



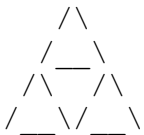
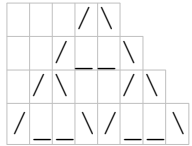
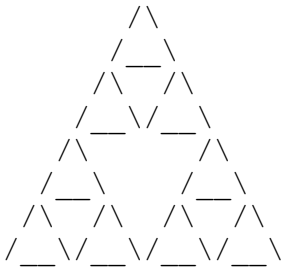
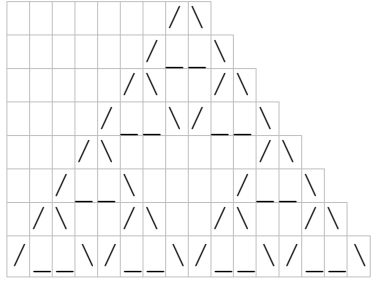
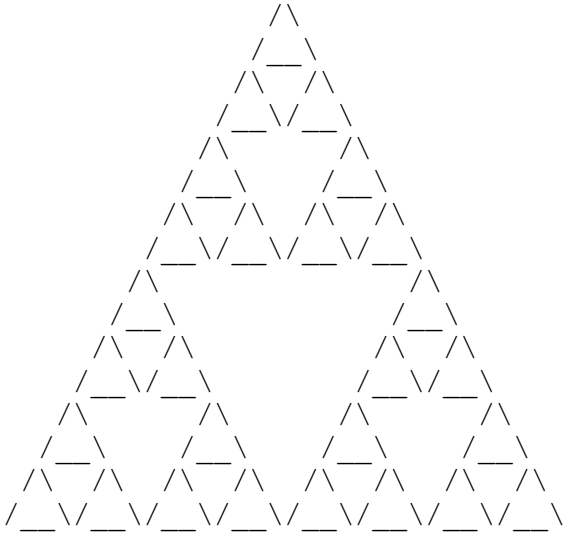
## Задача 12. Рекурсия

Отличный материал про рекурсию (правда, с примерами на Pascal):

<http://www.tvd-home.ru/recursion>

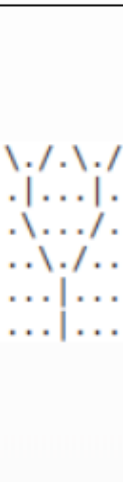
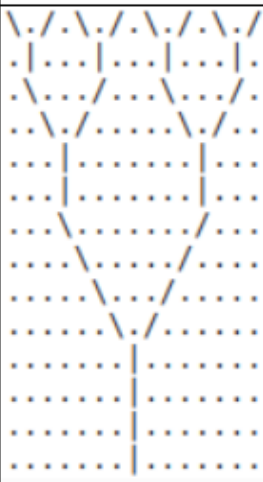
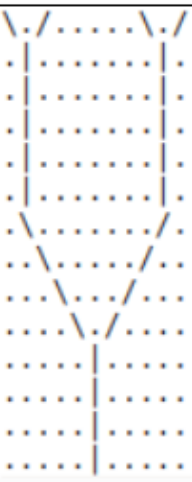
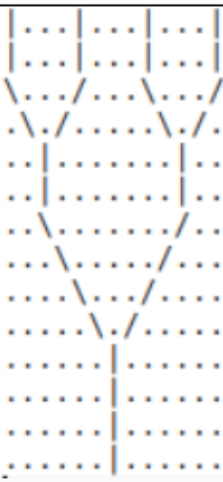

### Варианты:

1. Сгенерировать треугольник Серпинского с помощью псевдографики в следующем виде:

Глубина рекурсии, N	Результат	Комментарий
1		 <p>Для наглядности в виде сетки обозначены границы символов, т.е. каждая клеточка – один символ (пустые клетки – пробелы, символы перевода строк не показаны)</p>
2		
3		
4		

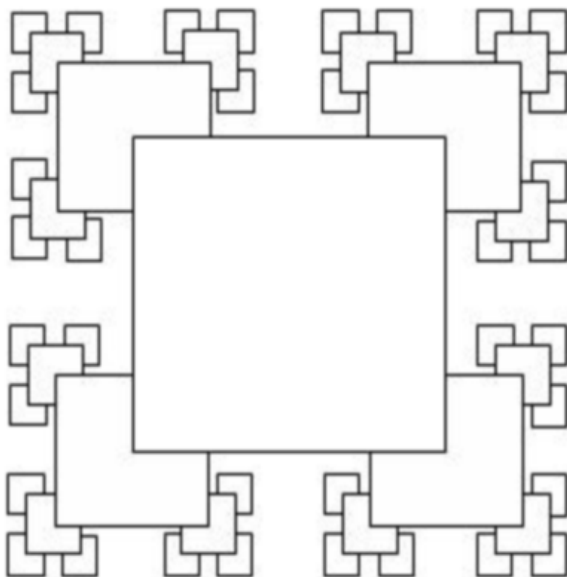
Примечание: Очевидно, что для просмотра псевдографики необходимо использовать моноширинный

2. На вход программы подаётся строка текста, состоящая из заглавных букв латинского алфавита Y и I. Длина строки не менее одного и не более 7 символов. Реализовать программу, которая по заданному шифру рисует соответствующий рисунок. Принцип построения рисунка по шифру заключается в следующем. В рисунке несколько рядов букв разного размера. Количество рядов равно количеству букв в шифре. Все буквы имеют высоту, равную различным степеням двойки. Самая верхняя буква рисунка (это последняя буква в шифре) всегда имеет высоту 2 символа. Вторая сверху буква (предпоследняя в строке шифра) имеет высоту 4 символа, и так далее, каждая следующая в два раза больше предыдущей. У буквы Y половину высоты занимает нижняя часть (вертикальная линия), а половину высоты - расходящиеся в стороны наклонные линии. Буква I - это вертикальная линия соответствующей высоты. Верхние буквы начинаются в концах нижних букв.

Шифр	YY	YYY	YIY	YII	YI
Рисунок					

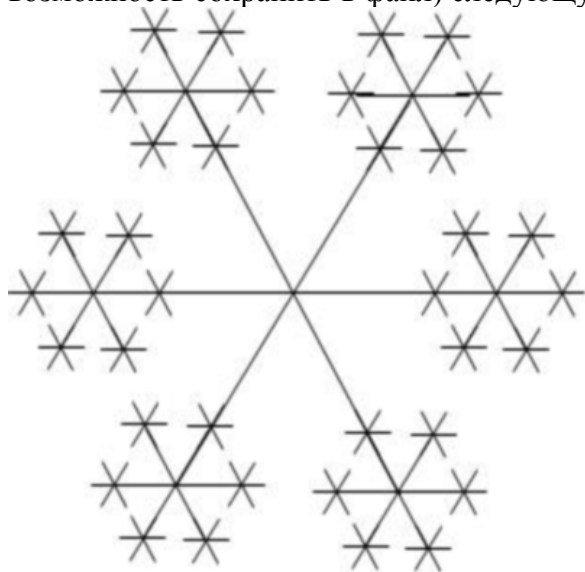
Примечание: Очевидно, что для просмотра псевдографики необходимо использовать моноширинный шрифт.

3. Нарисовать (в BufferedImage с использованием Graphics2D) и показать на форме (с возможность сохранить в файл) следующую фигуру в 2-х вариантах:
- большие квадраты рисуются поверх меньших
  - меньшие квадраты рисуются поверх больших



Задается размер фигуры (изображения), глубина рекурсии, а также флаг того, большие или меньшие фигуры должны рисоваться сверху.

4. Нарисовать (в `BufferedImage` с использованием `Graphics2D`) и показать на форме (с возможностью сохранить в файл) следующую фигуру:



Задается размер фигуры (изображения) и глубина рекурсии.

5. Проверить правильность расстановки скобок в выражении.  
Выражение - последовательность любых символов, среди которых могут встретиться скобки трех видов - круглые "()", квадратные "[]" и фигурные "{}".  
Скобки расставлены правильно, если каждой открывающейся скобке соответствует такая же закрывающаяся и для соответствующей пары скобок в выражении внутри этой пары скобок скобки также расставлены правильно, примеры
- "про(то)кол" - правильно
  - "aa[w[w(w)we{}]{we[[]we{}w{e(ee)ee}qw]wq}e32]]2e33" - правильно
  - "[[O{}]{[]{}{O}}]" - правильно (предыдущий пример, только без букв)
  - "#(0)(1)[2]." - правильно
  - "abc[]" - неправильно
  - "33[ф(ф)ф)ы" - неправильно
  - ")a(" - неправильно
  - и т.д.

Задача обязательно должна быть решена рекурсивно (т.к. возможно решение и без рекурсии).

6. Дано двумерное поле из клеток, некоторые клетки пустые, некоторые - стена (на данные клетки нельзя поставить фигуру). В какой-то пустой клетке находится шахматная фигура конь. Найти все клетки, до которых можно добраться этой фигурой за N (или менее) ходов.

Задача обязательно должна быть решена рекурсивно (т.к. возможно решение и без рекурсии).

7. (\*) Узлы дерева задаются в текстовом виде следующим образом: каждый узел дерева - описывается в круглых скобках, где первый элемент - вершина (очевидно, описываться в скобках не может), остальные - потомки. Необходимо нарисовать в виде дерева, как в примере ниже (формулировка несколько сумбурная, но примеры все поясняют):

Входные данные	Результат
(a, b, (c), d)	<pre>       a         -----               b c d </pre>
(a, second, (abc, y, (x, 7), uuu, (8, 9, (10, 1))), abcdcdcba)	<pre>               a                 -----                               second      abc      abcdcdcba                       -----                         y x uuu 8                           -   -                             7   9 10                               -                               1 </pre>

Примечание 1: Для описания дерева (узла дерева) реализовать класс `TreeNode`:

```
public class TreeNode {
    // еще лучше сделать поле ниже закрытым и добавить соответствующие методы
    public List<TreeNode> children = new ArrayList<>(TreeNode);
    ...
    // оставшая реализация
}
```

Примечание 2: Очевидно, что для просмотра псевдографики необходимо использовать моноширинный шрифт. При желании можно использовать более подходящие символы, показывающие сопряжение элементов.

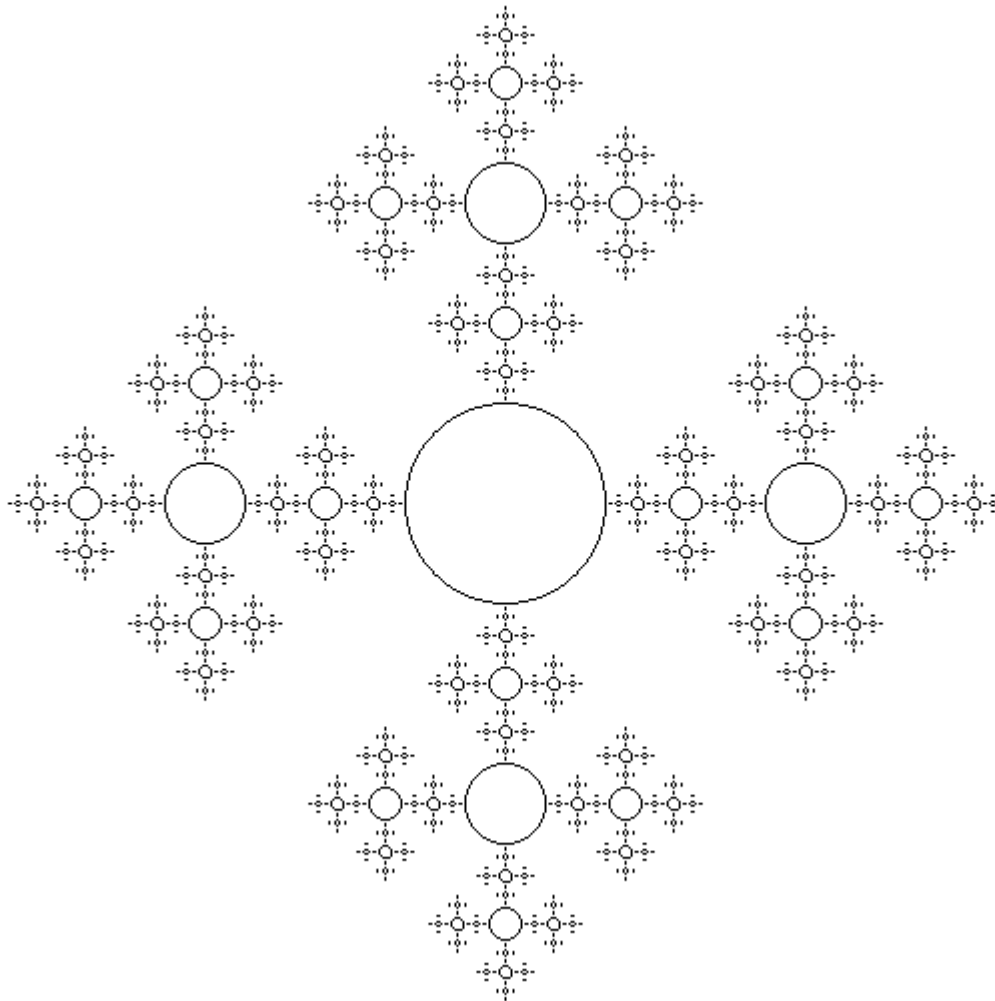
8. (\*) Предыдущая задача, только дерево теперь рисуется по-другому (см. примеры):

Входные данные	Результат
(a, b, (c), d)	a +-b +-c +-d
(a, second, (abc, y, (x, 7), uuu, (8, 9, (10, 1))), abcdcdcb)	a +-second +-abc   +-y   +-x     +-7   +-uuu   +-8   +-9   +-10   +-1 +-abcdcdcb

9. Число правильных скобочных структур длины 6 равно 5:  $()()()$ ,  $(())()$ ,  $()(())$ ,  $((()))$ ,  $(())()$ . Напишите рекурсивную программу генерации всех правильных скобочных структур длины  $2n$ .  
Указание: Правильная скобочная структура минимальной длины « $()$ ». Структуры большей длины получаются из структур меньшей длины, двумя способами:

- если меньшую структуру взять в скобки,
- если две меньших структуры записать последовательно.

10. Нарисовать (в BufferedImage с использованием Graphics2D) и показать на форме (с возможность сохранить в файл) следующую фигуру:



Задается размер фигуры (изображения) и глубина рекурсии.

11. Создайте функцию для перебора всех возможных представлений натурального числа N в виде суммы других натуральных чисел.

Для универсальности будем передавать в функцию другую функцию (а точнее функциональный интерфейс), которая будет использоваться в качестве callback-вызова, который будет вызываться для каждого варианта представления натурального числа N в виде суммы других натуральных чисел.

Функция должна иметь следующую сигнатуру:

```
public static void genAll(int k, Consumer<List<Integer>> callback)
```

И вызываться будет как-то так:

```
genAll(10, (List<Integer> next) -> {
    // например, распечатка очередного варианта представления
    // в виде суммы чисел из списка next
    ...
});
```

Если все еще непонятно, то почитайте про интерфейсы, функциональные интерфейсы (<https://metanit.com/java/tutorial/9.3.php>) и лямбда-выражения.

12. На вход передаётся путь к папке (в виде строки, например "Z:\\ВвП"). Напишите функцию, которая находит наиболее "глубоко" лежащие элементы ("листья") - это могут быть и файлы и папки.

Для чтения набора файлов и подпапок в папке использовать класс `java.io.File` (метод `listFiles()`).

Вначале надо найти максимальную глубину пути, а затем еще раз рекурсивно обойти все элементы и вернуть самые глубокие.

Для универсальности будем передавать в функцию другую функцию (а точнее функциональный интерфейс), которая будет использоваться в качестве callback-вызова, который будет вызываться для каждого варианта представления натурального числа N в виде суммы других натуральных чисел.

Функция должна иметь следующую сигнатуру:

```
public static void genMaxDepthElements(String path,
    Consumer<String> callback)
```

И вызываться будет как-то так:

```
genMaxDepthElements ("Z:\\", (List<Integer> path) -> {
    // например, печать элемента
    System.out.println(path);
});
```

13. Предыдущая задача, только теперь надо найти все файлы, имя которых (включая расширение) задано в виде маски со звездочками и вопросиками (например "input??.\*").  
Примечание: Для проверки соответствия имени файла шаблону можно воспользоваться функциями из проекта к лекции о рекурсии.

14. Рекурсивная реализация перебора всех возможных сочетаний элементов массива / списка:

<https://ru.wikipedia.org/wiki/Сочетание>

N определяется размером массива / списка. K передается в качестве параметра.

Для универсальности будем передавать в функцию другую функцию (а точнее функциональный интерфейс), которая будет использоваться в качестве callback-вызова, который будет вызываться для каждого нового сочетания.

Функция должна иметь следующую сигнатуру:

```
public static void genAll(int[] elements, int k,
    Consumer<int[]> callback)
```

или, если будут использованы списки, то:

```
public static void genAll(List<Integer> elements, int k,
    Consumer<List<Integer>> callback)
```

И вызываться будет как-то так (для массивов):

```
int[] elements = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
// 5 из 10 (10 - кол-во элементов в переданном массиве)
genAll(elements, 5, (int[] next) -> {
    // например, распечатка очередного сочетания next
    ...
});
```

Если все еще непонятно, то почитайте про интерфейсы, функциональные интерфейсы (<https://metanit.com/java/tutorial/9.3.php>) и лямбда-выражения.

15. Задача 9, но для размещений:

<https://ru.wikipedia.org/wiki/Размещение>

16. Задача 10, но для перестановок:

<https://ru.wikipedia.org/wiki/Перестановка>

17.