ГУАП

ФАКУЛЬТЕТ ДОПОЛНИТЕЛЬНОГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | С. Ю. Гуков |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ |
| ЗАДАНИЕ ПО ТЕМЕ: КЛАССЫ И МЕТОДЫ |
| по курсу: ВВЕДЕНИЕ В РАЗРАБОТКУ КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ НА JAVA |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4941 |  |  |  | Н. С. Горбунов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Задание:**

В банке решили освоить новую нишу, а именно открыть внутренний стартап по продаже доменов. Каждый домен представляет из себя строку, состоящую только из букв S, T, A, R. Каждому покупателю хочется не просто продать строку, а продать нужную строку, а именно, пусть у каждого покупателя есть строки P и Q. Тогда ему подходят только строки, у которых первые |P| символов совпадают с P и последние |Q| символов совпадают с Q. Для начала было закуплено n доменов для m покупателей. Теперь хочется понять, хватит ли их для всех покупателей. Чтобы получить ответ на этот вопрос, посчитайте число подходящих для каждого покупателя строк.

**Формат входных данных**

Первая строка содержит два числа n и m – количество закупленных доменов и количество покупателей. Следующие n строк содержат закупленные домены. В каждой строке встречаются только символы S, T, A, R. Последние m строк содержат запросы покупателей. Каждый запрос задается двумя строками Pi и Qi, состоящими из вышеупомянутых символов.

**Формат выходных данных**

Выведите m строк, состоящих из одного целого числа – количества подходящих строк для данных покупателей.

**Решение:**

*Solution.java*

**import** **java.util.Scanner**;

**public** **class** **Solution** {

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

Scanner input = **new** Scanner(System.in);

String INPUT = input.nextLine();

int \_DOM\_COUNT = Integer.parseInt(INPUT.split(" ")[0]);

int \_PATTERN\_COUNT = Integer.parseInt(INPUT.split(" ")[1]);

String[] DOMAINS = **new** String[\_DOM\_COUNT];

String[] PATTERNS = **new** String[\_PATTERN\_COUNT];

**for** (int i = 0; i < \_DOM\_COUNT; i++) {

DOMAINS[i] = input.nextLine();

}

**for** (int i = 0; i < \_PATTERN\_COUNT; i++) {

PATTERNS[i] = input.nextLine();

}

System.out.print("Choose your pollution: ");

int type = input.nextInt();

Pollution pollution = Pollution.getByType(type);

System.out.println("Pollution by " + pollution.name());

pollution.execute(DOMAINS, PATTERNS);

}

}

*Pollution.java*

**import** **java.util.Arrays**;

**public** **enum** Pollution {

GUKOV(1){

@Override

**public** void execute(String[] DOMAINS, String[] PQ\_STRINGS) {

**for** (int PATTERN\_INDEX = 0; PATTERN\_INDEX < PQ\_STRINGS.length; PATTERN\_INDEX++) {

int RESULT = 0;

**for** (int DOMAIN\_INDEX = 0; DOMAIN\_INDEX < DOMAINS.length; DOMAIN\_INDEX++) {

**if** (DOMAINS[DOMAIN\_INDEX].indexOf(PQ\_STRINGS[PATTERN\_INDEX].split(" ")[0]) == 0 &&

DOMAINS[DOMAIN\_INDEX].indexOf(PQ\_STRINGS[PATTERN\_INDEX]

.split(" ")[1]) == (DOMAINS[DOMAIN\_INDEX]

.length() - PQ\_STRINGS[PATTERN\_INDEX]

.split(" ")[1]

.length())) {

RESULT++;

}

}

System.out.println(RESULT);

}

}

},

GOLOSCHAPOV(2) {

@Override

**public** void execute(String[] DOMAINS, String[] PQ\_STRINGS) {

**for** (int PATTERN\_INDEX = 0; PATTERN\_INDEX < PQ\_STRINGS.length; PATTERN\_INDEX++) {

int RESULT = 0;

**for** (int DOMAIN\_INDEX = 0; DOMAIN\_INDEX < DOMAINS.length; DOMAIN\_INDEX++) {

String[] PQ = PQ\_STRINGS[PATTERN\_INDEX].split(" ");

**if** (DOMAINS[DOMAIN\_INDEX].startsWith(PQ[0]) &&

DOMAINS[DOMAIN\_INDEX].endsWith(PQ[1])){

RESULT++;

}

}

System.out.println(RESULT);

}

}

},

GORBUNOV(3) {

@Override

**public** void execute(String[] DOMAINS, String[] PQ\_STRINGS) {

**for** (int PATTERN\_INDEX = 0; PATTERN\_INDEX < PQ\_STRINGS.length; PATTERN\_INDEX++) {

int RESULT = 0;

**for** (int DOMAIN\_INDEX = 0; DOMAIN\_INDEX < DOMAINS.length; DOMAIN\_INDEX++) {

String[] PQ = PQ\_STRINGS[PATTERN\_INDEX].split(" ");

**if** (DOMAINS[DOMAIN\_INDEX].substring(0, PQ[0].length()).equals(PQ[0])

&& DOMAINS[DOMAIN\_INDEX].substring(DOMAINS[DOMAIN\_INDEX].length() - PQ[1].length(), DOMAINS[DOMAIN\_INDEX].length()).equals(PQ[1])) {

RESULT++;

}

}

System.out.println(RESULT);

}

}

};

**private** **final** int type;

**public** **abstract** void execute(String[] DOMAINS, String[] PQ\_STRINGS);

Pollution(int type) {

**this**.type = type;

}

**public** **static** Pollution getByType(int type){

**return** Arrays.stream(values())

.filter(o-> o.type==type)

.findFirst()

.orElseThrow(() -> **new** RuntimeException("Not implement"));

}

}