# Начальная постановка задачи

Программа должна форматировать исходный кода на Java, который задан в файле либо строке. Код не надо компилировать, только правильно расставить фигурные скобки и отступы в блоках. В Java принято делать отступ 4 пробела и открывающую фигурную скобку оставлять на той же строке, где началась конструкция. Например,

while (inputStream.hasNext()) {

char symbol = inputStream.read();

}

Также принято ставить пробелы вокруг знаков операций, перед открывающей круглой скобкой и после нее и т.д.

Уровни сложности алгоритма по возрастанию:

1. Расстановка фигурных скобок и отступов в управляющих конструкциях, перевод строки после ';' (Считаем, что в коде нет комментариев и строковых констант а также цикла for)

2. Расстановка пробелов вокруг знаков арифметических и логических операций и круглых скобок - перед '(' и после ')' (Считаем, что в коде нет комментариев и строковых констант)

3. Правильная обработка строковых констант и комментариев обоих видов

Обязательным является только первый уровень сложности. Если у вас не работает первый уровень, не делайте следующие, добейтесь корректной работы программы!

Хорошо продумайте набор классов, интерфейсов, пакетов. Придерживайтесь принятых в Java правил оформления кода - отступы, JavaDocs и так далее.

Также приветствуется, если вам удастся обойтись в основном методе форматирования без операторов if, switch, new, break, continue. Опять же - это не нужно делать, если не получилось выполнить задание по первому уровню сложности!

Дополнительным плюсом будет создание модульных тестов (unit tests) на метод форматирования.

# 2. Советы и замечания после 1-й проверки

1. Нельзя использовать методы типа replace, регулярные выражения и так далее. Алгоритм должен быть построен так, чтобы каждый символ был прочитан только один раз.
2. Не надо пытаться делать полноценный компилятор, с точки зрения нашего форматтера aa{b;} - это код. Не надо добавлять фигурные скобки в конструкции типа if (a > 0) a++; - такая задача не ставилась. Нужно правильно расставить отступы между уже имеющими фигурными скобками, новые скобки не добавляем.
3. Необходимо выделить классы и интерфейсы, каждый из которых имеет одну понятную ответственность (вспомните SOLID):
   1. IReader - интерфейс для входного потока символов, к нему должно быть 2 реализации: на основе строки и на основе файла, StringReader и FileReader соответственно. Методы как минимум для чтения символа и для проверки, что символы еще есть.
   2. IWriter - интерфейс для выходного потока символов, к нему 2 реализации: StringWriter и FileWriter. Методы как минимум для записи строки.
   3. Formatter - класс, который форматирует с публичным методом void format(IReader reader, IWriter writer), который всю работу и делает. Потоки символов ему передаются открытые и готовые к работе.
   4. Не нужно работать непосредственно со строкой, или списком символов, или StringBuffer - только через абстракции потоков!
4. К интерфейсам и классам нужно написать JavaDocs, хотя бы минимальные. Там не должно быть написано, что автор Вася Пупкин, там должна быть суть интерфейса или метода.
5. Внимательно перечитайте свои ответы про контракт класса, метода. Подумайте, как быть с исключениями, которые могут вылетать в процессе работы потоков и форматтера. Следует ли выбрасывать IOException, например, из потока или надо свой определить?
6. Не используйте в именах всякие символы вроде \_ - это не принято в Java. Ознакомьтесь с соглашениями именования и форматирования кода в Java, их легко найти.
7. Не забудьте, что надо использовать пакеты, а первые два уровня - это перевернутый домен - такое соглашение. Используйте хотя бы com.company.formatter - и только потом внутри уже пакеты по смыслу. Не надо также раскладывать по одному классу в пакет, собирайте их по смыслу вместе. Например, все, что нужно для входного потока.
8. В корень проекта добавьте файл .gitignore, в который добавьте все, что не должно попадать в репозиторий: папки и файлы, которые создаются компилятором и IDE.

# 3. Критерии проверки 3-й попытки

1. Алгоритм работает правильно по 1-му уровню сложности. Без этого пункта все остальные не имеют смысла.
2. В репозитории присутствует файл .gitignore (см. выше)
3. Именование пакетов, классов, интерфейсов, методов, параметров, переменных соответствуют соглашениям Java - с большой буквы только классы, все названия в camelCase.
4. Разбивка на пакеты сделана согласно соглашениям Java (см. выше) и здравого смысла (например, раскладывание классов в пакеты по одному противоречит здравому смыслу).
5. У всех интерфейсов и, где требуется, классов есть JavaDocs, кратко и ясно описывающие предназначение и параметры.
6. (Не обязательно, но желательно) Проект является Maven-проектом, можно откомпилировать и собрать исполняемый модуль из консоли при помощи Maven
7. Есть интерфейсы для потоков символов (см. выше).
8. (Не обязательно, но желательно) Есть реализация потоков символов с помощью файла.
9. Есть реализация потоков символов с помощью строки.
10. Определены и используются собственные классы исключений. Из потоков должны вылетать только исключения, определенные в интерфейсе, все специфичные для реализации исключения должны ловиться внутри и оборачиваться в собственное исключение.
11. (Не обязательно, но желательно) Используется внешний конфигурационный файл .properties, в котором определен символ, используемый для отступа, и количество этих символов на 1 уровень отступа. Для чтения и хранения конфигурации используются специальные классы, передаваемые форматтеру в конструктор.
12. (Не обязательно, но желательно) Есть модульные тесты на метод форматирования, в том числе на крайние случаи вроде пустого потока.
13. (Не обязательно, только для тех, кто сделал все вышеперечисленное) Есть модульные тесты на классы потоков, использующие mock объекты.
14. (Не обязательно, только для тех, кто сделал все вышеперечисленное) Метод форматирования сделан универсальным обработчиком потока символов, не меняющимся ради конкретных правил обработки символов.