3.5 Абстрактные классы и интерфейсы

Пришло время попробовать реализовать иерархию классов определенного вида и решить конкретную задачу.

Представим, вы делаете систему фильтрации комментариев на каком-то веб-портале, будь то новости, видео-хостинг, а может даже для системы онлайн-обучения :)

Вы хотите фильтровать комментарии по разным критериям, уметь легко добавлять новые фильтры и модифицировать старые.

Допустим, мы будем фильтровать спам, комментарии с негативным содержанием и слишком длинные комментарии.
Спам будем фильтровать **по наличию указанных ключевых слов в тексте**.
Негативное содержание будем определять по наличию одного из трех смайликов – :(=(:|
Слишком длинные комментарии будем определять исходя из данного числа – **максимальной длины** комментария.

Вы решили абстрагировать фильтр в виде следующего интерфейса:

```
interface TextAnalyzer {
    Label processText(String text);
}
```

Label – тип-перечисление, которые содержит метки, которыми будем помечать текст:

```
enum Label {
    SPAM, NEGATIVE_TEXT, TOO_LONG, OK
}
```

Дальше, **вам необходимо реализовать три класса, которые реализуют данный интерфейс: SpamAnalyzer, NegativeTextAnalyzer и TooLongTextAnalyzer.**

1. **SpamAnalyzer должен конструироваться от массива строк с ключевыми словами. Объект этого класса должен сохранять в своем состоянии этот массив строк в приватном поле keywords.**
2. **NegativeTextAnalyzer должен конструироваться конструктором по-умолчанию.**
3. **TooLongTextAnalyzer должен конструироваться от int'а с максимальной допустимой длиной комментария. Объект этого класса должен сохранять в своем состоянии это число в приватном поле maxLength.**

Наверняка вы уже заметили, что SpamAnalyzer и NegativeTextAnalyzer во многом похожи – они оба проверяют текст на наличие каких-либо ключевых слов (в случае спама мы получаем их из конструктора, в случае негативного текста мы заранее знаем набор грустных смайликов) и в случае нахождения одного из ключевых слов возвращают Label (SPAM и NEGATIVE_TEXT соответственно), а если ничего не нашлось – возвращают OK.
Давайте эту логику **абстрагируем в абстрактный класс KeywordAnalyzer** следующим образом:

1. **Выделим два абстрактных метода getKeywords и getLabel,** один из которых будет возвращать набор ключевых слов, а второй метку, которой необходимо пометить положительные срабатывания. Нам незачем показывать эти методы потребителям классов, поэтому **оставим доступ к ним только для наследников.**
2. **Реализуем processText** таким образом, чтобы он зависел только от getKeywords и getLabel.
3. **Сделаем SpamAnalyzer и NegativeTextAnalyzer наследниками KeywordAnalyzer и реализуем абстрактные методы.**

Последний штрих – написать метод checkLabels, который будет возвращать метку для комментария по набору анализаторов текста. checkLabels должен возвращать первую не-OK метку в порядке данного набора анализаторов, и OK, если все анализаторы вернули OK.
Используйте, пожалуйста, модификатор доступа по-умолчанию для всех классов.
В итоге, реализуйте классы KeywordAnalyzer, SpamAnalyzer, NegativeTextAnalyzer и TooLongTextAnalyzer и метод checkLabels.
TextAnalyzer и Label уже подключены, лишние импорты вам не потребуются.