LESSON 1

Задать целочисленный массив, состоящий из элементов 0 и 1. Например: [ 1, 1, 0, 0, 1, 0, 1, 1, 0, 0 ]. С помощью цикла и условия заменить 0 на 1, 1 на 0.

Задать пустой целочисленный массив размером 8. С помощью цикла заполнить его значениями 1, 4, 7, 10, 13, 16, 19, 22.

Создать метод, которому в качестве аргумента передается целочисленный массив. Метод должен элементы массива, которые меньше 6, умножить на 2.

Написать метод, которому в качестве параметра передается целое число. Метод должен напечатать в консоль, положительное число ему передали или отрицательное. Замечание: ноль считаем положительным числом.

Написать метод, принимающий на вход два числа и проверяющий, что их сумма лежит в пределах от 10 до 20 (включительно). Если да — вернуть true, в противном случае — false.

\* Задать одномерный массив и найти в нем минимальный и максимальный элементы (без помощи интернета).

\*\* Написать метод, в который передается непустой одномерный целочисленный массив. Метод должен вернуть true, если в массиве есть место, в котором сумма левой и правой части массива равны. Примеры: checkBalance([2, 2, 2, 1, 2, 2, || 10, 1]) → true, checkBalance([1, 1, 1, || 2, 1]) → true. Граница показана символами ||, которые в массив не входят.

\*\*\* Написать метод, которому на вход подается одномерный массив и число n (может быть положительным или отрицательным), при этом метод должен сместить все элементы массива на n позиций. Для усложнения задачи: нельзя пользоваться вспомогательными массивами.

LESSON 2

Создать классы Собака, Домашний Кот, Тигр, Животное (можете добавить два-три своих класса).

Животные могут бежать и плыть. В качестве аргумента каждому методу передается длина препятствия.

У каждого животного есть ограничения на действия (бег: кот — 200 м, собака — 500 м; плавание: кот — не умеет плавать, собака — 10 м). Результатом выполнения действия будет печать в консоль. Например: dogBobik.run(150); -> 'Бобик пробежал 150 м'.

Создать один массив с животными и заставляете их по очереди пробежать дистанцию и проплыть.

\* Добавить подсчет созданных Домашних Котов, Тигров, Собак, Животных.

LESSON 3

Продолжаем работать с участниками и выполнением действий. Создайте три класса Человек, Кот, Робот, которые не наследуются от одного класса. Эти классы должны уметь бегать и прыгать, все также с выводом информации о действии в консоль.

Создайте два класса: беговая дорожка и стена, при прохождении через которые, участники должны выполнять соответствующие действия (бежать или прыгать), результат выполнения печатаем в консоль (успешно пробежал, не смог пробежать и т.д.). У препятствий есть длина (для дорожки) или высота (для стены), а участников ограничения на бег и прыжки.

Создайте два массив: с участниками и препятствиями, и заставьте всех участников пройти этот набор препятствий. Если участник не смог пройти одно из препятствий, то дальше по списку он препятствий не идет.

LESSON 4

Создайте исключения: MyArraySizeException (неправильный размер массива), и MyArrayDataException (в ячейке массива лежит что-то не то);

Напишите метод, на вход которого подаётся двумерный строковый массив (String[][]) размером 4×4. При подаче массива другого размера необходимо бросить исключение MyArraySizeException.

Далее метод должен пройтись по всем элементам массива, преобразовать в int и просуммировать. Если в каком-то элементе массива преобразование не удалось (например, в ячейке лежит символ или текст вместо числа), должно быть брошено исключение MyArrayDataException с детализацией, в какой именно ячейке лежат неверные данные. Расчет данных для этой матрицы прекращается.

В методе main() нужно вызвать полученный метод, обработать возможные исключения MySizeArrayException и MyArrayDataException и вывести результат расчёта.

Заметка: для преобразования строки в число можно использовать метод класса Integer: Integer.valueOf("1")

LESSON 5

Создать массив с набором слов (10–20 слов, должны встречаться повторяющиеся). Найти и вывести список уникальных слов, из которых состоит массив (дубликаты не считаем). Посчитать, сколько раз встречается каждое слово.

Написать простой класс Телефонный Справочник, который хранит в себе список фамилий и телефонных номеров. В этот телефонный справочник с помощью метода add() можно добавлять записи, а с помощью метода get() искать номер телефона по фамилии. Следует учесть, что под одной фамилией может быть несколько телефонов (в случае однофамильцев), и при запросе такой фамилии должны выводиться все телефоны. Желательно не добавлять лишний функционал (дополнительные поля — имя, отчество, адрес, — взаимодействие с пользователем через консоль и т. д). Консоль использовать только для вывода результатов проверки телефонного справочника.

LESSON 6

Необходимо написать два метода, которые делают следующее:

Создают одномерный длинный массив, например:

static final int SIZE = 10 000 000;

static final int HALF = size / 2;

float[] arr = new float[size].

Заполняют этот массив единицами.

Засекают время выполнения: long a = System.currentTimeMillis().

Проходят по всему массиву и для каждой ячейки считают новое значение по формуле:

arr[i] = (float)(arr[i] \* Math.sin(0.2f + i / 5) \* Math.cos(0.2f + i / 5) \* Math.cos(0.4f + i / 2)).

Проверяется время окончания метода System.currentTimeMillis().

В консоль выводится время работы: System.out.println(System.currentTimeMillis() - a).

Отличие первого метода от второго:

первый просто бежит по массиву и вычисляет значения;

второй разбивает массив на два, в двух потоках вычисляет новые значения и потом склеивает эти массивы обратно в один.

Пример деления одного массива на два:

System.arraycopy(arr, 0, a1, 0, h);

System.arraycopy(arr, h, a2, 0, h).

Пример обратной склейки:

System.arraycopy(a1, 0, arr, 0, h);

System.arraycopy(a2, 0, arr, h, h).

Примечание:

System.arraycopy() — копирует данные из одного массива в другой:

System.arraycopy(массив-источник, откуда начинаем брать данные из массива-источника, массив-назначение, откуда начинаем записывать данные в массив-назначение, сколько ячеек копируем).

Замеры времени:

для первого метода надо считать время только на цикл расчёта:

for (int i = 0; i < size; i++) {

arr[i] = (float)(arr[i] \* Math.sin(0.2f + i / 5) \* Math.cos(0.2f + i / 5) \* Math.cos(0.4f + i / 2));

}

для второго метода нужно замерить время разбивки массива на 2, просчёта каждого из двух массивов и склейки.

LESSON 7

Написать метод, который меняет два элемента массива местами (массив может быть любого ссылочного типа).

Написать метод, который преобразует массив в ArrayList.

Задача:

Даны классы Fruit -> Apple, Orange.

Класс Box, в который можно складывать фрукты. Коробки условно сортируются по типу фрукта, поэтому в одну коробку нельзя сложить и яблоки, и апельсины.

Для хранения фруктов внутри коробки можно использовать ArrayList.

Написать метод getWeight(), который высчитывает вес коробки. Задать вес одного фрукта и их количество: вес яблока — 1.0f, апельсина — 1.5f (единицы измерения не важны).

Внутри класса Box написать метод Compare, который позволяет сравнить текущую коробку с той, которую подадут в Compare в качестве параметра. True, если их массы равны, False — в противном случае. Можно сравнивать коробки с яблоками и апельсинами.

Написать метод, который позволяет пересыпать фрукты из текущей коробки в другую. Помним про сортировку фруктов: нельзя высыпать яблоки в коробку с апельсинами. Соответственно, в текущей коробке фруктов не остаётся, а в другую перекидываются объекты, которые были в первой.

Не забываем про метод добавления фрукта в коробку.

LESSON 8

Создать базу данных с товарами (id, название, стоимость, количество).

Реализовать класс, позволяющий работать с товарами: добавлять в БД, удалять по id, искать по названию, стоимости в пределах от и до, получать список всех товаров. По одной команде на каждую операцию, плюс команда “Выход”.

Сделать консольное приложение, позволяющее выполнить все перечисленные во втором пункте операции.

LESSON 9

Реализуйте импорт списка объектов типа Cтудент (class Student с полями id, name, score) из csv-файла. В методе импорта укажите, что нужно делать, если встретилась битая строка: бросить исключение, пропустить такую строку.

Реализуйте экспорт списка объектов типа Студент в csv-файл.

Реализуйте простое шифрование/дешифрование текстового файла по ключу. Например, есть сообщение “Java message” и ключ “Key”, при шифровании мы должны получить новое сообщение путём побайтового сложения каждого символа.

Операция шифрования/дешифрования, имя файла и ключ указываются через консоль:

/encrypt 111.txt 111-encrypted.txt myKey (команда /encrypt, имя входящего файла, имя файла, в который его шифруем, ключ);

/decrypt 111-encrypted.txt 111-decrypted.txt myKey (команда /decrypt, имя входящего файла, имя файла, в который его дешифруем, ключ).

LESSON 10

Написать метод, которому в качестве аргумента передаётся непустой одномерный целочисленный массив. Метод должен вернуть новый массив, который получен путём извлечения из исходного массива элементов, идущих после последней четвёрки. Входной массив должен содержать хотя бы одну четвёрку, иначе в методе необходимо выбросить RuntimeException.

Написать набор тестов для этого метода (по 3-4 варианта входных данных).

Вх: [ 1 2 4 4 2 3 4 1 7 ] -> вых: [ 1 7 ].

Написать метод, который проверяет состав массива на наличие чисел 1 и 4. Если в массиве нет ни одной четвёрки и ни одной единицы, то метод вернёт false. Если в массиве кроме 1 и 4 есть другие числа, то метод тоже должен вернуть false. Написать набор тестов для этого метода (по 3-4 варианта входных данных).

[ 1 1 1 4 4 1 4 4 ] -> true

[ 1 1 1 1 1 1 ] -> false

[ 4 4 4 4 ] -> false

[ 1 4 4 1 1 4 3 ] -> false

[ 1 4 8 ] -> false

\* Написать тесты для задачи получения списка студентов из файлов из прошлого домашнего задания

LESSON 11

Создать класс, который может выполнять «тесты».

В качестве тестов выступают классы с наборами методов, снабжённых аннотациями @Test. Для этого у них должен быть статический метод start(), которому в качестве параметра передаётся объект типа Class или имя класса. Из «класса-теста» вначале должен быть запущен метод с аннотацией @BeforeSuite, если он присутствует. Далее запускаются методы с аннотациями @Test, а по завершении всех тестов — метод с аннотацией @AfterSuite.

К каждому тесту необходимо добавить приоритеты (int-числа от 1 до 10), в соответствии с которыми будет выбираться порядок их выполнения. Если приоритет одинаковый, то порядок не имеет значения. Методы с аннотациями @BeforeSuite и @AfterSuite должны присутствовать в единственном экземпляре. Если это не так, необходимо бросить RuntimeException при запуске «тестирования».

P. S. Это практическое задание пишется с нуля. Данная задача никак не связана с темой тестирования через JUnit.