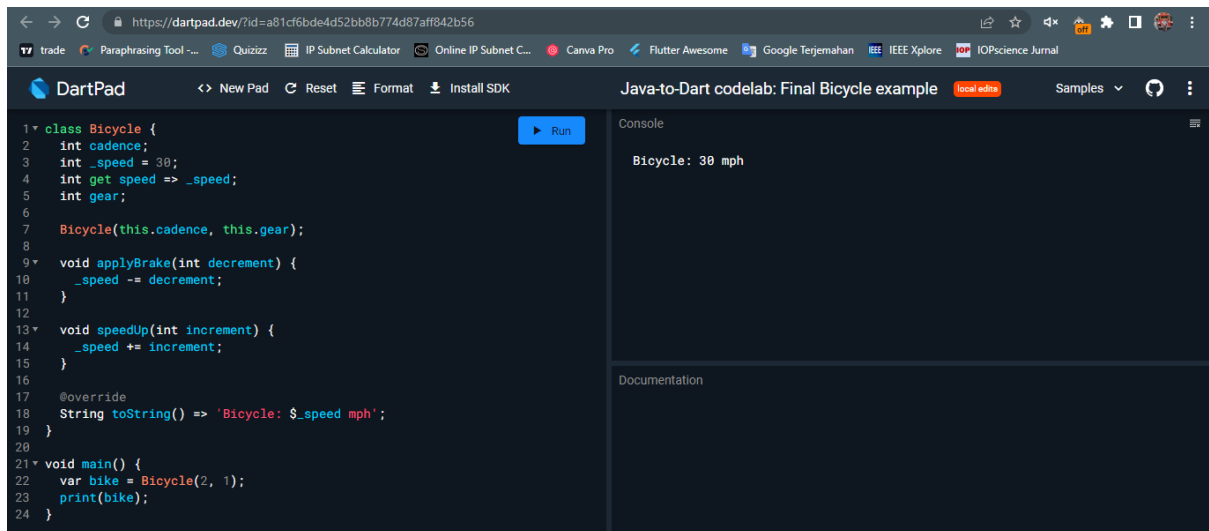


Nama : Rajendra Rakha A P  
Kelas : TSA Flutter Developer

## Membuat class Dart sederhana

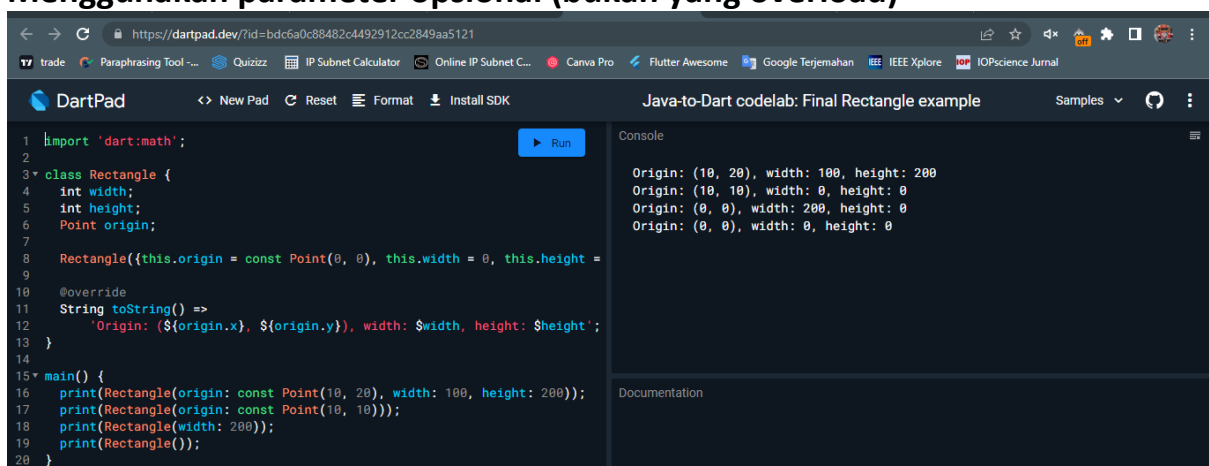


The screenshot shows the DartPad interface with a simple Dart class named `Bicycle`. The class has two private fields, `_cadence` and `_speed`, and a public field `gear`. It includes methods `applyBrake` and `speedUp` to modify the speed, and a `toString` method to format the output. The `main` function creates a `Bicycle` instance with a cadence of 2 and a gear of 1, then prints it. The console output shows "Bicycle: 30 mph".

```
1 class Bicycle {
2   int _cadence;
3   int _speed = 30;
4   int get speed => _speed;
5   int gear;
6
7   Bicycle(this.cadence, this.gear);
8
9   void applyBrake(int decrement) {
10    _speed -= decrement;
11  }
12
13  void speedUp(int increment) {
14    _speed += increment;
15  }
16
17  @override
18  String toString() => 'Bicycle: $_speed mph';
19 }
20
21 void main() {
22   var bike = Bicycle(2, 1);
23   print(bike);
24 }
```

Console output: Bicycle: 30 mph

## Menggunakan parameter opsional (bukan yang overload)

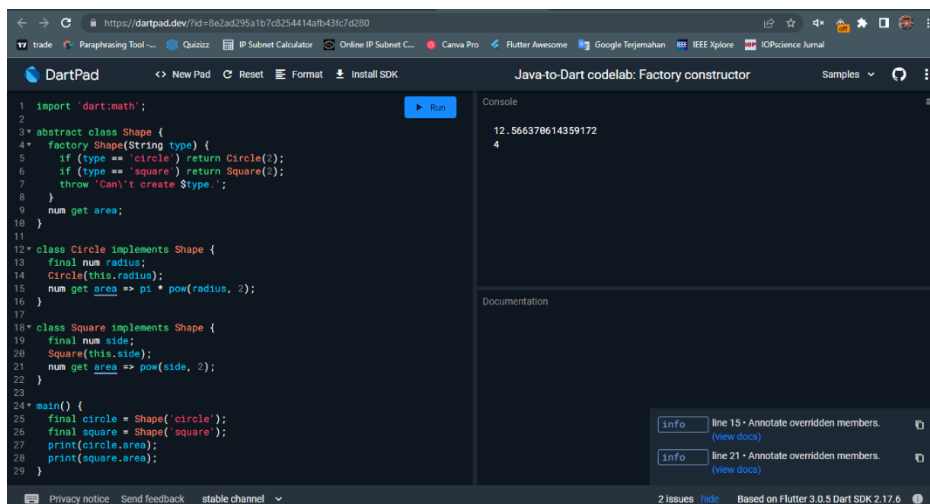


The screenshot shows the DartPad interface with a `Rectangle` class. The class has optional parameters for `width` and `height` in its constructor and `toString` method. The `main` function demonstrates various ways to instantiate the `Rectangle` class, including with and without optional parameters. The console output shows the formatted string for each instance.

```
1 import 'dart:math';
2
3 class Rectangle {
4   int width;
5   int height;
6   Point origin;
7
8   Rectangle({this.origin = const Point(0, 0), this.width = 0, this.height =
9
10  @override
11  String toString() =>
12    'Origin: (${origin.x}, ${origin.y}), width: $width, height: $height';
13 }
14
15 main() {
16   print(Rectangle(origin: const Point(10, 20), width: 100, height: 200));
17   print(Rectangle(origin: const Point(10, 10)));
18   print(Rectangle(width: 200));
19   print(Rectangle());
20 }
```

Console output: Origin: (10, 20), width: 100, height: 200  
Origin: (10, 10), width: 0, height: 0  
Origin: (0, 0), width: 200, height: 0  
Origin: (0, 0), width: 0, height: 0

## Membuat factory

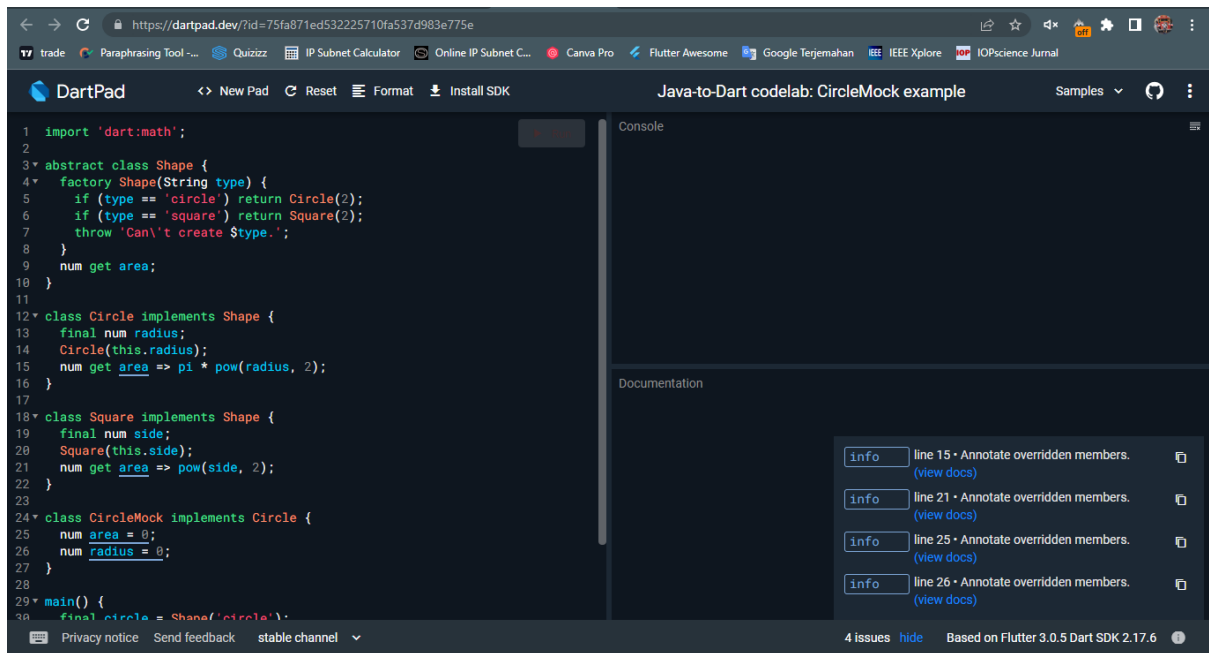


The screenshot shows the DartPad interface with an abstract class `Shape` that has a factory method `factory Shape(String type)`. This method returns a `Circle` object if the type is 'circle' and a `Square` object if the type is 'square'. The `main` function creates instances of `Shape` using the factory method and prints their areas. The console output shows the area of the circle and the square.

```
1 import 'dart:math';
2
3 abstract class Shape {
4   factory Shape(String type) {
5     if (type == 'circle') return Circle(2);
6     if (type == 'square') return Square(2);
7     throw 'Can't create $type';
8   }
9   num get area;
10 }
11
12 class Circle implements Shape {
13   final num radius;
14   Circle(this.radius);
15   num get area => pi * pow(radius, 2);
16 }
17
18 class Square implements Shape {
19   final num side;
20   Square(this.side);
21   num get area => pow(side, 2);
22 }
23
24 main() {
25   final circle = Shape('circle');
26   final square = Shape('square');
27   print(circle.area);
28   print(square.area);
29 }
```

Console output: 12.566370614359172  
4

## Mengimplementasikan antarmuka



The screenshot shows the DartPad web interface with a code editor on the left and a console/documentation panel on the right. The code defines an abstract class `Shape` with a `factory` constructor and a `get area` property. Two concrete classes, `Circle` and `Square`, implement the `Shape` interface. A `CircleMock` class is also shown, which implements `Circle` but has its own `area` and `radius` properties. The console panel shows four informational messages about overridden members.

```
1 import 'dart:math';
2
3 abstract class Shape {
4   factory Shape(String type) {
5     if (type == 'circle') return Circle(2);
6     if (type == 'square') return Square(2);
7     throw 'Can\'t create $type.';
8   }
9   num get area;
10 }
11
12 class Circle implements Shape {
13   final num radius;
14   Circle(this.radius);
15   num get area => pi * pow(radius, 2);
16 }
17
18 class Square implements Shape {
19   final num side;
20   Square(this.side);
21   num get area => pow(side, 2);
22 }
23
24 class CircleMock implements Circle {
25   num area = 0;
26   num radius = 0;
27 }
28
29 main() {
30   final circle = Shape('circle');
```

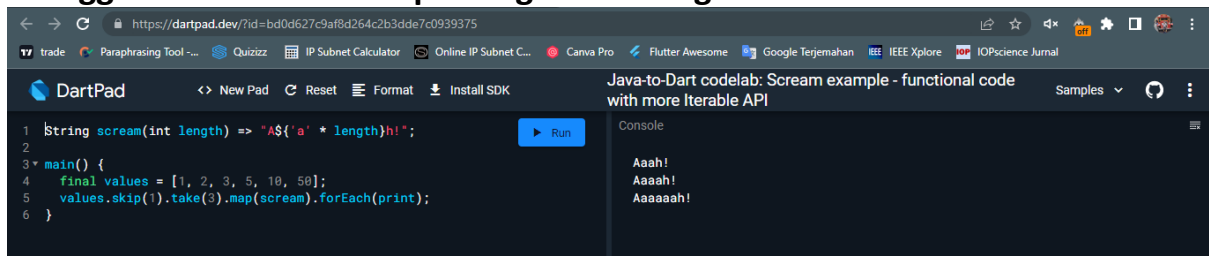
Console

Documentation

- info line 15 • Annotate overridden members. (view docs)
- info line 21 • Annotate overridden members. (view docs)
- info line 25 • Annotate overridden members. (view docs)
- info line 26 • Annotate overridden members. (view docs)

4 issues [hide](#) Based on Flutter 3.0.5 Dart SDK 2.17.6

## Menggunakan Dart untuk pemrograman fungsional



The screenshot shows the DartPad web interface with a code editor on the left and a console panel on the right. The code defines a `scream` function that takes a string and returns a new string with 'A's. The `main` function uses a list of numbers, skips the first one, takes the next three, maps them to the `scream` function, and prints the results. The console panel shows the output of the `print` statements.

```
1 String scream(int length) => "A${'a' * length}h!";
2
3 main() {
4   final values = [1, 2, 3, 5, 10, 50];
5   values.skip(1).take(3).map(scream).forEach(print);
6 }
```

Console

Aaah!  
Aaaah!  
Aaaaaah!