4B User inputs

Now that you've learned how to create input fields, we're going to put some in our Checkout widget. Don't forget that to make each field work, you'll need to:

- Add a class-level private property to hold the field value.
- Write each field with an onChanged event handler which will copy its value into the corresponding property.

Getting the scene ready

1. In your _CheckoutState class, add private String? properties for _firstName, _lastName, _email, _phone, _creditCardNumber, _CVV, _expiryMonth, _expiryYear

2. Inside the Scaffold at the root of the widget, put a SingleChildScrollView with a Column inside it. This way we can hold multiple things and can scroll it if needed.

3. Put a Text() that says "Your cart".

4. Put a Container() as a placeholder for the cart's contents and totals. Give it a child of a Text(). Again, it's just a placeholder.

We now know Flutter can get wordy; you often write a lot of code to accomplish a little. So it is often a good practice to either 1) break large widgets into smaller ones or 2) to create a private, internal lightweight function that returns a widget. Let's do the latter.

5. Create a function called _makeCheckoutForm() that returns a Widget. Have it return a simple Column() for now.

6. Call it instead of the Text("Form will go here") back in the build method.

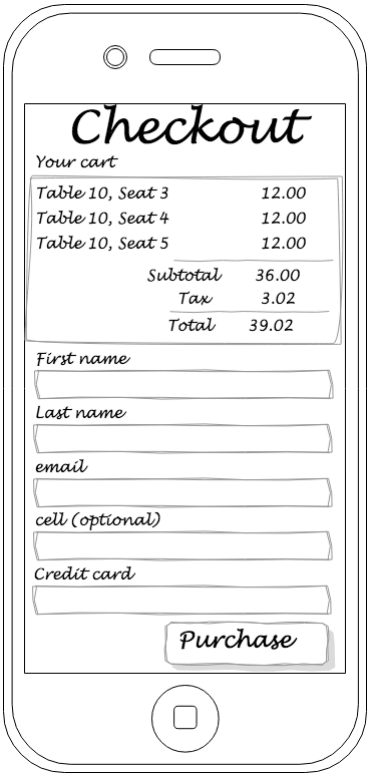
Try to do the above without peeking at the answer, but we're talking about something like this:

```
class _CheckoutState extends State<Checkout> {
  String? _firstName;
  String? _lastName;
  // Etc. Etc.
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Checkout"),
      ),
      body: SingleChildScrollView(
        child: Column(
          children: [
            Text("Checkout"),
            Text("Your cart"),
            Container(
```

```
              child: Text("Cart will go here"),
            ),
            _makeCheckoutForm(),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: const Icon(Icons.payment),
        onPressed: () => print("You pressed the FAB"),
      ),
    );
  }

  Widget _makeCheckoutForm() {
    return Column(
      children: [],
    );
  }
}
```

Having a separate function will allow us to focus on the form alone and also make our widget easier on our teammates to reason about. W clean coding!



Adding the fields

7. Add TextField() widgets for first name and last name. Just this will do:

```
Widget _makeCheckoutForm() {
  return Column(
    children: [
      TextField(),
      TextField(),
    ],
  );
}
```

8. Run and test. Put in some values.

You'll see that it totally works but it needs some UX love. Let's add some decorations.

9. Put labels on both.

10. Run and test. Looking better? Notice the behavior of the labels when the field gets focus.

11. Add a TextField() for email. Make the keyboard input an email keyboard. Give it a label and a hint text that says "you@yourEmail.com".

12. Add a TextField() for phone number. Make the keyboard input a phone keyboard. Its hint text should say "xxx-xxx-xxxx".

13. Add TextField() widgets for credit card number and CVV. Make the keyboard input numeric. The credit card hint text should be "xxxx-xxxx-xxxx-xxxx" and the CVV's hint text should be "Three-digit code on the back of the card".

Doing something with the values

So far we have the UI but we're not doing anything with them. Let's get the value that the user enters into the backing variables that we created earlier.

14. In the first name TextField(), create an onChanged event handler. All it needs to do is copy the value entered into its backing variable:

```
onChanged: (String val) {
  _firstName = val;
},
```

    or more concisely:

```
onChanged: (val) => _firstName = val,
```

15. Do the same for all the TextField()s.

Ready to see the values?

16. Create a new method in the class called _checkout().

17. _checkout() should just print out all of your values.

18. Attach _checkout() to the FAB's onPressed event handler.

19. Run and test. Do you see your values in the debug console?

Dropdowns

DropdownButtons are handled differently. They're easier on the user and easier on us when processing the data but they're a lot more wordy to create. They'd be perfect for the expiry month and year values.

20. Add a dropdown for the expiry month.
    Hints:
    - You'll have a DropdownButton of type String.
    - It'll have a value property which should be set to your holding variable, _expiryMonth.
    - It'll have an items property.
    - The items property will be a list of DropdownMenuItems.
    - Each of those will have a Text() and a value. For example the January option may look like this:

      ```
      DropdownMenuItem<String>(
        child: Text("Jan"),
        value: "01",
      ),
      ```

    - It'll have an onChanged property -- a function that receives a String val and assigns it to _expiryMonth.
    - onChanged works without setState, but if you want to see your value stick in the dropdown, wrap your val setting with setState().

21. Run and test, making sure your expiry month gets put in the proper variable.

22. Add a dropdown for the expiry year just like you did for the month.

23. Add them both to your _checkout() method.

24. Run and test.

25. Bonus!! If you have extra time, get clever and use a function to create your list of DropdownMenuItems() for the expiry year. Make it start with the current year and progress for the next 5 years. If you didn't do this, we'd have to manually change the list of expiry years every January 1st.

Now you've gotten the fields to appear and to look nice. The user can interact with them and can even read values but when it comes to save the data as a unit and/or validate our fields, there's more work to be done. We'll look at that in the next lab.