**Задания на тему “Массивы“**

* Сдавайте задания на проверку по одному
* Каждую задачу реализуйте в отдельном классе в пакете *array*
* Имена классов и методов должны быть осмысленными и отражать их назначение. *Это часть может вызвать у вас затруднение — уделите придумыванию имен достаточное количество времени*
* Сопровождайте выводимые значения переменных, ввод с клавиатуры или ошибки подходящим текстом, информируя пользователя о возникающих событиях
* Текст сообщения об ошибке должен быть информативным и включать, как ошибочное значение, так и допустимое
* Программа не должна завершаться с ошибкой — делайте нужные проверки
* Не используйте коллекции
* Вызывайте в *main()* метод, отвечающий за выполнение основной части программы отдельно от метода по отображению чего-либо (кроме текста ошибок) на консоль — для заданий не со \*

* Примените к своему коду правила из [главы](https://topjava.ru/blog/pravila-formatirovaniya-koda-v-java2#3) *2.7.*
* Перед сдачей ДЗ сверьте свое решение со списком [рекомендаций](https://docs.google.com/document/d/1XRobwahUDrbpMzyyqgzRsfHX31UgNWEGepwV1wSuZ-M/edit#bookmark=id.a0q4hke7bdoq)
* Присылайте на проверку только тот код, в котором нет замечаний [линтера](https://topjava.ru/blog/nastroyka-checkstyle-v-intellij-idea)

1. **РЕВЕРС БАНКОВСКИХ ТРАНЗАКЦИЙ**

*В банковской системе необходимо реализовать функционал для отображения истории транзакций клиента. Последние транзакции должны показываться первыми.*

* реализуйте метод, принимающий массив целых чисел разной длины, обозначающих суммы транзакций клиентов
* вызовите его для следующих [данных](https://topjava.ru/blog/rukovodstvo-po-massivam-v-java-ch1#6) (без использования *varargs*):

|  |
| --- |
| массив нулевой длины **null** 5 6, 8, 9, 1 13, 8, 5, 3, 2, 1, 1 |

* выполните реверс значений массива без сортировки
* исходный массив не должен меняться
* обработайте особые случаи:
  + пустой массив (нет транзакций)
  + null (ошибка в данных)
* выведите значения массива, соблюдая все отступы, как в образце:

|  |
| --- |
| Исходные транзакции: [1, 2, 3, 4]  В обратном порядке: [4, 3, 2, 1] |

1. **АНИМАЦИЯ ЗАГРУЗКИ**

*Программа должна имитировать процесс "взлома" с помощью анимированного спиннера (вращающегося символа).*

* используйте массив символов {'-', '\\', '|', '/'} для отображения спиннера
* все символы должны отображаться на одной строке (используйте \r для возврата каретки на один символ)
* спиннер должен выполнить не менее трех полных оборотов
* используйте задержку между выводами каждого символа спиннера с помощью [Thread.sleep()](https://javarush.com/quests/lectures/questsyntaxpro.level08.lecture03)
* после анимации в той же строке выведите результат. Если случайное число [0, 100) больше 70:
  + *"Access Granted!"*
  + иначе *"Access Denied!"*
* измените цвет вывода (например, красный для *"Access Granted!"*, зелёный для *"Access Denied!"*) с помощью [ANSI-кодов](https://stackoverflow.com/questions/5762491/how-to-print-color-in-console-using-system-out-println)

****

1. **ВЫЧИСЛЕНИЕ ФАКТОРИАЛА**

* реализуйте метод, принимающий массив чисел разной длины для вычисления [факториала](https://youtu.be/FR55Pdavza4?si=_rTKIl_x7bru0Wr-) каждого значения
* вызовите его для следующих данных (с использованием [*varargs*](https://topjava.ru/blog/rukovodstvo-po-massivam-v-java-varargs)):

|  |
| --- |
| массив нулевой длины **null** 8, 0, 9 -3, 1, 7, 13 -22, -0 |

* поместите в новый массив результат вычисления каждого факториала
* сверьте полученные результаты с [таблицей](https://ru.onlinemschool.com/math/formula/factorial_table/) факториалов
* выводите сообщение для отрицательных чисел, т. к. для них факториал не определен:

|  |
| --- |
| Ошибка: факториал -5! не определен |

* выведите, используя [тернарный оператор](https://topjava.ru/blog/ternarnyy-operator-v-java), каждое выражение в формате:

|  |
| --- |
| 4! = 1 \* 2 \* 3 \* 4 = 24 1! = 1 0! = 1 |

1. **УДАЛЕНИЕ ЭЛЕМЕНТОВ МАССИВА, ПРЕВЫШАЮЩИХ ЗАДАННОЕ ЗНАЧЕНИЕ**

* реализуйте метод, принимающий индекс ячейки массива
* перезапишите нулями все значения в массиве, превышающие значение из переданного индекса
* вызовите метод для следующих данных:

|  |
| --- |
| -1 15 0 14 |

* заполните массив пятнадцатью случайными вещественными числами в полуинтервале [0, 1)
* выведите:
  + исходный и измененный массив по восемь и семь чисел в строке
  + значение из ячейки по переданному индексу
  + количество обнуленных ячеек
* выводите все числа с тремя знаками после запятой
* если индекс не соответствует условию, то ничего кроме текста об ошибке выводиться не должно

1. **ВЫВОД ОТСОРТИРОВАННЫХ СИМВОЛОВ В ВИДЕ ТРЕУГОЛЬНИКА**

* реализуйте метод, принимающий границы [отрезков символов](https://www.freecodecamp.org/news/ascii-table-hex-to-ascii-value-character-code-chart-2/), а также маркер направления их сортировки (по возрастанию/убыванию от вершины к основанию)
* вызовите его для следующих данных:

|  |
| --- |
| 0, 9, **true** /, !, **false** A, J, **false** |

где *true* — сортировка по возрастанию, а *false* — по убыванию

* отобразите символы в виде треугольников, смотрящих вершиной вверх (каждый новый символ на своей строке). Пример:

|  |
| --- |
| 6     555    44444   3333333  222222222 |

* для формирования треугольника используйте [StringBuilder](https://youtu.be/Vw2GPl3APD4?si=QuINJmjVDGnRLfOA) и [repeat()](https://topjava.ru/blog/java-11-string-api-additions#:~:text=%D1%81%C2%A0%D0%BD%D0%B8%D0%BC%D0%B8%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D1%82%D1%8C.-,2.%20repeat(),-%D0%9A%D0%B0%D0%BA%20%D1%81%D0%BB%D0%B5%D0%B4%D1%83%D0%B5%D1%82%20%D0%B8%D0%B7)
* выводите сообщение об ошибке, если левая граница больше правой

|  |
| --- |
| Ошибка: левая граница (90) > правой (33) |

1. **ЗАПОЛНЕНИЕ МАССИВА УНИКАЛЬНЫМИ ЧИСЛАМИ**

* реализуйте метод, принимающий границы отрезка целых чисел и количество чисел, выводимых в каждой строке консоли (фактическое количество чисел в массиве может быть меньше ожидаемого)
* вызовите его для следующих данных:

|  |
| --- |
| -30, -10, 23 10, 50, 10 -34, -34, 0 -1, 2, -3  5, -8, 2 |

* заполните массив случайными уникальными числами в заданных границах
* длина массива должна составлять *75%* от длины отрезка
* выведите числа отсортированными по возрастанию
* левая граница отрезка не должна превышать правую
* выводите сообщение об ошибке, если данные не соответствуют критериям:

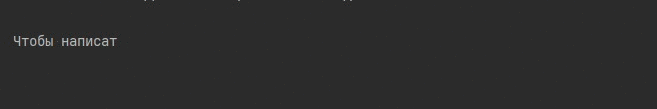
|  |
| --- |
| Ошибка: количество чисел в строке не должно быть < 1 (-8) Ошибка: длина массива должна быть > 0 (-3) Ошибка: левая граница (50) > правой (31) |

1. **ВЫВОД ТЕКСТА С ЭФФЕКТОМ ПИШУЩЕЙ МАШИНКИ**

* реализуйте метод, принимающий строку
* вызовите его для следующих данных:

|  |
| --- |
| 1) Java - это C++, из которого убрали все пистолеты, ножи и дубинки. - James Gosling  2) Чтобы написать чистый код, мы сначала пишем грязный код, затем рефакторим его. - Robert Martin  3) **null**  4) пустая строка |

* найдите в переданных строках самое короткое и самое длинное слово (без учета знаков препинания)
* выведите в консоль:
  + большими буквами все слова между найденными (включительно)
  + текст с эффектом пишущей машинки (побуквенно):



* для достижения требуемого эффекта используйте задержку между выводимыми буквами с помощью [Thread.sleep()](https://javarush.com/quests/lectures/questsyntaxpro.level08.lecture03)
* код должен уметь работать с любым текстом и знаками препинания

**Задание со \* (по желанию)**

1. **РЕАЛИЗАЦИЯ УТИЛИТНЫХ КЛАССОВ**

Код предыдущих заданий можно объединить в две группы: реализация условий задач (исключая код вывода на консоль) и вывод на консоль. Затем код каждой группы необходимо поместить в подходящие классы, например, *Arrays* и *Console*. Так как их реализация будет состоять только из статических методов, они будут называться *утилитными* ([1](https://topjava.ru/blog/rukovodstvo-po-modifikatoru-static-v-java#3), [2](https://www.baeldung.com/java-helper-vs-utility-classes#java-utility-classes)). Для запуска методов этих классов и передачи в них входных данных, требуется создать третий класс, например, *Main*.

Подобные манипуляции обычно выполняют с целью разделения ответственности между разными классами: интерфейс (вывод в консоль) размещают отдельно от реализации основного функционала (бизнес-логики). Данный подход упрощает сопровождение, использование и понимание кода.

* создайте три класса: *Arrays*, *Console* и *Main*
* переместите код предыдущих ДЗ в данные классы, удалив старые классы
* не дробите код отдельной задачи на множество методов в классе *Arrays*: одна реализация задачи — один метод, какой бы длины он не был. Дробление возможно только, если код используется в разных методах
* вызывайте методы из класса *Console* для отображения любой информации на консоль*,* включая текст ошибок*.* В *Arrays* не должно быть кода, выводящего что-либо на консоль *(кроме вызова метода Console для вывода ошибки)*
* текст, сопровождающий вывод результатов работы методов *Arrays*, реализуйте в виде отдельных методов в *Console*. Вызывайте их в *Main*
* все публичные методы *Arrays* должны возвращать результат своей работы, который необходимо выводит в *Main*, вызывая подходящие методы из *Console*
* вынесите код каждого задания из *main()* в отдельный метод
* реализуйте в *Arrays* и *Console* приватные конструкторы
* при именовании и реализации методов в *Console* исходите из принципов:
  + *имена должны отражать то, что они делают*  
    Неверным будет назвать метод, выводящий, например, выражения факториалов просто *print()*, так как такое имя, лишенное контекста, будет неинформативным
  + *методы ничего не знают про класс Arrays*Не связывайте их имена с действиями, которые выполняются в методах *Arrays*. Например, если метод отображает переданный в него массив, то неверно будет его назвать *printUniqueNums()*, так как ему все равно, уникальные числа в массиве или нет. Исходя из этого, реализуйте методы максимально универсально (где это возможно)
* разместите методы в *Arrays* и *Console* в алфавитном порядке по возрастанию

Рекомендации

1. Т. к. в коде часто используется длина массива, то, для его разгрузки от однотипных действий, создавайте локальные переменные для хранения длины (*len* или *length)*

1. Используйте для вывода данных массива (где это возможно) цикл [for-each](https://topjava.ru/blog/rukovodstvo-po-massivam-v-java-ch1#9)

1. Выносите дублирующий код в отдельные методы

1. Используйте [isBlank()](https://topjava.ru/blog/java-11-string-api-additions#:~:text=%D0%B1%D0%BE%D0%BB%D1%8C%D1%88%D0%B5%20U%2B0020.-,4.%20isBlank(),-%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%20%D1%8D%D0%BA%D0%B7%D0%B5%D0%BC%D0%BF%D0%BB%D1%8F%D1%80%D0%B0%20isBlank) для проверки строк на пустоту и пробелы

1. Применяйте правило [размещения](https://topjava.ru/blog/pravila-formatirovaniya-koda-v-java2#:~:text=%D0%9F%D1%80%D0%B8%D0%B2%D0%B0%D1%82%D0%BD%D1%8B%D0%B5%20%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B%20%D1%80%D0%B0%D0%B7%D0%BC%D0%B5%D1%89%D0%B0%D0%B9%D1%82%D0%B5,%D1%82%D0%B5%D0%BA%D1%81%D1%82%20%D0%B2%C2%A0%D0%BA%D0%BD%D0%B8%D0%B3%D0%B5) приватных методов в классе

1. Не используйте в именах массивов повсеместно слово *array* (кроме методов, в которых имя не важно или когда метод является универсальным). Придумывайте более информативные имена, сообщающие сведения о хранимых в них данных

Не давайте всем массивам, хранящим числа, имя *numbers* т. к. числа бывают разные и несут разный смысл. Давайте переменным осмысленные имена