Домашнее задание 1. 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

   Аргументы могут содержать цифры и произвольные пробельные символы.
3. При выполнении задания можно считать что для представления входных данных и промежуточных результатов достаточен тип `int`.
4. При выполнении задания полезно ознакомиться с документацией к классам String и Integer.

Тесты к домашним заданиям

Домашнее задание 2. Реверс

1. Разработайте класс `Reverse`, читающий числа из стандартного входа, и выводящий их на стандартный вывод в обратном порядке.
2. В каждой строке входа содержится некоторое количество целых чисел (может быть 0). 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 строки так же должен быть обратным к порядку чисел во входе.
4. Примеры работы программы:
   - Вход:

     ```
     1 2
     3
     ```

     Выход:

     ```
     3
     2 1
     ```

   - Вход:

     ```
     1

     2 -3
     ```

     Выход:

     ```
     -3 2

     1
     ```

Домашнее задание 3. Сумма чисел в файле

1. Разработайте класс `SumFile`, записывающий сумму чисел из входного файла в выходной файл.
2. Числа во входном файле разделены переводами строк и/или пробельными символами.
3. Имена входного и выходного файла задаются в качестве аргументов командной строки.
4. Примеры работы программы:
   - Входной файл:

     ```
     1 2 3
     ```

     Выходной файл:

     ```
     6
     ```

   - Входной файл:

     ```
     1 2 -3
     ```

     Выходной файл:

     ```
     0
     ```

- Входной файл:

```
1 2
  3
```

  Выходной файл:

```
6
```

5. При выполнении задания можно считать что для представления входных данных и промежуточных результатов достаточен тип `int`.
6. В этом и последующих домашних заданиях, метод `main` не должен выбрасывать никаких исключений при любых (в том числе некорректных) входных данных.
7. В этом и последующих домашних заданиях, все ресурсы должны закрываться при любых (в том числе некорректных) входных данных.

Домашнее задание 4. Статистика слов

1. Разработайте класс `WordStat`, который будет подсчитывать статистику встречаемости слов во входном файле.
2. Словом называется неперывная 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 появения. Для каждого слова должна быть выведена одна строка, содежащая слово и число его вхождений во входной файл.
4. Имена входного и выходного файла задаются в качестве аргументов командной строки. Кодировка файлов: UTF-8.
5. Примеры работы программы:
   - Входной файл:

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

     Выходной файл:

```
to 2
be 2
or 1
not 1
that 1
is 1
the 1
question 1
```

   - Входной файл:

```
Monday's child is fair of face.
Tuesday's child is full of grace.
```

     Выходной файл:

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

   - Входной файл:

```
Шалтай-Болтай
Сидел на стене.
Шалтай-Болтай
Свалился во сне.
```

     Выходной файл:

```
шалтай-болтай 2
сидел 1
на 1
стене 1
свалился 1
во 1
сне 1
```

Домашнее задание 5. Быстрый реверс

1. Реализуйте свой аналог класса `Scanner`. Разработайте класс `Reverse`, читающий числа из стандартного входа, и выводящий их на стандартный вывод в обратном порядке.
2. Примените разработанный `Scanner` для решения задания «Реверс».
3. Мо[illegible] Р[illegible]

3. Модифицируите решени задания «Реверс» так, что бы оно работало за линеиное время.

Домашнее задание 6. Статистика слов++

1. Разработайте класс `WordStatIndex`, который будет подсчитывать статистику встречаемости слов во входном файле.
2. Словом называется неперывная 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 появения. Для каждого слова должна быть выведена одна строка, содежащая 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 файлам время.
6. Для реализации программы используйте Collections Framework.
7. Примеры работы программы:
   - Входной файл:

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

     Выходной файл:

     ```
     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
     ```

   - Входной файл:

     ```
     Monday's child is fair of face.
     Tuesday's child is full of grace.
     ```

     Выходной файл:

     ```
     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
     ```

   - Входной файл:

     ```
     Шалтай-Болтай
     Сидел на стене.
     Шалтай-Болтай
     Свалился во сне.
     ```

     Выходной файл:

     ```
     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 поизвольное число других элементов разметки и текстовых элементов.
3. Класс `Text` – текстовый элемент.
4. Классы разметки `Emphasis`, `Strong`, `Strikeout` – выделеные, сильное выделение и зачеркивание. Элементы разметки могут содержать поизвольное число других элементов разметки и текстовых элементов.
5. Все классы должны реализовывать метод `toMarkdown(StringBuilder)`, которой должен генерировать

   Markdown-разметку по следующим правилам:
   1. текстовые элементы выводятся как есть;
   2. выделеный текст окружается символами '*';
   3. сильно выделеный текст окружается символами '__';
   4. зачеркнутый текст окружается символами '~'.

[Crawler]

[Домашнее задание 10. Offline Browser]

6. Следующий код должен успешно компилироваться:

```
Paragraph paragraph = new Paragraph(Collections.singletonList(
    new Strong(Arrays.asList(
        new Text("1"),
        new Strikeout(Arrays.asList(
            new Text("2"),
            new Emphasis(Arrays.asList(
                new Text("3"),
                new Text("4")
            )),
            new Text("5")
        )),
        new Text("6")
    ))
));
```

   Вызов `paragraph.toMakdown(new StringBuilder())` должен заполнять переданный `StringBuilder` следующим содержимым:

```
__1~2*34*5~6__
```

7. Разработанные классы должны находиться в пакете `markup`.

Домашнее задание 8. 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` 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. 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>
```

```
<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 9. Web Crawler

1. Напишите Web Crawler, обходящий HTML-страницы на заданную глубину и вытаскивающий из них картинки.
2. Информация о HTML странице (класс `Page`:
   - `String url` – URL страницы (идентификатор);
   - `String title` – заголовок страницы (содержимое элемента `title`);
   - `List<Page> links` – ссылки (атрибут `href` элемента `a`), в порядке появления на странице;
   - `List<Page> backLinks` – ссылки, ведущие на страницу;
   - `List<Image> images` – Картинки на странице (элемент `img`), в порядке появления на странице.
3. Информация о картинке (класс `Image`):
   - `String url` – URL картинки (идентификатор);
   - `String file` – имя файла, в котором сохранена картинка;
   - `List<String> pages` – страницы, на которых встречается картинка.
4. Интерфес Web Crawler:

```
public interface WebCrawler {
    Page crawl(String url, int depth);
```

```
}
```

5. При загрузке на глубину два, должны быть загружены и проанализированы переданная страница и страницы, на которые она ссылается.
6. Для загрузки страниц и картинок можно использовать метод openStream класса URL.
7. Вы можете считать, что все страницы имеют кодировку UTF-8.

Домашнее задание 10. Offline Browser

1. Напишите Offline Browser, обходящий HTML-страницы на заданную глубину и сохраняющий их для offline-просмотра.
2. Вместе с HTML-страницами должны быть загружены сопутствующие ресурсы: картинки, скрипты и css-файлы. При сохранении не должны создаваться лишние копии ресурсов.
3. Ссылки на сохраненные страницы должны быть изменены так, чтобы работать без подключения к Интернету. Ссылки на другие страницы должны остаться без изменений.
4. Вы можете считать, что все страницы имеют кодировку UTF-8.
5. *Примечание*. В результате работы наивного offline-браузера, некоторые страницы (например, использующие динамическую загрузку скриптов и CSS) могут отображаться некорректно. Правильная загрузка таких сайтов не входит в данное домашнее задание.