

Komponen dalam Mendesain Aplikasi

TI2137 - Pengembangan Aplikasi Mobil Front-End



Team Teaching

- Sunaryo Winardi, S.Kom., M.Tl.
- Sio Jurnal Pipin, S.Kom., M.Kom.
- M. Taufiq Hidayat Pohan, S.Kom.
- Richy Rotuahta Saragih, S.Kom.

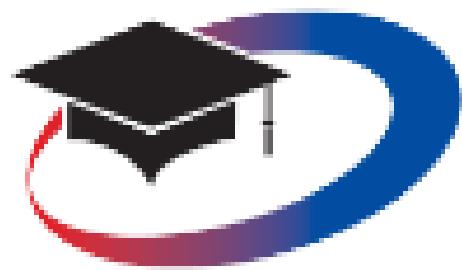


Context

- State
- Provider



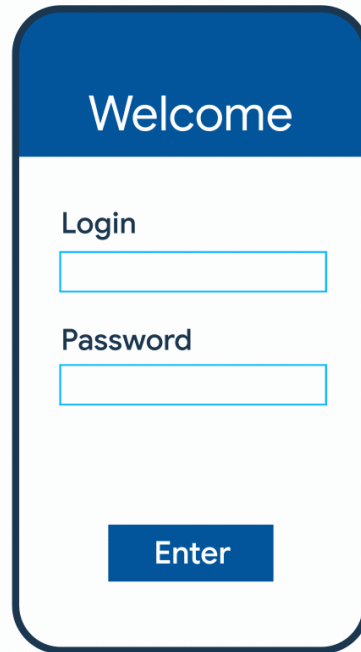
State



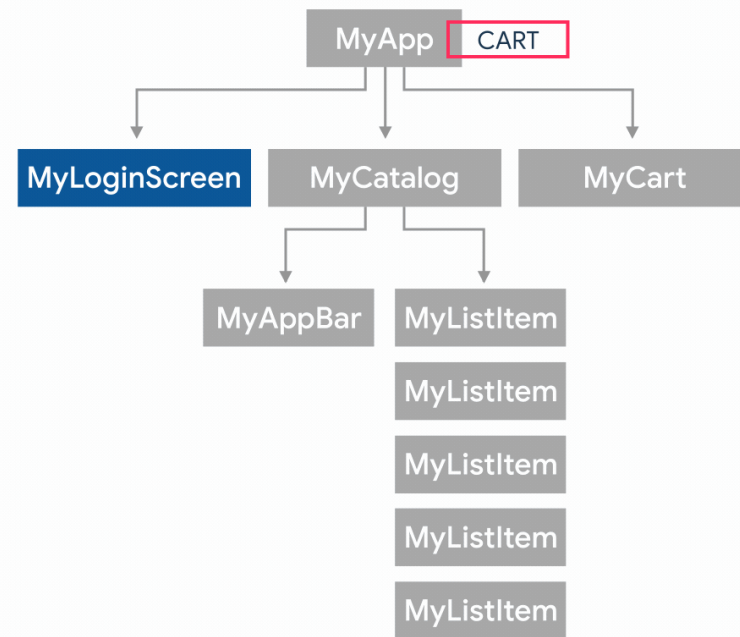
UNIVERSITAS
MIKROSKIL

PRODI. TEKNIK INFORMATIKA (S-1)

State



A mobile app login screen mockup. It features a blue header with the text "Welcome". Below the header, there is a "Login" label followed by a text input field. Underneath the input field is a "Password" label followed by another text input field. At the bottom of the screen is a blue button with the text "Enter".



Apa itu State?

- **Variable**
- **Data yang dinamis**
- Contoh : `function(state) => UI`
- Terbagi menjadi :
 - Local State
 - Global State

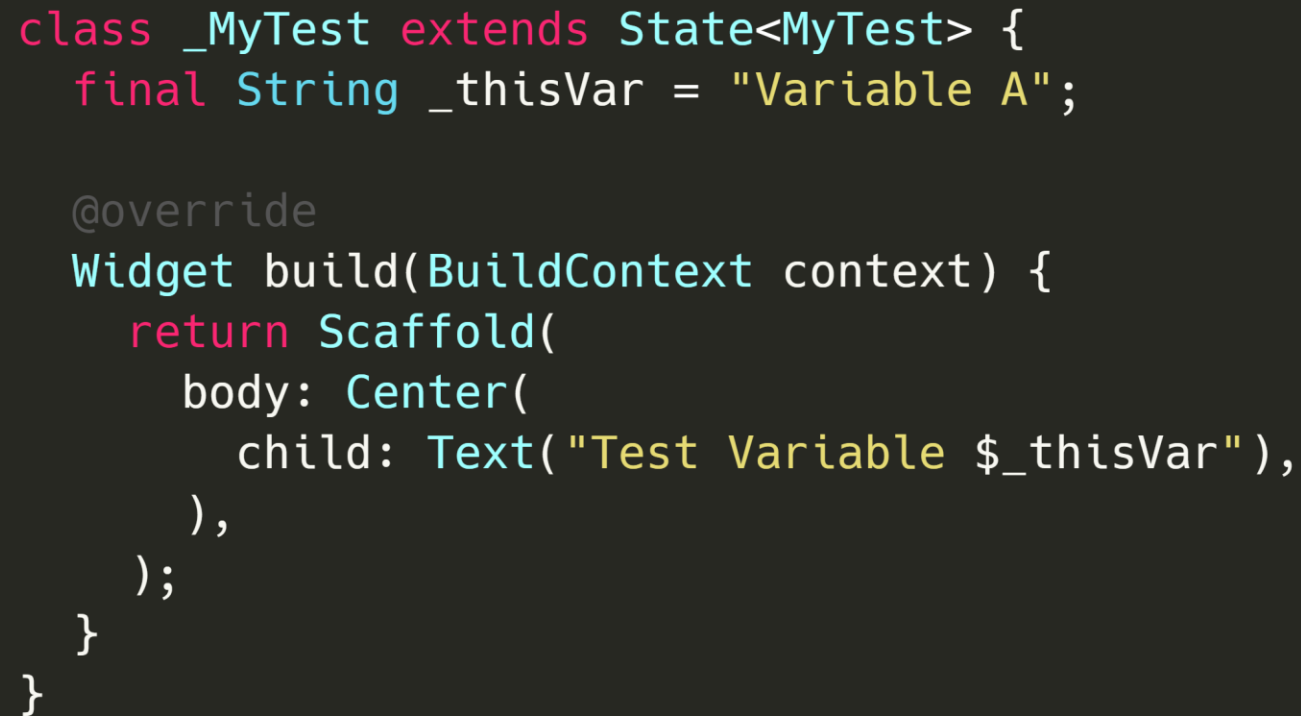


Local State

- **Ephemeral State** atau **UI State**
- **Data** yang disimpan pada **1 widget** atau **class** saja
- Bersifat Private
- Tidak dapat diakses oleh Class lain



Local State



```
class _MyTest extends State<MyTest> {  
  final String _thisVar = "Variable A";  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: Center(  
        child: Text("Test Variable $_thisVar"),  
      ),  
    );  
  }  
}
```


App State

- Terkadang disebut **Shared State**
- Dapat dibagikan ke seluruh layar aplikasi
- Dapat digunakan oleh **widget** atau **class lain**



App State

```
class MyTest extends StatefulWidget {
  @override
  _MyTest createState() => _MyTest();
}

class _MyTest extends State<MyTest> {
  int _thisNum = 0;

  void _increase() {
    setState(() {
      _thisNum++;
    });
  }

  void _decrease() {
    setState(() {
      _thisNum--;
    });
  }
}
```

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text("This number : $_thisNum"),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              OutlinedButton(
                onPressed: _decrease,
                child: Icon(Icons.text_decrease),
              ),
              ElevatedButton(
                onPressed: _increase,
                child: Icon(Icons.text_increase),
              ),
            ],
          ),
        ],
      ),
    ),
  );
}
```

State Management

- Tata cara dalam mengatur data/state bekerja
- Berfungsi untuk memisahkan **Logic** dan **View**
- Menyediakan **re-usable** logic



How State Managemen work?

- **Provide**

Menyimpan state yang akan berubah sewaktu-waktu

- **Listen**

Widget yang berhubungan dengan provider dan berubah sesuai state yang berubah



Stateless VS Stateful Widget

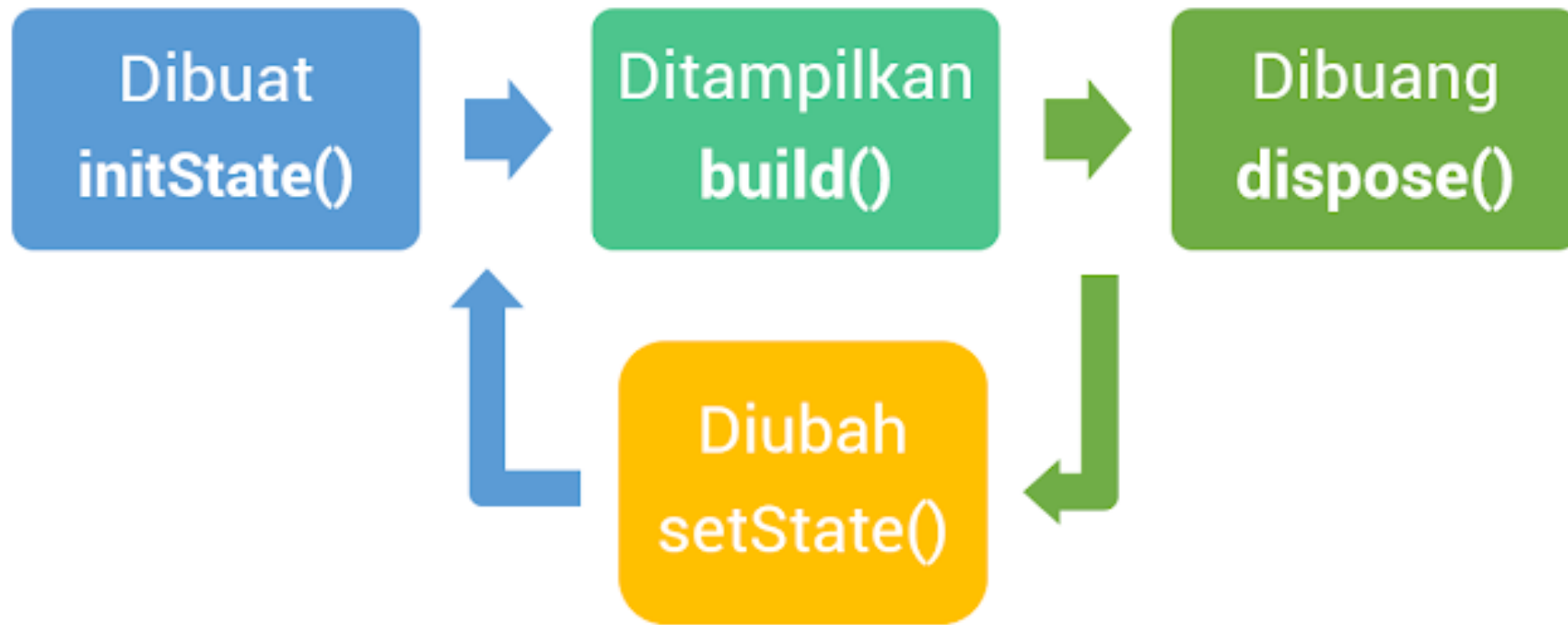
Stateless Widget

- Tidak Dapat mengubah data pada Widget
- Widget bersifat **tetap** atau **konstan**
- Tampilan dapat berubah dengan memanggil **Statefull Widget**

Stateful Widget

- Dapat mengubah data pada Widget
- Widget bersifat **dinamis**
- Tampilan dapat berubah

Stateful Widget Lifecycle



Stateful Widget Lifecycle

initState()

- Method yang dipanggil **pertama**
- Dipanggil saat widget **dibuat**
- Hanya **dipanggil sekali**
- **Tidak dapat dipanggil kembali** saat widget masih hidup
- Berfungsi untuk meng**inisialisasi**kan value



Stateful Widget Lifecycle

build()

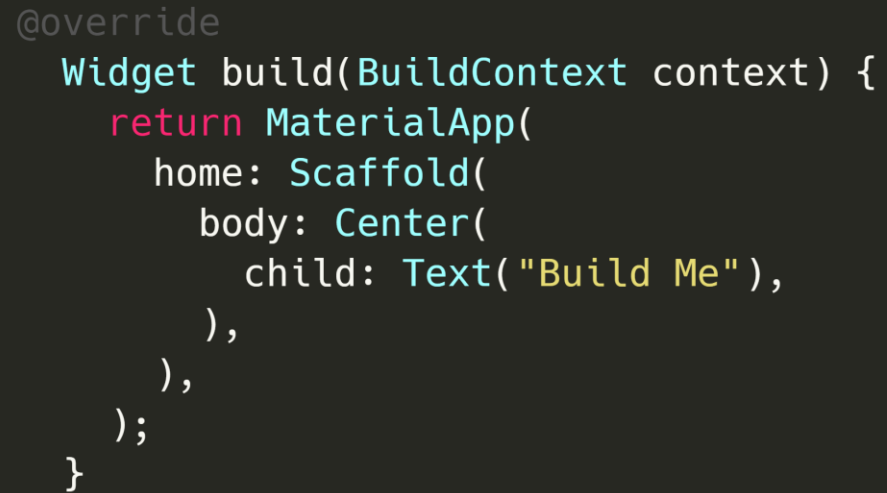
- **Membangun widget** ke tampilan layar
- Berfungsi untuk menampilkan **widget tree**
- **Dapat dipanggil kembali** ketika ada **perubahan** pada data



MIKROSKIL

Stateful Widget Lifecycle

build()



```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      body: Center(
        child: Text("Build Me"),
      ),
    ),
  );
}
```

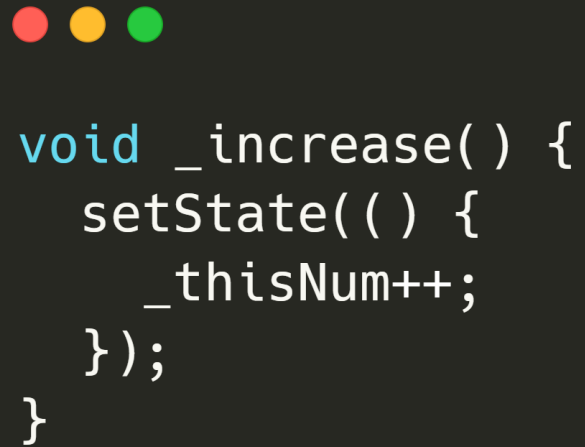
Stateful Widget Lifecycle

setState()

- Berfungsi untuk **update data** pada widget
- Ketika dijalankan, akan mentrigger method **build()**
- **Bukan** untuk **membuat ulang (re-create) widget**, hanya **update**
- Dipanggil melalui **widget input**, seperti textfield, button, inkwell, dll
- Dapat **mengubah tampilan pada widget**, seperti warna, teks, ukuran, dll

Stateful Widget Lifecycle

setState()



```
void _increase() {  
  setState(() {  
    _thisNum++;  
  });  
}
```

Stateful Widget Lifecycle

dispose()

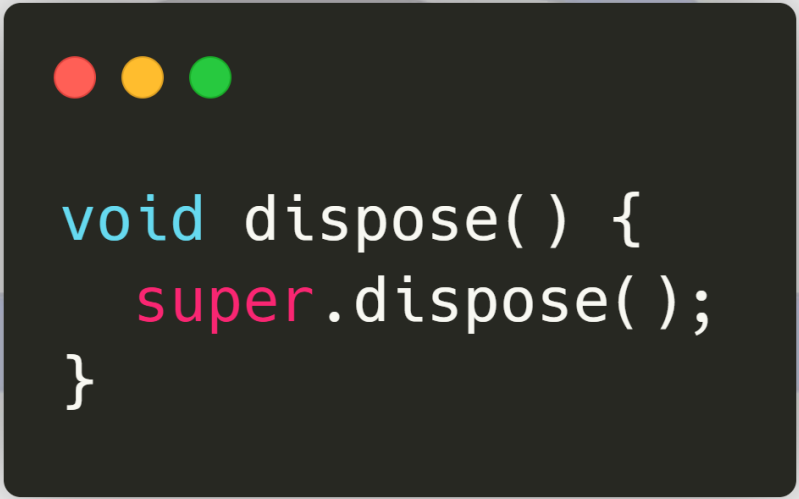
- Berfungsi untuk **meninggalkan focus** pada widget
- Dijalankan saat widget akan **ditinggalkan** atau **menutup aplikasi**
- Hanya dijalankan saat **widget ditinggalkan** atau **dibuang**



MIKROSKIL

Stateful Widget Lifecycle

dispose()



```
void dispose() {  
    super.dispose();  
}
```

Kekurangan Stateful Widget

Setiap adanya **perubahan data/state**, maka **seluruh widget** pada sebuah stateful widget akan di **build ulang**



State Management Package

- **Provider**
- Redux
- GetX
- BLoC
- RiverPod
- Binder
- Dan sebagainya

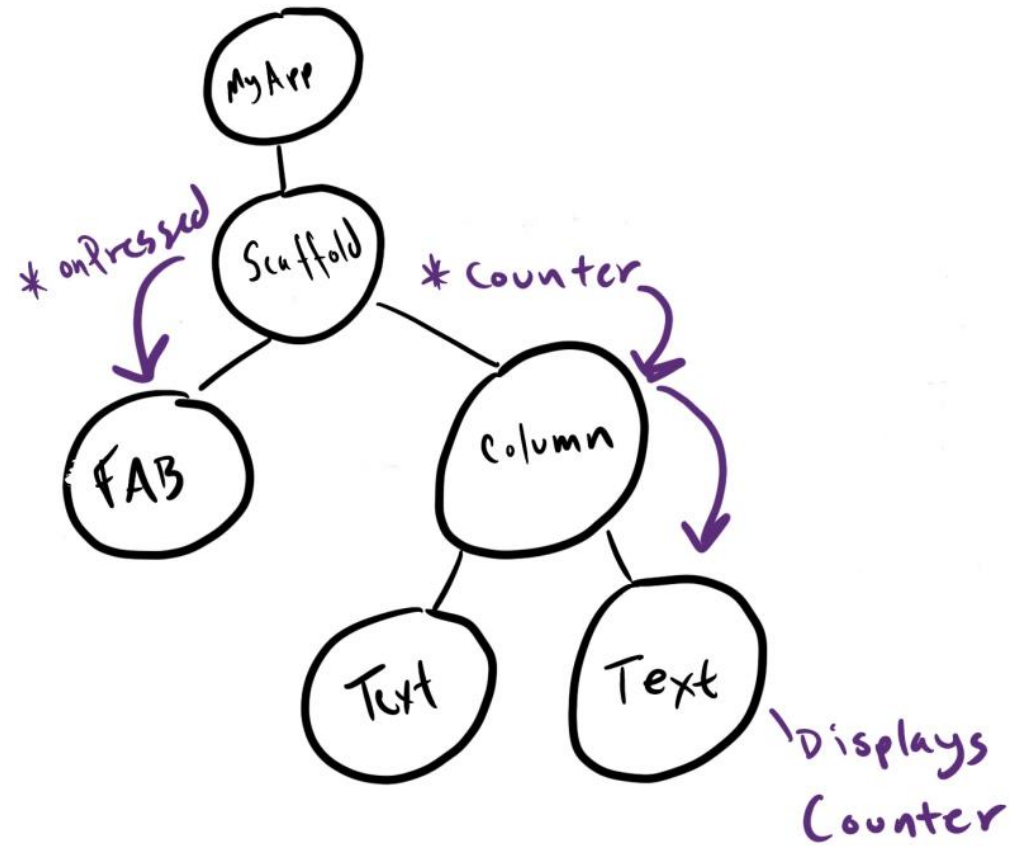


Provider

- Sebuah State Management Package untuk Flutter
- Berfungsi untuk mengubah state atau tampilan tanpa build ulang seluruh widget



Provider



Konsep Provider Package

- Memiliki satu pusat state yang diatur dalam state management
- Widget yang membutuhkan state/data dapat melakukan :
 - **Listen**
Mengambil variable dari provider dan merubah UI/Widget jika terdapat perubahan pada variable
 - **Once time access**
Mendapatkan hasil perubahan state/data dari provider pada saat tertentu

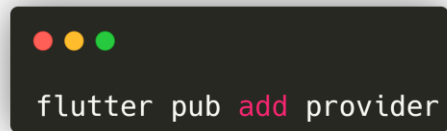
Fitur Provider

- Memudahkan dalam **mengalokasikan resource**
- Memiliki **Lazy-loading**
- Mengurangi **boilerplate** karena selalu membuat kelas baru
- State dapat terlihat dalam **Flutter Devtool**
- Mudah untuk menggunakan **InheritedWidgets**
- **Scalability** dapat ditingkatkan
- **Cepat** dan **kompleksitas** dapat mencapai **$O(N)$**

Instalasi Provider Package

#1. Melalui Pub Add

- Buka Terminal pada VSCode atau folder project
- Ketik command berikut :



```
flutter pub add provider
```

- Tekan Enter



Instalasi Provider Package

#2. Melalui file pubspec.yaml

- Buka file pubspec.yaml
- Tambahkan syntax berikut pada bagian **dependencies** :

```
provider: ^6.0.2
```

- Save file tersebut



Instalasi Provider Package

Note untuk #2 :

- Versi package dapat dilihat pada website
- Berbeda versi dapat menimbulkan perbedaan syntax
- Website package [Klik Disini](#)



MIKROSKIL

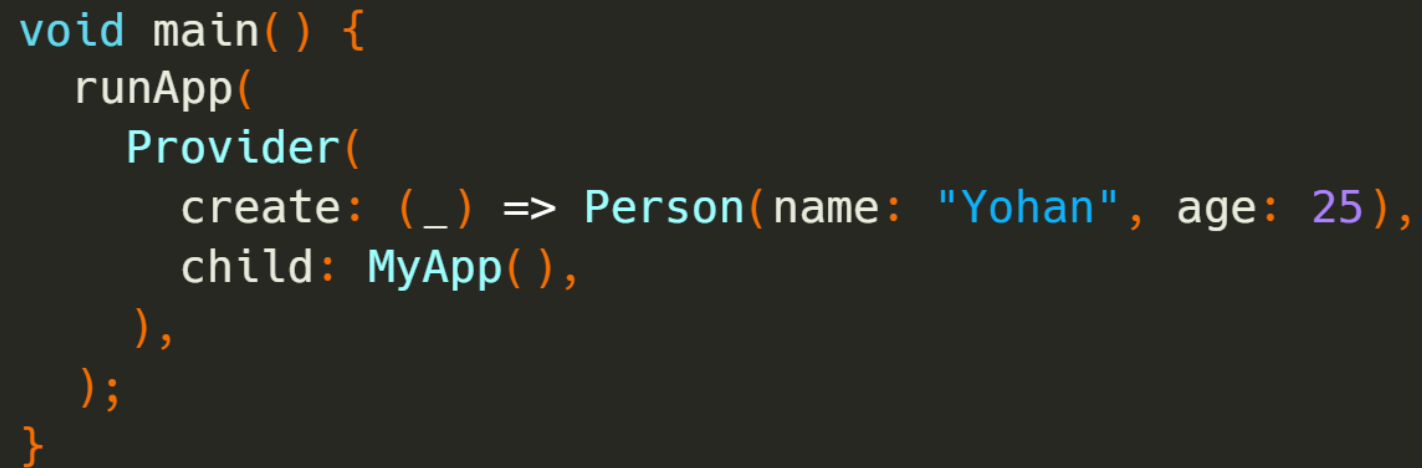
Provider Method

Provider()

- Merupakan InheritedWidget yang menyediakan class untuk Provider
- Setiap fungsi baru membutuhkan addListener() sebagai ChangeNotifier
- Digunakan hanya jika memiliki 1 Provider
- Digunakan pada main.dart
- Tidak cocok untuk multiple widget yang bercabang

Provider Method

Provider()

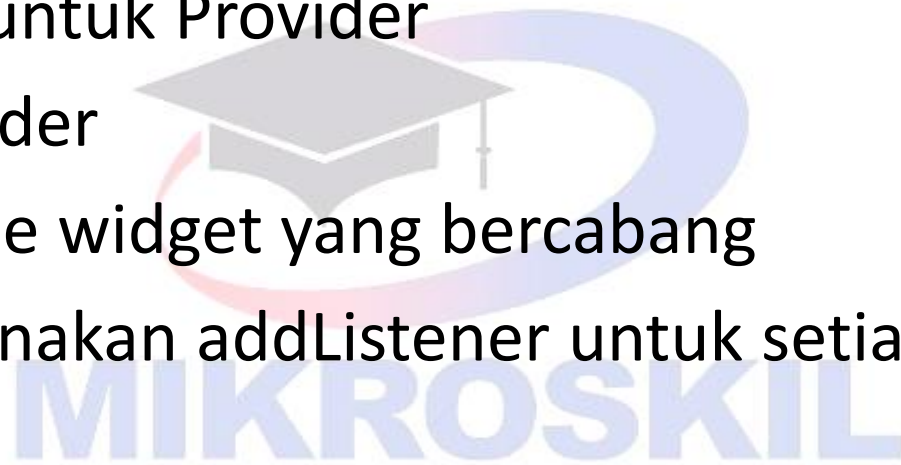


```
void main() {  
  runApp(  
    Provider(  
      create: (_) => Person(name: "Yohan", age: 25),  
      child: MyApp(),  
    ),  
  );  
}
```


Provider Method

ChangeNotifierProvider()

- Menyediakan class untuk Provider
- Cocok untuk 1 Provider
- Cocok untuk multiple widget yang bercabang
- Tidak perlu menggunakan addListener untuk setiap fungsi baru



Provider Method

ChangeNotifierProvider()



```
void main() {  
  runApp(  
    ChangeNotifierProvider(  
      create: (_) => Person(name: "Yohan", age: 25),  
      child: MyApp(),  
    ),  
  );  
}
```

Provider Method

MultiProvider()

- Menyediakan class untuk Provider untuk
- Cocok untuk layer provider yang bercabang
- Cocok untuk layer provider yang lebih dari 1



MIKROSKIL

Provider Method

MultiProvider()

```
void main() {  
  runApp(  
    MultiProvider(  
      providers: [  
        Provider<Person>(create: (_) => Person(name: 'Yohan', age: 25)),  
        FutureProvider<String>(create: (context) => Home().fetchAddress),  
      ],  
      child: MyApp(),  
    ),  
  );  
}
```

Provider Method

ChangeNotifier()

- Merupakan class yang dapat di **extended** atau **mixed** untuk menyediakan sebuah notifikasi perubahan API menggunakan VoidCallback.
- Sebagai listen pada ChangeNotifier object
- Memanggil notifyListener() saat ada perubahan nilai

Provider Method

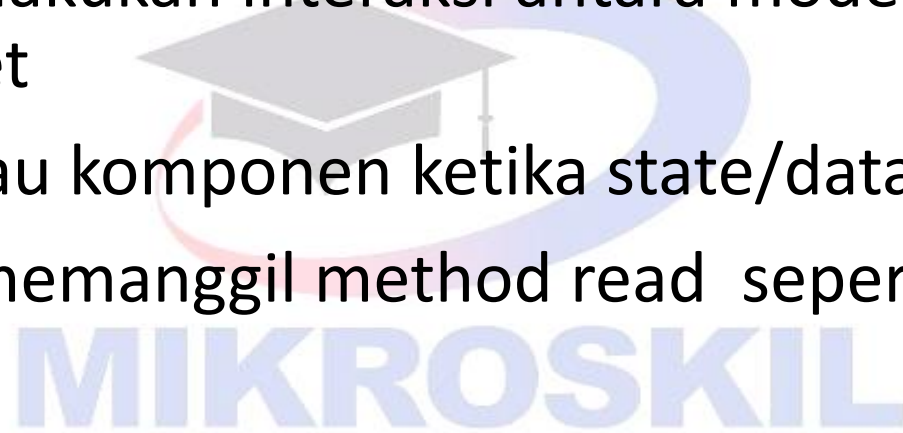
ChangeNotifier()

```
class Person with ChangeNotifier {  
  Person({this.name, this.age});  
  
  final String name;  
  int age;  
  
  void increaseAge() {  
    this.age++;  
    notifyListeners();  
  }  
}
```

Provider Method

Consumer()

- Berfungsi untuk melakukan interaksi antara model (nilai yang akan berubah) dan widget
- Merubah widget atau komponen ketika state/data berubah
- Dibutuhkan untuk memanggil method read seperti :
 - Context.watch
 - Context.read
 - Context.select
 - Provider.of()



Provider Method

Consumer()

```
@override
Widget build(BuildContext context) {
  return Consumer<Person>(
    builder: (context, person, _) {
      return Scaffold(
        appBar: AppBar(
          title: const Text('Provider Class'),
        ),
        body: Center(
          child: Text(
            '
            Hi ${person.name}!
            You are ${person.age} years old',
          ),
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: () => person.increaseAge(),
        ),
      );
    },
  );
}
```


Membaca value pada Provider

context.watch<T>()

Melakukan perubahan pada widget sesuai dengan T (tipe data)

Contoh :

context.watch<String> ()



Membaca value pada Provider

context.read<T>()

Melakukan perubahan pada widget sesuai dengan T (tipe data). Widget yang berubah tidak build ulang, hanya dilakukan update.

Contoh :

```
context.read<String> ()
```

MIKROSKIL

Membaca value pada Provider

context.select<T, R>(R selector (T value))

Membaca sebagian isi dari T (tipe data) untuk diubah pada Widget Consumer.

contoh

context.select((Person p) => p.name)



Membaca value pada Provider

Provider.of<T>(context, listen: false)

- Membaca dengan **metode statis**
- Jika listen **false**, akan sama seperti **context.watch**
- Jika listen **true**, akan sama seperti **context.read**

contoh

```
context.select((Person p) => p.name)
```

MIKROSKIL

Mulai menggunakan Provider

Membuat objek baru pada property **create**

```
Provider(  
  create: (_) => MyModel(),  
  child: ...  
)
```

Mulai menggunakan Provider

Jangan menggunakan **Provider.value** untuk membuat objek

```
ChangeNotifierProvider.value(  
  value: MyModel(),  
  child: ...  
)
```

Mulai menggunakan Provider

Gunakan **ProxyProvider** jika ingin membuat objek melalui **variable**

```
int count;  
  
ProxyProvider0(  
  update: (_, __) => MyModel(count),  
  child: ...  
)
```

Hindari ini pada Provider

Jangan membuat objek melalui **variable**

```
int count;  
  
Provider(  
  create: (_) => MyModel(count),  
  child: ...  
)
```


Hindari ini pada Provider

Jangan gunakan **ChangeNotifierProvider.value** untuk **ChangeNotifier** yang sudah ada

```
MyChangeNotifier variable;  
  
ChangeNotifierProvider.value(  
  value: variable,  
  child: ...  
)
```

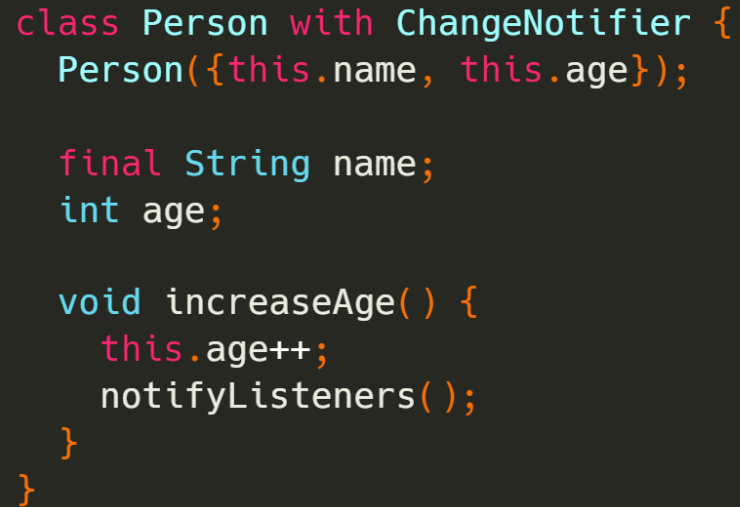
Hindari ini pada Provider

Jangan gunakan ulang **ChangeNotifier** yang ada menggunakan **default constructor**

```
MyChangeNotifier variable;  
  
ChangeNotifierProvider(  
  create: (_) => variable,  
  child: ...  
)
```

Contoh penggunaan Provider

1. Memiliki sebuah class Data, sebagai contoh ambil data Person dengan atribut nama dan umur

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Kotlin and defines a class 'Person' that implements 'ChangeNotifier'. The class has two attributes: 'name' of type 'String' and 'age' of type 'int'. It includes a constructor 'Person({this.name, this.age});', a 'final' property 'name', and a method 'increaseAge()' that increments 'age' and calls 'notifyListeners()'.

```
class Person with ChangeNotifier {  
    Person({this.name, this.age});  
  
    final String name;  
    int age;  
  
    void increaseAge() {  
        this.age++;  
        notifyListeners();  
    }  
}
```

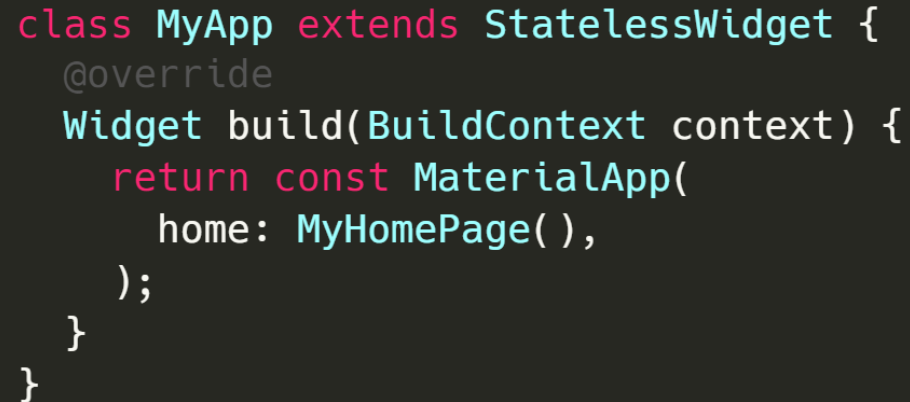
Contoh penggunaan Provider

2. Inisialisasi fungsi main() menggunakan provider

```
void main() {  
  runApp(  
    ChangeNotifierProvider(  
      create: (_) => Person(name: "Yohan", age: 25),  
      child: MyApp(),  
    ),  
  );  
}
```

Contoh penggunaan Provider

3. Membuat class utama MyApp sesuai yang dipanggil pada fungsi main()



```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return const MaterialApp(  
      home: MyHomePage(),  
    );  
  }  
}
```

Contoh penggunaan Provider

4. Membuat class MyHomePage untuk tampilan proses

```
class MyHomePage extends StatelessWidget {  
  const MyHomePage({Key key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('Provider Class'),  
      ),  
      body: Center(  
        child: Text(  
          ...  
          Hi ${Provider.of<Person>(context).name}!  
          You are ${Provider.of<Person>(context).age} years old'',  
        ),  
      ),  
    );  
  }  
}
```

Terima Kasih

