Designing Flutter

Key Points	Notes

1. Layout in Flutter

a. Flutter does **NOT** use any specific layout to place widgets. It depends on the widget that you use.

b. Some widgets are flexible and some require specific coordinates. You can mix and match between the two. However, you have to take note of the widget layout constraints and might end up getting `flutter layout infinite size` error.

2. Row and Column

a. Row and column are commonly used in Flutter and they are flexible similar to FlexBox in React Native.

b. Colum Example:

```
// _MyHomePageState
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      Text('You have pushed the button this many times:'),
      Text(
        '$_counter',
        style: Theme.of(context).textTheme.display1,
      ),
      RaisedButton(
        child: Text("Decrement Counter"),
        onPressed: _decrementCounter,
      ),
    ],
  ),
),

//The widget is aligned to the center of the column
```

Summary

Key Points	Notes

c. Row Example:

```
// _MyHomePageState.build
Row(                    // new
  children: <Widget>[   // new
    RaisedButton(
      color: Colors.red,
      child: Text(
        "Decrement",
        style: TextStyle(color: Colors.white),
      ),
      onPressed: _decrementCounter,
    ),
  ],                    // new
),                      // new

// All the widget will be aligned next to each other from
   left to right.
```

3. RenderObject

a. `RenderObject` is responsible for the actual painting to the screen by Flutter. It is used internally and you rarely use it directly in Flutter. However, it is good to understand how `RenderObject` works.

b. `RenderObject` has methods on it such as `performLayout` and `paint`. These methods are responsible to determine the layout of your widgets based on pixels of the screen.

c. All styling and layout in widgets that you use in Flutter are just abstraction over the render objects. `RenderObjectWidget` will render the whole widget tree and paint to the screen.

d. `Column` widget for example is just a layout, which you would **NOT** be able to see but `text` and `color` are concrete objects that can be painted on screen. The purpose of column is to provide constraints but not paint anything on the screen.

Summary

Key Points	Notes
	4. RenderObject and Constraints a. Render objects are closely tied to layout constraint and they are responsible to tell Flutter framework widget's actual physical size on the screen. b. For example, row will set a constraint that any widget placed in it will be placed in a row from the left to the right. The same goes with column. c. Constraints are concerned with `minWidth`, `minHeight`, `maxWidth`, `maxHeight`. Size is concerned with actual `width` and `height`. It will automatically allocate how much space it will take up. d. `RenderBox` is a subclass of `RenderObject`, which calculate a widget's size based on cartesian coordinate system (x,y). There are three (3) types of render boxes: i. Take as many spaces as possible such as `Center` widget. ii. Take the same size as their children such as `Opacity` widget. iii. Take to be in a particular size such as `Image` widget. **5. RenderBoxes and Layout Errors** a. The `flutter layout infinite size` error happens when a widget's constraints tell that it can be infinitely large on either the horizontal or vertical access (Width or Height). b. Some constraints are unbounded such as Row, Column and widgets that are scrollable. Even though Row and Column are flex boxes but their render objects are NOT. The render boxes are defined with bounded constraints.

Summary

Key Points	Notes

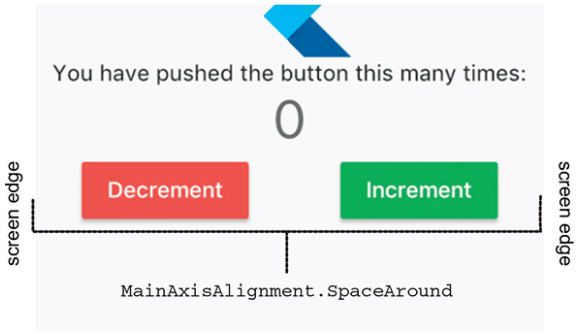
c. Example:

```
(Example:
child: Column(
 children: <Widget>[
   Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        Expanded(
          child: Text(
            'You have pushed the button this many times:',
          ),
        ),
      ],
    ),
  ],
),

// Error, column within another column causes infinite
space on the screen
```

6. Multi-Child Widgets
   a. To understand the basic idea of flexible layout constraint refer to the placement of Decrement and Increment buttons.

   b. In the example, `mainAxisAlignment` with `spaceAround` to create a space between the two buttons.



Row widget with spaceAround Alignment (Windmill, 2020)

Summary

Key Points	Notes

c. Example:

```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceAround,
  children: <Widget>[
    RaisedButton(
      color: Colors.red,
      child: Text(
        "Decrement",
        style: TextStyle(color: Colors.white),
      ),
      onPressed: _decrementCounter,
    ),
    RaisedButton(
      color: Colors.green,
      child: Text(
        "Increment",
        style: TextStyle(color: Colors.white),
      ),
      onPressed: _incrementCounter,
    ),
  ],
),
```

7. Icons and the FloatingActionButton

a. In Flutter, Material Design icons are built in and available without having to install external library.

b. The list of icons and themes are available at https://material.io/resources/icons/?style=baseline

c. Since Icons are constant you can call them everywhere in your app by passing the icon to the widget. This includes embedded an icon on top of a button with `FloatingActionButton`.

d. `FloatingActionButton` (FAB) is a circular action button and mostly used in the `Scaffold.floatingActionButton` field.

Summary

Key Points	Notes

e. Example:

```
floatingActionButton:  FloatingActionButton(
  onPressed: _resetCounter,
  tooltip: 'Reset Counter',
  child: Icon(Icons.refresh),
),
```

8. Images

a. `Image` widget makes it easier for you to include images in your Flutter application. It has different constructors depending on the source of your image.
   i. Images from Internet
   ii. Images from local project

b. For images from Internet you need use `Image.network` constructor. For example: `Image.network("https://funfreegifs.com/lion").`

c. For Images that are saved in local project folder you need use `Image.asset` constructor and modify `pubspec.yaml` to include the images path in the project folder.

d. Example:

```
//pubspec.yaml
flutter:
  uses-material-design: true
  assets:
    - flutter_logo_1080.png

//main.dart
children: <Widget>[
  Image.asset(
    'flutter_logo_1080.png',
    width: 100.0,
  ),
  Text(
    'You have pushed the button this many times:',
  ),
```

Summary

Key Points	Notes

9. Container Widget

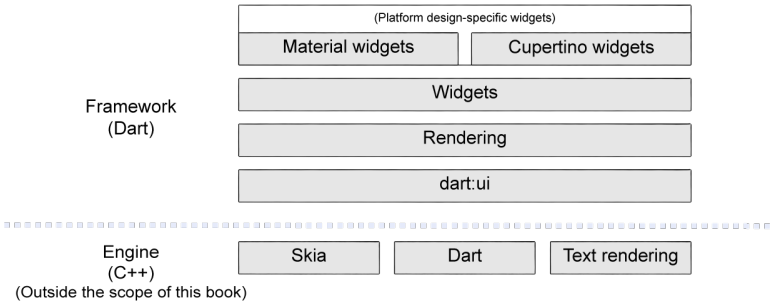
a. A `Container` widgets vary the way they size themselves based on the constructors arguments and children. By default they will try to be as big as possible. However, if a width is assigned they will try to size according to the width. If you need to style a widget you should use container widget.

b. Example:

```
Container(
  margin: EdgeInsets.only(bottom: 100.0),
  padding: EdgeInsets.all(8.0),
  decoration: BoxDecoration(
    color: Colors.blue.withOpacity(0.25),
    borderRadius: BorderRadius.circular(4.0),
  ),
  child: Image.asset(
    'flutter_logo_1080.png',
    width: 100.0,
  ),
),
```

10. Element Tree

a. To understand how widget works and how flutter render them under the surface you need to understand the layers of Flutter framework.



Flutter SDK Layers of Abstraction (Windmill, 2020)

Summary

Key Points	Notes

b. When we develop a Flutter application we will only deal with widgets. The widgets defined will be rendered based on the dart:ui library.

c. Element contains meta information and a reference to a widget. It changes and updates reference when the widget changes.

d. Widget tree is linked to element tree, which manages state objects.

11. Widget Keys

a. A widget `key` allows element tree to differentiate between the same widget. This will allow you to call the widget when it is needed.

b. Example:

```
_buttons = <Widget>[
  FancyButton(
    key: _buttonKeys.first,
    child: Text(
      "Decrement",
      style: TextStyle(color: Colors.white),
    ),
    onPressed: _decrementCounter,
  ),
  FancyButton(
    key: _buttonKeys.last,
    child: Text(
      "Increment",
      style: TextStyle(color: Colors.white),
    ),
    onPressed: _incrementCounter,
  ),
];
```

Summary

Key Points	Notes

c. Types of keys:

Key	Description
`GlobalKey`	Manage state and move widgets around the widget tree.
`ValueKey<T>`	When adding constant or unique property such as todo list app.
`ObjectKey`	Object with the same type but different property values. For example, products with the same title but different seller.
`UniqueKey`	When adding keys to children of a collection and the children do not know their values until they are created.
`PageStorageKey`	A specialized key used to store page information such as scroll location.

12. References

a. Windmill, E. (2020). Flutter in Action (1st Ed.). USA: Manning Publications.

b. Flutter Official Documentation. Retrieved on 1 December 2020 from https://flutter.dev/docs

Summary