Code Exercise

Create the package named by.epam.lab for entities classes.

Define the class called Byn that represents BYN.

Define the class called Product that represents a product.

Class fields:

- name,
- price in BYN.

Constructors:

- no-arg constructor,
- parameterized constructor.

Methods:

- getters/setters;
- toString( ) – converting of an object to a string in the csv–format: each field, separated by the ";" symbol.

Define the **abstract** superclass AbstractPurchase, describing the product purchase and implementing interface Comparable<AbstractPurchase>.

Superclass fields:

- product,
- number of purchased units.

Constructors:

- no-arg constructor,
- parameterized constructor.

Methods:

- getters/setters;
- getCost( ) – returns a purchase cost in BYN rounded down 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(AbstractPurchase purchase) – compares purchases for sorting by the cost decreasing.

Define the first subclass for the purchase with a price discount for every unit of the product purchased and override necessary methods.

Define the second subclass for the purchase with a discount to be presented if the number of purchased units is greater than the given subclass constant. A discount rate is given by the percent from the purchase cost. Override necessary methods.

Define the third subclass for the purchase with an addition for transport expenses and override necessary methods. Transport expenses don't depend on the number of purchased units.

Define the Runner class in the default package, where:

1. Create a unique product for purchasing.
2. Create an array for 6 objects (2 instances of every subclass by general-purpose constructors).
3. Print the array content to the console (one element per line).

4. Sort an array by the cost in descending order with the method sort( ) of the class Arrays.

5. Print the array content to the console (one element per line).

6. Print the minimum cost of purchase.

Example:

Minimum cost = 2.00

7. Create the method int search(Purchase[] purchases), where find an index of some purchase with cost equal to 5.00 BYN using the method binarySearch( ) of the class Arrays. Print the found purchase or a message about the failed search.

Define the TestRunner class in the **default** package for unit tests.

Замечания и ограничения

– Обратите внимание, что стоимость во всех классах должна быть округлена снизу (т.е. до ближайшего меньшего) до 1 рубля.
Пример. Стоимости из полуоткрытого диапазона [7.00; 8.00) округляются до 7.00 рублей.

– Проверьте, присутствует ли копи-паст при расчете стоимости. Если да, то попробуйте избавиться от него.
Я вижу два способа (какие?; не спешите изменять цвет текста следующих абзацев, попробуйте разобраться без подсказок):
---
1. определить метод getCost( ) в суперклассе, переопределить его в подклассах, а также определить статический метод округления в суперклассе (абстрактный метод в этом способе не нужен);
2. объявить вспомогательный абстрактный метод в суперклассе, переопределить его в подклассах, а метод getCost( ) достаточно определить только в суперклассе.

Конечно, второй способ предпочтительнее, так как первый это вообще-то антипаттерн.

– По условию нужно создать единственный товар для массива покупок (unique product). Отсюда следует, что если после создания покупки можно изменить атрибуты покупки, связанные с товаром, то условие единственности товара не соблюдается.
Можно ли обеспечить условие единственности товара, если использовать мутабельный класс Byn из предыдущей задачи?

Ответ: нет.
Класс Byn должен быть иммутабельным.
Для надежности добавьте модификатор final к полю value.

– Вспомните о наиболее рациональном способе инициализации массива. Нужно избегать явного задания количества элементов в массиве.

– Пункты 3 и 5 совпадают. Следовательно, реализацию нужно выносить в статический метод. Не забывайте, что в раннере тип доступа public только у метода main( ). Все остальные private!

– Пункт 7, задача – правильно задать второй аргумент метода поиска и использовать встроенный компаратор класса AbstractPurchase. Новый массив из стоимостей покупок создавать запрещено.

– Как и в задаче classes3, для полного тестирования задания с поиском элемента необходимо создать “много” наборов исходных данных. В частности, искомый элемент находится на границах массива, меньше минимума или больше максимума стоимости по массиву.
Причем для метода search() раннера нужно также создать юнит тесты.