Домашние задания

Домашнее задание 1. Hello World

Тесты к домашним заданиям

1. Установите JDK 17+
2. Скопируйте один из вариантов `HelloWorld`, рассмотренных на практике.
3. Откомпилируйте `HelloWorld.java` и получите `HelloWorld.class`.
4. Запустите `HelloWorld` и проверьте его работоспособность.
5. Создайте скрипт, компилирующий и запускающий `HelloWorld` из командной строки.

Домашнее задание 2. Сумма чисел

1. Разработайте класс `Sum`, который при запуске из командной строки будет складывать переданные в качестве аргументов целые числа и выводить их сумму на консоль.
2. Примеры запуска программы:

   ```
   java Sum 1 2 3
   ```
   Результат: 6
   ```
   java Sum 1 2 -3
   ```
   Результат: 0
   ```
   java Sum "1 2 3"
   ```
   Результат: 6
   ```
   java Sum "1 2" " 3"
   ```
   Результат: 6
   ```
   java Sum " "
   ```
   Результат: 0

   Аргументы могут содержать:
   - цифры;
   - знаки + и -;
   - произвольные пробельные символы.
3. При выполнении задания можно считать, что для представления входных данных и промежуточных результатов достаточен тип `int`.
4. Перед выполнением задания ознакомьтесь с документацией к классам String и Integer.
5. Для отладочного вывода используйте System.err, тогда он будет игнорироваться проверяющей программой.

Домашнее задание 3. Реверс

1. Разработайте класс `Reverse`, читающий числа из стандартного ввода, и выводящий их на стандартный вывод в обратном порядке.
2. В каждой строке входа содержится некоторое количество целых чисел (возможно ноль). Числа разделены пробелами. Каждое число помещается в тип `int`.
3. Порядок строк в выходе должен быть обратным по сравнению с порядком строк во входе. Порядок чисел в каждой строке также должен быть обратным к порядку чисел во входе.
4. Вход содержит не более 10^6 чисел и строк.
5. Для чтения чисел используйте класс Scanner.
6. Примеры работы программы:

Ввод	Вывод
1 2 3	3 2 1
3 2 1	1 2 3
1 2 -3	-3 2 1

1 2 3 4	4 3 2 1

Домашнее задание 4. Статистика слов

1. Разработайте класс `WordStatInput`, подсчитывающий статистику встречаемости слов во входном файле.
2. Словом называется непрерывная последовательность букв, апострофов (`'`) и дефисов (Unicode category Punctuation, Dash). Для подсчета статистики слова приводятся к нижнему регистру.
3. Выходной файл должен содержать все различные слова, встречающиеся во входном файле, в порядке их появления. Для каждого слова должна быть выведена одна строка, содержащая слово и число его вхождений во входном файле.
4. Имена входного и выходного файла задаются в качестве аргументов командной строки. Кодировка файлов: UTF-8.
5. Примеры работы программы:

Входной файл	Выходной файл

`To be, or not to be, that is the question:`	`to 2`
	`be 2`
	`or 1`
	`not 1`
	`that 1`
	`is 1`
	`the 1`
	`question 1`

`Monday's child is fair of face.`
`Tuesday's child is full of grace.`
	`monday's 1`
	`child 2`
	`is 2`
	`fair 1`
	`of 2`
	`face 1`
	`tuesday's 1`
	`full 1`
	`grace 1`

`Шалтай-Болтай`
`Сидел на стене.`
`Шалтай-Болтай`
`Свалился во сне.`
	`шалтай-болтай 2`
	`сидел 1`
	`на 1`
	`стене 1`
	`свалился 1`
	`во 1`
	`сне 1`

Домашнее задание 5. Свой сканер

1. Реализуйте свой аналог класса Scanner на основе Reader.
2. Примените разработанный `Scanner` для решения задания «Реверс».
3. Примените разработанный `Scanner` для решения задания «Статистика слов».
4. Нужно использовать блочное чтение. Код, управляющий чтением, должен быть общим.
5. *Сложный вариант*. Код, выделяющий числа и слова, должен быть общим.
6. Обратите внимание на:
    - Обработку ошибок.
    - На слова/числа, пересекающие границы блоков, особенно — больше одного раза.

Домашнее задание 6. Статистика слов++

1. Разработайте класс `Wspp`, который будет подсчитывать статистику встречаемости слов во входном файле.
2. Словом называется непрерывная последовательность букв, апострофов и тире (Unicode category Punctuation, Dash). Для подсчета статистики, слова приводятся к нижнему регистру.
3. Выходной файл должен содержать все различные слова, встречающиеся во входном файле, в порядке их появления. Для каждого слова должна быть выведена одна строка, содержащая слово, число его вхождений во входной файл и номера вхождений этого слова среди всех слов во входном файле.
4. Имена входного и выходного файла задаются в качестве аргументов командной строки. Кодировка файлов: UTF-8.
5. Программа должна работать за линейное от размера входного файла время.
6. Для реализации программы используйте Collections Framework.
7. *Сложный вариант.* Реализуйте и примените класс `IntList`, компактно хранящий список целых чисел.
8. Примеры работы программы:

Входной файл	Выходной файл
To be, or not to be, that is the question:	to 2 1 5 be 2 2 6 or 1 3 not 1 4 that 1 7 is 1 8 the 1 9 question 1 10
Monday's child is fair of face. Tuesday's child is full of grace.	monday's 1 1 child 2 2 8 is 2 3 9 fair 1 4 of 2 5 11 face 1 6 tuesday's 1 7 full 1 10 grace 1 12
Шалтай-Болтай Сидел на стене. Шалтай-Болтай Свалился во сне.	шалтай-болтай 2 1 5 сидел 1 2 на 1 3 стене 1 4 свалился 1 6 во 1 7 сне 1 8

Домашнее задание 7. Разметка

1. Разработайте набор классов для текстовой разметки.
2. Класс `Paragraph` может содержать произвольное число других элементов разметки и текстовых элементов.
3. Класс `Text` – текстовый элемент.
4. Классы разметки `Emphasis`, `Strong`, `Strikeout` – выделение, сильное выделение и зачеркивание. Элементы разметки могут содержать произвольное число других элементов разметки и текстовых элементов.
5. Все классы должны реализовывать метод `toMarkdown(StringBuilder)`, который должен генерировать Markdown-разметку по следующим правилам:
   - текстовые элементы выводятся как есть;
   - выделенный текст окружается символами '*';
   - сильно выделенный текст окружается символами '__';
   - зачеркнутый текст окружается символами '~'.
6. Следующий код должен успешно компилироваться:
```
Paragraph paragraph = new Paragraph(List.of(
    new Strong(List.of(
        new Text("1"),
        new Strikeout(List.of(
            new Text("2"),
            new Emphasis(List.of(
                new Text("3"),
                new Text("4")
            )),
            new Text("5")
        )),
        new Text("6")
    ))
));
```
   Вызов `paragraph.toMarkdown(new StringBuilder())` должен заполнять переданный `StringBuilder` следующим содержимым:
```
__1~2*34*5~6__
```
7. Разработанные классы должны находиться в пакете `markup`.

Домашнее задание 8. Git

Загрузите решения домашних заданий (в том числе, сданных) в ваш персональный git-репозиторий. Со следующей недели сдача домашних заданий будет производиться только через репозитории.

Персональные репозитории имеют URL `https://www.kgeorgiy.info/git-students/year2023/<USER>/prog-intro`, где `<USER>` — имя пользователя в PCMS (пароль так же используется из PCMS). Если у вас нет логина/пароля в

PCMS, то соберитесь группой и обратитесь к Николаю Викторовичу.

Персональные репозитории являются клонами этого репозитория. В нём вы можете ознакомиться с правилами ведения репозитория и рекомендациями по его настройке.

Домашнее задание 9. Markdown to HTML

1. Разработайте конвертер из Markdown-разметки в HTML.
2. Конвертер должен поддерживать следующие возможности:
    1. Абзацы текста разделяются пустыми строками.
    2. Элементы строчной разметки: выделение (* или _), сильное выделение (** или __), зачеркивание (-- -), код (`)
    3. Заголовки (# * уровень заголовка)
3. Конвертер должен называться `md2html.Md2Html` и принимать два аргумента: название входного файла с Markdown-разметкой и название выходного файла с HTML-разметкой. Оба файла должны иметь кодировку UTF-8.
4. При выполнении этого ДЗ можно повторно использовать код ДЗ `markup`.
5. Конвертер может хранить исходные и сконвертированные данные в памяти, в том числе, одновременно.
6. Пример
    - Входной файл

```
# Заголовок первого уровня

## Второго

### Третьего ## уровня

#### Четвертого
# Все еще четвертого

Этот абзац текста,
содержит две строки.

    # Может показаться, что это заголовок.
Но нет, это абзац начинающийся с `#`.

#И это не заголовок.

###### Заголовки могут быть многострочными
(и с пропуском заголовков предыдущих уровней)

Мы все любим *выделять* текст _разными_ способами.
**Сильное выделение**, используется гораздо реже,
но __почему бы и нет__?
Немного --зачеркивания-- еще ни кому не вредило.
Код представляется элементом `code`.

Обратите внимание, как экранируются специальные
HTML-символы, такие как `<`, `>` и `&`.

Знаете ли вы, что в Markdown, одиночные * и _
не означают выделение?
Они так же могут быть заэкранированы
при помощи обратного слэша: \*.



Лишние пустые строки должны игнорироваться.

Любите ли вы *вложеные __выделения__* так,
как __--люблю--__ их я?
```

    - Выходной файл

```
<h1>Заголовок первого уровня</h1>
<h2>Второго</h2>
<h3>Третьего ## уровня</h3>
<h4>Четвертого
# Все еще четвертого</h4>
<p>Этот абзац текста,
содержит две строки.</p>
<p>    # Может показаться, что это заголовок.
```

```
Но нет, это абзац начинающийся с <code>#</code>.</p>
<p>#И это не заголовок.</p>
<h6>Заголовки могут быть многострочными
(и с пропуском заголовков предыдущих уровней)</h6>
<p>Мы все любим <em>выделять</em> текст <em>разными</em> способами.
<strong>Сильное выделение</strong>, используется гораздо реже,
но <strong>почему бы и нет</strong>?
Немного <s>зачеркивания</s> еще ни кому не вредило.
Код представляется элементом <code>code</code>.</p>
<p>Обратите внимание, как экранируются специальные
HTML-символы, такие как <code>&lt;</code>, <code>&gt;</code> и <code>&amp;</code>.</p>
<p>Знаете ли вы, что в Markdown, одиночные * и _
не означают выделение?
Они так же могут быть заэкранированы
при помощи обратного слэша: *.</p>
<p>Лишние пустые строки должны игнорироваться.</p>
<p>Любите ли вы <em>вложеные <strong>выделения</strong></em> так,
как <strong><s>люблю</s></strong> их я?</p>
```

- Реальная разметка

Заголовок первого уровня

Второго

Третьего ## уровня

Четвертого # Все еще четвертого

Этот абзац текста, содержит две строки.

Может показаться, что это заголовок. Но нет, это абзац начинающийся с #.

#И это не заголовок.

Заголовки могут быть многострочными (и с пропуском заголовков предыдущих уровней)

Мы все любим *выделять* текст *разными* способами. **Сильное выделение**, используется гораздо реже, но **почему бы и нет**? Немного ~~зачеркивания~~ еще ни кому не вредило. Код представляется элементом `code`.

Обратите внимание, как экранируются специальные HTML-символы, такие как <, > и &.

Знаете ли вы, что в Markdown, одиночные * и _ не означают выделение? Они так же могут быть заэкранированы при помощи обратного слэша: *.

Лишние пустые строки должны игнорироваться.

Любите ли вы *вложеные **выделения*** так, как ~~**люблю**~~ их я?

Домашнее задание 10. Чемпионат

1. Решите как можно больше задач Чемпионата северо-запада России по программированию 2019.
2. Материалы соревнования:
   - PCMS: Java. North-Western Russia Regional Contest - 2019
   - Условия задач
   - Разбор задач
3. Задачи для решения

Задача	Тема	Сложность

A.	Accurate Movement	Формула	5
B.	Bad Treap	Циклы	10
C.	Cross-Stitch	Графы	40
D.	Double Palindrome	Массивы	40
E.	Equidistant	Деревья	30
H.	High Load Database	Массивы	20
I.	Ideal Pyramid	Циклы	15
J.	Just the Last Digit	Матрицы	20
K.	King's Children	Массивы	40
M.	Managing Difficulties	Коллекции	10

4. Рекомендуемое время выполнения задания: 3 часа

Домашнее задание 11. Игра m,n,k

В этом домашнем задании вы можете пользоваться кодом, написанным на лекции. Он есть на сайте курса.

1. Реализуйте игру m,n,k (*k* в ряд на доске $m \times n$).
2. Добавьте обработку ошибок ввода пользователя. В случае ошибочного хода пользователь должен иметь возможность сделать другой ход.
3. Добавьте обработку ошибок игроков. В случае ошибки игрок автоматически проигрывает.
4. *Простая версия*. Доска может производить обработку хода за *O(nmk)*.
5. *Сложная версия*.
   - Доска должна производить обработку хода (проверку корректности, изменение состояния и определение результата) за *O(k)*.
   - Предотвратите жульничество: у игрока не должно быть возможности достать `Board` из `Position`.
6. *Бонусная версия*. Реализуйте `Winner` — игрок, который выигрывает всегда, когда это возможно (против любого соперника).
7. Код должен находиться в пакете `game`.

Домашнее задание 12. Выражения

1. Разработайте классы `Const`, `Variable`, `Add`, `Subtract`, `Multiply`, `Divide` для вычисления выражений с одной переменной в типе `int` (интерфейс `Expression`).
2. Классы должны позволять составлять выражения вида
```
new Subtract(
    new Multiply(
        new Const(2),
        new Variable("x")
    ),
    new Const(3)
).evaluate(5)
```
   При вычислении такого выражения вместо каждой переменной подставляется значение, переданное в качестве параметра методу `evaluate`. Таким образом, результатом вычисления приведенного примера должно стать число 7.
3. Метод `toString` должен выдавать запись выражения в полноскобочной форме. Например
```
new Subtract(
    new Multiply(
        new Const(2),
        new Variable("x")
    ),
    new Const(3)
).toString()
```
   должен выдавать `((2 * x) - 3)`.
4. *Сложный вариант.* Метод `toMiniString` (интерфейс `ToMiniString`) должен выдавать выражение с минимальным числом скобок. Например
```
new Subtract(
    new Multiply(
        new Const(2),
        new Variable("x")
    ),
    new Const(3)
```

```
).toMiniString()
```

должен выдавать `2 * x - 3`.

5. Реализуйте метод `equals`, проверяющий, что два выражения совпадают. Например,
```
new Multiply(new Const(2), new Variable("x"))
    .equals(new Multiply(new Const(2), new Variable("x")))
```

должно выдавать `true`, а
```
new Multiply(new Const(2), new Variable("x"))
    .equals(new Multiply(new Variable("x"), new Const(2)))
```

должно выдавать `false`.

6. Для тестирования программы должен быть создан класс `Main`, который вычисляет значение выражения x^2-2x+1, для `x`, заданного в командной строке.
7. При выполнении задания следует обратить внимание на:
    - Выделение общего интерфейса создаваемых классов.
    - Выделение абстрактного базового класса для бинарных операций.

Домашнее задание 13. Разбор выражений

1. Доработайте предыдущее домашнее задание, так что бы выражение строилось по записи вида
```
x * (x - 2)*x + 1
```
2. В записи выражения могут встречаться:
    - бинарные операции: умножение `*`, деление `/`, сложение `+` и вычитание `-`;
    - унарный минус `-`;
    - переменные `x`, `y` и `z`;
    - целочисленные константы в десятичной системе счисления, помещающиеся в 32-битный знаковый целочисленный тип;
    - круглые скобки для явного обозначения приоритета операций;
    - произвольное число пробельных символов в любом месте, не влияющем на однозначность понимания формулы (например, между операцией и переменной, но не внутри констант).
3. Приоритет операций, начиная с наивысшего
    1. унарный минус;
    2. умножение и деление;
    3. сложение и вычитание.
4. Разбор выражений рекомендуется производить методом рекурсивного спуска.
    - Алгоритм должен работать за линейное время.
    - Лексический анализ (токенизация) не требуется.

Домашнее задание 14. Обработка ошибок

1. Добавьте в программу, вычисляющую выражения, обработку ошибок, в том числе:
    - ошибки разбора выражений;
    - ошибки вычисления выражений.
2. Для выражения `1000000*x*x*x*x*x/(x-1)` вывод программы должен иметь следующий вид:
```
x       f
0       0
1       division by zero
2       32000000
3       121500000
4       341333333
5       overflow
6       overflow
7       overflow
8       overflow
9       overflow
10      overflow
```

Результат `division by zero` (`overflow`) означает, что в процессе вычисления произошло деление на ноль (переполнение).

3. При выполнении задания следует обратить внимание на дизайн и обработку исключений.
4. Человеко-читаемые сообщения об ошибках должны выводиться на консоль.
5. Программа не должна «вылетать» с исключениями (как стандартными, так и добавленными).