day (create **enumeration** WeekDay).

Constructors:

- no-arg constructor,
- parameterized constructor.

Methods:

- getters/setters;
- getCost( ) – returns the purchase cost:
  price * number * (100 – percent) / 100, **rounded to 1.00 BYN**;
- toString( ) – returns a string representation of a purchase in the csv–format: each **non constant** field and the purchase cost, separated by the ";" symbol);
- compareTo(Purchase purchase) – compares **numbers** of purchased units of purchases and returns a negative integer, zero, or a positive integer as this purchase is less than, equal to, or greater than the specified purchase.

File src\in.txt consists of 11 lines **with correct data**.

The first line contains the number PURCHASES_NUMBER from 0 to 10. Next 10 lines contain information about 10 valid purchases. Values within a line are separated by spaces. The value for a weekday is given by numbers from 0 to 6, where 0 is Sunday, … , 6 is Saturday.

Define the Runner class in the **default** package, where:

1. Create an array for PURCHASES_NUMBER purchases.
2. Initialize this array by the file data.
3. Output the array content to the console in the following format:

```
class constants
purchase[0]
…
purchase[PURCHASES_NUMBER - 1]
```

4. Calculate the mean cost of all purchases (3 digits after the point), the total cost of all purchases on Monday, the day with the maximum purchase cost. Output them to the console.

5. Sort the array by the field number in the ascending order by the method sort( ) of the class Arrays.

6. Output the array content to the console in the format above.

7. Find some purchase with number equalled to 5 with the method binarySearch( ) of the class Arrays and output it.

Замечания и ограничения

– Напоминаю, в данной задаче также запрещаю создавать класс для финансовой величины (как и использовать функционал класса BigDecimal).
Рекомендую еще раз прочитать первое замечание к задаче 1 о недопустимости обработки финансовых величин с помощью вещественных типов.
Повторяю, что формат вывода финансовой величины d+.dd, т.е. рубли и копейки, разделенные точкой (без всяких руб, коп и т.п.), где рубли – целое неотрицательное число, копейки – две десятичные цифры.

– Наличие статического метода в классе - это отход от чистого ООП.
Имеет смысл собирать статические методы в утилитном классе, т.е. классе, который содержит только статические методы.
Следствие: **публичный** статический метод в **неутилитном** классе - антипаттерн.
Следствие 2: если статический метод используется в единственном классе, то его, как правило, лучше оставить в этом классе и объявить приватным, т.е. не надо выносить в **утилитный** класс. Гарантированно не надо выносить в отдельный утилитный класс методы, используемые в раннере, который сам является утилитным классом.

– Пункт 2 раннера выполнить в отдельном цикле.

– Пункты 3 и 6: обратите внимание на формат вывода массива. Я не понимаю почему, но статистика утверждает, что в 25+% решений игнорируется требование вывода констант класса до вывода элементов массива.

– Пункт 4 выполнить в одном цикле (без счетчика!).

– Класс Purchase должен реализовывать интерфейс Comparable, параметризованный объектами этого же класса. Следовательно, структура класса Purchase такая:

```
public class Purchase implements Comparable<Purchase> {
    …
    public int compareTo(Purchase purchase) {
      if(number < purchase.number) {
        return –1;
      }
      //и так далее.
      //Можно возвращать ЛЮБОЕ отрицательное число.
```

```
            //Подумайте, как обойтись без операторов if.
        }
}
```

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

```
//объявление и инициализация массива
final int PURCHASES_NUMBER = sc.nextInt();
Purchase[ ] purchases = new Purchase[PURCHASES_NUMBER];
//цикл инициализации массива из файла
…
//пункты 3, 4
...
//сортировка
Arrays.sort(purchases);
```

– Для дня недели создать перечисление. Причем не нужно его прятать внутрь класса покупки. Оно должно быть внешним, как следствие, в отдельном файле. Элементы перечисления – константы, что требует соответствующего именования.

– Иногда ошибочно интерпретируют фразу пункта 4: «Calculate …, the day with the maximum purchase cost». Т.е. речь идет о дне, когда была зарегистрирована покупка (в единственном числе) с максимальной стоимостью. А вот если бы покупки были во множественном числе (the day with the maximum of purchases costs), тогда следовало бы найти суммарные стоимости за каждый день, а потом среди них максимум.

– Начальные значения константам класса задать в самом классе, остальные данные ввести из файла.

– Читать файл с помощью функционала класса Scanner. См. пример ниже, а также И. Блинов стр. 242. Установить программно англоязычную локаль. По умолчанию она использует точку в качестве разделителя целой и дробной частей вещественного числа (в кирилличных локалях – запятая).

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;
public class ScannerSample {
        public static void main (String[] args) {
                try(Scanner sc = new Scanner(new FileReader("src/in.txt"))) {
                        sc.useLocale(Locale.ENGLISH);           //установка локали
                        String str = sc.next();                 //чтение строки,
                        int n = sc.nextInt();                   //целого,
                        double x = sc.nextDouble();             //вещественного
```

```
                System.out.println(str + ";" + n + ";" + x);
            } catch (FileNotFoundException e) {
                System.err.println("Input file is not found");
            }
        }
}
```

– По условию файл корректный. Не надо проверять вводимые значения. Если встретится любая ошибка ввода, то приложение должно **завершить работу с рантаймовым исключением**. Например, если задать в первой строке значение -1 (или 11, или строку), в следующих строках для дня недели 7 или строку и т.д.

– Обязательно проверить работу раннера на крайних значениях, т.е. когда в первой строке файла находится значение 0 или 10.
Если в первой строке файла находится значение 0, то вывод на консоль результатов пунктов 4 и 7 следующий:

```
Mean cost = 0.000
The total cost on Monday = 0.00
The day with the maximum cost purchase is null
Required purchase is not found
```

– В джаве, в отличие от C++, можно создавать массивы длиной 0, что часто практикуется. См. детали у Дж. Блоха “Эффективное программирование”, статья 27.

– Исходный файл корректный. Однако он может отсутствовать. И этот случай приложение должно обрабатывать корректно.
Правильно расположите блок `catch (FileNotFoundException e)` и обязательно протестируйте работу приложения на отсутствующем файле.

– rounded to 1.00 BYN, т.е. округленная до 1 рубля (это для тех, кто в танке :)