Training materials

- The Java Tutorials. Trail: Getting Started.
- The Java Tutorials. Lesson: Classes and Objects.
- **Course on learn.epam.com Java.Fundamentals.**
- **Course on learn.epam.com Java.Classes.**
- И. Блинов. Глава 3: стр. 54-63, 76-79.
- Методический материал для начинающих по разработке проекта в среде Eclipse (см. файл classes-eclipse.doc) и исходники к нему classes-eclipse.jar.
- Сайт skipy.ru. Объектная ориентация.
- Сайт skipy.ru. Четыре важных "p".

Code Exercise

Create the package by.epam.lab and define the class BusinessTrip inside which 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, and the business trip with maximum cost.

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 оперативной памяти переменные цена, количество и стоимость будут иметь значения 135, 4 и 540 соответственно, а при выводе – 1.35, 4 и 5.40.
Следствие из следствия: **преобразование в строку выполнять с помощью целочисленных арифметических операций**, а не через очевидные вещественные операции, т.е. anyFinancialValue / 100.0 или anyFinancialValue * 0.01, что может привести к неправильному результату в силу особенностей реализации вещественной арифметики (кому интересны подробности, повторяю, ищите у Дж. Блоха). Даже если результат правильный, то все равно это антипаттерн.
Наконец, вот еще один убедительный аргумент - ссылка на статью Representing money от авторов отличного ресурса Java practices.

– На всякий случай отдельно повторяю формат вывода финансовой величины, установленный в предыдущем замечании: рубли и копейки, разделенные точкой (без всяких руб, коп и т.п.), где рубли – целое неотрицательное число, копейки – две десятичные цифры. Примеры: 1.35, 5.40, 0.05, 3.05, 100.00.
Тщательно протестируйте преобразование в формат финансовой величины.

– Копи-паст – это грех программиста, за который я выписываю желтую карточку (

– Во всех задачах (как этой темы, так и других) запрещаю использовать функционал класса BigDecimal.

– Во всех задачах до темы Tasks Strings запрещаю использовать функционал класса StringBuilder (и тем более StringBuffer).

– Во всех задачах этой темы 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 (5, 6 и более) . Обращаю внимание, что согласно условию последний элемент всегда создается конструктором без аргументов.

– При создании объекта в конструктор передавать константы (т.е. значения полей класса не нужно вводить с клавиатуры или из файла):
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 static final 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( )
        }
```

– 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());
```

– 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[ ] trials = {
        ...
};
for (Trial trial : trials) {
        ...
}
```

– 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 classes нет необходимости в **блоке try-catch** для **обработки данных** (не путать с получением доступа к ресурсам, например, создание экземпляра Scanner, которое будет в задаче classes3).
Вообще есть жесткая рекомендация, согласно которой, если можно тривиально обойтись if оператором, то не надо применять блок try-catch.
См. детали у Дж. Блоха "Эффективное программирование", статья 39.

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