UNIT:5 Fundamentals of Dart Programming for Flutter

24 May 2024 07:33

[39] Explain control flow in Dart.

- Control flow in Dart refers to the order in which individual statements, instructions, or function calls are executed or evaluated within a Dart program.

- **Here are the main control flow constructs in Dart:**

1. **Conditional Statements**

   - **if-else Statement:** Used to execute a block of code if a condition is true, and optionally another block if the condition is false.

```
if (condition) {
  // code to execute if condition is true
} else {
  // code to execute if condition is false
}
```

   - **switch-case Statement**: Used to execute one of many code blocks based on the value of a variable.

```
switch (variable) {
  case value1:
    // code to execute if variable == value1
    break;
  case value2:
    // code to execute if variable == value2
    break;
  default:
    // code to execute if none of the cases match
}
```

2. **Looping Statements**

   - **for Loop:** Executes a block of code a specific number of times.

```
for (int i = 0; i < 5; i++) {
  // code to execute 5 times
}
```

   - **for-in Loop:** Iterates over elements of a collection (like a list).

```
var numbers = [1, 2, 3];
for (var number in numbers) {
  // code to execute for each number
}
```

   - **while Loop:** Executes a block of code as long as a condition is true.

```
while (condition) {
  // code to execute as long as condition is true
}
```

   - **do-while Loop:** Similar to while loop, but ensures that the code block is executed at least once.

```
do {
  // code to execute at least once
} while (condition);
```

3. **Control Flow Keywords**

   - **break:** Exits the nearest enclosing loop or switch statement.

```
for (int i = 0; i < 5; i++) {
  if (i == 3) {
    break; // exit the loop when i is 3
```

```
  }
}
```

- **continue:** Skips the current iteration of the loop and moves to the next iteration.

```
for (int i = 0; i < 5; i++) {
  if (i == 3) {
    continue; // skip the rest of the loop when i is 3
  }
}
```

- **return:** Exits a function and optionally returns a value.

```
int add(int a, int b) {
  return a + b; // returns the sum of a and b
}
```

[40] Write a program to find the area of a circle using class and object in dart.

```
import 'dart:math';

// Define a class named Circle
class Circle {
        // Class property to hold the radius of the circle
        double radius;

        // Constructor to initialize the radius
        Circle(this.radius);

        // Method to calculate the area of the circle
        double getArea() {
          return pi * radius * radius;
        }
}

void main() {
        // Create an object of Circle class with radius 5.0
        Circle circle = Circle(5.0);

        // Calculate the area using the getArea method
        double area = circle.getArea();

        // Print the area of the circle
        print('The area of the circle with radius ${circle.radius} is $area');
}
```

[41] Write a program to calculate simple interest using class and object in Dart.

```
// Define a class named SimpleInterest
class SimpleInterest {
        // Class properties to hold principal, rate, and time
        double principal;
        double rate;
        double time;

        // Constructor to initialize the properties
        SimpleInterest(this.principal, this.rate, this.time);

        // Method to calculate simple interest
        double calculateInterest() {
          return (principal * rate * time) / 100;
        }
}
```

```
void main() {
        // Create an object of SimpleInterest class with principal 1000, rate 5, and time 2 years
        SimpleInterest si = SimpleInterest(1000, 5, 2);

        // Calculate the simple interest using the calculateInterest method
        double interest = si.calculateInterest();

        // Print the simple interest
        print('The simple interest for a principal of ${si.principal}, rate of ${si.rate}%, and time of ${si.time} years is $interest');
}
```

[42] Differentiate positional arguments and named arguments in dart with examples.

- **Positional Arguments**
  - Positional arguments are passed to a function in the order they are defined. They are the most common way to pass arguments and are similar to how arguments are passed in many other programming languages.
  - **Example :**

```
void printDetails(String name, int age) {
  print('Name: $name, Age: $age');
}

void main() {
  // Calling the function with positional arguments
  printDetails('Alice', 30);
}
```

- **Named Arguments**
  - Named arguments are passed to a function using parameter names, allowing you to specify which arguments you are passing. Named arguments make function calls more readable and flexible, especially when a function has multiple parameters or when only some parameters need to be provided.
  - **Example :**

```
void printDetails({String name, int age}) {
  print('Name: $name, Age: $age');
}

void main() {
  // Calling the function with named arguments
  printDetails(name: 'Bob', age: 25);
  // Named arguments can be passed in any order
  printDetails(age: 40, name: 'Carol');
}
```

[43] Write a program to find even no in 1 to 10 in dart using class and object.

```
// Define a class named EvenNumbersFinder
class EvenNumbersFinder {
        // Method to find and return even numbers in a given range
        List<int> findEvenNumbers(int start, int end) {
                List<int> evenNumbers = [];
                for (int i = start; i <= end; i++) {
                        if (i % 2 == 0) {
                          evenNumbers.add(i);
                        }
                }
         return evenNumbers;
        }
}
```

```
void main() {
    // Create an object of EvenNumbersFinder class
    EvenNumbersFinder finder = EvenNumbersFinder();

    // Find even numbers between 1 and 10 using the findEvenNumbers method
    List<int> evenNumbers = finder.findEvenNumbers(1, 10);

    // Print the even numbers
    print('Even numbers between 1 and 10: $evenNumbers');
}
```

[44] How to write functions in Dart? Explain with an example.

- Functions are a set of statements that performs a specific task.
- The function declaration contains the function name, return type, and parameters.

1. **Basic Function**

   A basic function in Dart is defined using the void keyword if it does not return a value, or specifying a return type if it does.

```
void greet() {
    print('Hello, World!');
}

void main() {
    greet();
}
```

2. **Function with Parameters**

   Functions can take parameters, which are variables that are passed to the function when it is called.

```
void add(int a, int b) {
    int sum = a + b;
    print('Sum: $sum');
}

void main() {
    add(3, 5);
}
```

3. **Function with Return Value**

   Functions can return a value using the return keyword.

```
int multiply(int a, int b) {
    return a * b;
}

void main() {
    int result = multiply(4, 6);
    print('Product: $result');
}
```

4. **Default Parameter Values**

   Named parameters can also have default values.

```
void describe({required String name, String hobby = 'None'}) {
    print('Name: $name');
    print('Hobby: $hobby');
}

void main() {
    describe(name: 'Alice');
```

```
        describe(name: 'Bob', hobby: 'Reading');
    }
```

5. **Arrow Functions**

    For functions that contain a single expression, you can use arrow syntax (=>) for a more concise syntax.

```
    int square(int num) => num * num;

    void main() {
        int result = square(5);
        print('Square: $result');
    }
```

[45] Explain inheritance in Dart.

- Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class (called a subclass or derived class) to inherit properties and methods from another class (called a superclass or base class).
- In Dart, inheritance enables code reuse and the creation of a hierarchical relationship between classes.

- Example :

```
// Define a superclass named Animal
class Animal {
    // Property of the superclass
    String name;

    // Constructor of the superclass
    Animal(this.name);

    // Method of the superclass
    void eat() {
        print('$name is eating.');
    }
}

// Define a subclass named Dog that extends Animal
class Dog extends Animal {
    // Additional property of the subclass
    String breed;

    // Constructor of the subclass
    Dog(String name, this.breed) : super(name);

    // Additional method of the subclass
    void bark() {
        print('$name is barking.');
    }
}

void main() {
    // Create an object of the subclass Dog
    Dog myDog = Dog('Buddy', 'Golden Retriever');

    // Accessing properties and methods from the superclass
    myDog.eat();

    // Accessing properties and methods from the subclass
    myDog.bark();

    // Printing properties
    print('My dog\'s name is ${myDog.name} and breed is ${myDog.breed}.');
}
```

[46] What are the data types used in Dart?

- Dart offers a variety of built-in data types, the primary data types used in Dart:

1. Numbers

a. Integer (int)

- Represents whole numbers.

```
int age = 25;
int year = 2024;
```

b. Double (double)

- Represents fractional numbers (floating-point).

```
double height = 5.9;
double weight = 70.5;
```

2**. Strings**

String (String)

- Represents sequences of characters (text).
- Can be enclosed in single or double quotes.
- Supports string interpolation.

```
String name = 'Alice';
String greeting = "Hello, $name!";
```

3. Booleans (bool)

- Represents true or false values.

```
bool isLoggedIn = true;
bool isAvailable = false;
```

4. Lists

List (List<T>)

- Represents an ordered collection of objects.
- Can be homogeneous (all elements of the same type).
- Can be growable (default) or fixed-length.

```
List<int> numbers = [1, 2, 3, 4, 5];
List<String> names = ['Alice', 'Bob', 'Charlie'];
```

5. Sets

Set (Set<T>)

- Represents an unordered collection of unique objects.

```
Set<String> fruits = {'apple', 'banana', 'orange'};
```

6. Maps

Map (Map<K, V>)

- Represents a collection of key-value pairs.
- Keys are unique, values can be duplicated.

```
Map<String, int> scores = {
  'Alice': 90,
  'Bob': 85,
  'Charlie': 95
};
```

7. Null (null)

- Represents the absence of a value.
- By default, variables in Dart cannot be null unless specified using the ? operator for nullable types.

```
String? nullableString = null;
```

[47] How to create a function in Dart?

<--- Refer this answer --->

[48] Explain the class and object in Dart and how to create it.

- **Class :**
  - A class in Dart is a blueprint for creating objects. It encapsulates data (attributes) and behavior (methods) that define the characteristics and actions of objects.

```
class ClassName {
    // Fields (variables)
    type fieldName = value;

    // Constructor
    ClassName(constructorParameters) {
        // Constructor body
    }

    // Methods
    returnType methodName(methodParameters) {
        // Method body
    }
}
```

- **Object :**
  - An object is an instance of a class. It represents a specific instance of the class, with its own unique data and state.

```
ClassName objectName = ClassName(constructorArguments);
```

- **Example :**

```
// Define a class named Person
class Person {
    // Fields
    String name;
    int age;

    // Constructor
    Person(this.name, this.age);

    // Method
    void sayHello() {
        print('Hello, my name is $name and I am $age years old.');
    }
}

void main() {
    // Create an object of the Person class
    Person person1 = Person('Alice', 30);

    // Accessing object fields and methods
    print('Name: ${person1.name}');
    print('Age: ${person1.age}');
    person1.sayHello(); // Call the method
}
```

UNIT:7 Working with UI in Flutter

24 May 2024 08:33

[59] What are widgets, containers, and rows/columns, and how are they used in UI development?

- **Widgets :**
  - Widgets are objects used to construct the user interface in Flutter.
  - Each widget is an immutable declaration of part of the user interface, such as a button, text, image, or layout constraint.
- **Containers :**
  - Containers are layout widgets used to contain and position other widgets within them.
  - They provide properties like padding, margin, alignment, and decoration to control the layout and appearance of child widgets.
- **Rows and Columns :**
  - Rows and columns are layout widgets used to arrange child widgets horizontally (in a row) or vertically (in a column).
  - They automatically size their children according to their orientation and allow for easy alignment and spacing between widgets.
- **Usage in UI Development :**
  - Widgets, containers, rows, and columns are used extensively in Flutter UI development to create responsive and visually appealing layouts.
  - They allow developers to arrange and style widgets in various configurations to achieve the desired user interface design.
  - By combining these layout widgets with other widgets such as text, images, buttons, and input fields, developers can create complex and interactive user interfaces for their Flutter applications.

[60] Describe various widgets available in Flutter.

1. Text Widgets

- Text: Displays a string of text with styling options.
- RichText: Displays text with varying styles within the same widget.

2. Layout Widgets

- Container: A widget for controlling the layout and appearance of its child widget.
- Row: Arranges child widgets in a horizontal line.
- Column: Arranges child widgets in a vertical line.
- Stack: Overlays child widgets on top of each other.
- ListView: Displays a scrollable list of child widgets.
- GridView: Displays a scrollable grid of child widgets.

3. Input Widgets

- TextField: A widget for accepting text input from the user.
- Checkbox: A widget for toggling between checked and unchecked states.
- Radio: A widget for selecting a single option from a list of options.
- Switch: A widget for toggling between two mutually exclusive states.

4. Button Widgets

- ElevatedButton: A raised button with a material design style.
- TextButton: A text button with customizable text and style.
- OutlinedButton: A button with an outlined border and customizable style.

5. Image Widgets

- Image: Displays an image from various sources such as assets, network, or memory.
- Icon: Displays a material design icon.

6. Navigation Widgets

- Navigator: Manages a stack of routes and allows navigation between them.
- AppBar: A material design app bar with various customization options.
- BottomNavigationBar: A widget for navigating between multiple views using a bottom navigation bar.

7. Dialog Widgets

- AlertDialog: Displays an alert dialog box with customizable content.
- SimpleDialog: Displays a simple dialog box with a list of options.

8. Animation Widgets

- AnimatedContainer: A container that animates its properties when they change.
- AnimatedOpacity: An opacity widget that animates its opacity when it changes.
- AnimatedBuilder: A widget that rebuilds its child widget in response to an animation.

9. Other Widgets

- Padding: Adds padding around its child widget.
- SizedBox: A box with a fixed size.
- Expanded: A widget that expands to fill available space.
- ClipRRect: Clips its child widget to a rounded rectangle.

[61] What is the difference between row and column in Flutter?

- **Row :**

  - Primary Axis: Horizontal (from left to right).
  - Orientation: Arranges child widgets in a row, with each widget positioned side by side horizontally.
  - Flexibility: Children are stretched vertically to match the height of the tallest widget.
  - Use Cases: Suitable for arranging elements horizontally, such as buttons in a toolbar, navigation links, or form fields.

- **Column :**

  - Primary Axis: Vertical (from top to bottom).
  - Orientation: Arranges child widgets in a column, with each widget positioned one below the other vertically.
  - Flexibility: Children are stretched horizontally to match the width of the widest widget.
  - Use Cases: Ideal for arranging elements vertically, such as lists, menus, or sections of a page.

- **Key Differences :**

  - Direction: Row arranges widgets horizontally, while Column arranges them vertically.
  - Primary Axis: Row's primary axis is horizontal, and Column's primary axis is vertical.
  - Child Alignment: In a Row, children are stretched vertically to match the height of the tallest widget, while in a Column, children are stretched horizontally to match the width of the widest widget.

[62] What is a Container in Flutter? How to use it?

- In Flutter, a Container is a layout widget used to control the layout and appearance of its child widget.
- It provides properties for styling, padding, margin, alignment, decoration, and more, allowing developers to customize the vi sual presentation of the child widget.

- **Container Properties :**

  a. Alignment: Defines the alignment of the child widget within the container.
  b. Padding: Adds padding around the child widget.
  c. Margin: Adds margin around the container itself.
  d. Decoration: Allows customizing the appearance of the container using BoxDecoration, such as color, border, borderRadius, boxShadow, etc.
  e. Width and Height: Sets the width and height of the container.
  f. Constraints: Allows specifying minimum and maximum width and height constraints for the container.
  g. Transform: Applies transformations like rotation, translation, and scaling to the container.
  h. Clip Behavior: Controls how the container's child is clipped when it overflows.

- **How to Use Container :**

```
Container(
        alignment: Alignment.center,
        padding: EdgeInsets.all(20.0),
        margin: EdgeInsets.symmetric(vertical: 10.0),
        width: 200.0,
        height: 100.0,
        decoration: BoxDecoration(
                color: Colors.blue,
                borderRadius: BorderRadius.circular(10.0),
                boxShadow: [
                        BoxShadow(
```

```
                        color: Colors.grey,
                        offset: Offset(0, 3),
                        blurRadius: 6.0,
                    ),
                ],
            ),
            child: Text(
                'Hello, Container!',
                style: TextStyle(color: Colors.white),
            ),
    )
```

[63] Explain Row and Column and container in Flutter.

- **Row :**

  - Purpose: Row is a layout widget used to arrange child widgets horizontally in a row.
  - Usage: Suitable for arranging elements side by side horizontally, such as buttons, text fields, or icons in a toolbar.
  - Properties: Allows customization of alignment, spacing, and sizing of child widgets within the row.

```
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    Icon(Icons.home),
    Text('Home'),
    Icon(Icons.settings),
    Text('Settings'),
  ],
)
```

- **Column :**

  - Purpose: Column is a layout widget used to arrange child widgets vertically in a column.
  - Usage: Ideal for arranging elements vertically, such as lists, menus, or sections of a page.
  - Properties: Allows customization of alignment, spacing, and sizing of child widgets within the column.

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    Text('Name: Alice'),
    Text('Age: 30'),
  ],
)
```

- **Container :**

  - Purpose: Container is a layout widget used to control the layout and appearance of its child widget.
  - Usage: Provides properties for styling, padding, margin, alignment, decoration, and more, allowing developers to customize the visual presentation of the child widget.
  - Properties: Offers a wide range of properties, including alignment, padding, margin, decoration, width, height, constraints, and more.

```
Container(
  alignment: Alignment.center,
  padding: EdgeInsets.all(20.0),
  margin: EdgeInsets.symmetric(vertical: 10.0),
  width: 200.0,
  height: 100.0,
  decoration: BoxDecoration(
    color: Colors.blue,
    borderRadius: BorderRadius.circular(10.0),
  ),
  child: Text(
    'Hello, Container!',
    style: TextStyle(color: Colors.white),
  ),
)
```

[64] How to create button in flutter.

1. **ElevatedButton :**

   - The ElevatedButton widget creates a material design button with a raised appearance.

```
ElevatedButton(
	onPressed: () {
		// Add your button onPressed logic here
	},
	child: Text('Elevated Button'),
)
```

2. **TextButton :**

   - The TextButton widget creates a button with plain text.

```
TextButton(
	onPressed: () {
		// Add your button onPressed logic here
	},
	child: Text('Text Button'),
)
```

3. **OutlinedButton :**

   - The OutlinedButton widget creates a button with an outlined border.

```
OutlinedButton(
	onPressed: () {
		// Add your button onPressed logic here
	},
	child: Text('Outlined Button'),
)
```

- **Handling Button Press :**

  - To add functionality to the button, you can use the onPressed property and specify a function that will be called when the button is pressed.

```
ElevatedButton(
	onPressed: () {
		// Function to execute when the button is pressed
		print('Button pressed');
	},
	child: Text('Press Me'),
)
```

[66] How to display text in Flutter? How to increase text size in Flutter?

- **Display Text :**

  - To display text in Flutter, use the Text widget and provide the desired text content as the child property:

```
Text(
	'Hello, Flutter!',
)
```

- **Increase Text Size :**

  - To increase the text size, specify the fontSize property within the style property of the Text widget:

```
Text(
	'Hello, Flutter!',
	style: TextStyle(
		fontSize: 20, // Increase text size to 20
	),
)
```

[67] How to give font weight to text in Flutter ?

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

void main() {
	runApp(MaterialApp(
```

```
            home: Scaffold(
                appBar: AppBar(title: Text('Font Weight Example')),
                    body: Center(
                        child: Column(
                            mainAxisAlignment: MainAxisAlignment.center,
                            children: <Widget>[
                                Text('Normal Text'),
                                Text('Bold Text', style: TextStyle(fontWeight: FontWeight.bold)),
                                Text('Semi-Bold Text', style: TextStyle(fontWeight: FontWeight.w600)),
                                Text('Thin Text', style: TextStyle(fontWeight: FontWeight.w100)),
                            ],
                        ),
                    ),
                ),
            )
        );
}
```

[68] What is a text widget?

- The Text widget can display a string of text with customizable properties such as font size, color, font weight, alignment, text direction, overflow behavior, and more.
- It's versatile and can be used to render simple text as well as more complex text layouts within Flutter applications.

- Example :

```
Text(
    'Hello, Flutter!',
    style: TextStyle(
        fontSize: 20, // Set font size to 20
        fontWeight: FontWeight.bold, // Set font weight to bold
        color: Colors.blue, // Set text color to blue
    ),
)
```

[69] How to set the font family in Flutter?

1. Add the Font to Your Project

- First, create an assets directory in the root of your project if it doesn't exist. Then, place your font files (e.g., .ttf or .otf files) inside this directory.

```
your_project/
  assets/
    fonts/
      CustomFont-Regular.ttf
      CustomFont-Bold.ttf
```

2. Declare the Font in pubspec.yaml

- Open your pubspec.yaml file and declare the fonts under the flutter section

```
flutter:
  fonts:
    - family: CustomFont
      fonts:
        - asset: assets/fonts/CustomFont-Regular.ttf
        - asset: assets/fonts/CustomFont-Bold.ttf
          weight: 700
```

3. Use the Font in Your Widgets

- Text('Hello, Custom Font!', style: TextStyle(fontFamily: 'CustomFont', fontSize: 24,))

[70] how to add images in the flutter app.

1. Add the Image to Your Project

First, create an assets directory in the root of your project if it doesn't already exist. Then, place your image files (e.g., .png, .jpg) inside this directory.

```
your_project/
  assets/
    images/
      my_image.png
```

2. Declare the Image in pubspec.yaml

Open your pubspec.yaml file and declare the images under the flutter section

```
flutter:
  assets:
    - assets/images/my_image.png
```

3. Use the Image in Your Widgets

```
home: Scaffold(
        appBar: AppBar(title: Text('Image Example')),
        body: Center( child: Image.asset('assets/images/my_image.png')),
)
```

[71] Explain Dialogs in Flutter.

- dialogs are pop-up interfaces that require user interaction and typically appear in front of the current screen content to provide critical information, ask for user decisions, or display other temporary content.
- Flutter provides various types of dialogs, such as AlertDialog, SimpleDialog, and custom dialogs.

- **Common Types of Dialogs in Flutter :**

1. AlertDialog
2. SimpleDialog
3. Custom Dialog

1. AlertDialog

- AlertDialog is a material design dialog that can show a title, content, and actions (such as buttons).

```
return AlertDialog(
        title: Text('Alert'),
        content: Text('This is an alert dialog.'),
        actions: <Widget>[
                TextButton(
                        onPressed: () {
                                Navigator.of(context).pop();
                        },
                        child: Text('OK'),
                ),
        )
```

2. SimpleDialog

- SimpleDialog is a simpler dialog for presenting a list of options.

```
return SimpleDialog(
  title: Text('Choose an option'),
  children: <Widget>[
    SimpleDialogOption(
      onPressed: () {
        Navigator.of(context).pop();
      },
      child: Text('Option 1'),
    ),
    SimpleDialogOption(
      onPressed: () {
        Navigator.of(context).pop();
      },
      child: Text('Option 2'),
    )
```

3. Custom Dialog

- Custom dialogs can be created by using the Dialog widget and providing a custom content layout.

[72] Explain Navigations in Flutter.

- In Flutter, navigation refers to the process of managing the stack of pages (or routes) in an app, allowing users to move between different screens.
- Flutter provides several ways to implement navigation

1. Basic Navigation with Navigator

- The Navigator widget manages a stack of Route objects and provides methods to navigate to different routes.

a. Pushing a New Route

- To navigate to a new screen, you push a new route onto the stack:

```
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => NewScreen()),
);
```

b. Popping a Route

- To go back to the previous screen, you pop the current route off the stack:

```
Navigator.pop(context);
```

2. Named Routes

- Named routes allow you to define routes with string identifiers and manage navigation using those identifiers.

a. Define Routes in MaterialApp

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (context) => MainScreen(),
    '/second': (context) => SecondScreen(),
  },
);
```

b. Navigate Using Named Routes

```
Navigator.pushNamed(context, '/second');
Navigator.pop(context);
```

UNIT:8 Publishing App in Android

24 May 2024 13:01

[76] How can you publish your application in Google Play Store? Explain the entire 07 process.

Step 1: Make a Developer Account

i. Sign-In with Your Google Account
ii. Accept Terms
iii. Pay Registration Fee of $25.
iv. Complete Your Account Details

Step 2: Create a New App Listing

Step 3: Upload Your APK

Step 4: Set Pricing and Distribution

Step 5: Submit Your App for Review

Step 6: Manage Your App Listing

Step 7: Promote Your App

[77] Write steps to create an android application for Movie Ticket Booking.

a. Define Requirements
b. Design User Interface (UI)
c. Set Up Development Environment
d. Implement UI Layouts
e. Implement Functionality
f. Integrate External APIs
g. Implement Data Persistence
h. Test the App
i. Deploy and Publish
j. Monitor and Update