

Berinteraksi dengan UI dan status

Pada intinya, status di aplikasi adalah nilai yang dapat berubah dari waktu ke waktu.

Semua aplikasi Android menampilkan status kepada pengguna. Beberapa contoh status di aplikasi Android termasuk:

- Pesan yang muncul saat koneksi jaringan tidak dapat dibuat.
- Formulir, seperti formulir pendaftaran. Status dapat diisi dan dikirim.
- Kontrol yang dapat diketuk, seperti tombol. Status dapat *tidak diketuk*, *diketuk* (animasi tampilan), atau *diketuk* (tindakan `onClick`).

Mendapatkan Kode Awal

Kode awal adalah kode **Tip Time** yang telah ditulis sebelumnya yang dapat digunakan sebagai titik awal untuk project baru.

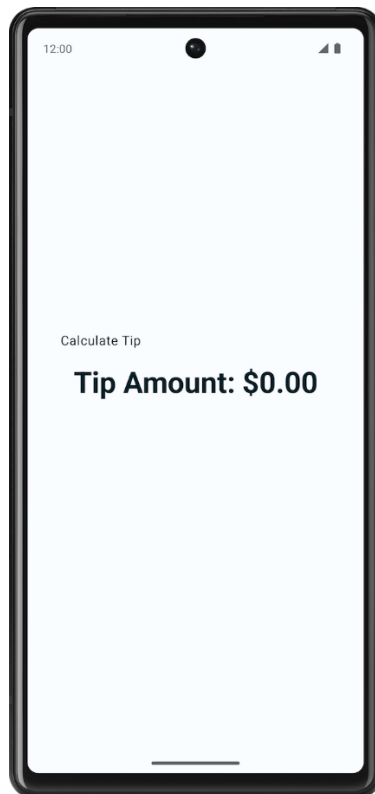
Mulai menggunakan kode awal dengan mendownloadnya di sini:

<https://github.com/google-developer-training/basic-android-kotlin-compose-training-tip-calculator/archive/refs/heads/starter.zip>

Ringkasan aplikasi awal

Untuk memahami kode awal, selesaikan langkah-langkah berikut:

1. Buka project dengan kode awal di Android Studio.
2. Jalankan aplikasi di perangkat Android atau emulator.
3. Anda akan melihat dua komponen teks; satu untuk label dan satunya lagi untuk menampilkan jumlah tip.



Beberapa file untuk membantu Anda memulai.

res > values > strings.xml

```
<resources>
  <string name="app_name">Tip Time</string>
  <string name="calculate_tip">Calculate Tip</string>
  <string name="bill_amount">Bill Amount</string>
  <string name="tip_amount">Tip Amount: %s</string>
</resources>
```

File ini adalah file `string.xml` dalam resource dengan semua string yang akan Anda gunakan dalam aplikasi ini.

MainActivity

File ini sebagian besar berisi kode yang dihasilkan oleh template dan fungsi berikut.

- Fungsi `TipTimeLayout()` berisi elemen `Column` dengan dua composable teks yang Anda lihat di screenshot. Fungsi ini juga memiliki composable `spacer` untuk menambahkan ruang untuk alasan estetika.
- Fungsi `calculateTip()` yang menerima jumlah tagihan dan menghitung jumlah tip 15%. Parameter `tipPercent` disetel ke nilai argumen default `15.0`. Untuk saat ini, nilai tip default ditetapkan ke 15%. Di codelab berikutnya, Anda akan mendapatkan jumlah tip dari pengguna.

```

@Composable
fun TipTimeLayout() {
    Column(
        modifier = Modifier
            .statusBarsPadding()
            .padding(horizontal = 40.dp)
            .safeDrawingPadding(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        Text(
            text = stringResource(R.string.calculate_tip),
            modifier = Modifier
                .padding(bottom = 16.dp, top = 40.dp)
                .align(alignment = Alignment.Start)
        )
        Text(
            text = stringResource(R.string.tip_amount, "$0.00"),
            style = MaterialTheme.typography.displaySmall
        )
        Spacer(modifier = Modifier.height(150.dp))
    }
}

```

```

/**
 * Calculates the tip based on the user input and format the tip amount
 * according to the local currency.
 * Example would be "$10.00".
 */
private fun calculateTip(amount: Double, tipPercent: Double = 15.0):
String {
    val tip = tipPercent / 100 * amount
    return NumberFormat.getCurrencyInstance().format(tip)
}

```

Mengambil input dari pengguna

Label
Hello

Fungsi composable `TextField` memungkinkan pengguna memasukkan teks dalam aplikasi.

1. Dalam file `MainActivity.kt`, tambahkan fungsi composable `EditNumberField()`, yang menggunakan parameter `Modifier`.

2. Dalam isi fungsi `EditNumberField()` di bawah `TipTimeLayout()`, tambahkan `TextField` yang menerima parameter bernama `value` yang disetel ke string kosong dan parameter bernama `onValueChange` yang disetel ke ekspresi lambda kosong:

```
@Composable
fun EditNumberField(modifier: Modifier = Modifier) {
    TextField(
        value = "",
        onValueChange = {},
        modifier = modifier
    )
}
```

- Parameter `value` adalah kotak teks yang menampilkan nilai string yang Anda teruskan di sini.
- Parameter `onValueChange` adalah callback lambda yang dipicu saat pengguna memasukkan teks di kotak teks.

3. Jangan lupa import:

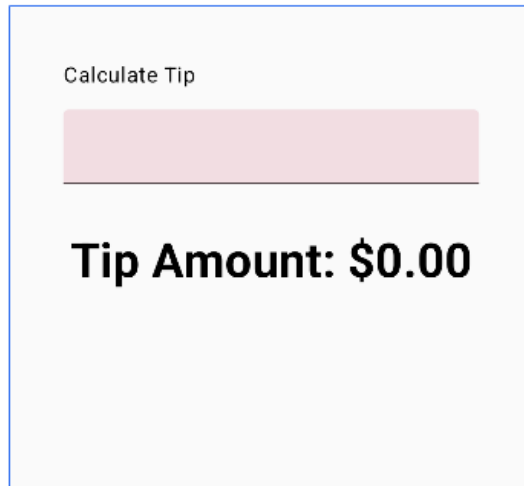
```
import androidx.compose.material3.TextField
```

4. Dalam composable `TipTimeLayout()`, di baris setelah fungsi composable teks pertama, panggil fungsi `EditNumberField()`

```
Text(
    . . .
)
EditNumberField(
    modifier = Modifier
        .padding(bottom = 32.dp)
        .fillMaxWidth()
)
Text(
    . . .
)
```

5. Di panel **Design**, Anda akan melihat teks `Calculate Tip`, kotak teks kosong, dan composable teks `Tip Amount`.

TipTimeLayoutPreview



Menggunakan status dalam Compose

Status di aplikasi adalah nilai yang dapat berubah dari waktu ke waktu. Di aplikasi ini, status adalah jumlah tagihan.

Tambahkan variabel ke status:

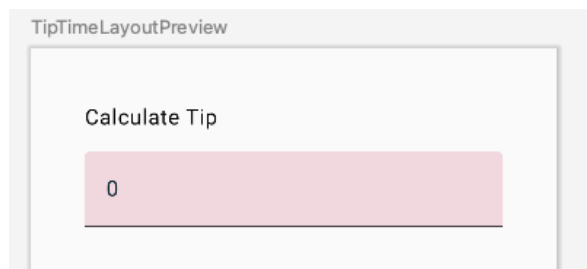
1. Di awal fungsi `EditNumberField()`, gunakan kata kunci `val` untuk menambahkan variabel `amountInput`, tetapkan ke nilai `"0"`:

```
val amountInput = "0"
```

2. Setel parameter bernama `value` ke nilai `amountInput`:

```
TextField(  
    value = amountInput,  
    onValueChange = {},  
)
```

3. Periksa pratinjau. Kotak teks menampilkan nilai yang disetel ke variabel status seperti yang dapat Anda lihat di gambar ini:



4. Jalankan aplikasi di emulator, coba masukkan nilai lain. Status hardcoded tetap tidak berubah karena composable `TextField` tidak memperbarui sendiri. Composable ini diperbarui saat parameter `value` berubah, yang disetel ke properti `amountInput`.

Variabel `amountInput` mewakili status kotak teks. Memiliki status `hardcode` tidak berguna karena tidak dapat dimodifikasi dan tidak menampilkan input pengguna. Anda harus mengubah status aplikasi saat pengguna memperbarui jumlah tagihan.

Komposisi

Apa yang terjadi jika Anda ingin UI berubah saat aplikasi sedang berjalan atau saat pengguna berinteraksi dengan aplikasi? Misalnya, bagaimana jika Anda ingin memperbarui variabel `amountInput` dengan nilai yang dimasukkan oleh pengguna dan menampilkannya di kotak teks? Saat itulah Anda mengandalkan proses yang disebut *rekomposisi* untuk memperbarui Komposisi aplikasi.

Komposisi adalah deskripsi UI yang di-build oleh Compose saat mengeksekusi *composable*. Aplikasi Compose memanggil fungsi *composable* untuk mengubah data menjadi UI. Jika terjadi perubahan status, Compose akan mengeksekusi kembali fungsi *composable* yang terpengaruh dengan status baru yang membuat UI yang diupdate—hal ini disebut *rekomposisi*. Compose menjadwalkan **rekomposisi** untuk Anda.

Rekomposisi adalah saat Compose mengeksekusi ulang *composable* yang mungkin telah berubah sebagai respons terhadap perubahan data, lalu memperbarui Komposisi untuk menampilkan setiap perubahan. Untuk melakukannya, *Compose perlu mengetahui status yang harus dilacak* sehingga dapat menjadwalkan rekomposisi saat menerima update. Dalam kasus Anda, status ini adalah variabel `amountInput`, sehingga setiap kali nilainya berubah, Compose menjadwalkan rekomposisi.

Anda menggunakan jenis `State` dan `MutableState` di Compose agar status di aplikasi Anda dapat diamati, atau dilacak, oleh Compose. Jenis `State` tidak dapat diubah, sehingga Anda hanya dapat membaca nilai di dalamnya, sedangkan jenis `MutableState` dapat berubah. Anda dapat menggunakan fungsi `mutableStateOf()` untuk membuat `MutableState` yang dapat diamati. Fungsi ini menerima nilai awal sebagai parameter yang digabungkan dalam objek `State`, yang kemudian membuat `value`-nya dapat diamati.

Nilai yang ditampilkan oleh fungsi `mutableStateOf()`:

- Mempertahankan status, yang merupakan jumlah tagihan.
- Dapat diubah, sehingga nilai dapat diubah.
- Dapat diamati, jadi Compose mengamati setiap perubahan pada nilai dan memicu rekomposisi untuk mengupdate UI.

Tambahkan status biaya layanan:

1. Dalam fungsi `EditNumberField()`, ubah kata kunci `val` sebelum variabel status `amountInput` menjadi kata kunci `var`:

```
var amountInput = "0"
```

- Gunakan tipe `MutableState<String>`, bukan variabel `String` hardcoded, sehingga Compose tahu cara melacak status `amountInput`, lalu meneruskan string `"0"` yang merupakan nilai default awal untuk variabel status `amountInput`:

```
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.mutableStateOf

var amountInput: MutableState<String> = mutableStateOf("0")
```

Inisialisasi `amountInput` juga dapat ditulis seperti ini dengan inferensi tipe:

```
var amountInput = mutableStateOf("0")
```

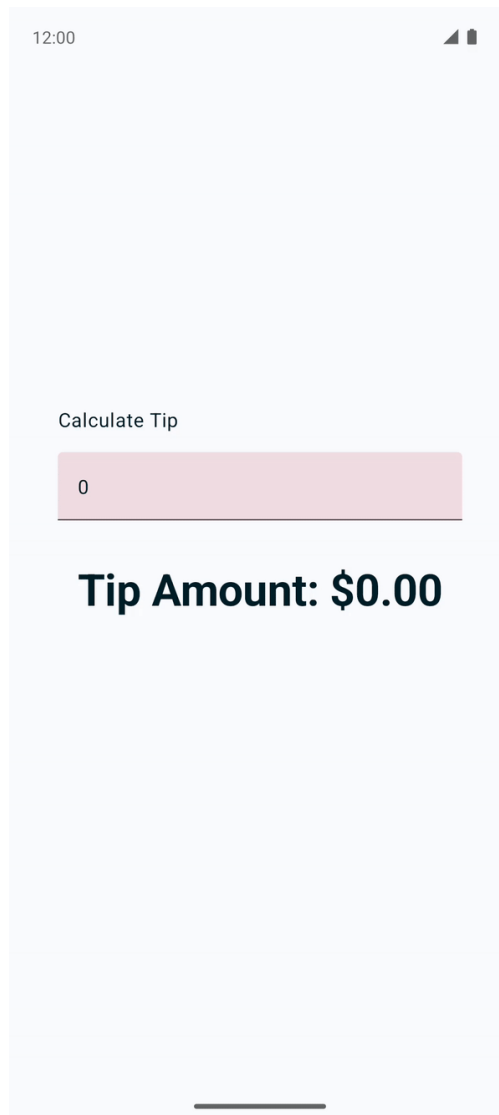
- Dalam fungsi composable `TextField`, gunakan properti `amountInput.value`:

```
TextField(
    value = amountInput.value,
    onChange = {},
    modifier = modifier
)
```

Compose melacak setiap composable yang membaca properti `value` status dan memicu rekomposisi saat `value` berubah.

Callback `onValueChange` dipicu saat input kotak teks berubah. Dalam ekspresi lambda, variabel `it` berisi nilai baru.

- Dalam ekspresi lambda parameter bernama `onValueChange`, setel properti `amountInput.value` ke variabel `it`:
- Jalankan aplikasi dan masukkan teks di kotak teks. Kotak teks masih menampilkan nilai `0` seperti yang dapat Anda lihat pada gambar ini:



Saat pengguna memasukkan teks di kotak teks, callback `onValueChanged` akan dipanggil dan variabel `amountInput` diperbarui dengan nilai baru. Status `amountInput` dilacak oleh Compose, sehingga saat nilainya berubah, rekomposisi dijadwalkan dan fungsi composable `EditNumberField()` dieksekusi lagi. Dalam fungsi composable tersebut, variabel `amountInput` direset ke nilai `0` awal. Dengan demikian, kotak teks menampilkan nilai `0`.

Menggunakan fungsi `remember` untuk menyimpan status

Fungsi composable dapat menyimpan objek di seluruh rekomposisi dengan `remember`. Nilai yang dihitung oleh fungsi `remember` disimpan dalam Komposisi selama komposisi awal dan nilai yang disimpan ditampilkan selama rekomposisi. Biasanya fungsi `remember` dan `mutableStateOf` digunakan bersama dalam fungsi composable agar status dan pembaruannya ditampilkan dengan benar di UI.

1. Dalam fungsi `EditNumberField()`, lakukan inisialisasi variabel `amountInput` dengan delegasi properti Kotlin `by remember`, dengan mengait panggilan ke fungsi `mutableStateOf()` dengan `remember`.
2. Dalam fungsi `mutableStateOf()`, teruskan string kosong, bukan string `"0"` statis:

```
var amountInput by remember { mutableStateOf("") }
```

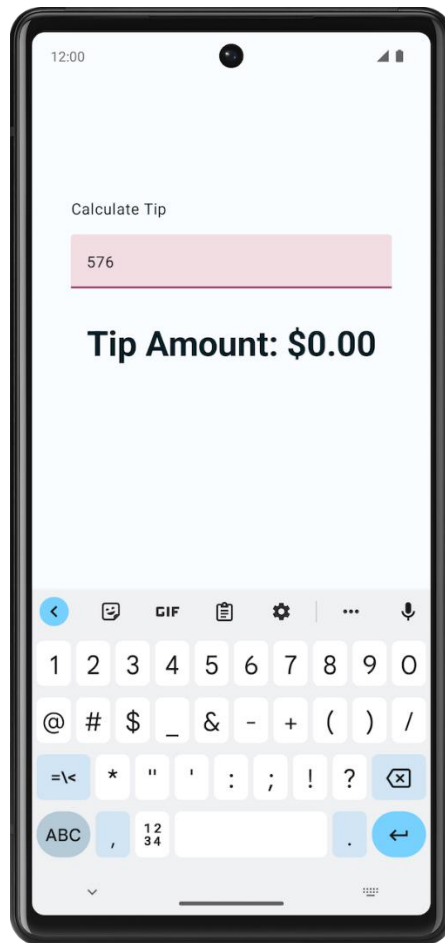
3. Impor fungsi ini:

```
import androidx.compose.runtime.remember
import androidx.compose.runtime.getValue
import androidx.compose.runtime.setValue
```

Fungsi `EditNumberField()` yang diubah akan terlihat seperti ini:

```
@Composable
fun EditNumberField(modifier: Modifier = Modifier) {
    var amountInput by remember { mutableStateOf("") }
    TextField(
        value = amountInput,
        onValueChange = { amountInput = it },
        modifier = modifier
    )
}
```

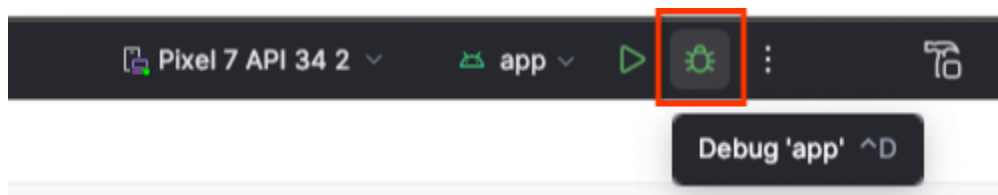
4. Jalankan aplikasi dan masukkan teks di kotak teks. Anda akan melihat teks yang Anda ketik sekarang.





Status dan cara kerja rekompresi

Di bagian ini, Anda menetapkan titik henti sementara dan men-debug fungsi composable `EditNumberField()` untuk melihat cara kerja komposisi awal dan rekompresi.

1. Pada fungsi `EditNumberField()` di samping parameter bernama `onValueChange`, tetapkan titik henti sementara baris.
2. Di menu navigasi, klik **Debug 'app'**. Aplikasi diluncurkan di emulator atau perangkat. Eksekusi aplikasi Anda dijeda untuk pertama kalinya saat elemen `TextField` dibuat.




3. Di panel Debug, klik  Resume Program. Kotak teks telah dibuat.
4. Di emulator atau perangkat, masukkan huruf di kotak teks. Eksekusi aplikasi Anda akan dijeda lagi saat mencapai titik henti sementara yang Anda tetapkan.

5. Di panel Debug, klik  **Resume Program**. Teks yang dimasukkan di emulator atau perangkat ditampilkan di samping baris dengan titik henti sementara seperti yang terlihat dalam gambar ini:

```
@Composable
fun EditNumberField() {
    var amountInput by remember { mutableStateOf( value: "" ) } amountInput$delegate: MutableState(value=2)
    TextField(
        value = amountInput,
        onChange = { amountInput = it }, amountInput$delegate: MutableState(value=2) 21493359
    )
}
```

Ini adalah status kolom teks.

6. Klik  **Resume Program**. Nilai yang dimasukkan akan ditampilkan di emulator atau perangkat.

Menambahkan label ke kotak teks

Setiap kotak teks harus memiliki label yang memungkinkan pengguna mengetahui informasi yang dapat dimasukkan.

1. Dalam fungsi composable `TextField()` dari fungsi `EditNumberField()`, tambahkan parameter bernama `label`. Dalam ekspresi lambda, panggil fungsi `Text()` yang menerima `stringResource(R.string.bill_amount)`:

```
label = { Text(stringResource(R.string.bill_amount)) },
```

2. Dalam fungsi composable `TextField()`, tambahkan parameter bernama `singleLine` yang disetel ke nilai `true`:

```
TextField(
    // ...
    singleLine = true,
)
```

3. Tambahkan parameter `keyboardOptions` yang disetel ke `KeyboardOptions()`. Setel tipe keyboard ke keyboard angka untuk memasukkan angka. Teruskan fungsi `KeyboardOptions` dengan parameter bernama `keyboardType` yang disetel ke `KeyboardType.Number`

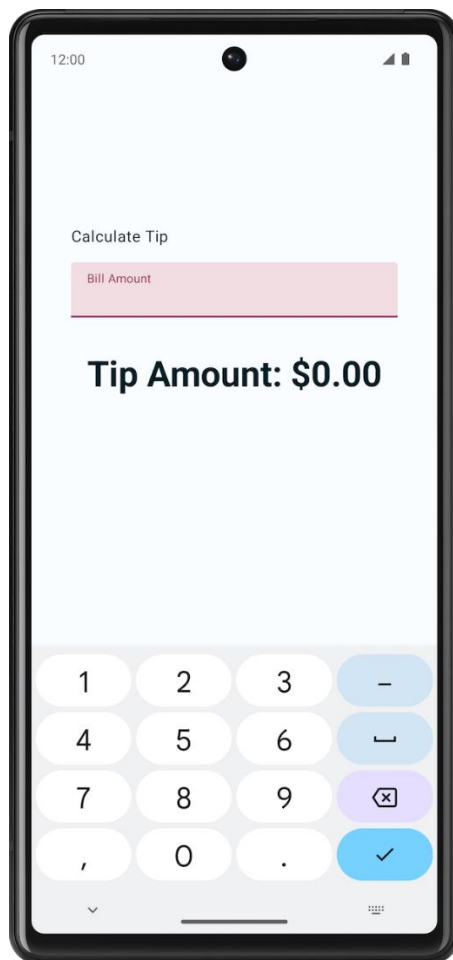
```
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.ui.text.input.KeyboardType

TextField(
    // ...
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
)
```

Fungsi `EditNumberField()` yang telah selesai akan terlihat seperti cuplikan kode ini:

```
@Composable
fun EditNumberField(modifier: Modifier = Modifier) {
    var amountInput by remember { mutableStateOf("") }
    TextField(
        value = amountInput,
        onValueChange = { amountInput = it },
        label = { Text(stringResource(R.string.bill_amount)) },
        singleLine = true,
        keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
        modifier = modifier
    )
}
```

4. Jalankan aplikasi. Anda dapat melihat perubahan pada keypad di screenshot ini:



Menampilkan jumlah tip

Dalam file `MainActivity.kt`, fungsi `private calculateTip()` diberikan kepada Anda sebagai bagian dari kode awal. Anda akan menggunakan fungsi ini untuk menghitung jumlah tip:

```
private fun calculateTip(amount: Double, tipPercent: Double = 15.0):
String {
    val tip = tipPercent / 100 * amount
    return NumberFormat.getCurrencyInstance().format(tip)
}
```

Menggunakan fungsi `calculateTip()`

1. Dalam fungsi composable `EditNumberField()`, buat variabel baru bernama `amount` setelah definisi `amountInput`. Panggil fungsi `toDoubleOrNull` pada variabel `amountInput`, untuk mengonversi `String` menjadi `Double`:

```
val amount = amountInput.toDoubleOrNull()
```

2. Di akhir pernyataan, tambahkan operator Elvis `?:` yang menampilkan nilai `0.0` jika `amountInput` adalah null:

```
val amount = amountInput.toDoubleOrNull() ?: 0.0
```

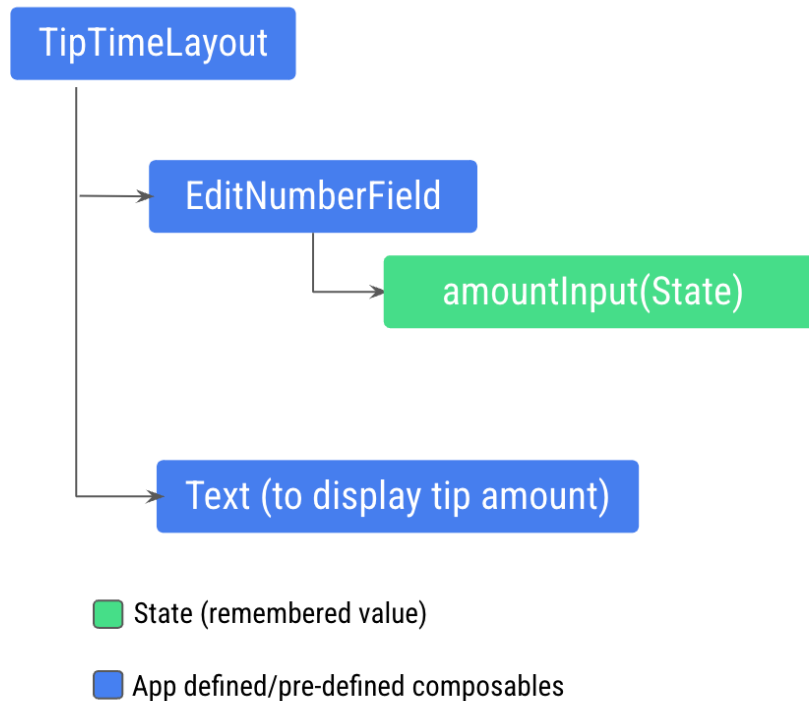
3. Setelah variabel `amount`, buat variabel `val` lain bernama `tip`. Lakukan inisialisasi dengan `calculateTip()`, dengan meneruskan parameter `amount`.

```
val tip = calculateTip(amount)
```

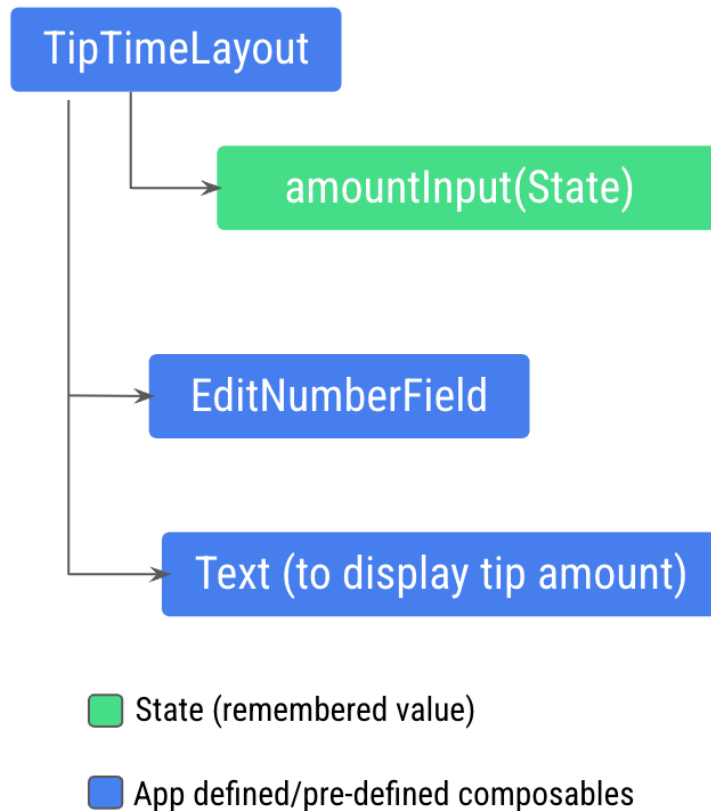
Menampilkan jumlah tip yang dihitung

1. Dalam fungsi `TipTimeLayout()` di akhir blok `Column()`, perhatikan composable teks yang menampilkan `$0.00`. Anda akan memperbarui nilai ini ke jumlah tip yang dihitung.

```
@Composable
fun TipTimeLayout() {
    Column(
        modifier = Modifier
            .statusBarsPadding()
            .padding(horizontal = 40.dp)
            .verticalScroll(rememberScrollState())
            .safeDrawingPadding(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    ) {
        // ...
        Text(
            text = stringResource(R.string.tip_amount, "$0.00"),
            style = MaterialTheme.typography.displaySmall
        )
        // ...
    }
}
```



4. Struktur ini tidak akan memungkinkan Anda menampilkan jumlah tip di composable `Text` baru karena composable `Text` perlu mengakses variabel `amount` yang dihitung dari variabel `amountInput`. Anda perlu mengekspos variabel `amount` ke fungsi `TipTimeLayout()`. Gambar ini mengilustrasikan struktur kode yang diinginkan yang membuat composable `EditNumberField()` menjadi stateless:



Pola ini disebut *pengangkatan status*.

Pengangkatan status

1. Perbarui definisi fungsi `EditNumberField()`, untuk mengangkat status dengan menambahkan parameter `value` dan `onValueChange`:

```
@Composable
fun EditNumberField(
    value: String,
    onValueChange: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    //...
```

2. Dalam fungsi `EditNumberField()`, perbarui fungsi composable `TextField()` untuk menggunakan parameter yang diteruskan:

```
TextField(
    value = value,
    onValueChange = onValueChange,
    // Rest of the code
)
```

3. Mengangkat status, memindahkan status yang diingat dari fungsi `EditNumberField()` ke fungsi `TipTimeLayout()`:

```
@Composable
fun TipTimeLayout() {
    var amountInput by remember { mutableStateOf("") }

    val amount = amountInput.toDoubleOrNull() ?: 0.0
    val tip = calculateTip(amount)

    Column(
        //...
    ) {
        //...
    }
}
```

4. Anda telah mengangkat status ke `TipTimeLayout()`, sekarang teruskan ke `EditNumberField()`. Dalam fungsi `TipTimeLayout()`, perbarui panggilan fungsi `EditNumberField()` untuk menggunakan status yang diangkat:

```
EditNumberField(
    value = amountInput,
    onValueChange = { amountInput = it },
    modifier = Modifier
        .padding(bottom = 32.dp)
        .fillMaxWidth()
)
```

Pemformatan posisi

1. Dalam fungsi, `TipTimeLayout()`, gunakan properti `tip` untuk menampilkan jumlah tip. Perbarui parameter `text` composable `Text` untuk menggunakan variabel `tip` sebagai parameter.

```
Text(
    text = stringResource(R.string.tip_amount, tip),
    // ...
)
```

Fungsi `TipTimeLayout()` dan `EditNumberField()` yang selesai akan terlihat seperti cuplikan kode ini:

```
@Composable
fun TipTimeLayout() {
    var amountInput by remember { mutableStateOf("") }
    val amount = amountInput.toDoubleOrNull() ?: 0.0
    val tip = calculateTip(amount)

    Column(
        modifier = Modifier
    ) {
        // ...
    }
}
```



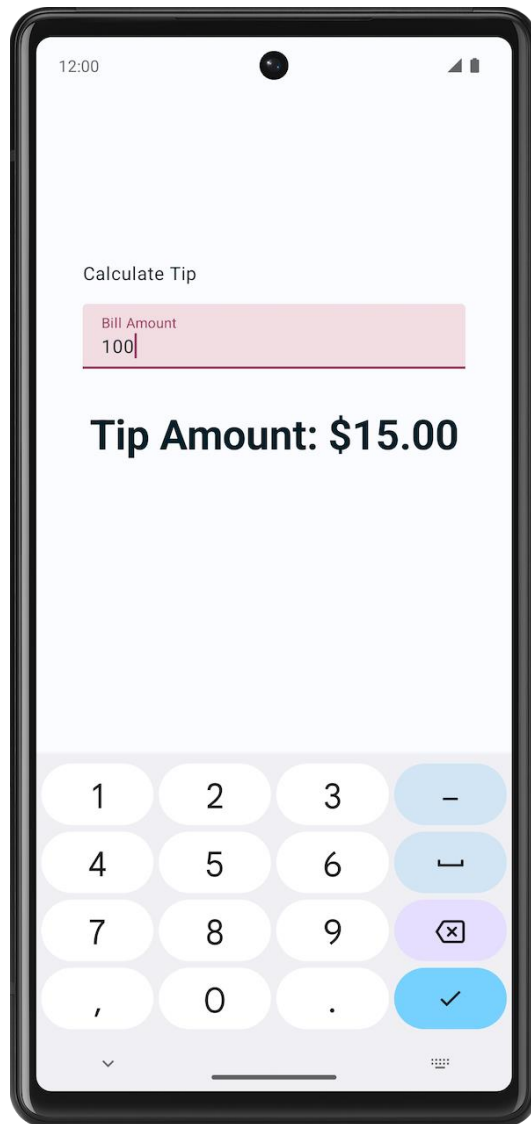
```

        .statusBarsPadding()
        .padding(horizontal = 40.dp)
        .verticalScroll(rememberScrollState())
        .safeDrawingPadding(),
horizontalAlignment = Alignment.CenterHorizontally,
verticalArrangement = Arrangement.Center
) {
    Text(
        text = stringResource(R.string.calculate_tip),
        modifier = Modifier
            .padding(bottom = 16.dp, top = 40.dp)
            .align(alignment = Alignment.Start)
    )
    EditNumberField(
        value = amountInput,
        onValueChange = { amountInput = it },
        modifier = Modifier
            .padding(bottom = 32.dp)
            .fillMaxWidth()
    )
    Text(
        text = stringResource(R.string.tip_amount, tip),
        style = MaterialTheme.typography.displaySmall
    )
    Spacer(modifier = Modifier.height(150.dp))
}
}

@Composable
fun EditNumberField(
    value: String,
    onValueChange: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    TextField(
        value = value,
        onValueChange = onValueChange,
        singleLine = true,
        label = { Text(stringResource(R.string.bill_amount)) },
        keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
        modifier = modifier
    )
}

```

2. Jalankan aplikasi di emulator atau perangkat, lalu masukkan nilai di kotak teks **jumlah tagihan**. Jumlah tip 15 persen dari jumlah tagihan ditampilkan seperti yang dapat Anda lihat pada gambar ini:



Sumber:

<https://developer.android.com/courses/pathways/android-basics-compose-unit-2-pathway-3?hl=id>