Лабораторная работа No1 «Графический пользовательский интерфейс в Dart»

Цель работы

Научиться создавать приложения с графическим пользовательским интерфейсом с использованием фреймворка Flutter на языке программирования Dart.

Задание и порядок выполнения

В течение лабораторной работы нужно разработать программу, рисующую на экране мобильного устройства одно из изображений, перечисленных в таблице ниже. Программа должна иметь графический пользовательский интерфейс, через который пользователь может задавать параметры изображения. Изображение должно перерисовываться автоматически при изменении любого параметра. Значения параметров, обозначенных в таблице латинскими буквами, представляют собой неотрицательные целые числа. Когда в описании изображения говорится о выборе цвета, подразумевается выбор из нескольких предопределённых альтернатив (например, красный, зелёный или синий).

Вариант 41

Треугольник со сторонами a, b и c, изображённый таким образом, что сторона a составляет угол альфа с осью OX.

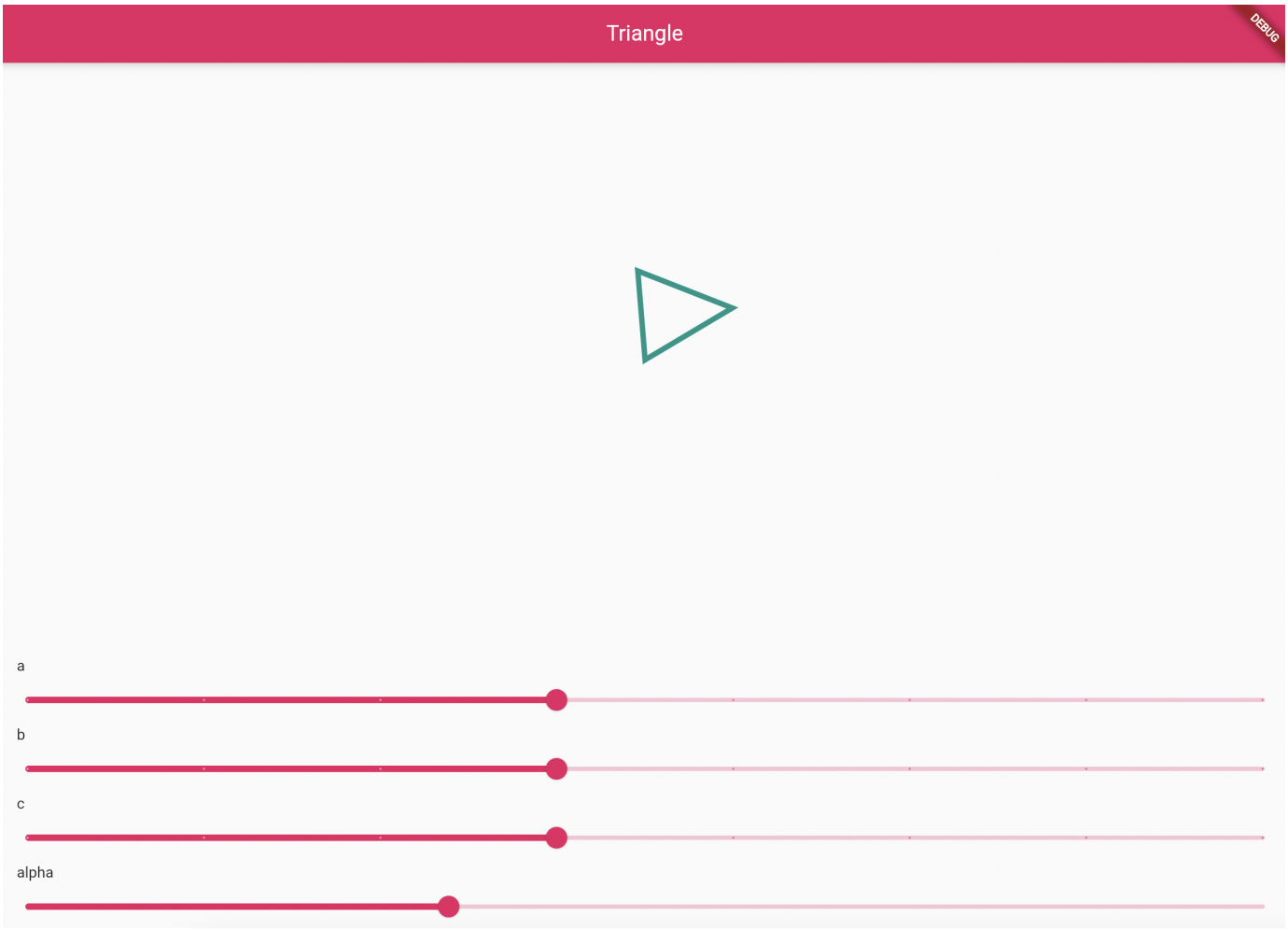
Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

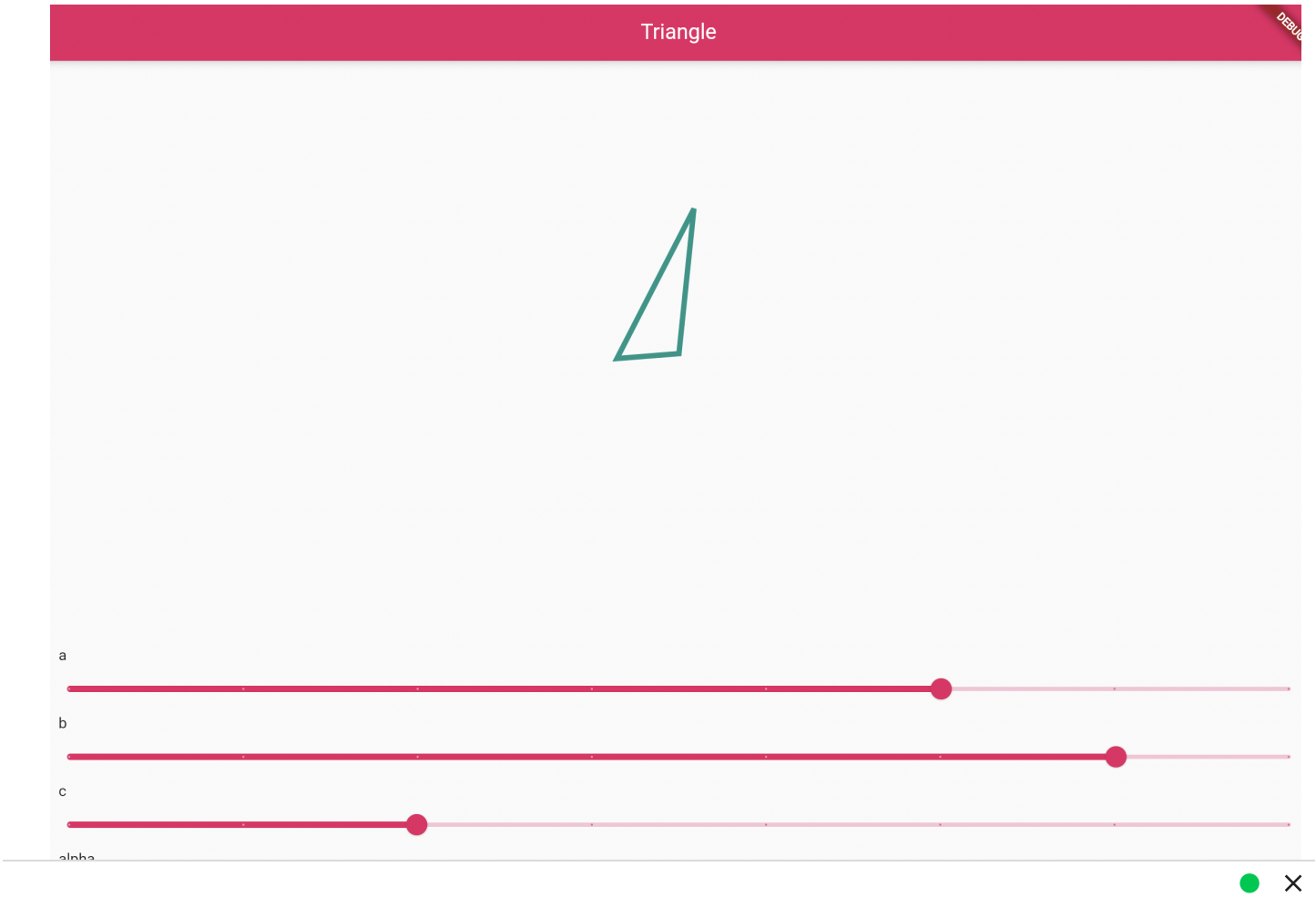
```
import 'dart:math';
import 'dart:ui';
import 'package:flutter/material.dart';
import 'dart:math' as math;

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Custom Painter',
      theme: ThemeData(
        primarySwatch: Colors.pink,
      ),
      home: MyPainter(),
```

```
      );
    }
  }

  class MyPainter extends StatefulWidget {
    @override
    _MyPainterState createState() => _MyPainterState();
  }

  class _MyPainterState extends State<MyPainter> {
    var a = 30.0;
    var b = 30.0;
    var c = 30.0;
    var angle = 0.0;

    @override
    Widget build(BuildContext context) {
      return Scaffold(
        appBar: AppBar(
          title: const Text('Triangle'),
        ),
        body: SafeArea(
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: <Widget>[
              Expanded(
                child: CustomPaint(
                  painter: TrianglePainter(b, c, a, angle),
                  child: Container(),
                ),
              ),
              const Padding(
                padding: EdgeInsets.only(left: 16.0),
                child: Text('a'),
              ),
              Slider(
                value: a,
                min: 30.0,
                max: 180.0,
                label: a.toInt().toString(),
                divisions: 7,
                onChanged: (value) {
                  setState(() {
                    a = value;
                  });
                },
              ),
              const Padding(
                padding: EdgeInsets.only(left: 16.0),
                child: Text('b'),
              ),
              Slider(
                value: b,
                min: 30.0,
                max: 180.0,
                label: b.toInt().toString()
```

```
                label: b.toInt().toString(),
                divisions: 7,
                onChanged: (value) {
                  setState(() {
                    b = value;
                  });
                },
              ),
              const Padding(
                padding: EdgeInsets.only(left: 16.0),
                child: Text('c'),
              ),
              Slider(
                value: c,
                min: 10.0,
                max: 180.0,
                label: c.toInt().toString(),
                divisions: 7,
                onChanged: (value) {
                  setState(() {
                    c = value;
                  });
                },
              ),
              const Padding(
                padding: EdgeInsets.only(left: 16.0),
                child: Text('alpha'),
              ),
              Slider(
                value: angle,
                min: 0.0,
                max: math.pi / 2,
                onChanged: (value) {
                  setState(() {
                    angle = value;
                  });
                },
              ),

            ],
          ),
        ),
      );
    }
  }


  double power(double x, int n) {
    double retval = 1;
    for (int i = 0; i < n; i++) {
      retval *= x;
    }

    return retval;
  }
```

```
// FOR PAINTING POLYGONS
class TrianglePainter extends CustomPainter {
  final double a;
  final double b;
  final double c;
  final double angle;

  TrianglePainter(this.a, this.b, this.c, this.angle);

  @override
  void paint(Canvas canvas, Size size) {
    var paint = Paint()
      ..color = Colors.teal
      ..strokeWidth = 5
      ..style = PaintingStyle.stroke
      ..strokeCap = StrokeCap.round;

    var path = Path();

    double cosB = (power(a, 2) - power(b,2) + power(c, 2))/(2*a*c);

    double angle_b = acos(cosB);
    Offset center = Offset(size.width / 2, size.height / 2);

    path.moveTo(center.dx, center.dy);

    double Cx = center.dx + c * cos(-angle);
    double Cy = center.dy + c * sin(-angle);

    path.lineTo(Cx, Cy);

    double Bx = Cx - (a * cos(angle_b - angle));
    double By = Cy - (a * sin(angle_b - angle));

    path.lineTo(Bx, By);

    path.lineTo(center.dx, center.dy);

    path.close();
    canvas.drawPath(path, paint);
  }

  @override
  bool shouldRepaint(CustomPainter oldDelegate) {
    return true;
  }
}
```


Triangle
DEBUG
a
b
c
alpha


Triangle
DEBUG
a
b
c
alpha