Training materials

- The Java Tutorials. Lesson: Interfaces and Inheritance.
- **Course on learn.epam.com Java.Classes.**
- И. Блинов. Глава 4: стр. 97-114.
- Сайт skipy.ru. Наследование как явление.
- Сайт skipy.ru. Overloading, overriding и другие звери.

Code Exercise

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

Define the **mutable** class called Byn that represents a financial entity for BYN.

Class field:

- value of a financial entity in kopecks.

Constructors:

- no-arg constructor,
- parameterized constructors.

Methods:

- operations with financial entity;
- toString( ) – returns a string representation of a financial entity in the format d+.dd;
- equals(Object obj) – compares financial entities and returns true if their kopecks are equal, false otherwise.
- compareTo(Byn byn) – compares kopecks and returns a negative integer, zero, or a positive integer as this byn is less than, equal to, or greater than the specified byn.

Define the superclass called Purchase that represents a product purchase.

Superclass fields:

- product name,
- price in BYN,
- number of purchased units.

Constructors:

- no-arg constructor,
- parameterized constructor,
- constructor of reading from an instance of the class Scanner.

Methods:

- getters/setters;
- getCost( ) – returns the purchase cost in 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;
- equals(Object obj) – compares names and prices of purchases and returns true if these fields are equal, false otherwise.

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 percentage from the purchase cost. Override necessary methods.

File src\in.txt consists of 6 lines **with correct data**. Every line contains needed data separated by spaces for 1 object of the superclass or the first subclass or the second one. Every line begins with some identifier of the purchase class, then other data follow.

The line example for the superclass object:

GENERAL_PURCHASE Milk 140 3

Define the Runner class in the default package, where:

1. Create an array for 6 objects.
2. Input data from the given file into the array.
3. Output the array content to the console (one element per line).
4. Output the purchase with maximum cost.
5. Determine whether all purchases are equal.

Implement subtasks 2–5 with **the single cycle**.

Define the TestRunner class in the **default** package, where create unit tests for the following functionality:

1. class Byn;
2. creation of purchase entities;
3. methods getCost( ) and equals( ) of purchase entities

Замечания и ограничения

– В данной задаче необходимо создать **мутабельный** класс для финансовой величины и использовать его функционал в классах покупок и раннере. Мутабельность предполагает, что методы класса могут изменять состояние экземпляра класса.

– Строковое представление финансовой величины d+.dd, т.е. то же самое, как в задачах classes – руб.коп, где копейки - всегда 2 цифры.

– Очень плохое решение - создавать геттер и сеттер в классе Byn.
Почему? Рекомендую обсудить данный вопрос.
Итак, ввожу ограничение!!!
Не создавать геттер и сеттер в классе Byn.

– Подкласс в своем имени содержит имя (по крайней мере ключевое слово) суперкласса (или интерфейса). Как правило, ключевое слово последнее. Примеры из jdk:
List, AbstractList, ArrayList, LinkedList.

– Класс Purchase предполагается использовать в раннере. Значит, все его поля должны иметь приватный уровень доступа.

– Методы подкласса не должны изменять значения полей суперкласса. Можно объявлять новые поля в подклассах и переопределять методы!

– Примеры расчета стоимости в подклассах (финансовые величины представлены в копейках):

1) пусть скидка в цене за каждую единицу товара 50, цена 300, количество 2; тогда стоимость = (300 – 50) * 2 = 500.

2) пусть процент скидки 5.825, цена 500, количество 20, количество, которое надо превысить 15.
т.к. 20 > 15, то стоимость = 500 * 20 * (1 – 5.825/100) = 9418 (стоимость по-прежнему целая!).

– В toString( ) подклассов сначала добавить поля суперкласса, затем неконстантные поля подкласса, а затем стоимость покупки. Чтобы избежать копи-паста, создать в каждом классе protected-метод fieldsToString( ).

– Учтите, что в правильной csv-строке символ-разделитель служит маркером наличия следующего элемента.
Рассмотрим примеры.
Пусть дана csv-строка "aaa;;200". Тогда в ней нулевой элемент равен "aaa", первый - "" (пустая строка), второй - "200".
Пусть дана csv-строка "aaa;200;". Тогда в ней нулевой элемент равен "aaa", первый - "200", второй - "" (пустая строка).
Следите за правильностью вывода элементов массива в формате csv. По условию при выводе не должны встречаться два подряд символа ";" или этот символ в конце выводимой строки.

– Применять конструкцию super в двух случаях:
1. для вызова конструктора суперкласса,
2. в переопределенном методе подкласса для вызова **одноименного** метода суперкласса, чтобы избежать бесконечной рекурсии.

– Так как строки являются не примитивными данными, а объектами класса String, то их содержимое сравнивается через метод equals(Object obj). **Аналогично сравниваются объекты других классов.**
Сгенерируйте код этого метода инструментами IDE.
Блестящий методический материал о сравнении объектов выложен по ссылке http://skipy.ru/technics/objCompTh.html. Его нужно изучить при наличии свободного времени, а не в контексте данной темы.
Из него следует, в частности, что в этой задаче можно не переопределять метод hashCode( ).

– Читать файл с помощью функционала класса Scanner.

– Для создания объектов применить шаблон Factory (И. Блинов, стр. 606–611, а также стр. 110–112, но хуже). Схема фабричного класса:

```
public class PurchasesFactory {
```

```
        private static enum PurchaseKind {
                GENERAL_PURCHASE, …
        }
        public static Purchase getPurchaseFromFactory(Scanner sc) {
                String id = sc.next();
                PurchaseKind kind = PurchaseKind.valueOf(id);
                switch(kind) {
                        case GENERAL_PURCHASE :
                                return new Purchase(sc);
                        case … :
                        …
                        default :
                                throw new IllegalArgumentException();
                }
        }
}
```

Можно еще более улучшить реализацию, вообще избавившись от оператора switch. Нужно использовать следующее свойство перечисления: каждая константа - это константный вложенный класс. Данный способ будет изложен на этапе 2.

На этапе 1 можно реализовать через switch.