Создайте пакет с именем by.gsu.epamlab и определите класс BusinessTrip, который представляет собой командировку сотрудника.

Поля класса:

-дневной нормы в BYN (константа),

-счет сотрудника,

-транспортные расходы в BYN,

-количество дней.

Конструкторы:

конструктор без параметров;

параметризованный конструктор.

Методы:

-геттеры / Сеттеры;

-getTotal () - возвращает общую сумму командировочных расходов в BYN

(= транспортные расходы + суточная ставка \* количество дней);

-show () - выводит все поля на консоль (каждое поле и общая стоимость командировки должны быть в отдельных строках в следующем формате: name = value);

Пример:

ставка = 7,00

счет = Антон Шумский

транспорт = 16,20

дней = 5

всего = 51,20

-toString () - возвращает строковое представление деловой поездки в формате csv: каждое не константное поле и общие расходы на командировку, разделенные «;»; символ.

Пример:

Антон Шумский, 16:20, 5, 51.20

Определите класс Runner в пакете по умолчанию, где:

1. Создайте массив из минимум 5 объектов (элемент с индексом 2 должен быть пустым, последний элемент массива должен быть создан конструктором no-arg, другие элементы действительны и должны быть созданы параметризованным конструктором).

2. Выведите содержимое массива на консоль, используя метод show ().

3. Обновите транспортные расходы сотрудника для последнего объекта массива.

4. Выведите общую продолжительность двух начальных командировок одним оператором.

Пример:

Продолжительность = 9

5. Вывести содержимое массива на консоль (по одному элементу в строке), используя метод toString () неявно.

Create the package named by.gsu.epamlab and define a class called BusinessTrip that represents a business trip of an employee.

**Class fields**:

* daily allowance rate in BYN (the constant),
* employee`s account,
* transportation expenses in BYN,
* number of days.

**Constructors**:

* no-arg constructor;
* parameterized constructor.

**Methods**:

* getters/setters;
* getTotal( ) – returns the total business trip expenses in BYN  
  (= transportation expenses + daily allowance rate \* number of days);
* show( ) – outputs all fields to the console (each field and the total business trip expenses should be in separate lines in the following format: name=value);

Example:

rate = 7.00

account = Anton Shumsky

transport = 16.20

days = 5

total = 51.20

* toString( ) – returns a string representation of a business trip in the csv–format: each **non constant** field and the total business trip expenses, separated by the ";" symbol.

Example:

Anton Shumsky;16.20;5;51.20

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

1. Create an array of **minimum** 5 objects (the element with the index 2 should be empty; the last element of the array should be created by the no-arg constructor; **other elements are valid** and should be created by the parameterized constructor).

2. Output the array content to the console, using show( ) method.

3. Update the employee`s transportation expenses for the last object of the array.

4. Output the total duration of two initial business trips **by the single operator**.

Example:

Duration = 9

5. Output the array content to the console (one element per line), using toString( ) method implicitly.

**Замечания и ограничения**

– Обратите внимание, в условиях задач денежные величины указаны в белорусских рублях. **Вещественный тип не подходит для финансовых расчетов**. Читайте у Джошуа Блоха раздел "Если требуются точные ответы, избегайте использования типов float и double".

В данной и следующих задачах на финансовые величины ввожу следующее ограничение: **максимальное значение стоимости в копейках не больше 0x7fffffff** (т.е. максимум диапазона int).

Отсюда следствие: все финансовые величины хранить и обрабатывать в копейках в диапазоне int, а отображать в рублях с двумя знаками после десятичной точки.

Например. Пусть цена товара равна 1 рубль 35 копеек, количество равно 4. Тогда в ОП переменные цена, количество и стоимость будут иметь значения 135, 4 и 540 соответственно, а при выводе – 1.35, 4 и 5.40.

– В данной задаче запрещаю создавать класс для финансовой величины.

– Под пустым элементом массива понимается равный null элемент.

– Создайте массив так, чтобы количество элементов (к примеру, число 5) явно не указывалось.

– Текст раннера, который находится ниже создания массива (пункты 2-5), **не должен зависеть** от количества элементов в массиве. Среди дополнительных элементов также могут быть пустые. Обращаю внимание, что согласно условию последний элемент всегда создается дефолтным конструктором.

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

BusinessTrip anton = new BusinessTrip("Anton Shumsky", 700, 5);

– Поля **экземпляра класса** должны иметь **приватный** уровень доступа.

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

final static, а также спецификатор уровня доступа.

   Пример:

public class Salary {

private **final static** int WORKING\_HOURS = 8;

…

}

Для **любой** константы класса геттер создавать не надо.

Если у нее уровень доступа публичный, то она видна за пределами класса.

public class Salary {

**public** final static int WORKING\_HOURS = 8;

~~public int getWORKING\_HOURS( ) {~~

~~return WORKING\_HOURS;~~

~~}~~

…

}

Если константа нужна только внутри класса, то для нее следует установить приватный уровень доступа (и геттер не имеет смысла).

public class Salary {

private final static int WORKING\_HOURS = 8;

…

}

– Если геттеры **тривиальные**, то внутри класса используйте прямое обращение к полю.

public int getMark1() {

return mark1;

}

public int getMark2() {

return mark2;

}

private int result() {

return **mark1** + **mark2**; //а не getMark1( ) + getMark2( )

}

[Здесь изложена точка зрения](http://www.ibm.com/developerworks/ru/library/j-noaccent/#N102C5) известного профессора, см. пункт *Ограничение доступа и области видимости* (ориентируйтесь на листинг 12).

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

   Пример. Такой код:

System.out.println("Business trip expenses of anton > " + anton);

   предпочтительнее, чем эквивалентный ему:

System.out.println("Business trip expenses of anton > " + anton.toString());

– На каждом предприятии Java code conventions обязательны для исполнения (см. пост на сайте). Внутри ЕПАМа дополнительно используют рекомендации по кодированию (см. документ [EPAM-Code-Instructions](https://docs.google.com/document/d/19b6TteRdFidvX2YzPC9y9EGIAx25miZUd4tcDobKk5I/view) в папке [texts](https://drive.google.com/drive/folders/0B9YPS_nAWjBRbVA0YjlORFh6MkU)).

– Общепринятая практика – именовать массив как класс, но во множественном числе. Элемент – итератор массива отличается от имени класса только начальной буквой (у класса заглавная, у итератора строчная).

   Пример:

**Trial**[ ] **t**rial**s** = {  
 ...  
};  
for (**T**rial **t**rial : **t**rial**s**) {  
 ...  
}

– Не изменяйте без необходимости конструкторы, геттеры/сеттеры, созданные Эклипсом.

– Ни один из циклов не требует формы со счетчиком. Используйте форму цикла for–each.

– Пункт 4: by the single operator – это значит “одним оператором”. Следовательно, не должно быть никаких ифов, циклов, вызовов собственных методов.

– Ни в одной задаче данной темы **нет необходимости в блоке трай-кэтч**.

Вообще есть жесткая рекомендация, согласно которой, если можно тривиально обойтись ифом, то не надо применять блок трай-кэтч.

См. детали у Дж. Блоха "Эффективное программирование", статья 39.

– ВСЕ тексты (исходные данные, вывод, комментарии) должны быть исключительно на инглише!

– Выложить решение в архиве **classes1.jar** в своей папке jars (см. пост Замечания по созданию архива с решением).

Но прежде, чем выложить архив в папку, на всякий случай откройте его и убедитесь, что в нем находятся только исходники. В крайнем случае, убирайте мусор руками из архива, если не получается Эклипсом.   
Иначе отгребете от меня массу негатива :(