Biomedical Informatics

Lefteris Koumakis,
Maria Chatzimina

Lecture 4

Class Map

- Map is a class of dart
- Uses key-value structure
- We declare it with { }
- Example:

```
var questions = [
  {
    'questionText': 'What\'s your favorite color?',
    'answers':'black',
  },
{
    'questionText': 'What\'s your favorite animal?',
    'answers': 'cat',
  },
];
```

Questions

- What is the difference between a list ([]) and a map ({}) in Dart / Flutter?
  - Lists give you an ordered list of individual values, identified by an index.
  - The map uses key-value pairs where we get the values based on their key.

  - Example:
    print(questions[0]['questionText']) prints “What’s your favorite color”

```
var questions = [
  {
    'questionText': 'What\'s your favorite color?',
    'answers': ['black', 'white', 'purple'],
  },
{
    'questionText': 'What\'s your favorite animal?',
    'answers': ['cat', 'dog', 'mouse'],
  },
];
```

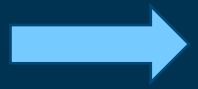
Final vs Const

- “Final“:
  - means that the variable is assigned once:
  - a final variable must be initialized. Once a value is assigned, the value of a final variable cannot be changed.
  - We know before the program runs that the value of this variable will not change but we still don't know the exact value. The value is assigned when the program is run
  - final birthday = getBirthDateFromDB()
- “Const”:
  - The value must be known at compile time,
    - const birthday = "2008/12/25“
  - Cannot be changed after initialization.
- “Static”:
  - static is used for a class-level variable that is the same for every instance of the class, this means that if the data is static, it can be accessed without creating an object.
    - Example the Colors class where we use the colors without creating an object:
    - static const MaterialColor deepPurple = MaterialColor(…..
    - color: Colors.deepPurple

One Line If statements

- If the condition we check is true, the code after the “?” is executed. while if it is false the code after the “:” is executed

```
void main() {
 var condition = true;
 condition ? print("Condition is true") : print("Condition is false");

  var value = (1>2)?"one is not greater thn 2":"2 is greater than 1";
  print(value);

}
```



```
Condition is true
2 is greater than 1
```

- Example with widgets:

```
body: index < myList.length
? Text("Index is less thn myList.length")
: Text("Index is greater than myList.length")
```

Drawer Widget

- Video and Details: https://api.flutter.dev/flutter/material/Drawer-class.html
- Example :

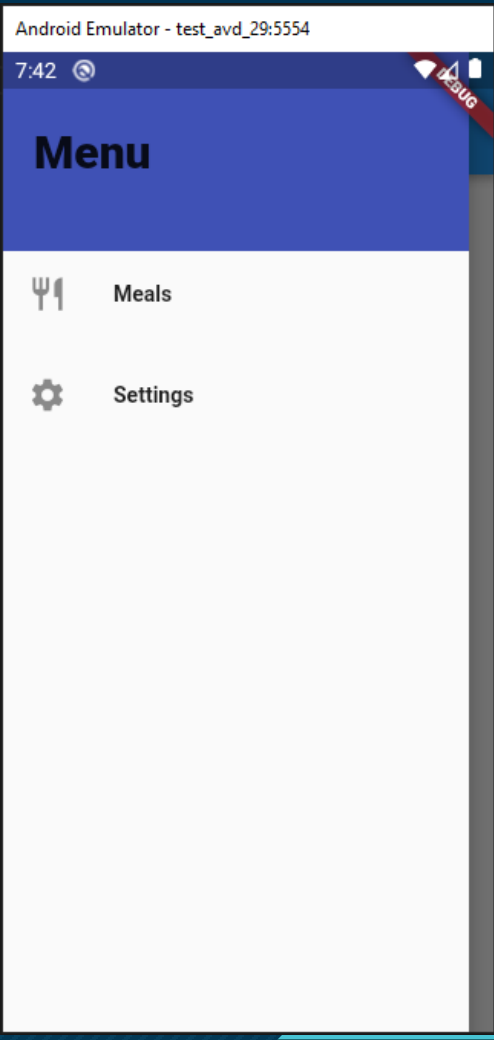
```
return Scaffold(
  appBar: AppBar(title: Text('Drawer')),
  body: Center(child: Text('My Page!')),
  drawer: Drawer(
    //Βαζουμε ενα Listview στο drawer. Γιατι θελουμε να ειμαστε σιγουροι
    // οτι ο χρηστης θα μπορει να κανει scroll σε ολα τα αντικειμενα
    // αν δεν χωρανε στην οθονη
    child: ListView(
      padding: EdgeInsets.zero,
      children: <Widget>[
        DrawerHeader(
          child: Text('Drawer Header'),
          decoration: BoxDecoration(
            color: Colors.blue,
          ), // BoxDecoration
        ), // DrawerHeader
        ListTile(
          title: Text('Item 1'),
          onTap: () {
            // κλεινουμε το drawer
            Navigator.pop(context);
          },
        ), // ListTile
        ListTile(
          title: Text('Item 2'),
          onTap: () {
            // κλεινουμε το drawer
            Navigator.pop(context);
          },
        ), // ListTile
      ], // <Widget>[]
    ), // ListView
  ), // Drawer
); // Scaffold
```

Custom Drawer

- ListTile builder

```dart
//builder για το ListTile
Widget buildListTile(String title, IconData icon, Function pageHandler) {
  return ListTile(
    leading: Icon(
      icon,
      size: 26,
    ), // Icon
    title: Text(
      title,
      style: TextStyle(
        fontWeight: FontWeight.bold,
      ), // TextStyle
    ), // Text
    onTap: pageHandler,
  ); // ListTile
}
```

```dart
@override
Widget build(BuildContext context) {
  return Drawer(
    child: Column(
      children: [
        Container(
          height: 130,
          width: double.infinity,
          padding: EdgeInsets.all(20),
          alignment: Alignment.centerLeft,
          color: Colors.indigo, //χρωμα του header του drawer
          // decoration:BoxDecoration(color:Colors.blue), διαφορετικος τροπος να περασουμε χρωμα
          child: Text(
            "Menu",
            style: TextStyle(
              fontWeight: FontWeight.w900,
              fontSize: 30,
            ), // TextStyle
          ), // Text
        ), // Container
        buildListTile("Meals", Icons.restaurant, () {
          Navigator.of(context)
              .pushNamed('/meals'); //η συναρτηση που περναμε στον pageHandler
        }),
        SizedBox(
          height: 10,
        ), //για να αφησουμε ενα κενο // SizedBox
        buildListTile("Settings", Icons.settings, () {
          Navigator.pushNamed(context,
              '/settings'); //η συναρτηση που περναμε στον pageHandler
        }),
      ],
    ), // Column
  ); // Drawer
}
```



Exercise 1: Drawer & Navigation

- Create a **Drawer menu** and navigate between screens
  - Add a ListTile for Home
  - Add a ListTile for Settings
  - Navigate to /settings when clicked
- Hints
  - Navigator.of(context).pushNamed('/settings’)
  - Use onTap
  - Add an Icon

Questions

- What about "Screens" and "regular Widgets"?
  - Both are regular widgets at the end, the only difference is how the widgets are used and what role they play.
- What is the difference between push() and pushNamed();
  - push() navigates to a new screen by creating it "on the fly", pushNamed() can only load screens that are registered in advance.
- What exactly is a “route”?
  - An object managed by a Navigator that represents a screen, typically implemented by classes like MaterialPageRoute
  - A route registered in the routes table - receives a "name" (key) with which it can be loaded.
- What is a "Stack of Pages" (or "Stack of Screens")?
  - New pages are usually promoted on top of the "Stack of Pages/Screens". The top (the last) page / screen is the visible screen. Removing the last screen (Popping) returns to an older screen.

Questions

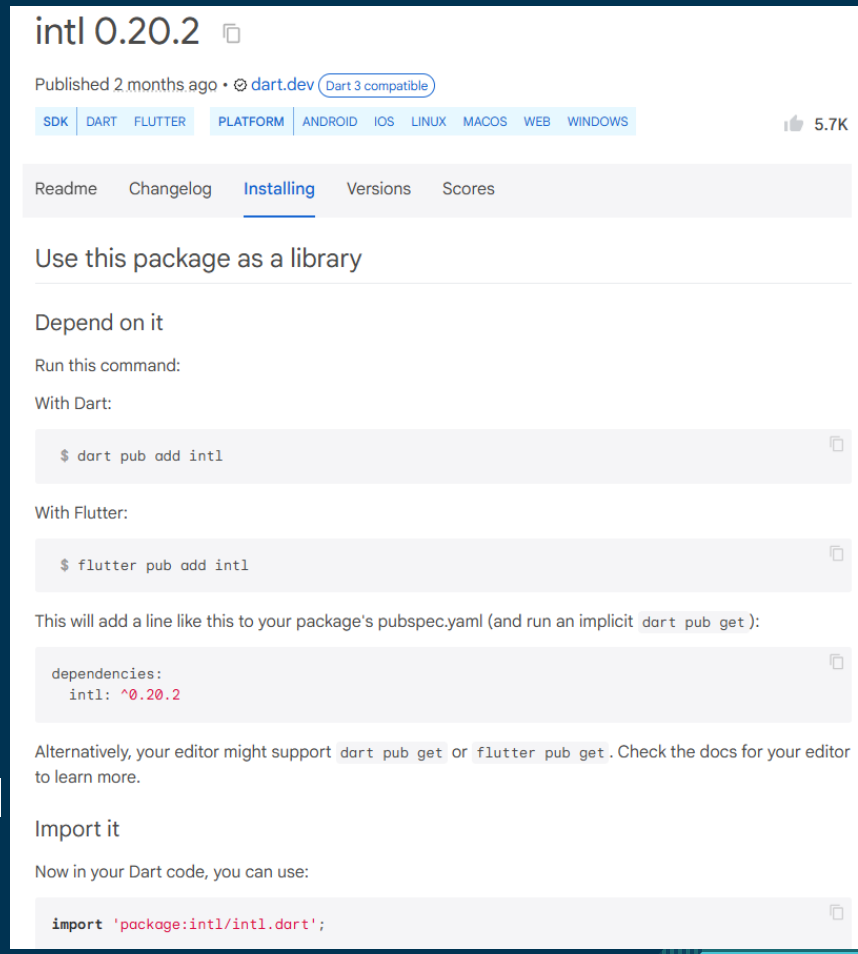
- What is the difference between using Tabs (regardless of which tabs) and using push() / pushNamed();
  - Tabs replace the current screen (or part of it) with a new one, push() / pushNamed() add a new screen to the stack.
- Which widget is important for both Tabs and Drawers?
  - The Scaffold widget - both are inserted there.

Installing External Packages

- Example: Format Date
  - Intl package → https://pub.dev/packages/intl
  - Pub.dev: Website with many packages that you can use in Flutter projects
  - Select “Installing” in the page which provides instructions
  - We copy what is written under dependencies and add it to pubspec.yaml just below the word flutter in the same level
  - dependencies:

    flutter

    intl: ^0.20.0
  - Save the file and flutter will install it automatically
  - We import it based on the instructions on the “Installing” page
  - If the package is not installed automatically, open the terminal and run
    - flutter packages get



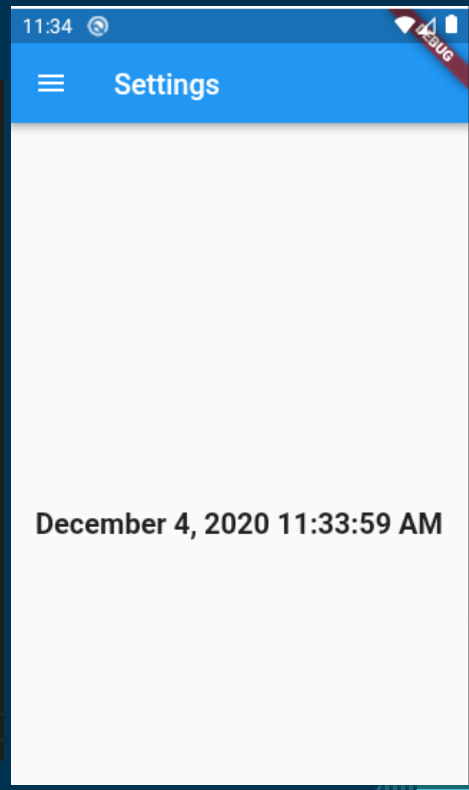
Date Format

- pubspec.yaml

```yaml
version: 1.0.0+1

environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  intl: ^0.20.2

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.0

dev_dependencies:
  flutter_test:
    sdk: flutter
```

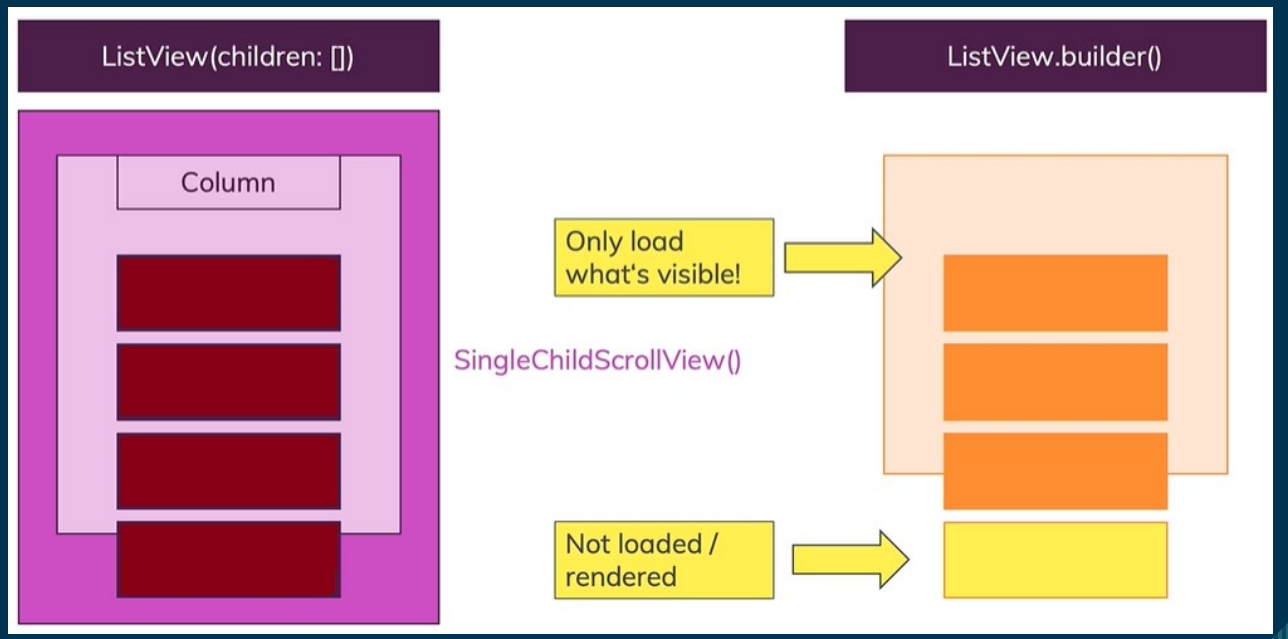
- settings.dart

```dart
import 'package:flutter/material.dart';
import './main_drawer.dart';
import 'package:intl/intl.dart';


class Settings extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title:Text("Settings")
      ), // AppBar
      drawer: MainDrawer(),
      body: Center(
        child: Text(
            DateFormat().format(DateTime.now()),
            style:TextStyle(fontSize: 20,fontWeight: FontWeight.bold),
        ), // Text
      ), // Center

    ); // Scaffold
  }
}
```



ListView Class

- ListView:
  - Infinity height
  - If it is inside a container we can define the height
  - Two ways to create it:
    - With named parameter children
      - Renders all τα widgets (visual and non-visual)
    - With a constructor .builder()
      - When we do not know the amount of objects we will have in the list or the number of objects is too large
      - Renders only visual widgets



Future Class

- Url: https://api.flutter.dev/flutter/dart-async/Future-class.html
- A Future is used to represent a potential value, or error, that will be available at some time in the future.
- It allows the creation of objects that will be given a value in the future
- We use it when we want to make HttpRequests, read json files etc.
- The Future class **represents a future result of an asynchronous computation**. This result will eventually appear in the Future after the processing is complete.
- The program will not wait for the value assignment to the future to complete but will execute normally
- FutureBuilder: Takes a named parameter “builder” that contains the current state of the future

Futures

- There are two ways to execute a “Future” and use the value it returns (If it returns anything). The most common way is to wait for the “Future” to return data. For this to work the function calling the code must be marked with “async”.
  - Example:

```
FlatButton(
  child: Text('Run Future'),
  onPressed: () async {
    var value = await myTypedFuture();
  },
)
```

- The other way to handle a “future” is to use the .then() function which is **a callback that's called when future completes successfully(with a value)**.
  - Example:

```
void runMyFuture() {
  myTypedFuture().then((value) {
    // Run the code here using the value
  });
}
```

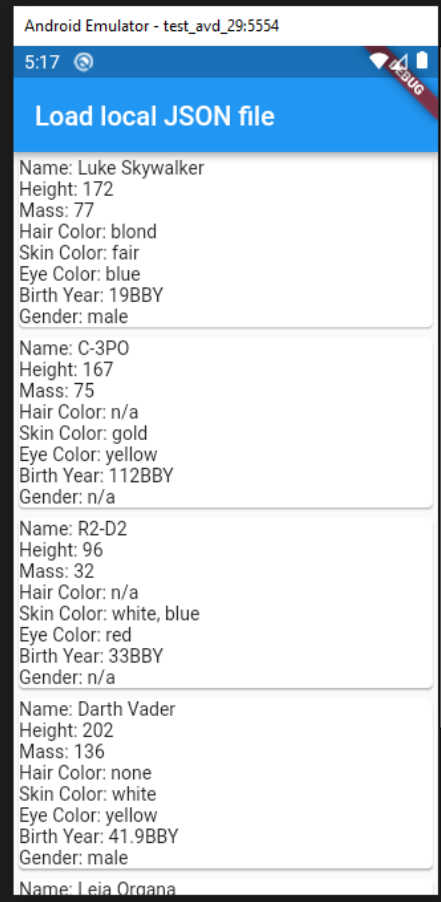
Synchronous and Asynchronous programming

- Synchronous programming is a type of programming where the program executes in a sequential manner, with each line of code being executed one after the other. The program must wait for each operation to complete before moving on to the next one. This can sometimes result in slower program execution.
- Asynchronous programming is a type of programming where tasks are executed independently and concurrently without waiting for each other to complete. It allows for better utilization of system resources and can improve performance. An asynchronous programming model allows multiple things to happen simultaneously. When an action/task starts, your program continues to run. When the action completes, the program is updated and accesses the result.

Read Local JSON file

- WE use DefaultAssetBundle to read the JSON file
- We use ListView.builder() because we don't know the number of objects to be created
- The named parameter "builder" in FutureBuilder has a snapshot of the future's current state.
  - snapshot.data : The data (if there are any)
  - snapshot.hasData : returns true/false if the snapshot has data or not
  - etc.

```
return Scaffold(
    appBar: AppBar(
      title: Text("Load local JSON file"),
    ), // AppBar
    body: Container(
      child: Center(
        // Χρησιμοποιουμε το FutureBuilder και το Future για να κανουμε load το Json
        child: FutureBuilder(
            future: DefaultAssetBundle // το future μπορει να παρει στο μελλον τιμη
                .of(context)
                .loadString('data_repo/starwars_Data.json'),
            builder: (context, snapshot) { //κραταει ενα snapshot της τωρινης καταστασης του future
              // Decode the JSON
              var newData = json.decode(snapshot.data.toString());

              return ListView.builder(
                // Χρησιμοποιουμε τον constructor builder γιατι δεν ξερουμε το πληθος των αντικειμενων
                itemBuilder: (BuildContext context, int index) {
                  return Card(
                    child: Column(
                      crossAxisAlignment: CrossAxisAlignment.stretch,
                      children: <Widget>[
                        Text("Name: " + newData[index]['name']),
                        Text("Height: " + newData[index]['height']),
                        Text("Mass: " + newData[index]['mass']),
                        Text(
                            "Hair Color: " + newData[index]['hair_color']), // Text
                        Text(
                            "Skin Color: " + newData[index]['skin_color']), // Text
                        Text(
                            "Eye Color: " + newData[index]['eye_color']), // Text
                        Text(
                            "Birth Year: " + newData[index]['birth_year']), // Text
                        Text("Gender: " + newData[index]['gender'])
                      ], // <Widget>[]
                    ), // Column
                  ); // Card
                },
                itemCount: newData == null ? 0 : newData.length, //ελεγχος αν η τιμη ειναι null αλλιως δινουμε
                //το πληθος των αντικειμενων της λιστας που θελουμε να γινουν build στο itemCount
              ); // ListView.builder
            }), // FutureBuilder
      ), // Center
    )); // Container // Scaffold
```



Exercise 2: JSON & FutureBuilder

- Load data from the **JSON file** in the folder and display it in a list
  - Load JSON using Future
  - Use FutureBuilder
  - Display name and age using ListView.builder

- Hints
  - snapshot.hasData
  - snapshot.data
  - data[index]['name']