LinesApp.py

```python
# Основний клас застосунку, який відповідає за взаємодію з користувачем
import sys

from LineClassifier import LineClassifier
from SegmentLine import SegmentLine
from SlopeInterceptLine import SlopeInterceptLine
from utils import InvalidLineException


class LinesApp:
    def __init__(self):
        pass

    def run(self):
        print("=== КАЛЬКУЛЯТОР ПЕРЕТИНІВ ПРЯМИХ ===")
        print("Введіть параметри для трьох прямих:")
        print("\nДля прямих 1 і 2 (формат: x/a + y/b = 1):")
        while True:
            try:
                # Зчитування параметрів для прямих 1 та 2
                a1 = self._input_number("Введіть значення a1 для прямої 1: ")
                b1 = self._input_number("Введіть значення b1 для прямої 1: ")
                a2 = self._input_number("Введіть значення a2 для прямої 2: ")
                b2 = self._input_number("Введіть значення b2 для прямої 2: ")
                # Зчитування параметрів для прямої 3
                print("\nДля прямої 3 (формат: y = kx + b):")
                k3 = self._input_number("Введіть значення k3 для прямої 3: ")
                b3 = self._input_number("Введіть значення b3 для прямої 3: ")

                # Створення об'єктів прямих (з перевіркою коректності параметрів)
                line1 = SegmentLine(a1, b1)
                line2 = SegmentLine(a2, b2)
                line3 = SlopeInterceptLine(k3, b3)

                # Класифікація взаємного розташування прямих
                classifier = LineClassifier(line1, line2, line3)
                result = classifier.classify()

                print("\nРезультат:")
                print(result)
                break  # Якщо все пройшло коректно, завершуємо цикл вводу
            except InvalidLineException as e:
                print(f"\nПомилка: {e}")
                print("Спробуйте ввести дані ще раз.\n")
            except ValueError:
                print("\nНекоректний формат вводу. Будь ласка, введіть числове значення.\n")

    def _input_number(self, prompt):
        # Зчитування числа з консолі з обробкою винятків
        while True:
            user_input = input(prompt)
```

```
            # Дозволяємо завершення роботи, якщо користувач вводить 'e'
            if user_input.strip().lower() == 'e':
                print("Завершення роботи програми.")
                sys.exit(0)
            try:
                value = float(user_input)
                if value < -126 or value > 126:
                    raise ValueError("Значення повинно бути в межах від -126 до 126.")
                return value
            except ValueError:
                print("Некоректний формат вводу. Спробуйте ще раз.")

if __name__ == "__main__":
    app = LinesApp()
    app.run()
```

line.py

```
from abc import ABC, abstractmethod

# Абстрактний клас, який задає інтерфейс для представлення прямої
class Line(ABC):
    @abstractmethod
    def get_coefficients(self):
        """
        Повертає коефіцієнти (A, B, C) загального рівняння прямої: Ax + By + C = 0
        """
        pass
```

SegmentLine.py

```
from line import Line
from utils import is_zero, InvalidLineException


class SegmentLine(Line):
    def __init__(self, a, b):
        if is_zero(a) or is_zero(b):
            raise InvalidLineException \
                ("Для відрізкового представлення прямої параметри a та b не можуть бути
нульовими.")
        # Перевірка на допустимість значень a та b
        if a < -126 or a > 126 or b < -126 or b > 126:
            raise InvalidLineException("Значення a та b повинні бути в межах від -126 до 126.")
        self.a = a
        self.b = b

    def get_coefficients(self):
        # Отримаємо рівняння: x/a + y/b = 1  <=>  b*x + a*y - a*b = 0
        A = self.b
        B = self.a
```

```
        C = -self.a * self.b
        return (A, B, C)

    def __str__(self):
        return f"SegmentLine(a={self.a}, b={self.b})"
```

SlopeInterceptLine.py

```
# Клас для прямої, заданої рівнянням: y = kx + b
from line import Line
from utils import is_zero, InvalidLineException


class SlopeInterceptLine(Line):
    def __init__(self, k, b):
        # За умовою роботи b не може бути нульовим
        if is_zero(b):
            raise InvalidLineException("Для прямої з кутовим коефіцієнтом параметр b не може
бути нульовим.")
        # Перевірка на допустимість значень k та b
        if k < -126 or k > 126 or b < -126 or b > 126:
            raise InvalidLineException("Значення k та b повинні бути в межах від -126 до 126.")

        self.k = k
        self.b = b

    def get_coefficients(self):
        # Перетворення: y = kx + b  <=>  kx - y + b = 0
        A = self.k
        B = -1
        C = self.b
        return (A, B, C)

    def __str__(self):
        return f"SlopeInterceptLine(k={self.k}, b={self.b})"
```

LineClassiier.py

```
from utils import compute_intersection, is_zero, normalize_point, InvalidLineException


class LineClassifier:
    def __init__(self, line1, line2, line3):
        self.line1 = line1
        self.line2 = line2
        self.line3 = line3

    def classify(self):
        # Отримуємо коефіцієнти для кожної прямої
        coeffs = [
```

```
            self.line1.get_coefficients(),
            self.line2.get_coefficients(),
            self.line3.get_coefficients()
        ]

        # Перевірка на співпадання: якщо всі коефіцієнти пропорційні
        if self._are_coincident(coeffs[0], coeffs[1]) and \
           self._are_coincident(coeffs[0], coeffs[2]) and \
           self._are_coincident(coeffs[1], coeffs[2]):
            raise InvalidLineException("Усі 3 прямі не можуть співпадати")

        # Перевіряємо, чи є прямі з однаковими коефіцієнтами
        if coeffs[0] == coeffs[1]:
            raise InvalidLineException("Прямі 1 і 2 не можуть бути однаковими.")
        if coeffs[0] == coeffs[2]:
            raise InvalidLineException("Прямі 1 і 3 не можуть бути однаковими.")
        if coeffs[1] == coeffs[2]:
            raise InvalidLineException("Прямі 2 і 3 не можуть бути однаковими.")

        # Обчислюємо точки перетину для кожної пари (якщо існують)
        intersections = []
        pairs = [(0, 1), (0, 2), (1, 2)]
        for i, j in pairs:
            pt = compute_intersection(coeffs[i], coeffs[j])
            if pt is not None:
                intersections.append(pt)

        # Видаляємо дублікати (з урахуванням допуску)
        unique_points = []
        for pt in intersections:
            if not any(self._points_equal(pt, upt) for upt in unique_points):
                unique_points.append(pt)

        num_points = len(unique_points)
        if num_points == 0:
            return "Прямі не перетинаються"
        elif num_points == 1:
            point_str = normalize_point(unique_points[0])
            return f"Прямі перетинаються в одній точці ({point_str})"
        elif num_points == 2:
            point1_str = normalize_point(unique_points[0])
            point2_str = normalize_point(unique_points[1])
            return f"Прямі перетинаються в двох точках ({point1_str}) та ({point2_str})"
        elif num_points == 3:
            point1_str = normalize_point(unique_points[0])
            point2_str = normalize_point(unique_points[1])
            point3_str = normalize_point(unique_points[2])
            s = f"({point1_str}), ({point2_str}), ({point3_str})"
            return f"Прямі перетинаються в трьох точках {s}"
        else:
            return f"Прямі перетинаються в {num_points} точках"

    def _are_coincident(self, coeffs1, coeffs2):
```

```
        A1, B1, C1 = coeffs1
        A2, B2, C2 = coeffs2

        # Для перевірки співпадання обчислюємо відношення коефіцієнтів (якщо знаменник
ненульовий)
        ratios = []
        if not is_zero(A2):
            ratios.append(A1 / A2)
        if not is_zero(B2):
            ratios.append(B1 / B2)
        if not is_zero(C2):
            ratios.append(C1 / C2)
        # Якщо всі отримані відношення практично рівні, прямі співпадають
        return all(is_zero(r - ratios[0]) for r in ratios)

    def _points_equal(self, pt1, pt2, tol=1e-6):
        x1, y1 = pt1
        x2, y2 = pt2
        return is_zero(x1 - x2, tol) and is_zero(y1 - y2, tol)
```

utils.py

```
# Функція порівняння чисел з допуском
def is_zero(value, eps=1e-8):
    return abs(value) < eps

# Виняток для некоректного задання прямої
class InvalidLineException(Exception):
    pass

# Функція для обчислення точки перетину двох прямих за їх коефіцієнтами
def compute_intersection(coeffs1, coeffs2):
    A1, B1, C1 = coeffs1
    A2, B2, C2 = coeffs2
    D = A1 * B2 - A2 * B1
    if is_zero(D):
        return None
    # Коректне обчислення:
    x = (B1 * C2 - B2 * C1) / D
    y = (A2 * C1 - A1 * C2) / D
    return (x, y)

def normalize_point(point):
    """
    Нормалізує кортеж (x, y) у рядок формату "(x, y)" з форматуванням до 6 знаків після коми.
    Якщо значення знаходиться дуже близько до нуля, воно формується як 0.000000.
    """
    x, y = point
    # Якщо значення практично рівні нулю, встановити 0.0
    if abs(x) < 1e-8:
        x = 0.0
    if abs(y) < 1e-8:
```

```
        y = 0.0
    return f"{x:.6f}, {y:.6f}"
```

test.py

```
import pytest
import SegmentLine, SlopeInterceptLine, LineClassifier
from app import LinesApp
from utils import InvalidLineException

#############################################################################
# Тести для допустимих класів ситуацій (коректні дані)
#############################################################################

#  Прямі не перетинаються (усі три паралельні, але не співпадають (можуть співпасти тільки дві))
@pytest.mark.parametrize("a1,b1,a2,b2,k3,b3", [
    (-126, -126, -125, -125, -1, -125),
    (-120, -120, -118, -118, -1, -119),
    (10, -10, 20, -20, 1, 15),
    (126, 126, 125, 125, -1, 125),
    (120, 120, 118, 118, -1, 119),
    (120, -120, 18, -18, 1, 119)
])
def test_parallel(a1, b1, a2, b2, k3, b3):
    """
    Якщо всі три прямі паралельні (але не співпадають), має повернутися:
      "Прямі не перетинаються"
    """
    line1 = SegmentLine.SegmentLine(a1, b1)
    line2 = SegmentLine.SegmentLine(a2, b2)
    line3 = SlopeInterceptLine.SlopeInterceptLine(k3, b3)
    classifier = LineClassifier.LineClassifier(line1, line2, line3)
    result = classifier.classify()
    print(result)
    assert result == "Прямі не перетинаються"


# Прямі перетинаються в одній точці (конкурентні прямі)
@pytest.mark.parametrize("a1, b1, a2, b2, k3, b3, common_point", [
    # Test 1 (Граничний нижній): використання значень, близьких до -126
    (-126, 10, -125, 10, -126, 10, (0, 10)),
    # Test 2 (Граничний верхній): використання значень, близьких до 126
    (126, 10, 125, 10, 126, 10, (0, 10)),
    # Test 3 (Типові значення): середній діапазон
    (20, 10, 30, 10, 1, 10, (0, 10)),
    # Test 4 (Комбінація граничного і середнього): перша з граничних, друга з типових
    (-126, 10, 30, 10, 0, 10, (0, 10)),
])
def test_concurrent(a1, b1, a2, b2, k3, b3, common_point):
    """
    Тест для конкурентних прямих - всі три перетинаються в одній точці.
```

```
    Очікуване: рядок результату містить фразу "Прямі перетинаються в одній точці"
    і має містити координати спільної точки, наприклад, "(0.000000, 10.000000)".
    """
    line1 = SegmentLine.SegmentLine(a1, b1)
    line2 = SegmentLine.SegmentLine(a2, b2)
    line3 = SlopeInterceptLine.SlopeInterceptLine(k3, b3)
    classifier = LineClassifier.LineClassifier(line1, line2, line3)
    result = classifier.classify()

    print(result)

    # Очікуємо, що рядок містить ключову фразу для конкурентних прямих:
    expected_str = "Прямі перетинаються в одній точці"
    assert expected_str in result, f"Результат '{result}' не містить '{expected_str}'"

    # Перевіряємо, що спільна точка співпадає з очікуваною (за форматуванням до 6 знаків після
коми)
    common_str = f"({common_point[0]:.6f}, {common_point[1]:.6f})"
    assert common_str in result, f"Спільна точка '{common_str}' не знайдена в результаті
'{result}'"


# Прямі перетинаються в двох точках
@pytest.mark.parametrize("a1,b1,a2,b2,k3,b3,expected", [
    # Test 1: Граничний випадок з великими від'ємними значеннями
    (-126, -126, -125, -125, 0, 50, ((-176.000000, 50.000000), (-175.000000, 50.000000))),
    # Test 2: Типовий випадок з позитивними значеннями
    (100, 100, 90, 90, 1, 10, ((45.000000, 55.000000), (40.000000, 50.000000))),
    # Test 3: Типовий випадок з малими значеннями
    (3, 4, 6, 8, 2, 5, ((-0.300000, 4.400000), (0.900000, 6.800000))),
    # Test 4: Спеціальний випадок - комбінація негативних і позитивних параметрів
    (-50, 20, -40, 16, 0.5, -20, ((400.000000, 180.000000), (360.000000, 160.000000))),
    # Test 5: Граничний випадок із використанням максимальних значень
    (126, -126, 120, -120, 0, -10, ((116.000000, -10.000000), (110.000000, -10.000000))),
    # Test 6: Спеціальний випадок із змішанням малих від'ємних значень
    (-10, 50, -8, 40, 2, -20, ((-23.333333, -66.666667), (-20.000000, -60.000000))),
])
def test_two_intersections(a1, b1, a2, b2, k3, b3, expected):
    """
    Тести, де одна пара прямих паралельна, а третя їх перетинає.
    Результатом має бути повідомлення з двома різними точками перетину.
    """
    line1 = SegmentLine.SegmentLine(a1, b1)
    line2 = SegmentLine.SegmentLine(a2, b2)
    line3 = SlopeInterceptLine.SlopeInterceptLine(k3, b3)
    classifier = LineClassifier.LineClassifier(line1, line2, line3)
    result = classifier.classify()

    print(result)

    expected_str = "Прямі перетинаються в двох точках"
    assert expected_str in result, f"Результат '{result}' не містить '{expected_str}'"
```

```
    point1, point2 = expected
    common_str1 = f"({point1[0]:.6f}, {point1[1]:.6f})"
    assert common_str1 in result, f"Спільна точка '{common_str1}' не знайдена в результаті
'{result}'"
    common_str2 = f"({point2[0]:.6f}, {point2[1]:.6f})"
    assert common_str2 in result, f"Спільна точка '{common_str2}' не знайдена в результаті
'{result}'"

# Прямі перетинаються в трьох точках
@pytest.mark.parametrize("a1,b1,a2,b2,k3,b3,expected", [
    # Test 1: Граничний випадок з великими від'ємними значеннями
    (-126, -126, -125, -124, 1, -50,
     ((-38.000000, -88.000000),
      (-37.148594, -87.148594),
      (-250.00000, 124.000000))),

    # Test 2: Типовий випадок з позитивними значеннями
    (100, 90, 80, 100, 1, 50,
     ((28.571429, 64.285714),
      (21.052632, 71.052632),
      (22.222222, 72.222222))),

    # Test 3: Типовий випадок з малими значеннями
    (3, 4, 4, 3, 2, 1,
     ((1.714286, 1.714286),
      (0.727273, 2.454545),
      (0.900000, 2.800000))),

    # Test 4: Спеціальний випадок - комбінація негативних і позитивних параметрів
    (-20, 50, 30, -60, 0.5, 10,
     ((-20.000000, 0.000000),
      (46.666667, 33.333333),
      (-220.000000, -500.000000))),

    # Test 5: Граничний випадок із використанням максимальних значень
    (126, 30, 125, 40, -1, 5,
     ((122.093023, 0.930233),
      (-32.812500, 37.812500),
      (-51.470588, 56.470588))),

    # Test 6: Спеціальний випадок із змішанням малих від'ємних значень та граничними
    (-126, 126, 50, -126, 0.75, -10,
     ((165.789474, 291.789474),
      (-544.000000, -418.000000),
      (65.536723, 39.152542))),
])
def test_three_intersections(a1, b1, a2, b2, k3, b3, expected):
    """
    Тести, де кожна пара прямих перетинається в унікальну точку (3 різні точки перетину).
    Результатом має бути повідомлення, яке містить фразу "Прямі перетинаються в трьох точках"
    та відформатований перелік трьох точок (точність 6 знаків після коми).
    """
    # Створюємо об'єкти прямої
```

```
    line1 = SegmentLine.SegmentLine(a1, b1)
    line2 = SegmentLine.SegmentLine(a2, b2)
    line3 = SlopeInterceptLine.SlopeInterceptLine(k3, b3)
    classifier = LineClassifier.LineClassifier(line1, line2, line3)
    result = classifier.classify()

    print(result)

    # Перевірка, що в результаті є фраза для трьох точок
    expected_phrase = "Прямі перетинаються в трьох точках"
    assert expected_phrase in result, f"Результат '{result}' не містить очікувану фразу
'{expected_phrase}'"


    # Формуємо рядки для кожної очікуваної точки
    point1, point2, point3 = expected
    common_str1 = f"({point1[0]:.6f}, {point1[1]:.6f})"
    assert common_str1 in result, f"Спільна точка '{common_str1}' не знайдена в результаті
'{result}'"
    common_str2 = f"({point2[0]:.6f}, {point2[1]:.6f})"
    assert common_str2 in result, f"Спільна точка '{common_str2}' не знайдена в результаті
'{result}'"
    common_str3 = f"({point3[0]:.6f}, {point3[1]:.6f})"
    assert common_str3 in result, f"Спільна точка '{common_str3}' не знайдена в результаті
'{result}'"

def test_exit_program_most_left(monkeypatch):
    """
    Тест для перевірки виходу з програми при введенні перших двох чисел -126 та виходу.
    Імітуємо ввід: "-126", "-126", "e"
    """
    inputs = iter(['-126', '-126', 'e'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    with pytest.raises(SystemExit):
        app = LinesApp.LinesApp()
        app.run()


def test_exit_program_left(monkeypatch):
    """
    Тест для перевірки виходу з програми при введенні чисел: "-100", "-100", "-120", "-99" та
команди виходу "e".
    """
    inputs = iter(['-100', '-100', '-120', '-99', 'e'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    with pytest.raises(SystemExit):
        app = LinesApp.LinesApp()
        app.run()


def test_exit_program_average(monkeypatch):
    """
```

```
    Тест для перевірки виходу з програми при введенні чисел: "-10", "-10", "10", "-9" та
команди виходу "e".
    """
    inputs = iter(['-10', '-10', '10', '-9', 'e'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    with pytest.raises(SystemExit):
        app = LinesApp.LinesApp()
        app.run()

def test_exit_program_right(monkeypatch):
    """

    """
    inputs = iter(['120', '120', 'e'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    with pytest.raises(SystemExit):
        app = LinesApp.LinesApp()
        app.run()


def test_exit_program_most_right(monkeypatch):
    """
    """
    inputs = iter(['126', '126', 'e'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    with pytest.raises(SystemExit):
        app = LinesApp.LinesApp()
        app.run()

################################################################################
# Група 1. Тести для відсутності вхідного значення або пустого поля
################################################################################

# Симуляція вводу користувача методом _input_number із застосуванням monkeypatch.
@pytest.mark.parametrize("inputs, expected", [
    (["", "50"], 50.0),           # порожній рядок, потім типове значення
    ["   ", "10"], 10.0),         # лише пробіли, потім "10"
    (["", "126"], 126.0),         # граничне значення з правої межі
    (["", "-126"], -126.0),       # граничне значення з лівої межі
    (["", "0"], 0.0),             # нульове значення — допустиме (якщо це дозволено)
    (["", "100"], 100.0)          # типове значення
])
def test_empty_input(monkeypatch, inputs, expected):
    """
    Перевіряє, що при відсутності вводу (порожній рядок, пробіли) система повторює запит і
    врешті дає коректне числове значення.
    """
    inputs_iter = iter(inputs)
    monkeypatch.setattr("builtins.input", lambda prompt: next(inputs_iter))
    app = LinesApp.LinesApp()
    val = app._input_number("Введіть значення:")
    assert val == expected
```

```python
# Тести для некоректного формату вводу (нечислові значення)
@pytest.mark.parametrize("inputs", [
    ["abc", "50"],              # текст замість числа
    ["!@#$", "100"],
    ["ten", "10"],
    ["12.34.56", "20"],
    ["", "abc", "30"],          # спочатку порожній, потім некоректний текст, потім валідне значення
    ["   ", "not a number", "126"]
])
def test_invalid_format(monkeypatch, inputs):
    """
    Перевіряє, що якщо користувач вводить некоректний формат (нечислове значення),
    функція _input_number продовжує запит, поки не буде отримано число.
    """
    inputs_iter = iter(inputs)
    monkeypatch.setattr("builtins.input", lambda prompt: next(inputs_iter))
    app = LinesApp.LinesApp()
    val = app._input_number("Введіть значення:")
    # Перевіряємо, що отримане значення є числом
    assert isinstance(val, float)

##############################################################################
# Група 2. Тести для числових значень, що не входять до проміжку [-126;126]
##############################################################################

# Припускаємо, що конструктори класів перевіряють, що значення має бути у проміжку [-126;126] та
# кидати ValueError, якщо це не так.

@pytest.mark.parametrize("a, b", [
    (130, 10),      # a більше верхньої межі
    (-130, 10),     # a нижче нижньої межі
    (10, 130),      # b більше верхньої межі
    (10, -130),     # b нижче нижньої межі
    (127, 0.5),     # a трохи перевищує
    (-127, -1)      # a трохи нижче допустимого
])
def test_segmentline_out_of_range(a, b):
    with pytest.raises(InvalidLineException):
        SegmentLine.SegmentLine(a, b)

@pytest.mark.parametrize("k, b", [
    (130, 10),      # k перевищує верхню межу
    (-130, 10),     # k нижче нижньої межі
    (10, 130),      # b перевищує верхню межу (для SlopeInterceptLine параметр b перевіряємо окремо)
    (10, -130),     # b нижче нижньої межі
    (127, 0.5),     # k трохи перевищує
    (-127, -1)      # k трохи нижче
])
def test_slopeinterceptline_out_of_range(k, b):
    with pytest.raises(InvalidLineException):
```

```
        SlopeInterceptLine.SlopeInterceptLine(k, b)

################################################################################
# Група 3. Тести для некоректного задання прямої: a=0 або b=0 (для рівняння x/a + y/b = 1)
################################################################################

@pytest.mark.parametrize("a, b", [
    (0, 10),     # a нульовий
    (0, -10),
    (10, 0),     # b нульовий
    (-10, 0),
    (0, 126),
    (126, 0)
])
def test_segmentline_zero_parameters(a, b):
    with pytest.raises(InvalidLineException):
        SegmentLine.SegmentLine(a, b)

################################################################################
# 4. Тести для некоректного визначення прямих (необчислювані коефіцієнти)
################################################################################

def test_segmentline_both_zero():
    """
    Якщо для прямої виду x/a + y/b = 1 задано a=0 та b=0,
    коефіцієнти розрахувати неможливо – має бути викликаний InvalidLineException.
    """
    with pytest.raises(InvalidLineException):
        SegmentLine.SegmentLine(0, 0)

def test_slopeinterceptline_b_zero():
    """
    Для прямої виду y = kx + b параметр b не може бути 0.
    """
    with pytest.raises(InvalidLineException):
        SlopeInterceptLine.SlopeInterceptLine(1, 0)


################################################################################
# 5. Тест для ситуації, коли всі 3 прямі співпадають
################################################################################

def test_all_three_lines_coincide():
    """
    Якщо всі три прямі задані однаковими параметрами, система вважає це некоректним.
    Припускаємо, що в такому випадку створення класифікатора або виклик classify() кине
винятоk
    з повідомленням, що "Усі 3 прямі не можуть співпадати".
    """
    with pytest.raises(InvalidLineException, match="Усі 3 прямі не можуть співпадати"):
        line1 = SegmentLine.SegmentLine(-100, -100)
        line2 = SegmentLine.SegmentLine(-100, -100)
        line3 = SlopeInterceptLine.SlopeInterceptLine(-1, -100)
```

```
        classifier = LineClassifier.LineClassifier(line1, line2, line3)
        classifier.classify()


################################################################################
# 6. Тести для ситуації, коли дві прямі задані однаковими точками
################################################################################

@pytest.mark.parametrize("x1,y1,x2,y2", [
    (10, 20, 10, 20),
    (-50, 60, -50, 60),
    (126, -126, 126, -126),
    (15, 15, 15, 15),
    (-30, 40, -30, 40)
])
def test_line_by_points_identical(x1, y1, x2, y2):
    """
    Якщо дві прямі задано однаковими, має бути кинутий InvalidLineException.
    Очікується повідомлення, яке містить інформацію про те, що точки співпадають.
    """
    with pytest.raises(InvalidLineException, match=".*однаковими.*"):
        line1 = SegmentLine.SegmentLine(x1, y1)
        line2 = SegmentLine.SegmentLine(x2, y2)
        line3 = SlopeInterceptLine.SlopeInterceptLine(1, 10)
        classifier = LineClassifier.LineClassifier(line1, line2, line3)
        classifier.classify()
```