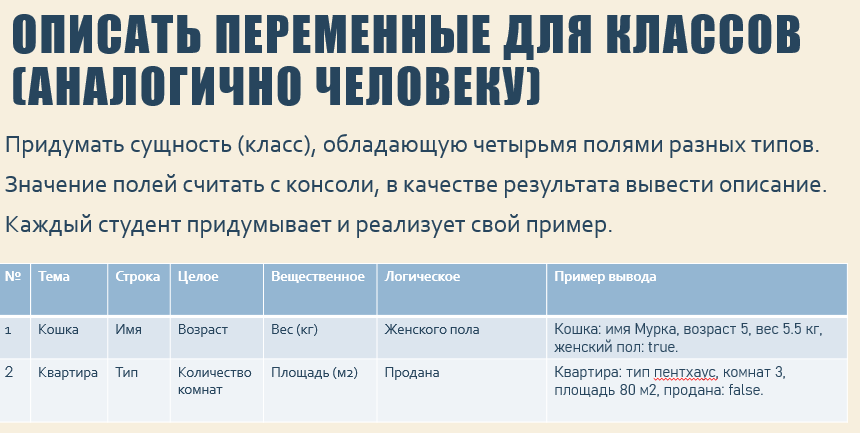
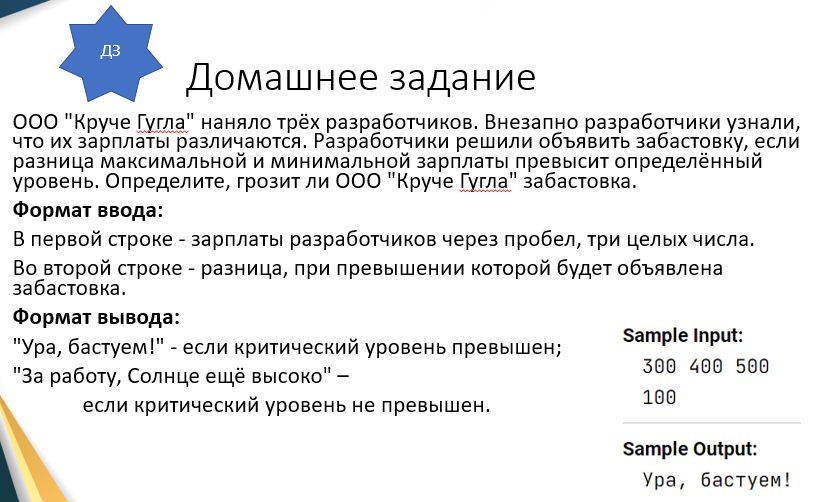
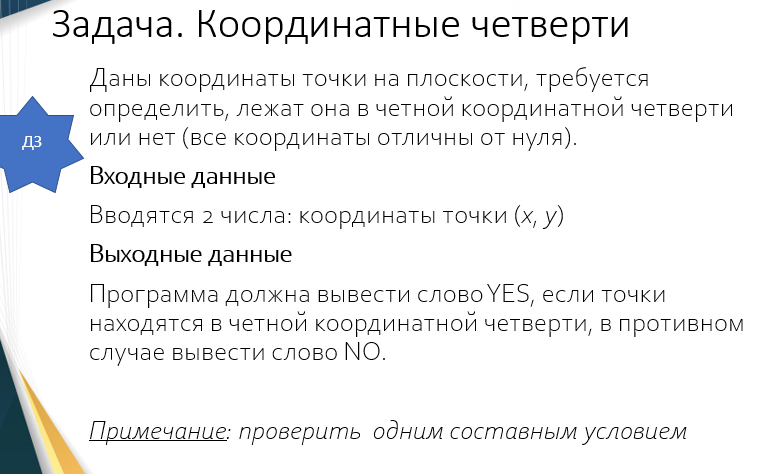
# 16.09.22 Задачи 1(базовые)

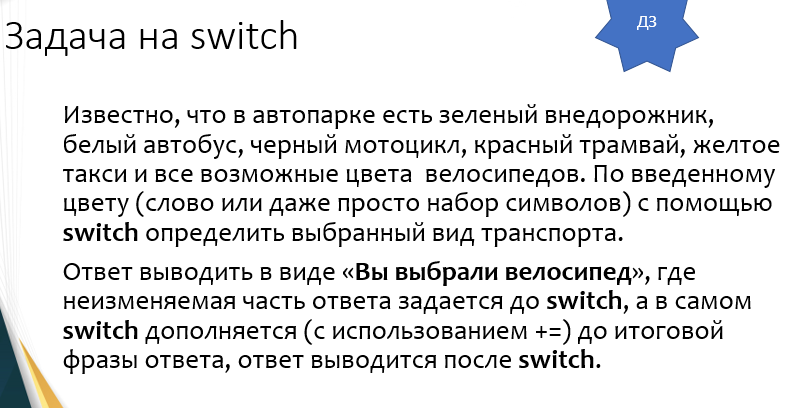




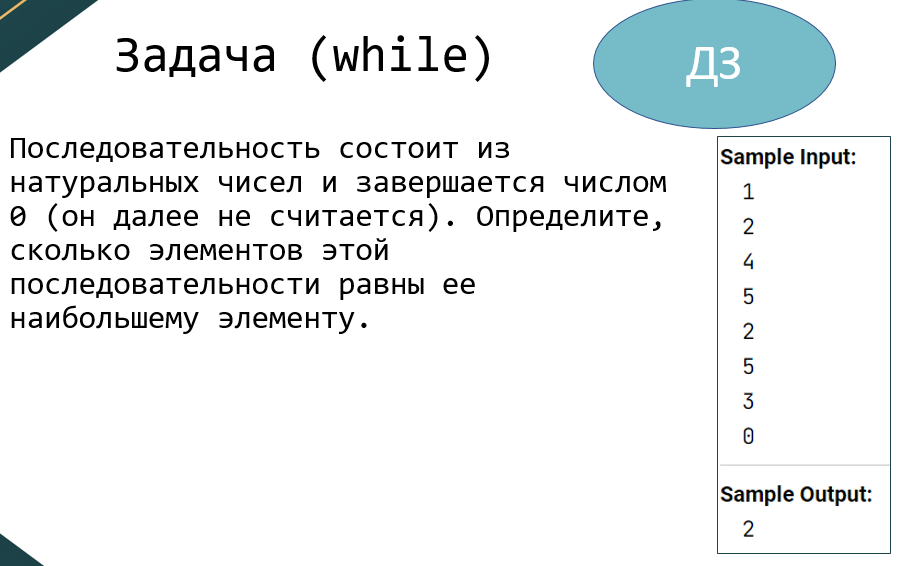
# 23.09.22 Задачи 2(if)

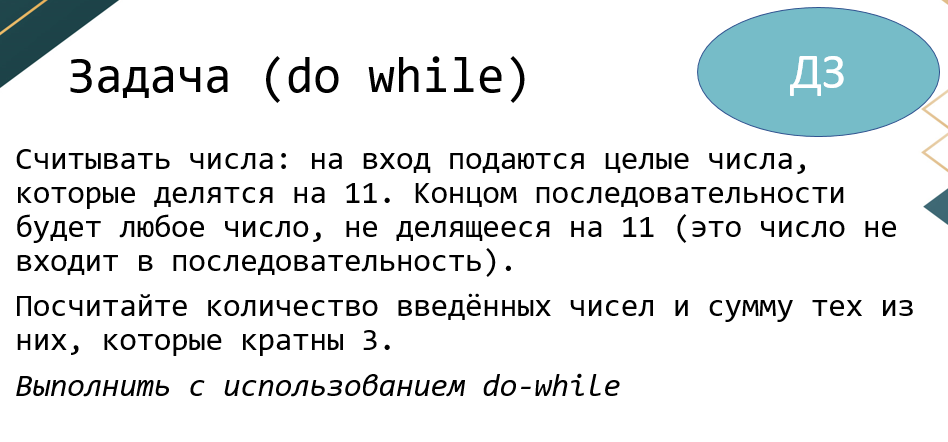




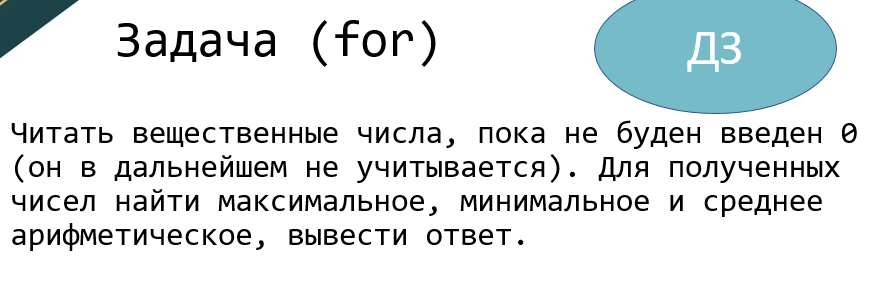


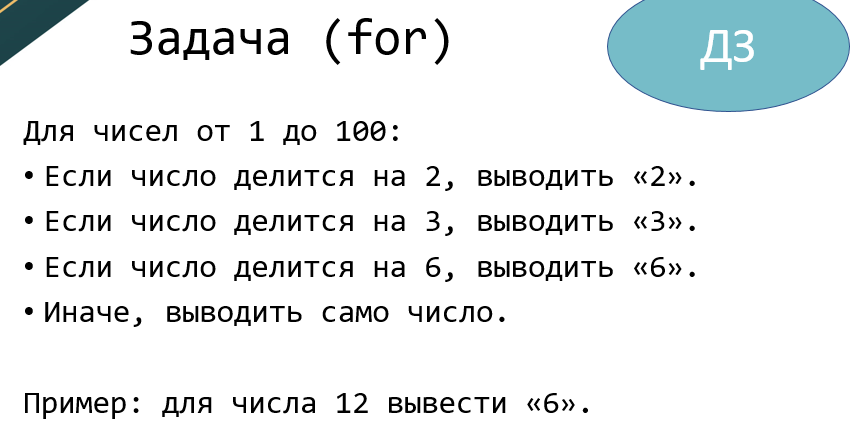
# 30.09.22 Задачи 3 (while)



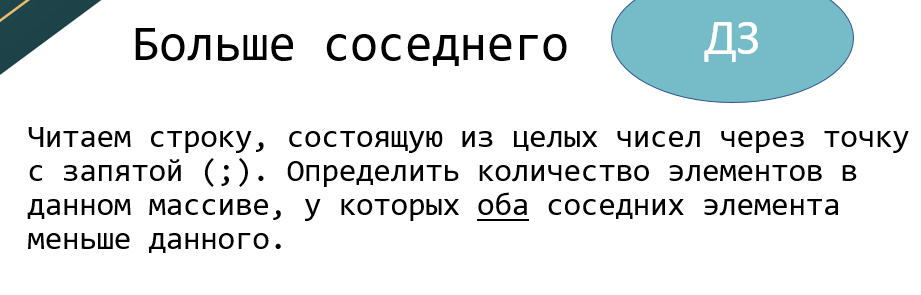


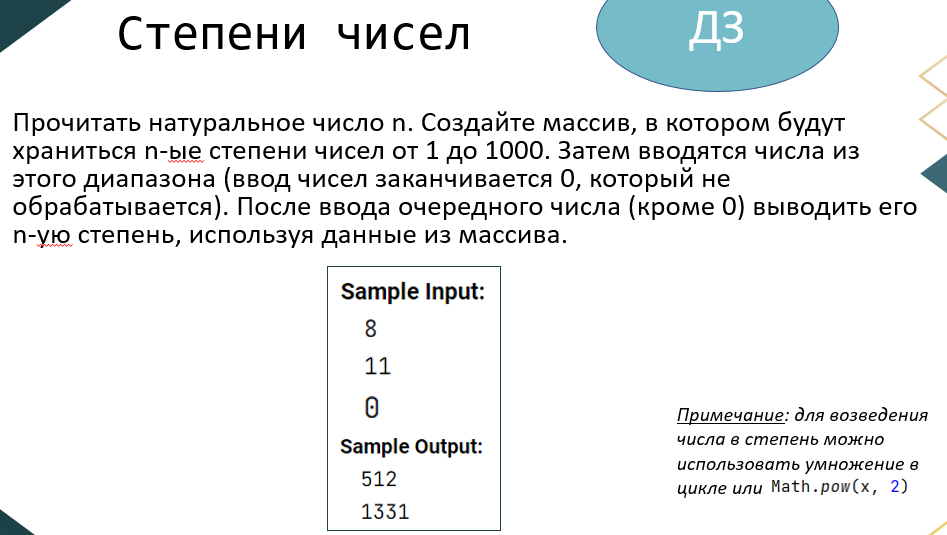
# 7.10.22 Задачи 4 (for)



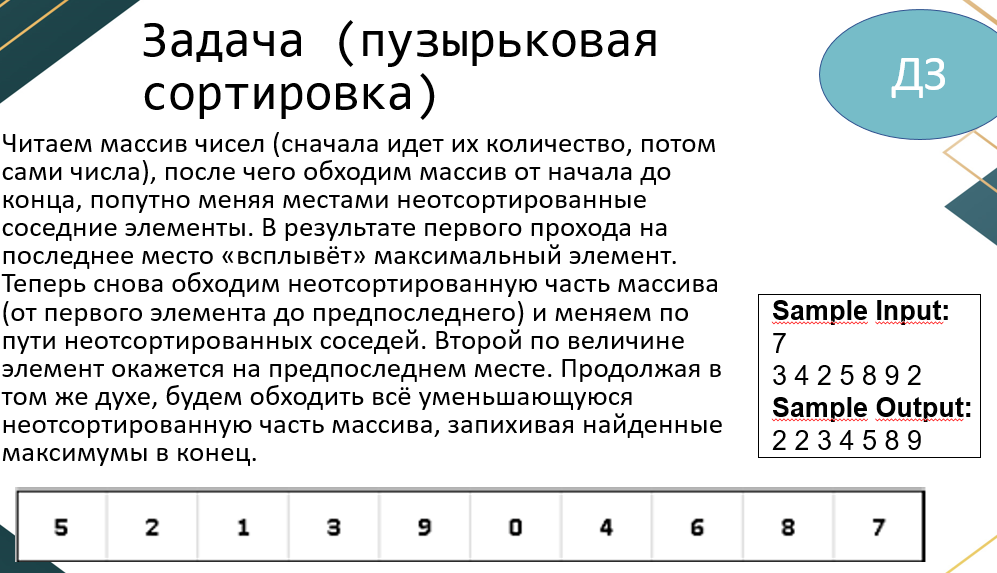


# 14.10.22 Задачи 5 (массивы начало)

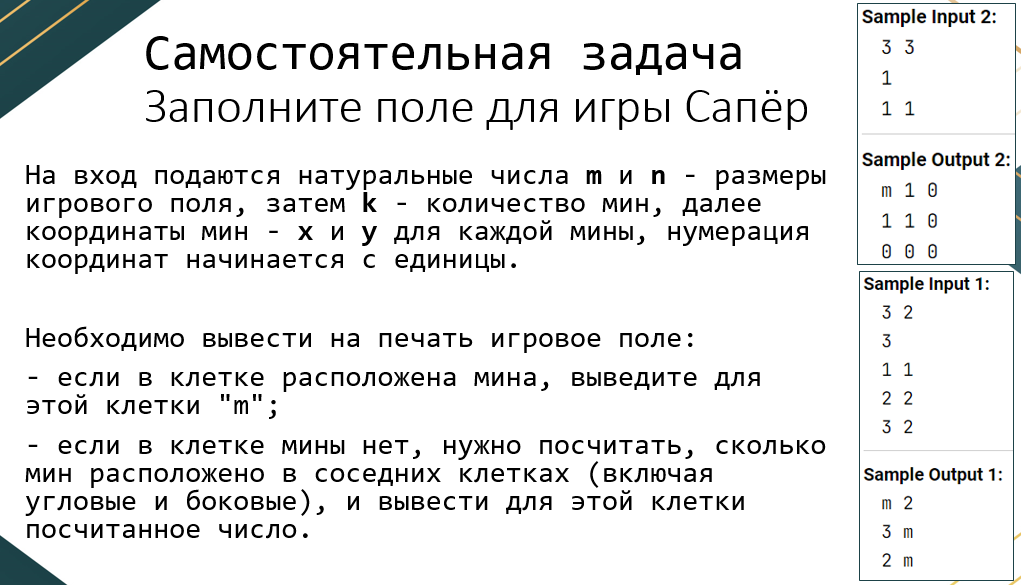




# 21.10.22 Задачи 6(пузырьковая сортировка)



# 28.10.22 СР 1(Сапер)



# 18.11.22 СР 2(Торговые индикаторы)

1. Создать достаточно большой массив вещественных чисел. В специальном методе в цикле проинициализировать его (размер массива и значения элементов определите самостоятельно)
2. Выбрать не совсем элементарный торговый индикатор. Индикаторы, выбранные студентами, не должны совпадать.

ФИО, название выбранного индикатора и ссылку с его описанием записать в таблицу:[Торговые индикаторы](https://docs.google.com/spreadsheets/d/11_0Qag5B__2Xq3ZAVZXXIQea4ZjN5pIJOidZzP12cZw/edit?usp=sharing)

Примеры можно увидеть по ссылке [Финам. Примеры индикаторов](https://www.finam.ru/publications/item/luchshie-indikatory-dlya-treiydinga-na-fondovom-rynke-20211103-144400/)

1. Реализовать считывание вводимых пользователем параметров индикатора (например, для EMA задаются период (количество измерений) N и доля использования значения цены, для MACD – задаются EMA\_s, EMA\_l и SMA\_a, для RSI – период, уровни перекупленности и перепроданности и так далее).
2. В отдельном методе реализовать расчет значений индикатора (результат хранить в новом массиве).
3. Выводить результаты в понятном виде (например, в столбик пары исходных значений и итоговых значений).
4. При сдаче показывать и формулы, по которым проводились расчеты.

Сдача проходит очно, требует объяснения студентом программы. Возможно получение дополнительных заданий по доработке программы.

Задание, выполненное в классе, оценивается выше выполненного дома.

# 25.11.22 Задачи 7. Списки

В одну строку через пробел записаны целые числа, нужно их прочитать и записать в список из целых чисел.

Из полученного списка нужно удалить все числа, которые стоят на четных позициях, а затем вывести полученный список в обратном порядке

**Sample Input:**

1 2 3 4 5 6 7

**Sample Output:**

6 4 2

# 2.12.22 Задача 8. Map. Статистика по работникам

Вводится число n. Затем 2\*n строк. Каждая пара строк - имя человека и его профессия. Определите, сотрудников какой профессии больше всего. Выведите на экран это количество, профессию и их имена на экран в том же порядке, в котором они вводились. Гарантируется, что будет введено не более 100 сотрудников.

**Sample Input:**

7

Иван

Токарь

Семён

Программист

Анатолий

Кузнец

Вася

Финансист

Кирилл

Программист

Оля

Программист

Валера

Токарь

**Sample Output:**

3

Программист

Семён

Кирилл

Оля

9.12.22. Самостоятельная задача

Необходимо реализовать алгоритм “Игра жизнь”.

Суть алгоритма: имеется поле некоторого размера, разбитое на клетки. Каждая клетка в данный момент находится в одном из двух возможных состояний: жива или мертва. Игра делится на шаги, на каждом следующем шаге состояние клеток может меняться на основании предыдущего шага. Формальная запись правил:

**Правило 1.** Действие происходит на плоскости, разделенной на четырехугольные клетки.

**Правило 2.** Каждая клетка может находиться в двух состояниях: быть живой или быть мёртвой.

**Правило 3.** У каждой клетки 8 соседей (кроме граничных случаев, когда соседей может быть меньше).

**Правило 4.** Если клетка жива и у нее 2−3 живых соседа, то она остается живой, иначе умирает (либо от одиночества, либо от перенаселения).

**Правило 5.** Если клетка мертва и у нее 3 живых соседа, то она становится живой, иначе остается мертвой.

**Правило 6.** Игра прекращается, если на поле не останется ни одной живой клетки.

**Правило 7.** Игра прекращается, если при очередном шаге ни одна из клеток не меняет своего состояния (то есть состояние поля остается неизменным).

Для подробного ознакомления с правилами можно воспользоваться ссылками:

[Игра жизнь](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0_%C2%AB%D0%96%D0%B8%D0%B7%D0%BD%D1%8C%C2%BB), [Game\_of\_Life](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life), [Игра "Жизнь"](https://neerc.ifmo.ru/wiki/index.php?title=%D0%98%D0%B3%D1%80%D0%B0_%C2%AB%D0%96%D0%B8%D0%B7%D0%BD%D1%8C%C2%BB)

Для удобства задача разделена на пункты, которые можно реализовывать последовательно:

1. Считывать два натуральных числа и создавать поле (массив) соответствующего размера. В нем хранить состояние клеток (можно хранить состояние клетки числом (например, 0 для мертвых и 1 для живых), можно как буквы м/ж или другим способом- как сами решите).

Примечание: чтобы можно было менять поле в функциях, и не надо было каждый раз передавать его, можно объявить его вне main и пометить ключевым словом static, как показано в примере в конце задания (подробности разберем на следующем занятии, пока можно просто так модифицировать)

1. Считать натуральное число n - сколько будет живых клеток в начальном состоянии. Необходимо проверить, что n>0 и не превышает количества клеток поля, иначе вывести соответствующее предупреждение и читать заново, пока не будет введено правильное число.
2. Заполнять поле: считывать пары х-у координат живых клеток; при считывании проверять, что координаты лежат в пределах поля и введенная точка не совпадает с заданными ранее. Если эти правила нарушаются, выводить соответствующее предупреждение и считывать новую точку вместо заданной. Так что в итоге на поле должно быть ровно n живых точек. Текущее состояние поля можно хранить в массиве, созданном на шаге 1.
3. Реализовать функцию печати состояния поля.
4. Реализовать функцию для расчета нового состояния на основе старого (по правилам, описанным ранее).
5. Реализовать функцию, которая будет рассчитывать k шагов игры на основании текущего (с использованием функции, реализованной на предыдущем шаге).
6. Сделать функцию, которое будет рассчитывать все дальнейшие шаги, пока не наступит условие завершения игры (не останется живых клеток, или состояние перестанет меняться). Для этого требуется использовать цикл while с указанными условиями завершения, плюс добавить ограничение по количеству шагов в 1000 - то есть цикл останавливается либо по достижению условий, описанных выше, либо если достигнуто максимальное количество шагов.
7. Модифицировать функцию из шага 5, чтобы печаталось новое состояние (функция из шага 4) и статистика шага (сколько стало живых клеток, сколько мертвых клеток, сколько клеток поменяло свое состояние с живого на мертвое, сколько клеток поменяло свое состояние с мертвого на живое). Добавить аргумент в функцию (аргумент типа boolean), который отвечал бы, будет ли печататься эта информация после хода или нет.
8. Модифицировать функцию из шага 7, чтоб по boolean аргументу она выводила (или нет) результаты моделирования - сколько шагов было сделано и по какому из трех условий остановился цикл.
9. Реализовать хранение истории изменения состояний поля: модифицировать функцию из шага 8, чтобы после каждого шага в динамическую коллекцию, хранящую массивы состояний за предыдущие ходы, добавлялось новое состояние.

Примечание: чтобы не передавать коллекцию каждый раз в функцию, лучше создать статическое поле, по аналогии с хранением состояния поля, см. пример после условия.

1. Сделать цикл while для считывания и выполнения команд (их можно считывать и выполнять последовательно):

* Команда **step** - сделать один шаг и вывести результат (функция из шага 8);
* Команда **forever 0** или **forever 1** для расчета всех дальнейших шагов в функции из пункта 9, где 0 или 1 отвечает за необходимость вывода описания моделирования (подробностей каждого шага из 8 пункта и общей статистики из 9 пункта);
* Команда вида **5 steps 0**, которая вызывает расчет k=5 (k может быть любым натуральным числом) шагов в функции из пункта 6, где 0/1 отвечает за необходимость вывода подробностей каждого шага (в функции из пункта 8);
* Команда **history** для вывода всей истории (из коллекции из пункта 10);
* Команда **clear**, чтобы очистить историю.

Дополнительные возможности (для желающих):

1. Добавить еще один критерий остановки в пункте 7 (проверять его выполнение по истории):

**Правило 8.** Игра прекращается, если конфигурация на очередном шаге в точности повторит себя же на одном из более ранних шагов.

1. Реализовать еще одну команду вида **back 4** для отката на шаг k=4 в истории (установить текущим состояние из этого шага, если есть (если нет, то вывести соответствующее уведомление пользователю) и очистить дальнейшую историю).

**Пример начальной реализации:**

**public class** GameOfLife {

**static** Integer[][] *field*;

**static** ArrayList<Integer[][]> *history* = **new** ArrayList<>();

**static void** step(){

*// рассчитать новое состояние*

*history*.add(*field*);

}

**public static void** main(String[] args) {

**int** n = 10, m=20; *// вместо этого считывать нужные размеры поля*

*field* = **new** Integer[10][20];

*// заполнить поле*

*step*();

System.***out***.println(*history*.size());

}

}