Лабораторна робота №2

**Тема:** Класи

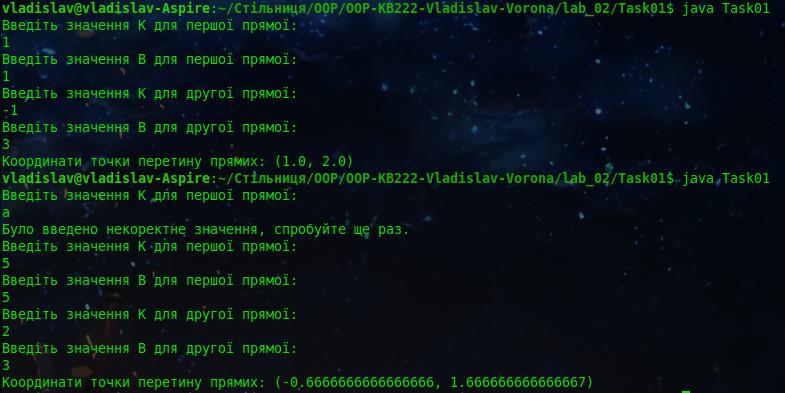
**Мета роботи**: Використовуючи теоретичне підґрунтя про об’єктно орієнтоване програмування виконати дії що будуть вказано в завданні до лабораторної роботи.

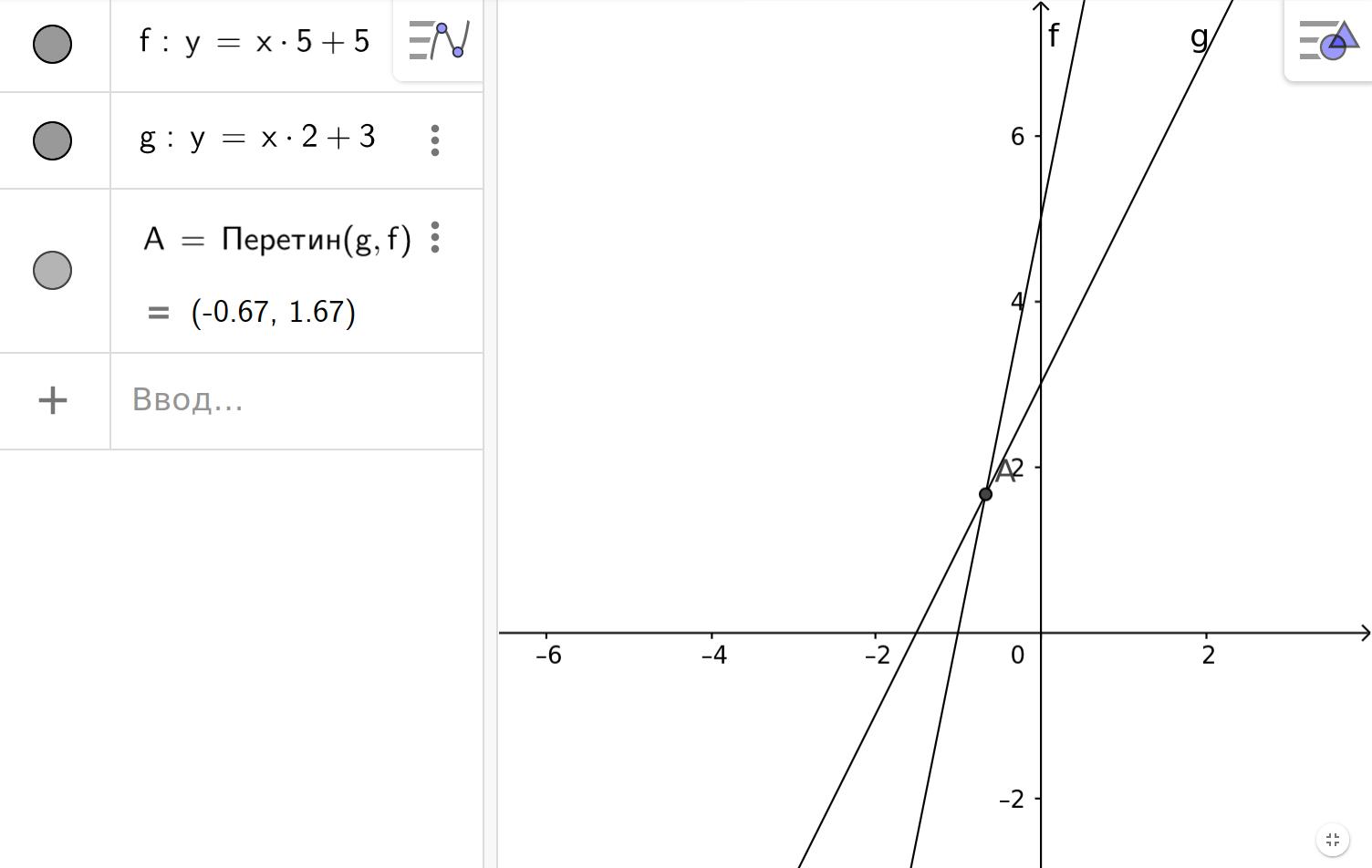
**Хід роботи:**

**Посилання на GitHub:** <https://github.com/VladislavVorona/OOP-KB222-Vladislav-Vorona>

**1. Вам необхідно реалізувати метод intersection у класі Line. Він повинен повертати точку перетину двох ліній (клас Point). Якщо лінії збігаються або перетинаються, метод повинен повертати значення null. Функція, що описує пряму y = k X + b. Користувач вводить значення k та b для двох прямих.**

Спочатку я імпортував два класи з пакету java.util: Scanner і InputMismatchException. Ці класи дозволять мені зчитувати введені користувачем значення з консолі і обробляти помилки під час введення інформації. Далі я створив класс Task01 і вказав там метод main, який є вхідною точкою виконання програми. У цьому методі я створюю об'єкт класу Scanner, який буде використовуватися для зчитування введених користувачем значень. Також я оголошую змінні для зберігання значень для прямих xk1, xb1, xk2, xb2, які відповідають за значення коефіцієнтів k і b для першої і другої прямих відповідно. Далі я використовую безкінечний цикл while (true), щоб перевіряти введенні користувачем значення. У цьому циклі я використовую блок try-catch, щоб перехопити можливі винятки, які можуть виникнути при некоректному введенні значень. Якщо користувач введе некоректне значення, програма виведе повідомлення про помилку і запросить користувача ввести значення ще раз, якщо введені дані коректні, цикл перерветься за допомогою оператора break. Після виходу з циклу я закриваю об'єкт Scanner, щоб уникнути витоку ресурсів. Далі я створюю два об'єкти класу Line на основі введених раніше значень xk1, xb1 та xk2, xb2. Ці об'єкти представляють собою прямі у просторі з координатами, заданими коефіцієнтами k і b. Після створення об'єктів прямих, я викликаю метод intersection на об'єкті l1 (перша пряма) і передаю в нього другу пряму l2. Цей метод обчислює точку перетину двох прямих за допомогою формул для розв'язання системи лінійних рівнянь. В йому спочатку заносяться коефіцієнти другої прямої з об'єкта l2, я зберігаю їх у відповідних змінних xk2 і xb2. Після цього я зберігаю коефіцієнти першої прямої із this.xk і this.xb у змінні xk і xb відповідно. Це робиться для зручності написання коду. Після збереження коефіцієнтів обох прямих я перевіряю, чи співпадають значення xk і xk2. Якщо вони співпадають, це означає, що прямі паралельні або співпадають, у цьому випадку неможливо знайти точку перетину, оскільки вони або ніколи не перетинаються, або мають безліч точок перетину і тому я виводжу відповідне повідомлення і повертаю null. Якщо ж коефіцієнти прямих різні, то я використовую формули для знаходження точки перетину прямих. Я обчислюю координату x точки перетину, використовуючи формулу: x = (b2 − b1) / (k1 − k2). Після цього, використовуючи отримане значення x, обчислюю відповідну координату y за допомогою формули прямої: y = k1 \* x + b1. Ці обчислені значення x і y створюють новий об'єкт класу Point, який представляє собою точку перетину двох прямих, цей об'єкт повертається як результат роботи методу intersection. Клас Point використовується для представлення точки з координатами x та y. Він має конструктор, який приймає координати точки і зберігає їх у відповідних полях, метод toString перевизначено, щоб представити точку у вигляді рядка "(x, y)". Клас Line представляє пряму на площині за допомогою рівняння y = kx + b. Він має конструктор, який приймає коефіцієнти k і b і зберігає їх у відповідних полях. Метод intersection обчислює точку перетину двох прямих, якщо вона існує. Якщо прямі паралельні або співпадають, метод виводить відповідне повідомлення і повертає null. Таким чином, ця програма дозволяє вводити коефіцієнти двох прямих і знаходити їх точку перетину, якщо вона існує. Приклад роботи програми та перевірку вірності результатів представлено на рисунках 1.1 та 1.2.

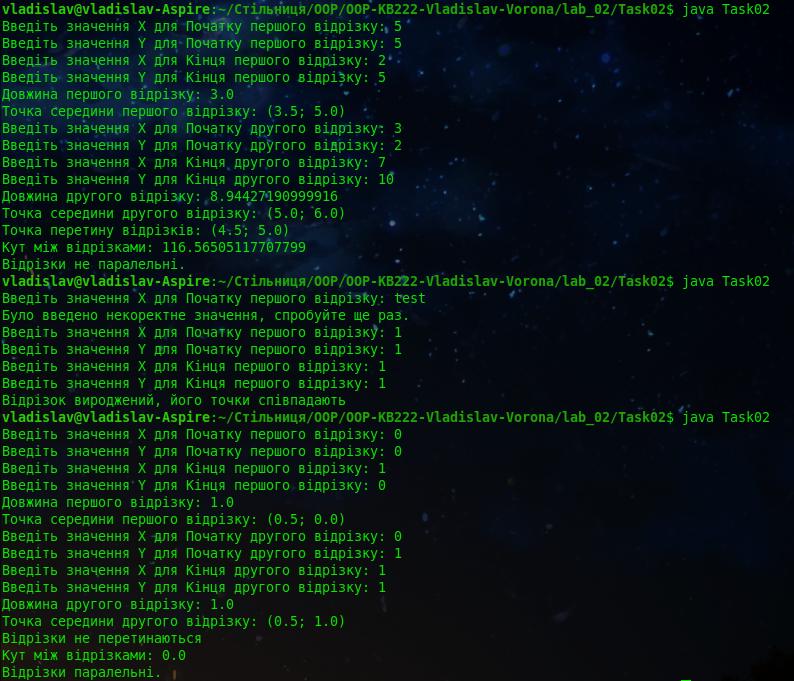
Рисунок 1.1 - Приклад роботи програми

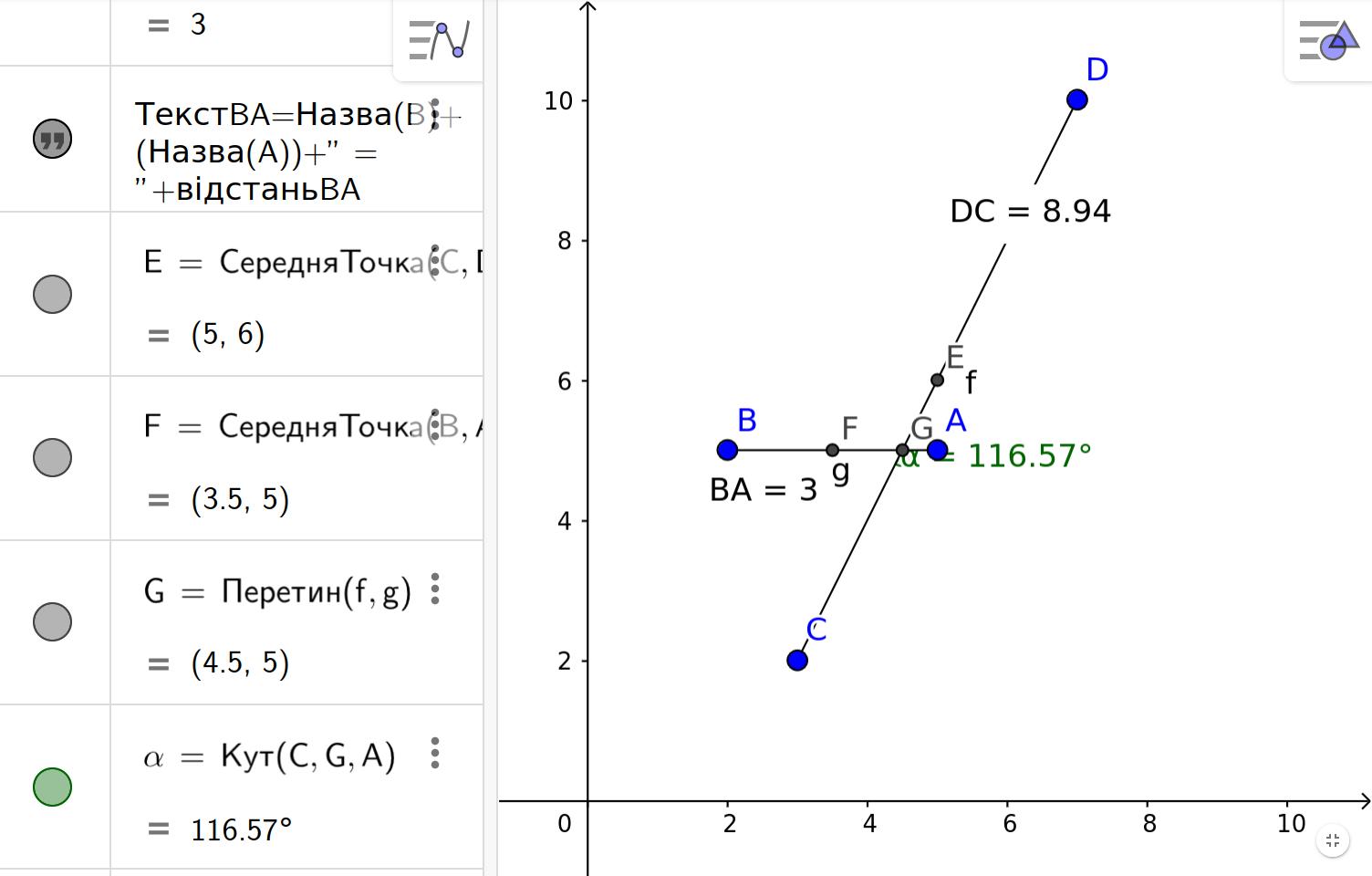
Рисунок 1.2 - Перевірка вірності відповіді програми на GeoGebra 2D

|  |
| --- |
| import java.util.Scanner;  import java.util.InputMismatchException;    public class Task01 {  public static void main(String[] args) {  Scanner InpScn = new Scanner(System.in);    double xk1 = 0, xb1 = 0, xk2 = 0, xb2 = 0;    while (true) {  try {  System.out.println("Введіть значення K для першої прямої: ");  xk1 = InpScn.nextDouble();    System.out.println("Введіть значення B для першої прямої: ");  xb1 = InpScn.nextDouble();    System.out.println("Введіть значення K для другої прямої: ");  xk2 = InpScn.nextDouble();    System.out.println("Введіть значення B для другої прямої: ");  xb2 = InpScn.nextDouble();    break;  } catch (InputMismatchException e) {  System.out.println("Було введено некоректне значення, спробуйте ще раз.");  InpScn.next();  }  }    InpScn.close();    Line l1 = new Line(xk1, xb1);  Line l2 = new Line(xk2, xb2);    Point res = l1.intersection(l2);    if (res != null) {  System.out.println("Координати точки перетину прямих: " + res);  }  }  }    class Point {  private double xx, yy;    public Point(double xx, double yy) {  this.xx = xx;  this.yy = yy;  }    @Override  public String toString() {  return "(" + xx + ", " + yy + ")";  }  }    class Line {  private double xk, xb;    public Line(double xk, double xb) {  this.xk = xk;  this.xb = xb;  }    public Point intersection(Line l2) {  double xk2 = l2.xk;  double xb2 = l2.xb;    xk = this.xk;  xb = this.xb;    if (xk == xk2) {  if (xb == xb2) {  System.out.println("Прямі співпадають, тому не можливо знайти точку перетину.");  } else {  System.out.println("Прямі паралельні, тому не можливо знайти точку перетину.");  }    return null;  }    double xx = (xb2 - xb) / (xk - xk2);  double yy = xk \* xx + xb;    return new Point(xx, yy);  }  } |

**2. Реалізуйте методи класу Segment (відрізок): Конструктор, в який як параметри передаються координати точок початку і кінця відрізка (використовуйте клас Point). Переконайтеся, що створений відрізок існує і не є виродженим, що означає, що початок і кінець відрізка не є однією точкою. Реалізуйте метод double length() – повертає довжину сегмента. Реалізуйте метода Point middle() – повертає середню точку сегмента. Реалізуйте метод Point intersection(Segment another) – повертає точку перетину поточного відрізка з іншим.**

Для виконання цього завдання я імпортував два класи з пакету java.util: Scanner для зчитування введених користувачем значень та InputMismatchException для обробки помилок при некоректному введенні. Далі я створив клас Task02, де вказав головний метод main, який є вхідною точкою виконання програми. У цьому методі я створив об'єкт класу Scanner, який буде використовуватися для зчитування координат точок. Потім я використовую статичний метод InpPoint класу Point, який відповідає за введення координат точки із консолі. Цей метод приймає об'єкт Scanner та рядок, що містить назву точки, і повертає об'єкт типу Point з введеними координатами. Також тут виконано перевірку введених даних користувачем за допомогою безкінечного циклу while true, як у попередньому завданні. Далі я створив об'єкт типу Segment для першого відрізку та вніс до його координати введених точок початку і кінця. Після цього я перевірив, чи відрізок не є виродженим, тобто чи його довжина не дорівнює нулю, що вказує на те, що початкова та кінцева точки співпадають. Якщо відрізок вироджений, програма виводить відповідне повідомлення та завершує роботу. Якщо ні, обчислюється довжина та середина відрізка і виводяться відповідні результати. Подібні дії виконуються і для другого відрізку. Після цього здійснюється пошук точки перетину відрізків за допомогою методу intersection. Цей метод використовує алгоритм перетину відрізків та обчислює координати точки перетину. Якщо точка перетину існує, вона виводиться на екран, інакше виводиться повідомлення, що відрізки не перетинаються. В кінці обчислюється кут між відрізками за допомогою методу anglSeek та визначається, чи вони паралельні за допомогою методу parralel, всі результати виводяться на екран. Далі я поясню, як працюють методи, які були описані вище. Метод length використовується для обчислення довжини відрізка між двома точками, він використовує формулу відстані між двома точками у просторі, яка відома як теорема Піфагора, за цією формулою довжина відрізка дорівнює квадратному кореню з суми квадратів різниць координат по осях X та Y між початковою та кінцевою точками, тобто якщо координати початкової точки відрізка - x1, y1, а кінцевої точки - x2, y2, то формула дорівнює L = sqrt((x2 - x1)^2 + (y2 - y1)^2), тому тут я використав статичний метод Math.hypot(), який обчислює гіпотенузу за заданими катетами (різницями координат по осях X та Y між двома точками) і таким чином метод length використовує цю функцію для обчислення довжини відрізка між двома заданими точками. Далі я створив метод middle, який використовується для обчислення координат точки, яка знаходиться на середині відрізка між двома його кінцями. Для цього він обчислює середнє значення координат по осях X та Y кінців відрізка, таким чином, що якщо координати початкової точки відрізка x1 та y1, а кінцевої точки x2 та y2, то середні значення координат точки на середині відрізка розраховуються за формулами x = (x1 + x2) / 2, y = (y1 + y2) / 2. Далі я розробив метод anglSeek, який призначений для обчислення кута між двома відрізками на площині. Він використовує властивість векторного добутку векторів та арктангенс для обчислення кута між двома відрізками. Спочатку обчислюється кут нахилу кожного відрізка відносно вісі X за допомогою функції Math.atan2, яка повертає арктангенс відношення координат точки до точки [0, 0]. Потім абсолютна різниця між цими кутами обчислюється і якщо вона більше за 180 градусів, то використовується доповнення до 360 градусів для обчислення правильного кута, остаточний результат повертається у радіанах і конвертується в градуси за допомогою функції Math.toDegrees. Далі я створив метод parallel, який використовується для перевірки паралельності двох відрізків. Для початку, я обчислюю різницю між координатами кінців першого відрізка і різницю між координатами кінців другого відрізка. Після цього я використовую формулу x1 \* y2 – x2 \* y1 для обчислення векторного добутку цих різниць. Цей векторний добуток дозволяє визначити, чи є відрізки паралельними. Якщо результат цього виразу дуже близький до нуля, тобто менше за 1e-9, то це означає, що вектори практично колінеарні, тобто відрізки майже паралельні. Таким чином, метод повертає true, якщо відрізки паралельні та false в іншому випадку. Також я створив метод класу з назвою intersection, який приймає на вхід об'єкт типу Segment і повертає об'єкт типу Point, яка є перетином цього відрізка з іншим відрізком, переданим у функцію як аргумент. В коді я вказую у змінних координати початку та кінця поточного відрізка і відрізка, переданого у функцію. Далі я виконую обчислення детермінанту xdem для визначення, чи перетинаються відрізки. Далі перевіряється, чи детермінант не рівний нулю, оскільки це означає, що вектори відрізків паралельні або лежать на одній прямій, тому вони не перетинаються. У цьому випадку функція повертає null, а якщо він не рівний нулю, то обчислюються координати точки перетину. Далі створюється новий об'єкт типу Point з обчисленими координатами перетину. Далі йде перевірка, чи знаходиться точка перетину на обох відрізках. Для цього використовується допоміжна функція seekPt, яка перевіряє, чи належить точка відрізку і якщо точка перетину не знаходиться на обох відрізках, повертається null, а якщо точка перетину належить обом відрізкам, то вона повертається, як результат роботи методу. І останнім методом у класі Segment є метод seekPt, цей метод використовується для перевірки того, чи знаходиться задана точка pt всередині відрізка, який визначений вказаним відрізком l. Спочатку я отримую координати кінців відрізка, а також координати нашої точки pt, після цього код знаходить мінімальні та максимальні значення координат x та y для кінців відрізка l, це допомагає визначити проміжок значень координат, в межах якого код шукає точку. Далі перевіряється, чи координати точки pt потрапляють всередину відрізка l, це робиться за допомогою умови, яка перевіряє, чи xx, координата x точки, знаходиться між minXX і maxXX, мінімальна та максимальна координати x відрізка l2, і чи xy, координата y точки, знаходиться між minYY і maxYY, мінімальна та максимальна координати y відрізка l. Якщо умова виконується для обох координат, то точка pt знаходиться всередині відрізка l2, і метод повертає true, що показує, що точка належить відрізку, в іншому випадку, метод повертає false, що означає, що точка знаходиться поза відрізком. Приклад роботи програми та порівняння вірності результатів представлено на рисунках 2.1 та 2.2.

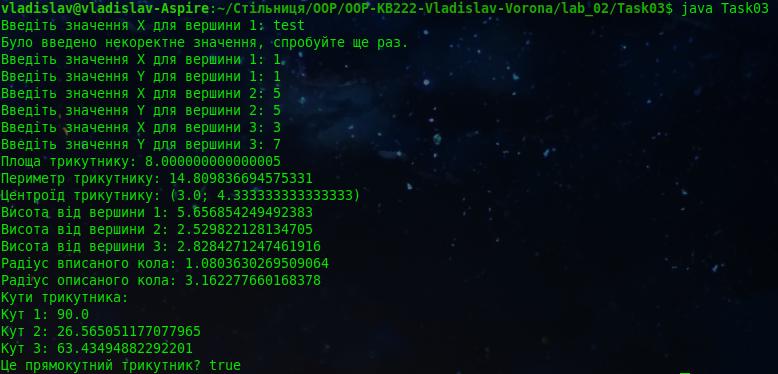
Рисунок 2.1 - Приклад роботи програми

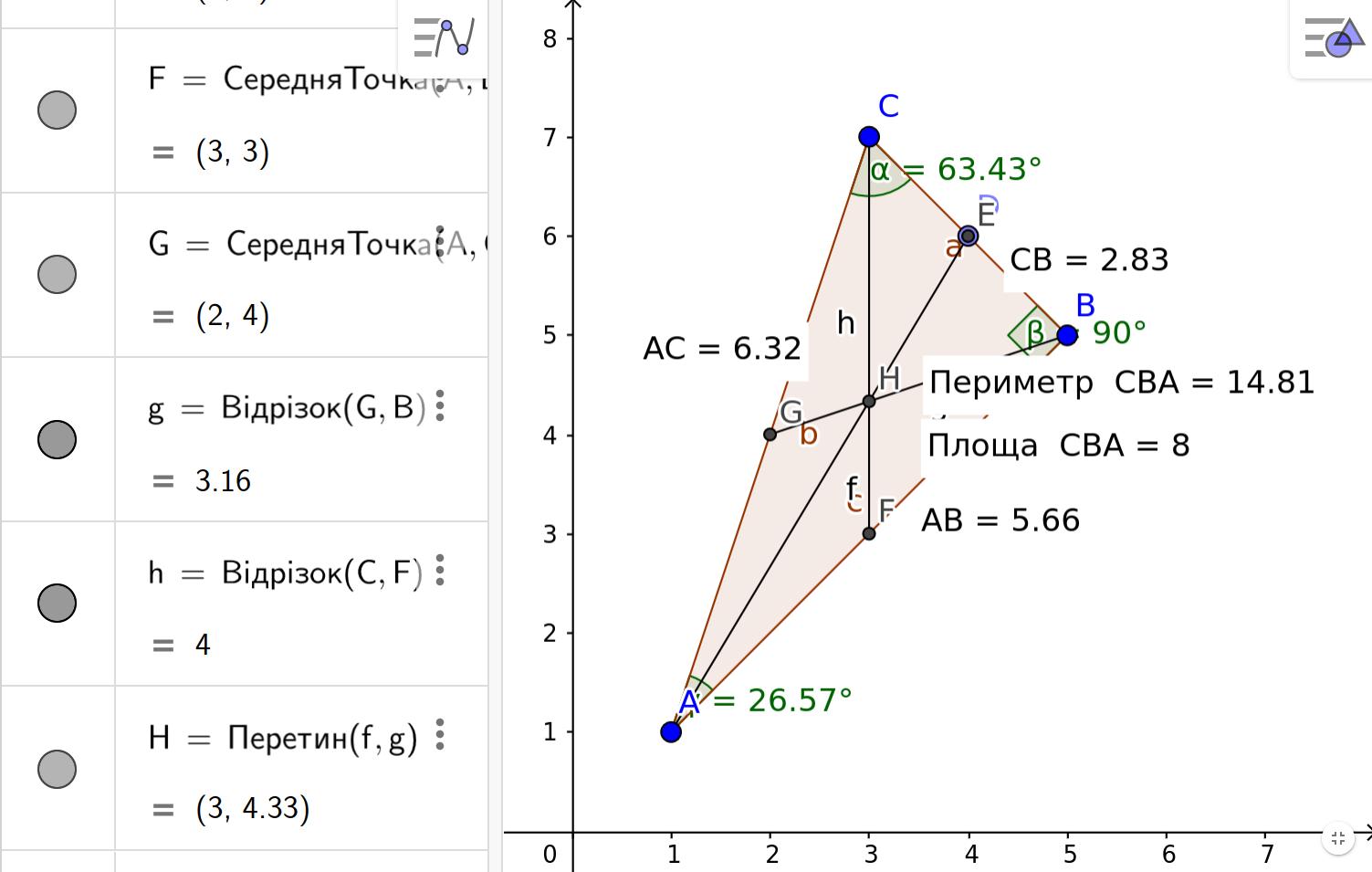
Рисунок 2.2 - Перевірка вірності відповіді програми на GeoGebra 2D

|  |
| --- |
| import java.util.Scanner;  import java.util.InputMismatchException;    public class Task02 {  public static void main(String[] args) {  Scanner InpScn = new Scanner(System.in);    Point startPt = Point.InpPoint(InpScn, "Початку першого відрізку");  Point endPt = Point.InpPoint(InpScn, "Кінця першого відрізку");    Segment l1 = new Segment(startPt, endPt);    if (l1.length() == 0) {  System.out.println("Відрізок вироджений, його точки співпадають");  return;  }    System.out.println("Довжина першого відрізку: " + l1.length());  System.out.println("Точка середини першого відрізку: " + l1.middle());    startPt = Point.InpPoint(InpScn, "Початку другого відрізку");  endPt = Point.InpPoint(InpScn, "Кінця другого відрізку");    Segment l2 = new Segment(startPt, endPt);    if (l2.length() == 0) {  System.out.println("Відрізок вироджений, його точки співпадають");  return;  }    System.out.println("Довжина другого відрізку: " + l2.length());  System.out.println("Точка середини другого відрізку: " + l2.middle());    Point res = l1.intersection(l2);    if (res != null) {  System.out.println("Точка перетину відрізків: " + res);  } else {  System.out.println("Відрізки не перетинаються");  }    double angl = l1.anglSeek(l2);  System.out.println("Кут між відрізками: " + Math.toDegrees(angl));  System.out.println("Відрізки " + (l1.parralel(l2) ? "" : "не ") + "паралельні.");  }  }    class Point {  double xx, xy;    Point(double xx, double xy) {  this.xx = xx;  this.xy = xy;  }    static Point InpPoint(Scanner InpScn, String ptName) {  while (true) {  try {  System.out.print("Введіть значення X для " + ptName + ": ");  double xx = InpScn.nextDouble();  System.out.print("Введіть значення Y для " + ptName + ": ");  double xy = InpScn.nextDouble();  return new Point(xx, xy);  } catch (InputMismatchException e) {  System.out.println("Було введено некоректне значення, спробуйте ще раз.");  InpScn.nextLine();  }  }    }    @Override  public String toString() {  return "(" + xx + "; " + xy + ")";  }  }    class Segment {  Point startPt, endPt;    Segment(Point startPt, Point endPt) {  this.startPt = startPt;  this.endPt = endPt;  }    double length() {  return Math.hypot(startPt.xx - endPt.xx, startPt.xy - endPt.xy);  }    Point middle() {  return new Point((startPt.xx + endPt.xx) / 2, (startPt.xy + endPt.xy) / 2);  }    double anglSeek(Segment l2) {  double l1Angle = Math.atan2(endPt.xy - startPt.xy, endPt.xx - startPt.xx);  double l2Angle = Math.atan2(l2.endPt.xy - l2.startPt.xy, l2.endPt.xx - l2.startPt.xx);  double angl = Math.abs(l1Angle - l2Angle);  return Math.min(angl, 2 \* Math.PI - angl);  }    boolean parralel(Segment l2) {  double dx1 = startPt.xx - endPt.xx;  double dy1 = startPt.xy - endPt.xy;  double dx2 = l2.startPt.xx - l2.endPt.xx;  double dy2 = l2.startPt.xy - l2.endPt.xy;  return Math.abs(dx1 \* dy2 - dx2 \* dy1) < 1e-9;  }    Point intersection(Segment l2) {  double xx1 = startPt.xx, yy1 = startPt.xy;  double xx2 = endPt.xx, yy2 = endPt.xy;    double xx1l2 = l2.startPt.xx, yy1l2 = l2.startPt.xy;  double xx2l2 = l2.endPt.xx, yy2l2 = l2.endPt.xy;    double xdem = (xx1 - xx2) \* (yy1l2 - yy2l2) - (yy1 - yy2) \* (xx1l2 - xx2l2);  if (xdem == 0.0) return null;    double xx = ((xx1 \* yy2 - yy1 \* xx2) \* (xx1l2 - xx2l2) - (xx1 - xx2) \* (xx1l2 \* yy2l2 - yy1l2 \* xx2l2)) / xdem;  double xy = ((xx1 \* yy2 - yy1 \* xx2) \* (yy1l2 - yy2l2) - (yy1 - yy2) \* (xx1l2 \* yy2l2 - yy1l2 \* xx2l2)) / xdem;    Point resPt = new Point(xx, xy);    if (!seekPt(resPt, this) || !seekPt(resPt, l2)) {  return null;  }    return resPt;  }    private boolean seekPt(Point pt, Segment l) {  double xx1 = l.startPt.xx, yy1 = l.startPt.xy;  double xx2 = l.endPt.xx, yy2 = l.endPt.xy;  double xx = pt.xx, xy = pt.xy;    double minXX = Math.min(xx1, xx2);  double maxXX = Math.max(xx1, xx2);  double minYY = Math.min(yy1, yy2);  double maxYY = Math.max(yy1, yy2);    return xx >= minXX && xx <= maxXX && xy >= minYY && xy <= maxYY;  }  } |

**3. Реалізуйте методи класу Triangle: Конструктор, що має як параметри координати трьох вершин (клас Point). Переконайтеся, що ці точки належать до вершин трикутника. Перевірте, що створений трикутник існує і не вироджений. Реалізуйте метод double area() – повертає площу трикутника. Реалізуйте метод Point centroid() – повертає центроїд трикутника.**

Спершу я імпортував два пакети, java.util.Scanner та java.util.InputMismatchException, перший дозволяє отримувати введення користувача, а другий потрібен для обробки помилок вводу, якщо користувач введе некоректні дані. Потім я створив головний клас програми Task03, в середині якого створив метод main, після чого я створив об'єкт Scanner для отримання введення від користувача. Далі я викликаю метод InpPoint класу Point тричі для введення координат трьох вершин трикутника. Після введення координат вершин я створюю об'єкт Triangle з цими вершинам. Потім виводяться різні властивості трикутника, такі як площа, периметр, центроїд, висоти, радіуси вписаного та описаного кола, кути трикутника та перевірка на те, чи є трикутник прямокутним. В середині всього цього описано два класи, Point та Triangle. Клас Point відповідає за представлення точок у двовимірному просторі, його конструктор отримує дві координати: x та y і ініціалізує поля об'єкта значеннями цих координат. Статичний метод InpPoint використовується для введення координат точок з консолі за допомогою об'єкта Scanner. Параметр ptName використовується для вказівки користувачу, для якої вершини відбувається введення координат, програма в циклі очікує введення користувача, якщо введене значення не відповідає типу double, викидається виняток InputMismatchException і користувач повторно запрошується ввести значення. Після введення обох координат створюється новий об'єкт класу Point з введеними координатами, який потім повертається методом. Метод toString перевизначений для представлення об'єкта класу Point у вигляді рядка, він повертає рядок, який містить значення обох координат. Клас Triangle відповідає за представлення трикутника та обчислення його різних характеристик на основі трьох вершин. Конструктор Triangle приймає три об'єкти типу Point, які представляють вершини трикутника, при створенні об'єкта класу Triangle перевіряється його валідність за допомогою приватного методу isValidTriangle, цей метод перевіряє, чи не співпадають вершини, і чи не лежать вони на одній прямій за допомогою методу areCollinear, якщо трикутник невалідний, викидається виняток IllegalArgumentException, для цього використовується формула площі трикутника, яку можна обчислити за допомогою детермінанта, якщо площа трикутника, утвореного цими трьома точками, дорівнює нулю або дуже близька до нуля, а саме менша за задане значення 1e-9, то ці точки лежать на одній прямій і метод areCollinear поверне true, в іншому випадку - false. Далі йде метод area, який обчислює площу трикутника за допомогою формули Герона. Спочатку обчислюються довжини сторін трикутника, використовуючи метод distance для обчислення відстаней між вершинами трикутника. Потім обчислюється полупериметр трикутника, який дорівнює сумі довжин всіх сторін, поділеній на 2 і в кінці за допомогою формули Герона, обчислюється площа трикутника, використовуючи довжини сторін та полупериметр. Метод perim обчислює периметр трикутника, який є сумою довжин його сторін, для цього метод викликає метод distance, щоб обчислити відстані між кожною парою сусідніх вершин, після цього периметр обчислюється як сума ціх сторін і повертається як результат. Метод height обчислює висоту трикутника з вершини, яка задається параметром xx. Для цього використовується формула: h = (2 \* S) / a , де h - висота трикутника, S - його площа, а a - довжина сторони, до якої проводиться висота. Для цього спочатку створюється масив point, що містить всі вершини трикутника і далі визначаються дві інші вершини трикутника, які не є вершиною xx, це робиться за допомогою арифметичних операцій, щоб обробити вершини в правильному порядку. Потім обчислюється довжина стороної сторони, використовуючи метод distance. В кінці висота обчислюється за вищезазначеною формулою, використовуючи площу трикутника, отриману за допомогою методу area() і довжину протилежної сторони, після чого висота повертається як результат. Далі йде метод radIC, який обчислює радіус вписаного кола в трикутник. Це виконується за допомогою формули: r = (2 \* S) / P, де r - радіус вписаного кола, S - площа трикутника, а P - його периметр. Спочатку метод викликає метод area для обчислення площі трикутника та метод perim для отримання його периметру, після чого за допомогою вищезазначеної формули обчислюється радіус вписаного кола, який повертається як результат. Після цього йде метод radCC, який обчислює радіус описаного кола навколо трикутника. Це виконується за допомогою формули: r = (a \* b \* c) / 4 \* S, де r - радіус описаного кола, a, b і c - довжини сторін трикутника, а S - його площа. Спочатку обчислюються довжини сторін трикутника за допомогою методу distance, потім обчислюється площа трикутника за допомогою методу area і пісдя чого, за допомогою цих значень і формули обчислюється радіус описаного кола, який повертається як результат. Метод right перевіряє, чи є трикутник прямокутним, для цього обчислюються кути між сторонами трикутника, а потім перевіряється, чи хоча б один з цих кутів дорівнює 90 градусам. Спочатку обчислюються кути між сторонами трикутника, використовуючи метод angl, який обчислює кут між двома сторонами трикутника за їхніми довжинами. Після цього для кожного кута перевіряється, чи він дорівнює 90 градусам, з точністю до десяткових дробів, використовуючи умову (Math.abs(angl - 90) < 1e-9). Якщо хоча б один з кутів дорівнює 90 градусам, метод поверне true, що означає, що трикутник є прямокутним, в іншому випадку метод поверне false, що означає, що трикутник не є прямокутним. Наступним методом йде метод angl, який обчислює кут між двома сторонами трикутника, які визначені трьома заданими вершинами. Цей метод використовує три сторони трикутника, визначені за допомогою відстаней між вершинами, які обчислюються за допомогою методу distance. Далі метод використовує косинусну теорему для обчислення кута. Косинус кута ∠A обчислюється за формулою: cos(∠A) = (b^2 + c^2 – a^2) / (2 \* b \* c). Після цього використовується обернена косинусна функція для обчислення самого кута в радіанах, метод Math.acos() повертає кут в радіанах і в кінці за допомогою методу Math.toDegrees() кут перетворюється з радіанів у градуси та повертається в якості результату. Перед останнім методом у класі Segment є метод distance, який обчислює відстань між двома точками, він використовує формулу відстані між двома точками в прямокутній системі координат, яка виглядає так: d = sqrt((x2 - x1)^2 + (y2 – y1)^2). У методі дана формула реалізується за допомогою використання методу Math.sqrt(), який обчислює квадратний корінь, та функцій Math.pow(), яка підносить значення до потрібної степені. Таким чином цей метод обраховує відстань між двома точками та повертає результат. Останнім йде метод cent(), який обчислює центроїд трикутника, який є середнім значенням координат вершин трикутника. У методі для кожної координати x та y обчислюється середнє арифметичне значення координат відповідних вершин трикутника за допомогою формул cx = (x1 + x2 + x3) / 3 та cy = (y1 + y2 + xy) / 3. Далі ці середні значення координат використовуються для створення нового об'єкта класу Point, який представляє центроїд трикутника і після чого, цей об'єкт повертається як результат виклику методу. Приклад роботи програми і порівняння вірності результатів представлено на рисунках 3.1 та 3.2.

Рисунок 3.1 - Приклад роботи програми

Рисунок 3.2 - Перевірка вірності відповіді програми на GeoGebra 2D

|  |
| --- |
| import java.util.Scanner;  import java.util.InputMismatchException;    public class Task03 {  public static void main(String[] args) {  Scanner InpScn = new Scanner(System.in);  Point xp1 = Point.InpPoint(InpScn, "вершини 1");  Point xp2 = Point.InpPoint(InpScn, "вершини 2");  Point xp3 = Point.InpPoint(InpScn, "вершини 3");    Triangle triang = new Triangle(xp1, xp2, xp3);  System.out.println("Площа трикутнику: " + triang.area());  System.out.println("Периметр трикутнику: " + triang.perim());  System.out.println("Центроїд трикутнику: " + triang.cent());    for (int xx = 0; xx < 3; xx++) {  System.out.println("Висота від вершини " + (xx + 1) + ": " + triang.height(xx));  }    System.out.println("Радіус вписаного кола: " + triang.radIC());  System.out.println("Радіус описаного кола: " + triang.radCC());    System.out.println("Кути трикутника:");  System.out.println("Кут 1: " + triang.angl(xp1, xp2, xp3));  System.out.println("Кут 2: " + triang.angl(xp2, xp1, xp3));  System.out.println("Кут 3: " + triang.angl(xp2, xp3, xp1));    System.out.println("Це прямокутний трикутник? " + triang.right());  }  }    class Point {  double xx, xy;    Point(double xx, double xy) {  this.xx = xx;  this.xy = xy;  }    static Point InpPoint(Scanner InpScn, String ptName) {  while (true) {  try {  System.out.print("Введіть значення X для " + ptName + ": ");  double xx = InpScn.nextDouble();  System.out.print("Введіть значення Y для " + ptName + ": ");  double xy = InpScn.nextDouble();  return new Point(xx, xy);  } catch (InputMismatchException e) {  System.out.println("Було введено некоректне значення, спробуйте ще раз.");  InpScn.nextLine();  }  }  }    @Override  public String toString() {  return "(" + xx + "; " + xy + ")";  }  }    class Triangle {  Point xp1, xp2, xp3;    Triangle(Point xp1, Point xp2, Point xp3) {  if (!isValidTriangle(xp1, xp2, xp3)) {  throw new IllegalArgumentException("Такий трикутник не може існувати");  }  this.xp1 = xp1;  this.xp2 = xp2;  this.xp3 = xp3;  }    private boolean isValidTriangle(Point xp1, Point xp2, Point xp3) {  return !xp1.equals(xp2) && !xp2.equals(xp3) && !xp3.equals(xp1) && !areCollinear(xp1, xp2, xp3);  }    private boolean areCollinear(Point xp1, Point xp2, Point xp3) {  return Math.abs((xp2.xy - xp1.xy) \* (xp3.xx - xp1.xx) - (xp3.xy - xp1.xy) \* (xp2.xx - xp1.xx)) < 1e-9;  }    double area() {  double xs1 = distance(xp1, xp2);  double xs2 = distance(xp2, xp3);  double xs3 = distance(xp3, xp1);  double xp = (xs1 + xs2 + xs3) / 2.0;  return Math.sqrt(xp \* (xp - xs1) \* (xp - xs2) \* (xp - xs3));  }    double perim() {  double xs1 = distance(xp1, xp2);  double xs2 = distance(xp2, xp3);  double xs3 = distance(xp3, xp1);  return xs1 + xs2 + xs3;  }    double height(int xx) {  Point[] point = {xp1, xp2, xp3};  Point oPoint = point[(xx + 1) % 3];  Point aPoint = point[(xx + 2) % 3];  double res = distance(oPoint, aPoint);  return 2 \* area() / res;  }      double radIC() {  return 2 \* area() / perim();  }    double radCC() {  double xs1 = distance(xp1, xp2);  double xs2 = distance(xp2, xp3);  double xs3 = distance(xp3, xp1);  return (xs1 \* xs2 \* xs3) / (4 \* area());  }    boolean right() {  double angl1 = angl(xp1, xp2, xp3);  double angl2 = angl(xp2, xp1, xp3);  double angl3 = angl(xp3, xp1, xp2);    return (Math.abs(angl1 - 90) < 1e-9) || (Math.abs(angl2 - 90) < 1e-9) || (Math.abs(angl3 - 90) < 1e-9);  }    public double angl(Point xp1, Point xp2, Point xp3) {  double xs1 = distance(xp1, xp2);  double xs2 = distance(xp2, xp3);  double xs3 = distance(xp3, xp1);    double cAngl = (Math.pow(xs1, 2) + Math.pow(xs2, 2) - Math.pow(xs3, 2)) / (2 \* xs1 \* xs2);  double aRad = Math.acos(cAngl);  return Math.toDegrees(aRad);    }    private double distance(Point p1, Point p2) {  return Math.sqrt(Math.pow(p2.xx - p1.xx, 2) + Math.pow(p2.xy - p1.xy, 2));  }    Point cent() {  double cx = (xp1.xx + xp2.xx + xp3.xx) / 3.0;  double cy = (xp1.xy + xp2.xy + xp3.xy) / 3.0;  return new Point(cx, cy);  }  } |

**Висновок:** Під час виконання цієї лабораторної роботи я покращив свої навички об'єктно-орієнтованим програмування. В процесі виконання завдань, я покращив безліч своїх навичок та вмінь, включно із навичками програмування на мові Java. У першому завданні я реалізував метод intersection у класі Line, що повертає точку перетину двох ліній і під час цього я покращив свої знання про поняття класу, методу, об'єкту, та їх взаємодії. Друге завдання дозволило мені поглибити розуміння роботи з класами та методами, я реалізував різноманітні методи для класу Segment, такі як конструктор, обчислення довжини сегмента, знаходження середньої точки, точки перетину з іншим відрізком та інше. Під час цього я покращив свої навички робота із арифметичними операціями, умовними операторами та вміння працювати з об'єктами. У третьому завданні я створив клас Triangle, в якому реалізував методи для роботи з трикутником, такі як обчислення площі, периметру, центроїда, висот, радіусів вписаного та описаного кола, а також кутів трикутника. Це дозволило мені краще закріпити свої покращені навички об'єктно-орієнтованого програмування та математичних концепцій, які лежать в основі цих обчислень. Загалом під час виконання цієї лабораторної роботи я розширив мої знання та навички в області об'єктно-орієнтованого програмування та покращив свої вміння роботи з класами, методами та об'єктами.