**Лабораторна робота №4**

**Тема:** Абстрактні класи та інтерфейси

**Мета роботи**: Використовуючи теоретичне підґрунтя абстракті класи та інтерфейси у мові Java, виконати дії що будуть вказано в завданні до лабораторної роботи.

**Хід роботи:**

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

**1. Створіть три класи Triangle, Quadrilateral, Circle, що розширяють абстрактний клас Figure. Реалізуйте конструктор для кожного з класів. Для трикутника конструктор у якості параметра приймає три вершини класу Point. Для чотирикутника конструктор у якості параметра приймає чотири вершини класу Point. Для кола конструктор у якості параметра приймає координати центу класу Point та радіус типу double. Конструктор кожної з фігур має перевіряти, що фігура є не виродженою. Реалізуйте методи підрахунку площі для кожної фігури. Реалізуйте метод toString() для кожної з фігур який виводить текст у вказаному вигляді: Трикутник - Triangle[A(x,y) B(x,y) C(x,y)], Чотирикутник - Quadrilateral[A(x,y) B(x,y) C(x,y) D(x,y)], Коло - Circle[(x,y) Radius]. Реалізуйте метод Point centroid(). Повернути центроїд фігури. Іншими словами, це має бути "центроїд площі".**

**Main.java**

Спочатку я імпортував два пакети, java.awt.\* і java.util.Scanner. Перший пакет забезпечує доступ до класів для роботи з графічним інтерфейсом користувача, а другий - дозволяє отримати введення від користувача через консоль. Далі я створив головний клас програми з назвою Main, у цьому класі є основний метод main, який є точкою входу у програму. В цьому методі я створив об'єкт класу Scanner для отримання введення від користувача через консоль. Також я оголосив змінну exit типу boolean, яка вказує на те, чи потрібно виходити з програми, чи ні. Далі йде цикл while, який буде виконуватися, доки змінна exit буде false. У цьому циклі виводиться повідомлення, що пропонує користувачеві обрати фігуру для обчислень. Користувач має вибрати одну з фігур: трикутник (T), чотирикутник (Q), коло (C) або вийти з програми (X). Це виводиться за допомогою методу System.out.println(), який виводить повідомлення у консоль. Після цього програма очікує введення користувача за допомогою методу inpScn.nextLine(), який зчитує рядок введений користувачем. Цей рядок переводиться в верхній регістр за допомогою методу toUpperCase(), щоб забезпечити роботу програми незалежно від регістру введеного користувачем символу. Далі використовується оператор switch, який дозволяє виконати різні дії в залежності від значення змінної cmd, яка визначається вибором користувача. Якщо користувач вибрав T, програма викликає метод startTriangle(inpScn), який починає обчислення параметрів трикутника. Аналогічно, якщо користувач вибрав Q, програма викликає метод startQuadrilateral(inpScn), а якщо C - startCircle(inpScn). Якщо користувач вибрав X, змінна exit встановлюється в true, і програма виходить з циклу. Якщо ж користувач ввів будь-який інший символ, програма виводить повідомлення про неправильний вибір і пропонує спробувати ще раз. Після цього йде рядок, який закриває об'єкт Scanner, щоб уникнути витоку ресурсів. Далі в коді я створюю два методи, startTriangle() і startQuadrilateral(), які відповідають за виклик обчислень параметрів трикутника та чотирикутника відповідно. У методі startTriangle спочатку виводиться повідомлення, яке просить користувача ввести координати точок для трикутника. Для зберігання координат кожної точки використовується масив Point[] points = new Point[3]. Цей масив містить три об'єкти класу Point, які представляють собою координати точок на площині. Після цього виконується цикл for, який повторюється три рази, так, як у трикутника три точки. У кожній ітерації циклу викликається метод seekPoint(inpScn, "Введіть координати точки " + (char)('A' + xx) + ": "), який запитує у користувача координати кожної точки, де (char)('A' + xx) використовується для виведення літери A, B або C в залежності від ітерації циклу. Після того, як користувач ввів координати всіх трьох точок, створюється об'єкт типу Triangle за допомогою конструктора new Triangle(points[0], points[1], points[2]). Цей конструктор приймає координати трьох точок і створює об'єкт типу Triangle, який представляє собою трикутник з вказаними координатами. Після створення об'єкта трикутника викликається метод drawFigur(xFigure), який відповідає за виведення інформації про фігуру та її параметрів. Аналогічно у методі startQuadrilateral спочатку виводиться повідомлення для введення координат точок для чотирикутника, виконується цикл for, який повторюється чотири рази, так як у чотирикутника чотири точки. Кожною ітерацією циклу користувач вводить координати кожної точки, а потім створюється об'єкт типу Quadrilateral за допомогою конструктора new Quadrilateral(points). Цей конструктор приймає масив точок і створює об'єкт типу Quadrilateral, який представляє собою чотирикутник з вказаними координатами. Після цього викликається метод drawFigur(xFigure), який відповідає за виведення інформації про фігуру та її параметри. Обидва методи мають обробку винятку NullPointerException, який може виникнути, якщо користувач буде вводити невірні значення, у такому випадку виводиться повідомлення про повернення в меню. У методі startCircle() реалізована логіка для початку обчислень параметрів кола. Спочатку користувачеві виводиться повідомлення, що просить його ввести координати центру кола. Для збереження цих координат використовується метод seekPoint, який повертає об'єкт типу Point, що містить координати x та y. Далі користувачеві пропонується ввести радіус кола, це виконується за допомогою методу checkInpt, який зчитує введене користувачем значення і перетворює його у тип даних double. Цей метод використовується для забезпечення правильного введення дійсного числа, а також обробки помилок, якщо користувач введе некоректне значення. Після того, як координати центру та радіус були введені користувачем, виконується перевірка на те, чи є радіус додатнім числом. Якщо так, то створюється об'єкт типу Circle за допомогою введених даних, а саме центру та радіуса і викликається метод drawFigur, який відповідає за виведення інформації про фігуру та її параметрів. Якщо радіус виявиться негативним або рівним нулю, виводиться відповідне повідомлення. Метод seekPoint() використовується для запитування координат точки від користувача. Спочатку виводиться повідомлення message, яке інформує користувача, яку саме координату він має ввести, а саме x або y. Після цього метод checkInpt використовується для зчитування введеного користувачем значення. Метод checkInpt() використовується для зчитування введеного користувачем значення типу double з консолі. Він використовує цикл while, щоб продовжувати запитувати користувача, доки він не введе коректне значення. Якщо користувач введе некоректне значення, наприклад якийсь текст, метод перехопить виняток NumberFormatException і виведе повідомлення про помилку, пропонуючи користувачеві ввести правильне число знову. Далі я описав методи, які використовуються для обчислення мінімальних значень координат x та y серед всіх точок фігури, а також для здійснення зсуву фігури на площині так, щоб всі координати були додатніми. Метод gtMinX() приймає об'єкт фігури xFigure і повертає мінімальне значення координати x серед усіх точок цієї фігури. Спочатку змінна minX ініціалізується значенням Double.MAX\_VALUE, яке є максимально можливим значенням типу double. Потім проходиться по всіх точках фігури і порівнюється координата x кожної точки з поточним мінімальним значенням minX. Якщо координата x точки менша за поточне minX, то ця координата стає новим мінімальним значенням minX і на завершення метод повертає обчислене мінімальне значення minX. Аналогічно, метод gtMinY() здійснює обчислення мінімального значення координати серед усіх точок фігури. Він виконує аналогічний алгоритм, але для координати y. Метод shiftFigur() використовується для зсуву фігури на площині так, щоб всі координати були додатніми. Спочатку обчислюються мінімальні значення координат x та y за допомогою методів gtMinX та gtMinY відповідно, а потім перевіряється, чи є хоча б одна з цих мінімальних значень від'ємною. Якщо так, то розраховуються значення зсуву shiftX та shiftY, щоб зробити всі координати додатними. Потім для кожної точки фігури обчислюються нові координати shiftX та shiftY, додаванням їх до оригінальних координат значення зсуву. Далі я описав метод, який відповідає за візуалізацію фігури. Він викликається з параметром xFigure, який є об'єктом фігури, яку потрібно візуалізувати. Спочатку викликається метод shiftFigur(), який здійснює зсув фігури на площині, щоб всі її координати були додатніми. Потім виводяться інформаційні повідомлення про фігуру, а саме назва фігури (xFigure.gtFName()) та площа (xFigure.calcArea()). Далі обчислюється центроїд фігури за допомогою методу calcCentroid(), результат зберігається у змінній centroid і виводиться у відповідний рядок. Після цього визначається масштаб для відображення фігури на екрані. Масштаб обчислюється так, щоб фігура була поміщена в область розміром 400x400 пікселів або більше, якщо це необхідно. Обчислюються максимальні та мінімальні значення координат x та y фігури за допомогою методів gtMaxX, gtMinX, gtMaxY, gtMinY. Далі виконується обчислення розмірів вікна для візуалізації фігури з урахуванням масштабу. Якщо мінімальне значення координати x або y менше за нуль, то розміри вікна коригуються так, щоб фігура могла вміститися цілком в область відображення. Далі створюється об'єкт Frame, який використовується для візуалізації фігури. Встановлюються його розміри за допомогою setSize(), назва вікна встановлюється на "Візуалізація фігури". На цьому вікні встановлюється graphBoard, який є класом, що відображає фігуру на площині, з параметрами xFigure, scale та centroid. В кінці вікно візуалізації стає видимим для користувача за допомогою setVisible(true). Далі йдуть методи для обчислення різних параметрів фігур: центроїду для будь-якої фігури та максимального розміру для кожного типу фігури. Метод calcCentroid(Figure xFigure) приймає об'єкт фігури xFigure і обчислює центроїду цієї фігури. Для цього проходиться по всім точкам фігури та сумує їх координати x та y. Потім ці суми діляться на кількість точок фігури, щоб отримати середнє значення координат x та y, які і представляють координати центроїду. Після чого створюється новий об'єкт Point з обчисленими значеннями центроїду і він повертається як результат. Далій йдуть методи gtMaxX() і gtMaxY(), які використовуються для обчислення максимальних значень координат x та y для кожного типу фігури, яка передається методу в якості аргументу. У методі gtMaxX() спочатку перевіряється тип фігури за допомогою умовного оператора instanceof. Якщо xFigure є екземпляром класу Triangle, то обчислюється максимальне значення x серед усіх вершин трикутника, використовуючи метод Math.max(). Якщо xFigure є екземпляром класу Quadrilateral, то обчислюється максимальне значення x серед усіх точок чотирикутника. Якщо xFigure є екземпляром класу Circle, то обчислюється максимальне значення x, яке є сумою координати x центра кола та радіусу. У методі gtMaxY() виконується аналогічний алгоритм, але для максимального значення координати y. Замість x використовується y для кожної точки фігури. Якщо xFigure не є екземпляром жодного з перерахованих класів, методи повертають 0. Це робиться для запобігання виникнення помилок, коли фігура не відповідає жодному з перерахованих типів. Далі я створив клас graphBoard, який є вікном для відображення фігури та її параметрів на графічному інтерфейсі. Він успадковує клас Panel для створення власного відображення. У конструкторі graphBoard передаються об'єкт фігури xFigure, масштаб для малювання scale та координати центроїду centroid, які зберігаються у відповідних полях класу. Метод paint() викликається автоматично при малюванні панелі і використовується для відображення фігури та її параметрів на екрані. Спочатку викликається метод super.paint(g), який виконує стандартну функцію малювання панелі. Далі створюється об'єкт Graphics2D з параметром g, щоб мати доступ до більш розширених можливостей малювання. Викликається метод draw() фігури, який передає графічну панель та масштаб для малювання. Після цього малюються мітки для кожної точки фігури з їх координатами x та y. Кожна мітка виводиться на екран з відповідним індексом точки та її координатами. Також на панелі виводяться значення площі фігури та координати центроїду.

|  |
| --- |
| import java.awt.\*;  import java.util.Scanner;    public class Main {  public static void main(String[] args) {  Scanner inpScn = new Scanner(System.in);  boolean exit = false;    while (!exit) {  System.out.println("Оберіть фігуру:");  System.out.println("T - Трикутник");  System.out.println("Q - Чотирикутник");  System.out.println("C - Коло");  System.out.println("X - Вихід");  System.out.print("Введіть команду: ");  String cmd = inpScn.nextLine().toUpperCase();    switch (cmd) {  case "T":  startTriangle(inpScn);  break;  case "Q":  startQuadrilateral(inpScn);  break;  case "C":  startCircle(inpScn);  break;  case "X":  exit = true;  break;  default:  System.out.println("Невідома команда.");  break;  }  }  inpScn.close();  }    private static void startTriangle(Scanner inpScn) {  System.out.println("Введіть координати точок для Трикутника:");  Point[] points = new Point[3];  try {  for (int xx = 0; xx < 3; xx++) {  points[xx] = seekPoint(inpScn, "Введіть координати точки " + (char)('A' + xx) + ": ");  }  Figure xFigure = new Triangle(points[0], points[1], points[2]);  drawFigur(xFigure);  } catch (NullPointerException e) {  System.out.println("Вас повернуто в меню вибору команд.");  }  }      private static void startQuadrilateral(Scanner inpScn) {  System.out.println("Введіть координати точок для Чотирикутника:");  Point[] points = new Point[4];  try {  for (int xx = 0; xx < 4; xx++) {  points[xx] = seekPoint(inpScn, "Введіть координати точки " + (char)('A' + xx) + ": ");  }  Figure xFigure = new Quadrilateral(points);  drawFigur(xFigure);  } catch (NullPointerException e) {  System.out.println("Вас повернуто в меню вибору команд.");  }  }      private static void startCircle(Scanner inpScn) {  try {  Point center = seekPoint(inpScn, "Введіть координати центру для Кола: ");  double radius = checkInpt(inpScn, "Введіть радіус для Кола: ");  Figure xFigure = new Circle(center, radius);  drawFigur(xFigure);  } catch (NullPointerException e) {  System.out.println("Вас повернуто в меню вибору команд.");  }  }    private static Point seekPoint(Scanner inpScn, String message) {  System.out.println(message);  double xx = checkInpt(inpScn, "Введіть координату x: ");  double xy = checkInpt(inpScn, "Введіть координату y: ");  return new Point(xx, xy);  }    private static double checkInpt(Scanner inpScn, String message) {  while (true) {  System.out.print(message);  try {  return Double.parseDouble(inpScn.nextLine());  } catch (NumberFormatException e) {  System.out.println("Неправильне значення, введіть число.");  }  }  }    private static double gtMinX(Figure xFigure) {  double minX = Double.MAX\_VALUE;  for (Point p : xFigure.gtPoints()) {  if (p.originX < minX) {  minX = p.originX;  }  }  return minX;  }    private static double gtMinY(Figure xFigure) {  double minY = Double.MAX\_VALUE;  for (Point p : xFigure.gtPoints()) {  if (p.originY < minY) {  minY = p.originY;  }  }  return minY;  }    private static void shiftFigur(Figure xFigure) {  double minX = gtMinX(xFigure);  double minY = gtMinY(xFigure);    if (minX < 0 || minY < 0) {  double shiftX = Math.max(-minX, 0);  double shiftY = Math.max(-minY, 0);    for (Point p : xFigure.gtPoints()) {  p.shiftX = p.originX + shiftX;  p.shiftY = p.originY + shiftY;  }  }  }    private static void drawFigur(Figure xFigure) {  shiftFigur(xFigure);  System.out.println(xFigure.gtFName() + " створено: " + xFigure);  System.out.println("Площа " + xFigure.gtFName() + ": " + xFigure.calcArea());  Point centroid = calcCentroid(xFigure);  System.out.println("Центроїд " + xFigure.gtFName() + ": " + centroid.originX + ", " + centroid.originY);    double scale = Math.max(400.0 / gtMaxX(xFigure), 1);    double maxX = gtMaxX(xFigure);  double minX = gtMinX(xFigure);  double maxY = gtMaxY(xFigure);  double minY = gtMinY(xFigure);    int windowWidth = (int) ((maxX - minX) \* scale) + 20;  int windowHeight = (int) ((maxY - minY) \* scale) + 20;    if (minX < 0) {  windowWidth += (int) (-minX \* scale);  }    if (minY < 0) {  windowHeight += (int) (-minY \* scale);  }    Frame frame = new Frame("Візуалізація фігури");  frame.setSize(windowWidth, windowHeight);  frame.add(new graphBoard(xFigure, scale, centroid));  frame.setVisible(true);  }    private static Point calcCentroid(Figure xFigure) {  double totalX = 0, totalY = 0;  for (Point p : xFigure.gtPoints()) {  totalX += p.originX;  totalY += p.originY;  }  double centroidX = totalX / xFigure.gtPoints().length;  double centroidY = totalY / xFigure.gtPoints().length;  return new Point(centroidX, centroidY);  }    private static double gtMaxX(Figure xFigure) {  if (xFigure instanceof Triangle) {  return Math.max(((Triangle) xFigure).A.originX, Math.max(((Triangle) xFigure).B.originX, ((Triangle) xFigure).C.originX));  } else if (xFigure instanceof Quadrilateral) {  return Math.max(((Quadrilateral) xFigure).points[0].originX, Math.max(((Quadrilateral) xFigure).points[1].originX, Math.max(((Quadrilateral) xFigure).points[2].originX, ((Quadrilateral) xFigure).points[3].originX)));  } else if (xFigure instanceof Circle) {  return ((Circle) xFigure).center.originX + ((Circle) xFigure).radius;  }  return 0;  }    private static double gtMaxY(Figure xFigure) {  if (xFigure instanceof Triangle) {  return Math.max(((Triangle) xFigure).A.originY, Math.max(((Triangle) xFigure).B.originY, ((Triangle) xFigure).C.originY));  } else if (xFigure instanceof Quadrilateral) {  return Math.max(((Quadrilateral) xFigure).points[0].originY, Math.max(((Quadrilateral) xFigure).points[1].originY, Math.max(((Quadrilateral) xFigure).points[2].originY, ((Quadrilateral) xFigure).points[3].originY)));  } else if (xFigure instanceof Circle) {  return ((Circle) xFigure).center.originY + ((Circle) xFigure).radius;  }  return 0;  }  }    class graphBoard extends Panel {  private Figure xFigure;  private double scale;  private Point centroid;    graphBoard(Figure xFigure, double scale, Point centroid) {  this.xFigure = xFigure;  this.scale = scale;  this.centroid = centroid;  }    @Override  public void paint(Graphics g) {  super.paint(g);  Graphics2D graph = (Graphics2D) g;    xFigure.draw(graph, scale);    graph.setColor(Color.BLACK);  for (int xx = 0; xx < xFigure.gtPoints().length; xx++) {  Point p = xFigure.gtPoints()[xx];  graph.drawString(String.format("%c(%.2f, %.2f)", (char)('A' + xx), p.originX, p.originY), (int)(p.shiftX \* scale), (int)(p.shiftY \* scale));  }    graph.drawString("Площа: " + xFigure.calcArea(), 10, 20);    graph.drawString("Центроїд: (" + String.format("%.2f", centroid.originX) + ", " + String.format("%.2f", centroid.originY) + ")", 10, 40);  }  } |

**Figure.java**

Цей файл визначає два класи: Figure і Point, клас Figure є абстрактним і визначає загальні методи та властивості для всіх фігур. Методи calcArea(), centroid(), gtFName() та draw() визначаються як абстрактні, що означає, що будь-який підклас Figure повинен надати свою реалізацію цих методів. Метод gtPoints() також визначається як абстрактний і призначений для отримання точок, які складають фігуру. Клас Point представляє точку у двовимірному просторі. Він має дві пари координат: originX і originY, які представляють початкові координати точки і shiftX та shiftY, які можуть змінюватися для зміщення точки для правильного відображення на графічній панелі. Конструктор класу приймає початкові координати точки і встановлює їх значення, як значення originX та originY, а також встановлює початкові значення shiftX та shiftY. Ці два класи створюють базові будівельні блоки для подальшого використання при створенні різних фігур та їх відображенні. Клас Figure визначає загальні властивості та методи, які можуть бути спеціалізовані в підкласах, таких як Circle чи Quadrilateral, тоді як клас Point визначає точку, яка може бути використана для визначення розміщення вершин фігур.

|  |
| --- |
| import java.awt.\*;    abstract class Figure {  abstract double calcArea();  abstract Point centroid();  abstract String gtFName();  abstract void draw(Graphics2D g, double scale);  abstract Point[] gtPoints();  }    class Point {  double originX, originY;  double shiftX, shiftY;    Point(double originX, double originY) {  this.originX = originX;  this.originY = originY;  this.shiftX = originX;  this.shiftY = originY;  }  } |

**Circle.java**

Клас Circle є підкласом Figure і представляє коло у двовимірному просторі. Основні характеристики кола - це центр та радіус. Конструктор класу Circle приймає параметри center та radius. Перевіряється, що радіус більше за нуль, інакше виводиться повідомлення про помилку. Метод gtPoints() перевизначається з батьківського класу Figure і повертає масив точок, що складається з однієї точки - центру кола. Метод calcArea() обчислює площу кола за формулою π \* r^2, де π - це число пі. Метод centroid() також перевизначається з батьківського класу і повертає центр кола. Метод toString() перевизначається для того, щоб представити об'єкт у вигляді рядка. У цьому випадку, виводиться рядок, що містить координати центра кола та його радіус. Метод gtFName() також перевизначається для надання назви фігури - "Коло". Метод draw() відповідає за малювання кола у вікні g з вказаним масштабом scale. У першому рядку методу обчислюються координати центра кола та його радіуса відповідно до вказаного масштабу. Координати центру кола множаться на масштаб, щоб відобразити їх у правильному масштабі на екрані. Радіус також масштабується. Далі встановлюється колір лінії, яка використовується для обводки кола, та викликається метод drawOval, який малює зовнішній контур кола за допомогою зазначених координат та розмірів. Після цього встановлюється колір заливки кола. Викликається метод fillOval, який зафарбовує внутрішню область кола в зазначений колір. Далі встановлюється колір та малюється точка в центрі кола.

|  |
| --- |
| import java.awt.\*;    class Circle extends Figure {  Point center;  double radius;    Circle(Point center, double radius) {  if (radius <= 0) {  System.out.println("Невірний радіус, введіть додатнє значення.");  return;  }  this.center = center;  this.radius = radius;  }    @Override  Point[] gtPoints() {  return new Point[]{center};  }    @Override  double calcArea() {  return Math.PI \* radius \* radius;  }    @Override  Point centroid() {  return center;  }    @Override  public String toString() {  return String.format("Коло[(%.2f, %.2f) %.2f]", center.originX, center.originY, radius);  }    @Override  String gtFName() {  return "Коло";  }    @Override  void draw(Graphics2D g, double scale) {  int x = (int)(center.shiftX \* scale);  int y = (int)(center.shiftY \* scale);  int r = (int)(radius \* scale);  g.setColor(Color.BLACK);  g.drawOval(x - r, y - r, 2 \* r, 2 \* r);  g.setColor(new Color(255, 192, 203));  g.fillOval(x - r, y - r, 2 \* r, 2 \* r);  g.setColor(Color.BLUE);  g.fillOval(x - 3, y - 3, 6, 6);  }  } |

**Quadrilateral.java**

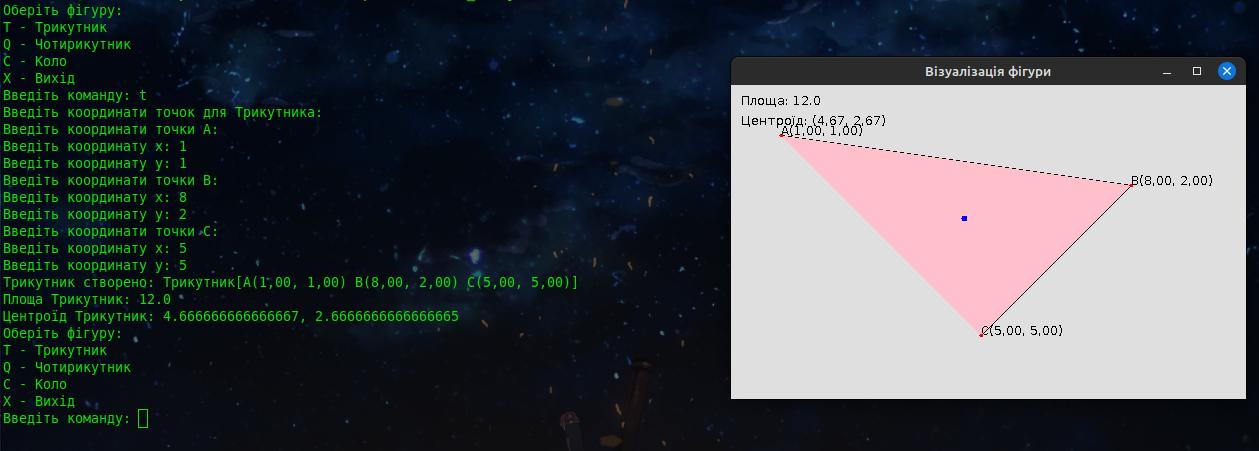
Клас Quadrilateral є підкласом Figure і відповідає за представлення чотирикутника у двовимірному просторі. Основними характеристиками чотирикутника є його вершини. Конструктор класу Quadrilateral приймає змінну кількість параметрів типу Point, які представляють вершини чотирикутника. Перевіряється, чи кількість переданих точок дорівнює 4 і чи не є чотирикутник виродженим, а саме, чи всі точки лежать на одній прямій або одна з точок повторюється. Якщо виявляються помилки, виводиться відповідне повідомлення про помилку. Приватний метод validQuad() перевіряє, чи передані точки утворюють чотирикутник, що не є виродженим. Для цього перевіряється, що всі точки чотирикутника не збігаються між собою і що площа чотирикутника не дорівнює нулю. Внутрішній цикл обчислює суму площі чотирикутника, використовуючи координати кожної пари сусідніх вершин і застосовуючи формулу площі трикутника, утвореного цими вершинами. Якщо площа чотирикутника дорівнює нулю, це означає, що всі вершини лежать на одній прямій і чотирикутник вироджений. Далій йде метод gtPoints() перевизначений з батьківського класу Figure і повертає масив вершин чотирикутника. Метод calcArea() обчислює площу чотирикутника. Для цього використовується формула площі чотирикутника за координатами його вершин, яка базується на формулі площі трикутника. Сума площ трикутників, утворених сусідніми вершинами чотирикутника, ділиться на 2. Метод centroid() обчислює координати центроїду чотирикутника. Для цього обчислюється середнє значення координат x та y усіх вершин чотирикутника. Кожна координата сумується та ділиться на кількість вершин, щоб отримати середнє значення, а потім створюється нова точка з обчисленими координатами центроїду і вона повертається. Далі йде метод toString() перевизначений для створення текстового представлення чотирикутника, яке містить назву фігури та координати його вершин. Для цього використовується об'єкт StringBuilder, у якому додаються інформаційні рядки про кожну вершину чотирикутника. Потім отриманий рядок завершується заміною останнього пробілу та додаванням квадратної дужки, після чого виконується повернення рядку. Метод gtFName() перевизначений для повернення назви фігури "Чотирикутник". Метод draw() відповідає за малювання чотирикутника на графічній панелі g з вказаним масштабом scale. Спочатку визначаються масиви xPoints та yPoints, які містять координати x та y кожної вершини чотирикутника, масштабовані до відповідного масштабу. Для цього використовується потік Stream, який відображає кожну вершину в відповідну координату, перетворену у ціле число типу int. Потім отримані масиви передаються методу drawPolygon, який малює контур чотирикутника. Після цього встановлюється колір заливки чотирикутника і викликається метод fillPolygon, який зафарбовує внутрішню область чотирикутника. Наступний блок коду малює червоні точки на місці кожної вершини чотирикутника. Для кожної вершини викликається метод fillOval, який малює коло заданого розміру. Останній блок коду малює синю точку в центрі чотирикутника, використовуючи координати центроїду, які обчислюються за допомогою методу centroid().

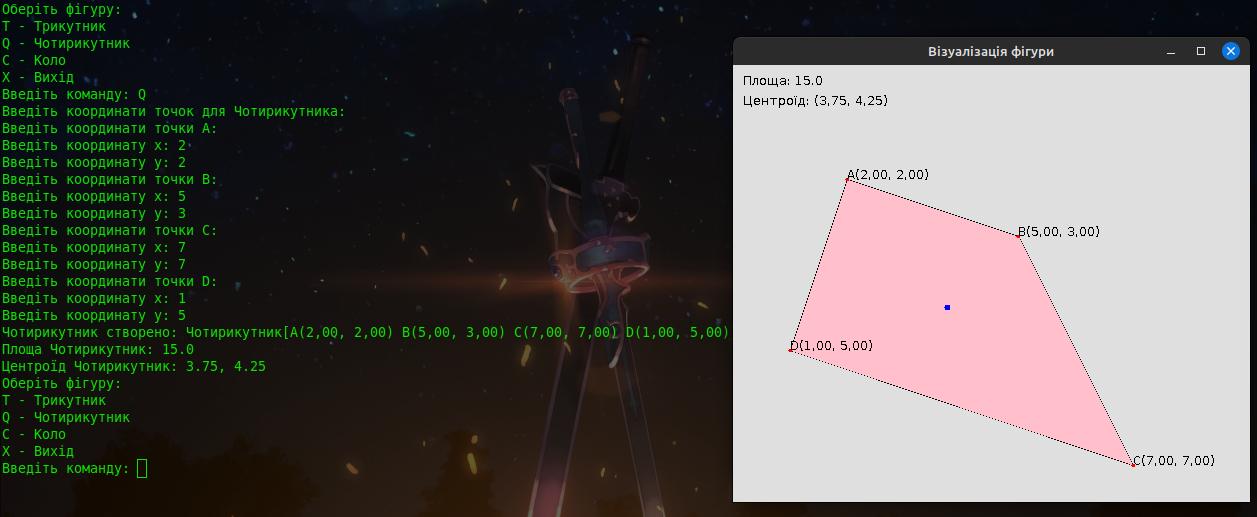
|  |
| --- |
| import java.awt.\*;  import java.util.stream.Stream;    class Quadrilateral extends Figure {  Point[] points;    Quadrilateral(Point... points) {  if (points.length != 4 || !validQuad(points)) {  System.out.println("Чотирикутник вироджений, введіть вірні точки.");  return;  }  this.points = points;  }    private boolean validQuad(Point[] points) {  for (int xx = 0; xx < points.length - 1; xx++) {  for (int xy = xx + 1; xy < points.length; xy++) {  if (points[xx].originX == points[xy].originX && points[xx].originY == points[xy].originY) {  return false;  }  }  }  double area = 0;  for (int xx = 0; xx < 4; xx++) {  int nextxx = (xx + 1) % 4;  area += (points[xx].originX \* points[nextxx].originY - points[nextxx].originX \* points[xx].originY);  }  return Math.abs(area) > 0;  }    @Override  Point[] gtPoints() {  return points;  }    @Override  double calcArea() {  double area = 0;  for (int xx = 0; xx < 4; xx++) {  int nextxx = (xx + 1) % 4;  area += (points[xx].originX \* points[nextxx].originY - points[nextxx].originX \* points[xx].originY);  }  return Math.abs(area) / 2.0;  }    @Override  Point centroid() {  double centroidX = 0, centroidY = 0;  for (Point p : points) {  centroidX += p.shiftX;  centroidY += p.shiftY;  }  centroidX /= 4.0;  centroidY /= 4.0;  return new Point(centroidX, centroidY);  }      @Override  public String toString() {  StringBuilder sb = new StringBuilder("Чотирикутник[");  for (int xx = 0; xx < 4; xx++) {  sb.append((char)('A' + xx)).append(String.format("(%.2f, %.2f) ", points[xx].originX, points[xx].originY));  }  sb.setLength(sb.length() - 1);  sb.append("]");  return sb.toString();  }    @Override  String gtFName() {  return "Чотирикутник";  }    @Override  void draw(Graphics2D g, double scale) {  int[] xPoints = Stream.of(points).mapToInt(p -> (int)(p.shiftX \* scale)).toArray();  int[] yPoints = Stream.of(points).mapToInt(p -> (int)(p.shiftY \* scale)).toArray();  g.setColor(Color.BLACK);  g.drawPolygon(xPoints, yPoints, 4);  g.setColor(new Color(255, 192, 203));  g.fillPolygon(xPoints, yPoints, 4);  for (Point p : points) {  g.setColor(Color.RED);  g.fillOval((int)(p.shiftX \* scale) - 2, (int)(p.shiftY \* scale) - 2, 4, 4);  }  Point centroid = centroid();  g.setColor(Color.BLUE);  g.fillOval((int)(centroid.shiftX \* scale) - 3, (int)(centroid.shiftY \* scale) - 3, 6, 6);  }  } |

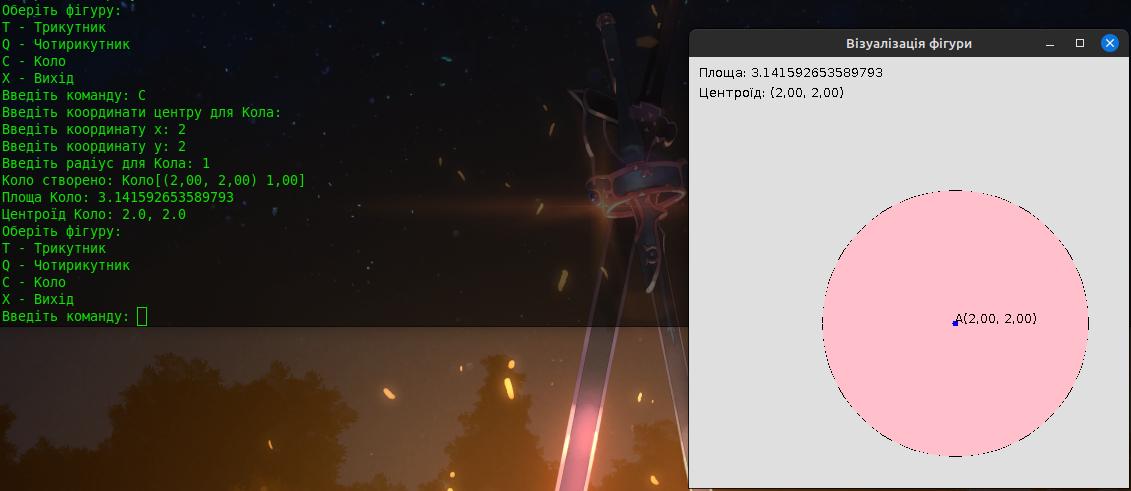
**Triangle.java**

В цьому файлі визначається клас Triangle, який є підкласом класу Figure. Клас Triangle представляє трикутник у двовимірному просторі та містить три вершини трикутника, позначені як A, B і C. Конструктор класу Triangle приймає три точки - вершини трикутника A, B і C. Також перевіряється правильність трикутника за допомогою приватного методу validTriangle(), цей метод перевіряє, чи є три точки непаралельними та чи неповторюються вони, а також чи не лежать вони на одній прямій, за допомогою перевірки умови, що сума площі трикутників, утворених парою точок трикутника, не рівна нулю. Якщо трикутник вироджений, то виводиться повідомлення про помилку. Якщо трикутник правильний, то вершини ініціалізуються із відповідними значеннями. Далі йде метод gtPoints(), який повертає масив з трьох точок, які утворюють вершини трикутника. Метод calcArea() обчислює площу трикутника за допомогою формули площі Гаусса, яка використовується для обчислення площі будь-якого трикутника за координатами його вершин. Метод centroid() обчислює центроїд трикутника, який є точкою перетину медіан трикутника. Для цього обчислюються середні значення координат x і y для кожної вершини. Метод toString() повертає рядок, що представляє трикутник у текстовому форматі, де кожна вершина позначена своїми координатами. Метод gtFName() повертає назву фігури, яка використовується для ідентифікації типу фігури при виведенні інформації про неї. Далі йде метод draw() для класу Triangle. Цей метод відповідає за візуалізацію трикутника на графічній панелі. У методі спершу створюються масиви xPoints та yPoints, які містять координати вершин трикутника у відповідних масштабованих величинах, це виконується шляхом множення кожної координати на масштаб scale. Далі йде малювання зовнішньої оболонки трикутника та його зафарбування з використанням методів drawPolygon() та fillPolygon() відповідно. Для цього використовуються отримані масиви координат вершин. Після цього виконується малювання червоних точок, які позначають вершини трикутника. Для цього використовується цикл for, який перебирає всі вершини трикутника. Кожна вершина малюється у вигляді червоного кола за допомогою методу fillOval(). В кінці малюється синя точка центроїду трикутника за допомогою методу fillOval().

|  |
| --- |
| import java.awt.\*;    class Triangle extends Figure {  Point A, B, C;    Triangle(Point A, Point B, Point C) {  if (!validTriangle(A, B, C)) {  System.out.println("Неправильний трикутник, введіть вірні точки.");  return;  }    this.A = A;  this.B = B;  this.C = C;  }    private boolean validTriangle(Point A, Point B, Point C) {  if (A.originX == B.originX && A.originY == B.originY || A.originX == C.originX && A.originY == C.originY || B.originX == C.originX && B.originY == C.originY) {  return false;  }    return (A.originX \* (B.originY - C.originY) + B.originX \* (C.originY - A.originY) + C.originX \* (A.originY - B.originY) != 0);  }    @Override  Point[] gtPoints() {  return new Point[]{A, B, C};  }    @Override  double calcArea() {  return Math.abs((A.originX \* (B.originY - C.originY) + B.originX \* (C.originY - A.originY) + C.originX \* (A.originY - B.originY)) / 2.0);  }    @Override  Point centroid() {  double centroidX = (A.shiftX + B.shiftX + C.shiftX) / 3.0;  double centroidY = (A.shiftY + B.shiftY + C.shiftY) / 3.0;  return new Point(centroidX, centroidY);  }    @Override  public String toString() {  return String.format("Трикутник[A(%.2f, %.2f) B(%.2f, %.2f) C(%.2f, %.2f)]", A.originX, A.originY, B.originX, B.originY, C.originX, C.originY);  }    @Override  String gtFName() {  return "Трикутник";  }    @Override  void draw(Graphics2D g, double scale) {  int[] xPoints = {(int)(A.shiftX \* scale), (int)(B.shiftX \* scale), (int)(C.shiftX \* scale)};  int[] yPoints = {(int)(A.shiftY \* scale), (int)(B.shiftY \* scale), (int)(C.shiftY \* scale)};  g.setColor(Color.BLACK);  g.drawPolygon(xPoints, yPoints, 3);  g.setColor(new Color(255, 192, 203));  g.fillPolygon(xPoints, yPoints, 3);  g.setColor(Color.RED);  for (Point p : new Point[]{A, B, C}) {  g.fillOval((int)(p.shiftX \* scale) - 2, (int)(p.shiftY \* scale) - 2, 4, 4);  }  Point centroid = centroid();  g.setColor(Color.BLUE);  g.fillOval((int)(centroid.shiftX \* scale) - 3, (int)(centroid.shiftY \* scale) - 3, 6, 6);  }  } |

Рисунок 1 - Приклад роботи програми при обрахунку трикутника

Рисунок 2 - Приклад роботи програми при обрахунку чотирикутника

Рисунок 3 - Приклад роботи програми при обрахунку кола

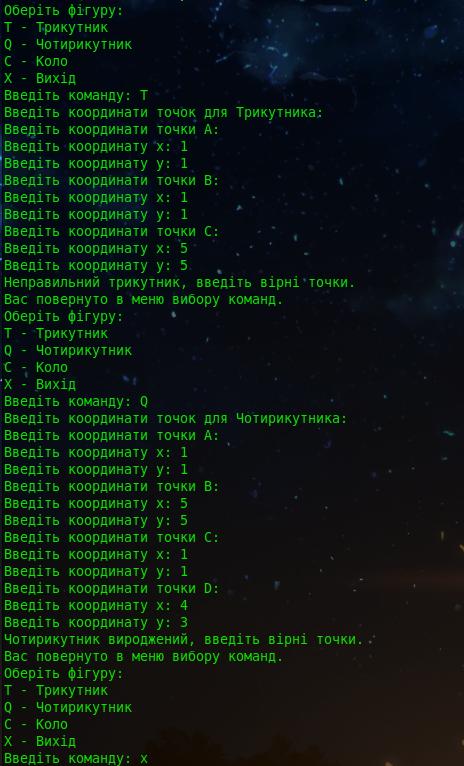


Рисунок 4 - Приклад роботи програми при введенні помилкових значень

**Висновок:** Під час виконання цієї лабораторної роботи я покращив свої знання про абстрактні класи та інтерфейси у мові програмування Java. Під час цієї лабораторної роботи я створив три класи: Triangle, Quadrilateral та Circle, які розширювали абстрактний клас Figure. Кожен з цих класів мав свої особливості та функціонал, що відображав їх геометричні характеристики. Також я покращив свої навички використання ключового слова extends для створення спадковості в Java та краще зрозумів, як можна організувати ієрархію класів для покращення структури програми. Кожен клас фігури мав свої методи, що обчислювали площу фігури та центроїд, використовуючи математичні формули. Під час виконання цієї лабораторної роботи я також покращив свої навички роботи з масивами, циклами, конструкторами та методами у Java. Також за допомогою використання вбудованих класів та методів для роботи з графічними об'єктами у пакеті java.awt, я покращив свої навички програмування та краще ознайомився із роботою пакетів у мові Java. Загалом ця лабораторна робота покращила мої навички програмування на мові Java та дозволила мені краще зрозуміти концепції абстракції та спадковості, а також краще закріпити отримані знання про роботу з абстрактними класами та інтерфейсами на практиці.