**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина

Институт радиоэлектроники и информационных технологий – РТФ

Центр ускоренного обучения

**ОТЧЕТ ПО ЗАДАНИЯМ С САЙТА TIMUS**

**По дисциплине «Программирование»**

Студент группы РИЗ-140938у Бабенко М.А.

Преподаватель: Архипов Н.А.

**Екатеринбург, 2025 г.**

**Решение заданий с сайта Tumus**

**Ссылка на репозиторий GitHub:** https://github.com/sverdlovsky00/LW\_0\_7

**Задание 1**. Папа у Васи силён в математике. В последнее время папа заинтересовался такими объектами, как «красивые» ориентированные графы. «Красивым» он называет ориентированный граф, удовлетворяющий следующим условиям:

1. Граф содержит ровно N узлов и N − 1 дугу.

2. Ровно у одной вершины графа нет ни одной входящей дуги.

3. Граф не содержит ориентированных циклов.

Папа говорит, что два «красивых» графа изоморфны, если можно перенумеровать вершины первого графа таким образом, чтобы получился второй граф.

Папа выбирает целое число N, запасается чистой бумагой и рисует на каждом листе ровно по одному «красивому» графу. При этом он следит, чтобы никакие два из нарисованных им графов не были изоморфны.

Зная число N, найдите, каким количеством листов бумаги должен предусмотрительно запастись Васин папа.

Массив dp хранит количество "красивых" графов для каждого числа вершин от 1 до N. [1] = 1, так как граф из одной вершины всегда считается красивым. Для каждой вершины i от 2 до N мы перебираем все возможные предыдущие размеры графов j (от 1 до i-1) и добавляем их к текущему значению dp[i].

После заполнения массива dp выводится значение dp[N] – количество "красивых" графов для N вершин.. Результат работы программы и ее программный код представлен на рисунке 1.

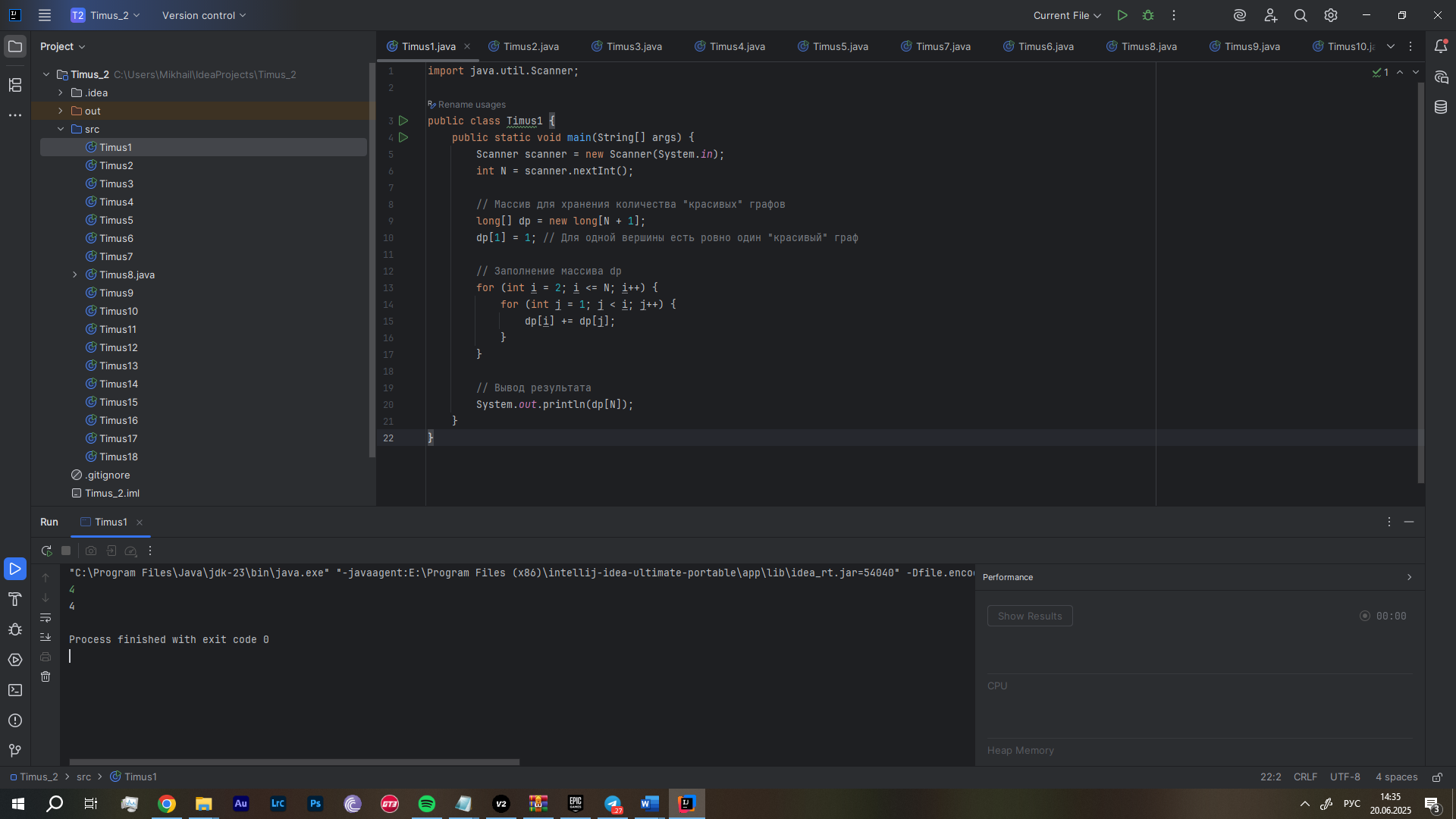


Рисунок 1 – Код и результат программы Timus1

**Задание 1044**. На шахматной доске стоят белый слон и чёрная пешка. Ходы делаются в соответствии с общепринятыми шахматными правилами. Белые ходят первыми. Чёрные побеждают, если они смогли провести свою пешку в ферзи и следующим своим ходом белый слон не может уничтожить ферзя. Партия заканчивается вничью, если очередь хода за чёрными, но пешка не может продвинуться вперед. В остальных случаях выигрывают белые. Требуется выяснить исход партии при наилучшей игре с обеих сторон.

Две строки в шахматной нотации (например, a1 и b2), обозначающие позиции слона и пешки. Преобразовываются координаты: **bishopX = bishopPosition.charAt(0) -** 'a' – переводит букву (a-h) в число (0-7); **bishopY = bishopPosition.charAt(1)** - '1' — переводит цифру (1-8) в число (0-7). Аналогично для пешки (pawnX, pawnY).

Если пешка находится на той же горизонтали (**bishopY == pawnY**) или вертикали (**bishopX == pawnX**), то слон может её съесть, и выводится "**WHITE**"; в противном случае **(else) – "DRAW"** (ничья).

Результат работы программы и ее программный код представлен на рисунке 2.

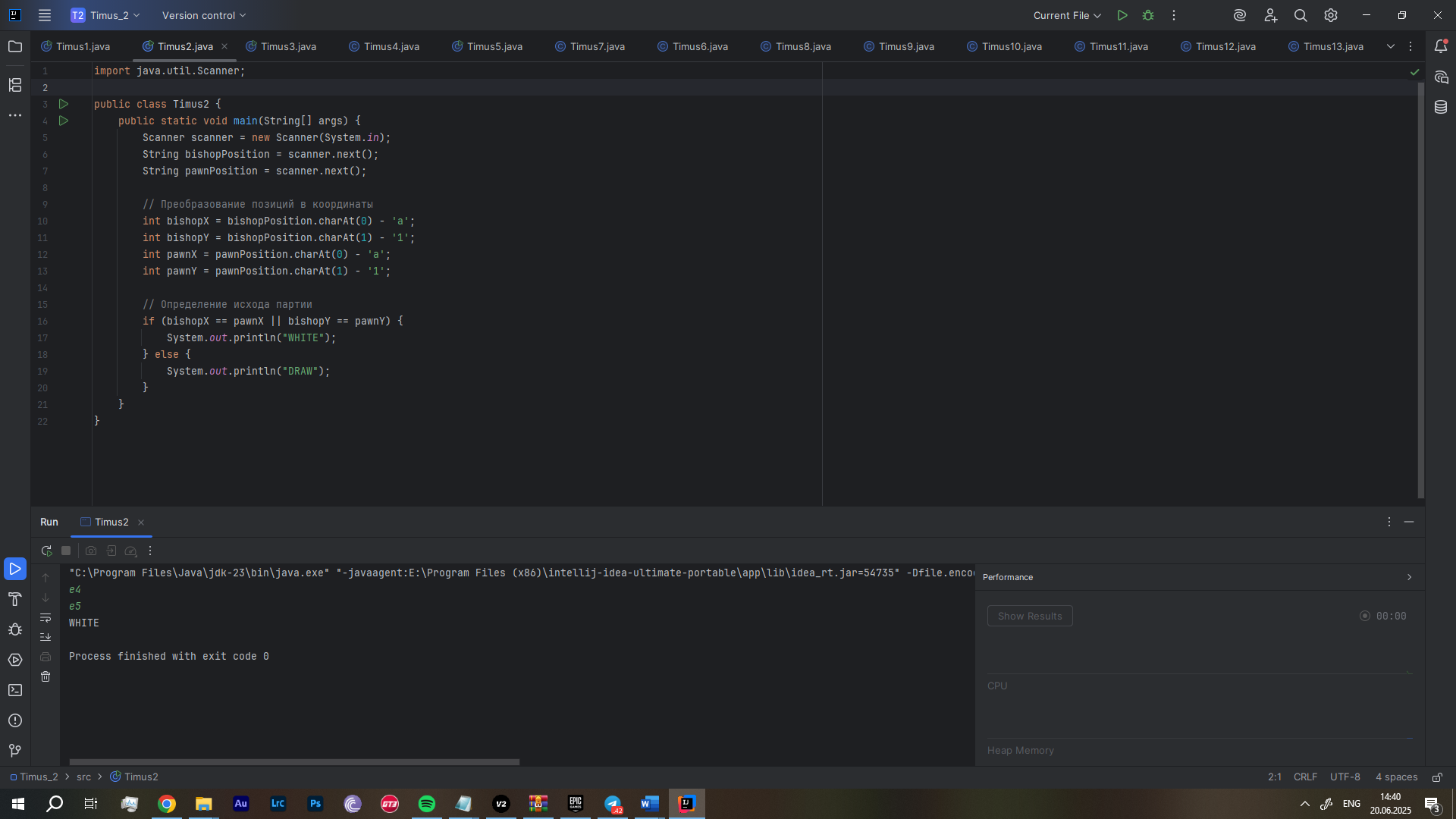


Рисунок 2 – Код и результат программы Timus2

**Задание 3**. Гондор — страна холмов. В незапамятные времена владыки Гондора решили построить на некоторых холмах сторожевые башни для охраны королевства. После подсчёта имеющихся в казне средств выяснилось, что их хватает только на постройку пяти башен. Король Гондо-ра развернул карту королевства с отмеченными на ней N холмами. Чтобы стражи на башнях могли эффективно обозревать местность, башни должны были располагаться в вершинах некоторого выпуклого пятиугольника. Мудрый Гэндальф, состоящий в дружбе с королём, нашёл решение этой задачи. И вот стражи на башнях не смыкают глаз в течение уже тысячи лет…Интересно, какие бы пять холмов для постройки башен выбрали вы на месте Гэндальфа?

Класс Point хранит координаты x, y и индекс точки (для вывода). Для построения выпуклой оболочки точки сортируются по возрастанию x (при равенстве — по y). Строится нижняя часть оболочки (алгоритм Эндрю); а затем строится верхняя часть оболочки, удаляется последняя точка (так как она дублируется).

**Orientation** проверяет, образуют ли три точки левый поворот, правый или лежат на одной прямой (через векторное произведение). Если в оболочке ≥5 точек, выводятся их индексы, иначе – "No".

Результат работы программы и ее программный код представлен на рисунке 3.

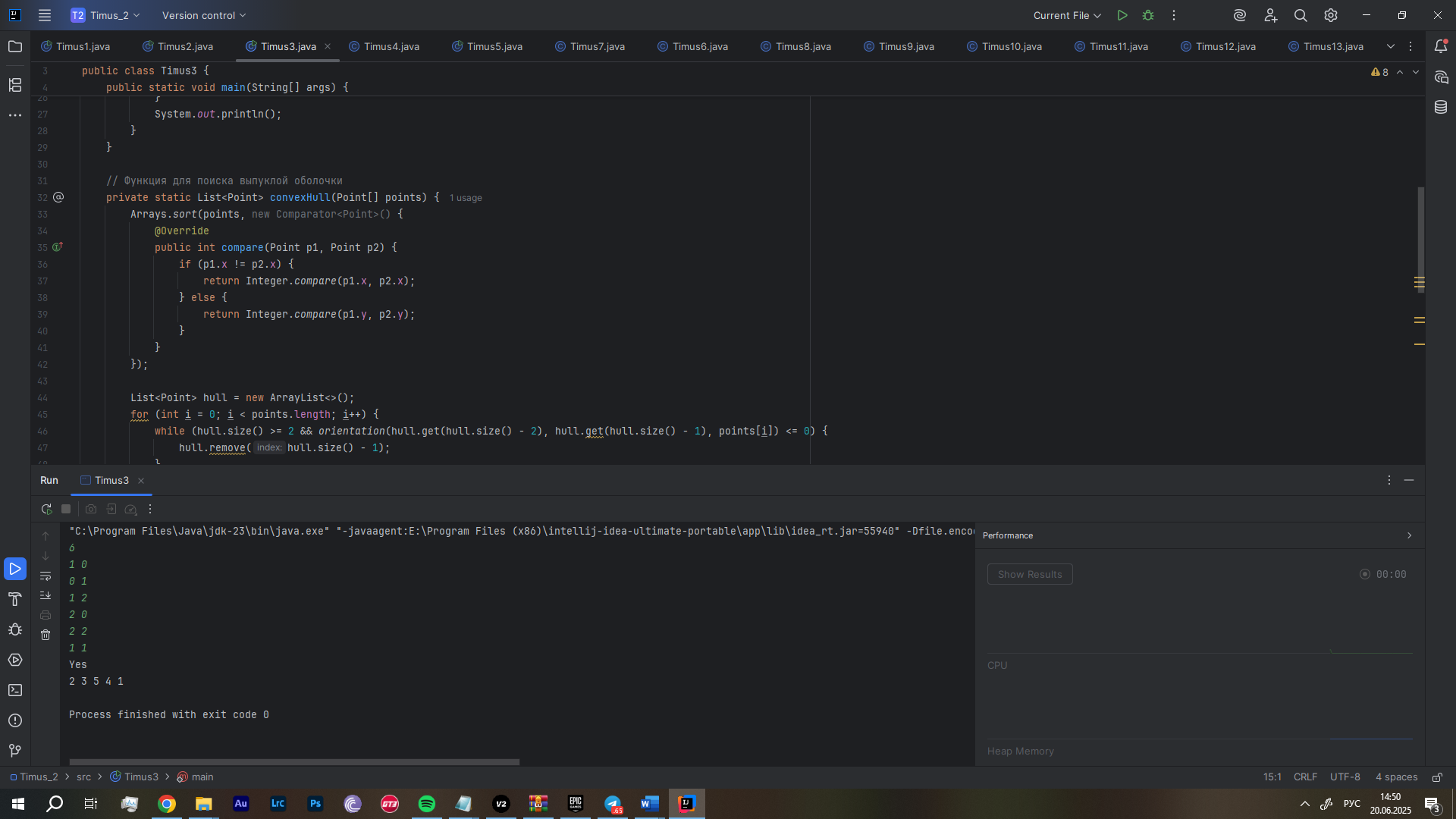


Рисунок 3 – Код и результат программы Timus3

**Задание 4.** Грядут очередные выборы. Снова все заборы оклеены листовками, почтовые ящики забиты макулатурой, с экранов телевизоров на нас взирают мордатые дядьки, обещающие сделать нашу жизнь лучше… А программист Васечкин снова завален работой. Необходимо написать программу, которая облегчит подсчет голосов избирателей.

Пользователь вводит с клавиатуры N – количество кандидатов и M – количество избирателей. Массив votes, где votes[i] – число голосов за кандидата i+1.

Для каждого кандидата i вычисляется процент голосов: **double percentage = (votes[i] \* 100.0) / M;** и выводится в консоль с округлением до двух знаков **(System.out.printf)**.

Результат работы программы и ее код показан на рисунке 4.

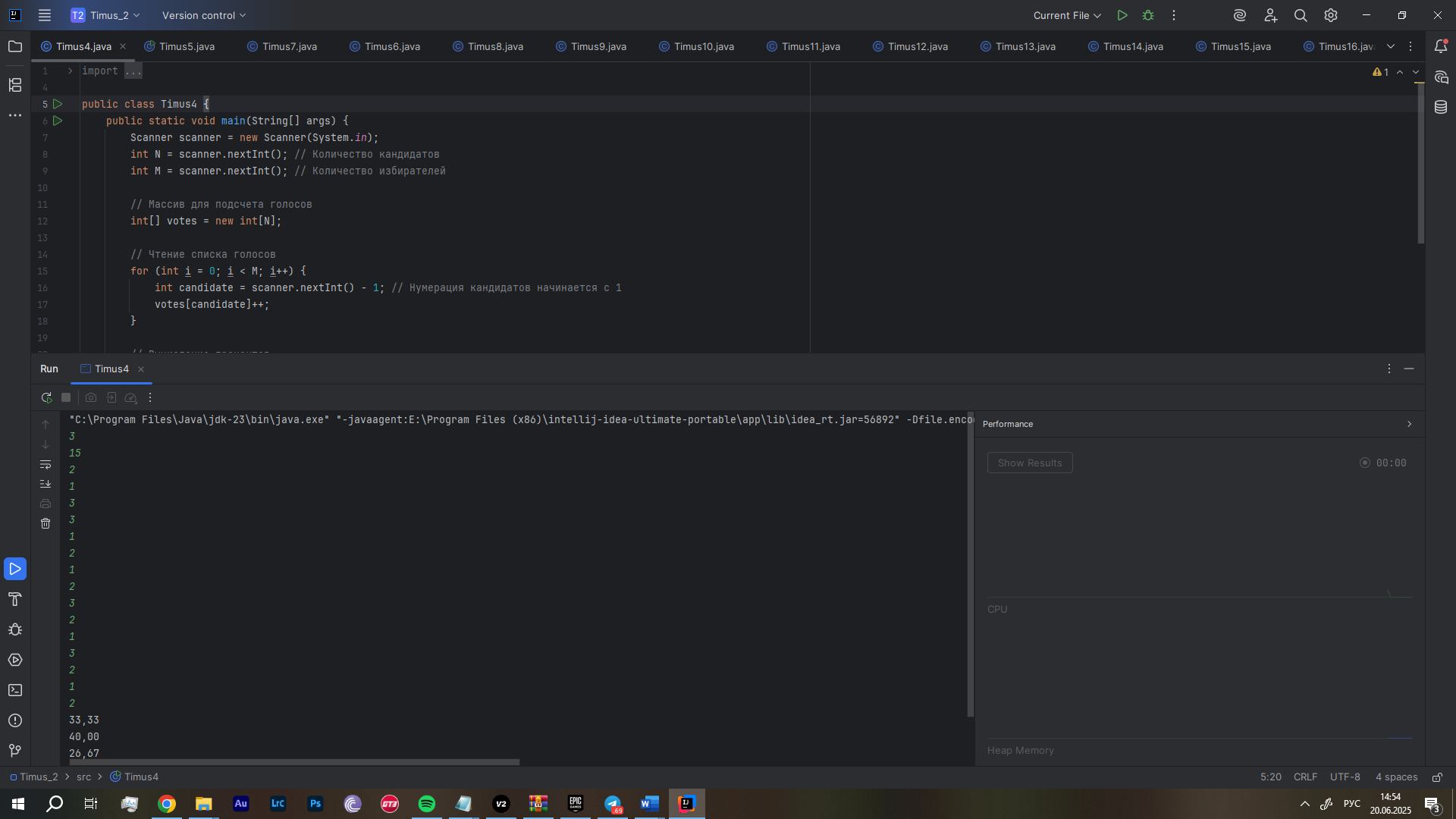


Рисунок 4 – Код и результат работы программы Timus4

**Задание 5.** В компьютерном клубе N компьютеров. Их надо соединить в сеть. Известно, какие компьютеры нужно соединить проводами для правильной работы сети. Осталось расставить компьютеры так, чтобы никакие два провода не пересекались, и расстояние между любыми двумя компьютерами было больше единицы. Компьютеры можно считать точками, провода — отрезками. Сеть является связной, т.е. любые два компьютера соединены некоторой последовательностью проводов.

Программа генерирует координаты для N компьютеров, расположенных на прямой линии (планарный граф). Координаты: **x = i \* 2** (чтобы компьютеры не накладывались). **y = 0** (все на одной линии). Координаты каждого компьютера выводятся в консоль.

Результат работы программы и ее код показан на рисунке 5.

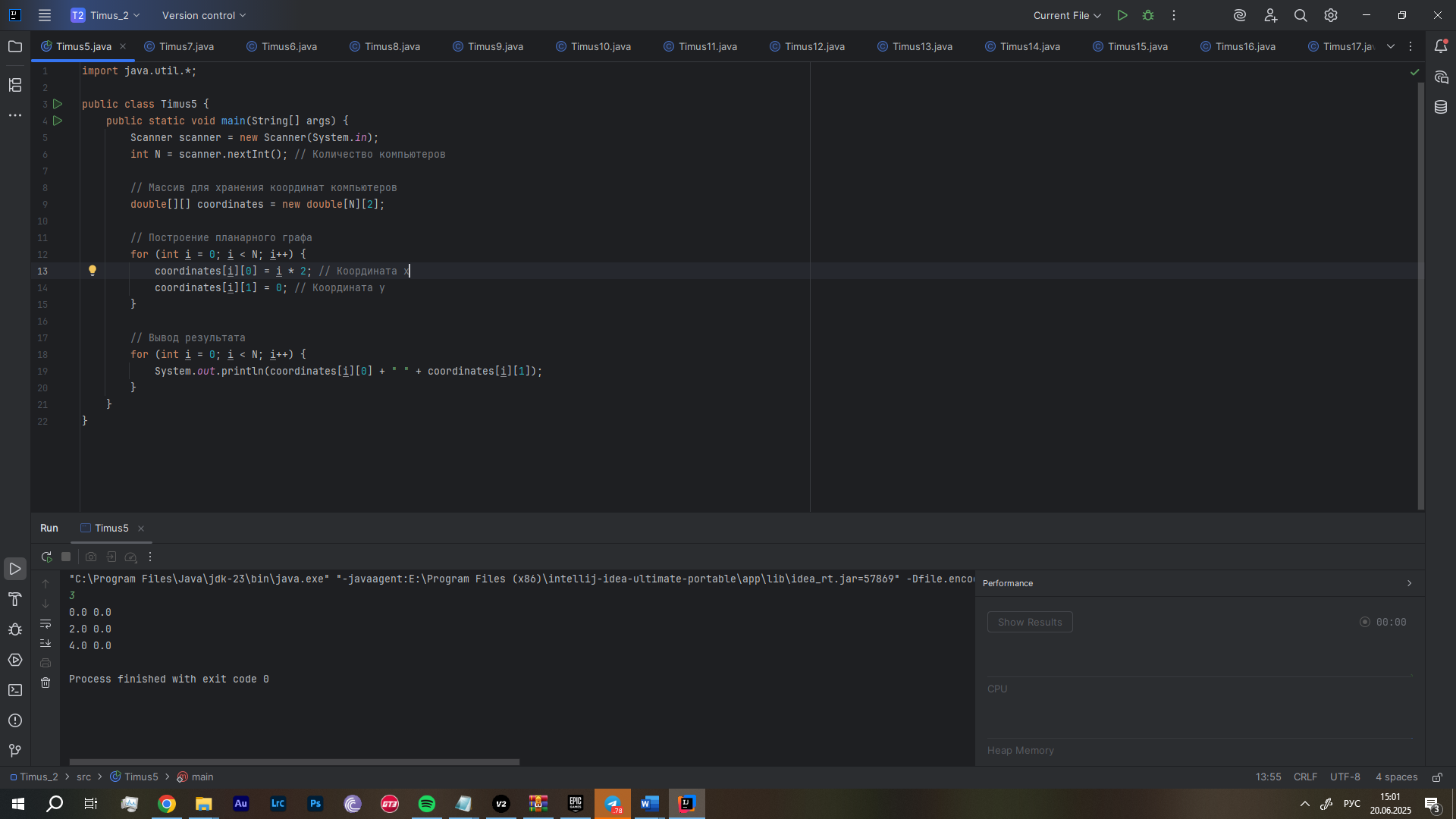


Рисунок 5 – Код и результат работы программы Timus5

**Задание 6**. У Никифора есть некоторое положительное целое число, содержащее в своей десятичной записи каждую из цифр 1, 2, 3, 4. Он просит вас переставить цифры этого числа таким образом, чтобы получившееся число делилось на семь.

Программа ищет перестановку цифр числа, которая делится на 7. Из строки **numberStr** извлекаются цифры 1, 2, 3, 4. Затем, генерируются все возможные перестановки **(generatePermutations)**. Проверяется, делится ли каждая перестановка на 7 **(num % 7 == 0)**. Если найдена подходящая перестановка — она выводится, иначе — "0".

Результат работы программы и ее программный код представлен на рисунке 6.

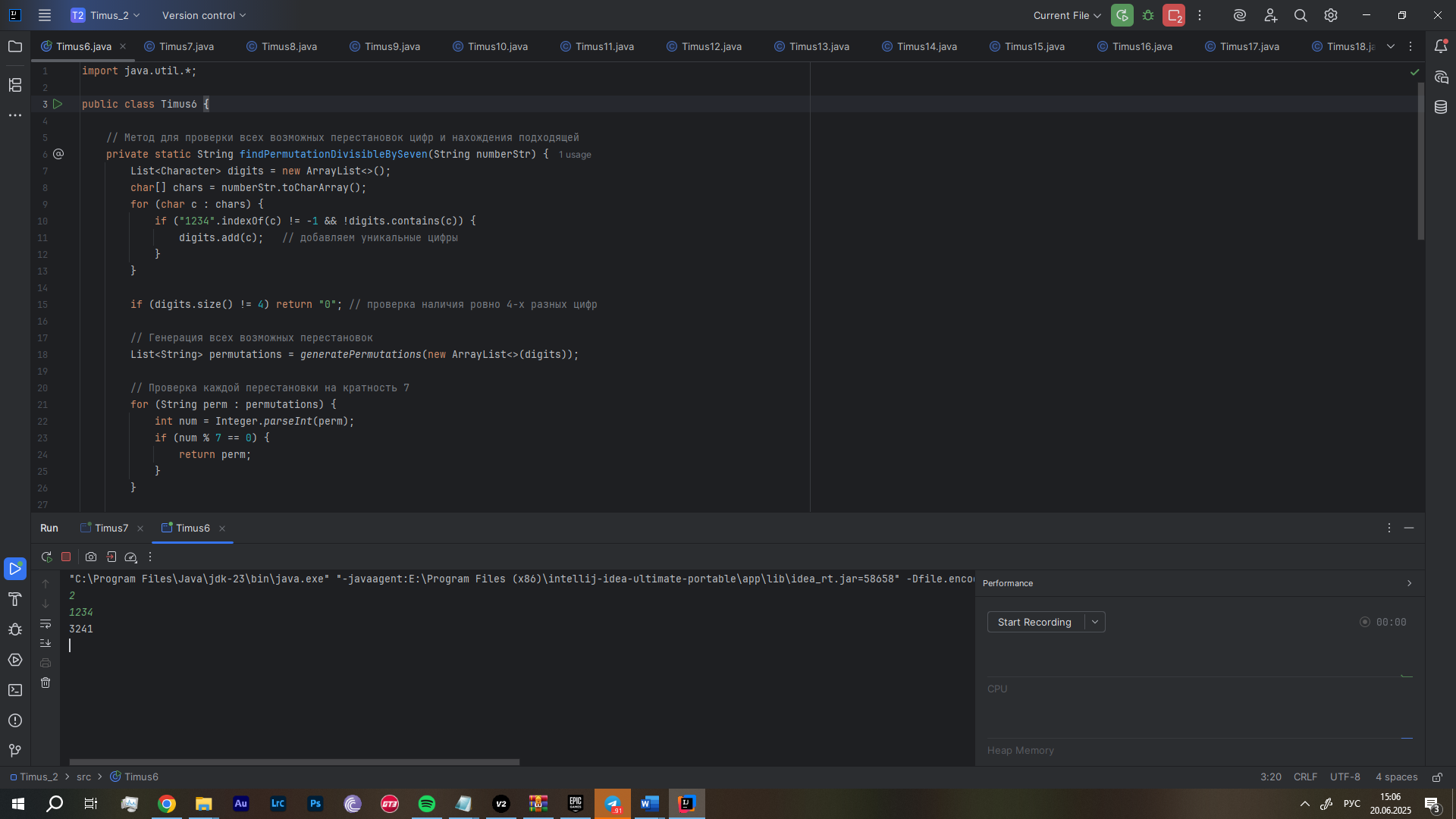


Рисунок 6 – Код и результат программы Timus6

**Задание 7**. Программист Денис c детства мечтал побывать в Антарктиде, но почему-то регулярных рейсов туда нет. Поэтому Денис все лето изучал Антарктиду с помощью соседнего кинотеатра. Теперь он знает, что в Антарктиде водится несколько видов пингвинов:

1. Императорские пингвины (Emperor Penguins) — любители петь;

2. Малые пингвины (Little Penguins) — любители потанцевать;

3. Пингвины Макарони (Macaroni Penguins) — любители сёрфинга.

К сожалению, в мультфильмах не было сказано, какой вид пингвинов самый многочисленный. Денис решил выяснить это: он посмотрел эти мультфильмы еще раз, и каждый раз, когда видел пингвина, записывал в блокнот название его вида. Сейчас Денис дал вам блокнот с просьбой выяснить, какой вид пингвинов самый многочисленный.

Сначала пользователь вводит число **n** – количество пингвинов в списке. Затем считывается **n** строк, каждая из которых содержит название пингвина (например, "Emperor Penguin", "Little Penguin" и т. д.). Для хранения количества вхождений каждого названия используется **HashMap<String, Integer>** (название → количество).

Если название уже есть в HashMap, его счётчик увеличивается на 1; если названия нет, оно добавляется в **HashMap** со значением 1. Для поиска самого популярного названия инициализируются переменные: **mostPopularPenguin** и **maxCount**. Проходим по всем записям в **HashMap** и сравниваем значения с **maxCount**. Если находим название с большим количеством вхождений, обновляем **mostPopularPenguin** и **maxCount**.

Программа выводит в консоль название пингвина, которое встречалось чаще всего.

Результат работы программы и ее программный код представлен на рисунке 7.

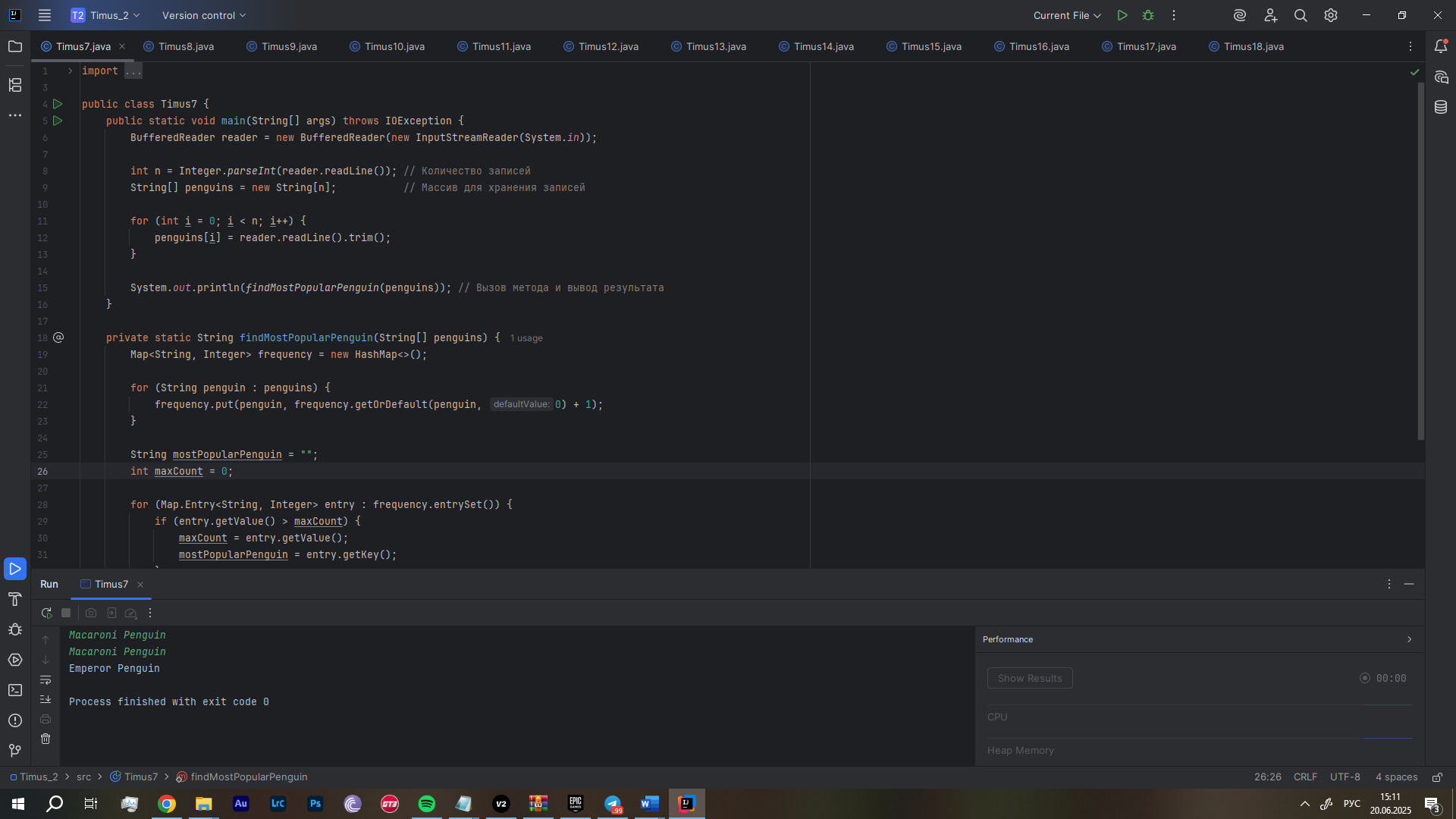


Рисунок 7 – Код и результат программы Timus7

**Задание 8.** Программисту Андрею очень повезло: генерал-губернатор Ямайки предложил ему бесплатно отдохнуть и поразвлечься на его знаменитом острове в обмен на небольшую услугу. Дело в том, что правительство Ямайки решило построить сеть скоростных дорог, устроенную таким образом, чтобы любые два города связывала дорога, проложенная строго по прямой линии. При этом одна дорога может связывать несколько городов, если они расположены на одной прямой.

Сначала пользователем вводится число n – количество городов, затем – координаты каждого города (пары чисел x и y). Для поиска точек для каждой пары городов определяется прямая, проходящая через них; проверяется, какие ещё города лежат на этой прямой **(isCollinear)**.

Все такие города группируются в список и добавляются в множество **HashSet<List<Point>>** (чтобы избежать дублирования линий).

Для каждой группы коллинеарных городов сортируются города по координате x (или y, если x совпадают), суммируются расстояния между соседними городами в отсортированном списке **(distance)**, и общая длина дорог округляется до целого числа.

После выполнения программа выводит суммарную длину всех дорог.

Результат работы программы и ее код показан на рисунке 8.

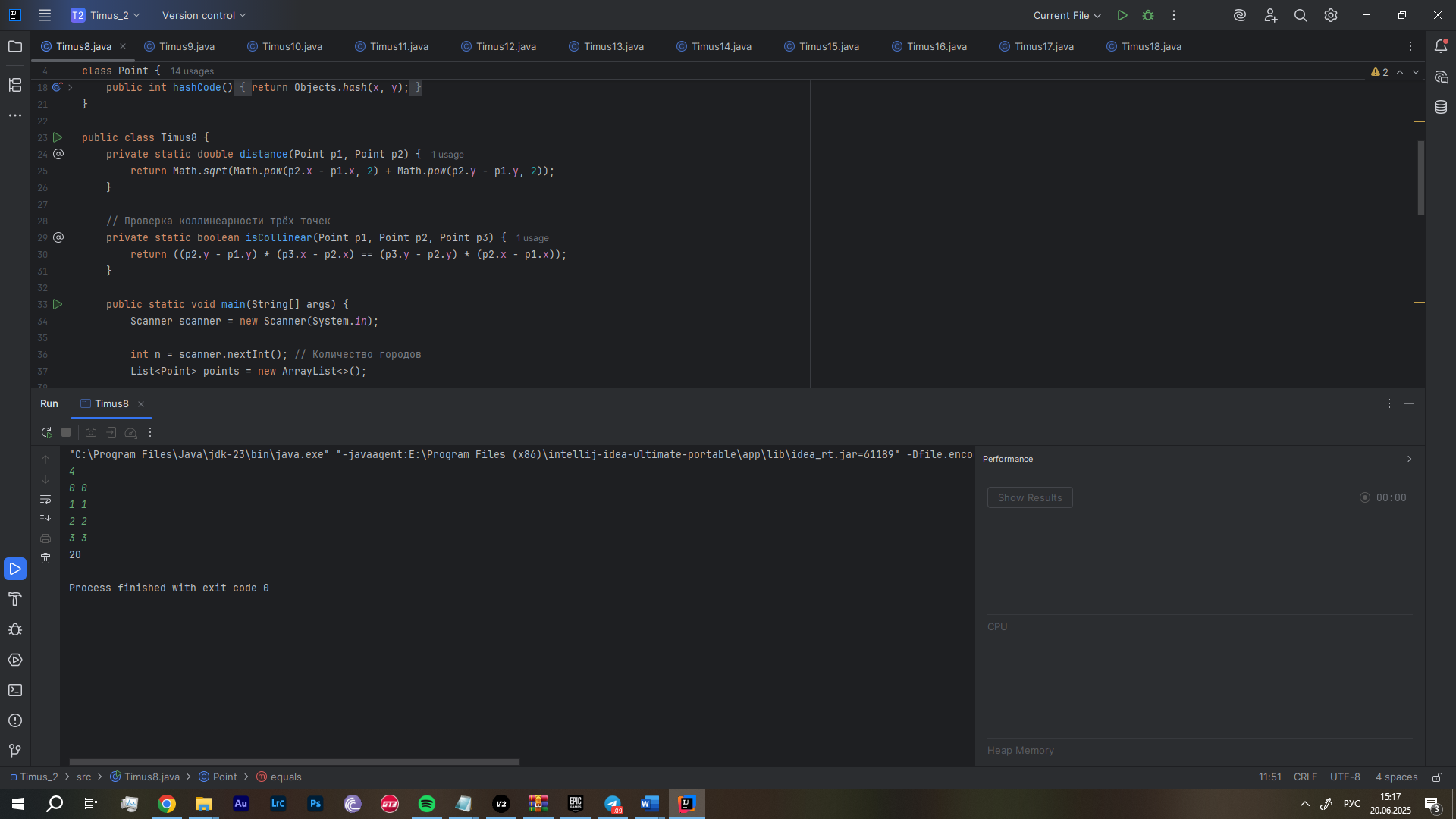


Рисунок 8 – Код и результат работы программы Timus8

**Задание 9.** Петя любит ездить на такси. Для него это не только удовольствие от быстрой и комфортной поездки, но и возможность всласть поторговаться с таксистом о цене проезда. Торг между Петей и таксистами всегда происходит по одному и тому же сценарию:

— Мне в аэропорт, заплачу 150 рублей.

— Нет, за 150 не повезу, поехали за 1000.

— Да ты что?! У меня и нет столько! Согласен за 200.

— Ты смеешься? Давай хотя бы за 900.

— Ну хорошо, дам 250.

— Да ты хоть знаешь, сколько бензин стоит? 800 и поехали!

…

Этот диалог продолжается до тех пор, пока они не сойдутся в цене. Петя всегда увеличивает свою цену на одну и ту же сумму, а таксист таким же образом снижает ее. Таксист не станет называть цену ниже той, которую предложит Петя. В этом случае он согласится с его ценой. Аналогичным образом поступит и Петя.

Вводятся четыре числа: a, b, c, d (начальные цены и шаги изменения), в цикле проверяется условие: если текущая цена таксиста (c) больше цены Пети (a), плюс его шаг (b), и при этом цена Пети не превышает цену таксиста, минус его шаг (d), торги продолжаются.

На каждом шаге Петя увеличивает цену: a += b, а таксист уменьшает цену: c -= d. Цикл завершается, когда условие нарушается. Если цена Пети (a) стала больше или равна цене таксиста (c), итоговая цена – c, иначе – a. Программа выводит итоговую цену.

Результат работы программы и ее программный код представлен на рисунке 9.

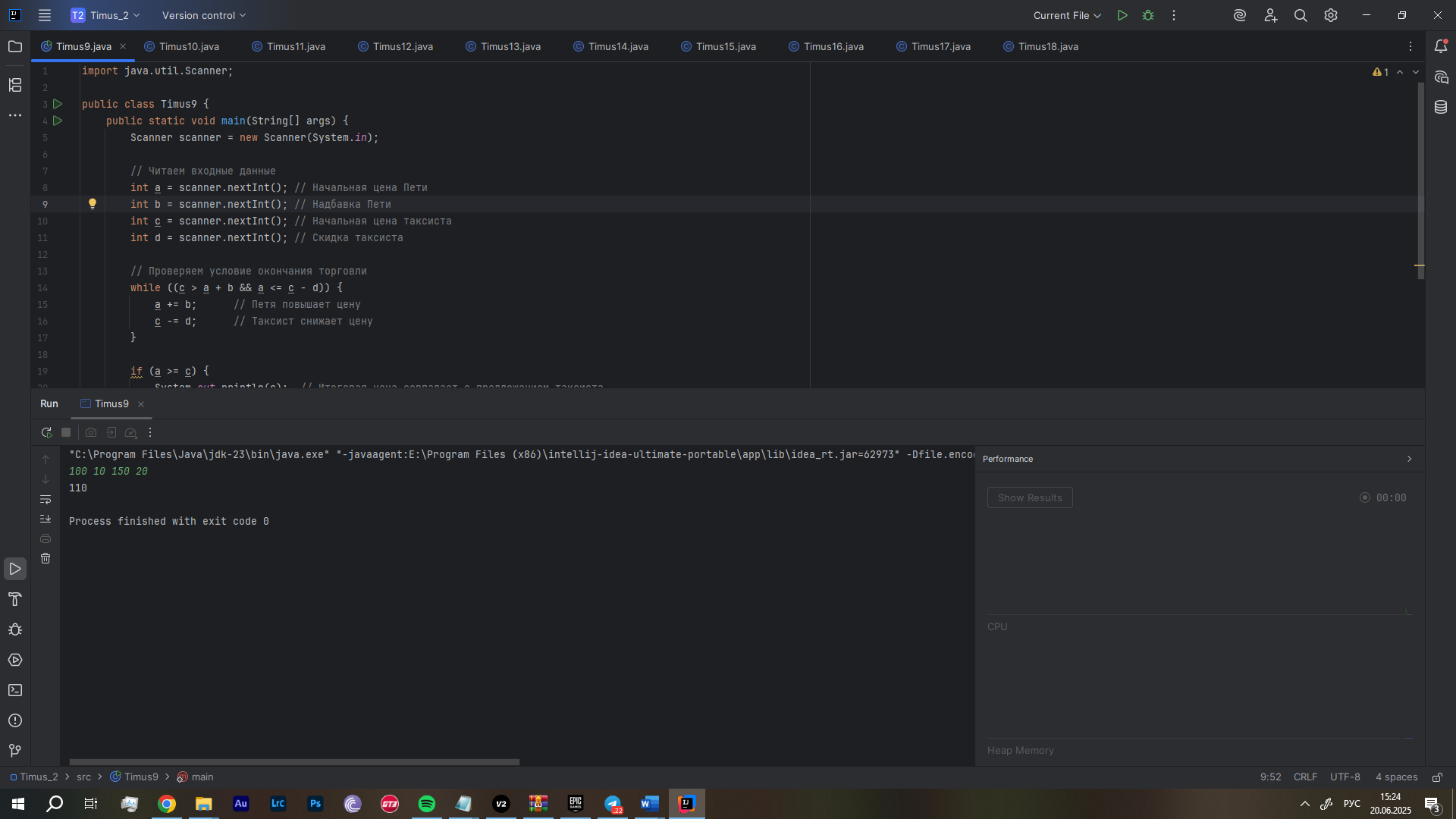


Рисунок 9 – Код и результат программы Timus9

**Задание 10**. Марсопрыг — это новая, усовершенствованная модель лунохода. Умники из конструктор-ского бюро решили, что в условиях низкой гравитации прыгать выгоднее, чем ходить. Кста-ти, прыгает он всегда одинаково, ровно на один метр. Для дистанционного управления эти умники приспособили обычную клавиатуру компьютера, точнее, дополнительные цифро-вые клавиши, те самые, которые справа. Это очень удобно и привычно: цифра 8 значит прыжок на север, 2 — на юг, 6 — на восток, 7 — прыжок на северо-запад, и так далее. Циф-ра 5 означает команду на взятие пробы грунта. Еще осталась неиспользованной цифра 0, и главный конструктор посоветовал назначить на эту клавишу команду для самоуничтожения марсопрыга, так как это очень полезная функция.

Ваша задача — подсказать поисковой команде, где может находиться этот злополучный прибор. Теоретически он должен находиться в той точке, где впервые была нажата клавиша 0, или (если эту клавишу не нажимали) в той точке, куда он допрыгал к концу испытаний. Вам будет дана последовательность нажатий клавиш. Выведите конечное местоположение марсопрыга. Испытательный полигон считать бесконечной плоскостью.

Сначала пользователем вводится строка, содержащая последовательность команд. Инициализируются координаты **x** и **y** (тип BigDecimal для точности). В зависимости от цифры изменяются координаты: 8 – север (y += 1). 2 – юг (y -= 1). 6 – восток (x += 1), и т. д. Если встречается 0, движение прекращается.

Координаты выводятся с точностью до 10 знаков после запятой.

Результат работы программы и ее программный код представлен на рисунке 10.



Рисунок 10 – Код и результат программы Timus10

**Задание 11**. Рассмотрим квадратную таблицу размером (2N + 1) × (2N + 1), в каждой из ячеек которой стоит знак «+» или «-». Трансверсаль – это произвольное множество из 2N + 1 ячейки, такое, что каждая строка и каждый столбец таблицы содержат ровно одну ячейку, принадлежащую этому множеству.

Рассмотрим операцию, состоящую в изменении всех знаков на противоположные во всех ячейках некоторой трансверсали. Вам нужно определить, можно ли с помощью последовательности таких операций получить таблицу, содержащую не более 2N ячеек со знаком «+».

Вводятся число **n** (количество точек) и координаты каждой точки (пары чисел **x, y**). Для каждой пары точек вычисляется направляющий вектор (**dx,** **dy**). Для каждой третьей точки проверяется, лежит ли она на той же прямой (условие коллинеарности через векторное произведение), запоминается максимальное количество точек, удовлетворяющих условию.

Программа выводит максимальное число коллинеарных точек.

Результат работы программы и ее программный код представлен на рисунке 11.

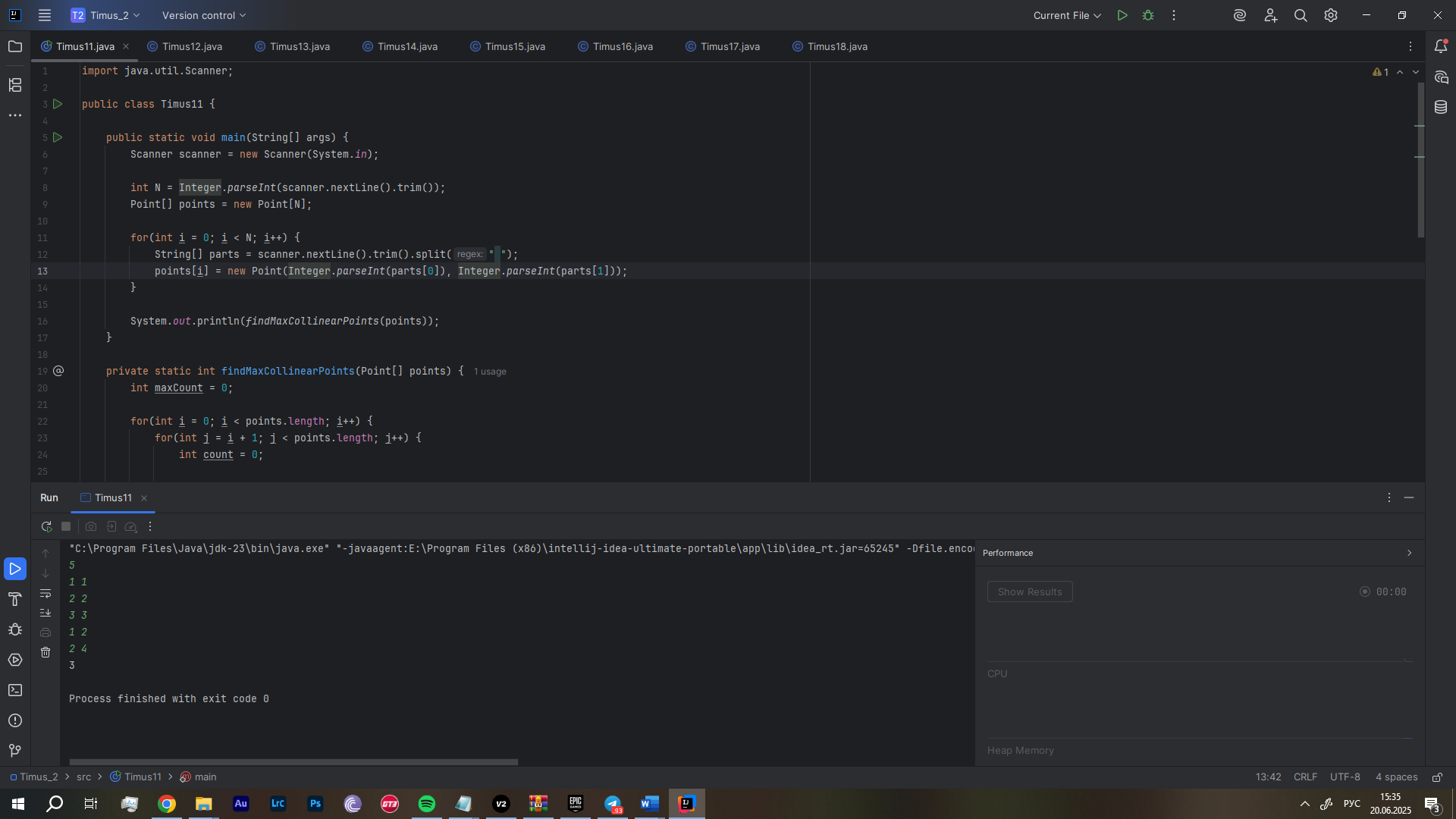


Рисунок 11 – Код и результат программы Timus11

**Задание 12.** Рассмотрим квадратную таблицу размером (2N + 1) × (2N + 1), в каждой из ячеек которой стоит знак «+» или «-». Трансверсаль – это произвольное множество из 2N + 1 ячейки, такое, что каждая строка и каждый столбец таблицы содержат ровно одну ячейку, принадлежащую этому множеству.

Рассмотрим операцию, состоящую в изменении всех знаков на противоположные во всех ячейках некоторой трансверсали. Вам нужно определить, можно ли с помощью последовательности таких операций получить таблицу, содержащую не более 2N ячеек со знаком «+».

Пользователь вводит с клавиатуры число **n**, которое определяет матрицу размером (2n + 1) × (2n + 1). Подсчитывается общее количество '+' в матрице: если их **≤ 2n**, выводится «There is solution:», иначе предпринимаются попытки уменьшить количество '+' путём переворота строк и столбцов. Если решение не найдено – выводится «No solution».

Результат работы программы и ее код показан на рисунке 12.

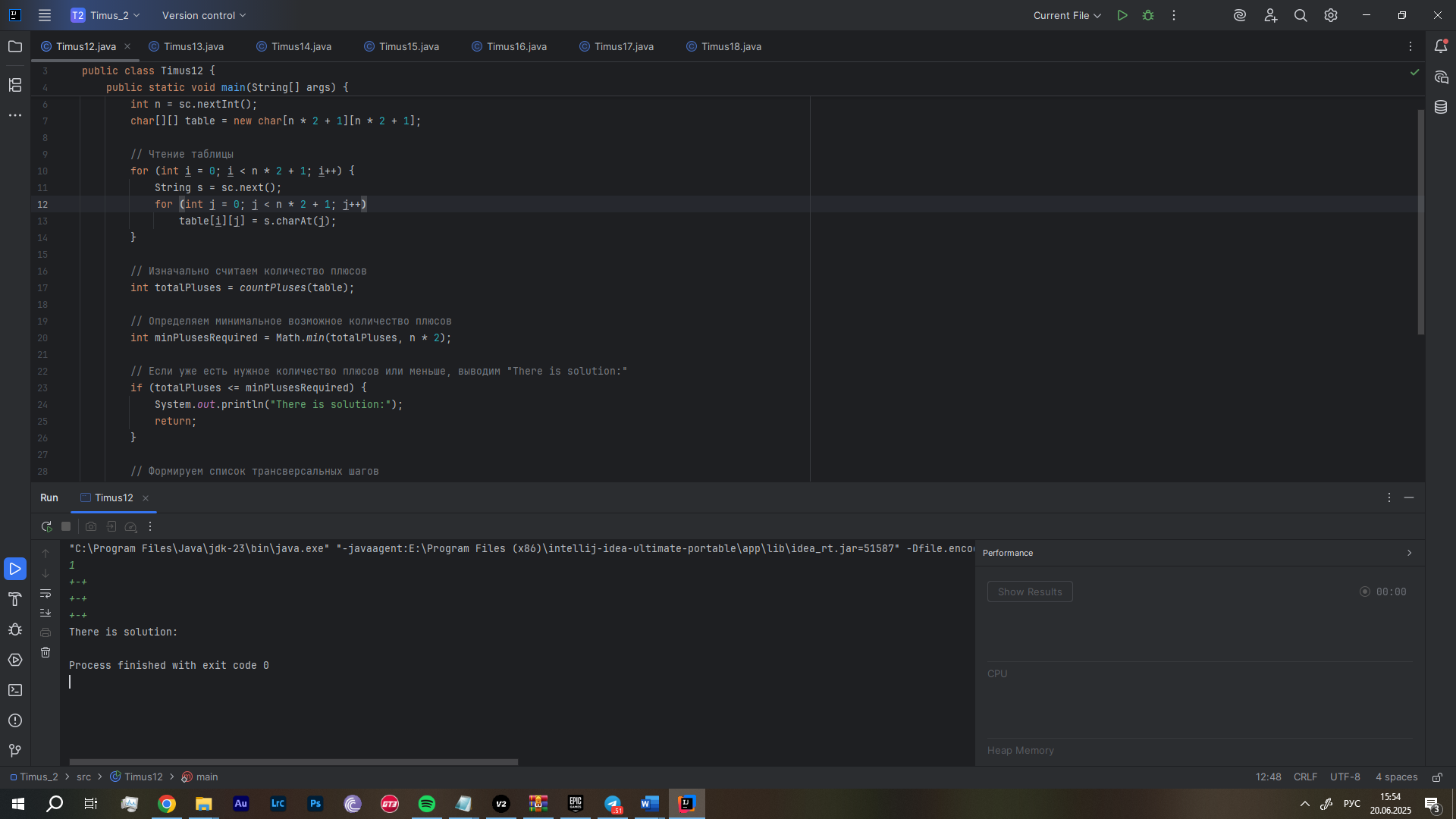
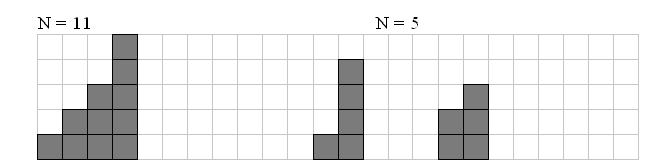


Рисунок 12 – Код и результат работы программы Timus12

**Задание 13**. У маленького мальчика есть набор из N кубиков. Из этих кубиков можно сложить различные лестницы. Лестницы имеют ступени различного размера, следующие в порядке возрастания этого размера (обратите особое внимание на то, что лестница не может иметь две одинаковые ступени). Каждая лестница должна иметь минимум две ступени, и каждая ступень должна состоять минимум из одного кубика. На рисунке приведены примеры лестниц для N=11 и N=5:



Найдите число Q различных лестниц, которые маленький мальчик может построить ровно из N кубиков.

Пользователь вводит число **n** – общее количество ступенек в лестнице. Затем, создается массив **dp** размером **n+1**, где **dp[i]** – будет хранить количество способов построить лестницу из i ступенек. При **dp[0] = 1** – существует один способ построить "пустую" лестницу (базовый случай), при **dp[1] = 1** – лестница из 1 ступеньки может быть построена только одним способом (одинарная ступенька)

Для каждой длины лестницы от 2 до N добавляется одинарная ступенька, количество способов равно **dp[i-1]** (так как к любой лестнице из i-1 ступенек просто добавляем одну ступеньку). Затем, если предыдущая ступенька была одинарной, добавляется двойная. Поэтому количество способов равно **dp[i-2]** (берем все лестницы из i-2 ступенек и добавляем двойную ступеньку); после чего считается общее количество ступенек: **dp[i]**.

После заполнения массива программа выводит значение **dp[n]** – количество способов построить лестницу из т ступенек.

Результат работы программы и ее программный код представлен на рисунке 13.

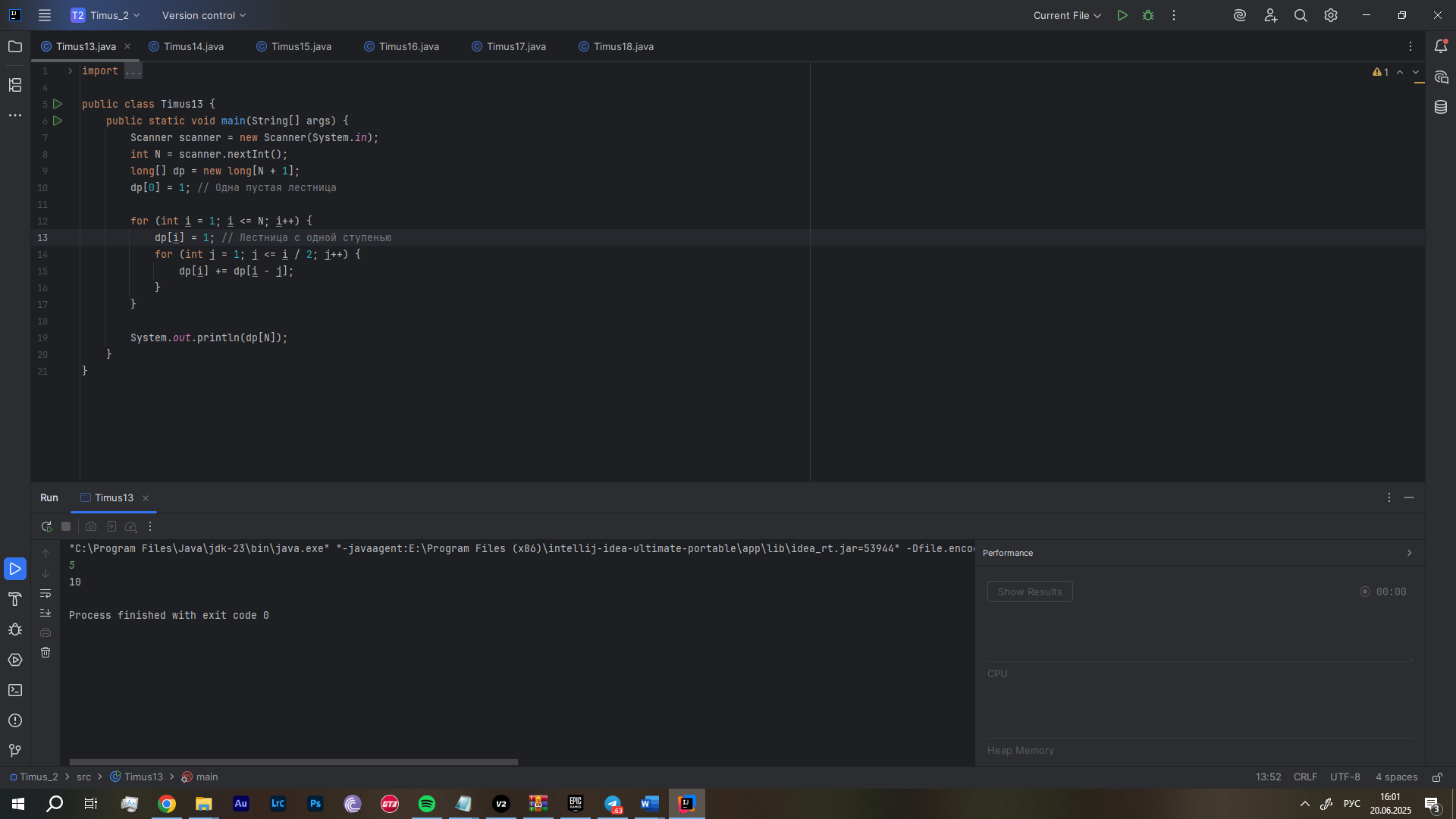


Рисунок 13 – Код и результат программы Timus13

**Задание 14**. Группа людей состоит из N членов. У каждого члена группы есть друзья в этой группе, один или более. Напишите программу, которая разделит группу на две команды. Каждый член каждой команды должен иметь друзей в другой команде.

Пользователем вводится число **N** – количество человек в группе, и для каждого человека перечисляются его друзья (до символа 0). Граф друзей хранится как список – **List<List<Integer>>**

Массив color (раскраска вершин) выглядит следующим образом: 0 – первая команда, 1 – вторая команда, -1 – еще не раскрашено. Для каждой еще не раскрашенной вершины: запускается DFS с цветом 0. При посещении соседа: если цвет такой же – конфликт, решение невозможно; если не раскрашен – красим в противоположный цвет

Если раскраска возможна – выводится первая команда, если найден конфликт – выводится 0.

Результат работы программы и ее программный код представлен на рисунке 14.

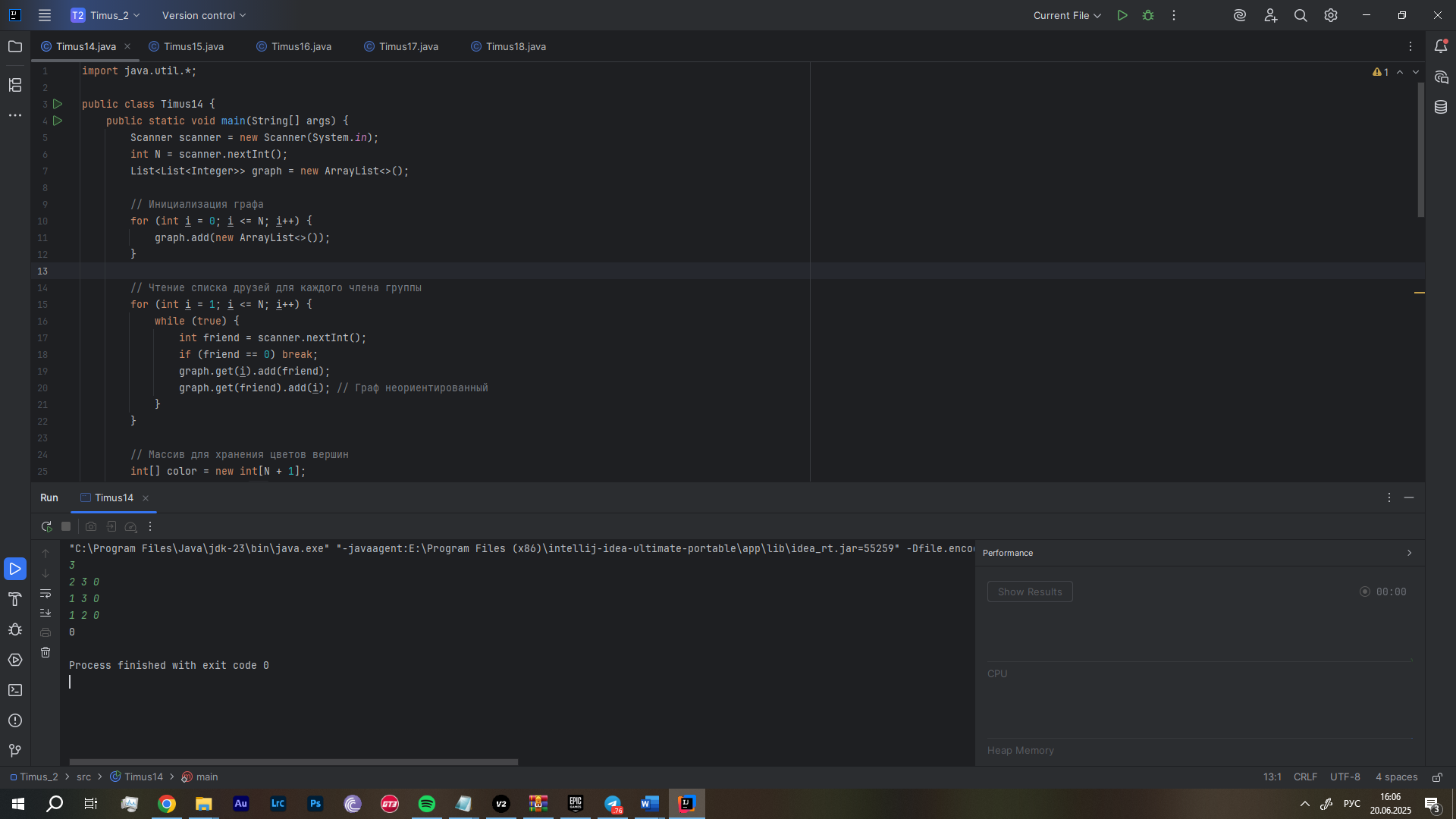


Рисунок 14 – Код и результат программы Timus14

**Задание 15.** N коробок расположены в ряд (1 ≤ N ≤ 20). Также есть A красных шаров и B синих шаров (0 ≤ A ≤ 15, 0 ≤ B ≤ 15). Красные шары (также как и синие) между собой одинаковы. Можно положить шары в коробки. Можно класть в коробку шары двух видов или только одного вида. Некоторые коробки можно оставить пустыми. Не обязательно класть все шары в коробки. Напишите программу, которая находит количество разных способов разместить шары в коробках описанным образом.

Вводятся пользователем: N – количество коробок, A – красные шары и B – синие шары. Создается трехмерный массив: **i** – использовано коробок, **j** – распределено красных шаров, **k** – распределено синих шаров.

Для каждой коробки существуют различные состояния:

а) оставить пустой;

б) положить только красный шар;

в) положить только синий шар;

г) положить оба шара.

В результате выполнения программы выводится массив **dp[N][A][B]**, в котором содержится ответ

Результат работы программы и ее код показан на рисунке 15.

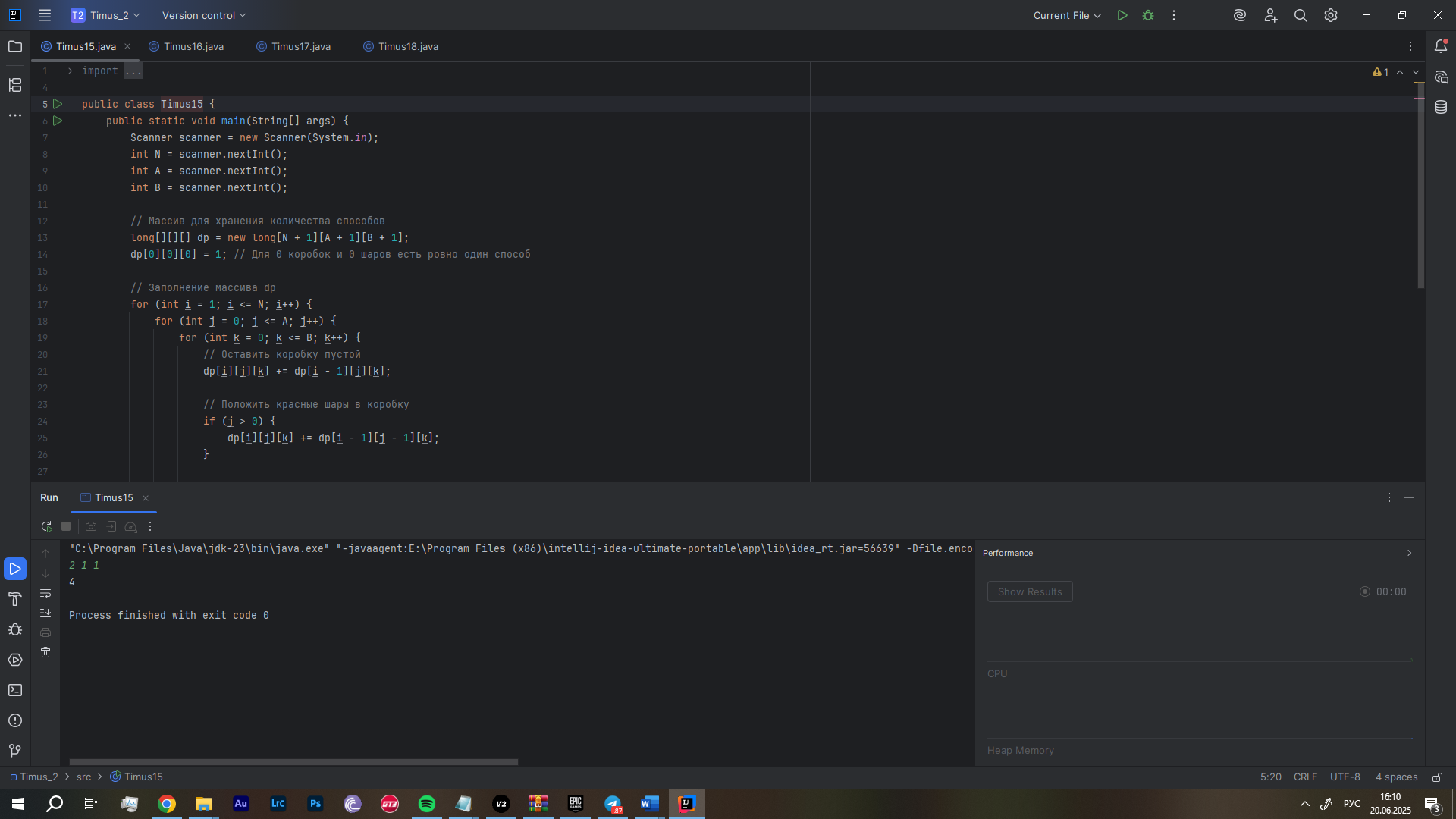


Рисунок 15 – Код и результат работы программы Timus15

**Задание 16.** Наш джип находится в пустыне и должен достичь точки, находящейся на расстоянии N километров от него. Рельеф сложный, машина старая, и похоже, что топливо вытекает, так что на каждый пройденный машиной километр тратится один литр топлива. В джипе есть канистры, общая ёмкость которых вместе с бензобаком составляет M литров (M < N). В начале дороги есть неограниченное количество топлива, а повсюду в пустыне есть пустые цистерны, в которых джип, проезжая, может оставить неограниченное количество топлива.

Напишите программу, которая находит минимальное количество топлива в литрах, которое необходимо для достижения заданной точки.

Пользователем вводятся числа **N** – общее расстояние и **M** – максимальный рывок. Создается массив **dp**, где **dp[i]** – минимальный расход для расстояния **i**. Создается массив dp, в котором **dp[0]** = 0, а остальные **dp[i]** равны бесконечности. Для каждого i от 1 до N рассматриваются все возможные прыжки j. Результатом для **dp[i]** будет **min(dp[i], dp[i-j] + j)**. После, выводится **dp[N]** – искомый минимальный расход.

Результат работы программы и ее программный код представлен на рисунке 16.

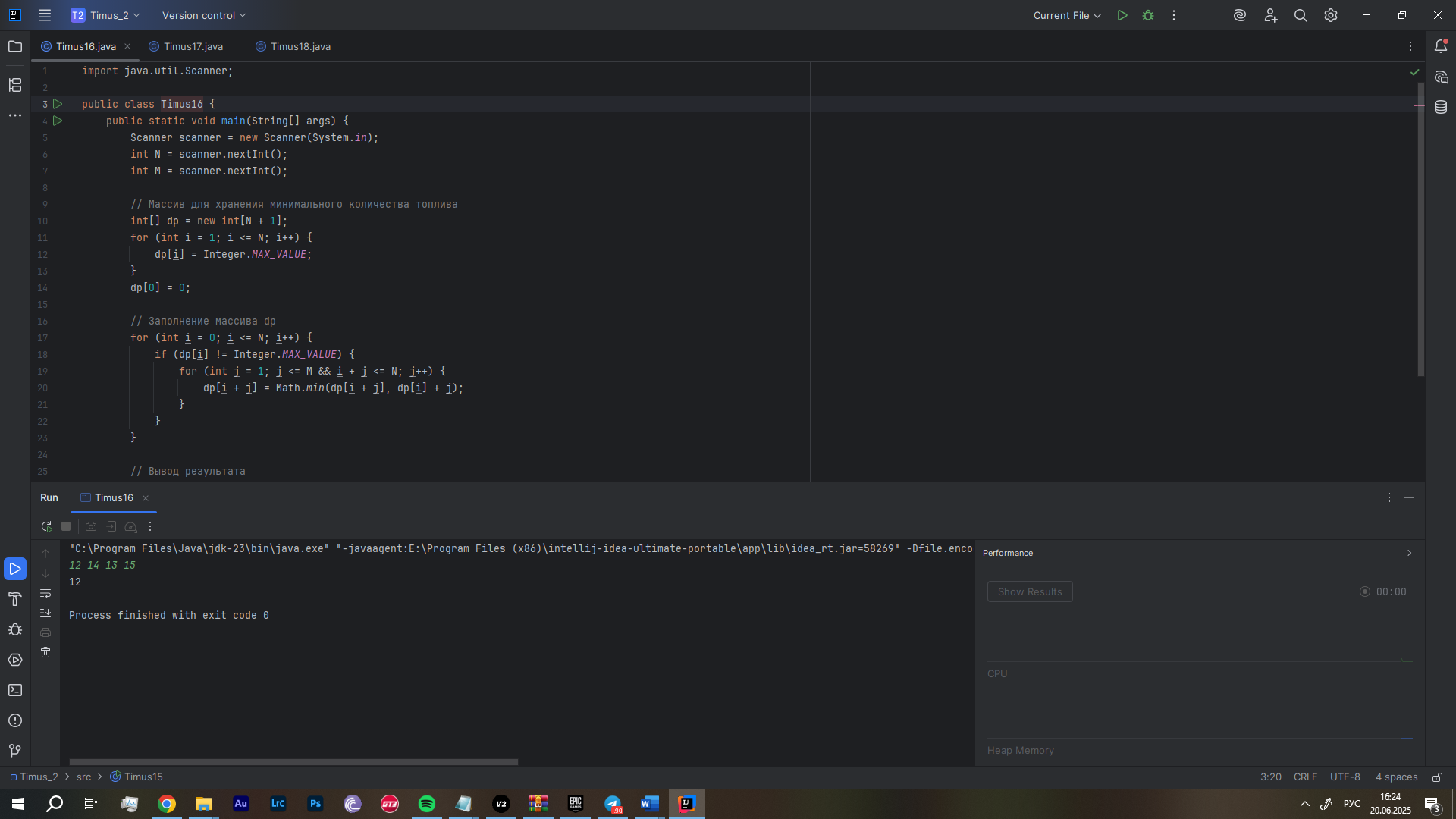


Рисунок 16 – Код и результат программы Timus16

**Задание 17**. Даны целые числа N, M и Y. Напишите программу, которая найдёт все целые числа X в диапазоне [0, M – 1], такие что XN mod M = Y.

Пользователь вводит данные: **N** – степень, **M** – модуль и **Y** – остаток. Программой осуществляется перебор всех **X от 0 до M-1**; для каждого **X** вычисляется **X^N mod M**. После возведения в степень, если результат равен Y – добавляем в ответ.

Результат работы программы и ее программный код представлен на рисунке 17.

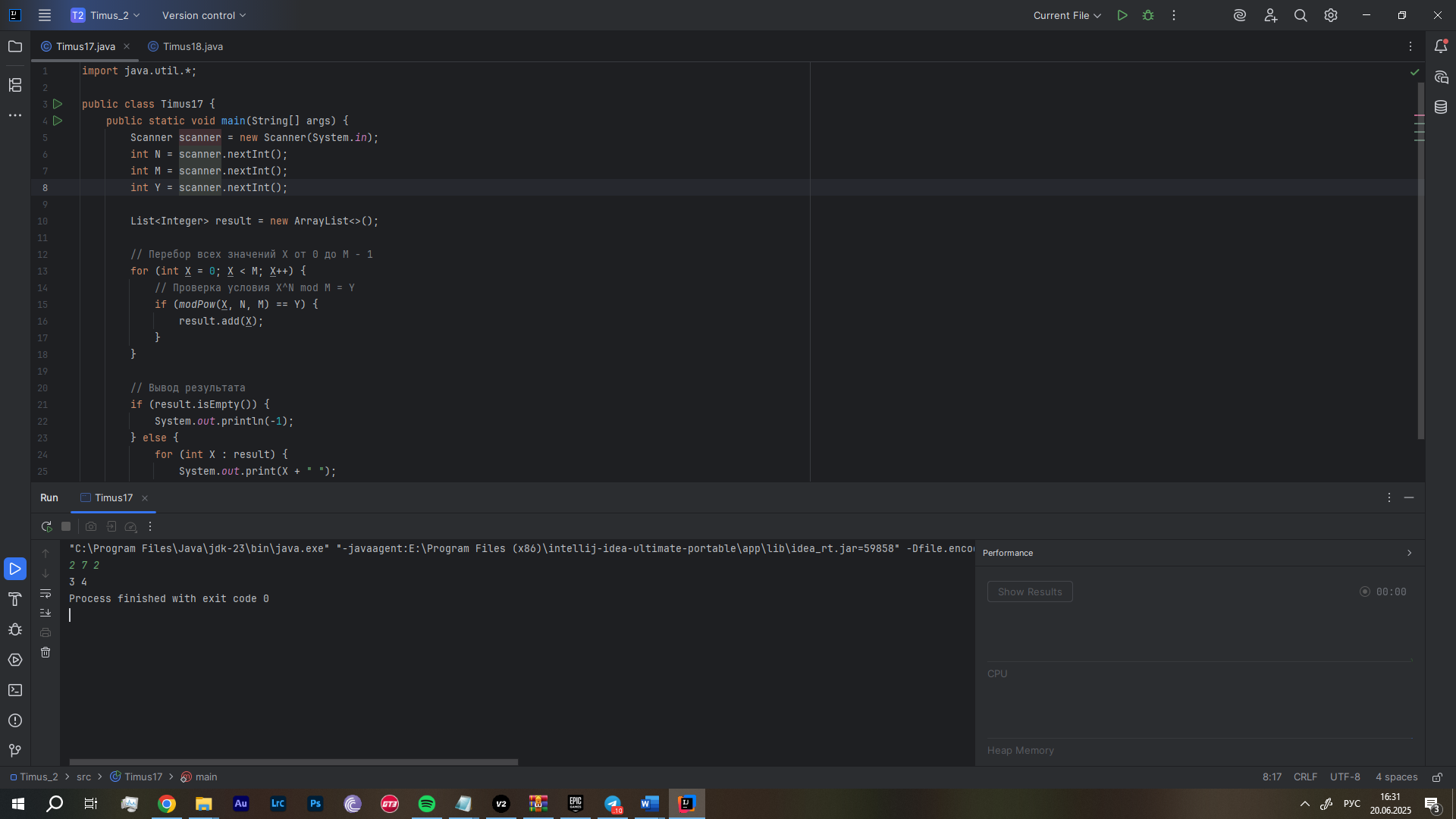


Рисунок 17 – Код и результат программы Timus17

**Задание 18**. Число x называется квадратным корнем числа a по модулю n (root (a, n)) тогда и только тогда когда x \* x = a (mod n). Напишите программу, которая находит все значения квадратных корней числа a по модулю n.

Вводится значение **K** – количество тестов, для каждого теста: a и n Далее, проверяются **все x от 1 до n-1**, и вычисляется **x^2 mod n**. Если полученный результат равен **a** – он добавляется в ответ и выводится в порядке возрастания.

Результат работы программы и ее программный код представлен на рисунке 18.

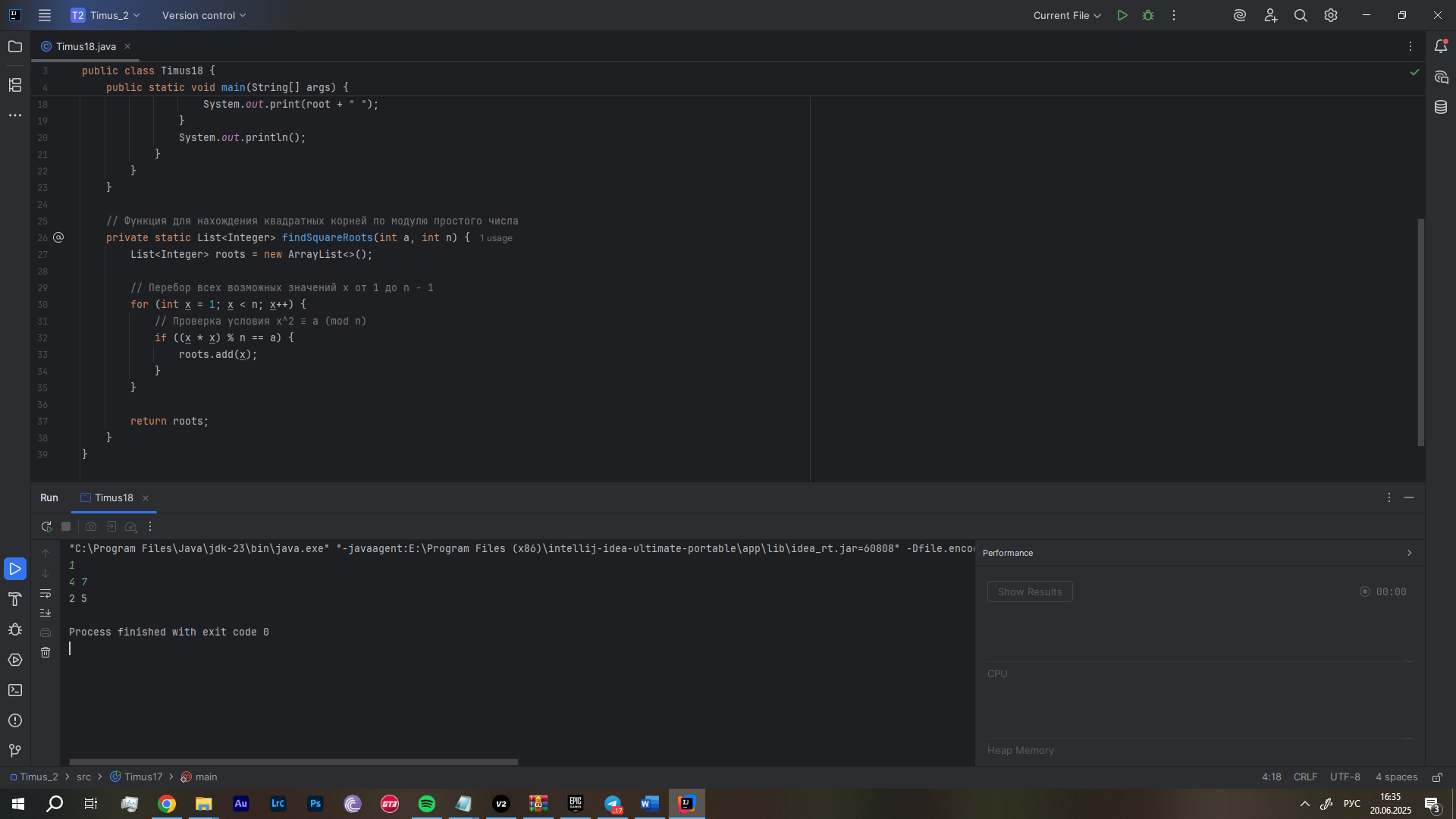


Рисунок 18 – Код и результат программы Timus18

**Задание 19**. Хороший охотник убивает двух зайцев одним выстрелом. Конечно же это может быть легко сделано, поскольку через любые две точки можно провести прямую. Но убить трёх и более зайцев одним выстрелом — намного более сложная задача. Чтобы стать лучшим охотником в мире, нужно уметь убить максимально возможное количество зайцев. Представим зайца точкой на плоскости. Точка задаётся целочисленными координатами x и y. Вам нужно найти максимальное число зайцев, которые могут быть убиты одним выстрелом, то есть максимальное количество точек заданного множества, лежащих точно на одной прямой. Никакие два зайца не находятся в одной точке.

Пользователь вводит **N** – количество точек, и для каждой точки вводятся вводятся координаты (x, y). Для их хранения используется класс Point, а массив **points** хранит сами введенные точки.

Для каждой пары точек (i, j) вычисляется вектор направления dx, а для каждой третьей точки k – проверяется коллинеарность через векторное произведение. Если проверяемое условие выполняется, точка k лежит на прямой (i, j). Далее, подсчитывается количество коллинеарных точек для каждой прямой, и выводится максимальное количество точек, лежащих на одной прямой.

Результат работы программы и ее программный код представлен на рисунке 19.

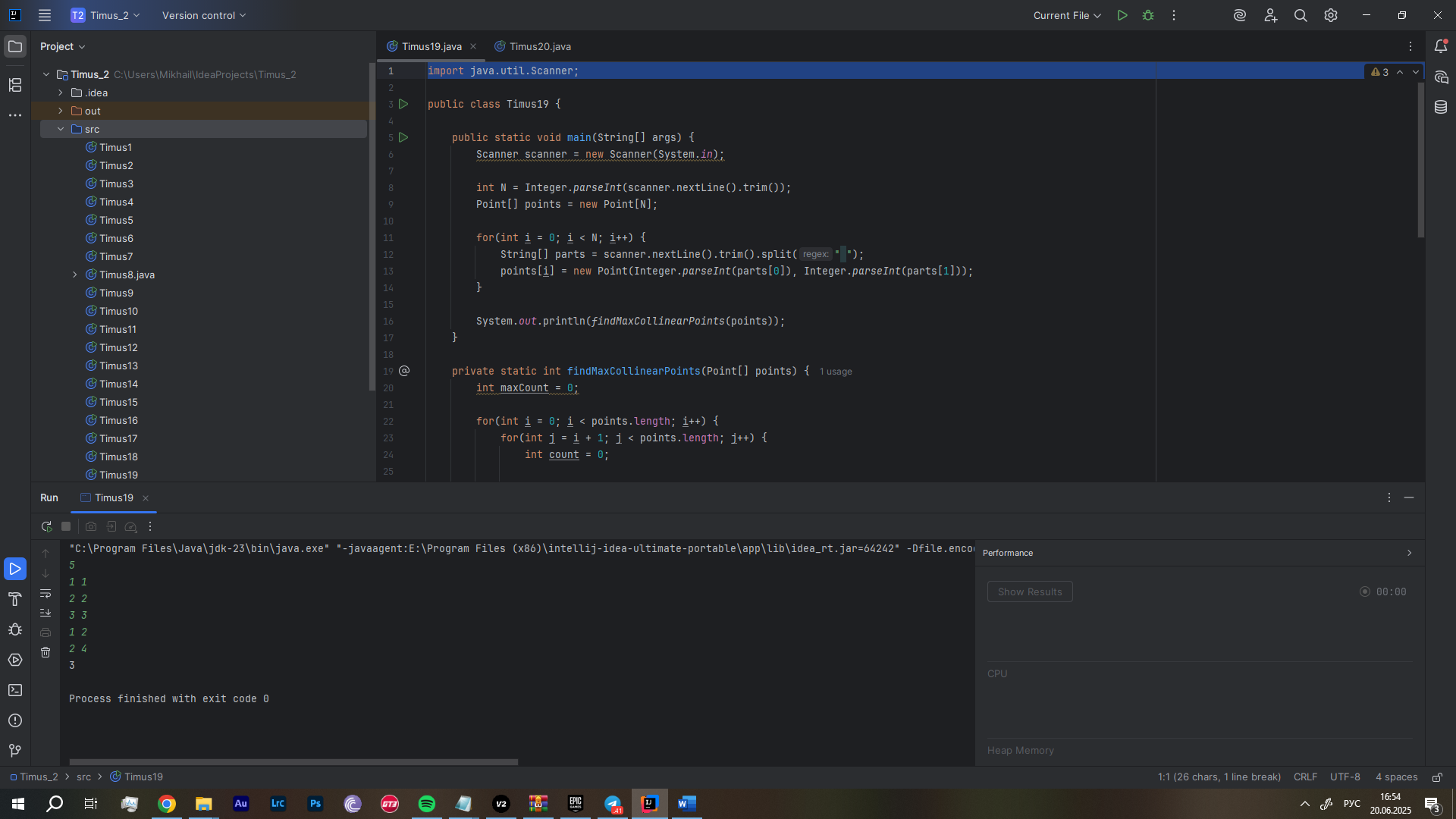


Рисунок 19 – Код и результат программы Timus19

**Задание 20**. 12 Месяцев кружат свой небесный хоровод. Один за другим они занимают трон — от самого молодого и свирепого Января до пожилого и мудрого Декабря. Уходя, каждый Месяц громко выкрикивает цифру. За год из цифр складывается 12-значное число. Старый Год использует это число как щит на своём пути в Бездну Времени, защищаясь им от кошмарных созданий Вечности. От ударов щит разлетается на куски, соответствующие делителям числа.

Вводятся **months** (месяцы) и **prefix** – начальные цифры искомого числа. Создается 12-значное число, префикс дополняется нулями до 12 цифр. Далее, последовательно проверяются числа начиная с сгенерированного:

а) числа меньше или равные 1 – не простые;

б) числа 2 и 3 – простые

Четные числа и кратные 3 исключаются, проверяются делители. Программа выводит найденное 12-значное простое число. Результат работы программы и ее программный код представлен на рисунке 20.

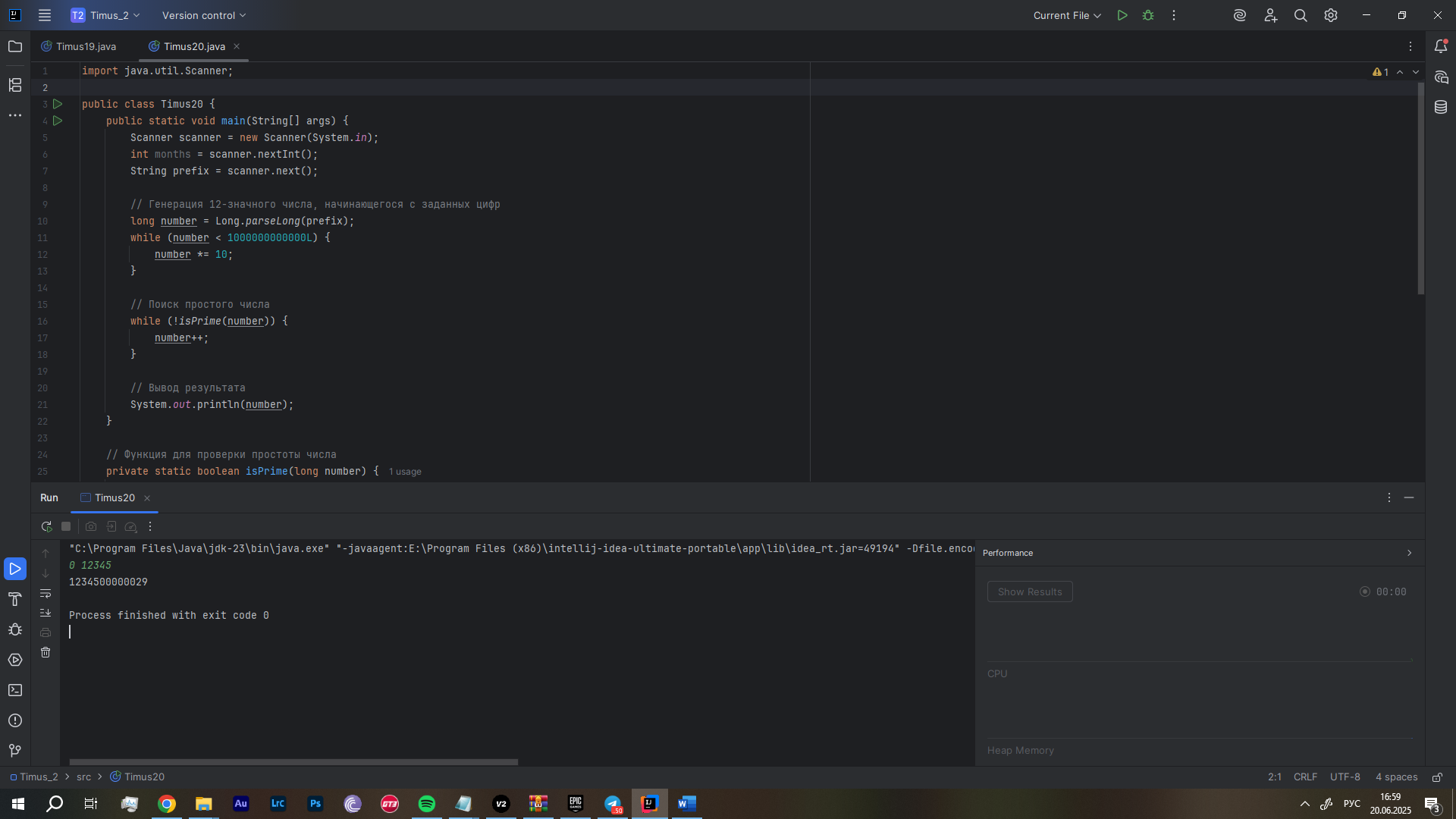


Рисунок 20 – Код и результат программы Timus20