

Lecture 4 Module 1

Dart I/O, Classes ,Constructors-default ,parameterized,

Dart – Standard Input Output

- * **Standard Input in Dart:**
 - * **.readLineSync()** method to read
 - * To take input from the console you need to import a library, named **dart:io** from libraries of Dart.
- * **Standard Output in Dart:**
 - * In dart, there are two ways to display output in the console:
 - * Using **print** statement.
 - * Using **stdout.write()** statement.
 - * The **print()** statement brings the cursor to next line while **stdout.write()** don't bring the cursor to the next line, it remains in the same line.

Console input/output

usrinp.dart > main

```
1 import 'dart:io';
Run | Debug
2 void main() {
3     // Asking for Student details
4     print("Enter your Name:");
5     var name = stdin.readLineSync();
6     print("Enter your age:");
7     var age = int.parse(stdin.readLineSync()); // Scanning number
8     print("---STUDENT DETAILS---");
9     stdout.write("Your Name is $name ");
10    print("age is $age");
11    print("--No more details to display--");
12 }
```

```
Enter your Name:
leena
Enter your age:
30
---STUDENT DETAILS---
Your Name is leena age is 30
--No more details to display--
```

Classes and objects

Class and objects

Default Constructor

Parameterized Constructor

Named Constructor

Default Constructor

DartPad <> New Pad C Reset ┌ Format └ Install SDK

void main() {
 var student1 = Student();
 student1.id = 111;
 student1.name = "leena";
 print("\${student1.id} and \${student1.name}");
 student1.study(); student1.code();
}
class Student {
 int id = -1;
 String name;
 Student() {
 print("Not Work if parameterized constructor is present");
 }
 void study() {
 print("\${this.name} is now studying");
 }

 void code() {
 print("\${this.name} is now coding");
 }
}

▶ RUN

Console

Not Work if parameterized constructor is present
111 and leena
leena is now studying
leena is now coding

Documentation

Parameterized constructor

The screenshot shows a DartPad interface with the following code:

```
void main() {
  var student2 = Student(26, "Anjali");
  print("${student2.id} and ${student2.name}");
  student2.study();
  student2.code();
}

class Student {
  int id = -1;
  String name;

  Student(var id, var name) {
    this.id = id;
    this.name = name;
  }
  void study() {
    print("${this.name} is now studying");
  }

  void code() {
    print("${this.name} is now coding");
  }
}
```

A yellow brace on the left side of the constructor definition groups it with the constructor call in the main function. A yellow box highlights the constructor call `Student(this.id, this.name);`. The DartPad interface includes a RUN button, a CONSOLE output area, and a Documentation section for the `int id` field.

Console

```
26 and Anjali
Anjali is now studying
Anjali is now coding
```

Documentation

```
int id
```

main.dart

Named or Custom Constructors

Named Constructor: As you can't define multiple constructors with the same name, named constructor is the solution to the problem. They allow the user to make multiple constructors with a different name.

```
void main() {
    var student2 = Student(26, "Anjali");
    print("${student2.id} and ${student2.name}");
    student2.study(); student2.code();
    var stu1=Student.myCustomConstructor();
    stu1.study();stu1.code();
    var stu2=Student.myAnotherNamedConstructor(31, "Gopal");
    print("${stu2.id} and ${stu2.name}");
    stu2.code();
}
class Student {
    int id ;
    String name;
    Student(this.id,this.name){
        print("--- parameterized constructor called---");
    }
    Student.myCustomConstructor() {
        print("---my custom constructor called---");
        this.id=000; this.name="new student";
    }
    Student.myAnotherNamedConstructor(this.id, this.name)
    {print("---my another custom constructor called---");}
    void study() {
        print("${this.name} is now studying");
    }
    void code() {
        print("${this.name} is now coding");
    }
}
```

▶ RUN

Console

```
--- parameterized constructor called---
26 and Anjali
Anjali is now studying
Anjali is now coding
---my custom constructor called---
new student is now studying
new student is now coding
---my another custom constructor called---
31 and Gopal
Gopal is now coding
```

Documentation

getter/setter in dart

```
void main() {  
    var student = Student();  
    student.name = "Leena"; //Calling default Setter  
    print("NAME:: ${student.name}"); //Calling default Getter  
  
    student.percentage = 418.0; //Calling Custom Setter  
    print("PERCENTAGE:: ${student.percentage}"); //Calling Custom Getter  
}  
  
class Student {  
    String name; // Instance Variable with default Getter and Setter  
  
    double _per; // Private Instance Variable for its own library  
  
    // Instance variable with Custom Setter  
    void set percentage(double marksSecured) => _per = (marksSecured / 500) * 100;  
    // Instance variable with Custom Getter  
    double get percentage => _per;  
}
```

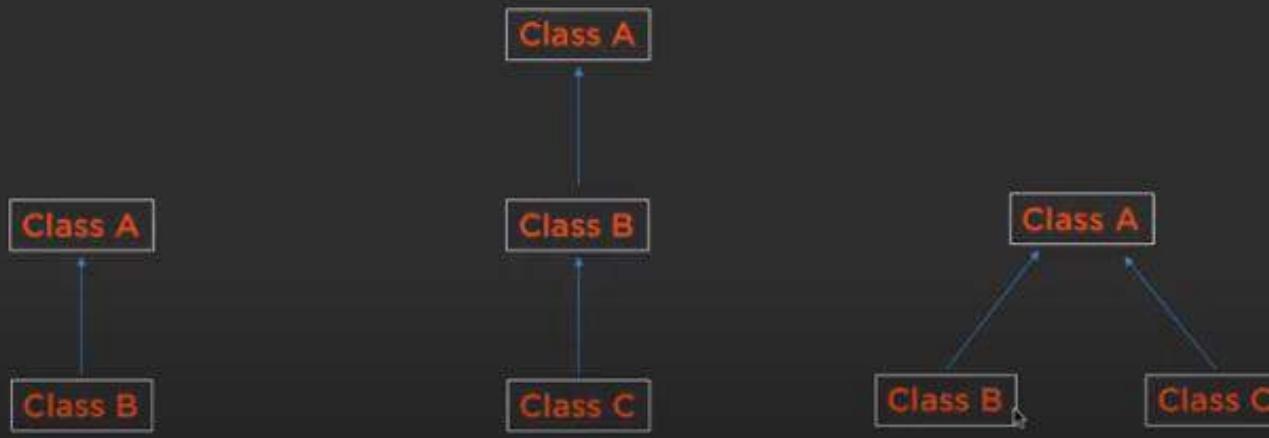
▶ RUN

Console

NAME:: Leena
PERCENTAGE:: 83.6

Documentation

Inheritance



1. Single Inheritance

2. Multi-level Inheritance

3. Hierarchial Inheritance

- Inheritance is a mechanism in which one object acquires properties of its parent class object.
- Super class of any class is Object :
 - Provides default implementation of:
 - `toString()`, returns the String representation of the object
 - `hashCode` Getter, returns the Hash Code of an object
 - `operator ==` , to compare two objects
- Advantages:
 - Code reusability
 - Method Overriding

Method Overriding in Inheritance

```
void main() {
    var mob = Mobile();
    mob.setMode();
    print(mob.color);
}
class Phone {
    String color = "White";
    void setMode() {
        print("Phone is in Ringing mode !");
    }
}
class Mobile extends Phone{
    String brand;
    String color = "Black";      // Property Overriding
    void changeTune() {
        print("Musical !");
    }
    void setMode() {      // Method Overriding
        print("Mobile can be in Silent or Ringing Mode!");
        super.setMode();
        print("So many more features to explore in mobile");
    }
}
```

▶ RUN

Console

```
Mobile can be in Silent or Ringing Mode!
Phone is in Ringing mode !
So many more features to explore in mobile
Black
```

Constructors in Inheritance

- By default, a constructor in a subclass calls the superclass's no-argument constructor.
- Parent class constructor is always called before child class constructor
- If default constructor is missing in Parent class, then you must manually call one of the constructors in super class

```
void main() {  
  
    var mob = Mobile();    mob.setMode();  
    print("Mobile Phone Color ${mob.color} OS type not mentioned ");  
  
    var mob1=Mobile.myPhnTypeCons("IOS", "Red");  
    mob1.changeTone();mob1.setMode();  
    print("MobileFeatures:: ${mob1.color}, ${mob1.osType}");  
}  
class Phone {  
    String color="White";  
    Phone.myPhoneCons(this.color);  
  
    void setMode() {  
        print("Phone-Ringing mode & in $color color only!");  
    }  
    Phone(){print("----PhoneDemo----");}  
}  
class Mobile extends Phone{  
    String osType;  
    String color;    // Property Overriding  
    Mobile(){this.color=super.color;}  
    Mobile.myPhnTypeCons(this.osType, this.color):super.myPhoneCons(color)  
    {print("----Mobile Phone Demo----");}  
    // Child class own method  
    void changeTone() => print("set musical Ringtone !");  
    void setMode() {    // Method Overriding  
        super.setMode();  
        print("Mobile -Silent/Vibration/Ringing Mode!");  
    }  
}
```

▶ RUN

Console

```
---PhoneDemo----  
Phone-Ringing mode & in White color only!  
Mobile -Silent/Vibration/Ringing Mode!  
Mobile Phone Color White OS type not mentioned  
---Mobile Phone Demo----  
set musical Ringtone !  
Phone-Ringing mode & in Red color only!  
Mobile -Silent/Vibration/Ringing Mode!  
MobileFeatures:: Red, IOS
```

Documentation:

Abstract Methods and Class

→ Abstract Method

- To make a method abstract, use **semicolon (;) instead of method body**
- Abstract method can only exist with Abstract class
- You need to override Abstract methods in sub-class

→ Abstract Class

- Use **abstract keyword to declare Abstract Class**
- Abstract class can have **Abstract Methods, Normal Methods and Instance Variables** as well.
- The Abstract Class **cannot** be instantiated, **you cannot create objects**

Abstract Methods and Class

```
void main() {  
    //var ui=PhoneUI(); //error Abstract class  
    var sp = SmartPhone();  
    sp.setLayout(); sp.appCounter();  
    var np = NormalPhone();  
    np.setLayout(); np.appCounter();  
}  
  
abstract class PhoneUI {  
    // Define your Instance variable if needed  
    String layout; int appCount = 4;  
    void setLayout(); // Abstract Method  
    void appCounter()=>print("Default App per page:: $appCount");  
}  
  
class SmartPhone extends PhoneUI {  
    var spc = 6;  
    void setLayout() => print("SmartPhone Custom Layout ");  
    void appCounter()=>print("${appCount + spc} apps icon/page");  
}  
  
class NormalPhone extends PhoneUI {  
    var npc = 2;  
    void setLayout() => print("NormalPhone Layout");  
    void appCounter()  
    {  
        super.appCounter();  
        print("${appCount + npc} apps icon/page");  
    }  
}
```

▶ RUN

Console

```
SmartPhone Custom Layout  
10 apps icon/page  
NormalPhone Layout  
Default App per page:: 4  
6 apps icon/page
```

Documentation

Interface

- Dart **does not have any special syntax to declare INTERFACE**
- An INTERFACE in dart is a Normal Class
- An INTERFACE is used when you need concrete implementation of all of its functions within it's sub class
 - It is mandatory to override all methods in the implementing class
- You can **implement** multiple classes but
 - You **cannot** **extend** multiple classes during Inheritance

Interface Example

```
void main() {
    var tv = Television();
    tv.volumeUp(); tv.volumeDown(); tv.command();
    var rem=Remote(); rem.volumeDown(); rem.volumeUp();
}
class Remote {
    void volumeUp()=>print("_____Volume Up from Remote_____");
    void volumeDown() =>print("_____Volume Down from Remote_____");
}
class Alexa {
    void command()=>print("Use A key to command alexa");
}
// Here Remote and AnotherClass acts as Interface
class Television implements Remote, Alexa {

    void volumeUp() {
        //super.volumeUp();
        print("_____Volume Up in Television_____");
    }
    void volumeDown()=> print("_____Volume Down in Television_____");
    void command()=>print("***Alexa can perform all Operation***");
}
```

▶ RUN

Console

```
_____Volume Up in Television_____
_____Volume Down in Television_____
***Alexa can perform all Operation***
_____Volume Down from Remote_____
_____Volume Up from Remote_____
```

Static keyword in dart

- Static Variables are also known as Class Variables
- Static Methods are also known as Class Methods
- Static variables are **lazily** initialized
 - i.e. they are not initialized until they are used in program
 - So they consume memory only when they are used
- Static methods has nothing to do with class object or instance
 - You cannot use '**this**' keyword within a Static Method
- From a Static Method
 - You can **ONLY** access Static Method and Static Variables
 - But you **cannot** access Normal Instance Variables and methods of the class

Static Example

```
void main() {  
    var s1 = Student("Leena", 111);  
    var s2 = Student("Anjali", 222);  
    var s3 = Student("New Student", 001);  
    s1.stuData();s2.stuData();  
    // print(s3.cname); //gives error static cannot be accessed through instance  
    Student.changeCname(); //static method called using classname//s1.changeCname invalid  
    s3.stuData();  
    s1.stuData();  
}  
class Student {  
    String name;  
    int rno;  
    static String cname = "KJSCE"; //memory allotted once shared by all instances  
    Student(this.name, this.rno);  
    void stuData() {  
        print("Student Name $name and roll no is $rno");  
        print("College Name:: $cname");  
        //changeCname(); //can call static method from normal function  
    }  
    static void changeCname() {  
        cname = "K.J.S.C.E";  
        // stuData(); instance members cannot be accessed from static method  
    }  
}
```

▶ RUN

Console

```
Student Name Leena and roll no is 111  
College Name:: KJSCE  
Student Name Anjali and roll no is 222  
College Name:: KJSCE  
Student Name New Student and roll no is 1  
College Name:: K.J.S.C.E  
Student Name Leena and roll no is 111  
College Name:: K.J.S.C.E
```

Documentation

Lecture 6 –Module 1

Lambda ,Higher order Functions, Closures
Lists, Collections

Lambda Expression

- * A function without a name
- * Also known as Anonymous Function or Lambda
- * Functions in darts are objects

SYNTAX:

```
(parameterlist)
{
    Statements;
}
```

The diagram illustrates the components of a Dart lambda expression. It shows two examples of lambda functions with annotations:

- Variable Name:** Points to the identifier `addNumbers`.
- Parameters:** Points to the parameters `a` and `b` in the parameter list.
- Method Body:** Points to the code block following the colon, which includes the assignment `var sum = a + b;` and the `return sum;` statement.

```
Function addNumbers = ( int a, int b ) {  
  var sum = a + b;  
  return sum;  
}  
  
Function addNumbers = ( int a, int b ) => a + b;  
  
// Calling a lambda function  
addNumbers( 2, 5 ); // Returns 7  
addNumbers( 3, 9 ); // Returns 12
```

Lambda Expression

```
void main() {
    print("----FIRST WAY----");
    showOp("Addition",11,22);
    print("\n----USING FAT ARROW-SINGLE LINE STATEMENT----");
    result("Subtraction",11,22);
    print("\n----WITH RETURN VALUE----");
    print("Multiplication is:: ${result2(5,7)}");
    print("\n----NO PARAMETERS----");
    test();
}

Function showOp= (String op, int n1, int n2){
    print("Num1= $n1 and Num2= $n2");
    print("Operation Performed $op :: ${n1 + n2}");
};

Function result =
    (String op, int n1, int n2) => print("$op performed ${n1 - n2}");

var result2 = (int n1, int n2) => n1 * n2;
var test=()=>print("All types done ");

```

▶ RUN

Console

```
----FIRST WAY----
Num1= 11 and Num2= 22
Operation Performed Addition :: 33

----USING FAT ARROW-SINGLE LINE STATEMENT----
Subtraction performed -11

----WITH RETURN VALUE----
Multiplication is:: 35

----NO PARAMETERS----
All types done
```

Documentation

Lambda Example

```
import 'dart:io';

void main() {
  var obj1 = Operation();  obj1.showOp(obj1.op, obj1.n1, obj1.n2);
  var obj2 = Operation();
  obj2.result(obj2.op, obj2.n1, obj2.n2);
  var mulOp = Operation();
  print("Result is= ${mulOp.result2(mulOp.n1, mulOp.n2)}");
}

class Operation {
  String op;
  var n1, n2;
  Operation() {
    print("Enter Operation to perform:");
    op = stdin.readLineSync();
    print("Enter Num1 and Num2");
    n1 = int.parse(stdin.readLineSync());
    n2 = int.parse(stdin.readLineSync());
  }
  //Lambda with multiple statement using "Function"
  Function showOp= (String op, int n1, int n2){
    print("Num1= $n1 and Num2= $n2");
    print("Operation Performed $op :: ${n1 + n2}");
  };
  //Lambda Expression using "Function or var" both will work
  Function result =
    (String op, int n1, int n2) => print("$op performed ${n1 - n2}");
  var result2 = (int n1, int n2) => n1 * n2;
}
```

```
Enter Operation to perform:  
add  
Enter Num1 and Num2  
11  
22  
Num1= 11 and Num2= 22  
Operation Performed add :: 33  
Enter Operation to perform:  
sub  
Enter Num1 and Num2  
33  
22  
sub performed 11  
Enter Operation to perform:  
mult  
Enter Num1 and Num2  
11  
22  
Result is= 242
```

result no return type and
result2 returning int

Higher Order Functions

- * Passing function as parameter in a function
- * Return a function from another function
- * Returns function and parameter passed function

```
void main() {  
    Function findlargest = (a, b) => print("${(a > b) ? a : b}");  
    checkingNo("Largest Number", findlargest);  
  
    var myFunc = taskToPerform(); //returns multiplyFour here  
    print("--calling myfunc--");  
    print(myFunc(10)); //multiplyFour(10) // 10*4 //40  
}  
  
void checkingNo(String message, Function myFunction) {  
    print(message);  
    myFunction(12, 45); //findlargest(12,45)// 45  
}  
  
Function taskToPerform() {  
    print("inside task to perform ");  
    Function multiplyFour = (int number) => number * 4;  
    return multiplyFour;  
}
```

▶ RUN

Console

```
Largest Number  
45  
inside task to perform  
--calling myfunc--  
40
```

Documentation

Lexical Scoping and Closure

- * **Lexical scoping** means, an inner function has access to variables defined in the parent scopes. In lexical scoping, variables are passed down from parent to child.
- * A **closure** is a bundle of a function and variables from the outer scope that the function depends on. For example, if a function depends on the variable from the scope of the parent function which has already returned, a closure holds the variables of the returned function.
- * **Using lexical scoping, main function has access to the global scope. This means anything declared outside main function can be accessed within main function.**

Lexical scoping

```
void main() {  
    print('--Lexical Scoping-main function  
has access to the global scope--\n');  
    var head = topic.substring(0, 9);  
    var getUppercase = () {  
        var mid = topic.substring(9, 13);  
        print("$mid");  
        print(head.toUpperCase());  
    };  
    getUppercase(); //inside main  
    print(getUppercase1()); //outside main  
}  
  
String topic = "learning dart programming";  
Function getUppercase1 = () {  
    var text = topic.substring(14, 25);  
    return ("${text.toUpperCase()}");  
};
```

▶ RUN

Console

```
--Lexical Scoping-main function  
has access to the global scope--
```

```
dart  
LEARNING  
PROGRAMMING
```

Documentation

Closure

- * A closure is a function that has access to the parent scope, even after the scope has closed.
- * A closure is a function object that has access to variables in its lexical scope , even when the function is used outside of its original scope.

The screenshot shows a code editor interface with a dark theme. On the left, there is a code editor window containing Java code. In the center, there is a blue button labeled "RUN". To the right, there is a "Console" window displaying the output of the code execution.

```
void main() {
    var addNo = (int i) {
        print("Closure");
        return (int j) => i++ + j; //returns a func which adds j with
        i
        //return function creates closure with i ,i lives even when
        addNo is returned
    };
    print("before");
    var add2 = addNo(4); //addNo function is returned
    print("After");
    print("add2(6):: ${add2(6)}");
    print("add2(6):: ${add2(6)}");
    print("add2(6):: ${add2(6)}");
}
```

Console

```
before
Closure
After
add2(6):: 10
add2(6):: 11
add2(6):: 12
```

Example

```
void main() {  
    String fname = "Leena";  
    String lname = "SAHU";  
    String mname = "P";  
    print("-----Case1-CLOSURE-----");  
    Function showName = () {  
        fname = "Aleena";  
        String lname = "sahu";  
        print("Name inside showName $fname $mname $lname");  
    };  
    showName();  
    print("Name outside-In main $fname $mname $lname");  
  
    print("\n-----Case2-LEXICAL SCOPE Function outside main-----");  
  
    Function nameChange = myname();  
    nameChange();  
}  
  
Function myname = () {  
    String name = "Leena Sahu";  
    Function nameUpper = () {  
        name = name.toUpperCase();  
        print("Name Changed in Uppercase:$name");  
    };  
    print("Name in Default case:$name");  
    return nameUpper;  
};
```

▶ RUN

Console

-----Case1-CLOSURE-----

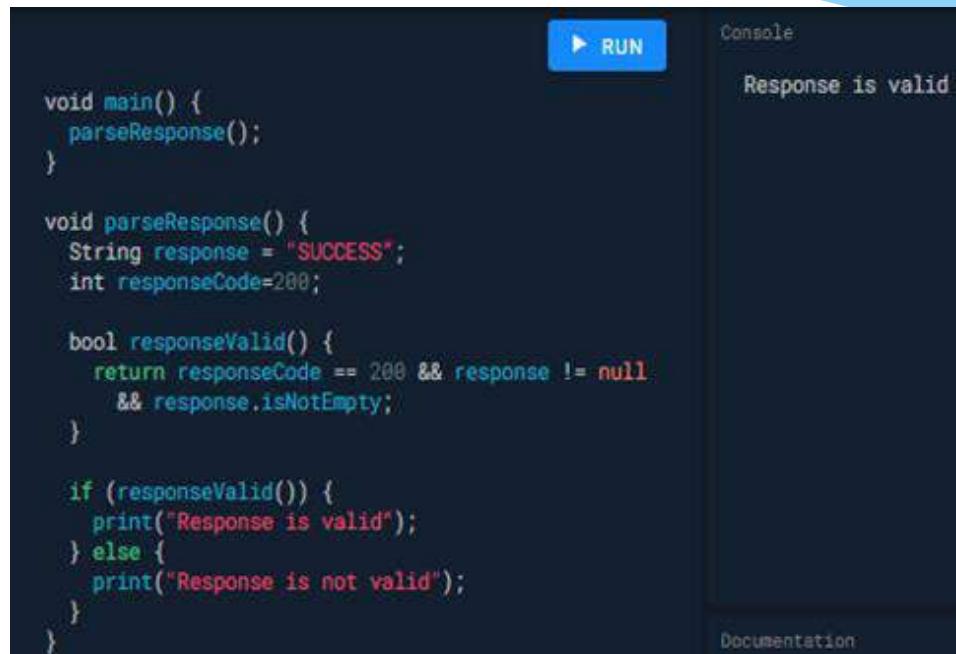
Name inside showName Aleena P sahu
Name outside-In main Aleena P SAHU

-----Case2-LEXICAL SCOPE Function outside main-----

Name in Default case::Leena Sahu
Name Changed in Uppercase::LEENA SAHU

Documentation

More example on lexical scope and Lexical Closures



A screenshot of a code editor showing Dart code. The code defines a `main` function that calls `parseResponse`. Inside `parseResponse`, it prints "Response is valid". The code then checks if `responseValid` returns true, printing "Response is valid" or "Response is not valid". A `RUN` button is visible at the top, and the console output shows "Response is valid".

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

void parseResponse() {
  String response = 'SUCCESS';
  int responseCode=200;

  bool responseValid() {
    return responseCode == 200 && response != null
      && response.isNotEmpty;
  }

  if (responseValid()) {
    print("Response is valid");
  } else {
    print("Response is not valid");
  }
}
```

```
ListView.builder(
  itemBuilder: (context, index) {
    return GestureDetector(
      onTap: () => print("Tapped: $index"),
      child: Text("Index: $index"),
    );
  },
  itemCount: 10,
);
```

This is standard example of `ListView.builder` in flutter. Here, `onTap` function can access to `index` variable so we can know on which item user has tapped. `onTap` might be called after 20 seconds or 2 minutes, however we will get a value of index. That's the lexical closure.

Here, `response` and `responseCode` are accessible to `responseValid` method.

Dart Collections

- **List**
 - **Ordered Collection**
 - Fixed-length list
 - Growable list
- **Set**
 - **Unordered Collection**
 - Implementation: **HashSet**
- **Map**
 - **Unordered Collection**
 - Implementation: **HashMap**

List

```
void main()
{
    print(''---Creating List and
accessing using for each loop ---'');
var subjects=[ "WP2", "AI", "AOA", "ML",
              "NLP", "Data Science"];
print(subjects);
subjects.forEach((i) {
    print('${subjects.indexOf(i)}: $i');
});

List<int> numbersList = List(5); // Fixed-length list
print("\nFIXED LENGTH MARKSLISTS");
numbersList[0] = 73;
numbersList[1] = 64;
numbersList[2] = 64;
numbersList[3] = 21;
numbersList[4] = 12;
for (int ele in numbersList) {
    print(ele);
}

print("----Growable List----");
List<double> val = List();
val.add(12); val.add(24);
val.add(67);
val.add(87); val.add(12);
//val.remove(12);
val.removeAt(1);
val.add(98);
val.forEach((e) => print(e));
}
```

▶ RUN

Console

```
---Creating List and
accessing using for each loop ---
[WP2, AI, AOA, ML, NLP, Data Science]
0: WP2
1: AI
2: AOA
3: ML
4: NLP
5: Data Science
```

FIXED LENGTH MARKSLISTS

```
73
64
64
21
12
```

-----Growable List-----

```
12
67
87
12
98
```

Documentation

Set

--> Unordered Collection
--> All elements are unique

```
void main() {  
  
    Set<String> countries = Set.from(["USA", "INDIA", "CHINA"]); // From a list  
    countries.add("Nepal");  
    countries.add("Japan");  
    print("COUNTRIES::\n \$countries");  
  
    Set<int> numbersSet = Set(); //Using Constructor  
    numbersSet.add(73); numbersSet.add(64);  
    numbersSet.add(21); numbersSet.add(12);  
    numbersSet.add(73); // Duplicate entries are ignored  
    numbersSet.add(73); // Ignored  
    numbersSet.add(77); numbersSet.add(12);  
    print("\n");  
    for (int element in numbersSet) {  
        print(element);  
    }  
  
    print("Search for 75 \$\{numbersSet.contains(75)\}");  
    print("Search for 73 \$\{numbersSet.contains(73)\}");  
    print("Remove 64 \$\{numbersSet.remove(64)\}");  
    print("IsEmpty \$\{numbersSet.isEmpty\}");  
    print("Length of set is \$\{numbersSet.length\}");  
    // numbersSet.clear(); // Deletes all elements  
    print("\n");  
    numbersSet.forEach((element) => print(element)); // Using Lambda  
}
```

▶ RUN

COUNTRIES::
{USA, INDIA, CHINA, Nepal, Japan}

73
64
21
12
77
Search for 75 false
Search for 73 true
Remove 64 true
IsEmpty false
Length of set is 4

73
21
12
77

Map

--> KEY is unique
--> VALUE can be duplicated

```
void main() {  
  
    Map<String, int> countryDialingCode = { "USA": 1, "INDIA": 91, "Japan": 23};  
    countryDialingCode.forEach((key,value) => print("key: $key and value: $value"));  
  
    Map<String, String> fruits = Map(); // Using Constructor  
    fruits["apple"] = "red"; fruits["banana"] = "yellow";  
    fruits["guava"] = "green"; fruits["pineapple"] = "mustord";  
    fruits["papaya"] = "yellow";  
    print("\n ----Operations on Map----");  
    print("Searching for Fruits:: ${fruits.containsKey("apple")});  
    print("Adding Fruits ::${fruits.update("apple", (value) => "green")});  
    fruits.remove("apple");  
    print("Fruit have items ${fruits.isEmpty}");  
    print("No of fruits in list is ${fruits.length}");  
//    fruits.clear();  
    print(fruits["apple"]);print(fruits["banana"]);  
    print("\n ---- Keys----");  
    for (String key in fruits.keys) {  
        print(key);  
    }  
    print("\n ----Values----");  
  
    for (String value in fruits.values) {  
        print(value);  
    }  
    print("\n---- fruit Keys Values----");  
    fruits.forEach((key, value) => print("key: $key and value: $value"));  
}
```

▶ RUN

Console

```
key: USA and value: 1  
key: INDIA and value: 91  
key: Japan and value: 23
```

```
----Operations on Map----  
Searching for Fruits:: true  
Adding Fruits ::green  
Fruit have items false  
No of fruits in list is 4  
null  
yellow
```

```
---- Keys----  
banana  
guava  
pineapple  
papaya
```

```
----Values----  
yellow  
green  
mustord  
yellow
```

```
---- fruit Keys Values----  
key: banana and value: yellow  
key: guava and value: green  
key: pineapple and value: mustord  
key: papaya and value: yellow
```

Documentation

typedef keyword

- * Functions are first-class objects.
- * We can define functions same as any-other type like int or string.

The screenshot shows a code editor interface with a dark theme. On the left, there is a code editor window containing Dart code. On the right, there is a 'Console' window showing the output of the code execution.

```
typedef FilterFunction = bool Function(dynamic a);

void main() {
    print("Printing evens:");
    filter((i) => i % 2 == 0);
    print("\nPrinting odds:");
    filter((i) => i % 2 != 0);
}

void filter(FilterFunction filterNumbers) {
    List elements = [1, 2, 3, 4, 5, 6];
    List evenElements = elements.where(filterNumbers).toList();
    for (int e in evenElements) {
        print(e);
    }
}
```

▶ RUN

Console

Printing evens:
2
4
6

Printing odds:
1
3
5

- * Here, we created **FilterFunction** using **typedef**, which takes one integer and returns a boolean. So, whenever we use FilterFunction, we know that what does it mean! It makes code more readable

Task 3

* Will Add soon

Introduction to Mobile Programming development

Leena Sahu
IT Dept. KJSCE

Syllabus

- * [flutter syll.pdf](#)

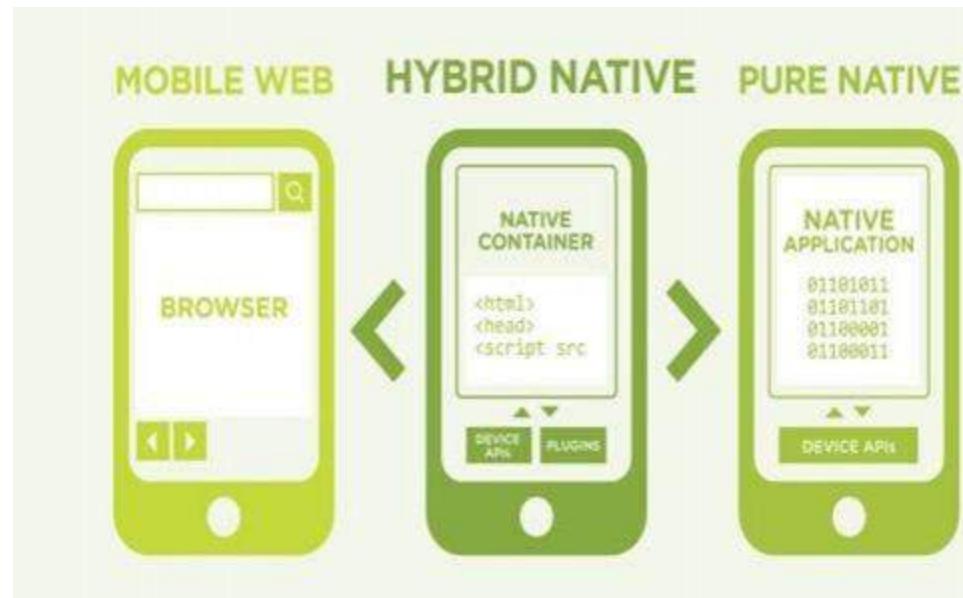
Types of Mobile App



HYBRID MOBILE APPS combine elements of native and web-based apps, but wrapped within a native app, giving it the ability to have its own icon or be downloaded from an app store.

WEB-BASED APPS are responsive versions of websites designed in order to work on any mobile device

NATIVE APPS include Android, Windows Phone, and iOS



Types of Mobile Apps by Technology

- * There are three basic types of mobile apps if we categorize them by the technology used to code them:
 - * **Native apps** are created for one specific platform or operating system.
 - * **Web apps** are responsive versions of websites that can work on any mobile device or OS because they're delivered using a mobile browser.
 - * **Hybrid apps** are combinations of both native and web apps, but wrapped within a native app, giving it the ability to have its own icon or be downloaded from an app store.

Characteristics of a Purely-Native Mobile App

- A **binary “executable image”**, that is **explicitly downloaded and stored on the file system** of the mobile device
- Distributed through the popular **app store** or marketplace of the device, or via an enterprise distribution mechanism
- **Executed directly** by the operating system
 - Launched from the home screen
 - Does not require another “container app” to run it

Intro

Native Apps

```
101101101011011  
110110110110110  
110110110110101  
101101011011011  
011011011011011  
011101011101111  
110110110110101  
101000001101011
```

Web Apps

```
<!DOCTYPE html PUBLIC  
<html>  
<!-- created 2003-12-12 -->  
<head><title>XYZ</title>  
</head>  
<body>  
<p>  
Voluptatem accusantium do  
Totam rem aperiam eaque  
</p>  
</body>  
</html>
```

Hybrid Apps

```
101101  
101011  
011110  
110110  
110110  
110110  
101101
```

```
<!DOCTYPE html  
PUBLIC  
<html>  
<!-- created  
2003-12-12 -->  
<head><title>XY  
Z</title>  
</head>  
<body>  
<p>  
Voluptatem  
</p>  
</body>  
</html>
```



Examples of Purely –Native Mobile Apps



Native Apps

TECHNOLOGY USED

Java, Kotlin, Python, Swift, Objective C, etc.



PROS

- 1** Faster, better performance
- 2** Native UI
- 3** Can access device features



CONS

- 1** Higher cost to maintain
- 2** Takes up space in the device
- 3** Updates must be downloaded

Web Apps

TECHNOLOGY USED

HTML5, CSS, JavaScript, Ruby, etc.



PROS

- 1** Web-based so performs on all devices
- 2** Easier to maintain
- 3** Users don't run out of storage



CONS

- 1** Dependent on a browser
- 2** Needs an internet connection
- 3** May not always integrate with device hardware

Hybrid Apps

TECHNOLOGY USED
Ionic, Objective C, Swift, HTML5, etc.



PROS

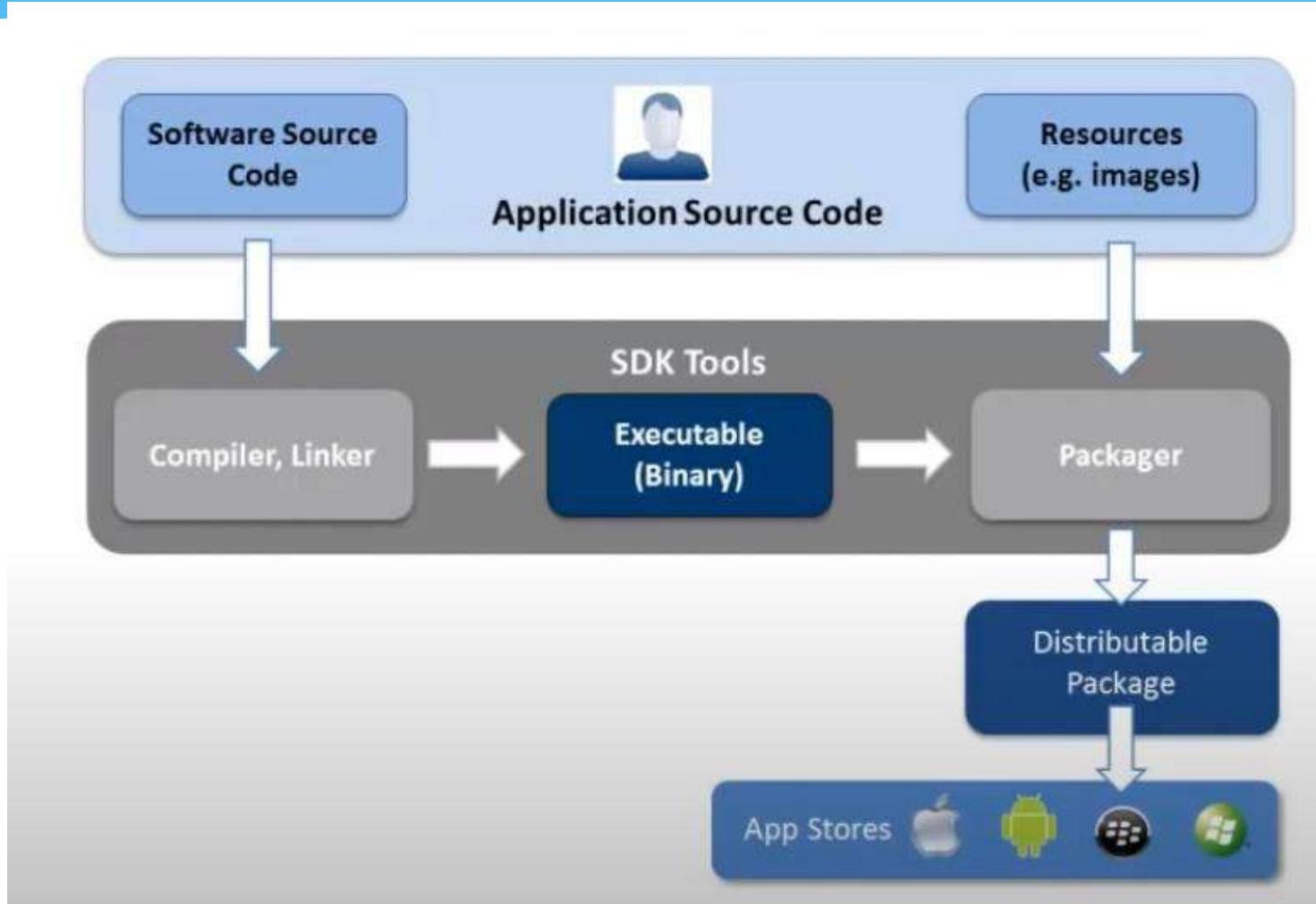
- 1** Quicker/cheaper to build
- 2** Load quickly
- 3** Less code to maintain

CONS

- 1** Lacks power of native apps
- 2** Slower since it has to download each element
- 3** Certain features might not be usable on devices

- * **Decision Factor: I Need an App ASAP!**
 - * If you absolutely must have an app in the shortest amount of time possible, then you need to invest in building a **web app**. Not only will one codebase drastically speed up development time, but it will also mean that your users already have what they need to use it: a mobile browser.
- * **Decision Factor: I Have Limited Resources**
 - * If time and money are not on your side, then consider either a **web app or a hybrid app**. The hybrid app gives you a chance to test the market with a minimum viable product that can be in the hands of users within a few months. And if successful, you can decide to build a full-fledged native version later on.
- * **Decision Factor: My App Must Be Fast and Stable**
 - * If performance is of the utmost importance, then there's no way around it: you need to develop a **native app**. This type of app will give you the speed, stability, and customization features you deem crucial to your success.
- * **In the end, choosing which type of mobile app you will build is not a one-and-done decision. You can always choose to build another type down the road, depending on your user's needs.**

Native App Development

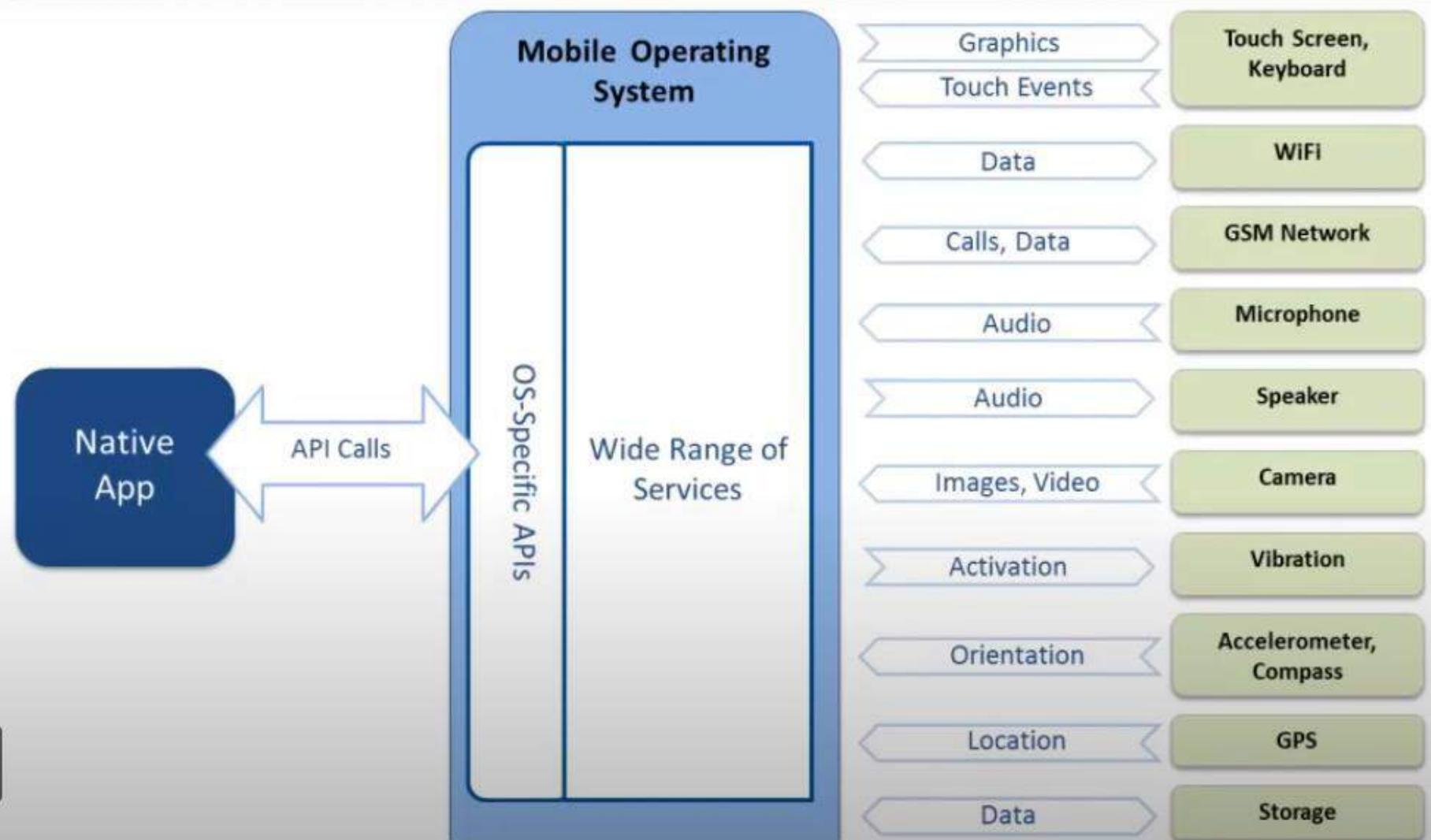


Native App Development-Summary

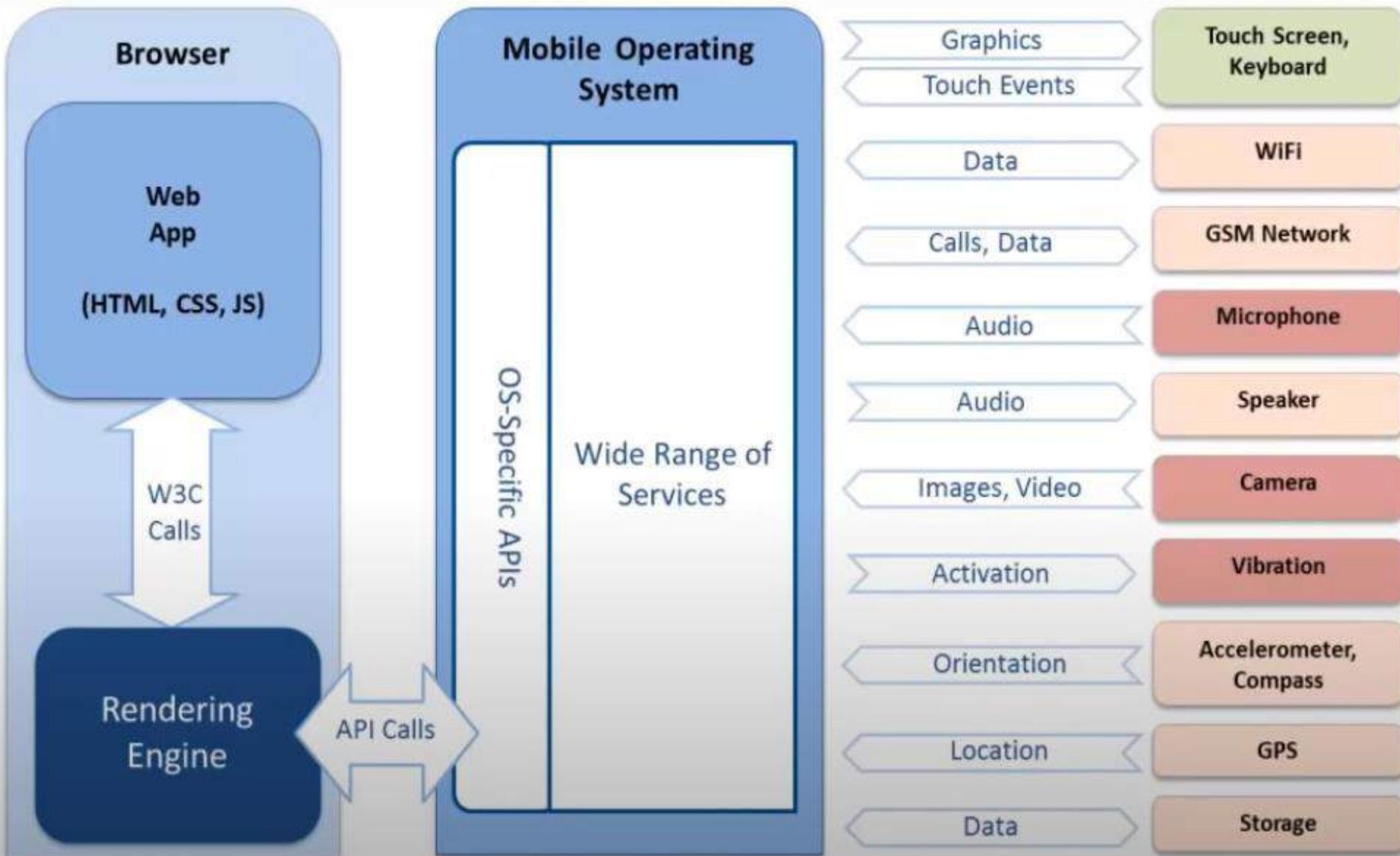
				
Languages	Obj-C, C, C++	Java (Some C, C++)	Java	C#, VB.NET, etc
Tools	Xcode	Android SDK	BB Java Eclipse Plug-In	Visual Studio, Windows Phone Dev Tools
Executable Files	.app	.apk	.cod	.xap
Application Stores	Apple iTunes	Android Market	BlackBerry App World	Windows Phone Market

- Similar approach, but different source code and expertise results in expensive development and maintenance

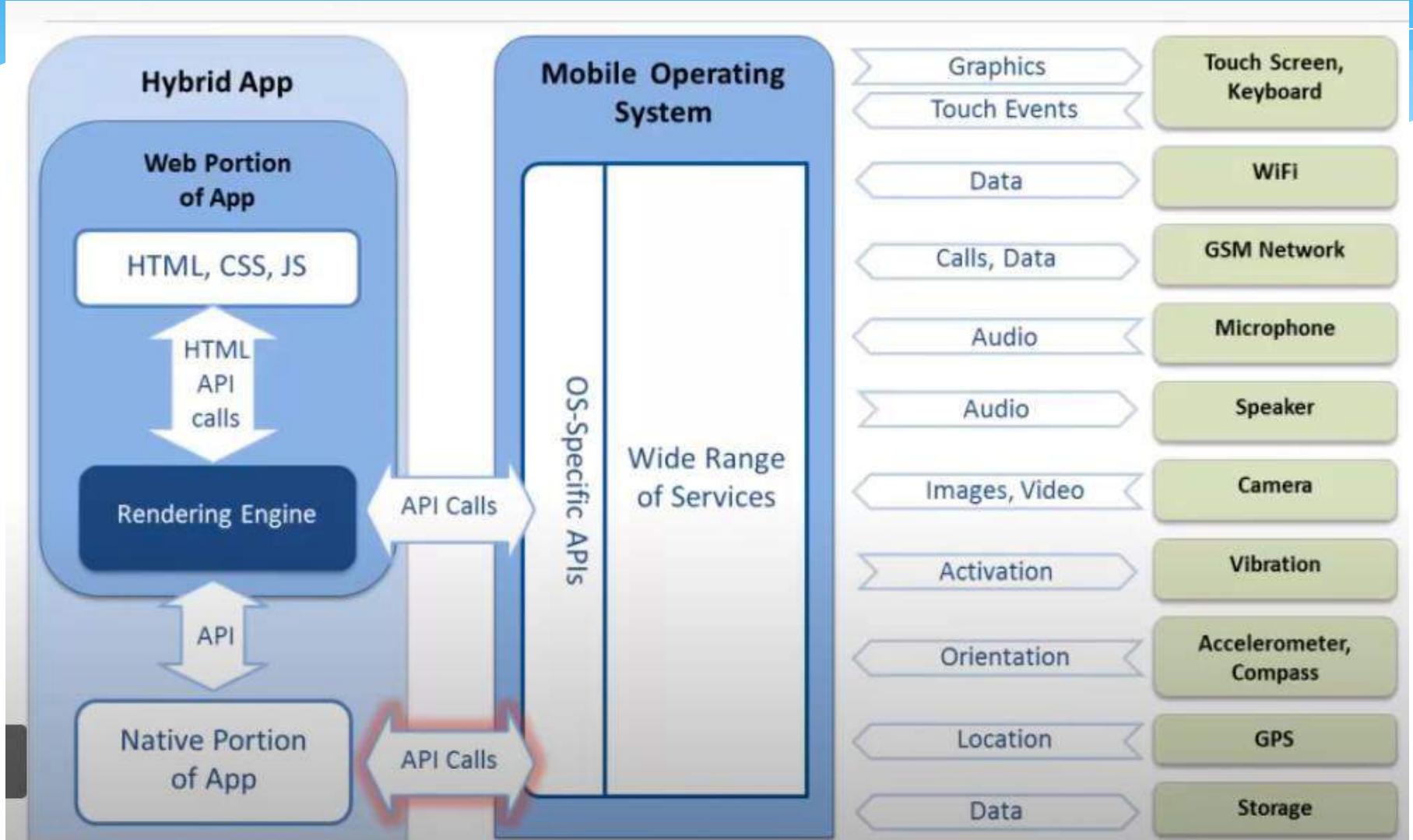
Native Apps- Interaction with Mobile Devices



Web App – Interaction with Mobile Device



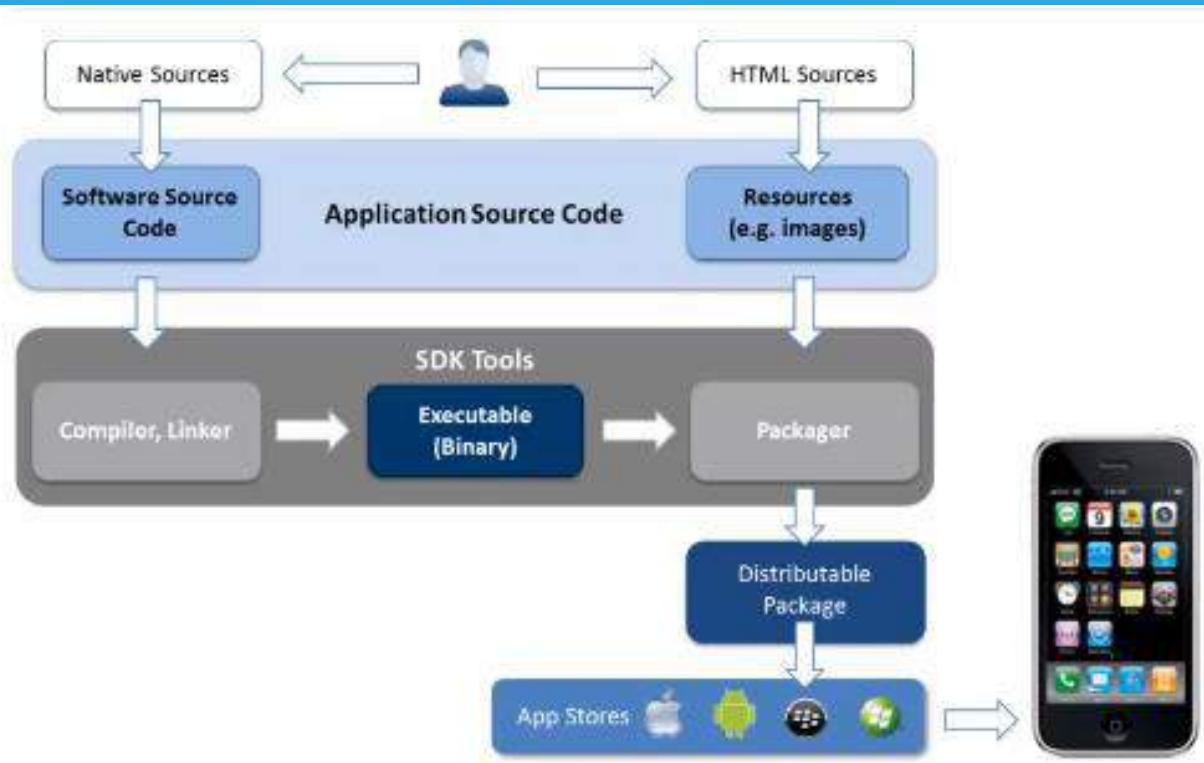
Hybrid App –Interaction with Mobile Device



App Development Comparisons

	Device Access	Speed	Development Cost	App Store	Approval Process
Native	Full	Very Fast	Expensive	Available	Mandatory
Hybrid	Full	Native Speed as Necessary	Reasonable	Available	Low Overhead
Web	Partial	Fast	Reasonable	Not Available	None

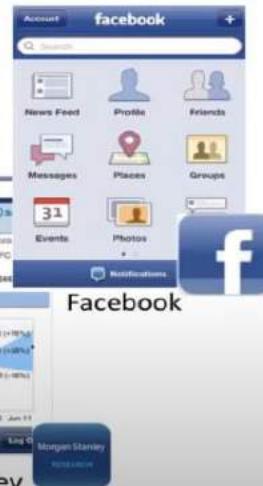
Some Examples of Hybrid Apps



Bank of America



Morgan Stanley



Why Flutter Uses Dart

- * Flutter enable us to ship an application for Android/IOS and web from a Single codebase.
- * Dart provides-
 - * Multi Platform
 - * Fast Development
 - * Enables High Fidelity Flutter app for all platform.
 - * Dart production quality compiler compiles to ARM & X64 machine code for mobile or optimized JS for the web, enabling quick app startup times and smooth animations.
- * Flutter feature
 - * Hot reload injects updated dart source into running app and rebuilds UI in few second, changes reflected instantly.
- * Dart is easy to learn if you know Java, JavaScript Swift
- * Dart + Flutter → Amazing experiences across all platforms

References

- * <https://www.youtube.com/watch?v=Ns-JS4amlTc>
(Images referred)
- * <https://clevertap.com/blog/types-of-mobile-apps/>

Navigation and multiple screens

Navigator and routes

- In flutter each *screen* or *page* is just a *widget*, and in context of navigation, each screen or page is called a ***route***. It is equivalent to *Activity* in Android or *ViewController* in iOS.
- The ***Navigator*** class provided by Flutter is used for navigating between screens.

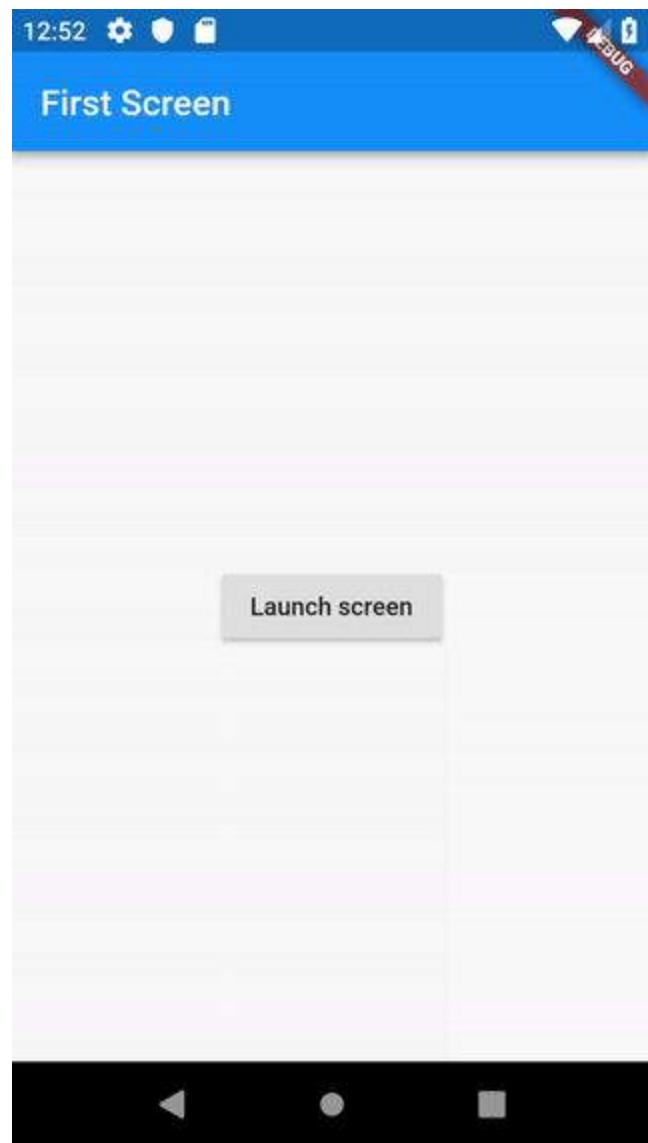
- In flutter, **screens** and **pages** are called *routes*.
- When we start the app, the **home** widget of the *MaterialApp* becomes the **root** of the app, the bottom route of the *Navigator stack*.
When another route is pushed, it is added to the top of the stack.

First Way

- **How to navigate between two routes**
 - Create two routes
 - Navigate to the second route using **Navigator.push()**
 - Return to the first route using **Navigator.pop()**

```
class FirstRoute extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('First Route'),  
      ),  
      body: Center(  
        child: RaisedButton(  
          child: Text('Open route'),  
          onPressed: () {  
            Navigator.push(  
              context,  
              MaterialPageRoute(builder: (context) => SecondRoute()),  
            );  
          }  
        )  
      )  
    );  
  }  
}
```

```
class SecondRoute extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Second Route"),  
      ),  
      body: Center(  
        child: RaisedButton(  
          onPressed: () {  
            Navigator.pop(context);  
          },  
          child: Text('Go back!'),  
        ),  
      ),  
    );  
  }  
}
```



Second Way –Named Routes

- If we need to navigate to the same screen in many parts of our apps, this can result in code duplication. In these cases, it can be handy to define a “named route,” and use the named route for Navigation.
- To work with named routes, we can use the **Navigator.pushNamed** function. This example will replicate the functionality from the original recipe, demonstrating how to use named routes instead.

Second Way

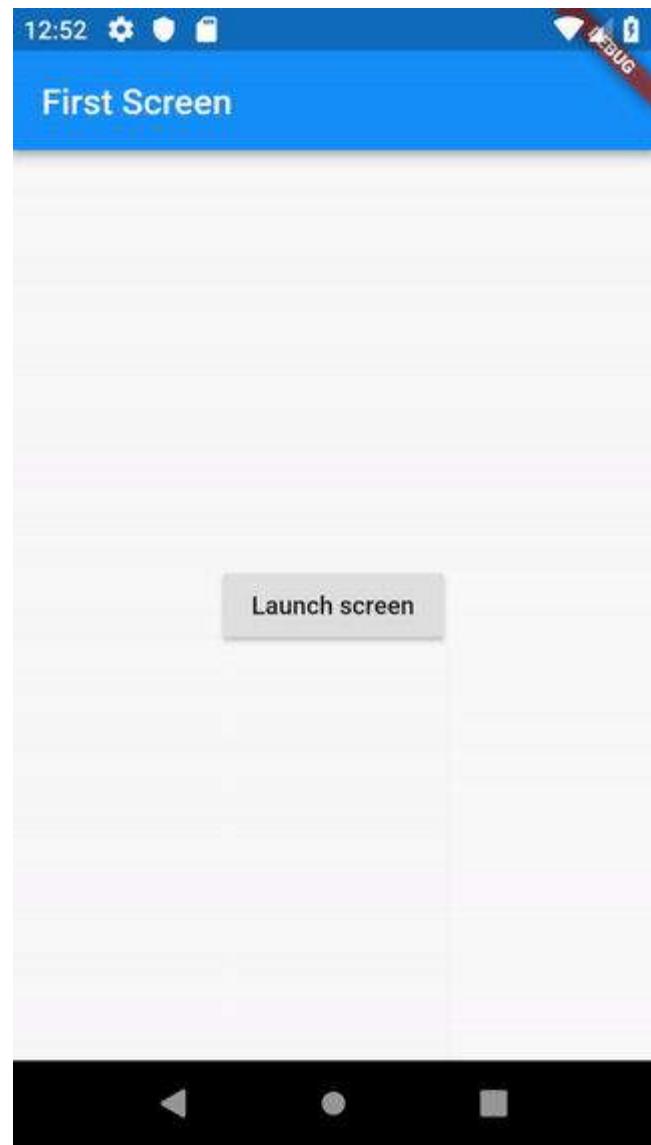
- Directions
 - Create two screens
 - Define the routes
 - Navigate to the second screen using `Navigator.pushNamed`
 - Return to the first screen using `Navigator.pop`

```
// Start the app with the "/" named route. In our case, the app will start

import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
    title: 'Named Routes Demo',
    // on the FirstScreen Widget
    initialRoute: '/',
    routes: {
      // When we navigate to the "/" route, build the FirstScreen Widget
      '/': (context) => FirstScreen(),
      // When we navigate to the "/second" route, build the SecondScreen Widget
      '/second': (context) => SecondScreen(),
    },
  )));
}
```

```
class FirstScreen extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('First Screen'),  
        body: Center(  
          child: RaisedButton(  
            child: Text('Launch screen'),  
            onPressed: () {  
              // Navigate to the second screen using a named route  
              Navigator.pushNamed(context, '/second');  
            },  
          ),  
        ),  
      );  
    );  
  },  
},
```

```
class SecondScreen extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Second Screen"),  
        body: Center(  
          child: RaisedButton(  
            onPressed: () {  
              // Navigate back to the first screen by  
              // popping the current route  
              // off the stack  
              Navigator.pop(context);  
            },  
            child: Text('Go back!'),  
          ),  
        ),  
      );  
    }  
  }  
},
```



Pass arguments to a named route

- The [Navigator](#) provides the ability to navigate to a named route from any part of an app using a common identifier. In some cases, you may also need to pass arguments to a named route.
- *For example, you may wish to navigate to the /user route and pass information about the user to that route.*
- In Flutter, you can accomplish this task by providing additional arguments to the [Navigator.pushNamed](#) method. You can extract the arguments using the [ModalRoute.of](#) method or inside an [onGenerateRoute](#) function provided to the [MaterialApp](#) or [CupertinoApp](#) constructor.
- This recipe demonstrates how to pass arguments to a named route and read the arguments using [ModelRoute.of](#) and [onGenerateRoute](#).

- Directions
 - Define the arguments you need to pass
 - Create a widget that extracts the arguments
 - Register the widget in the routes table
 - Navigate to the widget

1. Define the arguments you need to pass

```
class ScreenArguments
{
    final String title;
    final String message;
    ScreenArguments(this.title, this.message);
}
```

2. Create a widget that extracts the arguments

```
// A Widget that extracts the necessary arguments from the ModalRoute.
class ExtractArgumentsScreen extends StatelessWidget
{
    static const routeName = '/extractArguments';
    @override
    Widget build(BuildContext context)
    {
        // Extract the arguments from the current ModalRoute settings and
        // cast
        // them as ScreenArguments.
        final ScreenArguments args =
            ModalRoute.of(context).settings.arguments;
        return Scaffold(
            appBar: AppBar(
                title: Text(args.title),
            ),
            body: Center(
                child: Text(args.message),
            ),
        );
    }
}
```

3. Register the widget in the routes table

Add an entry to the `routes` provided to the `MaterialApp` Widget.

The `routes` define which widget should be created based on the name of the route.

```
MaterialApp(  
  routes: {  
    ExtractArgumentsScreen.routeName:  
      (context) => ExtractArgumentsScreen(),  
  },  
);
```

4. Navigate to the widget

Navigate to the `ExtractArgumentsScreen` when a user taps a button using [`Navigator.pushNamed`](#).

Provide the arguments to the route via the `arguments` property. The `ExtractArgumentsScreen` extracts the `title` and `message` from these arguments.

```
// A button that navigates to a named route that. The named route
// extracts the arguments by itself.
RaisedButton(
    child: Text("Navigate to screen that extracts arguments"),
    onPressed: () {
        // When the user taps the button, navigate to the specific rout
        // and provide the arguments as part of the RouteSettings.
        Navigator.pushNamed(
            context,
            ExtractArgumentsScreen.routeName,
            arguments: ScreenArguments(
                'Extract Arguments Screen',
                'This message is extracted in the build method.',
            ),
        );
    },
);
```

5. Alternatively, extract the arguments using `onGenerateRoute`

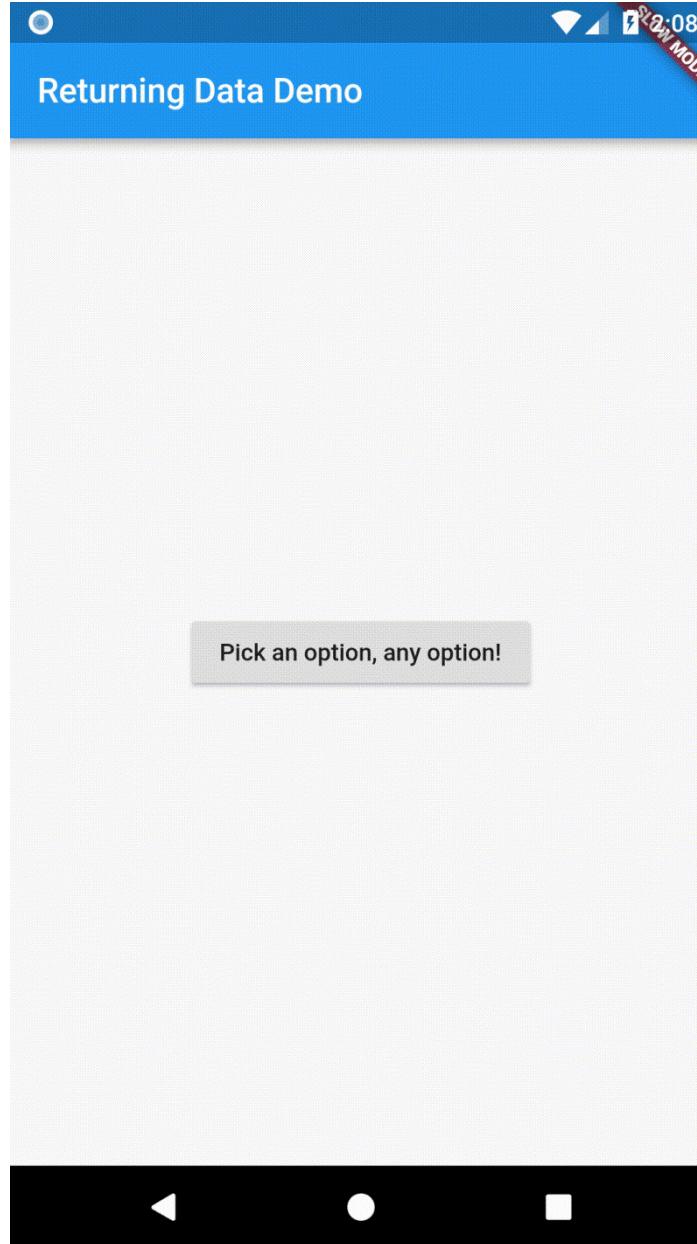
Instead of extracting the arguments directly inside the widget, you can also extract the arguments inside an [`onGenerateRoute`](#) function and pass them to a widget.

The `onGenerateRoute` function creates the correct route based on the given `RouteSettings`.

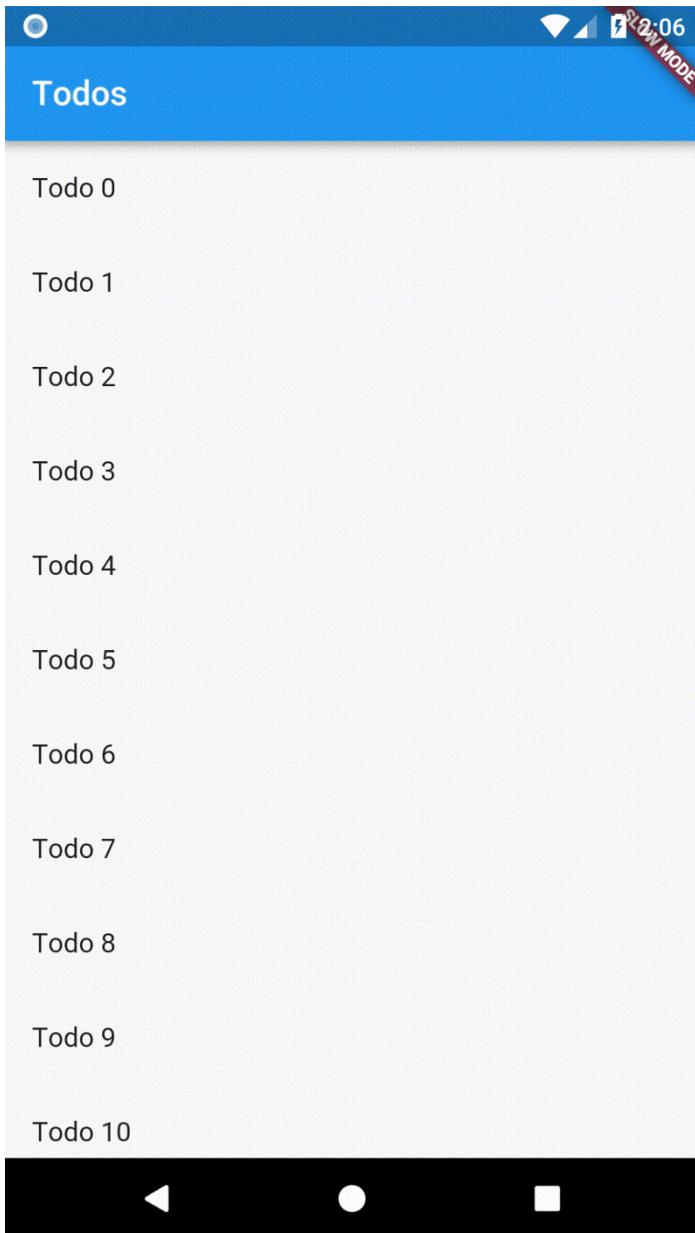
```
MaterialApp(  
  // Provide a function to handle named routes. Use this function to  
  // identify the named route being pushed and create the correct  
  // Screen.  
  onGenerateRoute: (settings) {  
    // If you push the PassArguments route  
    if (settings.name == PassArgumentsScreen.routeName) {  
      // Cast the arguments to the correct type: ScreenArguments.  
      final ScreenArguments args = settings.arguments;  
  
      // Then, extract the required data from the arguments and  
      // pass the data to the correct screen.  
      return MaterialPageRoute(  
        builder: (context) {  
          return PassArgumentsScreen(  
            title: args.title,  
            message: args.message,  
          );  
        },  
      );  
    }  
  },  
);
```



<https://www.bookstack.cn/read/flutter-cookbook/8cadc773e043d4f1.md>



<https://www.bookstack.cn/read/flutter-cookbook/20530fb3aa561b1a.md>



Handle Undefined Routes

- There's two ways to handle Undefined routes.
 - Returning your `UndefinedView` as the default route in `generateRoute`
 - Returning your `UndefinedView` from the `onUnknownRoute`
- <https://www.youtube.com/watch?v=YXDFlpdpp3g&feature=youtu.be>

References

- <https://medium.com/flutter-community/flutter-navigation-cheatsheet-a-guide-to-named-routing-dc642702b98c>
- <https://medium.com/fabcoding/navigating-between-screens-in-flutter-navigator-named-routes-passing-data-e3deab46c9e6>
- <https://www.filledstacks.com/snippet/clean-navigation-in-flutter-using-generated-routes/>
- <https://blog.usejournal.com/flutter-advance-routing-and-navigator-971c1e97d3d2>

Dart Programming

Lecture -2 Module 1

Introduction to Dart Programming Language

- * Coding style and naming convention, Declaring variables, Numbers, Booleans, const and final keywords Dart Types, Dart operators, Control flow and functions, Understanding classes and constructors, Data structures

Installation

IntelliJ IDEA

- Install Dart SDK
- Install IntelliJ IDEA
- Integrate Dart Plugin

DartPad

- No-download or setup needed
- Go online and write code

What is Dart

- * Developed by Google
- * Object oriented language
- * Runs on multiple environment

In Visual Code

- * Dart extension – dart
- * Create file save as filename.dart
- * Run file – In Terminal
 - * - path to file >dart filename.dart

Fundamentals

- * Static-type
- * Compiled - **Ahead-of-Time (AOT) and Just-In-Time(JIT)**
- * Print and read
- * Comments - Inline, block, Documentation

Comments

// This is a normal, one-line comment.

/// This is a documentation comment, used to document libraries,
/// classes, and their members. Tools like IDEs and dartdoc treat
/// doc comments specially.

/* Comments like these are also supported. */

Data Types

EVERYTHING IN DART IS AN OBJECT

- Dart has special support for these data types
 - Numbers
 - int
 - double
 - Strings
 - Booleans
 - Lists (also known as Arrays)
 - Maps
 - Runes (for expressing Unicode characters in a String)
 - Symbols

Values are by default null



```
main() {  
  print("EVEERYTHING IN DART IS OBJECT\n");  
  int price1 = 250;  
  var price2 = 500;  
  print("Price1 is \$price1|Price2 is \$price2\n");  
  double dprice1 = 250.98;  
  var dprice2 = 500.78;  
  print("Double Price1 is \$dprice1|Double Price2 is \$dprice2\n");  
  String name1 = "Leena";  
  var name2 = 'Sahu';  
  print("Name1 is \$name1|Surname is \$name2\n");  
  bool isOn1 = true;  
  var isOn2 = false;  
  print("isOn1 is \$.isOn1|isOn2 is \$isOn2");  
  /*dynamic variables without static type are dynamic,  
  can be used in place of var*/  
  dynamic simple = 'Hello';  
  print('Simple varible = \$simple');  
  simple = 400;  
  print('Simple varible now  = \$simple');  
  simple = null;  
  print('Simple varible now  = \$simple');
```

▶ RUN

Console

EVEERYTHING IN DART IS OBJECTN

Price1 is 250|Price2 is 500

Double Price1 is 250.98|Double Price2 is 500.78

Name1 is Leena|Surname is Sahu

isOn1 is true|isOn2 is false

Simple varible = Hello

Simple varible now = 400

Simple varible now = null

Variables

- * Even in type-safe Dart code, most variables don't need explicit types, due to **type inference**

```
var name = 'Voyager I';      or  String name= 'Voyager I';
var year = 1977;              or int year=1977;
var antennaDiameter = 3.7;
var flybyObjects = ['Jupiter', 'Saturn', 'Uranus', 'Neptune'];
var image = { 'tags': ['saturn'],
  'url': 'https://upload.wikimedia.org/wikipedia/commons/1/17/Google-flutter-logo.png' };
```

Defining Constants and final

- Difference between `final` and `const`
 - `final` variable can only be set once and it is initialized when accessed.
 - `const` variable is implicitly final but it is a compile-time constant
 - i.e. it is initialized during compilation
- Instance variable can be `final` but cannot be `const`.
 - If you want a Constant at Class level then make it `static const`

```
main(){  
    final Car = "Honda City";  
    final wheels = 4;  
    // Car = "Fortuner";  
    print(Car);  
    print(wheels);  
    const pi = 3.14;  
    const gravity = 9.8;  
    //pi = 1.1;  
    print(pi);  
    print(gravity);  
}
```

Honda City
4
3.14
9.8

```
constandfinal.dart:4:3: Error: Can't assign to the final variable 'Car'.  
    Car = "Fortuner";  
    ^^^  
constandfinal.dart:9:3: Error: Can't assign to the const variable 'pi'.  
    pi = 1.1;  
    ^^
```

String Interpolation

DartPad [New Pad](#) [Reset](#) [Format](#) [Install SDK](#)

```
main() {  
  var s1 = 'Single quote works.';  
  var s2 = "Double quotes works as well.";  
  var s3 = 'We can use string delimiter. For eg, it\'s';  
  var s4 = "String delimiter works too. For eg, it's";  
  print(s1); print(s2); print(s3); print(s4); print('');  
 // String Interpolation : Use ["My name is $name"]  
 //instead of ["My name is " + name]  
 String name = "Leena"; print("My name is $name");  
 //print("The no.of characters in String $name is $name.length");  
 print("The number of characters in String $name is ${name.length}\n");  
 //Raw strings treat backslashes (\) as a literal character.  
 var s = r'Raw string does not escapes even \n.';// \n not works  
 print(s);  
 var notraw="what raw not does \n this is an example";  
 print(notraw);  
 //String Interpolation  
 var age = 20; var str = "\nYour age is $age"; print(str);  
 //MultiLine string  
 var s5 = ''  
 \nMulti-line string with 3 single quotes.'';  
 var s6 = """  
 Multi-line string with 3 double quotes.""";  
 print(s5); print(s6);  
}
```

▶ RUN

billowing-fog-9181

Console

Single quote works.

Double quotes works as well.

We can use string delimiter. For eg, it's

String delimiter works too. For eg, it's

My name is Leena

The number of characters in String Leena is 5

Raw string does not escapes even \n.

what raw not does

this is an example

Your age is 20

Multi-line string with 3 single quotes.

Multi-line string with 3 double quotes.

Documentation

Control flow statements

- * Dart supports the usual control flow statements:

```
if (year >= 2001) {  
    print('21st century');  
}  
else if (year >= 1901) {  
    print('20th century');  
}
```

```
for (var object in flybyObjects) {  
    print(object);  
}
```

```
for (int month = 1; month <= 12;  
month++) {  
    print(month);  
}
```

```
while (year < 2016) {  
    year += 1;  
}
```

If-else example

DartPad <> New Pad ⌂ Reset ⌒ Format ⬇ Install SDK b

```
void main() {
  // IF and ELSE Statements
  var salary = 15000;

  if (salary > 20000) {
    print("You got promotion. Congratulations !");
  } else {
    print("You need to work hard !");
  }

  // IF ELSE IF Ladder statements
  var marks = 70;

  if (marks >= 90 && marks < 100) {
    print("A+ grade");
  } else if (marks >= 80 && marks < 90) {
    print("A grade");
  } else if (marks >= 70 && marks < 80) {
    print("B grade");
  } else if (marks >= 60 && marks < 70) {
    print("C grade");
  } else if (marks > 30 && marks < 60) {
    print("D grade");
  } else if (marks >= 0 && marks < 30) {
    print("You have failed");
  } else {
    print("Invalid Marks. Please try again !");
  }
}
```

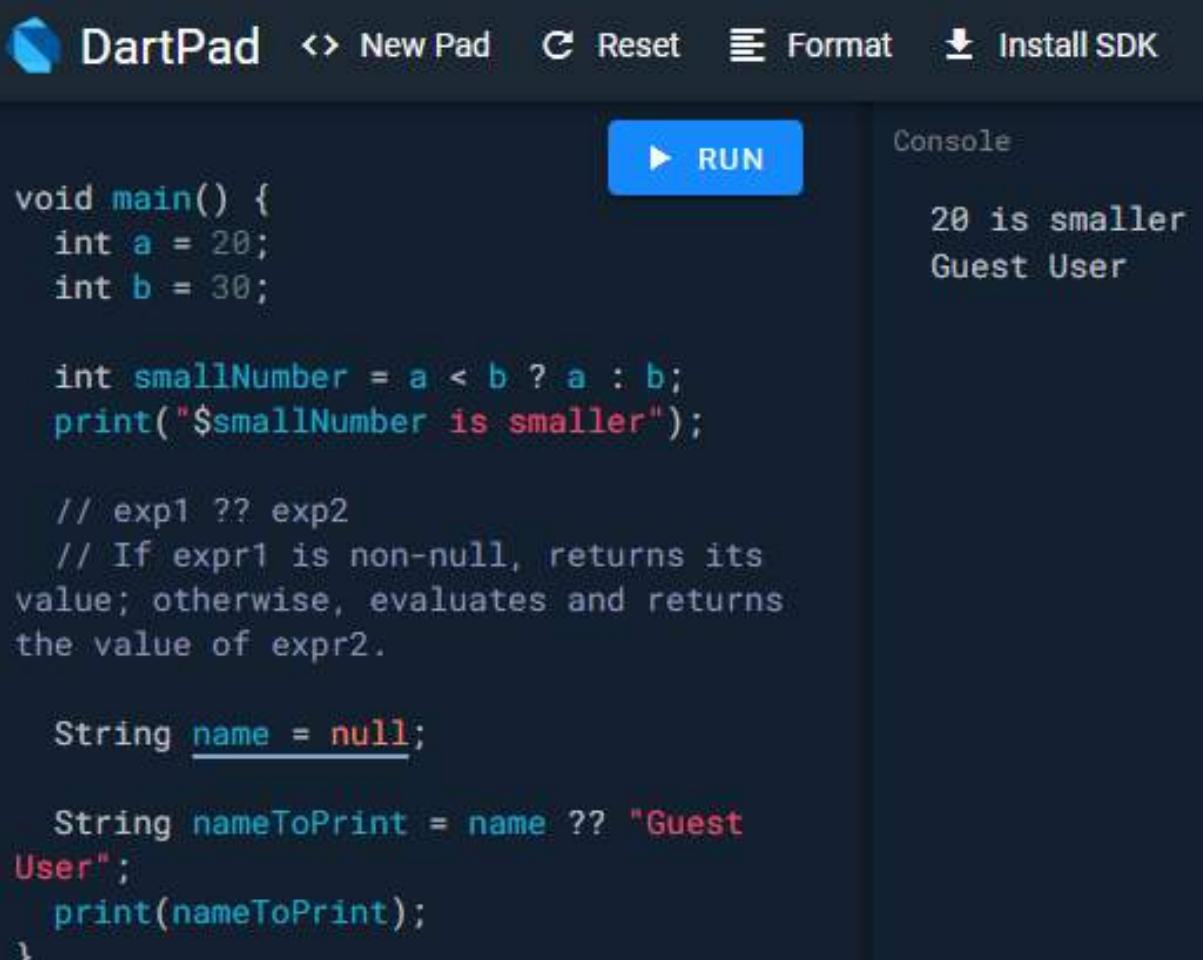
▶ RUN

Console

```
You need to work hard !
B grade
```

Documentation

Conditional statements



DartPad < New Pad C Reset E Format D Install SDK

```
void main() {
  int a = 20;
  int b = 30;

  int smallNumber = a < b ? a : b;
  print("smallNumber is smaller");

  // expr1 ?? expr2
  // If expr1 is non-null, returns its
  value; otherwise, evaluates and returns
  the value of expr2.

  String name = null;

  String nameToPrint = name ?? "Guest
User";
  print(nameToPrint);
}
```

▶ RUN

Console

20 is smaller
Guest User

Switch

DartPad <> New Pad C Reset ┌ Format ┌ Install SDK

► RUN

```
void main() {
// Switch Case Statements: Applicable for only 'int' and
'String'
String grade = 'F';
switch (grade) {
  case 'A':
    print("Excellent grade of A");      break;
  case 'B':
    print("Very Good !");      break;
  case 'C':
    print("Good enough. But work hard"); break;
  case 'F':
    print("You have failed");      break;
  default:
    print("Invalid Grade");
}
}
```

Console

You have failed

Looping

DartPad < New Pad C Reset Format Install SDK bill

```
void main() {
  print("FOR Loop demo");
  for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
      print(i);
    }
  }
  List planetList = ["Mercury", "Venus", "Earth", "Mars"];
  for (String planet in planetList) {
    print(planet);
  }
  print("WHILE Loop demo");
  var i = 10;
  while (i <= 20) {
    if (i % 2 == 0) {
      print(i);
    }
    i++;
  }
  print("DO-WHILE Loop demo");
  var j = 1;
  do {
    if (j % 2 == 0) {
      print(j);
    }
    j++;
  } while (j <= 10);
}
```

▶ RUN

Console

```
4
6
8
10
Mercury
Venus
Earth
Mars
WHILE Loop demo
10
12
14
16
18
20
DO-WHILE Loop demo
2
4
6
8
10
```

Documentation

abstract class int ex

An integer number.

break and continue

DartPad <> New Pad C Reset E Format ⬇ Install SDK billowi

```
void main() {
  print("Break and Using Labels");
  myOuterLoop:
  for (int i = 1; i <= 3; i++) {
    //innerLoop:
    for (int j = 1; j <= 3; j++) {
      print("$i $j");

      if (i == 2 && j == 2) {
        break myOuterLoop;
      }
    }
  }
  print("Continue and Using Labels");
  myLoop:
  for (int i = 1; i <= 3; i++) {
    //myInnerLoop:
    for (int j = 1; j <= 3; j++) {
      if (i == 2 && j == 2) {
        continue myLoop;
      }
      print("$i $j");
    }
  }
}
```

▶ RUN

Console

```
Break and Using Labels
1 1
1 2
1 3
2 1
2 2
Continue and Using Labels
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Documentation

Functions and its properties

- Functions in Dart are **Objects**.
 - Functions can be assigned to a variable or passed as parameter to other functions.
- All functions in Dart returns a value
 - If no return value is specified the function return **null**
- Specifying return type is optional but is recommended as per code convention

```
int findArea(int length, int breadth) {  
    // Function Body: Put your code here  
    return length * breadth;  
}
```

```
findArea(int length, int breadth) {  
    // by default, returns null  
}
```

Same

```
int findArea(int length, int breadth) {  
    // Again, it returns null as return statement is missing  
}
```

```
void findArea(int length, int breadth) {  
    print(length * breadth);  
}
```

Same

```
findArea(int length, int breadth) {  
    print(length * breadth);  
}
```

Functions



DartPad

↔ New Pad

↻ Reset

≡ Format

⬇ Install SDK

billowing-fog

```
//1. Function with no arguments and return type
int findAge() {
  int age = 25;
  return age;
}

//2. Function with arguments and no return type
aboutMySelf(int age, int totalGf) {
  print("Five years ago When I was ${age - 5} ");
  print("I had $totalGf Friends");
}

//3. Function with arguments and with return type
int countFriends(int newF, int oldF) {
  int friends = newF + oldF;
  return friends;
}

main() {
  int myAge = findAge();
  print("My age is $myAge");
  aboutMySelf(myAge, 35);
  int friendcount = countFriends(20, 30);
  print("My Old and New Friend count is $friendcount");
}
```

▶ RUN

Console

```
My age is 25
Five years ago When I was 20
I had 35 Friends
My Old and New Friend count is 50
```

Documentation

Shorthand Expression

- Short manner to represent function
- It is used when function returns single expression

Syntax -

```
return type function_name(arguments) =>  
expression;
```

A shorthand => (arrow) syntax is handy for functions that contain a single statement. This syntax is especially useful when passing anonymous functions as arguments:

Shorthand Expression Example

DartPad <> New Pad C Reset  Format  Install SDK

```
//4. Lamda Expression
void main() {
    findPerimeter(4, 2);
    int rectArea = getArea(10, 5);
    print("The area is $rectArea");
}

void findPerimeter(int length, int breadth) =>
    print("The perimeter is ${2 * (length + breadth)}");
int getArea(int length, int breadth) => length * breadth;
```

▶ RUN

Console

The perimeter is 12
The area is 50

Task to do

WAP which will take details of an Employee (eid , eage, ename, salary).(take static values for now)

Use Shorthand exp in function for calculation of salary based on da :45% of salary

And Calculate Remaining years of service (retirement age 60 assume)

If age 25- 35 :ecategory ' Team Lead'

36-50:ecategory 'Manager'

51-60:ecategory ' Executive '

Use switch Case for below

If emp is team lead print "Keep motivating your team "

If emp is Manager print" Know all perspectives"

If emp is Executive print" Support your manager and team "

Print All details of the employee

Expected output pattern

Employee Name: Leena Sahu

Employee Age : 31

Employee ID: KJSCE112

Years of service remaining:29

Employee basic salary: 25000

Employee total salary: calculated value

Employee Category: Team Lead (based on age)

Message to Employee : Keep motivating your team

Employee Name: Leena1 Sahu1

Employee Age : 55

Employee ID: KJSCE20

Years of service remaining:6 **calculated**

Employee basic salary: 30000

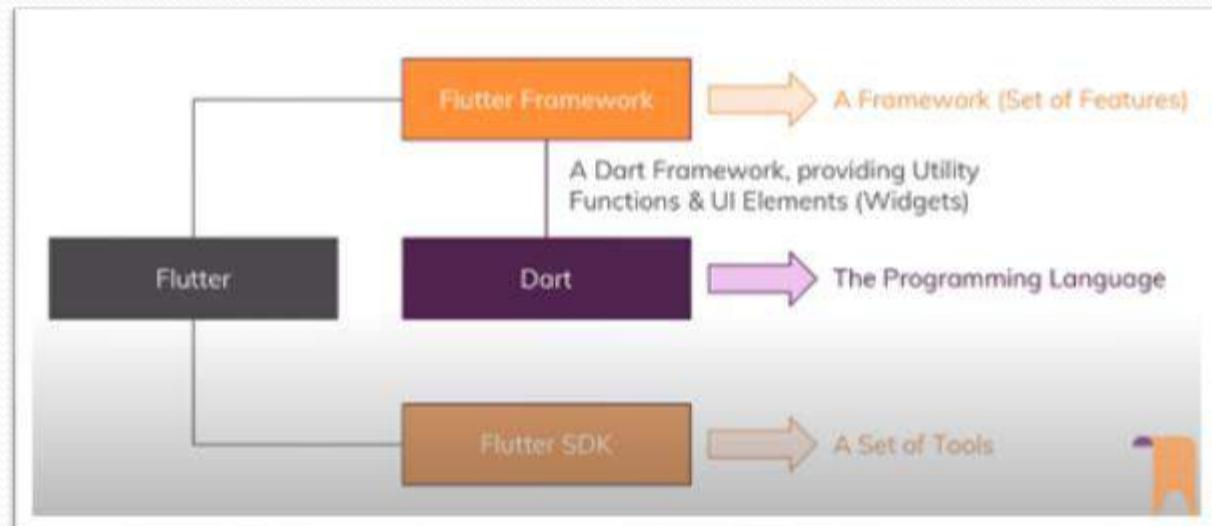
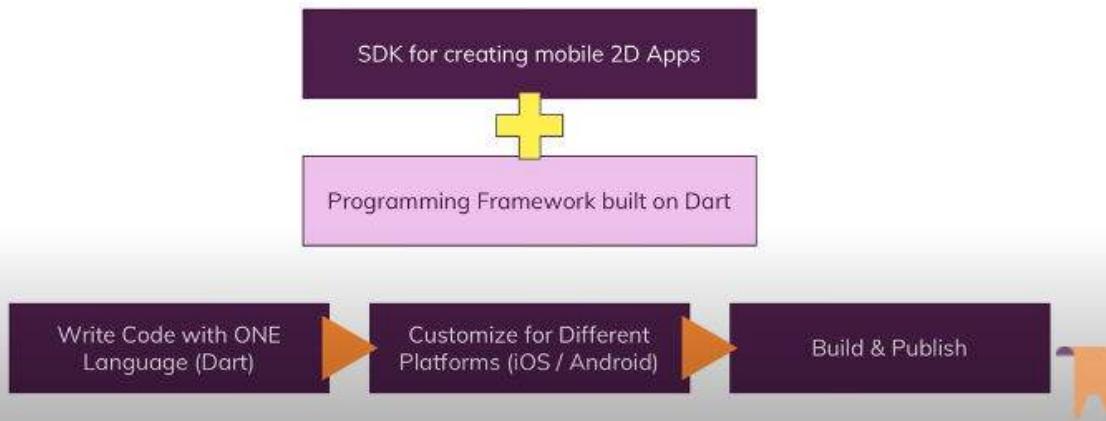
Employee total salary: calculated value

Employee Category: Executive(based on age)

Message to Employee :Support your manager and team

Flutter: Introduction

What is Flutter



Flutter Architecture



Build a **Widget Tree**

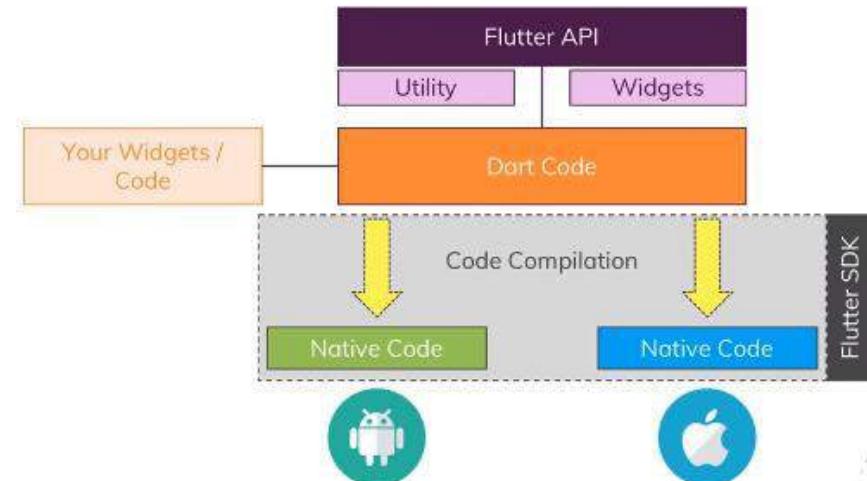


Embrace **Platform Differences**



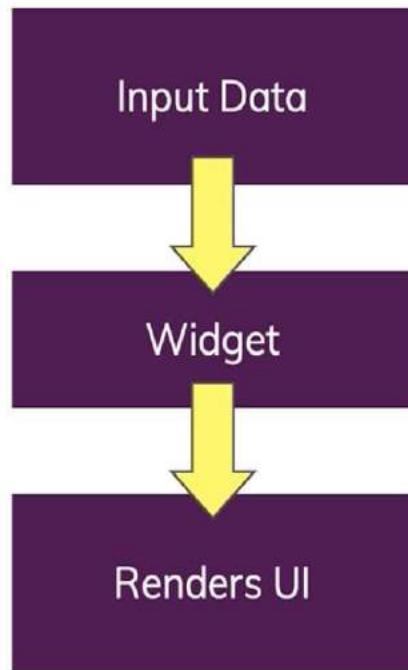
One Codebase

Flutter & Dart converted to native apps



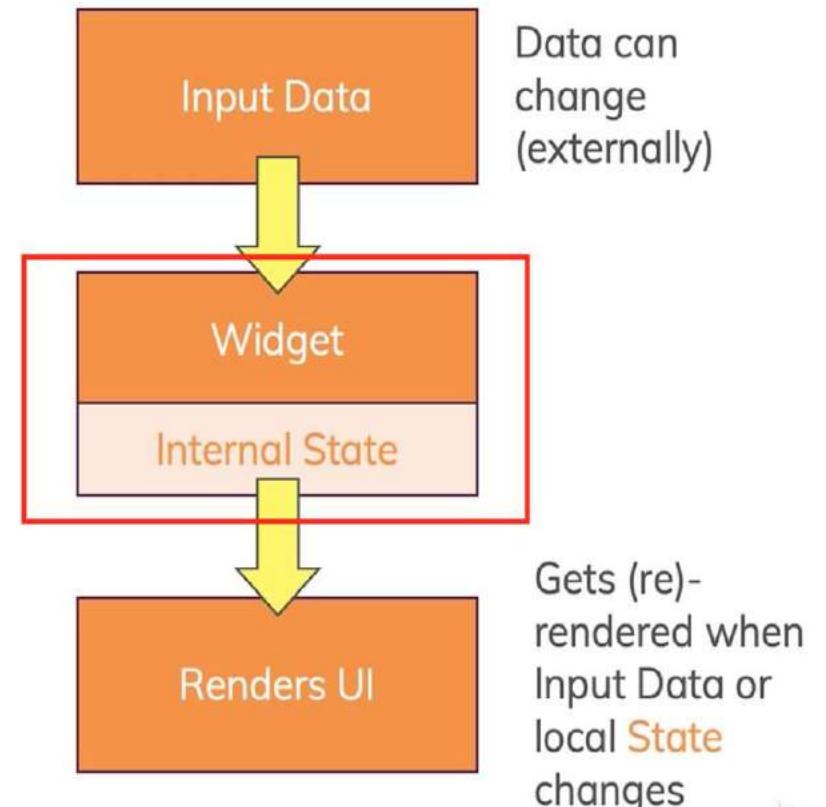
Stateless vs Stateful

Stateless



Data can
change
(externally)

Stateful



Gets (re)-
rendered
when **Input
Data**
changes

Data can
change
(externally)

Gets (re)-
rendered when
Input Data or
local **State**
changes

StatelessWidget

constructor

build

StatefulWidget

constructor

createState

A single StatelessWidget can build in many different BuildContexts

A StatefulWidget creates a new State object for each BuildContext

Widgets

- The central idea is that you build your UI out of widgets.
- **Widgets describe what their view should look like given their current configuration and state.**
- When a widget's state changes, the widget rebuilds its description, which the framework diffs against the previous description in order to determine the minimal changes needed in the underlying render tree to transition from one state to the next.

Basic

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

void main() {
  runApp(
    Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
}
```

- The `runApp()` function takes the given **Widget** and makes it the root of the widget tree. In this example, the widget tree consists of two widgets, the **Center** widget and its child, the **Text** widget.

Stateful and Stateless widget

- Widgets that are subclasses of either `StatelessWidget` or `StatefulWidget`, depending on whether your widget manages any state.
- A widget's main job is to implement a `build()` function, which describes the widget in terms of other, lower-level widgets.
- The framework builds those widgets in turn until the process bottoms out in widgets that represent the underlying `RenderObject`, which computes and describes the geometry of the widget.

Basic widgets

- Text
- Row,Column
- Stack(Stacks are based on the web's absolute positioning layout model.)
- Container-The Container widget lets you create a rectangular visual element. A container can be decorated with a **BoxDecoration**, such as a background, a border, or a shadow. A Container can also have margins, padding, and constraints applied to its size.

Using Material Components

- Flutter provides a number of widgets that help you build apps that follow Material Design.
- A Material app starts with the **MaterialApp** widget, which builds a number of useful widgets at the root of your app, including a **Navigator**, which manages a stack of widgets identified by strings, also known as “routes”. The Navigator lets you transition smoothly between screens of your application.

Scaffold and AppBar

- Widgets are passed as arguments to other widgets.
- The **Scaffold** widget takes a number of different widgets as named arguments, each of which are placed in the Scaffold layout in the appropriate place.
- Similarly, the **AppBar** widget lets you pass in widgets for the **leading** widget, and the **actions** of the **title** widget. This pattern recurs throughout the framework and is something you might consider when designing your own widgets.

Handling gestures

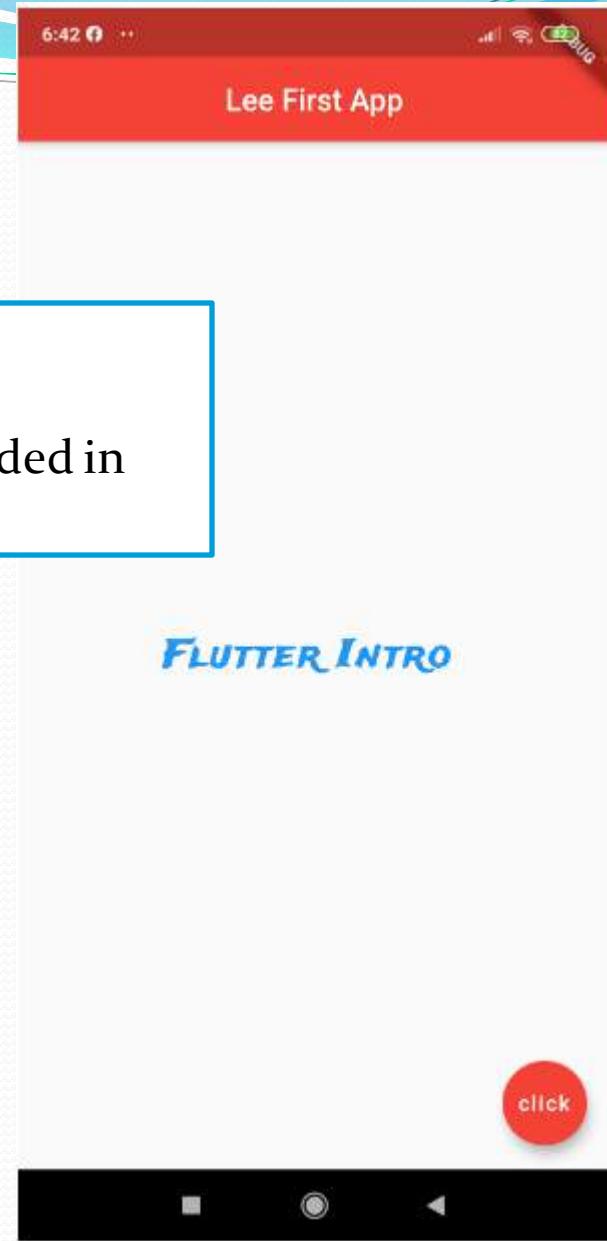
- Most applications include some form of user interaction with the system. The first step in building an interactive application is to detect input gestures.

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

```
void main() => runApp(MaterialApp(home: Home()));
```

```
class Home extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Lee First App'),  
        centerTitle: true,  
        backgroundColor: Colors.red,  
      ),  
      body: Center(  
        child: Text('FlutterIntro',  
          style: TextStyle(  
            fontSize: 25.0,  
            fontWeight: FontWeight.bold,  
            letterSpacing: 2.0,  
            color: Colors.blue,  
            fontFamily: 'TradeWinds',  
          )),  
      ),  
      floatingActionButton: FloatingActionButton(  
        onPressed: () {},  
        child: Text('click'),  
        backgroundColor: Colors.red,  
      ),  
    );  
  }  
}
```

Here **TradeWinds**
fontfamily I have
downloaded and added in
pubspec.yaml file

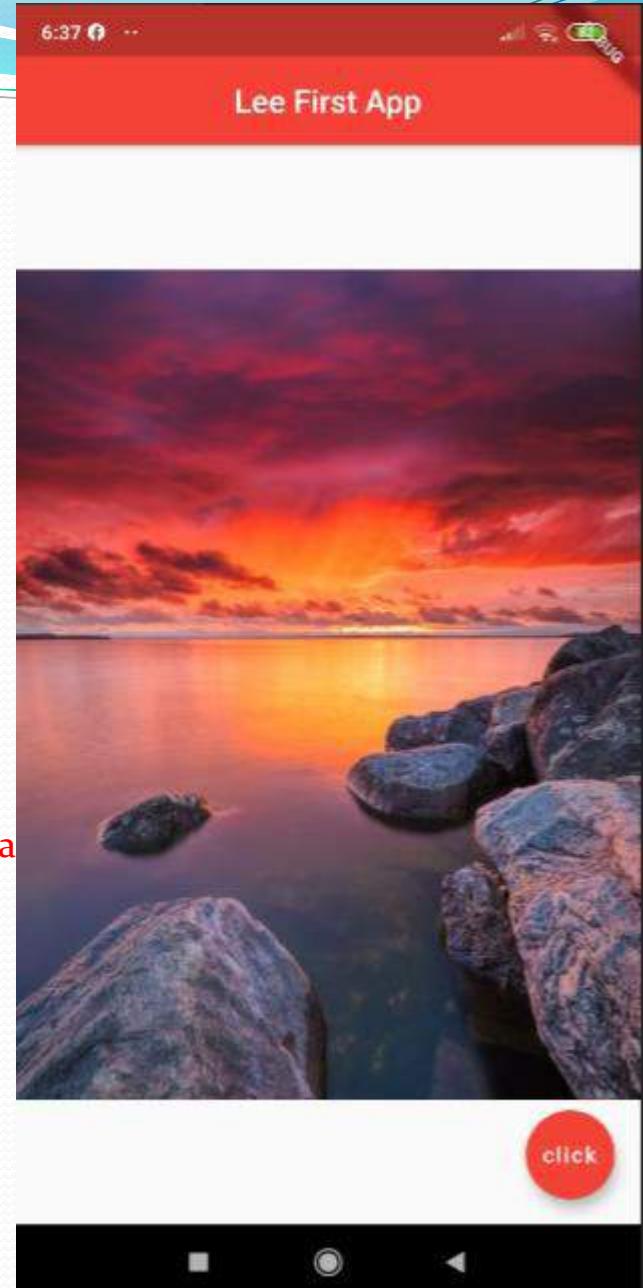


```
import 'package:flutter/material.dart';

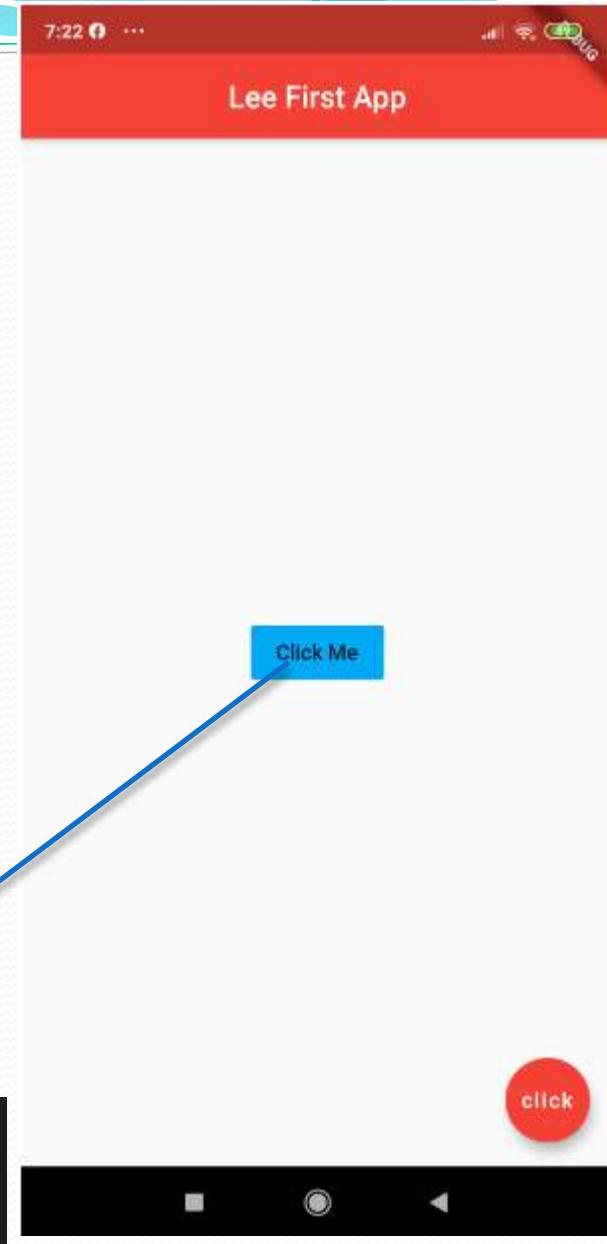
void main() => runApp(MaterialApp(
    home: Home(),
));

class Home extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title: Text('Lee First App'),
                centerTitle: true,
                backgroundColor: Colors.red,
            ),
            body: Center(
                child: Image.asset(
                    'assets/55.jpg'),
                /*Image(image: AssetImage('assets/33.jpg'),), or NetworkImage('http://...')*/
            ),
            floatingActionButton: FloatingActionButton(
                onPressed: () {},
                child: Text('click'),
                backgroundColor: Colors.red,
            ),
        );
    }
}
```

Icon(
 Icons.airport_shuttle,
 color: Colors.lightBlue
,
 size: 50.0,
)



```
class Home extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Lee First App'),  
        centerTitle: true,  
        backgroundColor: Colors.red,  
      ),  
      body: Center(  
        child: FlatButton(  
          onPressed: () {  
            print('You Clicked Me');  
          },  
          child: Text('Click Me'),  
          color: Colors.lightBlue,  
        ),  
      ),  
      floatingActionButton: FloatingActionButton(  
        onPressed: () {},  
        child: Text('click'),  
        backgroundColor: Colors.red,  
      ),  
    );  
  }  
}
```



```
class Home extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Lee First App'),  
        centerTitle: true,  
        backgroundColor: Colors.red,  
      ),  
      body: Center(  
        child: RaisedButton.icon(  
          onPressed: () {  
            print('mail button clicked');  
          },  
          icon: Icon(Icons.mail),  
          label: Text('Mail Me'),  
          color: Colors.amber,  
        ),  
        ),  
        floatingActionButton: FloatingActionButton(  
          onPressed: () {},  
          child: Text('click'),  
          backgroundColor: Colors.red,  
        ),  
      );  
    }  
}
```

Lee First App

 Mail Me

```
child: IconButton(  
  onPressed: () {  
    print('Mail Icon clicked');  
  },  
  icon: Icon(Icons.alternate_email),  
  color: Colors.red,  
,
```

click

Container

```
body: Container(
```

```
    padding: EdgeInsets.fromLTRB(10.0, 20.0, 30,
```

```
    margin: EdgeInsets.all(30),
```

```
    color: Colors.grey,
```

```
    child: Text('Hello Leena'),
```

```
),
```

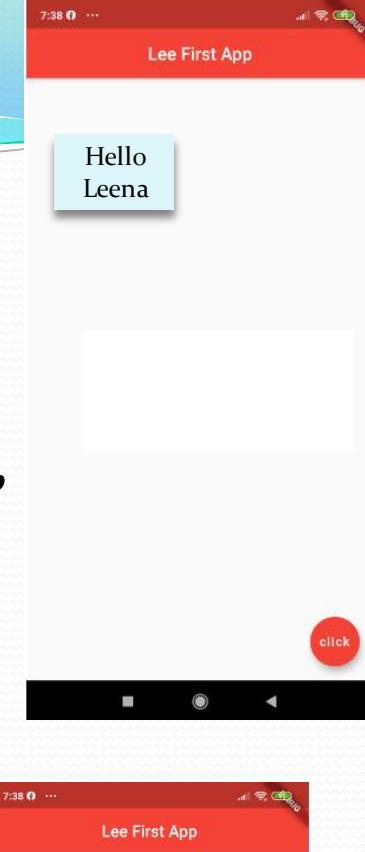
Or

```
body: Padding(
```

```
    padding: EdgeInsets.all(90.0),
```

```
    child: Text("AppTest"),
```

```
),
```



AppTest



click

Rows

body: Row(

 mainAxisAlignment: MainAxisAlignment.spaceEvenly,

 crossAxisAlignment: CrossAxisAlignment.start,

 children: <Widget>[

 Text('Hello All'),

 FlatButton(

 onPressed: () {},

 color: Colors.amber,

 child: Text('Click Me'),

),

 Container(

 color: Colors.cyan,

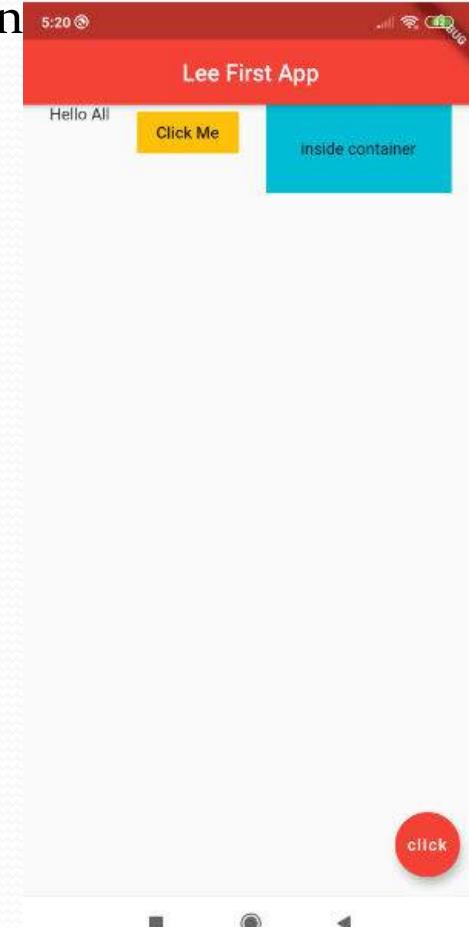
 padding: EdgeInsets.all(30),

 child: Text('inside container'),

),

],

),



Columns

```
body: Column(
```

```
    mainAxisAlignment: MainAxisAlignment.end,
```

```
    crossAxisAlignment: CrossAxisAlignment.start,
```

```
    children: <Widget>[
```

```
        Container(
```

```
            padding: EdgeInsets.all(20),
```

```
            color: Colors.pinkAccent,
```

```
            child: Text('one'),
```

```
        ),
```

```
        Container(
```

```
            padding: EdgeInsets.all(30),
```

```
            color: Colors.cyan,
```

```
            child: Text('two'),
```

```
        ),
```

```
        Container(
```

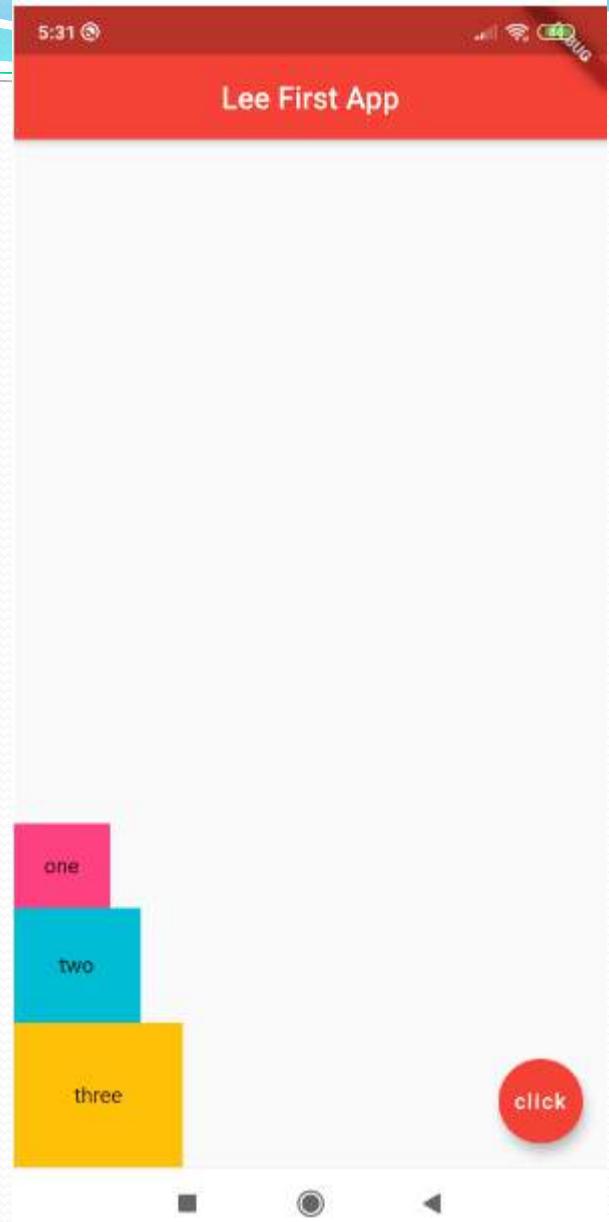
```
            padding: EdgeInsets.all(40),
```

```
            color: Colors.amber,
```

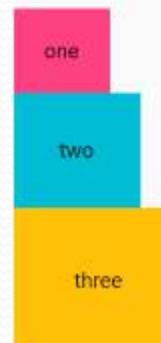
```
            child: Text('three'),
```

```
        ),
```

```
    ],
```



Lee First App



click

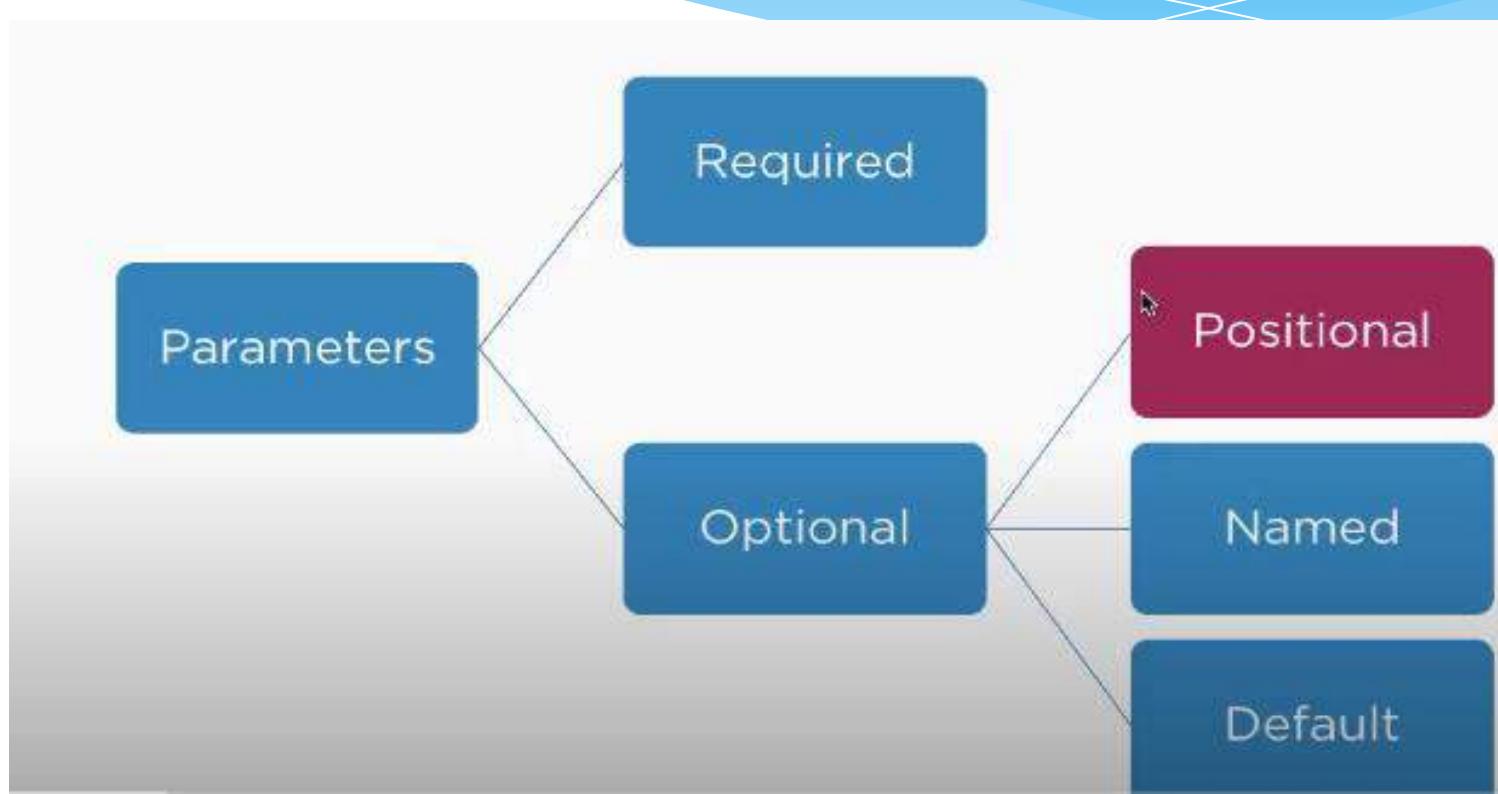
References

- https://www.youtube.com/watch?v=r_yQ6RFoZ1k&list=PLRAV69dS1uWT-ooTYHCqgxMTGA233JMrP&index=10
- <https://flutter.dev/docs/development/ui/layout#1-select-a-layout-widget>

Lecture 3-Module 1

Functions and Exceptions

Optional Positional Parameters in Functions



Named Parameters

Prevent errors if there are large number of parameters

```
int findVolume( int length, { int breadth, int height } ) {  
    return length * breadth * height;  
}  
  
var result = findVolume(2, breadth:3, height: 10);  
print(result);          // 2*3*10 = 60  
  
var result = findVolume(2, height: 10, breadth: 3); // Sequence does not matter  
print(result);          // 2*3*10 = 60
```

Required and positional parameters

DartPad <> New Pad C Reset ┌ Format └ Install SDK

```
void Branches(String b1, String b2, String b3) {
  print("Branch 1 $b1");
  print("Branch 2 $b2");
  print("Branch 3 $b3");
}

void Subjects(String s1, [String s2, String s3]) {
  print("Branch 1 $s1");
  print("Branch 2 $s2");
  print("Branch 3 $s3");
}

void main(){
  Branches("IT", "COMPS", "MECH");
  print("");
  Subjects("MPD");
}
```

► RUN

Console

```
Branch 1 IT
Branch 2 COMPS
Branch 3 MECH
```

```
Branch 1 MPD
Branch 2 null
Branch 3 null
```

named and default parameters



DartPad

↔ New Pad

C Reset

≡ Format

⬇ Install SDK

billowing-fog-9181

```
void findVol(int len, int dep, {int br = 6, int ht = 8}) {  
  print("Length is $len || Breadth is $br");  
  print("height is $ht || depth is $dep");//dep used just for demo  
  print("Volume is ${len * br * ht}");  
}  
  
void main() {  
  //Named parameters can be called using their names  
  findVol(10, 40, ht: 30, br: 30); //sequence can be changed  
  print("");  
  findVol(10, 5, br: 30, ht: 40);  
  print("");  
  findVol(40, 6); // will use default values  
  print("");  
  findVol(40, 6, ht: 10); // will use default values  
}
```

▶ RUN

Console

```
Length is 10 || Breadth is 30  
height is 30 || depth is 40  
Volume is 9000
```

```
Length is 10 || Breadth is 30  
height is 40 || depth is 5  
Volume is 12000
```

```
Length is 40 || Breadth is 6  
height is 8 || depth is 6  
Volume is 1920
```

```
Length is 40 || Breadth is 6  
height is 10 || depth is 6  
Volume is 2400
```

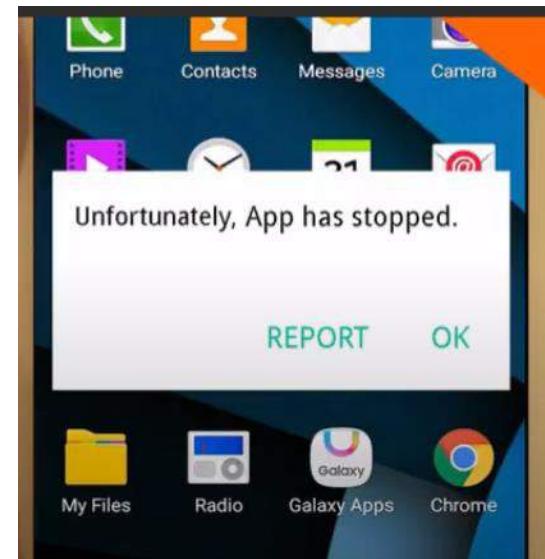
Default Parameters

You can assign default values to parameters

```
int findVolume( int length, int breadth, { int height = 10 } ) {  
    return length * breadth * height;  
}  
  
var result = findVolume(2, 3);  
print(result);      // 2*3*10 = 60  
  
var result = findVolume(2, 3, height: 20);           // Overrides the default value  
print(result);      // 2*3*20 = 120
```

Exception Handling

- * When normal flow of Execution is disrupted and App Crashes.
- * Exceptions can be handled using following clauses to avoid app crash
 - * try
 - * on
 - * catch
 - * Finally
 - * Custom exceptions



try and on clause

```
void main() {  
    print("CASE 1");  
    /* CASE 1: When you know the exception  
     * to be thrown, use ON Clause */  
    try {  
        int result = 12 ~/ 4;  
        print("The result is $result");  
    } on IntegerDivisionByZeroException {  
        print("Cannot divide by Zero");  
    }  
    print("");  
    print("CASE 2");  
    // CASE 2: When you do not know the  
    // exception use CATCH Clause  
    try {  
        int result = 12 ~/ 0;  
        print("The result is $result");  
    } catch (e) {  
        print("The exception thrown is $e");  
    }
```

- * When you know the exception to be thrown, use “on” Clause
- * Use “catch” clause when exception type is not known

CASE 1
Cannot divide by Zero

CASE 2
The exception thrown is IntegerDivisionByZeroException

stacktrace and finally

```
void main() {  
  
  print("CASE 3");  
  // CASE 3: Using STACK TRACE to know the events  
  //occurred before Exception was thrown  
  try {  
    int result = 12 ~/ 0;  
    print("The result is $result");  
  } catch (e, s) {  
    print("The exception thrown is $e");  
    print("STACK TRACE \n $s");  
  }  
  print("");  
  print("CASE 4");  
  // CASE 4: Whether there is an Exception or not,  
  //FINALLY Clause is always Executed  
  try {  
    int result = 12 ~/ 3;  
    print("The result is $result");  
  } catch (e) {  
    print("The exception thrown is $e");  
  } finally {  
    print("This is FINALLY Clause and is always executed.");  
  }  
}
```

```
CASE 3  
The exception thrown is IntegerDivisionByZeroException  
STACK TRACE
```

```
#0      int.~/ (dart:core-patch/integers.dart:24:7)  
#1      main (file:///D:/class%202020-21%20odd/dart%20n%20flutter/dart/DartPrograms/18_exception_handling.dart:3:  
2:21)  
#2      _startIsolate.<anonymous closure> (dart:isolate-patch/isolate_patch.dart:301:19)  
#3      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:168:12)
```

```
CASE 4  
The result is 4  
This is FINALLY Clause and is always executed.
```

User defined /custom exceptions

```
void main() {  
  
    print("CASE 5");  
    // CASE 5: Custom Exception  
    try {  
        depositMoney(-5000);  
    } catch (e) {  
        print(e.errorMessage());  
    } finally {  
        print("Logout Success");  
    }  
}  
  
class DepositException implements Exception {  
    String errorMessage() {  
        return "Transaction Unsuccessful-Amount less than 0";  
    }  
}  
  
void depositMoney(int amount) {  
    if (amount < 0) {  
        throw new DepositException();  
    }  
    else  
        print("Transaction Successful");  
}
```

```
void main() {  
  
    print("CASE 5");  
    // CASE 5: Custom Exception  
    try {  
        depositMoney(2000);  
    } catch (e) {  
        print(e.errorMessage());  
    } finally {  
        print("Logout Success");  
    }  
}  
  
class DepositException implements Exception {  
    String errorMessage() {  
        return "Transaction Unsuccessful-Amount less than 0";  
    }  
}  
  
void depositMoney(int amount) {  
    if (amount < 0) {  
        throw new DepositException();  
    }  
    else  
        print("Transaction Successful");  
}
```

CASE 5
Transaction Unsuccessful-Amount less than 0
Logout Success

CASE 5
Transaction Successful
Logout Success

Imports

```
// Importing core libraries  
import 'dart:math';
```

```
// Importing libraries from external packages  
import 'package:test/test.dart';
```

```
// Importing files  
import 'path/to/my_other_file.dart';
```

class

```
class Spacecraft {  
    String name;  
    DateTime launchDate;  
    // Constructor, with syntactic sugar for assignment to members.  
    Spacecraft(this.name, this.launchDate) {  
        // Initialization code goes here.  
    }  
    // Named constructor that forwards to the default one.  
    Spacecraft.unlaunched(String name) : this(name, null);  
    int get launchYear => launchDate?.year; // read-only non-final property  
    // Method.  
    void describe() {  
        print('Spacecraft: $name');  
        if (launchDate != null) {  
            int years = DateTime.now().difference(launchDate).inDays ~/ 365;  
            print('Launched: $launchYear ($years years ago)');  
        } else {  
            print('Unlaunched');  
        }  
    }  
}
```

```
main() {  
    var voyager =  
        Spacecraft('Voyager I', DateTime(1977, 9, 5));  
    voyager.describe();  
  
    var voyager3 = Spacecraft.unlaunched('Voyager III');  
    voyager3.describe();  
}
```

```
D:\class 2020-21 odd\dart n flutter\dart>dart classdemo.dart  
Spacecraft: Voyager I  
Launched: 1977 (42 years ago)  
Spacecraft: Voyager III  
Unlaunched
```

Task 2-Practice Problem

- * WAP for banking apps with functions -Withdrawal ,Deposit, BalanceCheck functionality. Make use of attributes ex.- Account no., customer name ,account balance, amount to be deposited and withdrawal .Make use of default ,named parameters in the program.
 - * If balance is less than withdrawal amount throw LessBalanceException (custom)
 - * Withdrawal()
 - * Deposit()
 - * BalanceCheck()

NOTE:: what parameters should be passed to functions assume yourself ,should be relevant and display account balance after deposit and withdrawal (Take Static inputs for now check for exceptions) Use concepts covered in the class

Dart References

- * <https://dart.dev/samples#hello-world>

Chapter 3:

Navigation and Multiple Screens

Outline

- Passing Data via the Constructor
- Using Named Routes & Passing Data With Named Routes
- Screens & Navigation
- onGenerateRoute & onUnknownRoute

Routing and Navigation

- **Idea:** Traversing from one Widget to another is important to create an action of switching between pages for the user
- Since everything in Flutter is a Widget, the Navigation and Routing are also going to follow the same concept.
- The Routing is going to create a route to travel between one widget to another, passing data and information between them.
- This will create a better user experience showing a flow from one page to another on click of a button

Routing and Navigation

AlertDialog

BottomNavigationBar

Drawer

MaterialApp

Navigator

SimpleDialog

TabBar

TabBarView

Navigator Widget

- Navigator widget manages a set of child widgets with a **stack discipline**.
- Ex: Dinner plates
- Mobile apps typically reveal their contents via full-screen elements called “screens” or “pages.”
- In Flutter, these elements are called routes, and they’re managed by a **Navigator** widget.
- The navigator leads a stack of **Route** objects and provides methods for controlling the stack, like **Navigator.push** and **Navigator.pop**

Navigator Widget

Navigator.push

It pushes the given route onto the navigator that most tightly encloses the given context. The syntax is following.

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

Navigator.pop

It pops the top-most route off the navigator that most tightly encloses the given context. The syntax is following.

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

Navigation Types

■ Direct Navigation:

Direct navigation is implemented with **MaterialPageRoute**. This is also known as *un-named routing*.

■ Static Navigation:

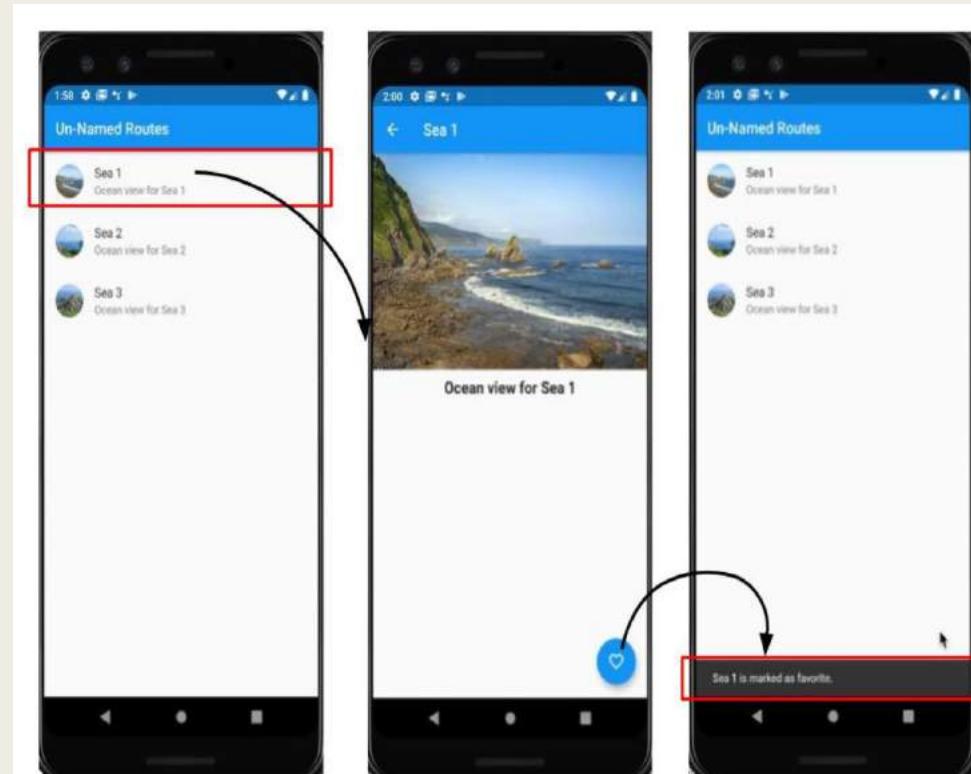
Static navigation is implemented by assigning a map of routes to MaterialApp's **routes** property. The route name is pushed using `Navigator.pushNamed(...)`. This is known as *Named Routing*

■ Dynamic Navigation:

In this navigation, routes are generated by implementing `onGenerateRoute` callback in the `MaterialApp` class. This is a type of Named Routing as well, and the route name is pushed using `Navigator.pushNamed(...)`.

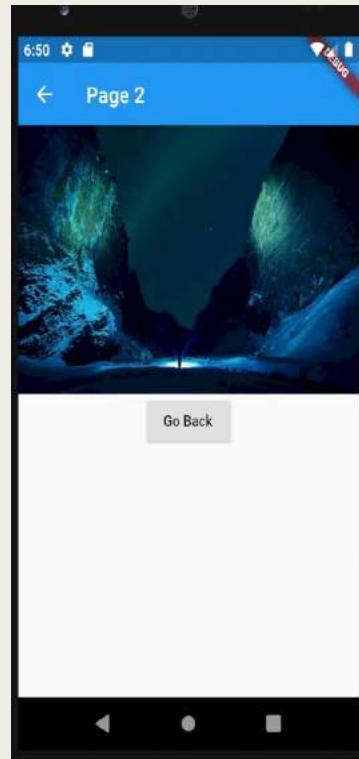
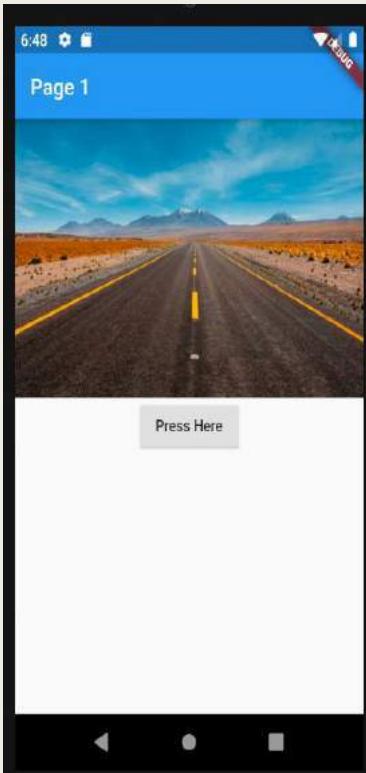
1. Direct Navigation

- Direct navigation is also known as un-named routing.
- It is implemented with MaterialPageRoute.
- The MaterialPageRoute is pushed directly to the navigator.



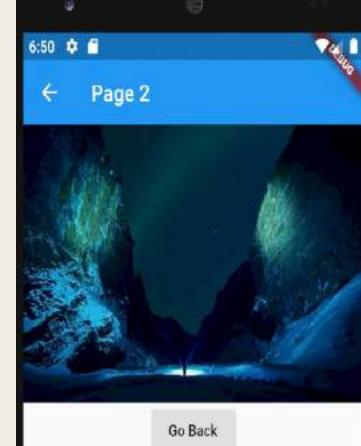
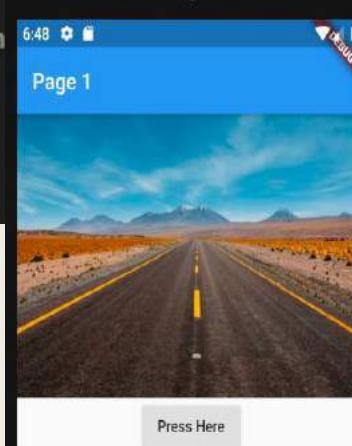
1. Direct Navigation Example (Routing to a widget and back)

1. Create two routes
2. Navigate to the second route using Navigator.push()
3. Return to the first route using Navigator.pop()



```
1 import 'package:flutter/material.dart';
Run | Debug
2 void main()
3 {
4   runApp(MaterialApp(
5     title: 'Androidmonks',
6     home: FirstWidget(),
7   )); // MaterialApp
8 }
9 class FirstWidget extends StatelessWidget
10 {
11   @override
12   Widget build(BuildContext context) {
13     return Expanded(
14       child: Scaffold(
15         appBar: AppBar(title: Text("Page 1")),
16         body: Column(children: <Widget>[
17           Image.asset('assets/images/image1.jpg'),
18           RaisedButton(child: Text("Press Here"),
19             onPressed: () {
20               Navigator.push(
21                 context,
22                 MaterialPageRoute(builder: (context) => SecondWidget()),
23               );
24             },
25           ), // RaisedButton
26         ],), // <Widget>[] // Column
27       ), // Scaffold
28     ); // Expanded
29 }
```

```
31     class SecondWidget extends StatelessWidget
32     {
33         @override
34         Widget build(BuildContext context) {
35             return Scaffold(
36                 appBar: AppBar(title: Text("Page 2"),),
37                 body: Column(children: <Widget>[
38                     Image.asset('assets/images/image2.jpg'),
39                     RaisedButton(child:Text("Go Back"),
40                         onPressed: () {
41                             Navigator.pop(context);
42                         }
43                     ), // RaisedButton
44                 ],) // <Widget>[] // Column
45             ); // Scaffold
46         }
47     }
```



Direct Navigation Examples (Send Data across Screens)

Send Values to another Screen

In order to send Values between the Screens, we can make use of the constructor inside the receiving Screen

Steps to perform:

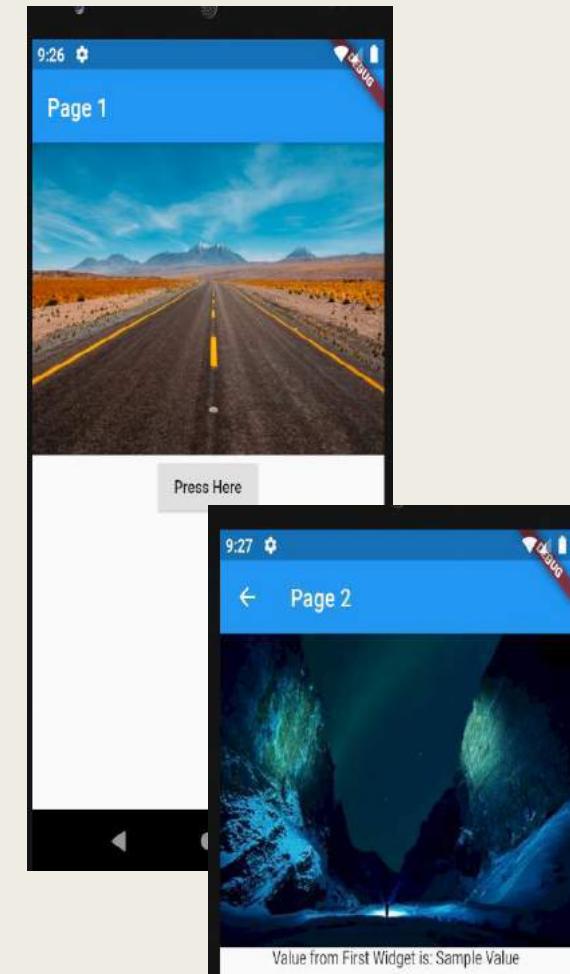
1. Create two screen
2. Define the routes
3. Create a constructor to receive the value that is passed from the first screen
4. Store the value in a class variable
5. Print the value

Send Values to another Screen(1)

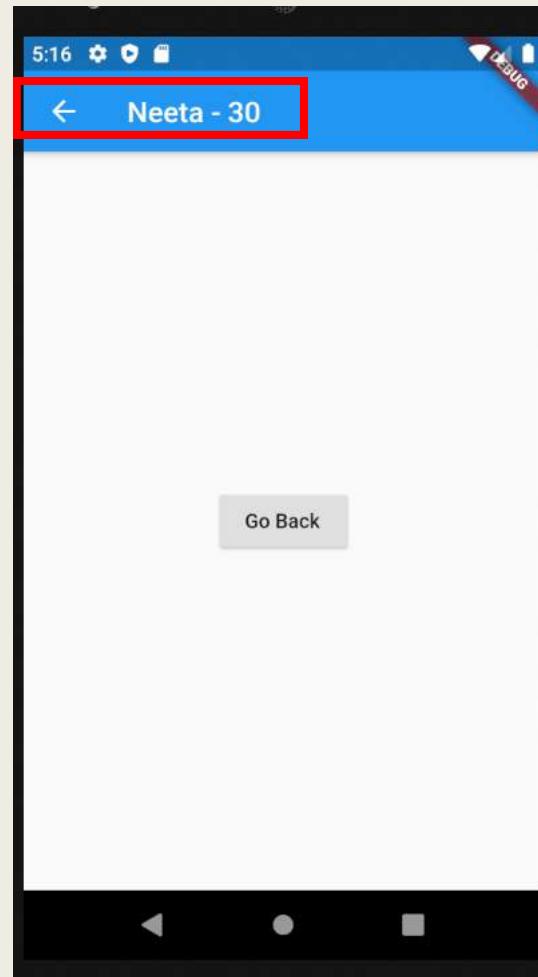
```
1 import 'package:flutter/material.dart';
2
3 Run | Debug
4 void main()
5 {
6     runApp(MaterialApp(
7         home:FirstWidget(),
8     )); // MaterialApp
9 }
10 class FirstWidget extends StatelessWidget
11 {
12     @override
13     Widget build(BuildContext context) {
14         return Scaffold(
15             appBar: AppBar(title: Text("Page 1")),
16             body: Column(children: <Widget>[
17                 Image.asset("assets/images/image1.jpg"),
18                 RaisedButton(child: Text("Press Here"),
19                 onPressed: () {
20                     Navigator.push(
21                         context,
22                         MaterialPageRoute(builder: (context) => SecondWidget(value: "Sample Value")))
23                 );
24             ) // RaisedButton
25         ],) // <Widget>[] // Column
26     ); // Scaffold
27 }
28
29 }
```

Send Values to another Screen(2)

```
30 class SecondWidget extends StatelessWidget
31 {
32     String value;
33     SecondWidget({Key key, @required this.value}):super(key:key);
34     @override
35     Widget build(BuildContext context) {
36         return Scaffold(
37             appBar: AppBar(title: Text("Page 2"),),
38             body: Column(children: <Widget>[
39                 Image.asset("assets/images/image2.jpg"),
40                 Text("Value from First Widget is: "+this.value),
41                 RaisedButton(child: Text("Go Back"),
42                         onPressed: () {
43                             Navigator.pop(context);
44                         }), // RaisedButton
45             ],) // <Widget>[] // Column
46         ); // Scaffold
47     }
48 }
```



Example:2



Example: 2

```
Run | Debug
2 void main() {
3     runApp(MaterialApp(
4         home: HomePage(),
5     )); // MaterialApp
6 }
7
8 class User {
9     final String name;
10    final int age;
11    User({this.name, this.age});
12 }
13
14 class HomePage extends StatelessWidget {
15     @override
16     Widget build(BuildContext context) {
17         return Scaffold(
18             appBar: AppBar(
19                 title: Text('Home'),
20             ), // AppBar
21             body: Center(
22                 child: RaisedButton(
23                     onPressed: () {
24                         User user = new User(name: 'Neeta', age: 30);
25                         Route route =
26                             MaterialPageRoute(builder: (context) => SecondHome(user: user));
27                         Navigator.push(context, route);
28                     },
29                     child: Text('Second Home'),
30                 ), // RaisedButton
31             ) // Center
32     }
33 }
```

Example: 2

```
36 < class SecondHome extends StatelessWidget {  
37   final User user;  
38  
39   SecondHome({this.user});  
40  
41   @override  
42   Widget build(BuildContext context) {  
43     return Scaffold(  
44       appBar: AppBar(  
45         title: Text('${this.user.name} - ${this.user.age}'),  
46       ), // AppBar  
47       body: Center(  
48         child: RaisedButton(  
49           onPressed: () {  
50             Navigator.pop(context);  
51           },  
52             child: Text('Go Back'),  
53           ), // RaisedButton  
54           ), // Center  
55       ); // Scaffold  
56     }  
57   }
```

Example:3 ToDo List

The image displays two screenshots of a Flutter application for a 'ToDo List'.

Left Screenshot (Todos List):

- Header: 'Todos' (in blue bar)
- Items: Todo 7, Todo 8, Todo 9, Todo 10, Todo 11, Todo 12, Todo 13, Todo 14, Todo 15
- Bottom Bar: 'no issues'
- Top Right Corner: 'DEBUG'

Right Screenshot (Todo 15 Details):

- Header: 'Todo 15' (in blue bar) with a back arrow icon
- Text: 'A description of what needs to be done for Todo 15'
- Bottom Bar: 'no issues'
- Top Right Corner: 'DEBUG'

Example:3 ToDo List

```
1 import 'package:flutter/foundation.dart';
2 import 'package:flutter/material.dart';
3
4 class Todo {
5     final String title;
6     final String description;
7
8     Todo(this.title, this.description);
9 }
10
11 void main() {
12     runApp(MaterialApp(
13         title: 'Passing Data',
14         home: TodosScreen(
15             todos: List.generate(
16                 20,
17                 (i) => Todo(
18                     'Todo $i',
19                     'A description of what needs to be done for Todo $i',
20                 ), // Todo
21             ), // List.generate
22         ), // TodosScreen
23     )); // MaterialApp
24 }
```

Example:3 ToDo List

```
26  <code>class TodosScreen extends StatelessWidget {</code>
27      <code>final List<Todo> todos;</code>
28
29      TodosScreen({Key key, @required this.todos}) : super(key: key);
30
31      @override
32      Widget build(BuildContext context) {
33          return Scaffold(
34              appBar: AppBar(
35                  title: Text('Todos'),
36              ), // AppBar
37              body: ListView.builder(
38                  itemCount: todos.length,
39                  itemBuilder: (context, index) {
40                      return ListTile(
41                          title: Text(todos[index].title),
42                          onTap: () {
43                              Navigator.push(
44                                  context,
45                                  MaterialPageRoute(
46                                      builder: (context) => DetailScreen(todo: todos[index]),
47                                  ), // MaterialPageRoute
48                          );
49                      },
50                  ); // ListTile
51                  },
52              ), // ListView.builder
53          ); // Scaffold
54      }
55  }</code>
```

Example:3 ToDo List

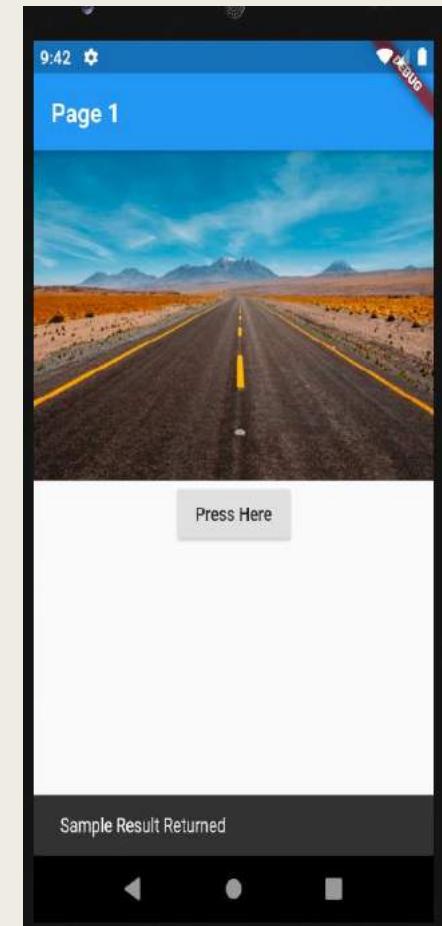
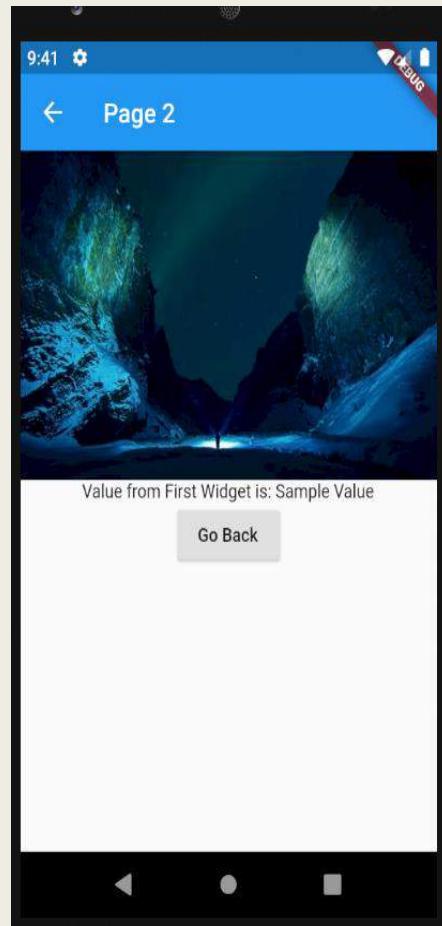
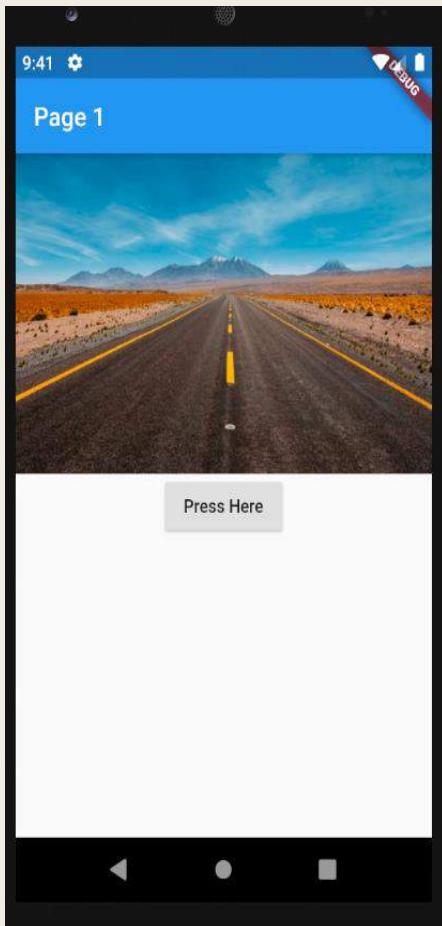
```
57   class DetailScreen extends StatelessWidget {
58     // Declare a field that holds the Todo.
59     final Todo todo;
60
61     // In the constructor, require a Todo.
62     DetailScreen({Key key, @required this.todo}) : super(key: key);
63
64     @override
65     Widget build(BuildContext context) {
66       // Use the Todo to create the UI.
67       return Scaffold(
68         appBar: AppBar(
69           title: Text(todo.title),
70         ), // AppBar
71         body: Padding(
72           padding: EdgeInsets.all(16.0),
73           child: Text(todo.description),
74         ), // Padding
75       ); // Scaffold
76     }
77 }
```

Direct Navigation (Receive Data across Screens)

Receive Values from Second Screen

- Once the data is sent to the Second Screen, there can be ways to return another value from the Second Widget. This is possible by making use of **the await call on the Navigator.push()** method in the First Widget from which we pass a data.
 1. Create two screen
 2. Define the routes
 3. Push the content and the Second Widget to the Navigator class
 4. Wait for a response from the Second Widget using the **await** on the Navigator.push()
 5. Pop the Widget from the Navigator by passing a value to the Pop() method
 6. Print the Value on the first screen.

Receive value from another screen(output)



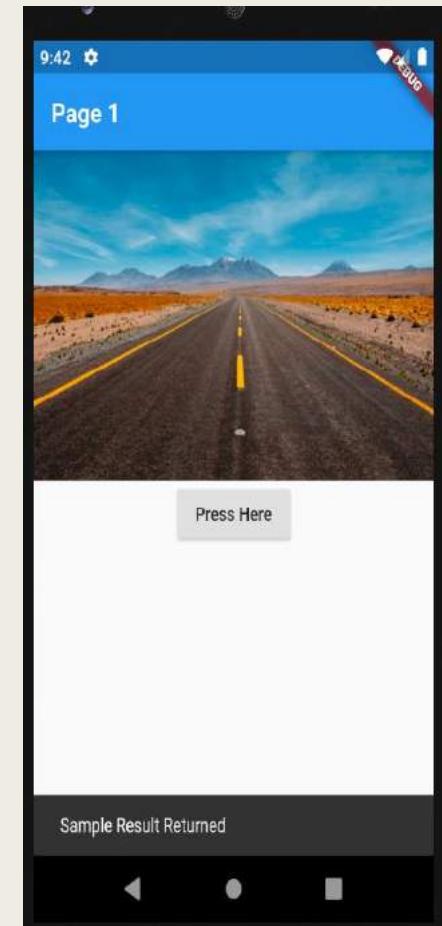
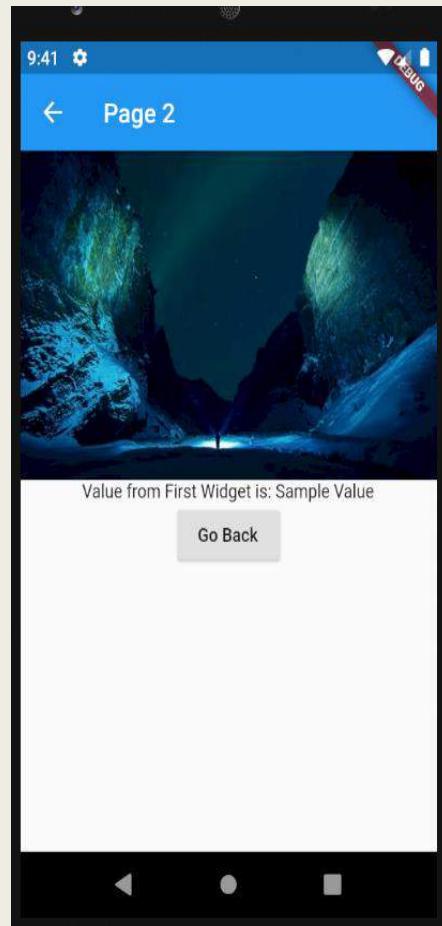
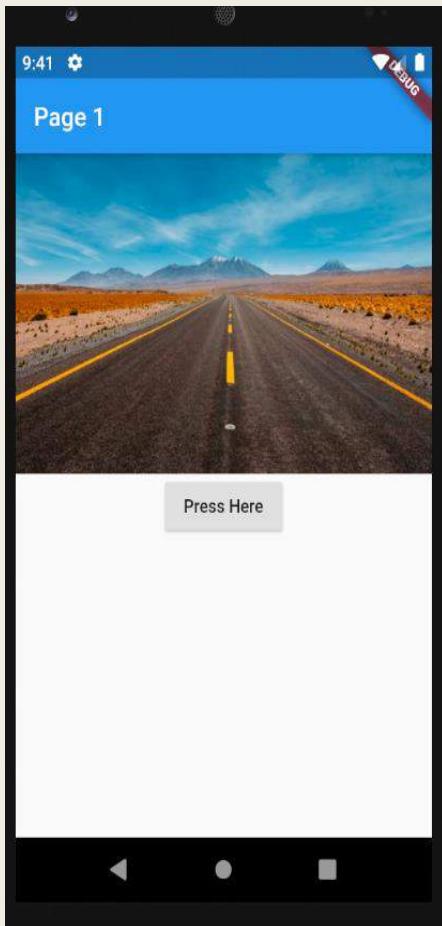
Receive value from another screen(1)

```
8  class FirstWidget extends StatelessWidget
9  { String result;
10  @override
11  Widget build(BuildContext context) {
12    return Scaffold(
13      appBar: AppBar(title: Text("Page 1"),),
14      body: Builder(builder: (BuildContext context){
15
16        return Column(children: <Widget>[
17          Image.asset("assets/images/image1.jpg"),
18          RaisedButton(child: Text("Press Here"),
19                      onPressed: (){
20                        displayValue(context);
21                      }
22                    ), // RaisedButton
23      ],); // <Widget>[] // Column
24    }) // Builder
25
26  ); // Scaffold
27
28 }
29 displayValue(BuildContext context) async [
30   this.result = await Navigator.push(
31     context,
32     MaterialPageRoute(builder: (context) => SecondWidget(value:"Sample Value"))
33   );
34   Scaffold.of(context)
35     ..removeCurrentSnackBar()
36     ..showSnackBar(SnackBar(content: Text("$result")));
37 }
```

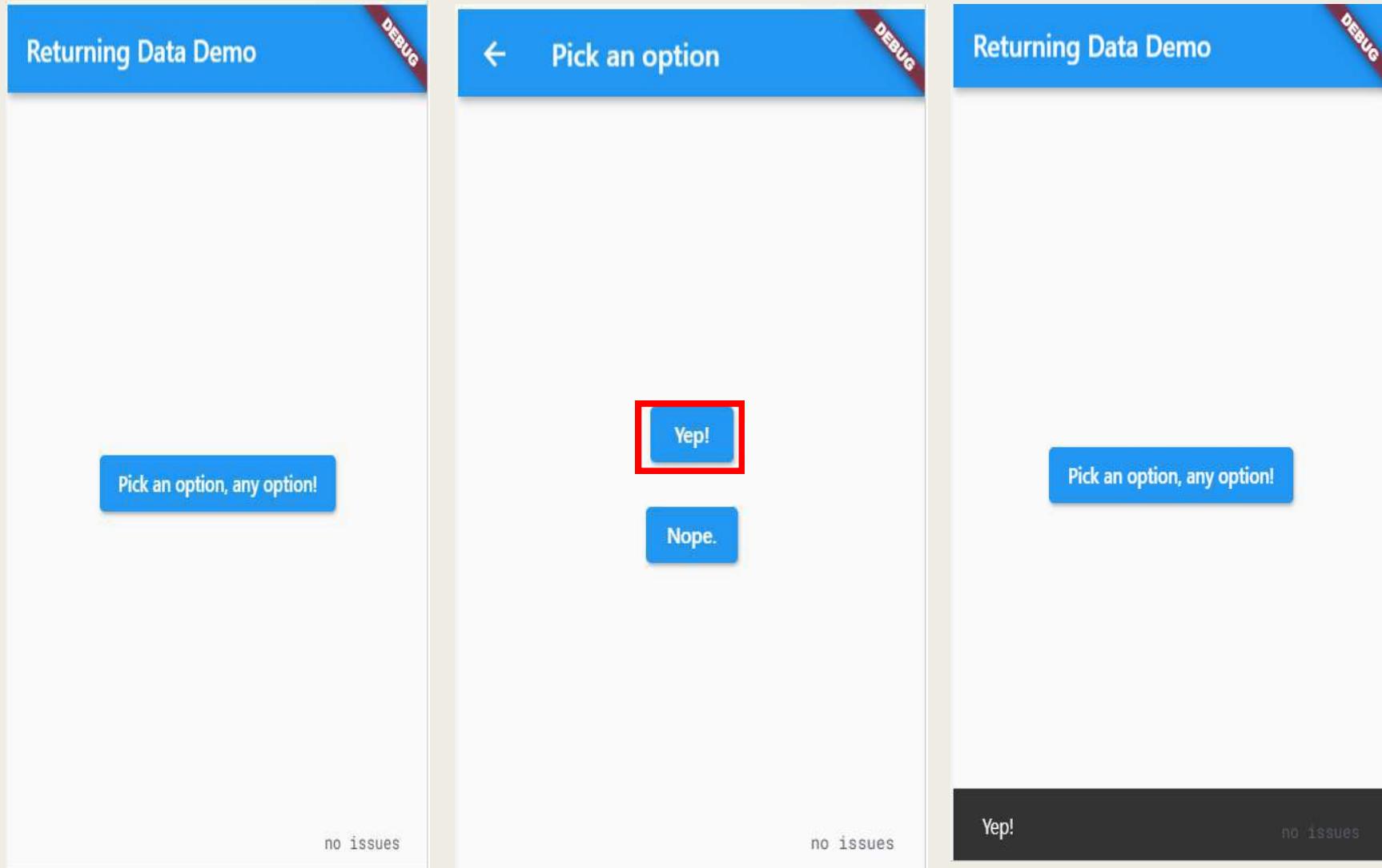
Receive value from another screen(2)

```
39   class SecondWidget extends StatelessWidget
40   {
41     String value;
42     SecondWidget({Key key, @required this.value}):super(key:key);
43     @override
44     Widget build(BuildContext context) {
45       return Scaffold(
46         appBar: AppBar(title: Text("Page 2"),),
47         body: Column(children: <Widget>[
48           Image.asset("assets/images/image2.jpg"),
49           Text("Value from First Widget is: "+this.value),
50           RaisedButton(child: Text("Go Back"),
51             onPressed: () {
52               Navigator.pop(context, "Sample Result Returned");
53             }, // RaisedButton
54           ],) // <Widget>[] // Column
55         ); // Scaffold
56       }
57     }
58 }
```

Receive value from another screen(output)



Example:2



Example:2(1)

```
1 import 'package:flutter/material.dart';
Run | Debug
2 void main() {
3   runApp(MaterialApp(
4     title: 'Returning Data',
5     home: HomeScreen(),
6   )); // MaterialApp
7
8 class HomeScreen extends StatelessWidget {
9   @override
10  Widget build(BuildContext context) {
11    return Scaffold(
12      appBar: AppBar(
13        title: Text('Returning Data Demo'),
14      ), // AppBar
15      body: Center(child: SelectionButton()),
16    ); // Scaffold
17  }
18 }
```

Example:2(2)

```
20   class SelectionButton extends StatelessWidget {  
21     @override  
22     Widget build(BuildContext context) {  
23       return ElevatedButton(  
24         onPressed: () {  
25           _navigateAndDisplaySelection(context);  
26         },  
27         child: Text('Pick an option, any option!'),  
28       ); // ElevatedButton  
29     }  
30     _navigateAndDisplaySelection(BuildContext context) async {  
31       final result = await Navigator.push(  
32         context,  
33         MaterialPageRoute(builder: (context) => SelectionScreen()),  
34       );  
35       Scaffold.of(context)  
36         ..removeCurrentSnackBar()  
37         ..showSnackBar(SnackBar(content: Text("$result")));  
38     }  
39   }
```

Example:2(3)

```
41   class SelectionScreen extends StatelessWidget {
42     @override
43     Widget build(BuildContext context) {
44       return Scaffold(
45         appBar: AppBar(
46           title: Text('Pick an option'),
47         ), // AppBar
48         body: Center(
49           child: Column(
50             mainAxisAlignment: MainAxisAlignment.center,
51             children: <Widget>[
52               Padding(
53                 padding: const EdgeInsets.all(8.0),
54                 child: ElevatedButton(
55                   onPressed: () {
56                     // Close the screen and return "Yep!" as the result.
57                     Navigator.pop(context, 'Yep!');
```

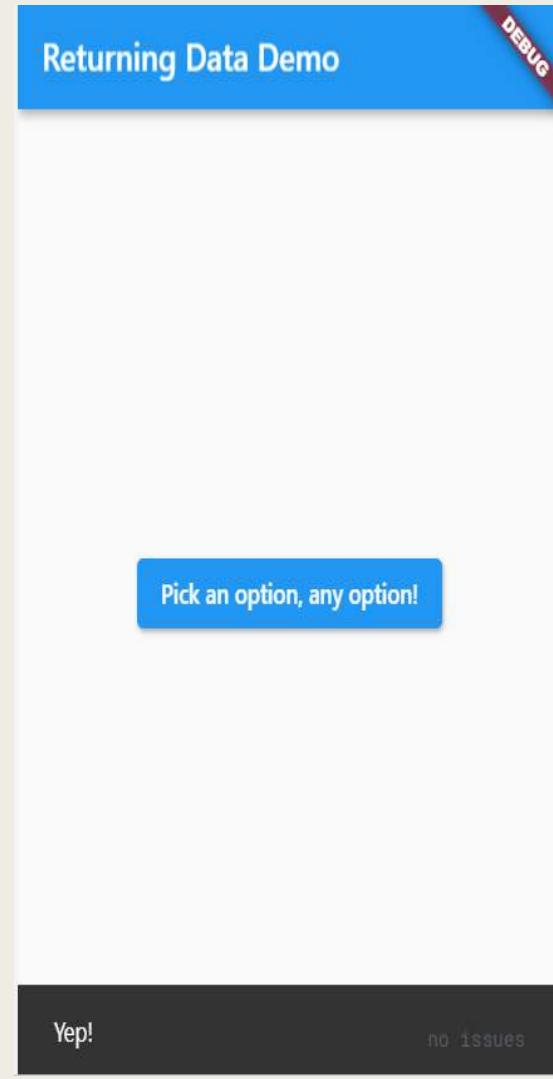
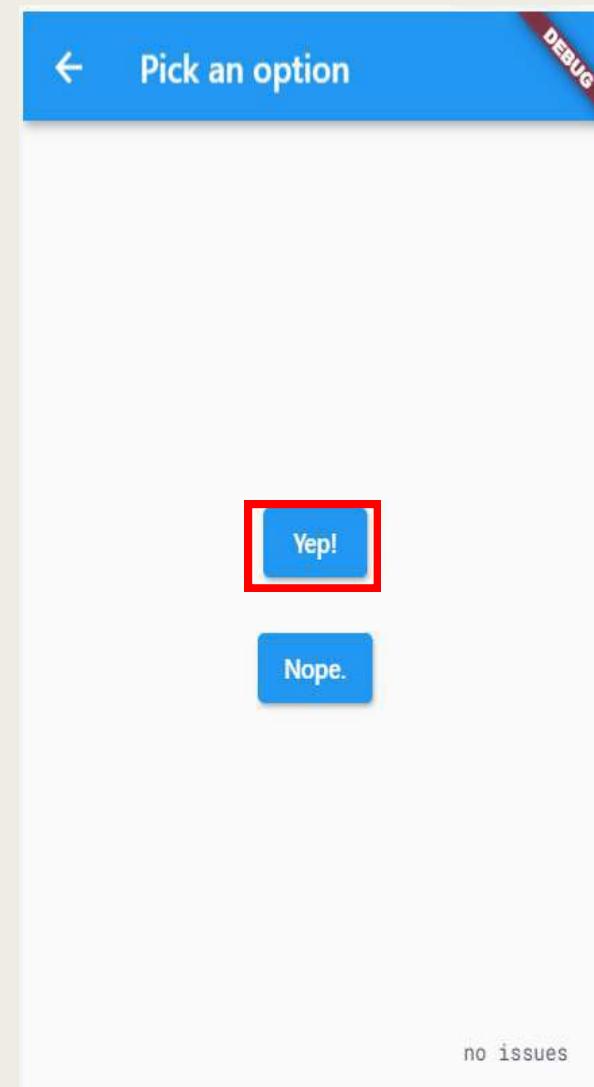
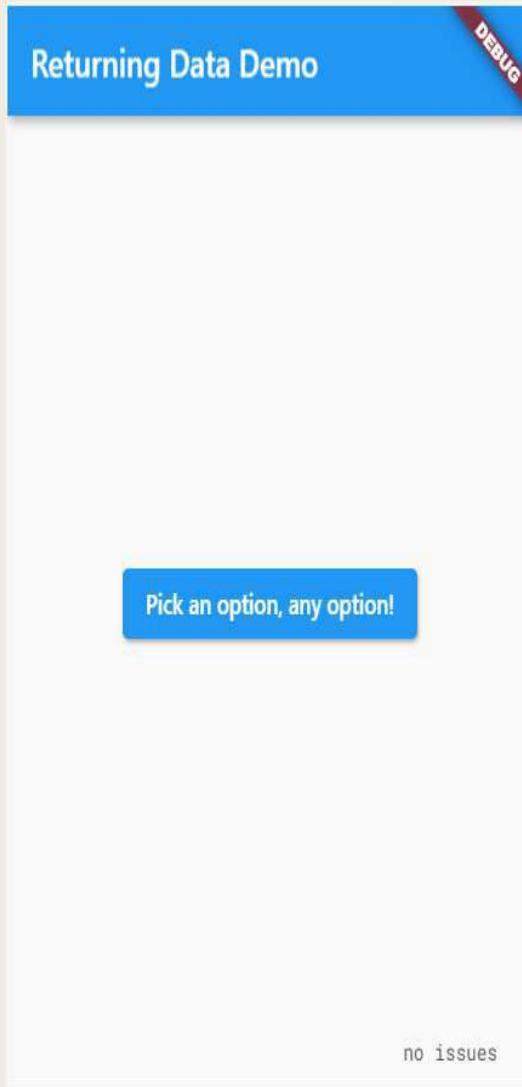
Navigator.pop(context, 'Yep!');

```
58                   },
59                   child: Text('Yep!'),
60                 ), // ElevatedButton
61               ), // Padding
62               Padding(
63                 padding: const EdgeInsets.all(8.0),
64                 child: ElevatedButton(
65                   onPressed: () {
66                     // Close the screen and return "Nope!" as the result.
67                     Navigator.pop(context, 'Nope.');
```

Navigator.pop(context, 'Nope.');

```
68                   },
69                   child: Text('Nope.'),
70                 ), // ElevatedButton
```

Example:2(Output)



Limitations: Direct Navigation

- This approach contributes to boilerplate code which multiplies with growing screens/pages.
- It is very hard to keep track of logic around these routes since its spread out in multiple classes.

2. Static Navigation

- The static navigation is implemented by assigning a Map of routes (pages/screens) to MaterialApp's routes property.
- The route name is pushed using Navigator.pushNamed(...).

This is known as **Named Routing** because each page is given a unique name, which is pushed on Navigator widget.

- The MaterialApp and StatelessWidget provides the routes property.

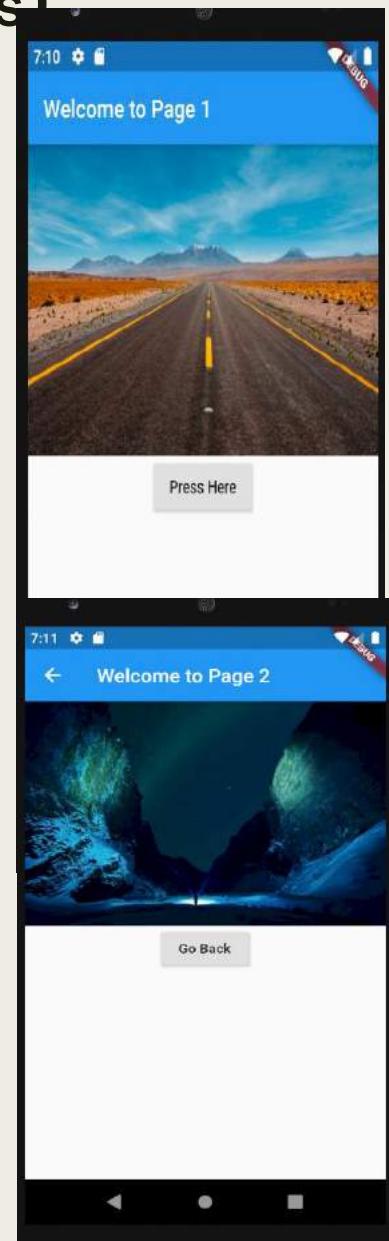
This property allows to specify routes in Map<String, WidgetBuilder>. This option is great when there is no logic around the routes.
- For example, authentication or verification before you show the page. Only the data available global to app can be passed on to the second page.

2. Static Navigation Example

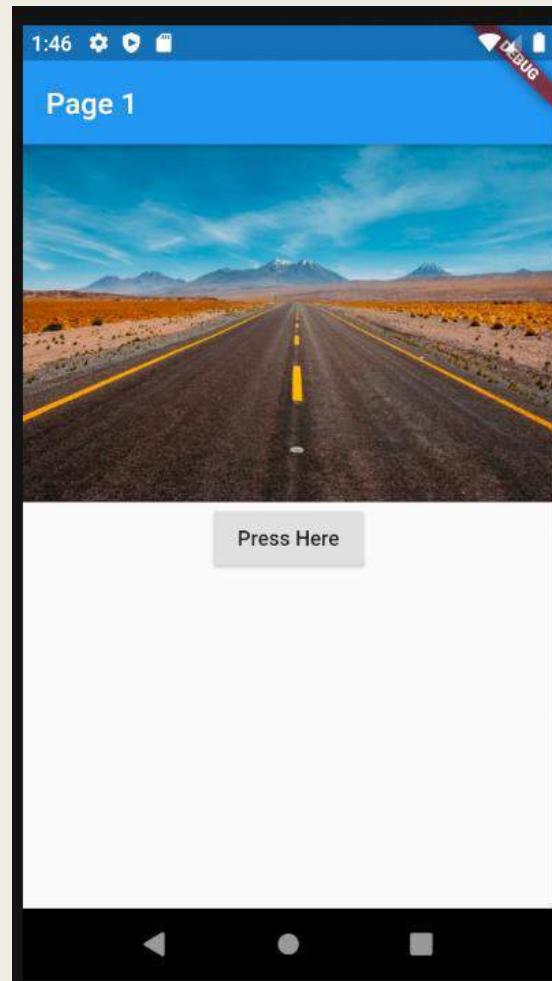
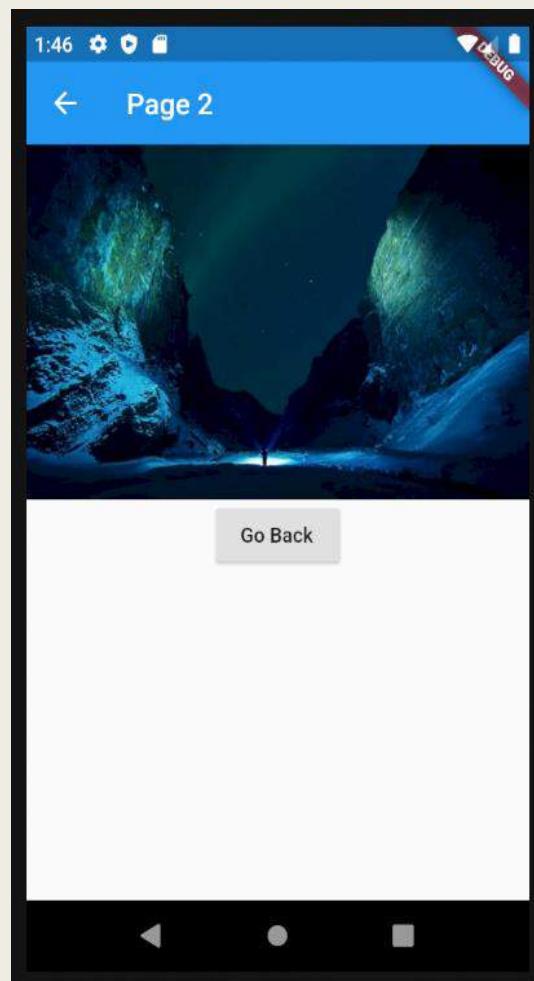
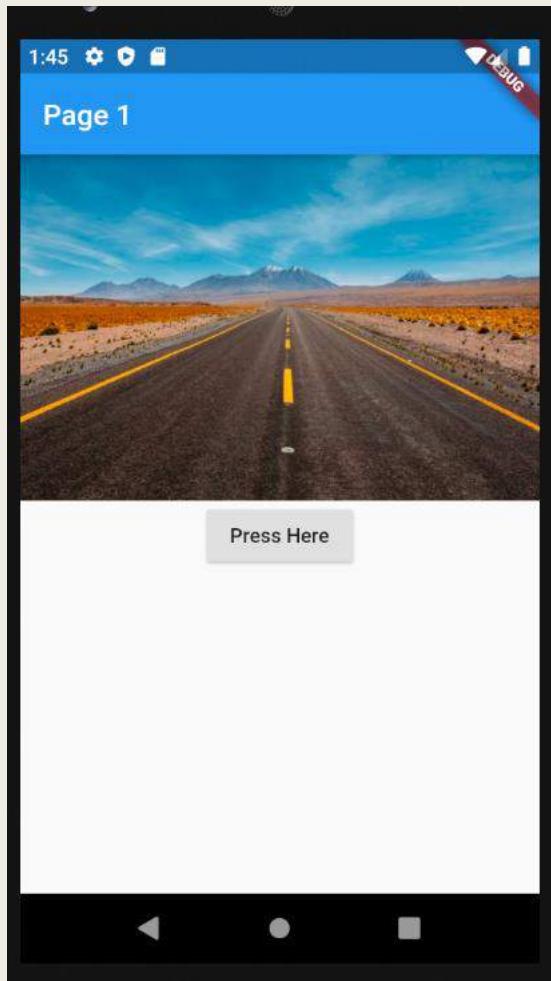
(Creating Named Routed to Navigate between Widgets)

1. Create two screens
2. Define the routes
3. Navigate to the second screen using `Navigator.pushNamed`
4. Return to the first screen using `Navigator.pop`

```
runApp(MaterialApp(  
    title:'Androidmonks',  
    routes:{  
        '/':(context)=>FirstWidget(),  
        '/second':(context)=>SecondWidget()  
    },  
    initialRoute: '/'  
));
```



Example



```
1 import 'package:flutter/material.dart';
Run | Debug
2 void main()
3 {
4     runApp(MaterialApp(
5         title: 'DEMO',
6         initialRoute: '/',
7         routes: {
8             '/':(context) => FirstWidget(),
9             '/second': (context) => SecondWidget()
10        },
11    )); // MaterialApp
12 }
13 class FirstWidget extends StatelessWidget
14 {
15     @override
16     Widget build(BuildContext context) {
17         return Scaffold(
18             appBar: AppBar(title: Text("Page 1"),),
19             body: Column(children: <Widget>[
20                 Image.asset('assets/images/image1.jpg'),
21                 RaisedButton(child: Text("Press Here"),
22                 onPressed: () {
23                     Navigator.pushNamed(context, '/second');
24                 },
25             ) // RaisedButton
26         ],) // <Widget>[] // Column
27     );} // Scaffold
28 }
29 }
```

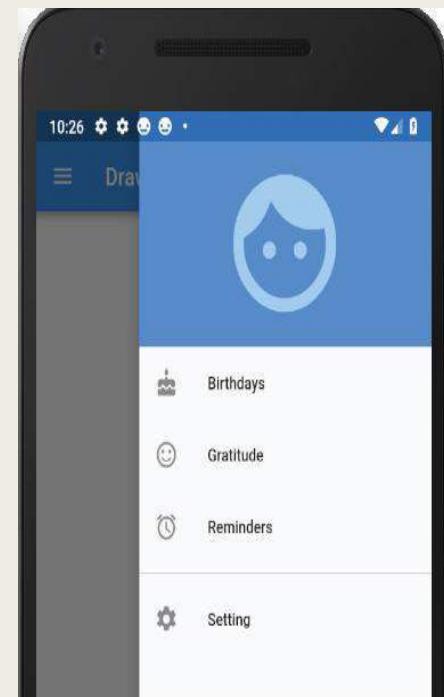
```
30     class SecondWidget extends StatelessWidget
31     {
32         @override
33         Widget build(BuildContext context) {
34             return Scaffold(
35                 appBar: AppBar(title: Text("Page 2"),),
36                 body: Column(children: <Widget>[
37                     Image.asset('assets/images/image2.jpg'),
38                     RaisedButton(child: Text("Go Back"),
39                         onPressed: () {
40                             Navigator.pop(context);
41                         }), // RaisedButton
42                     ],), // <Widget>[] // Column
43             ); // Scaffold
44         }
45     }
```

Drawer Application

DrawerHeader Types:

- DrawerHeader is intended to display generic or custom information by setting the padding, child, decoration, and other properties.

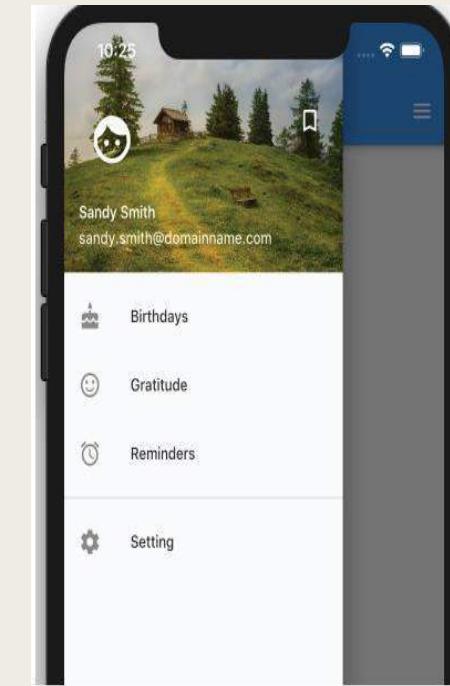
```
// Generic or custom information  
DrawerHeader(  
    padding: EdgeInsets.zero,  
    child: Icon(Icons.face),  
    decoration: BoxDecoration(color: Colors.blue),  
)
```



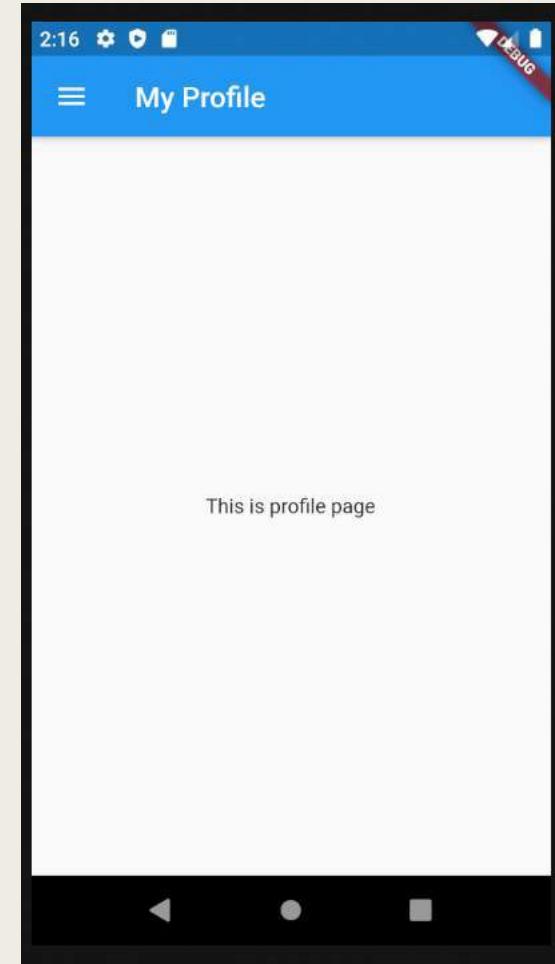
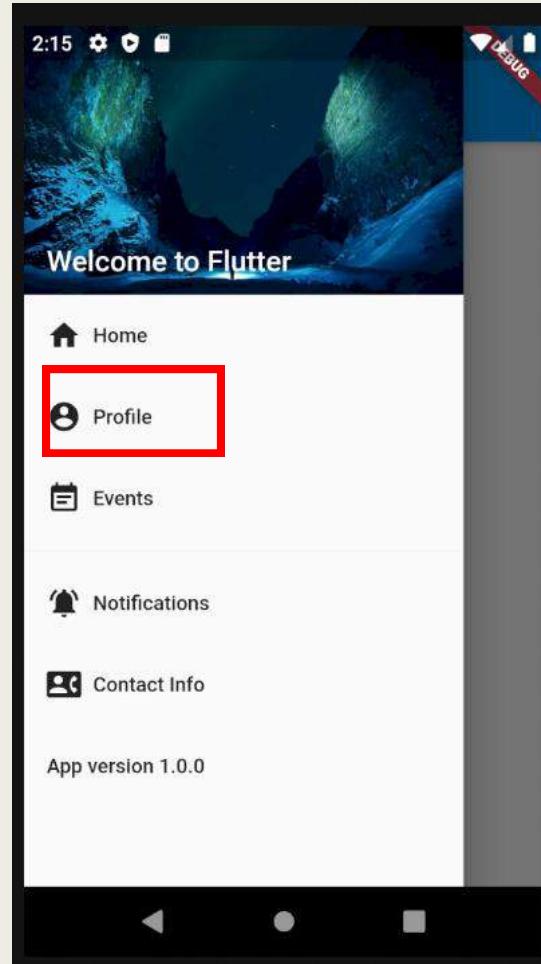
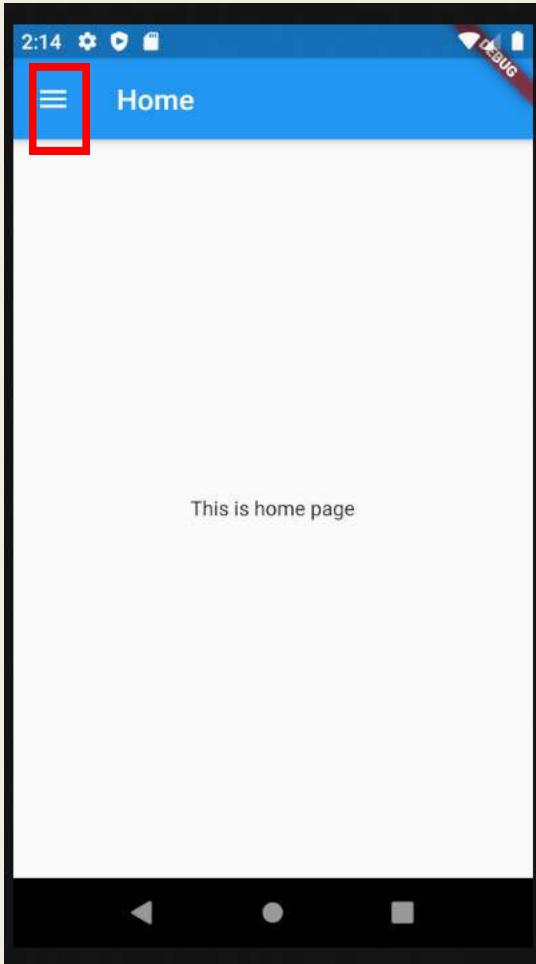
UserAccountsDrawerHeader

- The UserAccountsDrawerHeader is intended to display:
- the app's user details : currentAccountPicture, accountName, accountEmail, otherAccountsPictures, and decoration properties.

```
// User details
UserAccountsDrawerHeader(
    currentAccountPicture: Icon(Icons.face,),
    accountName: Text('Sandy Smith'),
    accountEmail: Text('sandy.smith@domainname.com'),
    otherAccountsPictures: <Widget>[
        Icon(Icons.bookmark_border),
    ],
    decoration: BoxDecoration(
        image: DecorationImage(
            image: AssetImage('assets/images/home_top_mountain.jpg'),
            fit: BoxFit.cover,
        ),
    ),
),
```



Drawer Application using Navigation



1. Drawer Widget: Create Drawer Header

```
1 import 'package:flutter/material.dart';
2
3 Widget createDrawerHeader() {
4     return DrawerHeader(
5         margin: EdgeInsets.zero,
6         padding: EdgeInsets.zero,
7         decoration: BoxDecoration(
8             image: DecorationImage(
9                 fit: BoxFit.cover,
10                image: AssetImage('assets/images/image2.jpg'))), // DecorationImage // BoxDecoration
11     child: Stack(children: <Widget>[
12         Positioned(
13             bottom: 12.0,
14             left: 16.0,
15             child: Text("Welcome to Flutter",
16                 style: TextStyle(
17                     color: Colors.white,
18                     fontSize: 20.0,
19                     fontWeight: FontWeight.w500))), // TextStyle // Text // Positioned
20     ]); // <Widget>[] // Stack // DrawerHeader
21 }
```

2. Drawer Widget: Create Drawer Body

```
1 import 'package:flutter/material.dart';
2
3 Widget createDrawerBodyItem(
4     {IconData icon, String text, GestureTapCallback onTap}) {
5     return ListTile(
6         title: Row(
7             children: <Widget>[
8                 Icon(icon),
9                 Padding(
10                     padding: EdgeInsets.only(left: 8.0),
11                     child: Text(text),
12                 ) // Padding
13             ], // <Widget>[]
14         ), // Row
15         onTap: onTap,
16     ); // ListTile
17 }
```

3. Drawer Widget: Navigation Drawer

```
lib > navigationDrawer > navigationDrawer.dart > ...
1  import 'package:flutter/material.dart';
2  import 'package:app_drawer_widget/widgets/createDrawerBodyItem.dart';
3  import 'package:app_drawer_widget/widgets/createDrawerHeader.dart';
4  import 'package:app_drawer_widget/routes/pageRoute.dart';
5  class navigationDrawer extends StatelessWidget {
6      @override
7      Widget build(BuildContext context) {
8          return Drawer(
9              child: ListView(
10                 padding: EdgeInsets.zero,
11                 children: <Widget>[
12                     createDrawerHeader(),
13                     createDrawerBodyItem(
14                         icon: Icons.home,
15                         text: 'Home',
16                         onTap: () =>
17                             Navigator.pushReplacementNamed(context, pageRoute.home),
18                     ),
19                     createDrawerBodyItem(
20                         icon: Icons.account_circle,
21                         text: 'Profile',
22                         onTap: () =>
23                             Navigator.pushReplacementNamed(context, pageRoute.profile),
24                     ),
25                 ],
26             );
27         }
28     }
29 }
```

4. Drawer Widget: Create Navigation Pages

```
1 import 'package:flutter/material.dart';
2 import 'package:app_drawer_widget/navigationDrawer/navigationDrawer.dart';
3 class homePage extends StatelessWidget {
4     static const String routeName = '/homePage';
5
6     @override
7     Widget build(BuildContext context) {
8         return new Scaffold(
9             appBar: AppBar(
10                 title: Text("Home"),
11             ), // AppBar
12             drawer: navigationDrawer(),
13             body: Center(child: Text("This is home page"))); // Scaffold
14     }
15 }
```

```
1 import 'package:flutter/material.dart';
2 import 'package:app_drawer_widget/navigationDrawer/navigationDrawer.dart';
3 class profilePage extends StatelessWidget {
4     static const String routeName = '/profilePage';
5
6     @override
7     Widget build(BuildContext context) {
8         return new Scaffold(
9             appBar: AppBar(
10                 title: Text("My Profile"),
11             ), // AppBar
12             drawer: navigationDrawer(),
13             body: Center(child: Text("This is profile page"))); // Scaffold
14     }
15 }
```

4. Drawer Widget: Create Navigation Pages

```
1 import 'package:flutter/material.dart';
2 import 'package:app_drawer_widget/navigationDrawer/navigationDrawer.dart';
3
4 class contactPage extends StatelessWidget {
5   static const String routeName = '/contactPage';
6
7   @override
8   Widget build(BuildContext context) {
9     return new Scaffold(
10       appBar: AppBar(
11         title: Text("Contacts"),
12       ), // AppBar
13       drawer: navigationDrawer(),
14       body: Center(child: Text("This is contacts page")));
15     }
16 }
```

```
1 import 'package:flutter/material.dart';
2 import 'package:app_drawer_widget/navigationDrawer/navigationDrawer.dart';
3
4 class eventPage extends StatelessWidget {
5   static const String routeName = '/eventPage';
6
7   @override
8   Widget build(BuildContext context) {
9     return new Scaffold(
10       appBar: AppBar(
11         title: Text("Events"),
12       ), // AppBar
13       drawer: navigationDrawer(),
14       body: Center(child: Text("Hey! this is events list page")));
15     }
16 }
```

5. Drawer Widget: Create Navigation Route

```
1 import 'package:app_drawer_widget/fragments/contactPage.dart';
2 import 'package:app_drawer_widget/fragments/homePage.dart';
3 import 'package:app_drawer_widget/fragments/eventPage.dart';
4 import 'package:app_drawer_widget/fragments/notificationPage.dart';
5 import 'package:app_drawer_widget/fragments/profilePage.dart';
6
7 class pageRoute {
8     static const String home = HomePage.routeName;
9     static const String contact = ContactPage.routeName;
10    static const String event = EventPage.routeName;
11    static const String profile = ProfilePage.routeName;
12    static const String notification = NotificationPage.routeName;
13 }
```

6. Drawer Widget: Main.dart

```
1  ✓ import 'package:flutter/material.dart';
2  import 'package:app_drawer_widget/fragments/contactPage.dart';
3  import 'package:app_drawer_widget/fragments/eventPage.dart';
4  import 'package:app_drawer_widget/fragments/homePage.dart';
5  import 'package:app_drawer_widget/fragments/notificationPage.dart';
6  import 'package:app_drawer_widget/fragments/profilePage.dart';
7  import 'package:app_drawer_widget/routes/pageRoute.dart';
Run | Debug
8  void main() => runApp(new MyApp());
9
10 ✓ class MyApp extends StatelessWidget {
11     // This widget is the root of your application.
12     @override
13     Widget build(BuildContext context) {
14         return new MaterialApp(
15             title: 'NavigationDrawer Demo',
16             theme: new ThemeData(
17                 primarySwatch: Colors.blue,
18             ), // ThemeData
19             home: HomePage(),
20             routes: {
21                 pageRoute.home: (context) => HomePage(),
22                 pageRoute.contact: (context) => contactPage(),
23                 pageRoute.event: (context) => eventPage(),
24                 pageRoute.profile: (context) => profilePage(),
25                 pageRoute.notification: (context) => notificationPage(),
26             },
27         ), // MaterialApp
28     }
29 }
```

Limitations: Static Navigation

- You cannot pass a value inside of a route
- The possibility to pass arguments to the route will be handled by the **dynamic navigation**

Limitations: Static Navigation

- You cannot pass a value inside of a route
- The possibility to pass arguments to the route will be handled by the **dynamic navigation**
- This option is great when there is no logic around the routes.
 - For example, authentication or verification before you show the page is not required.
- Only the data available global to app can be passed on to the second page.

3. Dynamic Navigation

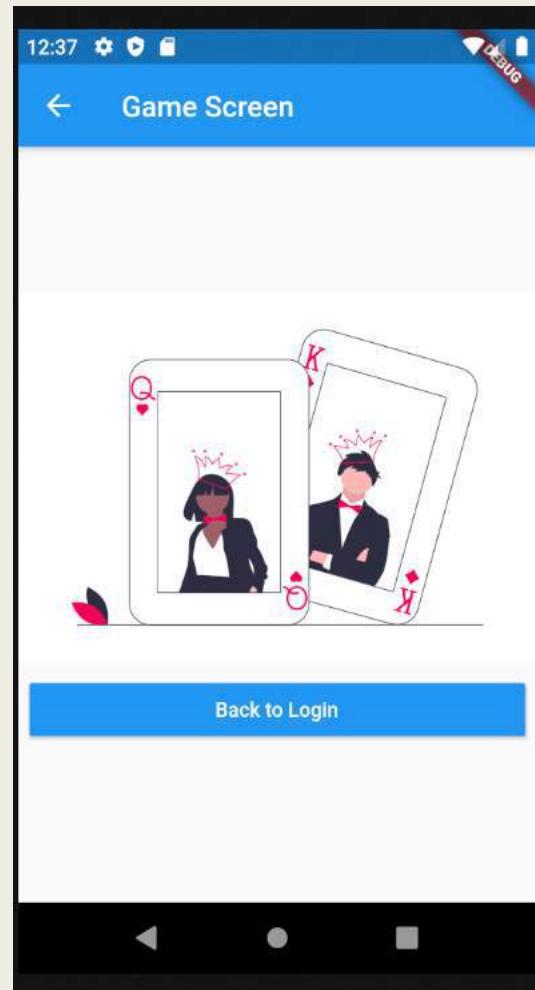
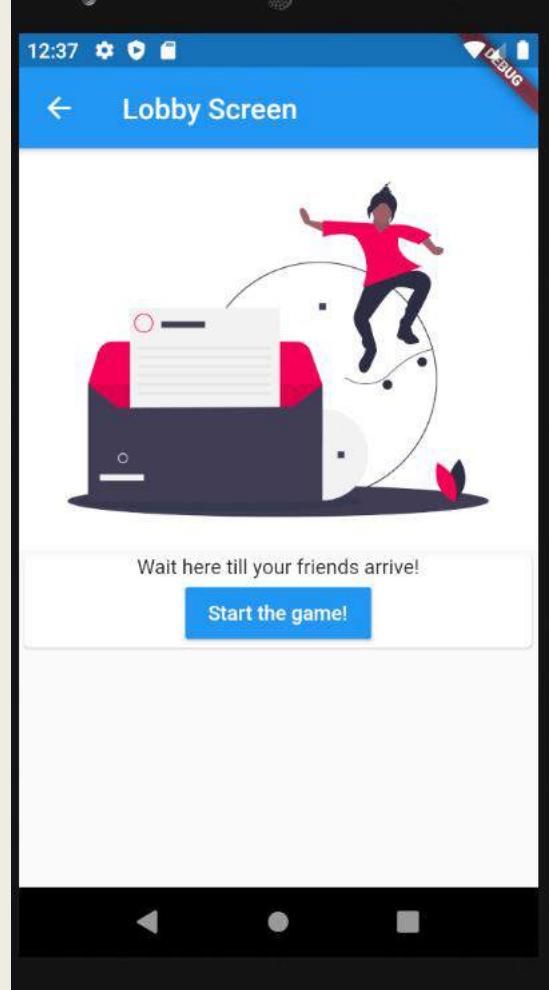
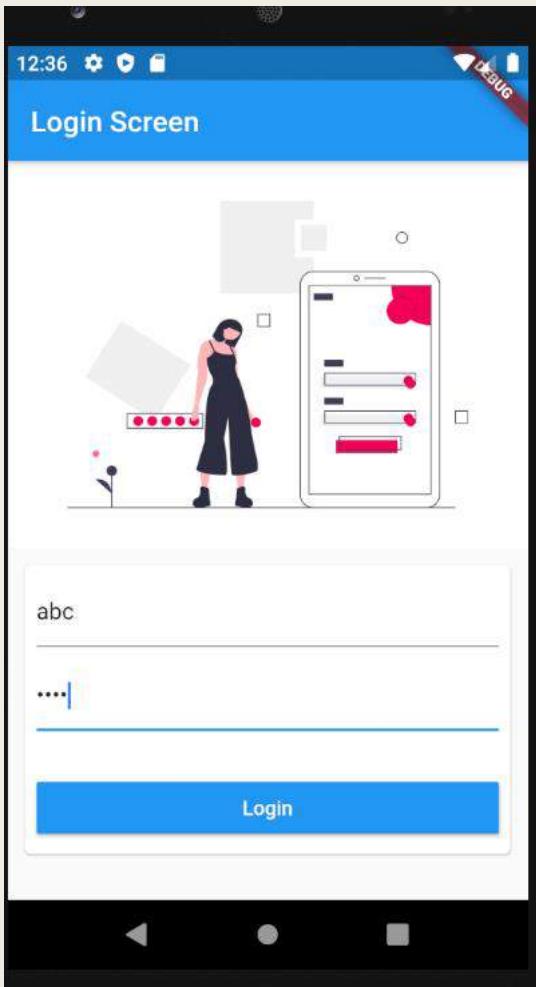
- In dynamic navigation, routes are generated by implementing `onGenerateRoute` callback in the `MaterialApp` class.
- This is a type of Named Routing that makes use of `onGenerateRoute` property.
- The `MaterialApp` and `WidgetApp` provides the `onGenerateRoute` property. It lets you specify a function, say `generateRoute` returning a route.

It allows the data pass using `RouteSettings`. It carries the data that might be useful in constructing a Route.

- Any verification logic can be easily be applied before showing the target page.

There's always an option to default "not found" page when route or match is not found.

Example(1)



Example(1)

```
main.dart •  
lib > main.dart > ...  
1 import 'package:flutter/material.dart';  
2 import 'package:dynaminc_navigation_00/routes.dart';  
3  
Run | Debug  
4 void main() => runApp(MyApp());  
5  
6 class MyApp extends StatelessWidget {  
7     // This widget is the root of your application.  
8     @override  
9     Widget build(BuildContext context) {  
10         return MaterialApp(  
11             onGenerateRoute: generateRoute,  
12             initialRoute: 'LoginScreen',  
13             title: 'Flutter Demo',  
14             theme: ThemeData.light(),  
15         ); // MaterialApp  
16     }  
17 }
```

Example(1)

```
routes.dart X

lib > routes.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package
```

Example(1) Login.dart

```
lib > screens > LoginScreen.dart > build
1 import 'package:flutter/material.dart';
2 import 'LobbyScreen.dart';
3 class LoginScreen extends StatelessWidget {
4     @override
5     Widget build(BuildContext context) {
6         return Scaffold(
7             appBar: AppBar(
8                 title: Text("Login Screen"),
9             ), // AppBar
10            body: Container(
11                child: Column(
12                    children: <Widget>[
13                        Flexible(child: Image.asset("assets/images/mobile_login.png")),
14                        Padding(
15                            padding: const EdgeInsets.all(8.0),
16                            child: Card(
17                                child: Padding(
18                                    padding: const EdgeInsets.all(8.0),
19                                    child: Column(
20                                        children: <Widget>[
21                                            TextField(
22                                                decoration: InputDecoration(hintText: "Username"),
23                                            ), // TextField
24                                            SizedBox(height: 10),
25                                            TextField(
26                                                decoration: InputDecoration(hintText: "Password"),
27                                            ), // TextField
28                                            SizedBox(height: 30),
29                                        ],
30                                    ),
31                                ),
32                            ),
33                        ),
34                    ],
35                ),
36            ),
37        );
38    }
39}
```

Example(1) Login.dart

```
29          Container(
30              width: double.infinity,
31              child: RaisedButton(
32                  child: Text(
33                      "Login",
34                      style: TextStyle(
35                          color: Colors.white,
36                      ), // TextStyle
37                  ), // Text
38                  onPressed: () {
39                      Navigator.push(
40                          context,
41                          MaterialPageRoute(
42                              builder: (BuildContext context) =>
43                                  LobbyScreen(),
44                          ), // MaterialPageRoute
45                      );
46                  },
47                  color: Colors.blue,
48                  ), // RaisedButton
49              ) // Container
50          ], // <Widget>[]
51      ), // Column
52      ), // Padding
53      ), // Card
54      ) // Padding
```

Example(1) Lobby.dart

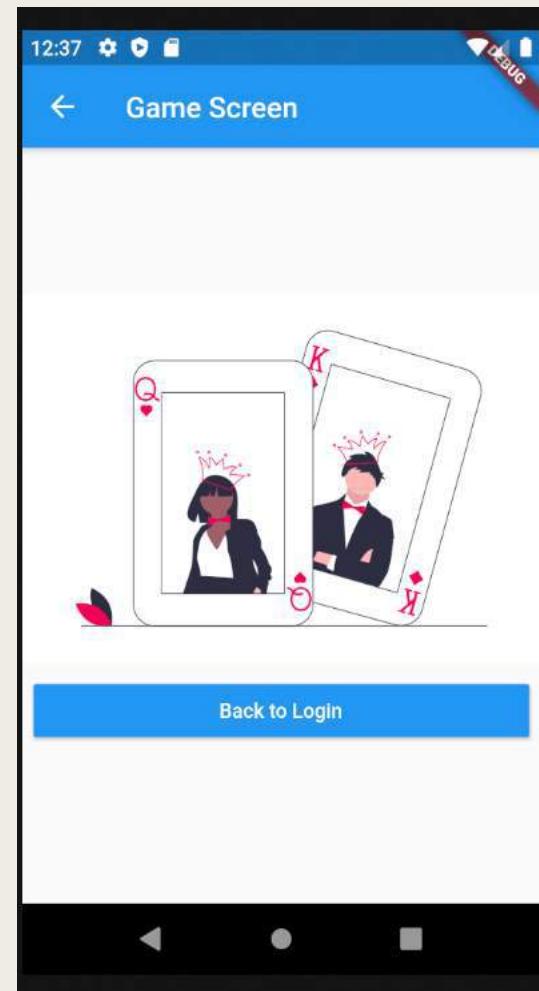
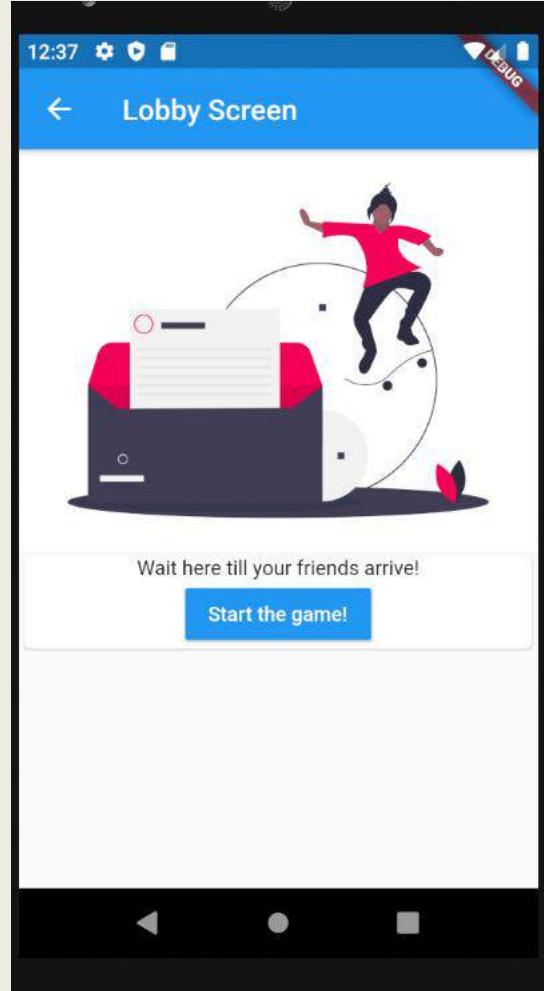
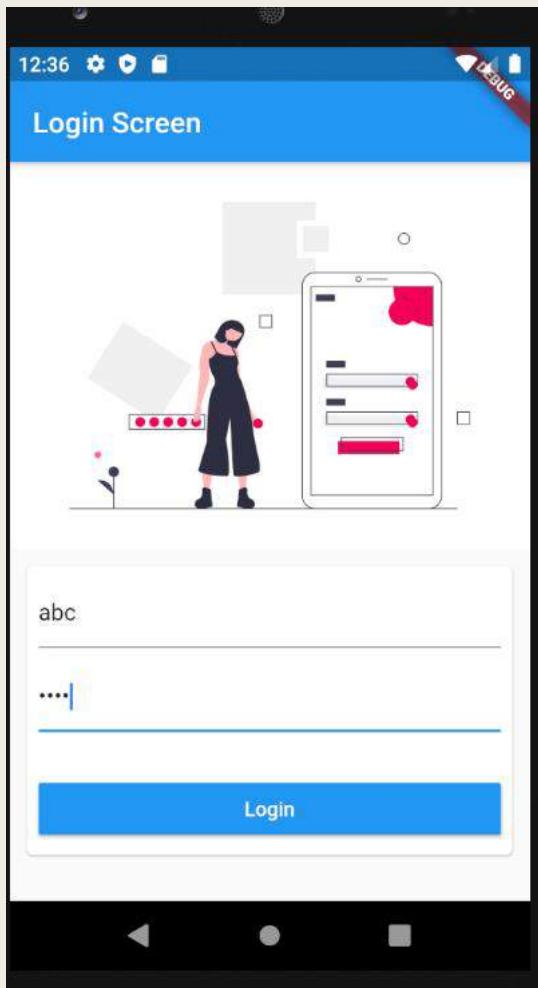
```
lib > screens > LobbyScreen.dart > build
1 import 'package:flutter/material.dart';
2 import 'GameScreen.dart';
3
4 class LobbyScreen extends StatelessWidget {
5     @override
6     Widget build(BuildContext context) {
7         return Scaffold(
8             appBar: AppBar(
9                 title: Text("Lobby Screen"), ), // AppBar
10            body: Column(
11                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
12                children: <Widget>[ Flexible(
13                    child: Image.asset("assets/images/arrived.png"),
14                ), // Flexible
15                Container(
16                    child: Card(
17                        child: Column(
18                            children: <Widget>[
19                                Text("Wait here till your friends arrive!",
20                                    textAlign: TextAlign.center,
21                                ), // Text
22                                RaisedButton(color: Colors.blue,
23                                    onPressed: () {
24                                        Navigator.push(context, MaterialPageRoute(
25                                            builder: (context) => GameScreen()
26                                        )); // MaterialPageRoute
27                                    },
28                                    child: Text("Start the game!",
29                                        style: TextStyle(color: Colors.white),
30                                    ), // Text
31                                ),
32                            ],
33                        ),
34                    ),
35                ),
36            ],
37        );
38    }
39}
```

Example(1) GameScreen.dart

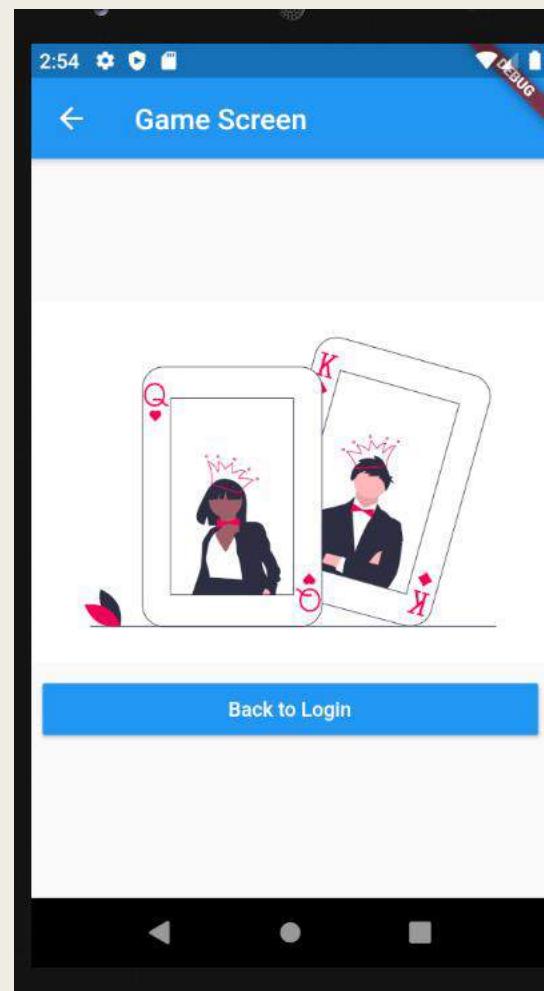
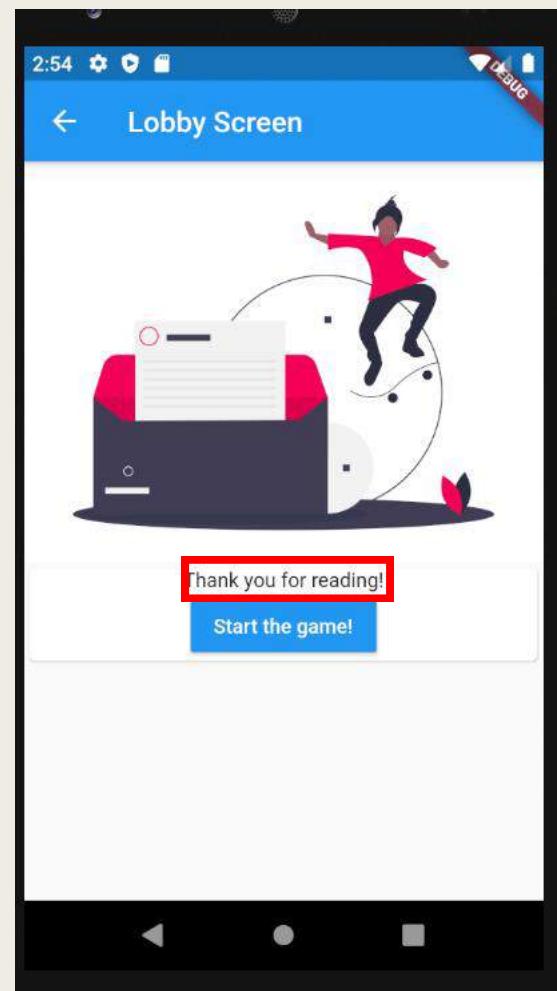
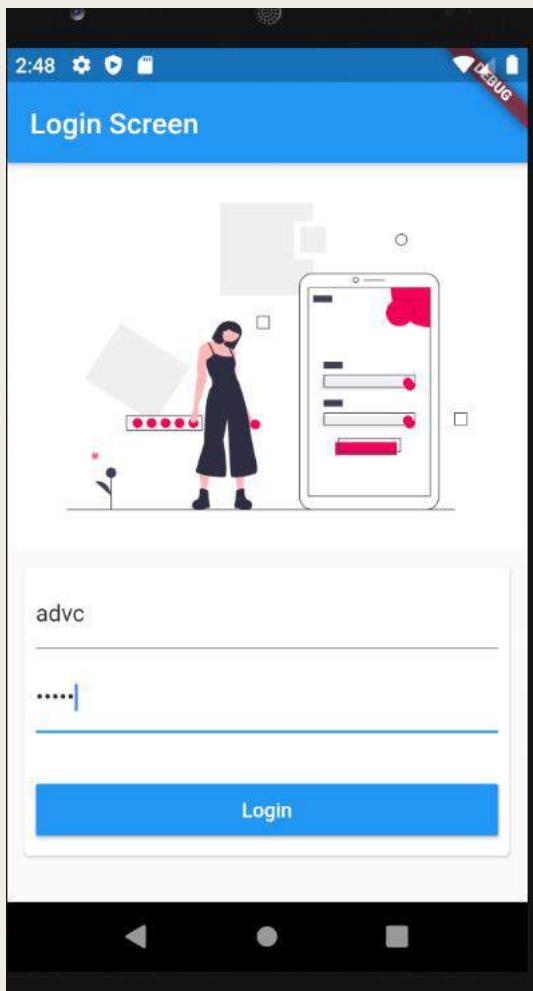
```
GameScreen.dart ×

lib > screens > GameScreen.dart > GameScreen > build
  1 import 'package:flutter/material.dart';
  2
  3 class GameScreen extends StatelessWidget {
  4   @override
  5   Widget build(BuildContext context) {
  6     return Scaffold(
  7       appBar: AppBar(
  8         title: Text("Game Screen"),
  9       ), // AppBar
 10      body: Column(
 11        mainAxisAlignment: MainAxisAlignment.center,
 12        crossAxisAlignment: CrossAxisAlignment.stretch,
 13        children: <Widget>[
 14          Image.asset("assets/images/cards.png"),
 15          Padding(
 16            padding: const EdgeInsets.all(8.0),
 17            child: RaisedButton(
 18              color: Colors.blue,
 19              child: Text(
 20                "Back to Login",
 21                style: TextStyle(color: Colors.white),
 22              ), // Text
 23              onPressed: () {
 24                Navigator.popUntil(
 25                  context,
 26                  (route) => route.isFirst,
 27                );
 28              },
 29            ), // RaisedButton
 30          ) // Padding
 31        ],
 32      ), // Column
 33    ); // Scaffold
 34  } // build
 35
 36  @override
 37  void dispose() {
 38    super.dispose();
 39  }
 40}
 41
```

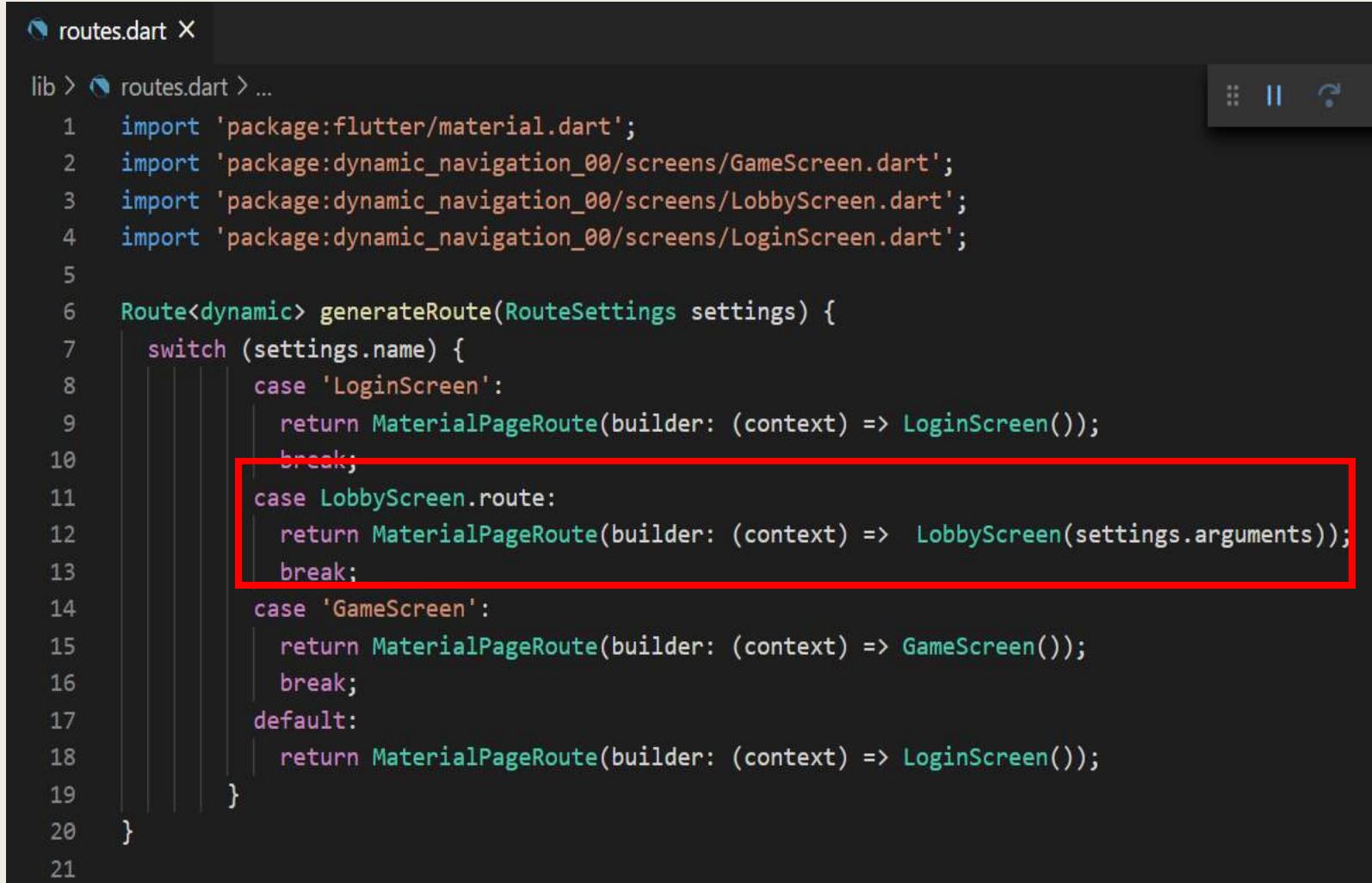
Output



Example: 2 (Pass value from one screen to other)



Example: 2 route.dart



```
routes.dart X

lib > routes.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:dynamic_navigation_00/screens/GameScreen.dart';
3 import 'package:dynamic_navigation_00/screens/LobbyScreen.dart';
4 import 'package:dynamic_navigation_00/screens/LoginScreen.dart';
5
6 Route<dynamic> generateRoute(RouteSettings settings) {
7     switch (settings.name) {
8         case 'LoginScreen':
9             return MaterialPageRoute(builder: (context) => LoginScreen());
10        break;
11        case LobbyScreen.route:
12            return MaterialPageRoute(builder: (context) => LobbyScreen(settings.arguments));
13            break;
14        case 'GameScreen':
15            return MaterialPageRoute(builder: (context) => GameScreen());
16            break;
17        default:
18            return MaterialPageRoute(builder: (context) => LoginScreen());
19    }
20}
21
```

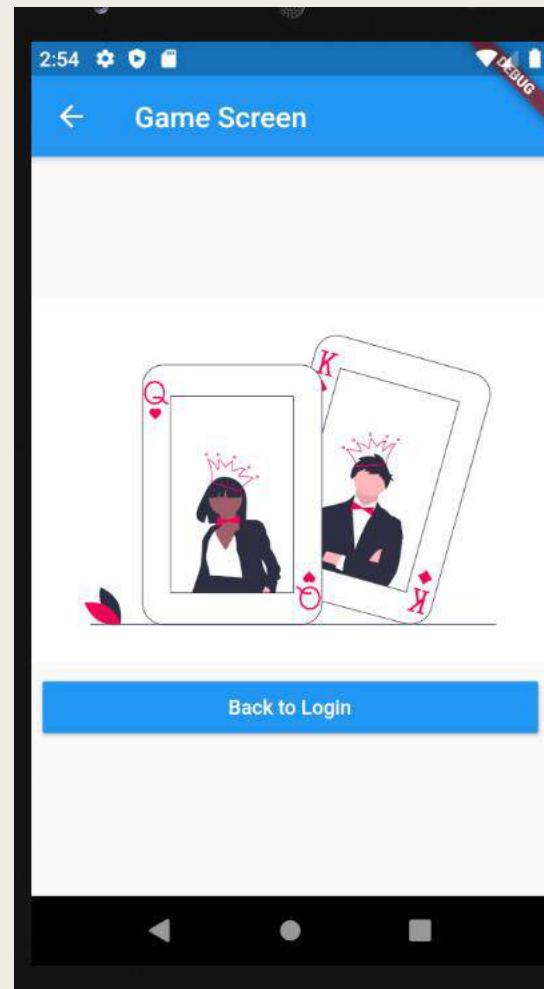
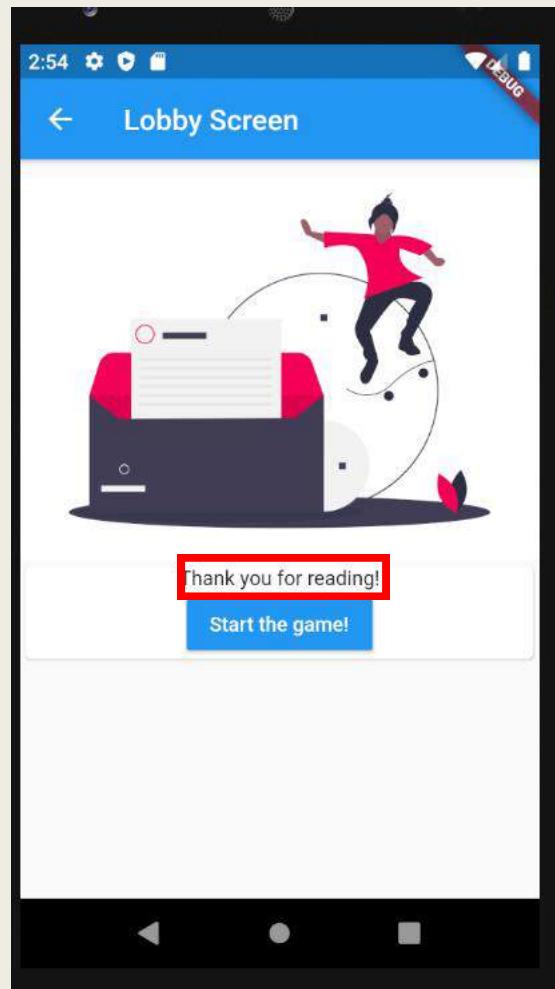
Example: 2 Login Screen

```
31   <Container(          width: double.infinity,          child: RaisedButton(            child: Text(              "Login",              style: TextStyle(                color: Colors.white,              ), // TextStyle            ), // Text            onPressed: () {              Navigator.pushNamed(                context, LobbyScreen.route,                arguments: {"user-msg": "Thank you for reading!"},);            },            color: Colors.blue,          ), // RaisedButton        ) // Container      ], // <Widget>[]    ), // Column  ), // Padding  ), // Card  ) // Padding53  ], // <Widget>[]54  [], // Column55  ), // Container56 ); // Scaffold57 }58 }
```

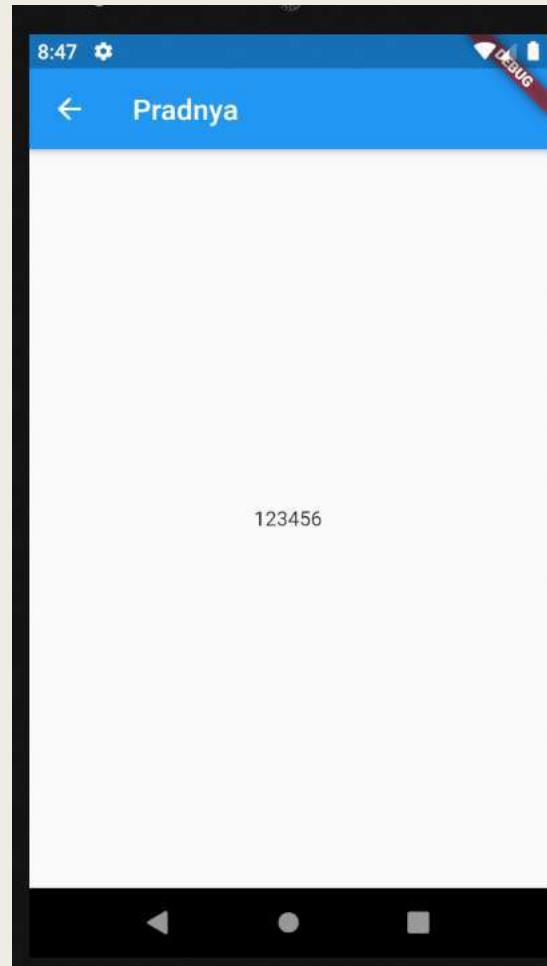
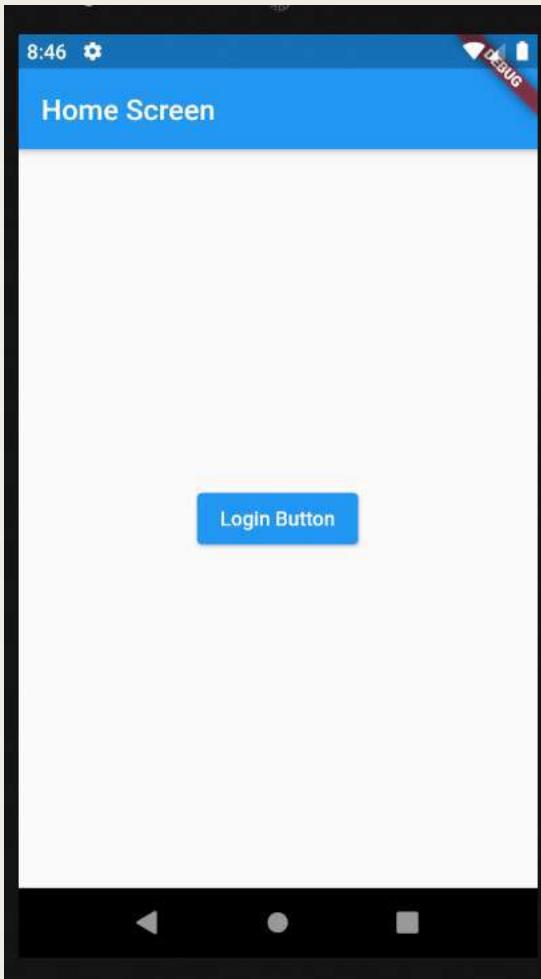
Example: 2 LobbyScreen

```
lib > screens > LobbyScreen.dart > LobbyScreen > build
  1 import 'package:flutter/material.dart';
  2 import 'GameScreen.dart';
  3
  4 class LobbyScreen extends StatelessWidget {
  5
  6     static const String route = "lobby-new";
  7     final Map<String, String> arguments;
  8
  9     LobbyScreen(this.arguments);
 10
 11    @override
 12    Widget build(BuildContext context) {
 13        return Scaffold(
 14            appBar: AppBar(
 15                title: Text("Lobby Screen"), ), // AppBar
 16            body: Column(
 17                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
 18                children: <Widget>[ Flexible(
 19                    child: Image.asset("assets/images/arrived.png"),
 20                ), // Flexible
 21                Container(
 22                    child: Card(
 23                        child: Column(
 24                            children: <Widget>[ Text(
 25                                arguments['user-msg'],
 26                                textAlign: TextAlign.center,
 27                            ), // Text
 28                            RaisedButton(color: Colors.blue,
 29                                onPressed: () {
 30                                    Navigator.push(context, MaterialPageRoute(
```

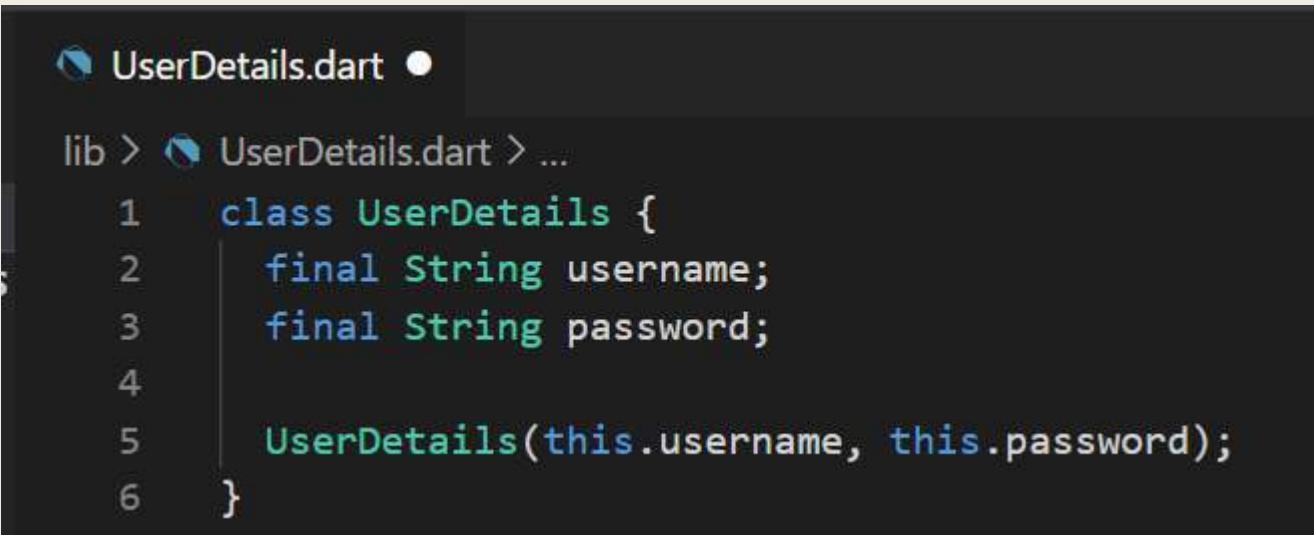
Output:



Example:3



Code:



A screenshot of a code editor showing a Dart file named `UserDetails.dart`. The code defines a class `UserDetails` with two final String properties: `username` and `password`, and a constructor that takes `this.username` and `this.password`.

```
lib > UserDetails.dart > ...
1   class UserDetails {
2     final String username;
3     final String password;
4
5     UserDetails(this.username, this.password);
6 }
```

main.dart X

```
lib > main.dart > generateRoute
1 import 'package:flutter/material.dart';
2 import 'package:dynamic_navigation_user_credentials/HomeScreen.dart';
3 import 'package:dynamic_navigation_user_credentials/ExtractArgumentScreen.dart';
4 Run | Debug
5 void main() => runApp(MyApp());
6
7 class MyApp extends StatelessWidget {
8     @override
9     Widget build(BuildContext context) {
10         return MaterialApp(
11             home:HomeScreen(),
12             title: 'Navigation with Arguments',
13             onGenerateRoute: generateRoute,
14             initialRoute: '/',
15         ); // MaterialApp
16     }
17 }
18 Route<dynamic> generateRoute (RouteSettings settings)
19 {
20     switch(settings.name)
21     {
22         case '/': return MaterialPageRoute(builder: (context) => HomeScreen(),);
23         break;
24
25         case '/extractArguments':
26             return MaterialPageRoute(builder: (context) =>ExtractArgumentsScreen(settings.arguments));
27             break;
28     }
29 }
```

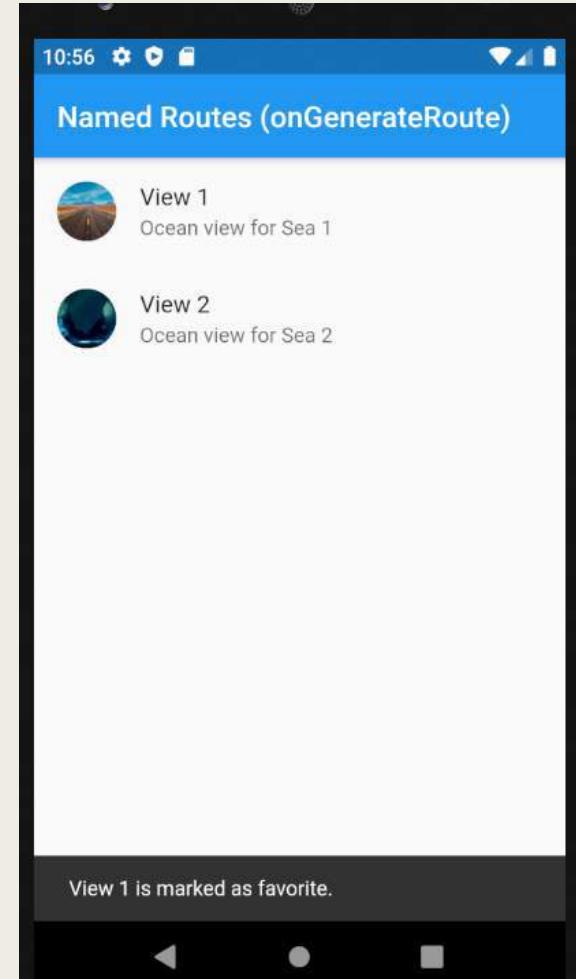
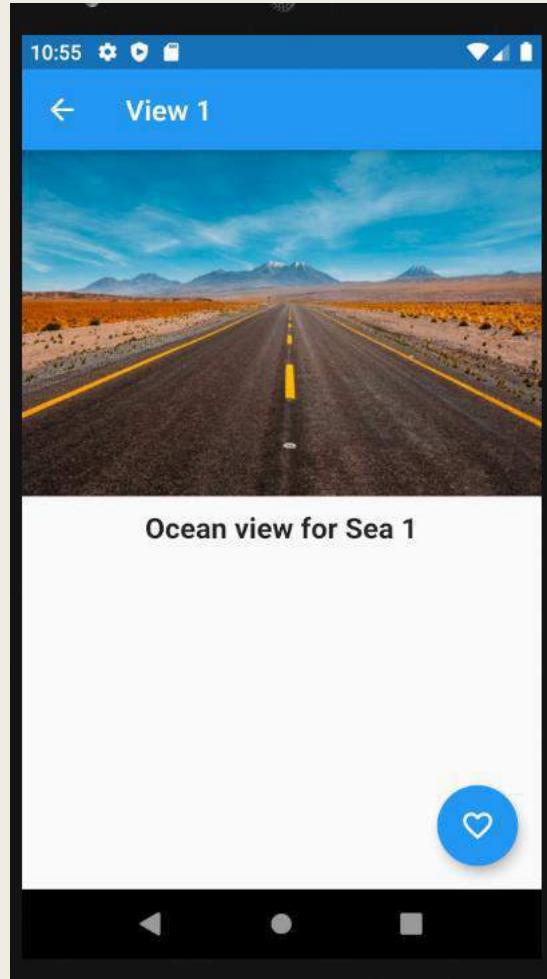
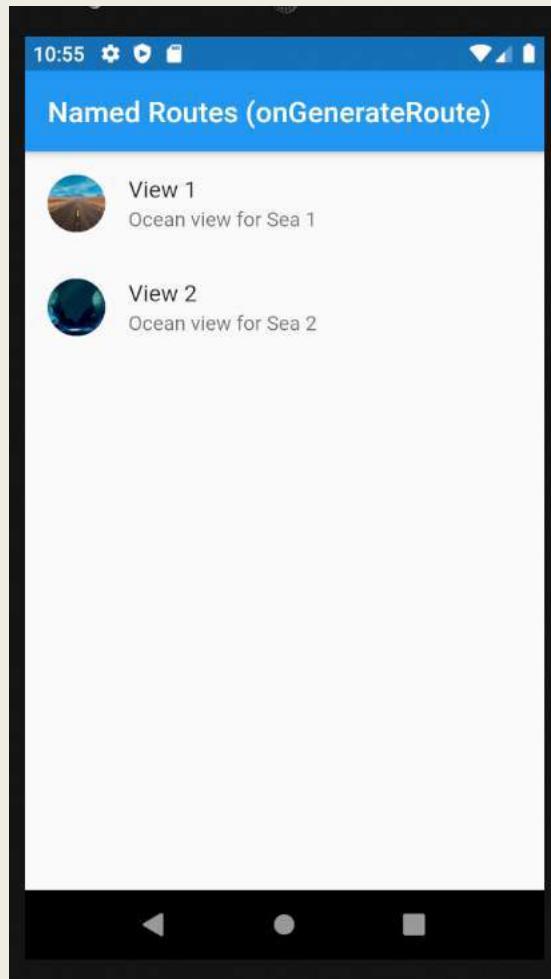
HomeScreen.dart

```
lib > HomeScreen.dart > HomeScreen
1 import 'package:flutter/material.dart';
2 import 'package:dynamic_navigation_user_credentials/UserDetails.dart';
3
4 class HomeScreen extends StatelessWidget {
5     @override
6     Widget build(BuildContext context) {
7         return Scaffold(
8             appBar: AppBar(
9                 title: Text('Home Screen'),
10            ), // AppBar
11            body: Center(
12                child: Column(
13                    mainAxisAlignment: MainAxisAlignment.center,
14                    children: <Widget>[
15                        ElevatedButton(
16                            child: Text("Login Button"),
17                            onPressed: () {
18                                Navigator.pushNamed(
19                                    context,
20                                    '/extractArguments',
21                                    arguments: UserDetails('Pradnya', '123456'),
22                                ),
23                            },
24                        ), // ElevatedButton
25                    ], // <Widget>[]
26                ), // Column
27            ), // Center
28        ); // Scaffold
29    }
30}
```

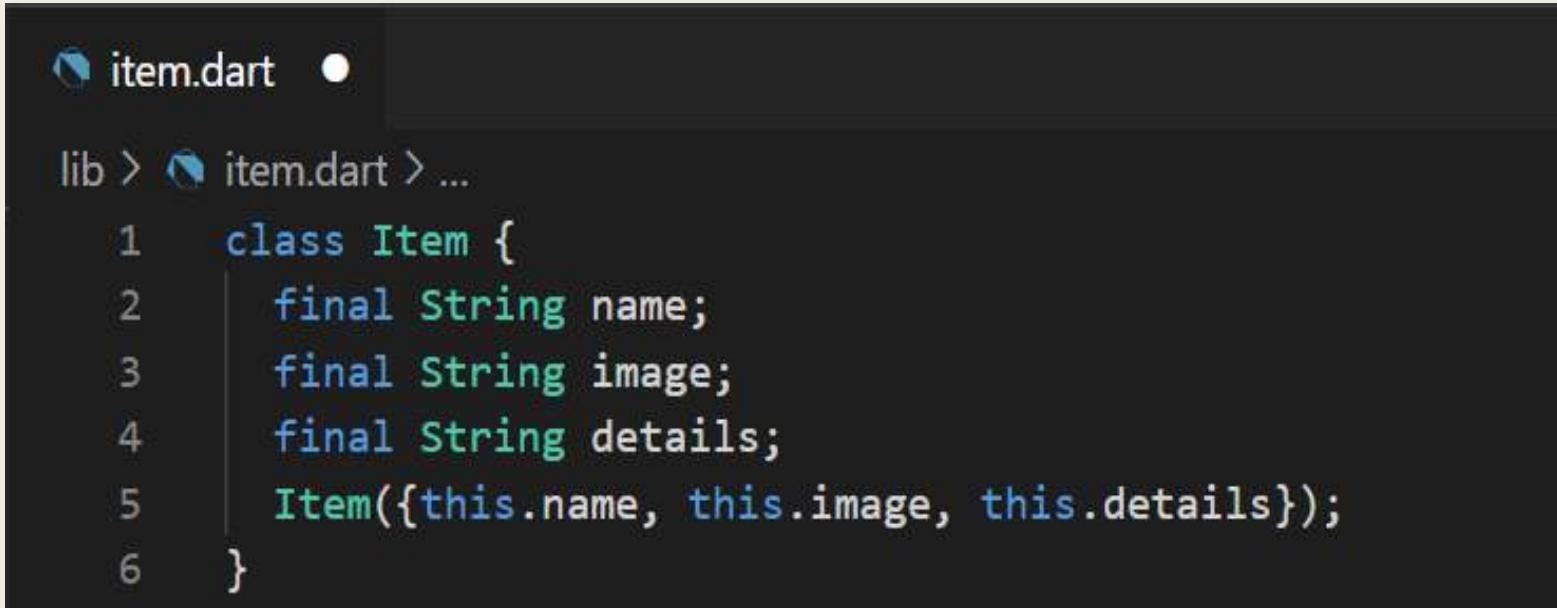
ExtractArgumentScreen.dart

```
lib > ExtractArgumentScreen.dart > ...
1  import 'package:flutter/material.dart';
2  import 'package:dynaminc_navigation_user_credentials/UserDetails.dart';
3
4  class ExtractArgumentsScreen extends StatelessWidget {
5      static const routeName = '/extractArguments';
6      final UserDetails user;
7      ExtractArgumentsScreen(this.user);
8
9      @override
10     Widget build(BuildContext context) {
11         return Scaffold(
12             appBar: AppBar(
13                 title: Text(this.user.username),
14             ), // AppBar
15             body: Center(
16                 child: Text(this.user.password),
17             ), // Center
18         ); // Scaffold
19     }
20 }
```

Example:4



Code:



A screenshot of a code editor showing a file named `item.dart`. The code defines a class `Item` with three final properties: `name`, `image`, and `details`. It also includes a constructor that initializes these properties.

```
lib > item.dart > ...
1   class Item {
2     final String name;
3     final String image;
4     final String details;
5     Item({this.name, this.image, this.details});
6 }
```



main.dart

```
lib > main.dart > main
  1 import 'package:flutter/material.dart';
  2 import 'item.dart';
  3 import 'notfound.dart';
  4 import 'page_details.dart';
  5
  6 List<Item> seasideList = [
  7   Item(
  8     name: 'View 1',
  9     image: 'assets/images/image1.jpg',
 10     details: "Ocean view for Sea 1"),
 11   Item(
 12     name: 'View 2',
 13     image: 'assets/images/image2.jpg',
 14     details: "Ocean view for Sea 2"),
 15 ];
 16 void main() => runApp(
 17   MaterialApp(
 18     debugShowCheckedModeBanner: false,
 19     home: PageListing(),
 20     initialRoute: '/',
 21     onGenerateRoute: generateRoute,
 22   ), // MaterialApp
 23 );
 24 );
```

```
26     Route<dynamic> generateRoute(RouteSettings routeSettings) {
27         final args = routeSettings.arguments;
28
29         switch (routeSettings.name) {
30             case '/':
31                 return MaterialPageRoute(
32                     builder: (context) => PageListing(),
33                 ); // MaterialPageRoute
34
35             case '/details':
36                 if (args is Item) {
37                     return MaterialPageRoute(
38                         builder: (context) => PageDetails(
39                             item: args,
40                         ), // PageDetails
41                     ); // MaterialPageRoute
42                 }
43
44                 return MaterialPageRoute(
45                     builder: (context) => PageNotFound(),
46                 ); // MaterialPageRoute
47
48             default:
49                 return MaterialPageRoute(
50                     builder: (context) => PageNotFound(),
51                 ); // MaterialPageRoute
52         }
53     }
```

```
55  class PageListing extends StatelessWidget {
56    @override
57    Widget build(BuildContext context) {
58      return Scaffold(
59        appBar: AppBar(
60          title: Text('Named Routes (onGenerateRoute)'),
61        ), // AppBar
62        body: ListView.builder(
63          itemCount: seasideList != null ? seasideList.length : 0,
64          itemBuilder: (BuildContext context, int index) {
65            return ListTile(
66              leading: CircleAvatar(
67                backgroundImage: ExactAssetImage(seasideList[index].image),
68              ), // CircleAvatar
69              title: Text("${seasideList[index].name}"),
70              subtitle: Text("${seasideList[index].details}"),
71              onTap: () {
72                _navigateToPageDetails(context, seasideList[index]);
73              },
74            ); // ListTile
75          ), // ListView.builder
76        ); // Scaffold
77      }
78    }
79  }
```

```
80     _navigateToPageDetails(BuildContext context, Item item) async {
81         final result = await Navigator.pushNamed(
82             context,
83             '/details',
84             arguments: item,
85         );
86
87         Scaffold.of(context)
88             ..removeCurrentSnackBar()
89             ..showSnackBar(SnackBar(content: Text("$result")));
90     }
91 }
```

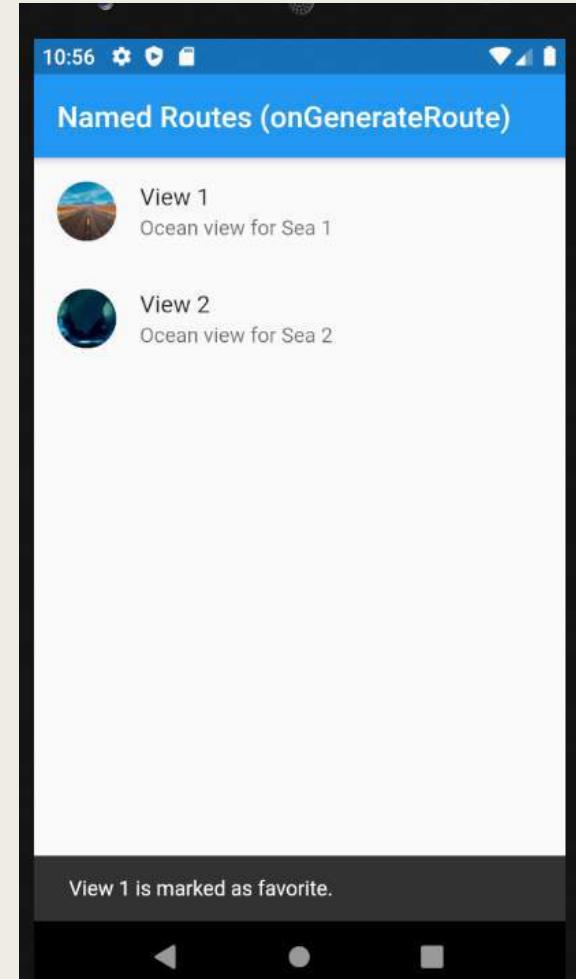
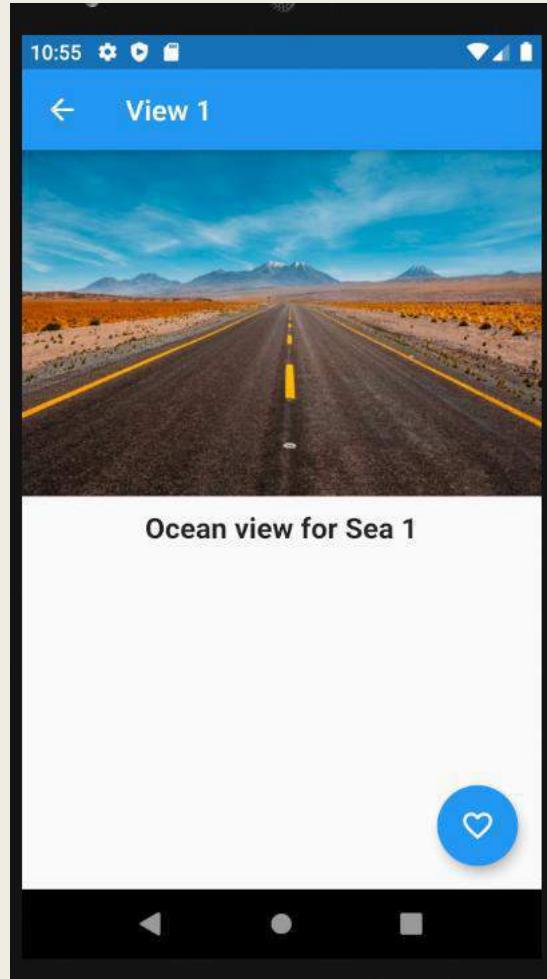
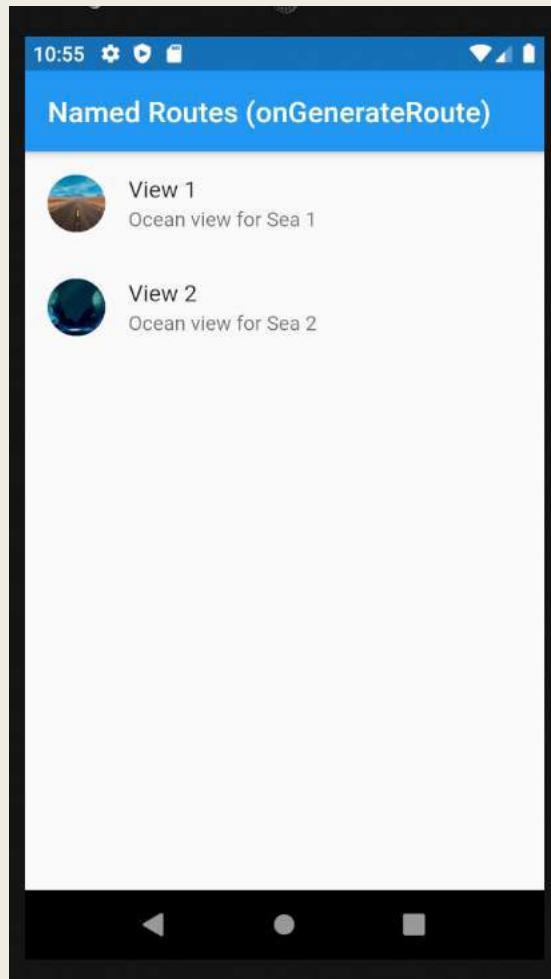
page_details.dart

```
lib > page_details.dart > ...
1 import 'package:flutter/material.dart';
2 import 'item.dart';
3
4 class PageDetails extends StatelessWidget {
5     final Item item;
6     const PageDetails({Key key, this.item}) : super(key: key);
7
8     @override
9     Widget build(BuildContext context) {
10         return Scaffold(
11             appBar: AppBar(
12                 title: Text(item.name),
13             ), // AppBar
14             body: Column(
15                 children: [
16                     Image.asset(item.image),
17                     SizedBox(height: 10),
18                     Text(item.details,
19                         style: TextStyle(fontWeight: FontWeight.bold, fontSize: 20),
20                     )), // Text
21             ), // Column
22             floatingActionButton: FloatingActionButton(
23                 child: Icon(Icons.favorite_border),
24                 onPressed: () {
25                     Navigator.pop(context, '${item.name} is marked as favorite.');
26                 },
27             ), // FloatingActionButton
28         ); // Scaffold
29     }
30 }
```

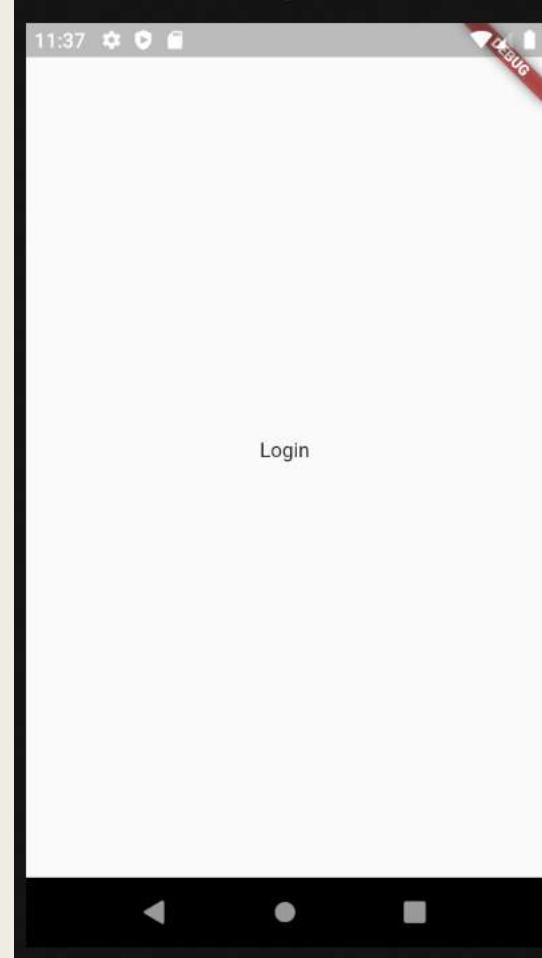
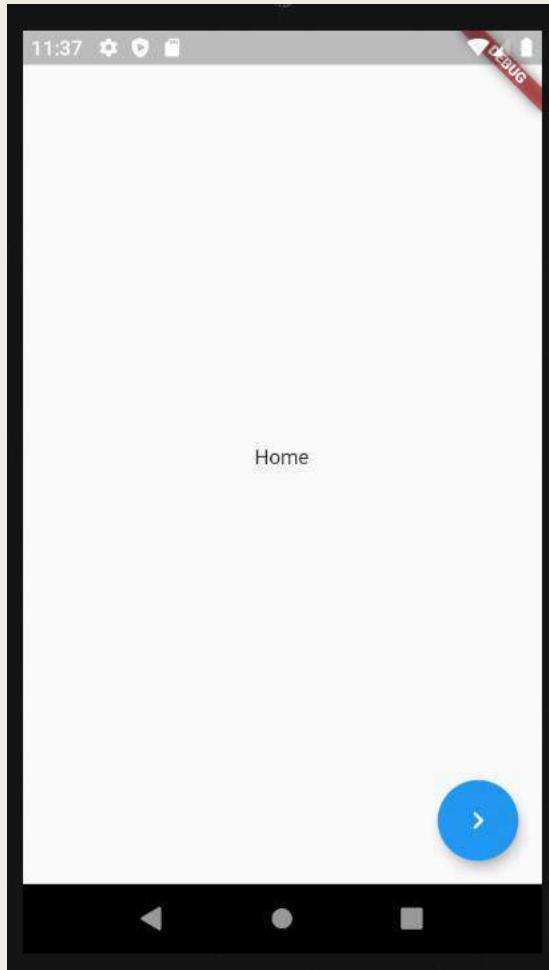
notfound.dart

```
lib > notfound.dart > ...
1 import 'package:flutter/material.dart';
2
3 class PageNotFound extends StatelessWidget {
4   @override
5   Widget build(BuildContext context) {
6     return Scaffold(
7       appBar: AppBar(
8         title: Text("Page not found"),
9       ), // AppBar
10      body: Center(
11        child: Text("Page is not available."),
12      ), // Center
13    ); // Scaffold
14  }
15 }
```

Example:4



Example:5 onUnknownRoute



main.dart

```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:dynamic_navigation_01/routing_constants.dart'
3 import 'package:dynamic_navigation_01/router.dart';
4 import 'package:dynamic_navigation_01/UndefinedView.dart';
Run | Debug
5 void main() => runApp(MyApp());
6
7 class MyApp extends StatelessWidget {
8     @override
9     Widget build(BuildContext context) {
10         return MaterialApp(
11             title: 'Named Routing',
12             onGenerateRoute: generateRoute,
13
14             onUnknownRoute: (settings) => MaterialPageRoute(
15                 builder: (context) => UndefinedView(
16                     name: settings.name,
17                 )), // UndefinedView // MaterialPageRoute
18
19             initialRoute: HomeViewRoute,
20             theme: ThemeData(
21                 primarySwatch: Colors.blue,
22             ), // ThemeData
23         ); // MaterialApp
24     }
25 }
```

routing_constants.dart

```
lib > routing_constants.dart > ...
1 const String HomeViewRoute = '/';
2 const String LoginViewRoute = 'login';
3
```

router.dart

```
lib > router.dart > ...
2 import 'package:dynamic_navigation_01/LoginView.dart';
3 import 'package:dynamic_navigation_01/HomeView.dart';
4 import 'package:dynamic_navigation_01/routing_constants.dart';
5
6 Route<dynamic> generateRoute(RouteSettings settings) {
7     switch (settings.name) {
8         case HomeViewRoute:
9             return MaterialPageRoute(builder: (context) => HomeView());
10        case LoginViewRoute:
11            return MaterialPageRoute(builder: (context) => LoginView());
12    }
13}
```

HomeView.dart

```
lib > HomeView.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:dynamic_navigation_01/routing_constants.dart';
3
4 class HomeView extends StatefulWidget {
5     @override
6     _HomeViewState createState() => _HomeViewState();
7 }
8
9 class _HomeViewState extends State<HomeView> {
10    @override
11    Widget build(BuildContext context) {
12        return Scaffold(
13            body: Center(child: Text('Home'),
14            ), // Center
15            floatingActionButton: FloatingActionButton(
16                onPressed:(){
17                    Navigator.pushNamed(context, LoginViewRoute);
18                },
19
20                child: Icon(Icons.navigate_next),), // FloatingActionButton
21            ); // Scaffold
22        }
23    }
```

 *LoginView.dart* Xlib >  *LoginView.dart* > ...

```
1 import 'package:flutter/material.dart';
2
3 class LoginView extends StatelessWidget {
4     @override
5     Widget build(BuildContext context) {
6         return Scaffold(
7             body: Center(child: Text('Login'),),
8         ); // Scaffold
9     }
10 }
```

UndefinedView.dart

```
lib > UndefinedView.dart > ...
1 import 'package:flutter/material.dart';
2
3 class UndefinedView extends StatelessWidget {
4
5     final String name;
6     const UndefinedView({Key key, this.name}) : super(key: key);
7
8     @override
9     Widget build(BuildContext context) {
10         return Scaffold(
11             body: Center(
12                 child: Text('Route for $name is not defined'),
13             ), // Center
14         ); // Scaffold
15     }
16 }
```

Asynchronous Programming

- Uses following API:

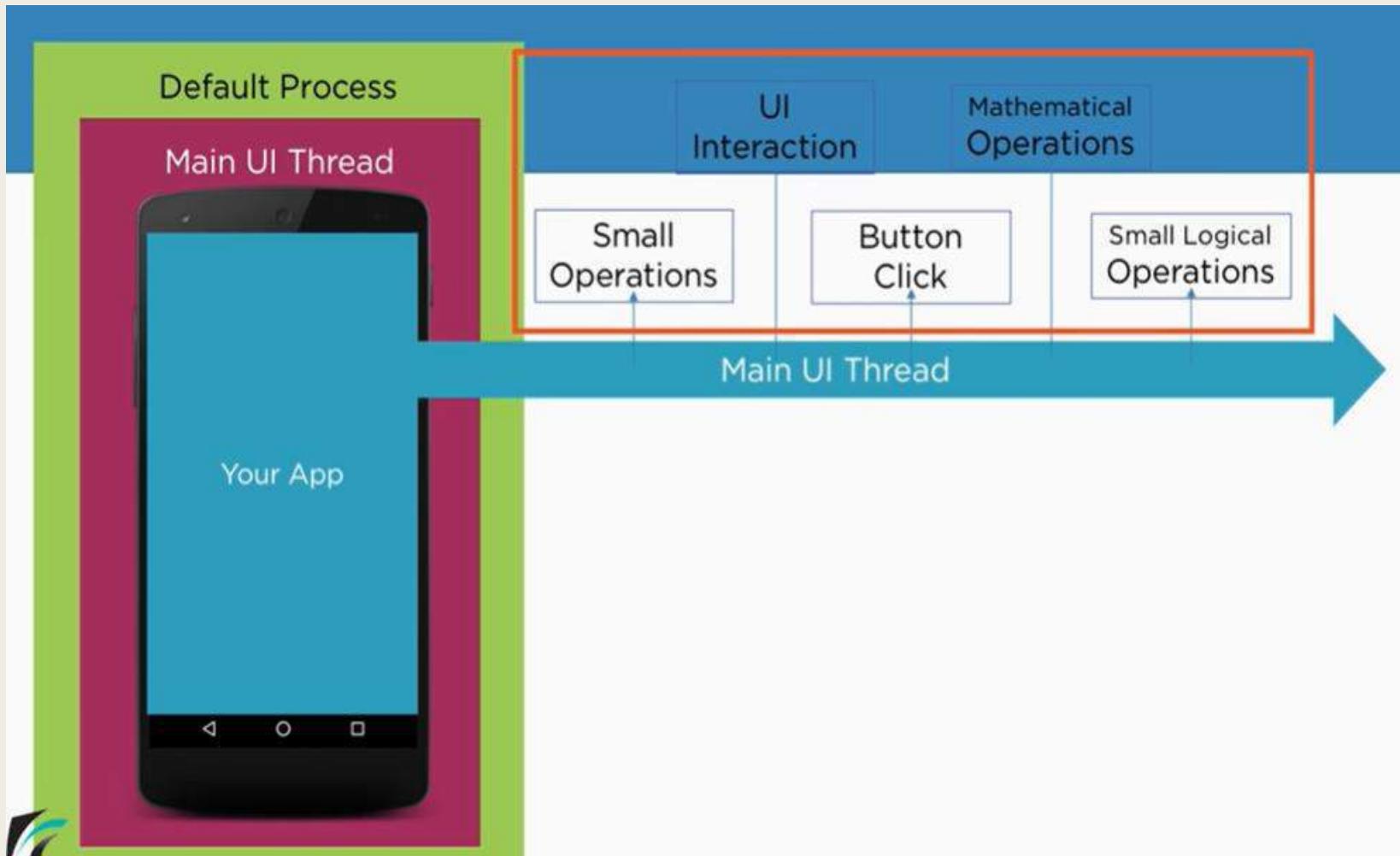
- *Future*

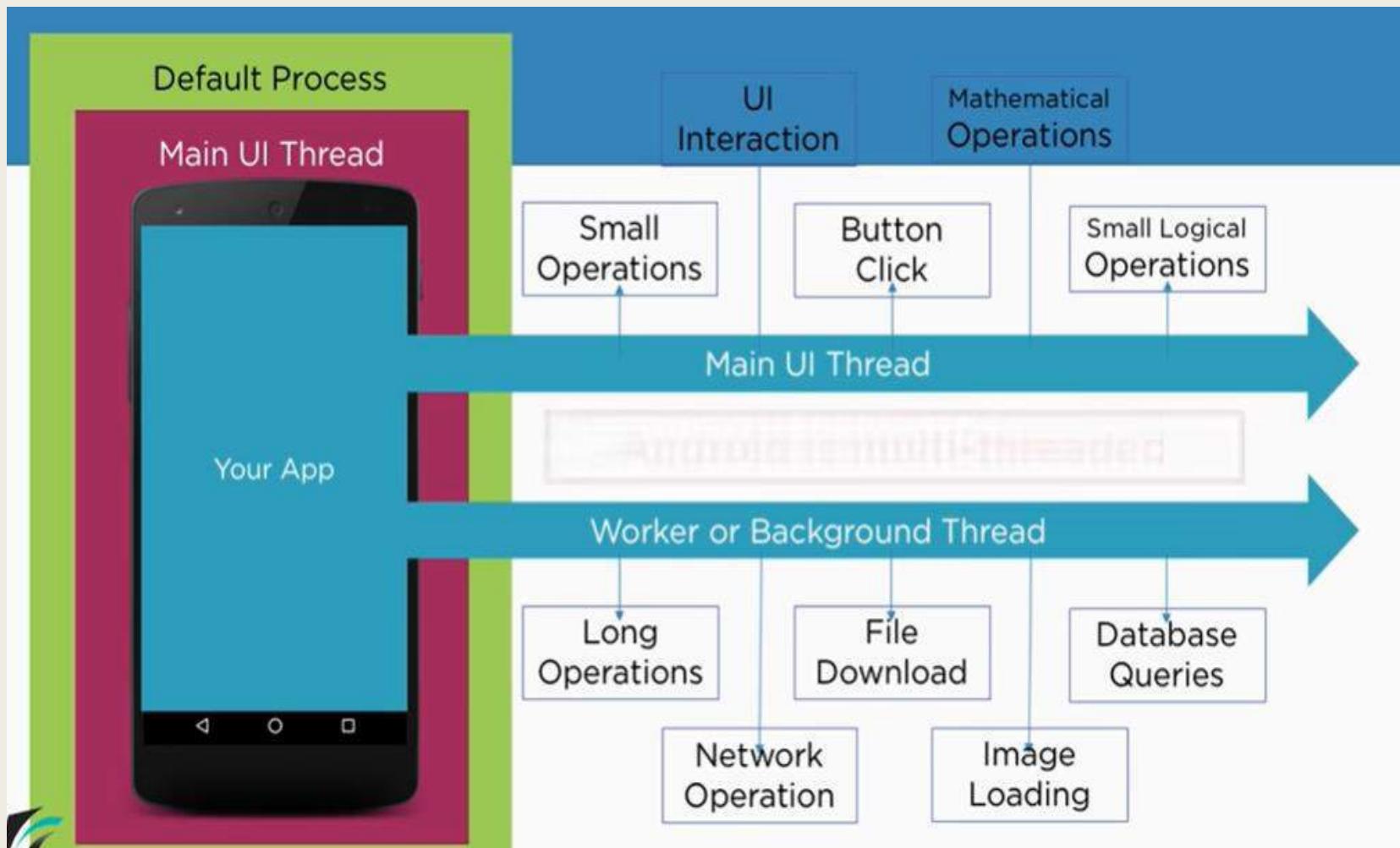
- Functions:

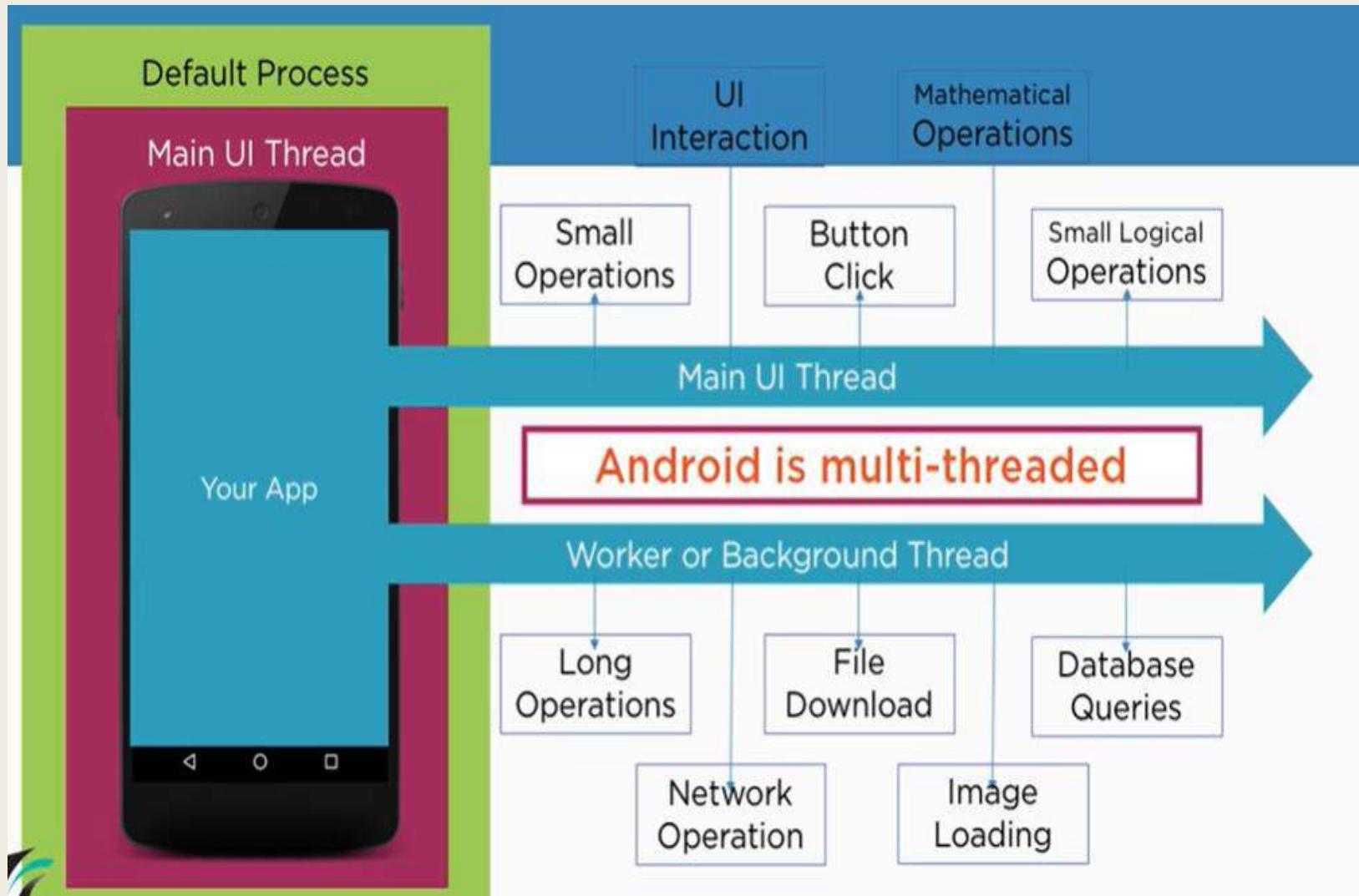
- *Await*

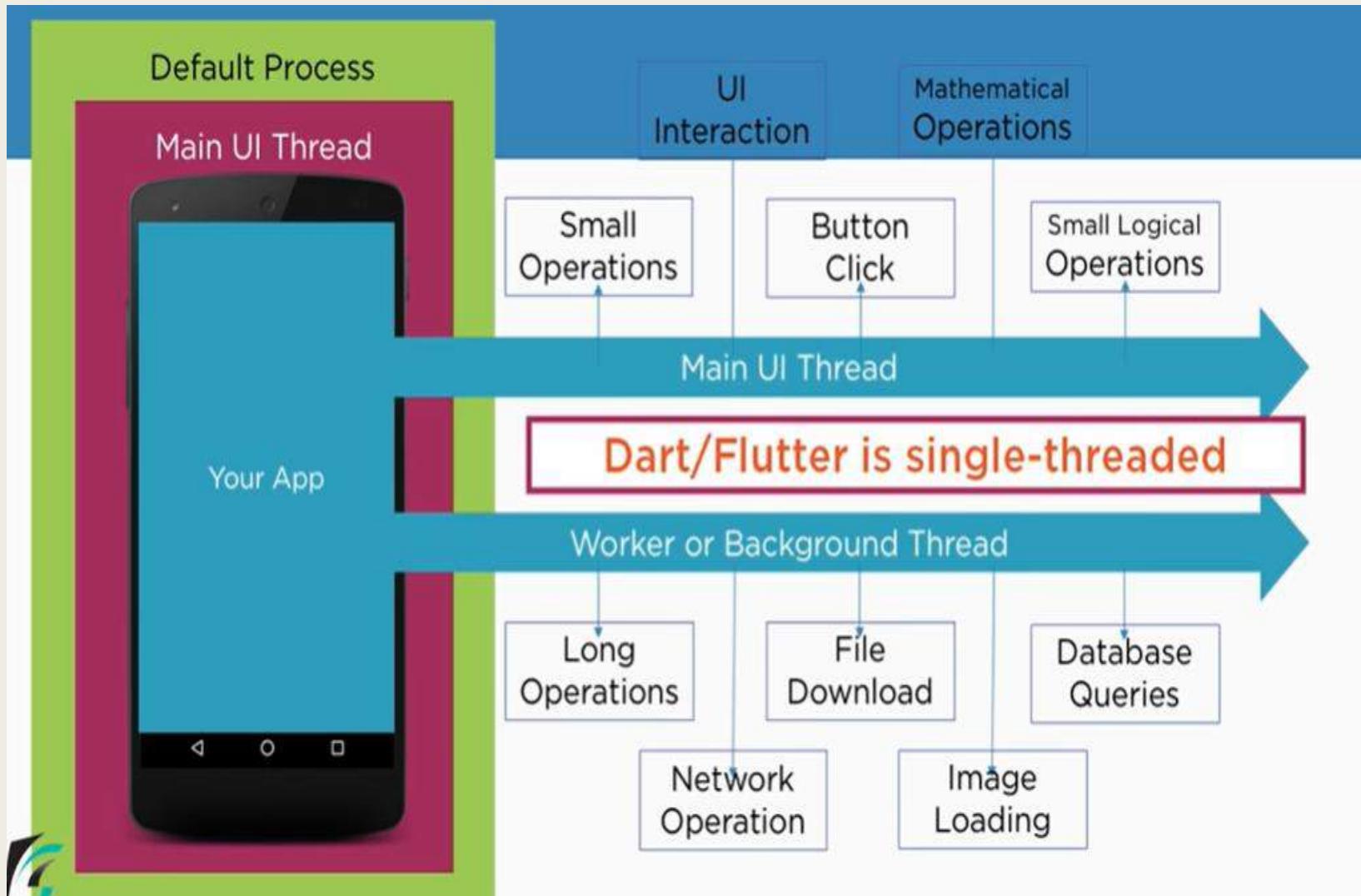
- *Async*

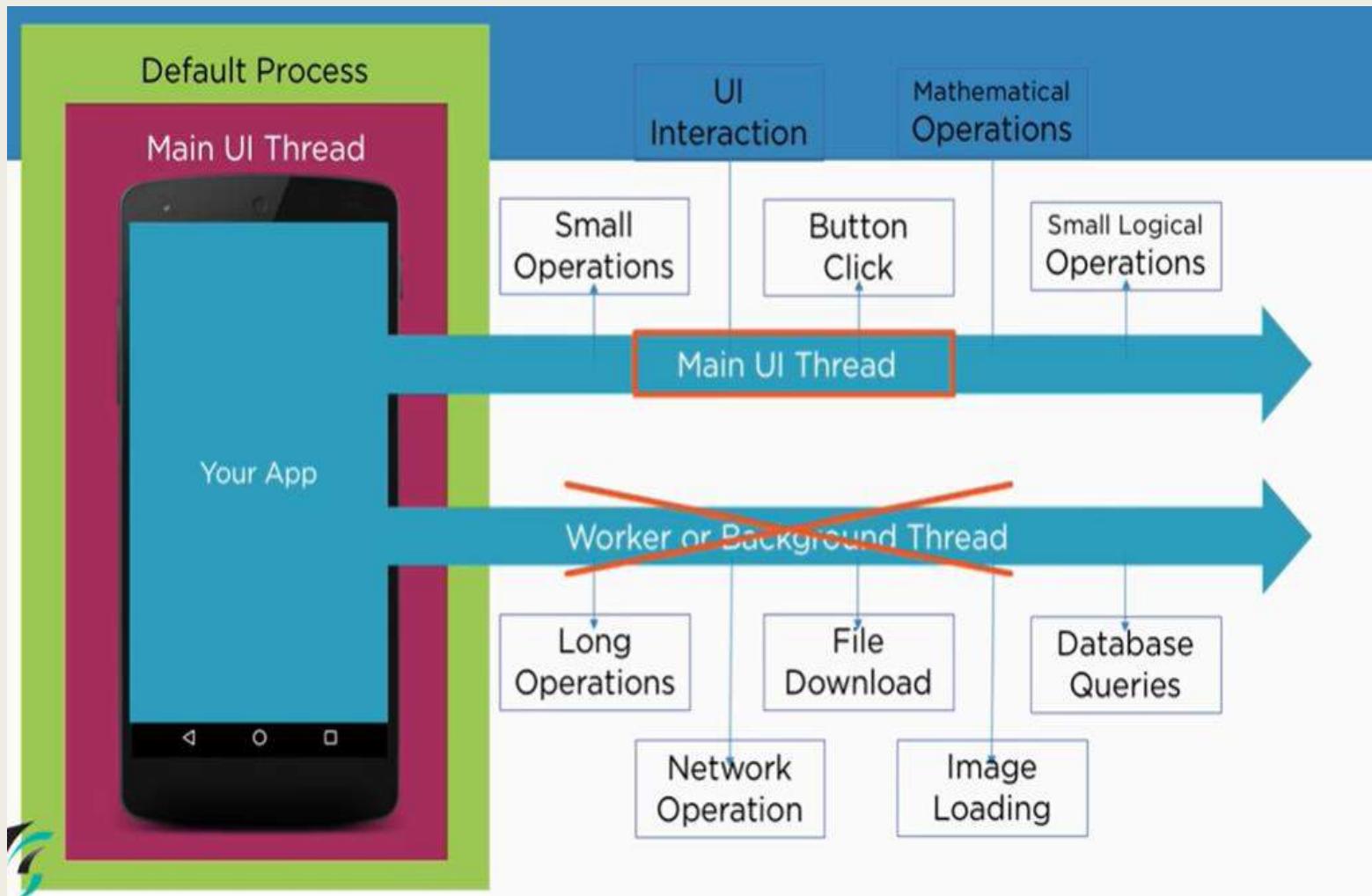
Introduction

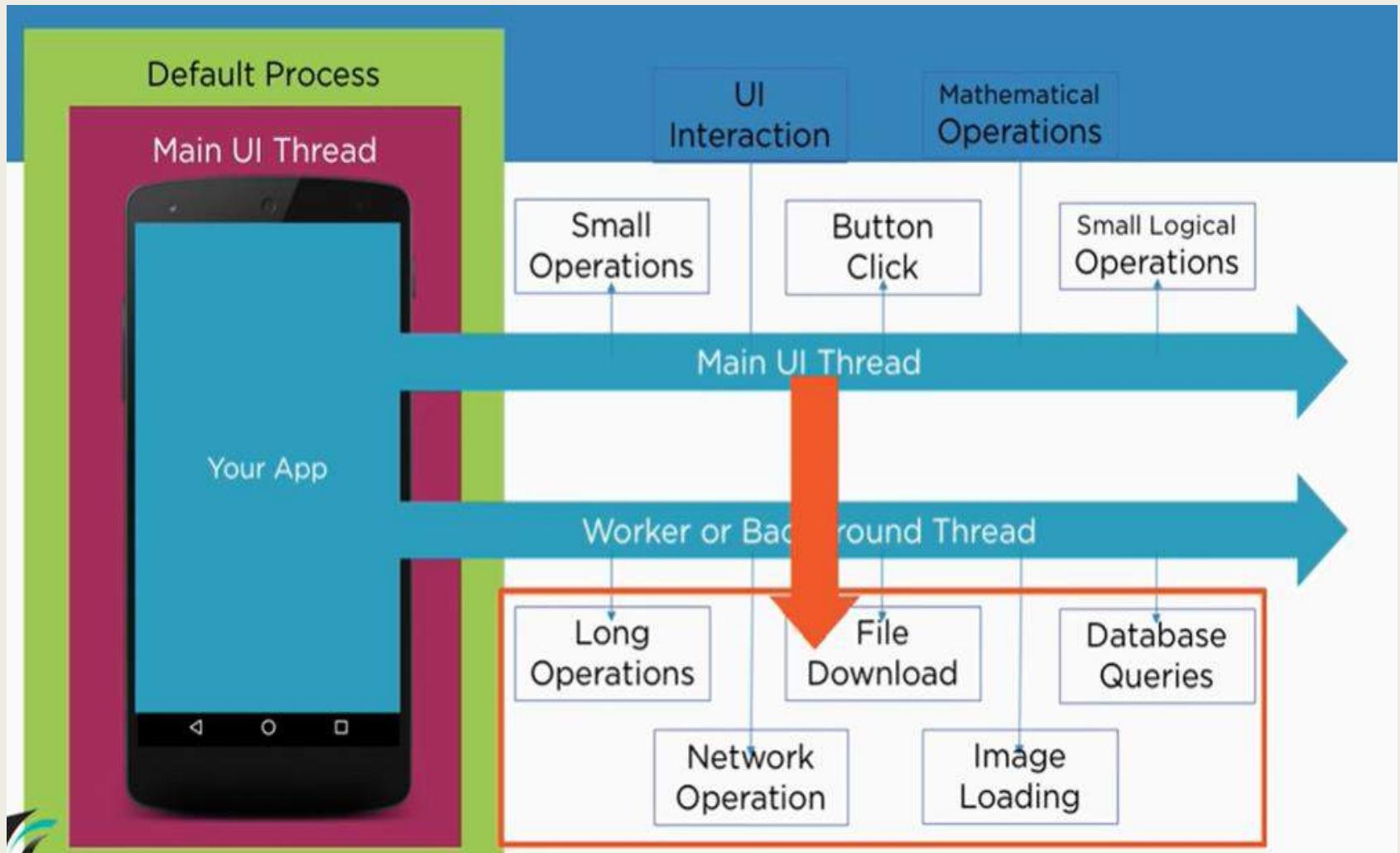












Future, Async and Await API's...

Example:

```
import 'dart:async';

main()
{
}

printFileContent()
{
}

//Ex: long run operation of downloading file from server
downloadFile()
{
    Future <String> result = Future.delayed(Duration(seconds:6),()
    {
        return 'My secret file content';
    });
    return result;
}
```

```
import 'dart:async';

main()
{
}

printFileContent()
{
    Future<String> fileContent= downloadFile();
    print('File content is $fileContent');
}

//Ex: long run operation of downloading file from server
downloadFile()
{
    Future <String> result = Future.delayed(Duration(seconds:6),()
    {
        return 'My secret file content';
    });
    return result;
}
```

```
import 'dart:async';

main()
{
  print('main thread starts');
  printFileContent();
  print('main thread ends');
}

printFileContent()
{
  Future<String> fileContent= downloadFile();
  print('File content is $fileContent');
}

//Ex: long run operation of downloading file from server
downloadFile()
{
  Future <String> result = Future.delayed(Duration(seconds:6),()
  {
    return 'My secret file content';
  });
  return result;
}


```

Console

```
main thread starts
File content is Instance of '_Future<String>'
main thread ends
```

```
import 'dart:async';

main()
{
    print('main thread starts');
    printFileContent();
    print('main thread ends');
}

printFileContent() async
{
    String fileContent= await downloadFile();
    print('File content is $fileContent');
}

//Ex: long run operation of downloading file from server
downloadFile()
{
    Future <String> result = Future.delayed(Duration(seconds:6),()
    {
        return 'My secret file content';
    });
    return result;
}
```

Console

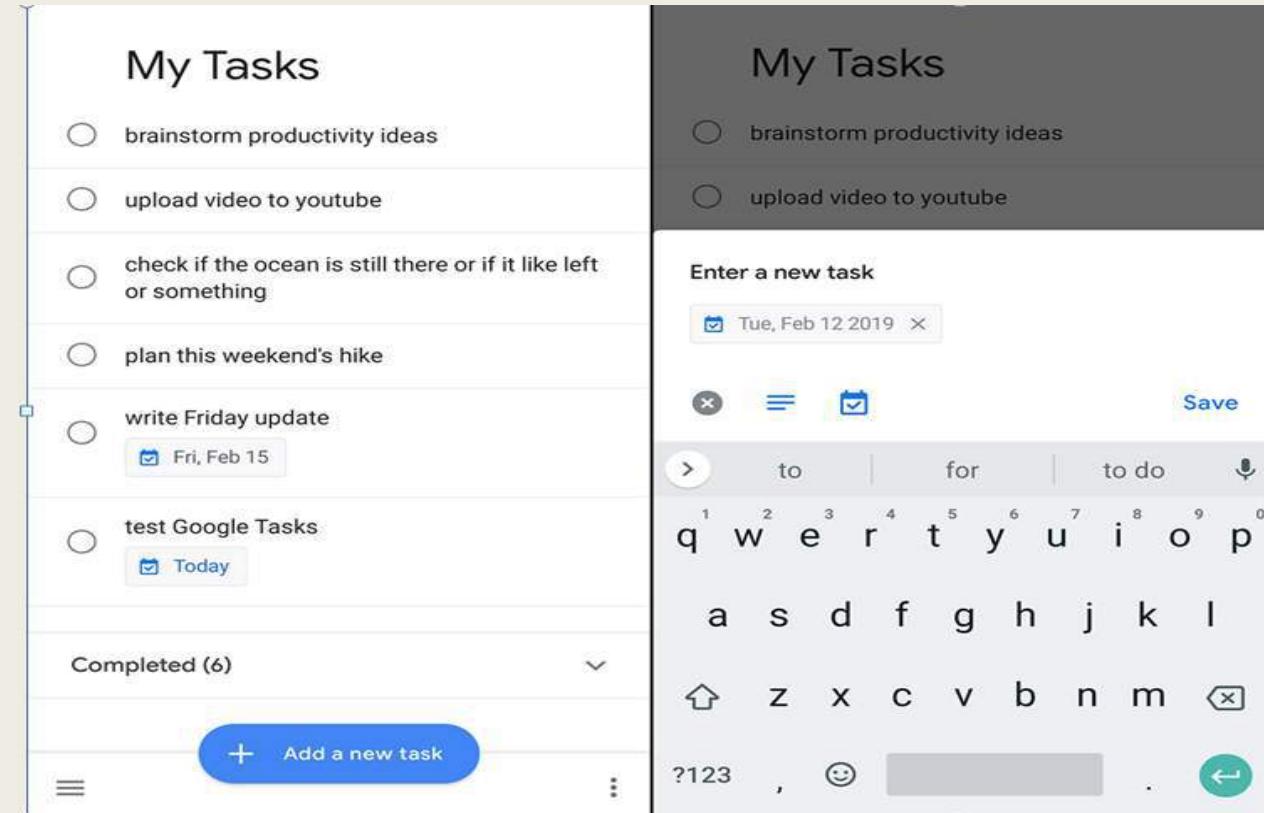
```
main thread starts
main thread ends
File content is My secret file
```

User Notification

- Bottom Sheet
- Snack Bar
- Toast Notification
- Alert Dialog
- Simple Dialog

BottomSheet

- Bottom sheet is a nice component given by Material design.
- It like a dialog which is open from the bottom.
- Most of the apps use this bottom sheet to add some extra setting kind of things.
- Example: Google Tasks application
- Can use any widget inside the bottom sheet based on your requirement.



Two kinds of BottomSheet

■ Persistent

- A persistent bottom sheet shows *information that supplements the primary content of the app.*
- A persistent bottom sheet remains visible even when the user interacts with other parts of the app.
- Persistent bottom sheets can be created and displayed with the **ScaffoldState.showBottomSheet function** or by specifying the **Scaffold.bottomSheet constructor** parameter.

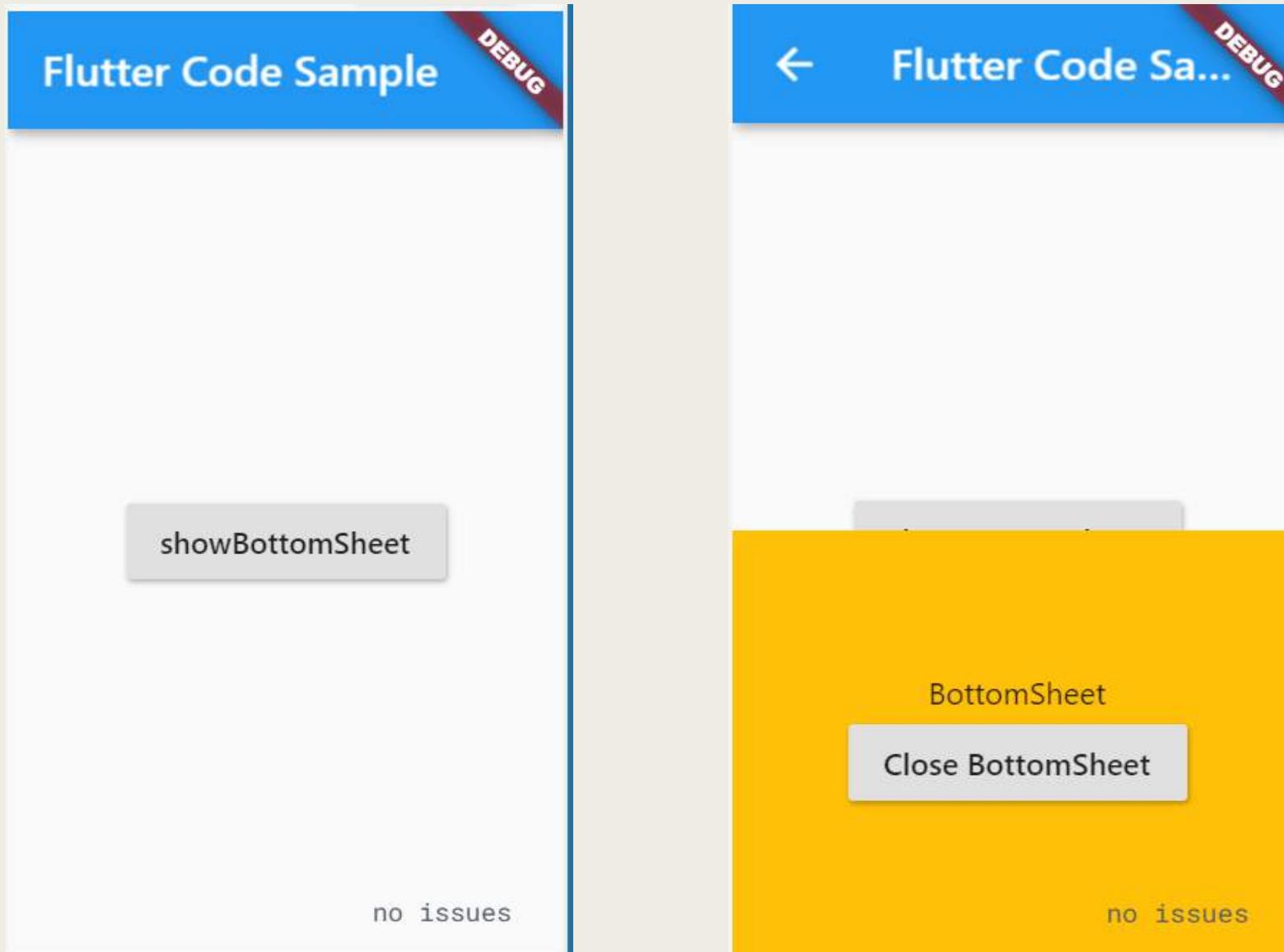
```
PersistentBottomSheetController<T> showBottomSheet <T>(  
    WidgetBuilder builder,  
    {Color backgroundColor,  
     double elevation,  
     ShapeBorder shape,  
     Clip clipBehavior}  
)
```

Persistent Bottom Sheet

```
10 class _State extends State<MyApp>
11 {
12     static const String _title = 'Flutter Code Sample';
13     void _showBottom()
14     {
15         showBottomSheet<void>(
16             context: context,
17             builder: (BuildContext context)
18         {
19             return Container(
20                 height: 200,
21                 color: Colors.amber,
22                 child: Center(
23                     child: Column(
24                         mainAxisAlignment: MainAxisAlignment.center,
25                         mainAxisSize: MainAxisSize.min,
26                         children: <Widget>[
27                             const Text('BottomSheet'),
28                             RaisedButton(
29                                 child: const Text('Close BottomSheet'),
30                                 onPressed: () => Navigator.pop(context),
31                             ],
32                         ),
33                     ),
34                 );
35         });
36     }
```

```
38     @override
39     Widget build(BuildContext context) {
40         return MaterialApp(
41             title: _title,
42             home: Scaffold(
43                 appBar: AppBar(title: const Text(_title)),
44                 body: Center(
45                     child: RaisedButton(
46                         child: const Text('showBottomSheet'),
47                         onPressed: _showBottom,
48                     ),
49                 ),
50             ),
51         );
52     }
53 }
```

Persistent Bottom Sheet



Modal Bottom Sheet

■ *Modal*

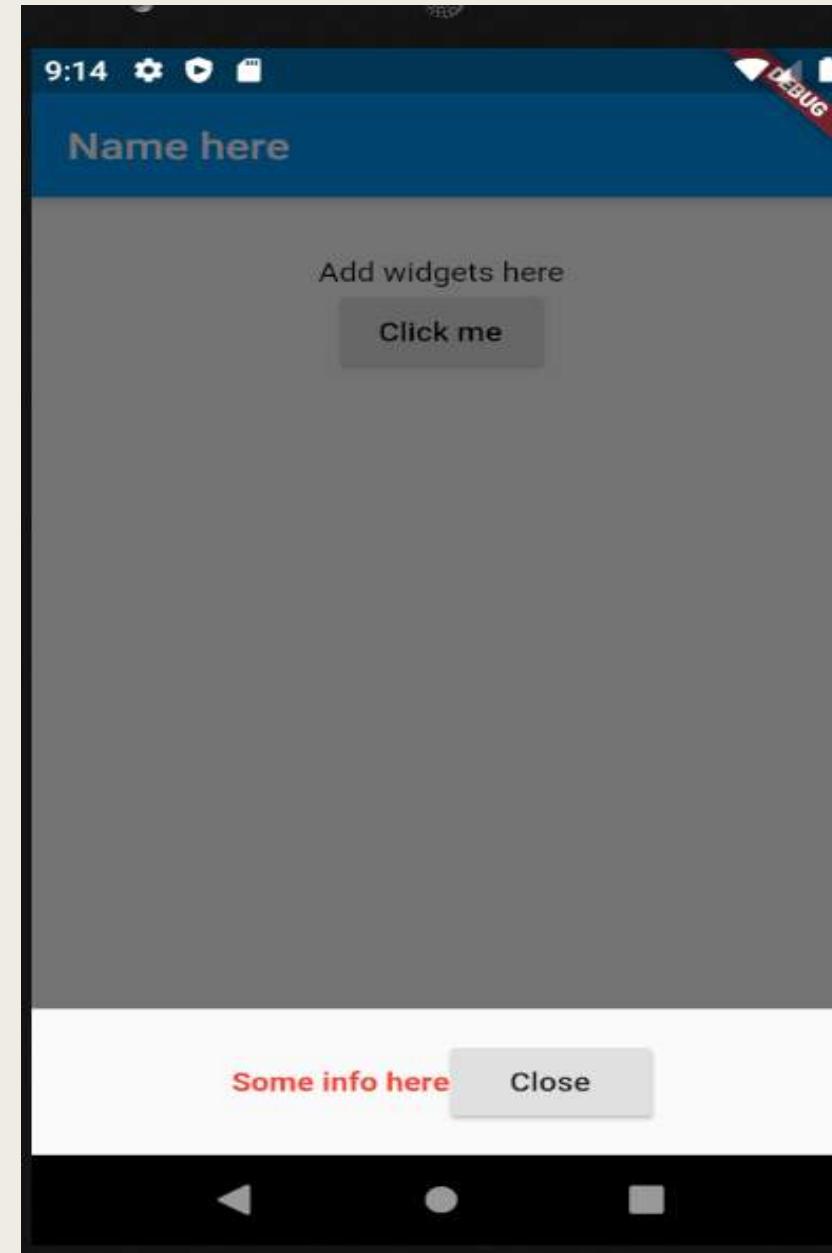
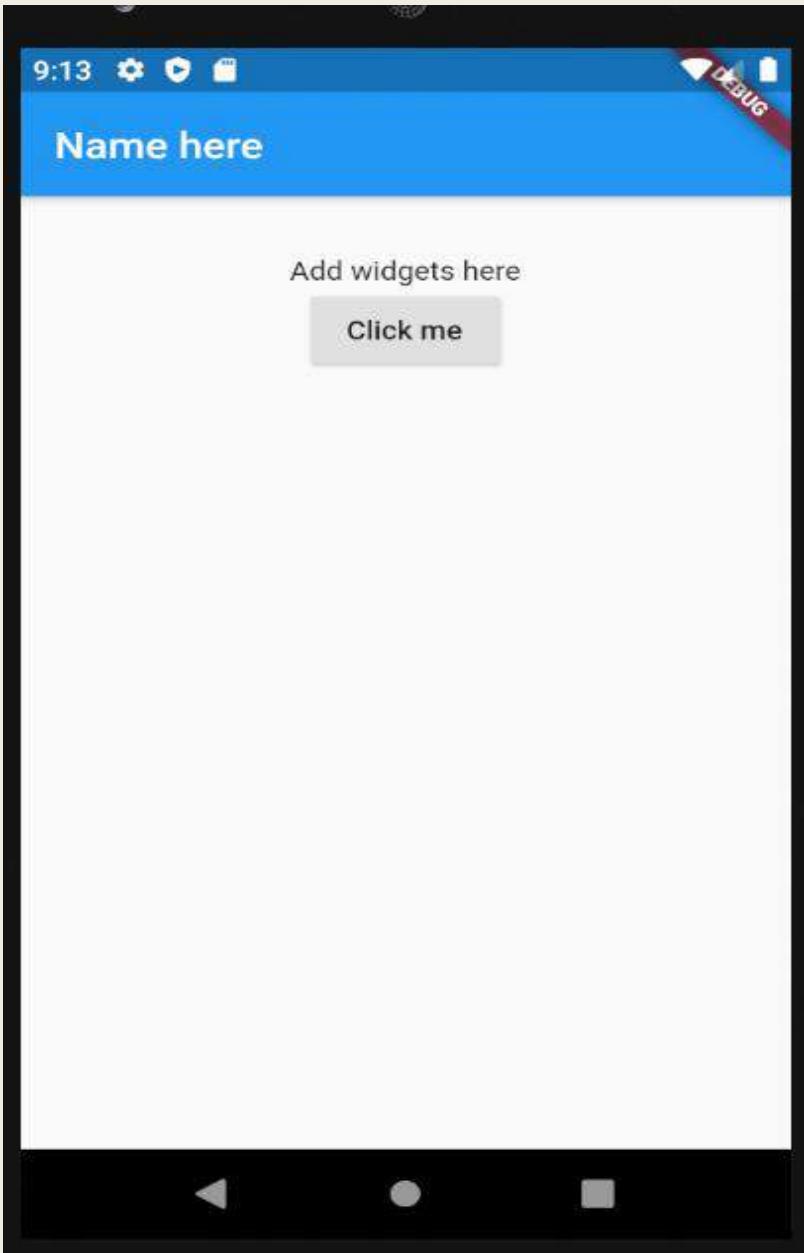
- A modal bottom sheet is an alternative to a menu or a dialog and prevents the user from interacting with the rest of the app.
- Modal bottom sheets can be created and displayed with the **showModalBottomSheet** function.

```
Future<T> showModalBottomSheet <T>(  
    {@required BuildContext context,  
     @required WidgetBuilder builder,  
     Color backgroundColor,  
     double elevation,  
     ShapeBorder shape,  
     Clip clipBehavior,  
     Color barrierColor,  
     bool isScrollControlled: false,  
     bool useRootNavigator: false,  
     bool isDismissible: true,  
     bool enableDrag: true}  
)
```

Modal Bottom Sheet

```
15 class _State extends State<MyApp>
16 {
17     void _showBottom()
18     {
19         showModalBottomSheet<void>(
20             context: context,
21             builder: (BuildContext context)
22             {
23                 return new Container(
24                     padding: EdgeInsets.all(15.0),
25                     child: new Row(
26                         mainAxisAlignment: MainAxisAlignment.center,
27                         children:<Widget>[
28                             new Text('Some info here',
29                                 style: new TextStyle(color:Colors.red,
30                                     fontWeight : FontWeight.bold)), // TextStyle // Text
31                             new RaisedButton(onPressed: ()=>Navigator.pop(context),
32                                 child:new Text('Close')),) // RaisedButton
33                         ],), // <Widget>[] // Row
34                 ); // Container
35             });
36 }
37 @override
38     Widget build(BuildContext context){
39         return new Scaffold(
40             appBar:new AppBar(
41                 title: new Text('Name here'),
42             ), // AppBar
43             body: new Container(
44                 padding: new EdgeInsets.all(32.0),
45                 child: new Center(
46                     child: new Column(children: <Widget>[
47                         new Text('Add widgets here'),
48                         new RaisedButton(onPressed: _showBottom,
49                             child: new Text('Click me'))], // RaisedButton
50                     ], // <Widget>[]
51                 ) // Column
52             ), // Center
53         ), // Container
54     ); // Scaffold
55 }
56 }
57 }
```

Modal Bottom Sheet(contd..)

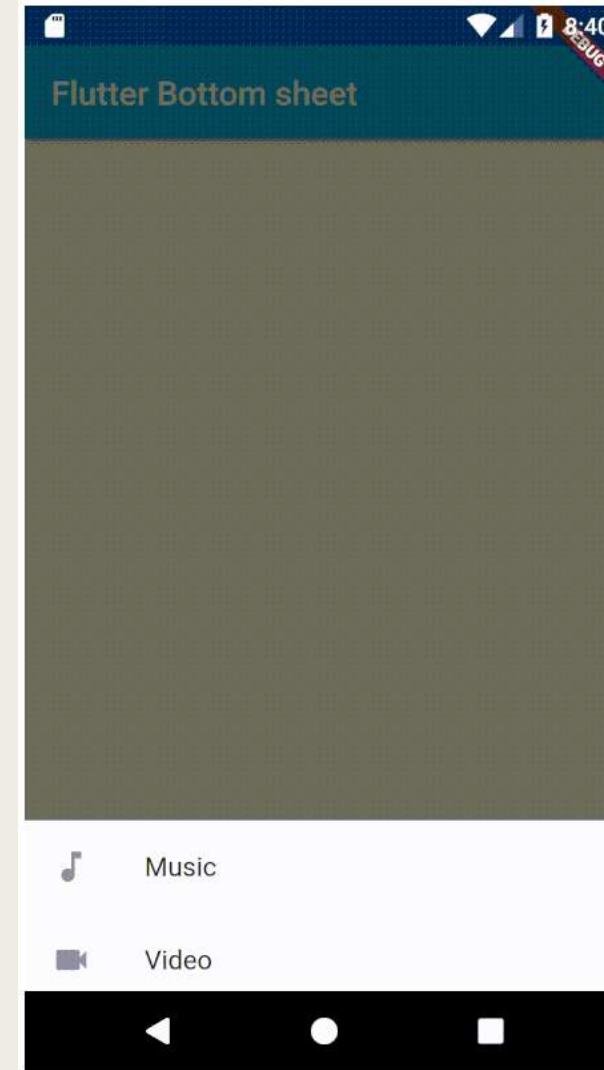
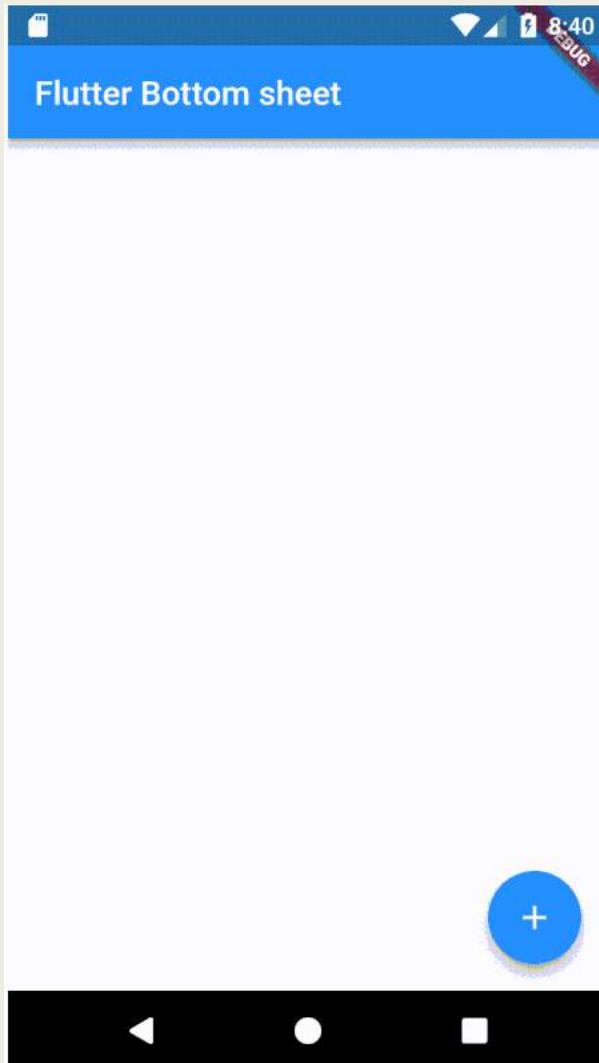


Modal Bottom Sheet(2)

```
2 class _MyHomePageState extends State<MyHomePage> {  
3  
4     void _settingModalBottomSheet(context){  
5         showModalBottomSheet(  
6             context: context,  
7             builder: (BuildContext bc){  
8                 return Container(  
9                     child: new Wrap(  
10                        children: <Widget>[  
11                            new ListTile(  
12                                leading: new Icon(Icons.music_note),  
13                                title: new Text('Music'),  
14                                onTap: () => {}  
15                            ),  
16                            new ListTile(  
17                                leading: new Icon(Icons.videocam),  
18                                title: new Text('Video'),  
19                                onTap: () => {},  
20                            ),  
21                            ],  
22                        ),  
23                    );  
24                }  
25            );  
26        }  
27    }  
28 }
```

```
28     @override  
29     Widget build(BuildContext context) {  
30         return new Scaffold(  
31             appBar: new AppBar(  
32                 title: new Text(widget.title),  
33             ),  
34             floatingActionButton: new FloatingActionButton(  
35                 onPressed: (){  
36                     _settingModalBottomSheet(context);  
37                 },  
38                 child: new Icon(Icons.add),  
39             ),  
40         );  
41     }  
42 }
```

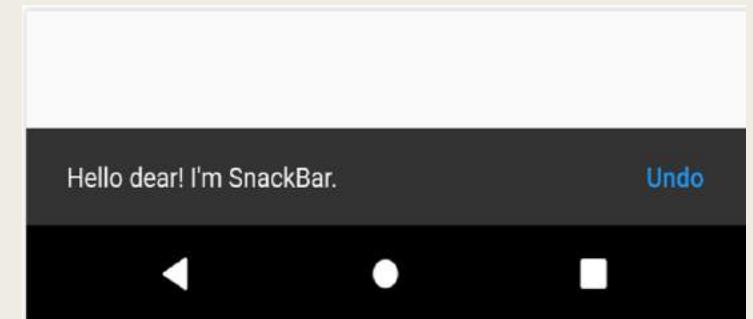
Modal Bottom Sheet(2)



SnackBar Widget

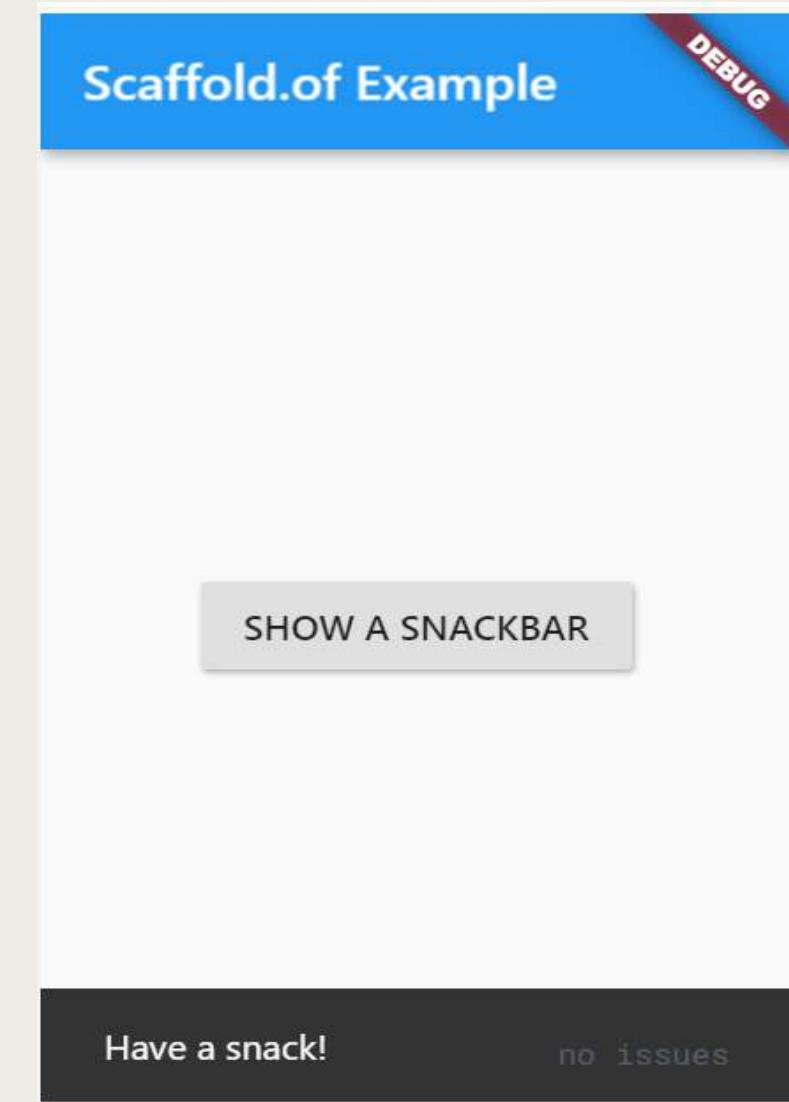
- SnackBar is usually used with Scaffold widget
- SnackBar is used to show a message to user for a brief period of time. Also, you can provide an action label in SnackBar to perform an action when pressed on it.

```
final snackBar = SnackBar(  
    content: Text('Hello! I am a SnackBar!'),  
    duration: Duration(seconds: 5),  
    action: SnackBarAction(  
        label: 'Undo',  
        onPressed: () {  
            // Some code to undo the change.  
        },  
    ),  
);
```



SnackBar Widget(1)

```
10  class MyApp extends StatelessWidget {  
11    // This widget is the root of your application.  
12    @override  
13    Widget build(BuildContext context) {  
14      return MaterialApp(  
15        title: 'Flutter Code Sample for Scaffold.of.',  
16        theme: ThemeData(  
17          primarySwatch: Colors.blue,  
18        ),  
19        home: Scaffold(  
20          appBar: AppBar(title: Text('Scaffold.of Example')),  
21          body: Center(  
22            child: RaisedButton(  
23              child: Text('SHOW A SNACKBAR'),  
24              onPressed: () {  
25                Scaffold.of(context).showSnackBar(  
26                  SnackBar(  
27                    content: Text('Have a snack!'),  
28                  ),  
29                );  
30              },  
31            ),  
32          ),  
33        ),  
34      );  
35    }  
36  }  
37 }
```

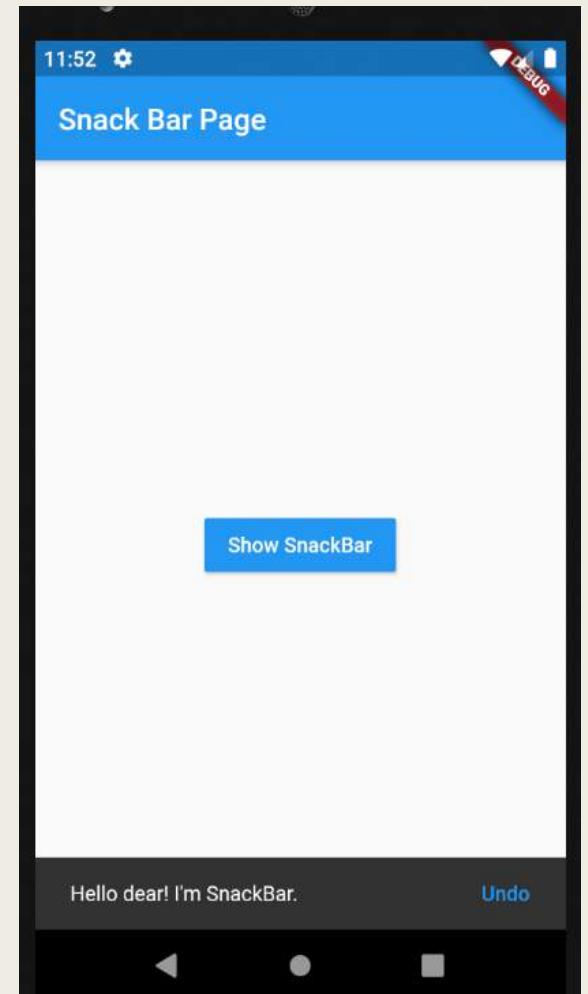


SnackBar Widget(2)

```
12 class _State extends State<MyApp> {
13   final GlobalKey<ScaffoldState> _scaffoldKey = new GlobalKey<ScaffoldState>();
14
15   void _showMessageInScaffold(String message){
16
17     _scaffoldKey.currentState.showSnackBar(
18       SnackBar(
19         content: Text(message),
20         action: SnackBarAction(
21           label: 'Undo',
22           onPressed: () {
23             // some code
24             print('Action in Snackbar has been pressed.');
25           },
26         ), // SnackBarAction
27       ) // SnackBar
28     );
29 }
```

SnackBar Widget(2)

```
30  @override
31  Widget build(BuildContext context) {
32    return MaterialApp(
33      home:Scaffold(
34        key: _scaffoldKey,
35        appBar: AppBar(
36          title: Text('SnackBar Page'),
37        ), // AppBar
38        body: Center(
39          child: RaisedButton(
40            textColor: Colors.white,
41            color: Colors.blue,
42            child: Text('Show SnackBar'),
43            onPressed: (){
44              _showMessageInScaffold("Hello dear! I'm SnackBar.");
45            },
46          ) // RaisedButton
47        ) // Center
48      ) // Scaffold
49    ); // MaterialApp
50  }
51 }
```



Toast Widget

This is a Toast message



- Toast has been used as a standard way to show flash message in Android.
- It is a very small message which mainly popup at the bottom of the device screen.
- It will disappear on its own after completing the time provided by the developers.
- A developer mostly used the toast notification for showing feedback on the operation performed by the user.
- To implement toast notification, we need to import **fluttertoast** library in Flutter.

pubspec.yaml

```
dependencies:  
  flutter:  
    sdk: flutter  
  cupertino_icons: ^0.1.2  
  fluttertoast: ^3.1.0
```

main.dart

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

Toast Widget Constructor

Option	Description
<code>String msg</code> (required)	The message to be displayed
<code>Toast toastLength</code>	How long the it will be displayed. Values: <ul style="list-style-type: none">• <code>Toast.LENGTH_SHORT</code> (default)• <code>Toast.LENGTH_LONG</code>
<code>int timeInSecForIos</code>	How long the it will be displayed in iOS (default is 1 second)
<code>double fontSize</code>	The font size (default is 16.0)
<code>ToastGravity gravity</code>	Vertical gravity. Values: <ul style="list-style-type: none">• <code>ToastGravity.TOP</code>• <code>ToastGravity.CENTER</code>• <code>ToastGravity.BOTTOM</code> (default)
<code>Color backgroundColor</code>	The background color
<code>Color textColor</code>	The text color

Toast Widget

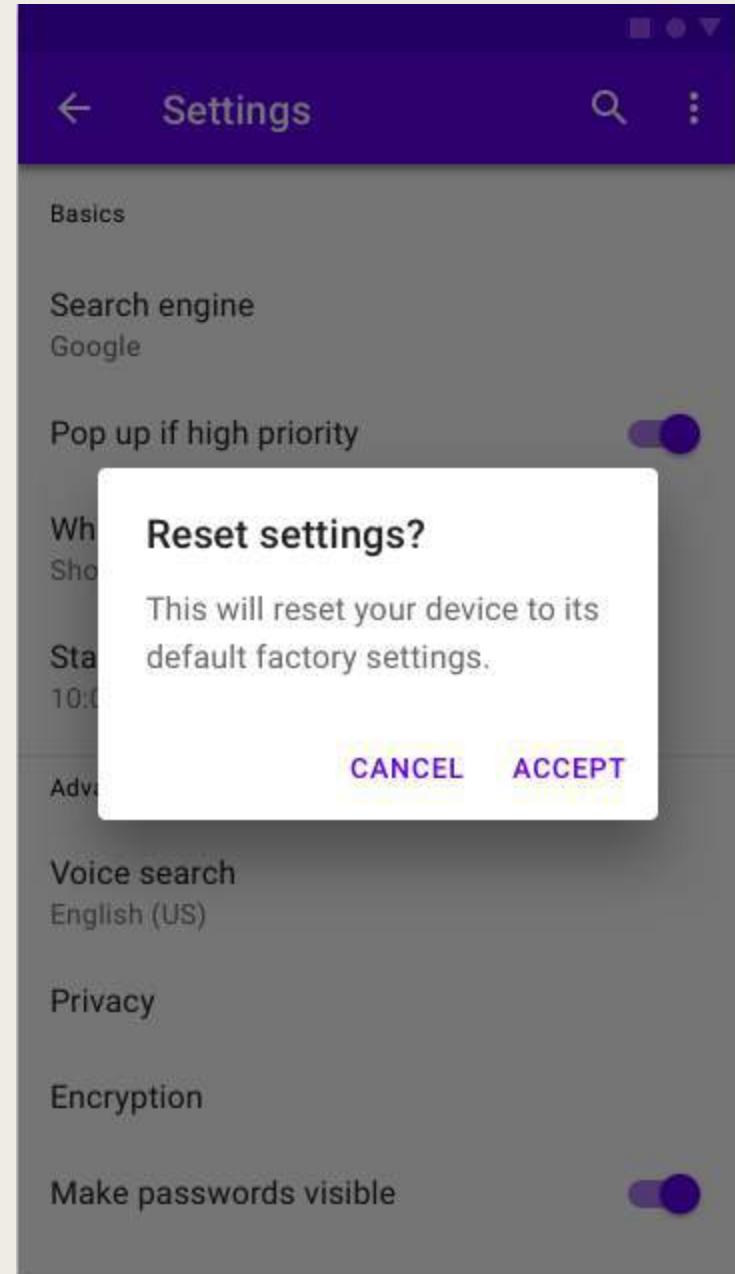
```
13  class _ToastExampleState extends State {
14      void showToast() {
15          fluttertoast.showToast(
16              msg: 'This is toast notification',
17              toastLength: Toast.LENGTH_LONG,
18              gravity: ToastGravity.TOP,
19              backgroundColor: Colors.red,
20              textColor: Colors.yellow
21      );
22  }
23
24  @override
25  Widget build(BuildContext context) {
26      return MaterialApp(
27          title: 'Toast Notification Example',
28          home: Scaffold(
29              appBar: AppBar(
30                  title: Text('Toast Notification Example'),
31              ), // AppBar
32              body: Padding(
33                  padding: EdgeInsets.all(15.0),
34                  child: Center(
35                      child: RaisedButton(
36                          child: Text('click to show'),
37                          onPressed: showToast,
38                      ), // RaisedButton
39                  ), // Center
40              ), // Padding
41          ), // Scaffold
42  }
```



Alert Dialog

- An alert dialog is a useful tool that alerts the app's user.
- It is a pop up in the middle of the screen which places an overlay over the background.
- Most commonly, it is used to confirm one of the user's potentially unrevertable actions.
- First, you must call the `showDialog()` function, which alters the app's state to show a dialog.

```
void _showDialog() {
    showDialog(
        context: context,
        builder: (BuildContext context) {
            return AlertDialog();
        },
    );
}
```



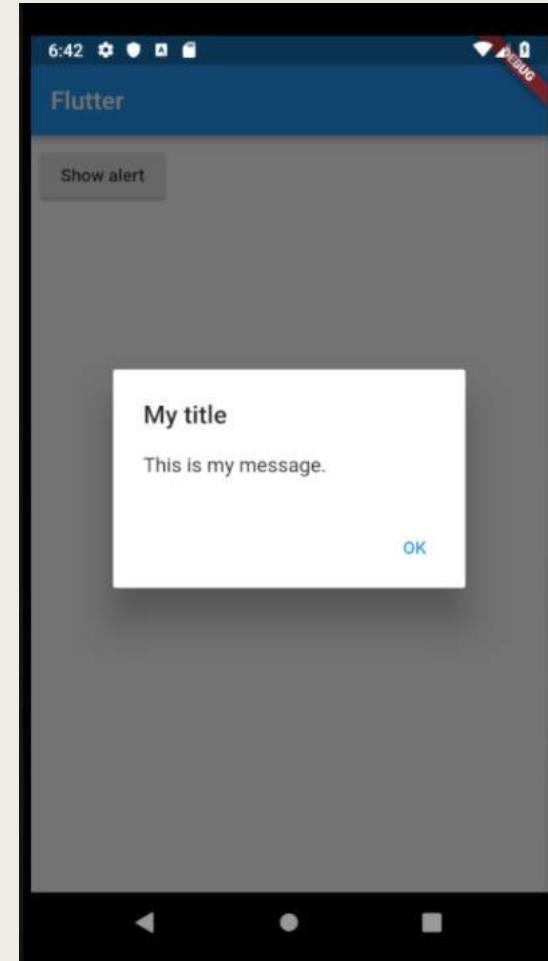
Alert Dialog

- In alert dialog widget, various attributes can be set, including title, content, and actions.
- Actions are where you put the buttons, usually at the bottom of the dialog. The actions field takes in a list of widgets, therefore allowing you to put as many action widgets as you like.

```
void _showDialog() {  
    showDialog(  
        context: context,  
        builder: (BuildContext context) {  
            return AlertDialog(  
                title: new Text("Alert Dialog title"),  
                content: new Text("Alert Dialog body"),  
                actions: <Widget>[  
                    new FlatButton(  
                        child: new Text("Close"),  
                        onPressed: () {  
                            Navigator.of(context).pop();  
                        },  
                    ), // FlatButton  
                ], // <Widget>[]  
            ); // AlertDialog  
        },  
    );  
}
```

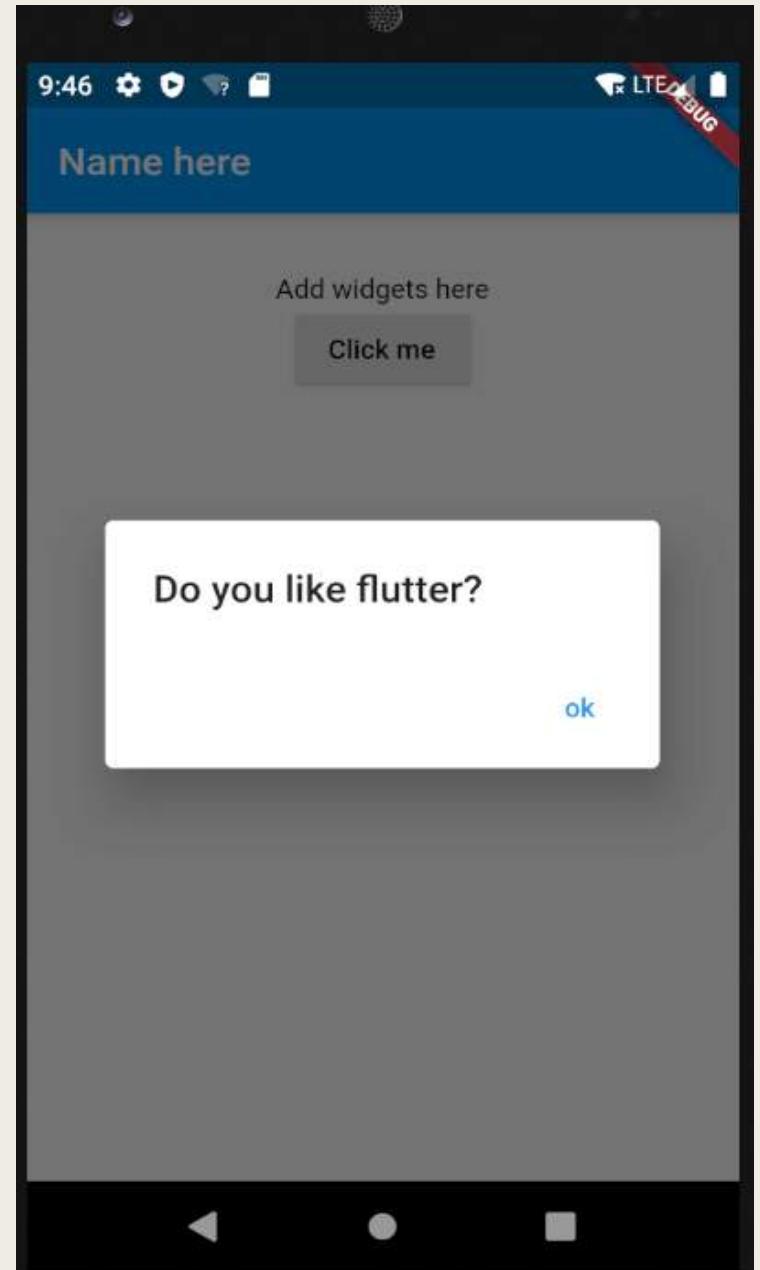
Alert Dialog

```
showAlertDialog(BuildContext context) {  
  
    // set up the button  
    Widget okButton = FlatButton(  
        child: Text("OK"),  
        onPressed: () { },  
    );  
  
    // set up the AlertDialog  
    AlertDialog alert = AlertDialog(  
        title: Text("My title"),  
        content: Text("This is my message."),  
        actions: [  
            okButton,  
        ],  
    );  
  
    // show the dialog  
    showDialog(  
        context: context,  
        builder: (BuildContext context) {  
            return alert;  
        },  
    );  
}
```



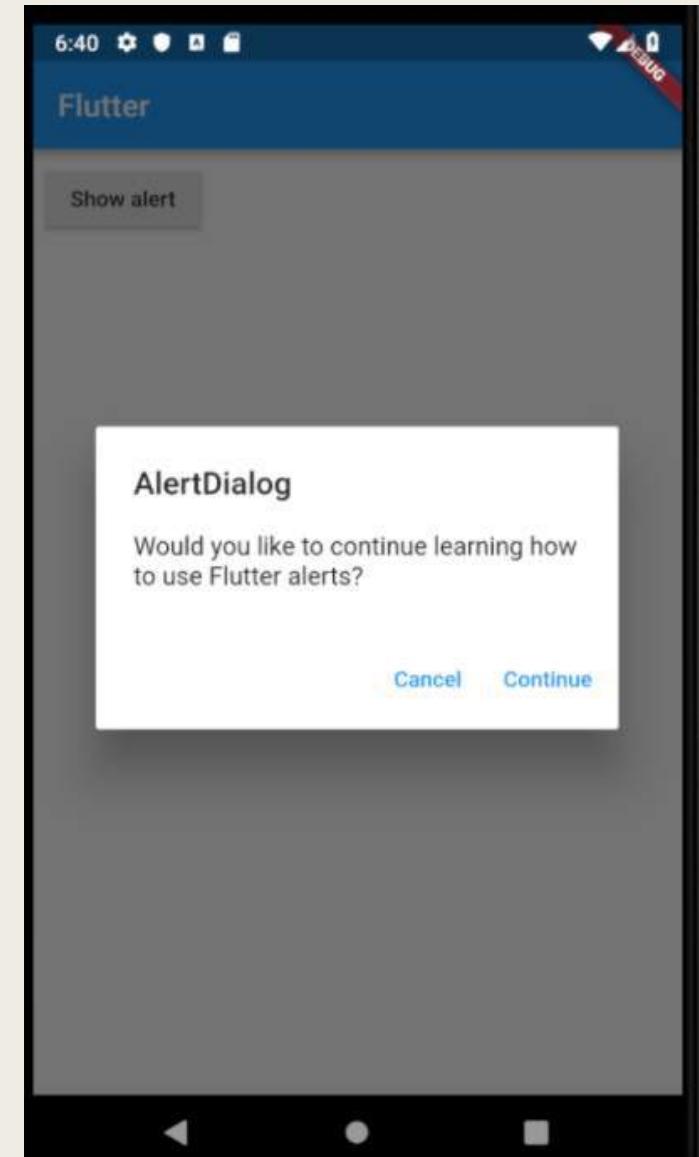
Alert Dialog

```
15  class _State extends State<MyApp>
16  {
17      Future _showAlert(BuildContext context, String message)async{
18          return showDialog(
19              context: context,
20              child: new AlertDialog(
21                  title: new Text(message),
22                  actions:<Widget>[
23                      new FlatButton(onPressed: ()>Navigator.pop(context), child: new Text('ok'))
24                  ] // <Widget>[]
25              ) // AlertDialog
26          );
27      }
28      @override
29      Widget build(BuildContext context){
30          return new Scaffold(
31              appBar:new AppBar(
32                  title: new Text('Name here'),
33              ), // AppBar
34              body: new Container(
35                  padding: new EdgeInsets.all(32.0),
36                  child: new Center(
37                      child: new Column(<Widget>[
38                          new Text('Add widgets here'),
39                          new RaisedButton(onPressed: ()> _showAlert(context,'Do you like flutter?'),
40                              child: new Text("Click me")) // RaisedButton
41                      ], // <Widget>[])
42                  )
43              )
44          );
45      }
46  }
```



Alert Dialog with Two Buttons

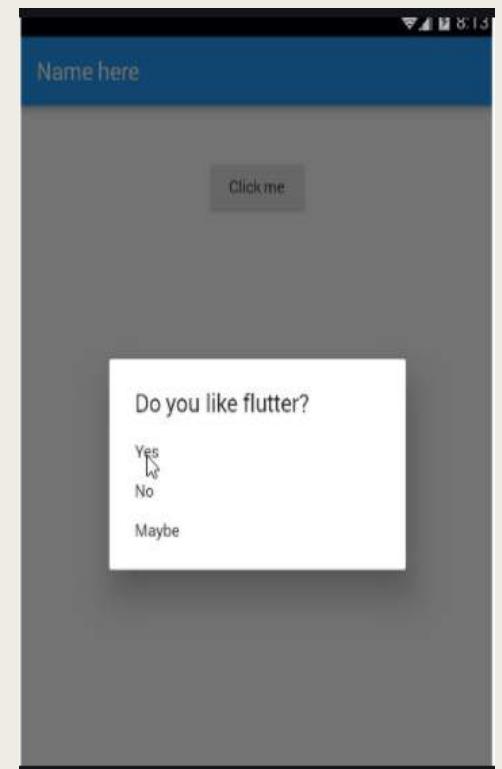
```
showAlertDialog(BuildContext context) {  
  
    // set up the buttons  
    Widget cancelButton = FlatButton(  
        child: Text("Cancel"),  
        onPressed: () {},  
    );  
    Widget continueButton = FlatButton(  
        child: Text("Continue"),  
        onPressed: () {},  
    );  
  
    // set up the AlertDialog  
    AlertDialog alert = AlertDialog(  
        title: Text("AlertDialog"),  
        content: Text("Would you like to continue learning how to use  
Flutter alerts?"),  
        actions: [  
            cancelButton,  
            continueButton,  
        ],  
    );  
  
    // show the dialog  
    showDialog(  
        context: context,  
        builder: (BuildContext context) {  
            return alert;  
        },  
    );  
}
```



Simple Dialog

- A simple dialog offers the user a choice between several options.
- A simple dialog has an optional title that is displayed above the choices.
- Choices are normally represented using **SimpleDialogOption** widgets.
- Typically passed as the child widget to **showDialog**, which displays the dialog.
- For dialogs that inform the user about a situation, consider using an AlertDialog

```
1 SimpleDialog({  
2   Key key,  
3   this.title,  
4   this.titlePadding = const EdgeInsets.fromLTRB(24.0, 24.0, 24.0, 0.0),  
5   this.children,  
6   this.contentPadding = const EdgeInsets.fromLTRB(0.0, 12.0, 0.0, 16.0),  
7   this.backgroundColor,  
8   this.elevation,  
9   this.semanticLabel,  
10  this.shape,  
11})
```

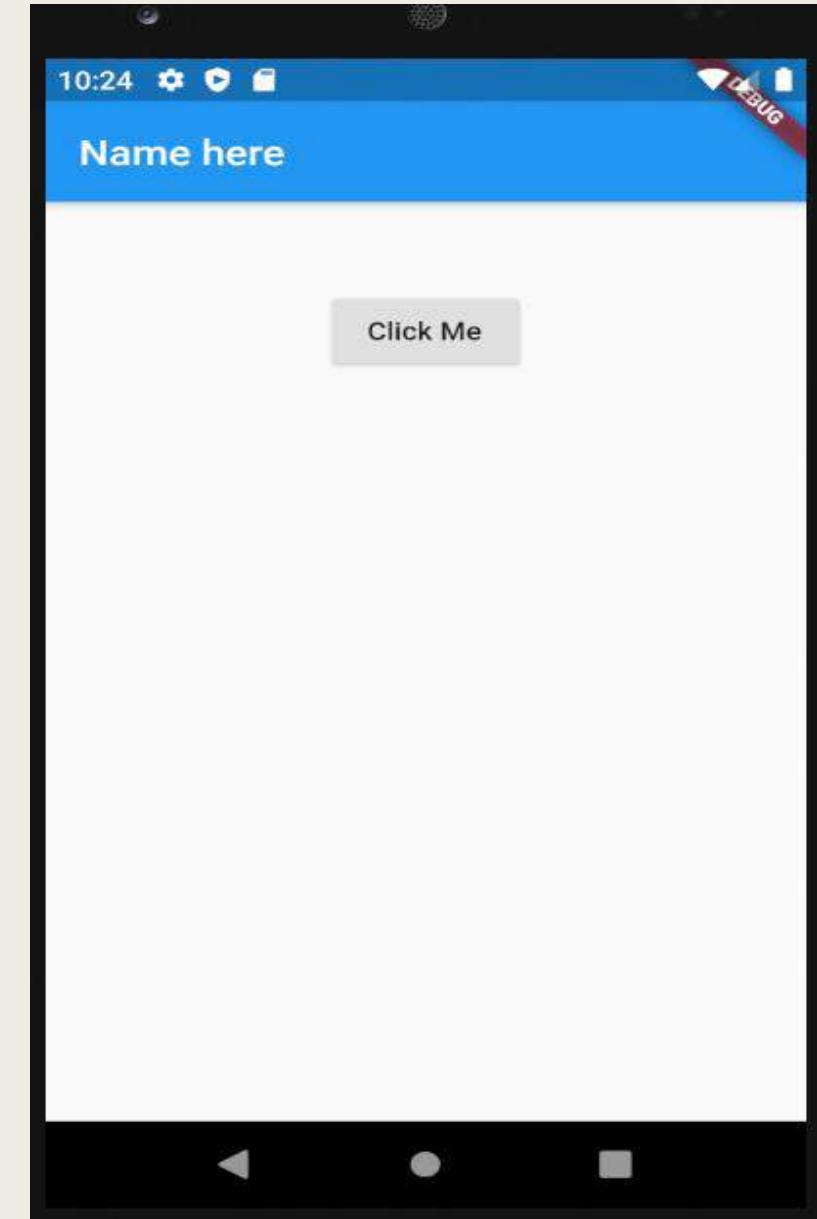


Simple Dialog

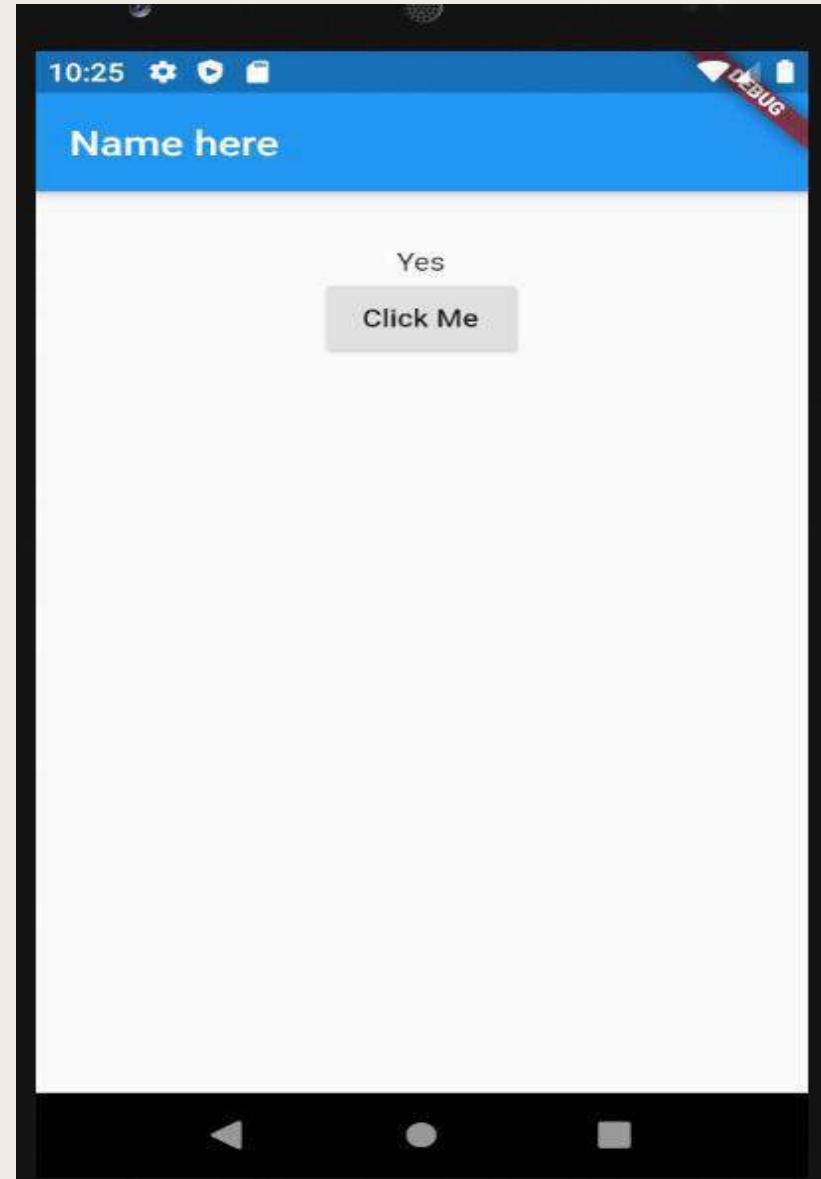
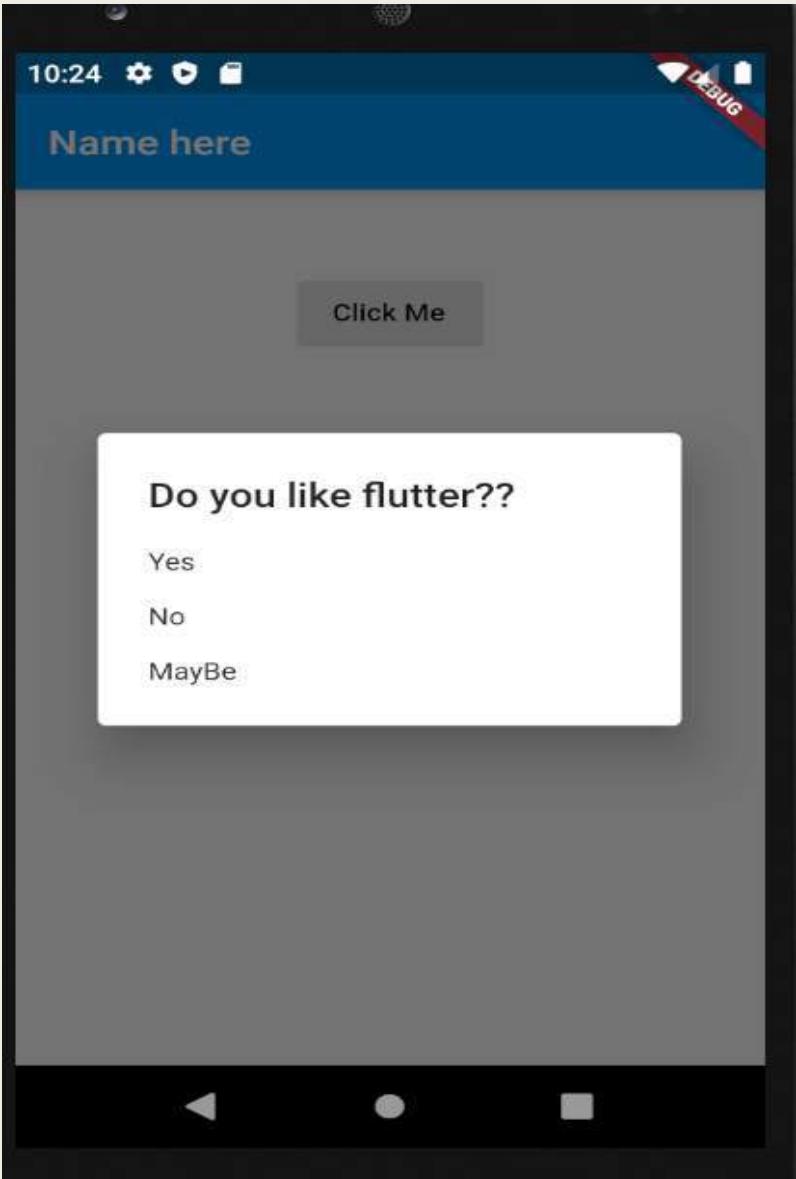
```
15 enum Answers{YES, NO, MAYBE}
16 class _State extends State<MyApp>
17 {
18     String _value = '';
19     void _setValue(String value)=>setState(() => _value=value);
20     Future _askUser() async{
21         switch(
22             await showDialog(
23                 context:context,
24                 child: new SimpleDialog(
25                     title: new Text("Do you like flutter??"),
26                     children:<Widget>[
27                         new SimpleDialogOption(child: new Text('Yes'),onPressed: (){Navigator.pop(context,Answers.YES)}
28                         new SimpleDialogOption(child: new Text('No'),onPressed: (){Navigator.pop(context,Answers.NO);}
29                         new SimpleDialogOption(child: new Text('MayBe'),onPressed: (){Navigator.pop(context,Answers.MAYBE)})
30                     ],
31                 ) // SimpleDialog
32             )
33         {
34             case Answers.YES:
35                 _setValue('Yes');
36                 break;
37             case Answers.NO:
38                 _setValue('No');
39                 break;
40             case Answers.MAYBE:
41                 _setValue('MayBe');
42                 break;
43         }
44     }
}
```

Simple Dialog(contd..)

```
45  @override
46  Widget build(BuildContext context){
47    return new Scaffold(
48      appBar: new AppBar(
49        title: new Text('Name here'),
50      ), // AppBar
51      body: new Container(
52        padding: new EdgeInsets.all(32.0),
53        child: new Center(
54          child: new Column(children: <Widget>[
55            new Text(_value),
56            new RaisedButton(onPressed: _askUser, child: new Text("Click Me"))
57          ], // <Widget>[]
58        ) // Column
59      ), // Center
60    ), // Container
61  ); // Scaffold
62 }
63 }
64 }
```



Simple Dialog(contd..)

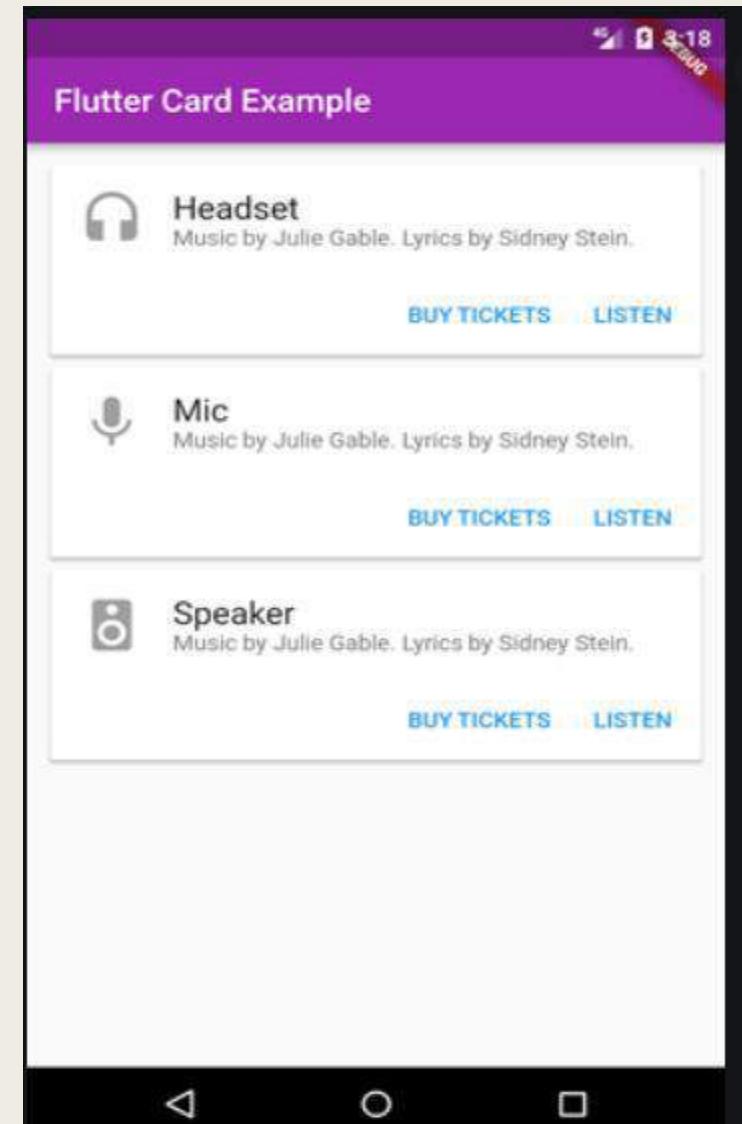


Some Additional Widgets

- Card
- Stack
- Expanded
- Wrap

Card Widget

- A card is a sheet used to represent the information related to each other, such as an album, a geographical location, contact details, etc.
- ***A card in Flutter is in rounded corner shape and has a shadow.***
- We mainly used it to store the content and action of a single object.
- Use **card constructor** and then pass a widget as child property for displaying the content and action inside the card.



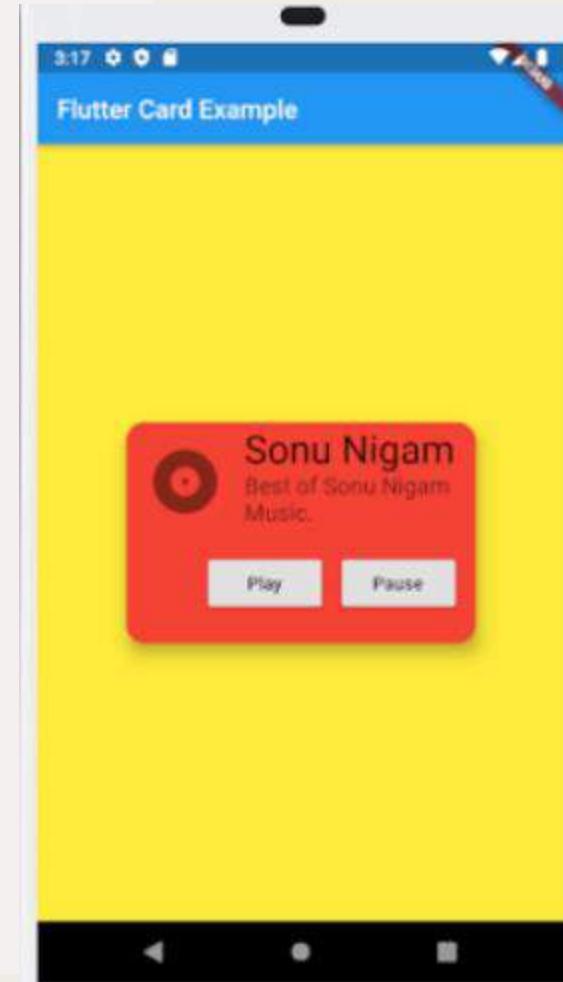
Card Widget

Attribute Name	Descriptions
borderOnForeground	It is used to paint the border in front of a child. By default, it is true. If it is false, it painted the border behind the child.
color	It is used to color the card's background.
elevation	It controls the shadow size below the card. The bigger elevation value makes the bigger shadow distance.
margin	It is used to customize the card's outer space.
shape	It is used to specify the shape of the card.
shadowColor	It is used to paint the shadow of a card.
clipBehavior	It is used to clip the content of the card.

Card Widget

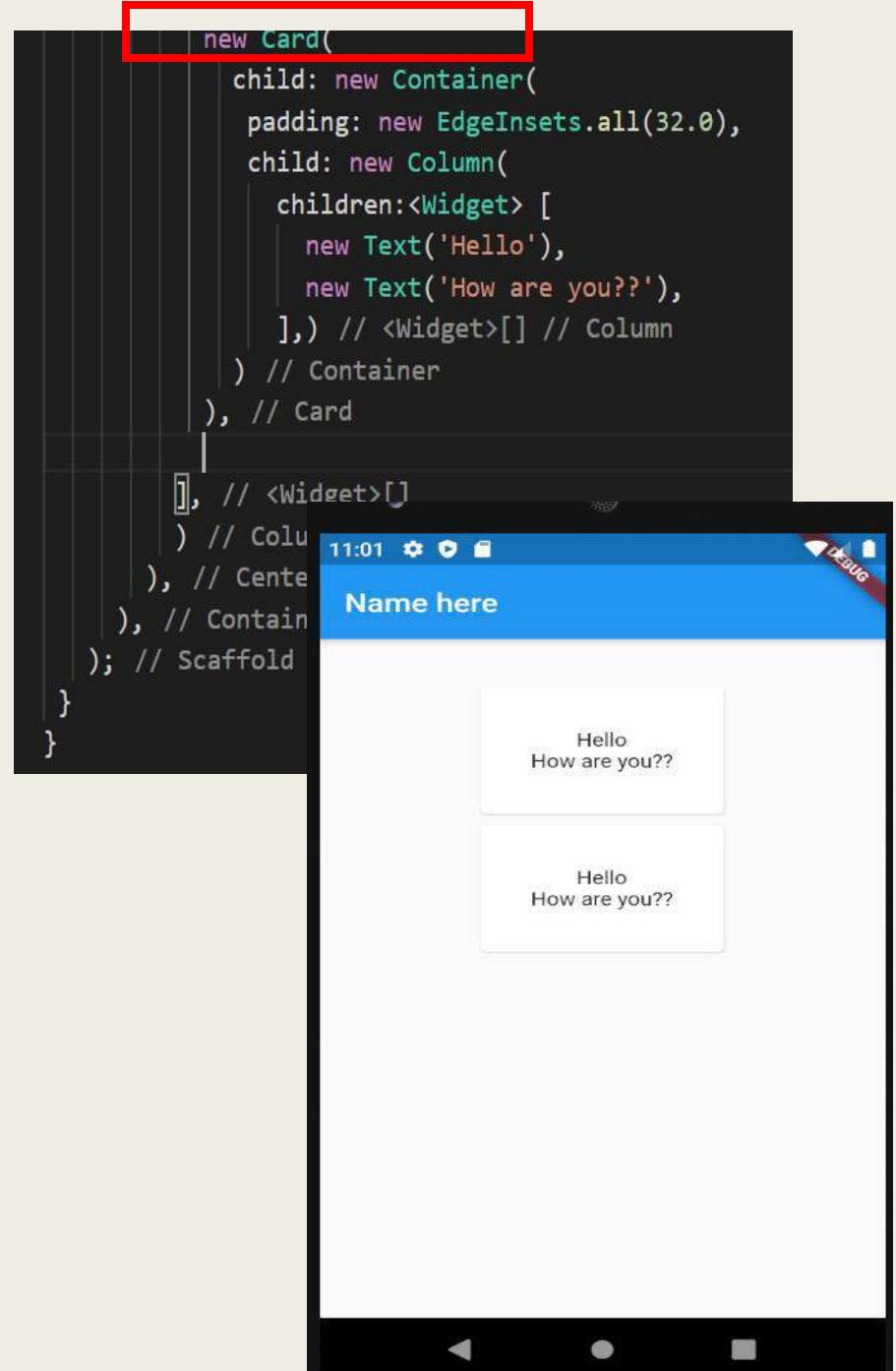
```
@override  
Widget build(BuildContext context) {  
  return Center(  
    child: Container(  
      width: 300,  
      height: 200,  
      padding: new EdgeInsets.all(10.0),  
      child: Card(  
        shape: RoundedRectangleBorder(  
          borderRadius: BorderRadius.circular(15.0),  
        ),  
        color: Colors.red,  
        elevation: 10,  
        child: Column(  
          mainAxisSize: MainAxisSize.min,  
          children: <Widget>[  
            const ListTile(  
              leading: Icon(Icons.album, size: 60),  
              title: Text(  
                'Sonu Nigam',  
                style: TextStyle(fontSize: 30.0)  
              ),  
              subtitle: Text(  
                'Best of Sonu Nigam Music.',  
                style: TextStyle(fontSize: 18.0)  
              ),  
            ),  
          ],  
        ),  
      ),  
    ),  
  );  
}
```

```
ButtonBar(  
  children: <Widget>[  
    RaisedButton(  
      child: const Text('Play'),  
      onPressed: () {/* ... */},  
    ),  
    RaisedButton(  
      child: const Text('Pause'),  
      onPressed: () {/* ... */},  
    ),  
  ],  
,  
),  
],  
,
```



Card Widget

```
15  class _State extends State<MyApp>
16  {
17      @override
18      Widget build(BuildContext context){
19          return new Scaffold(
20              appBar:new AppBar(
21                  title: new Text('Name here'),
22              ), // AppBar
23              body: new Container(
24                  padding: new EdgeInsets.all(32.0),
25                  child: new Center(
26                      child: new Column(children: <Widget>[
27                          new Card(
28                              child: new Container(
29                                  padding: new EdgeInsets.all(32.0),
30                                  child: new Column(
31                                      children:<Widget> [
32                                          new Text('Hello'),
33                                          new Text('How are you??'),
34                                      ],) // <Widget>[] // Column
35                                  ) // Container
36                          ), // Card
37                      ],
38                  ),
39              );
40      }
41  }
```

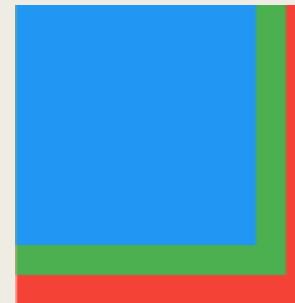


Stack Widget

- Stack Widget contains list of widgets and it place them one on top the other.
- As a result while rendering, It renders them from the ground up position – bottom to top.
- Size of the Stack depends on it's child widgets.

Stack size will be same as the max size occupied by one of its child.

```
Stack(  
  children: <Widget>[  
    Container(  
      width: 100,  
      height: 100,  
      color: Colors.red,  
    ),  
    Container(  
      width: 90,  
      height: 90,  
      color: Colors.green,  
    ),  
    Container(  
      width: 80,  
      height: 80,  
      color: Colors.blue,  
    ),  
  ],  
)
```

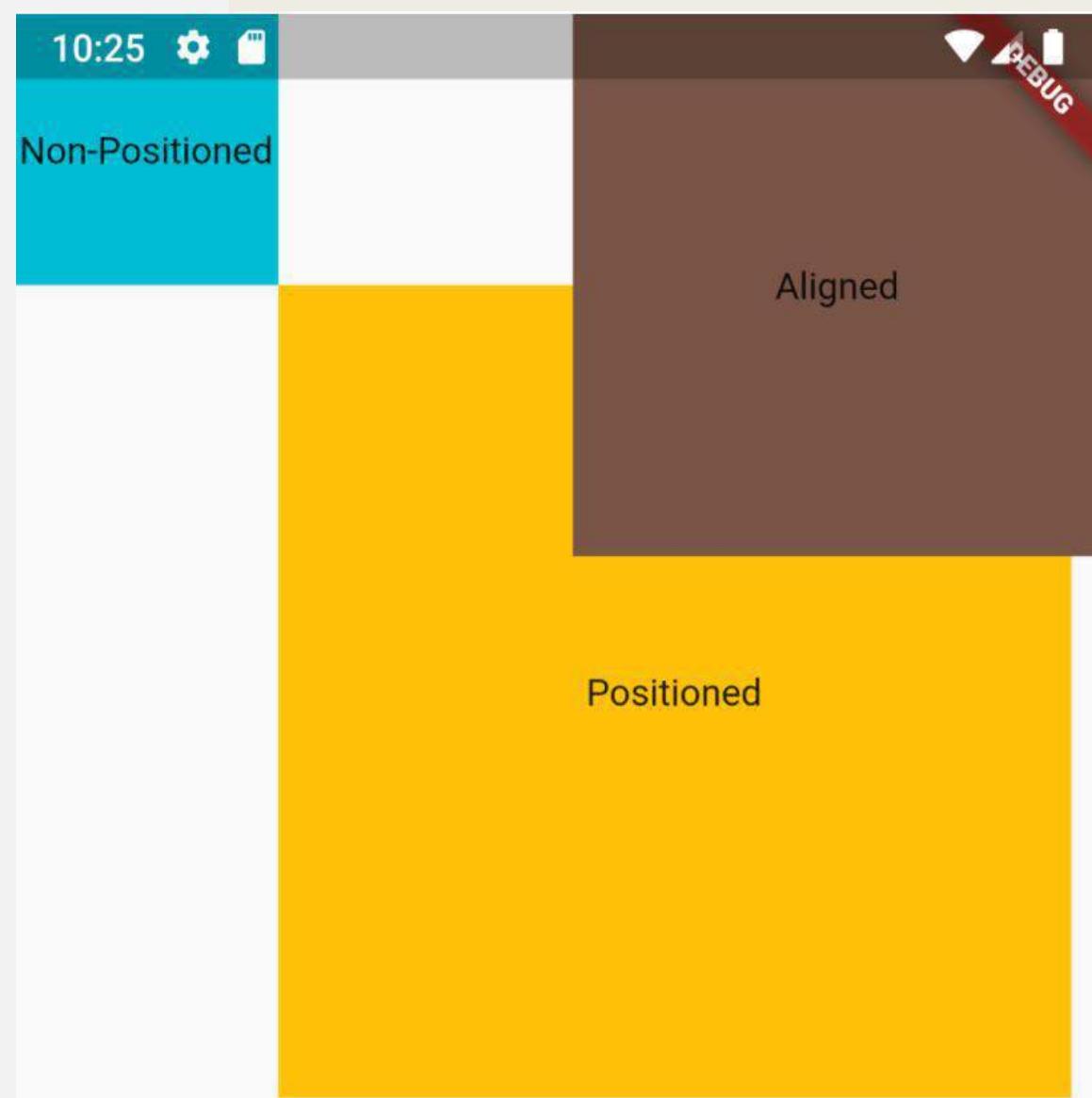


Stack Widget

- *Each member of a Stack widget is either positioned or non-positioned.*
- **Positioned Widget :**
 - *A child of Stack wrapped with Positioned Widget.*
 - *It works with a combination of parameters - vertical (top, bottom, height) and horizontal (left, right and width) to position the widgets within the Stack.*
 - *Note: If not positioned widget, Align Widget is used to position the member of Stack.*
- **Non-positioned Widget :**
 - *If Stack's member is not wrapped with Align or Positioned Widget, then it considered as Non-positioned widget.*
 - *Non-positioned widgets ends up in the screen based on Stack's alignment property. By default, top left corner on the screen.*

Stack Widget(with Positioned and non positioned widgets)

```
Stack(  
  alignment: Alignment.topLeft,  
  children: <Widget>[  
    Positioned(  
      top: 100,  
      left: 100,  
      child: Container(  
        height: 300,  
        width: 300,  
        child: Center(child: Text('Positioned')),  
        color: Colors.amber,  
      ),  
    ),  
    Align(  
      alignment: Alignment.topRight,  
      child: Container(  
        height: 200,  
        width: 200,  
        child: Center(child: Text('Aligned')),  
        color: Colors.brown,  
      ),  
    ),  
    Container(  
      height: 100,  
      width: 100,  
      child: Center(child: Text('Non-Positioned')),  
      color: Colors.cyan,  
    ),  
  ],
```



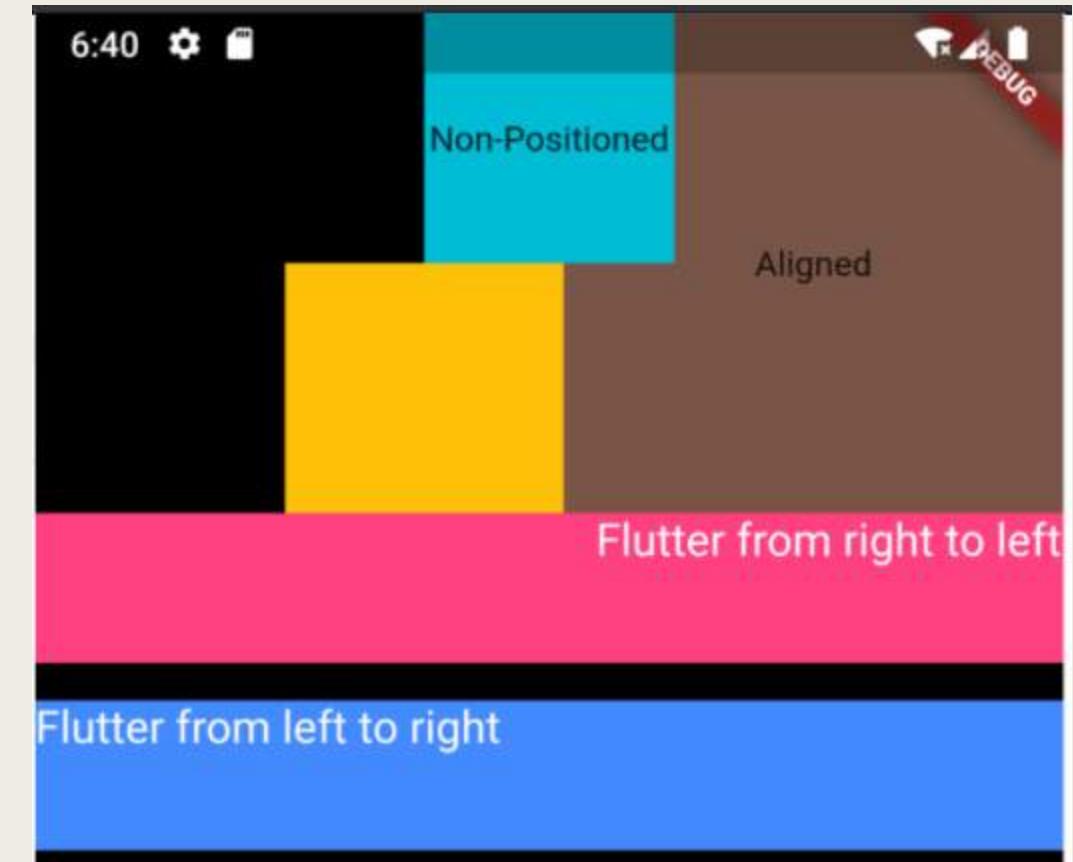
Stack Widget Constructor

- **alignment**: It basically determines the placement of Non-positioned Widget available in Stack
- **textDirection**: One can change direction of text, whether it should flow from left to right (LTR) or right to left (RTL). Which becomes quite handy while working with multiple languages.
- **fit**: It determines the size of Non-positioned Widget available in Stack.
 - **StackFit.loose**: *non-positioned children of the stack can have any width from 0 to 300 and height from 0 to 400.*
 - **StackFit.expand**: *non-positioned children of the stack will have highest width (i.e 300) and also highest height (i.e 400).*
 - **StackFit.passThrough**: *The constraints passed to the stack from its parent will remain unmodified for the non-positioned children.*
- **overflow**: To manages whether overflowing content of child widget should be clipped, or remain as it is, i.e visible.

```
Stack({  
    this.alignment = Alignment.topCenter,  
    this.textDirection,  
    this.fit = StackFit.loose,  
    this.overflow = Overflow.clip,  
    List<Widget> children = const <Widget>[],  
})
```

Stack Widget textDirection Property

```
Container(  
  color: Colors.pinkAccent,  
  constraints: BoxConstraints.expand(height: 60),  
  child: Stack(  
    textDirection: TextDirection rtl,  
    children: <Widget>[  
      Text(  
        "Flutter from right to left",  
        style: TextStyle(color: Colors.white, fontsize: 18),  
      ),  
    ],  
  ),  
,  
  
//textDirection => left to right...  
  
Container(  
  margin: EdgeInsets.only(top: 15),  
  color: Colors.blueAccent,  
  constraints: BoxConstraints.expand(height: 60),  
  child: Stack(  
    textDirection: TextDirection.ltr,  
    children: <Widget>[  
      Text(  
        "Flutter from left to right",  
        style: TextStyle(color: Colors.white, fontsize: 18),  
      ),  
    ],  
  ),  
,
```



Stack Widget fit Property

```
Container(  
    margin: EdgeInsets.only(top: 15),  
    constraints: BoxConstraints.expand(height: 60),  
    color: Colors.deepOrangeAccent,  
    child: Stack(  
        fit: StackFit.loose,  
        children: <Widget>[  
            Container(  
                color: Colors.brown,  
                child: Text(  
                    "Loose StackFit",  
                    style: TextStyle(color: Colors.white, fontSize: 18),  
                ),  
            ),  
        ],  
    ),  
,  
  
// StackFit.expand  
  
Container(  
    margin: EdgeInsets.only(top: 15),  
    constraints: BoxConstraints.expand(height: 60),  
    color: Colors.black,  
    child: Stack(  
        fit: StackFit.expand,  
        children: <Widget>[  
            Container(  
                color: Colors.purpleAccent,  
                child: Text(  
                    "Expanded StackFit",  
                    style: TextStyle(color: Colors.white, fontSize: 18),  
                ),  
            ),  
        ],  
    ),  
,
```

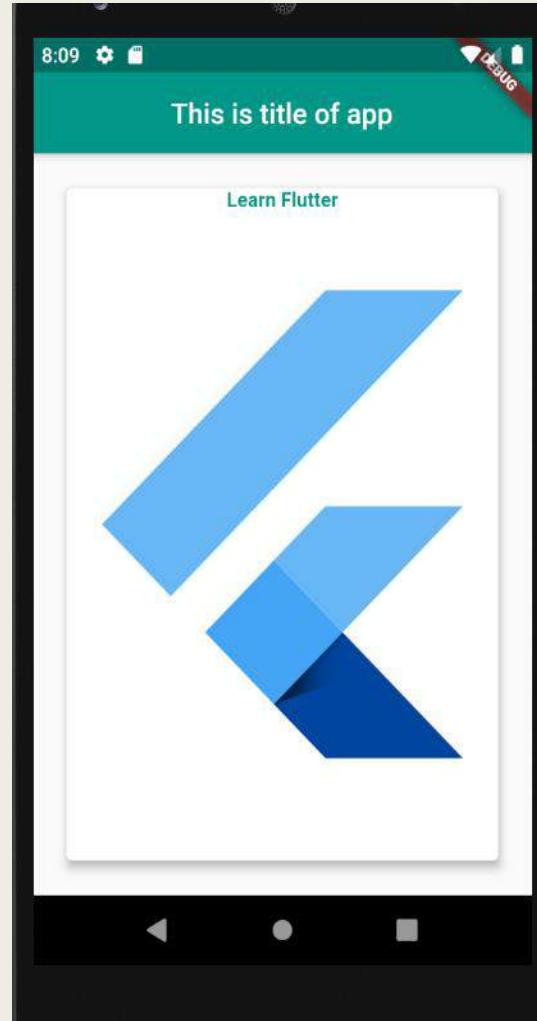


Stack Widget overflow Property



Stack Widget

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('This is title of app'),
          backgroundColor: Colors.teal,
          centerTitle: true,
        ), // AppBar
        body: Stack(
          alignment: AlignmentDirectional.center,
          fit: StackFit.expand,
          children: [
            Padding(padding: const EdgeInsets.all(20.0),
            child: Card(
              elevation: 5.0,
              child: Text('Learn Flutter', textAlign: TextAlign.center,
              style: TextStyle(color: Colors.teal, fontWeight: FontWeight.bold)
            ) // Text
          ), // Card
          ), // Padding
          FlutterLogo(),
        ],
      ) // Stack
    ), // Scaffold
  ); // MaterialApp
}
```

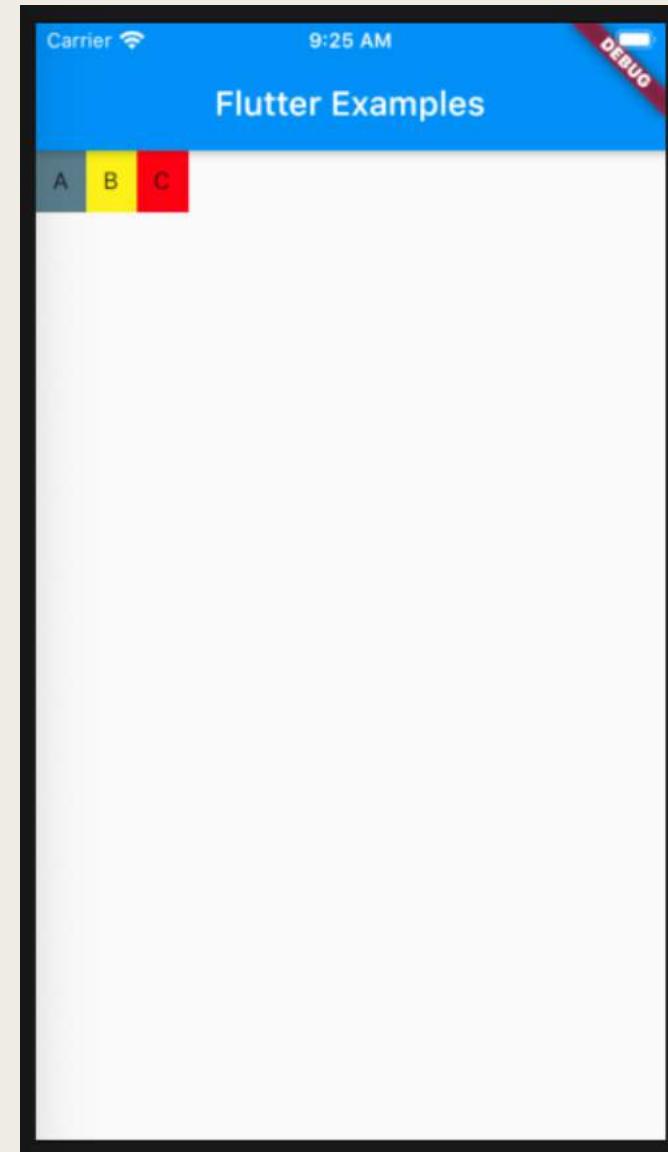


Expanded Widget

- Using an Expanded widget makes a child of a Row or Column (also for Flex) expand to fill the available space in the main axis (horizontally for a Row or vertically for a Column).
- If multiple children are expanded, the available space is divided among them according to the flex factor.

Expanded Widget (not applied)

```
@override  
Widget build(BuildContext context) {  
  return Row(  
    children: <Widget>[  
      Container(  
        padding: EdgeInsets.all(10),  
        color: Colors.blueGrey,  
        child: Text('A'),  
      ), // Container  
      Container(  
        padding: EdgeInsets.all(10),  
        color: Colors.yellow,  
        child: Text('B'),  
      ), // Container  
      Container(  
        padding: EdgeInsets.all(10),  
        color: Colors.red,  
        child: Text('C'),  
      ) // Container  
    ], // <Widget>[]  
  ); // Row  
}  
}
```



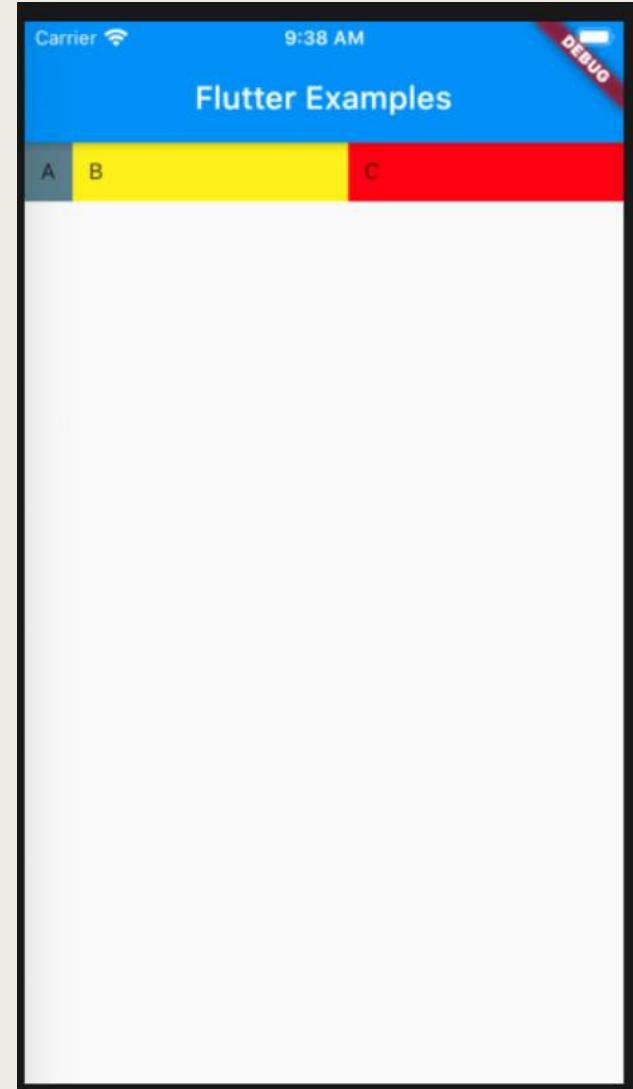
Expanded Widget

```
@override  
Widget build(BuildContext context) {  
  return Row(  
    children: <Widget>[  
      Container(  
        padding: EdgeInsets.all(10),  
        color: Colors.blueGrey,  
        child: Text('A'),  
      ), // Container  
      Container(  
        padding: EdgeInsets.all(10),  
        color: Colors.yellow,  
        child: Text('B'),  
      ), // Container  
      Expanded(  
        child: Container(  
          padding: EdgeInsets.all(10),  
          color: Colors.red,  
          child: Text('C'),  
        ), // Container  
      ) // Expanded  
    ], // <Widget>[]  
  ); // Row  
}
```



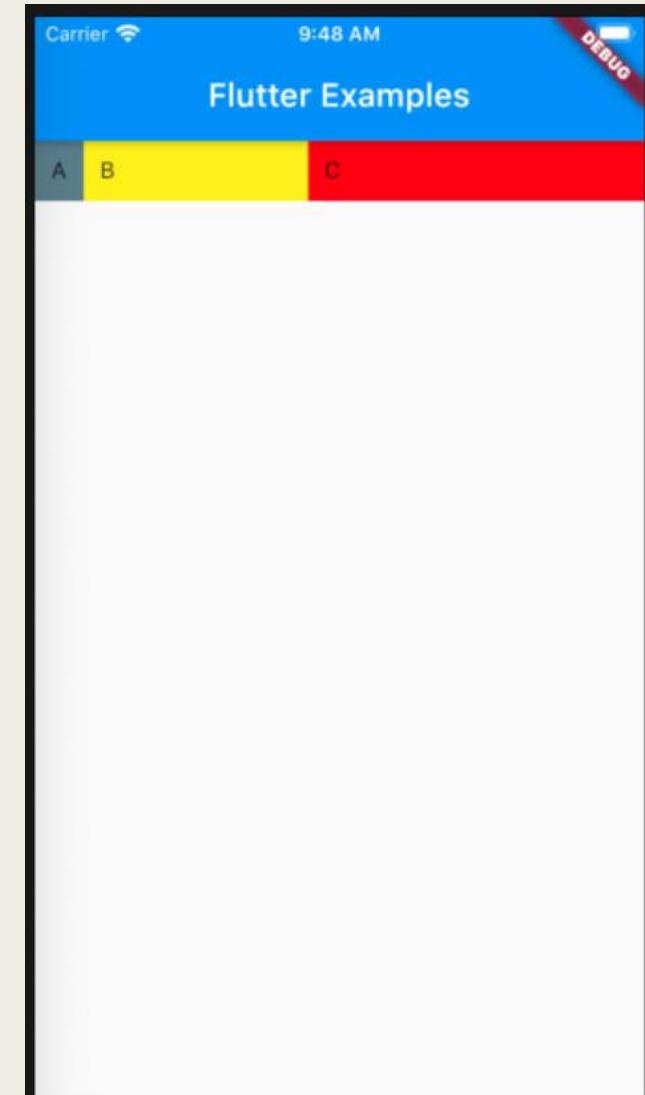
Expanded Widget

```
class Example extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: <Widget>[
        Container(
          padding: EdgeInsets.all(10),
          color: Colors.blueGrey,
          child: Text('A'),
        ), // Container
        Expanded(
          child: Container(
            padding: EdgeInsets.all(10),
            color: Colors.yellow,
            child: Text('B'),
          ), // Container
        ), // Expanded
        Expanded(
          child: Container(
            padding: EdgeInsets.all(10),
            color: Colors.red,
            child: Text('C'),
          ), // Container
        ), // Expanded
      ], // <Widget>[]
    ); // Row
  }
}
```



Expanded Widget with Flex Factor

```
class Example extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Row(  
      children: <Widget>[  
        Container(  
          padding: EdgeInsets.all(10),  
          color: Colors.blueGrey,  
          child: Text('A'),  
        ), // Container  
        Expanded(  
          flex: 2,  
          child: Container(  
            padding: EdgeInsets.all(10),  
            color: Colors.yellow,  
            child: Text('B'),  
          ), // Container  
        ), // Expanded  
        Expanded(  
          flex: 3,  
          child: Container(  
            padding: EdgeInsets.all(10),  
            color: Colors.red,  
            child: Text('C'),  
          ), // Container  
        ), // Expanded  
      ], // <Widget>[]  
    ); // Row  
  }  
}
```



Wrap Widget

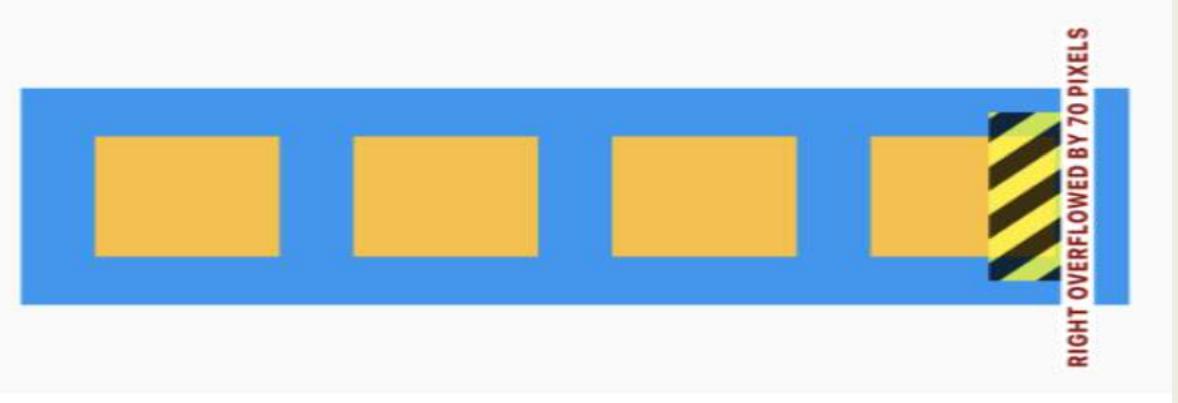
- **Wrap widgets** automatically align the Widgets items one by one and if the Row is filled then it will move row content automatically to next line in flutter
- Wrap widgets can support both Horizontal alignment and Vertical Alignment like Row and Column widgets in flutter.
- Using the Wrap widget we can adjust multiple widgets at once without disturbing each other.

```
Row(  
  children: [  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
  ],  
) ,
```



Wrap Widget

```
Row(  
  children: [  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
  ],  
) ,
```

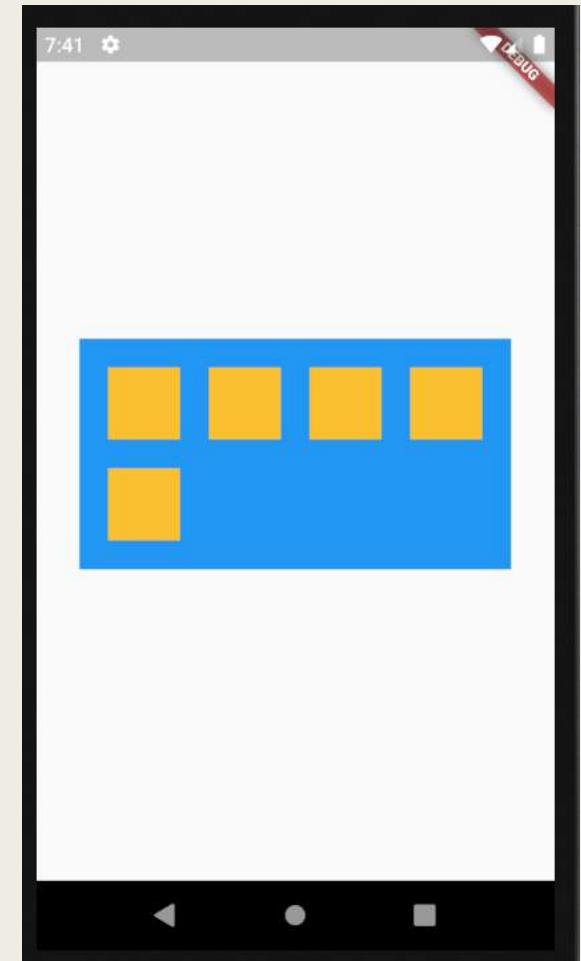


```
Wrap(  
  children: [  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
  ],  
) ,
```



```
1 import 'package:flutter/material.dart';
Run | Debug
2 void main() => runApp(MyApp());
3
4 class MyApp extends StatelessWidget {
5     @override
6     Widget build(BuildContext context) {
7         return MaterialApp(
8             home: Scaffold(
9                 body: WrapWidgetDemo(),
10            ), // Scaffold
11        ); // MaterialApp
12    }
13}
14
15 class WrapWidgetDemo extends StatelessWidget {
16     @override
17     Widget build(BuildContext context) {
18         return Center(
19             child: Container(
20                 width: 300,
21                 color: Colors.blue,
22                 margin: EdgeInsets.all(10),
23                 padding: EdgeInsets.all(10),
24                 child: Wrap(
25                     children: [
26                         MyWidget(),
27                         MyWidget(),
28                         MyWidget(),
29                         MyWidget(),
30                         MyWidget(),
31                     ],
32                 ), // Wrap
33             ), // Container
34         ); // Center
35     }
36 }
```

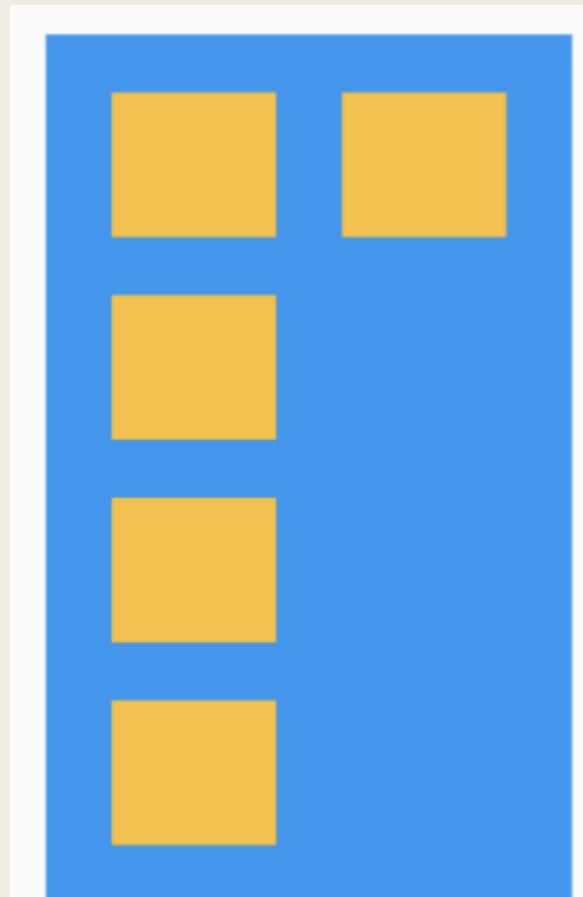
```
38 class MyWidget extends StatelessWidget {  
39     //final String text;  
40     //MyWidget(this.text);  
41     @override  
42     Widget build(BuildContext context) {  
43         return Container(  
44             margin: EdgeInsets.all(10),  
45             width: 50,  
46             height: 50,  
47             color: Colors.yellow.shade700,  
48             //child: Center(child: Text(text)),  
49         ); // Container  
50     }  
51 }
```



Wrap widget with direction property

- **direction:** Used to set items position on screen from where the icons will start printing.(default its horizontal)

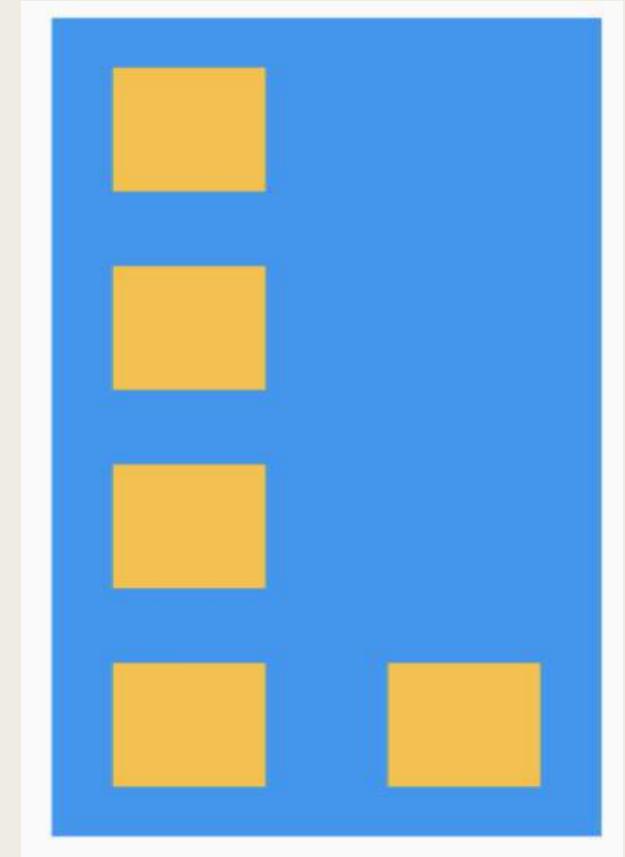
```
Wrap(  
  direction: Axis.vertical,  
  children: [  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
  ],  
) ,
```



Wrap widget with spacing and runSpacing property

- **spacing** is the added space before the next widget.
- **runSpacing** is the added space between rows or columns.

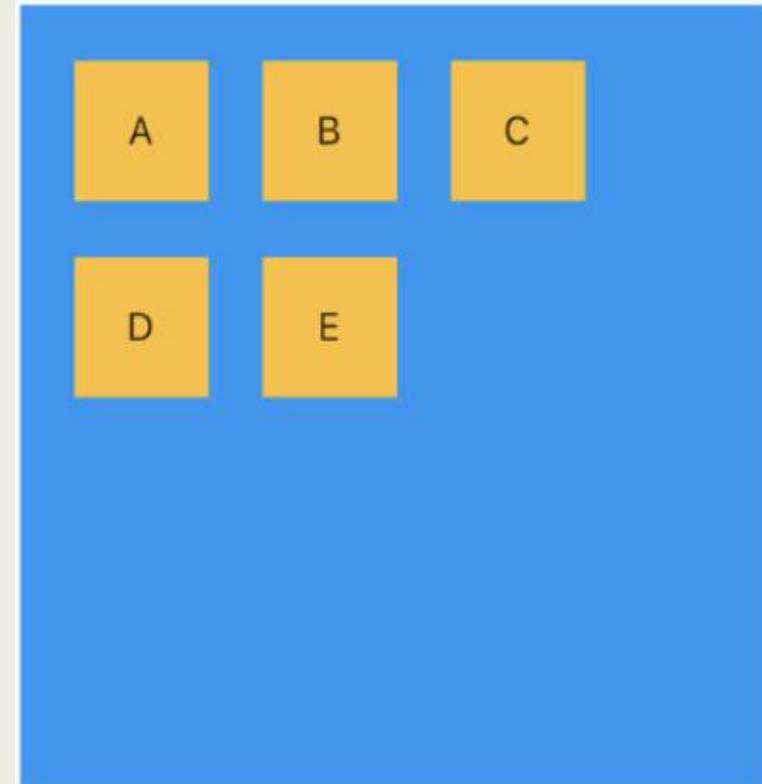
```
Wrap(  
  direction: Axis.vertical,  
  alignment: WrapAlignment.end,  
  spacing: 10.0,  
  runSpacing: 20.0,  
  children: [  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
    MyWidget(),  
  ],  
) ,
```



Wrap widget with alignment property

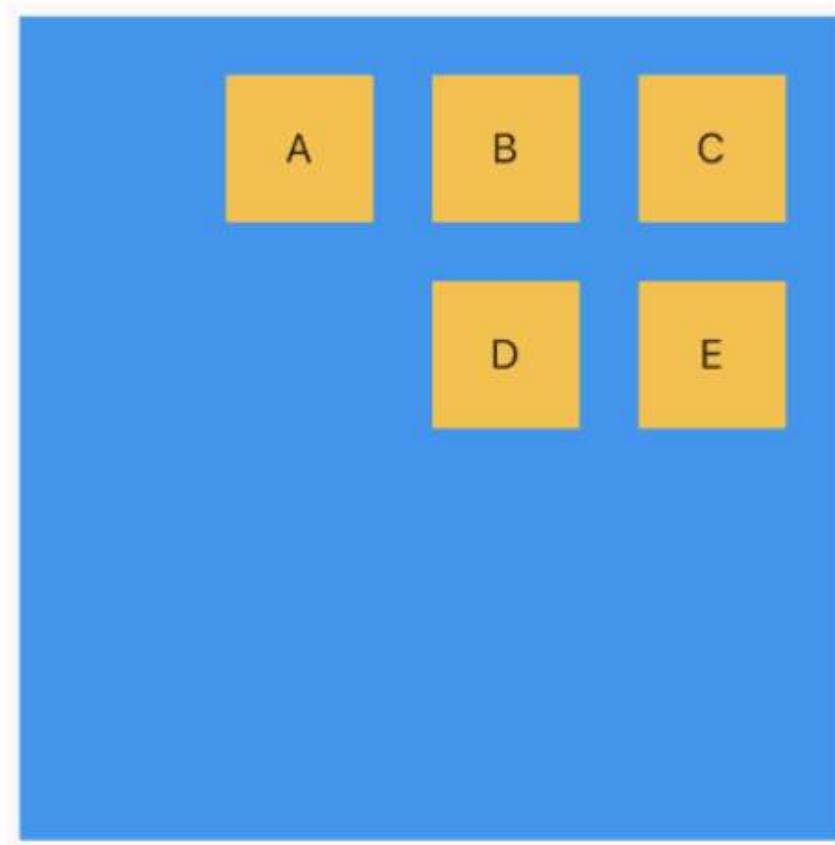
- **alignment:** Can support multiple values like `center`, `end`, `spaceAround`, `spaceBetween`, `start` and `spaceEvenly`.
- alignment affects the placement of the elements

```
Wrap(  
  alignment: WrapAlignment.start, // default  
  children: [  
    MyWidget('A'),  
    MyWidget('B'),  
    MyWidget('C'),  
    MyWidget('D'),  
    MyWidget('E'),  
    //MyWidget('F'),  
    //MyWidget('G'),  
  ],  
,
```

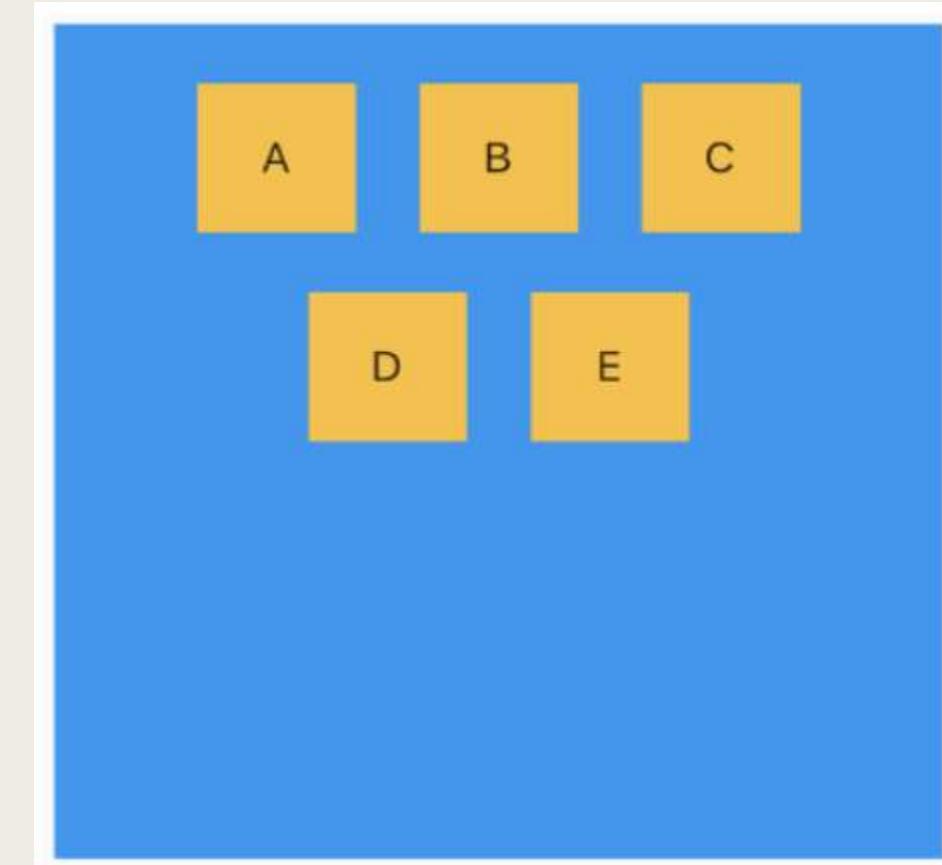


Wrap widget with alignment property

```
alignment: WrapAlignment.end,
```

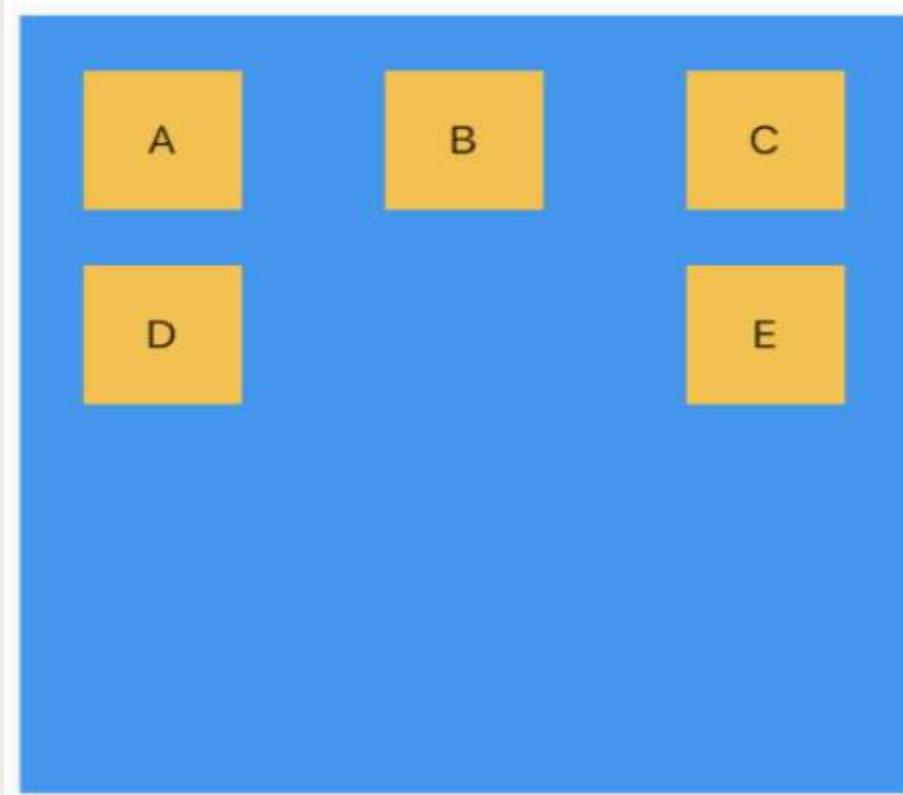


```
alignment: WrapAlignment.center,
```

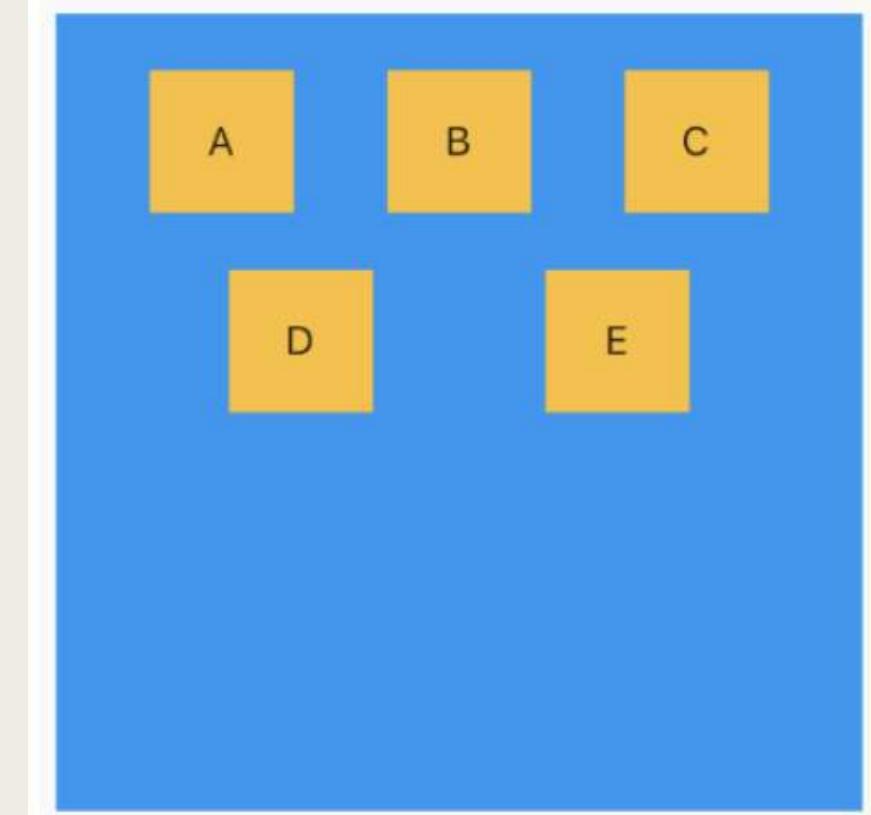


Wrap widget with alignment property

```
alignment: WrapAlignment.spaceBetween,
```



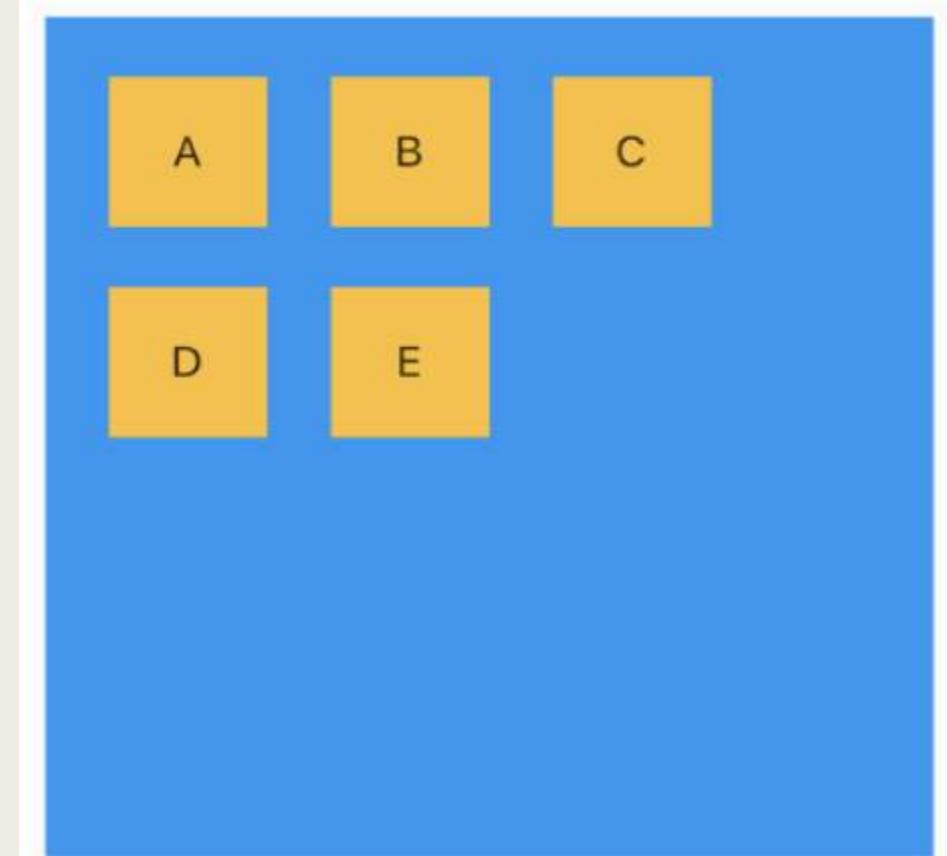
```
alignment: WrapAlignment.spaceEvenly,
```



Wrap widget with runAlignment property

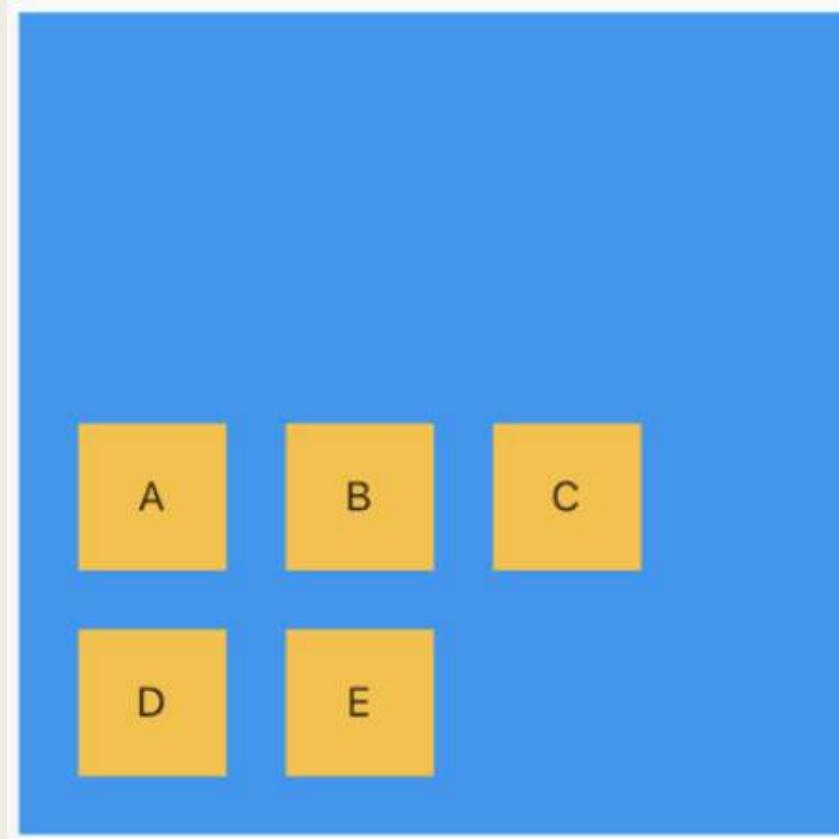
- **run alignment** affects how consecutive rows or columns are aligned.

```
Wrap(  
  runAlignment: WrapAlignment.start, // default  
  children: [  
    MyWidget('A'),  
    MyWidget('B'),  
    MyWidget('C'),  
    MyWidget('D'),  
    MyWidget('E'),  
  ],  
,
```

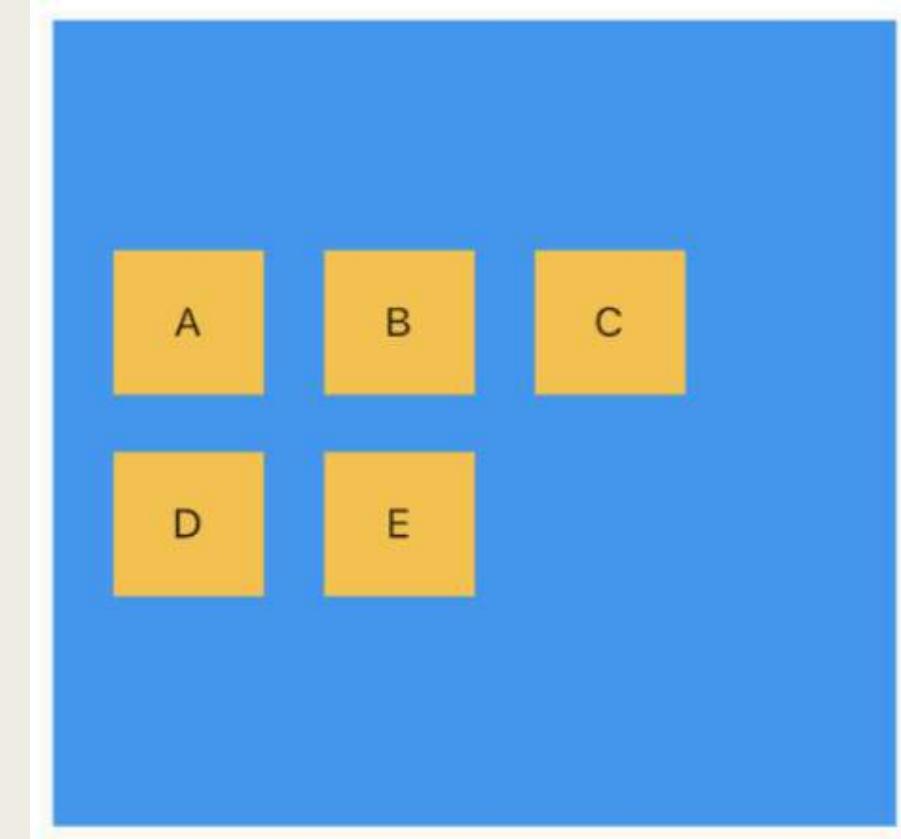


Wrap widget with runAlignment property

```
runAlignment: WrapAlignment.end,
```

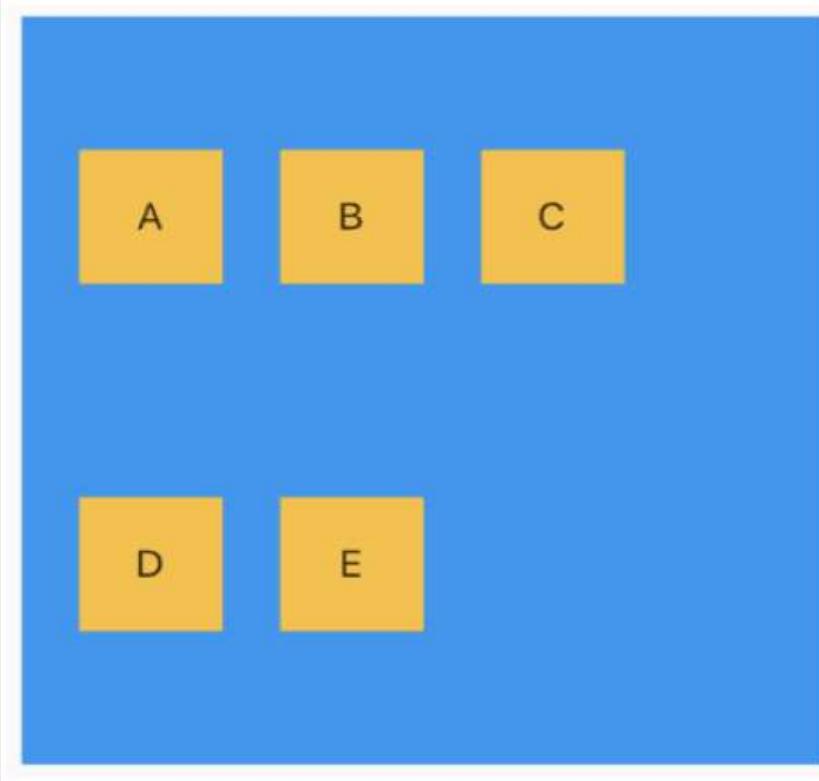


```
runAlignment: WrapAlignment.center,
```

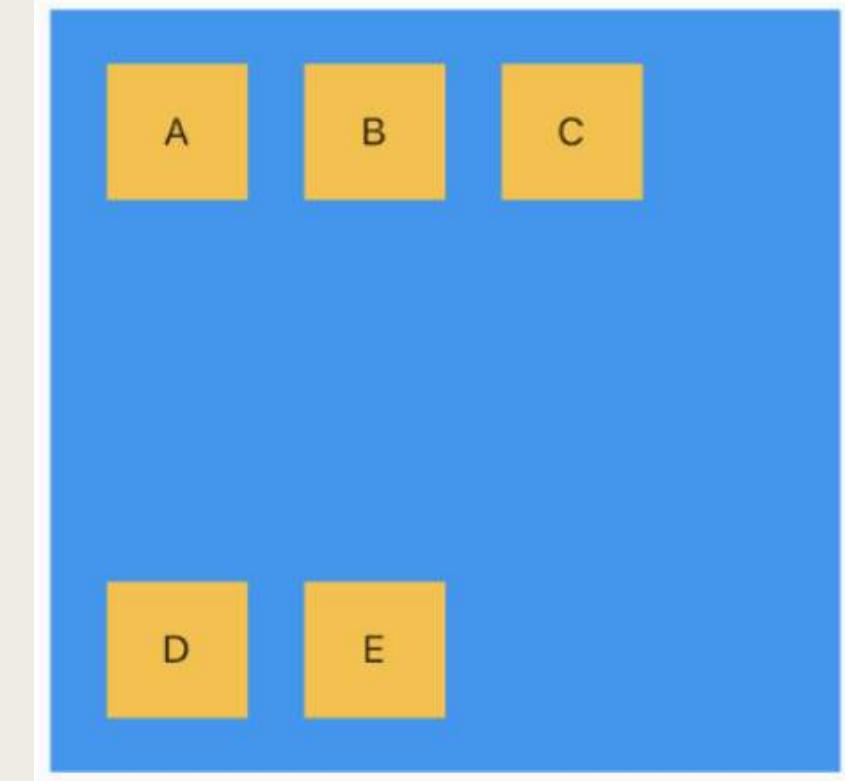


Wrap widget with runAlignment property

```
runAlignment: WrapAlignment.spaceAround,
```



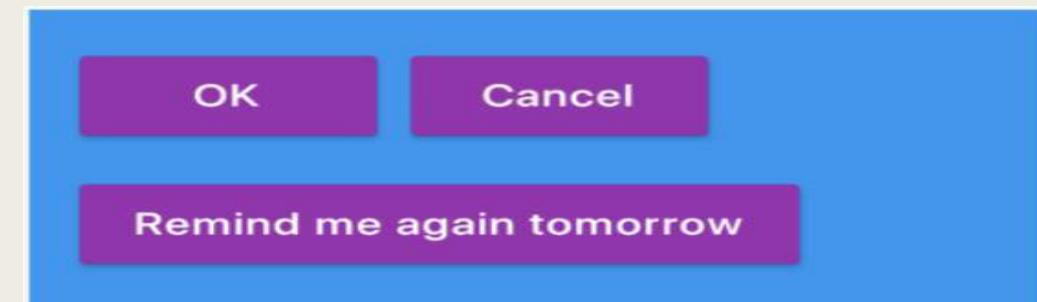
```
runAlignment: WrapAlignment.spaceBetween,
```



Application: Buttons

```
53  class MyButton extends StatelessWidget {  
54    final String text;  
55    MyButton(this.text);  
56    @override  
57    Widget build(BuildContext context) {  
58      return Padding(  
59        padding: const EdgeInsets.all(5),  
60        child: RaisedButton(  
61          color: Colors.purple,  
62          textColor: Colors.white,  
63          child: Text(text),  
64          onPressed: () {},  
65        ), // RaisedButton  
66      ); // Padding  
67    }  
68 }
```

```
Wrap(  
  children: [  
    MyButton('OK'),  
    MyButton('Cancel'),  
    MyButton('Remind me again tomorrow'),  
  ],  
,
```



Dart: Intermediate

- Dart is an object oriented programming language which supports concepts like:
- **Classes**
- **Scope**
- **Packages(Imports)**
- **Polymorphism**
- **Inheritance**
- **Generics**

Classes

Dart Classes

- Dart is an object oriented programming language and supports the concept of class, objects, interfaces, inheritance etc.
- Class can be defined as **blueprint or prototype of associated objects**
- Class is a wrapper that binds/encapsulates the data and methods together, which can be later accessed by objects of that class

Class can be user defined data type that describe behavior and characteristics shared by all of its instances

- Once a class has been defined, we can create objects(instances) of that class which has access to class properties and methods

Dart Classes(contd..)

Declaring a Class in Dart

- In dart, class can be defined using **class** keyword followed by class name; and class body enclosed by pair of curly braces {}

Syntax:-

```
class ClassName {  
    <fields>  
    <getters/setters>  
    <constructors>  
    <functions>  
}
```

Example:-

```
class Employee {  
    var empName;  
    var empAge;  
    var empSalary;  
  
    showEmpInfo() {  
        print("Employee Name Is : ${empName}");  
        print("Employee Age Is : ${empAge}");  
        print("Employee Salary Is : ${empSalary}");  
    }  
}
```

Fields

method

Dart Classes(contd..)

Creating Class objects in Dart

- Once class has been defined, we can create objects(instance) of class which has access to class fields and function
- Syntax:**

```
var objectName = new ClassName(<constructor_arguments>);
```

- Example:**

```
var emp = new Employee();
```

Dart Classes(contd..)

Accessing Instance

Variables and Functions

- In dart, once instance of class is created, we can access properties and methods of class using property/method name separated by dot(.)operator after instance name

Syntax for Property:-

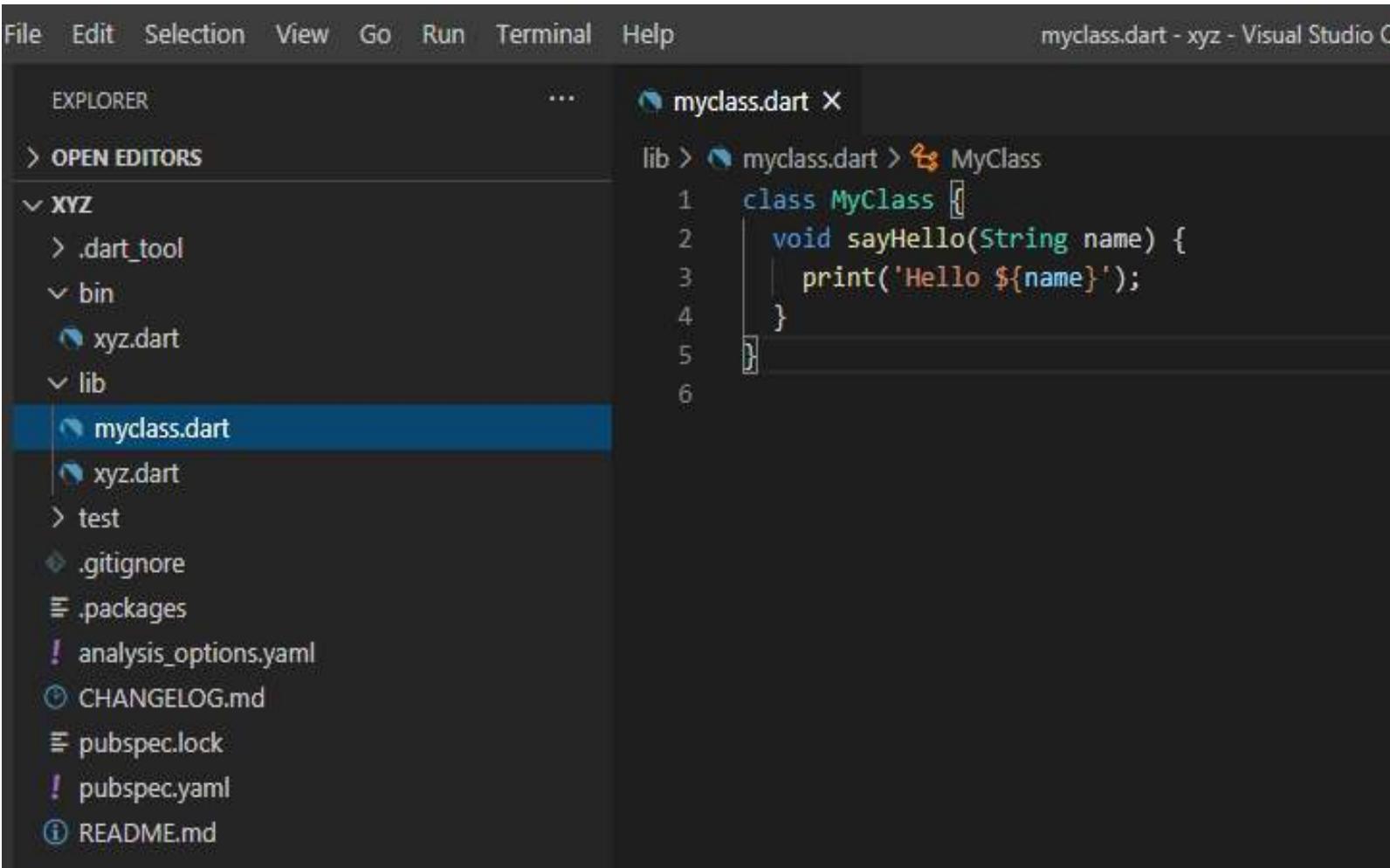
objectName . propName

Syntax for Method:-

objectName . methodName ()

```
class Employee {  
    var empName;  
    var empAge;  
    var empSalary;  
  
    showEmplInfo(){  
        print("Employee Name Is : ${empName}");  
        print("Employee Age Is : ${empAge}");  
        print("Employee Salary Is : ${empSalary}");  
    }  
}  
  
void main(){  
    var emp = new Employee();  
    emp.empName = "John";  
    emp.empAge = 30;  
    emp.empSalary = 45000;  
    print("Dart Access Class Property and Method");  
    emp.showEmplInfo();  
}
```

Classes(1)



The screenshot shows the Visual Studio Code interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates the file is "myclass.dart - xyz - Visual Studio Code".

The left sidebar displays the project structure under "OPEN EDITORS":

- XYZ
 - .dart_tool
 - bin
 - xyz.dart
 - lib
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - ! analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - ! pubspec.yaml
 - README.md

The "myclass.dart" file in the lib folder is selected and highlighted with a blue background. The code editor on the right shows the following Dart code:

```
myclass.dart X
lib > myclass.dart > MyClass
1 class MyClass {
2     void sayHello(String name) {
3         print('Hello ${name}');
4     }
5 }
6
```

To create dart application in vscode

Classes(2)

The screenshot shows a code editor interface with a dark theme. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates the current file is xyz.dart - xyz - Vis. The left sidebar is labeled EXPLORER and shows the project structure:

- > OPEN EDITORS
- < XYZ
 - > .dart_tool
 - < bin
 - xyz.dart
 - < lib
 - myclass.dart
 - xyz.dart
 - > test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md

The code editor tab for xyz.dart is active, showing the following code:

```
lib > xyz.dart > calculate
1 int calculate() {
2   return 6 * 7;
3 }
4
```

Classes(3)

The screenshot shows a Visual Studio Code interface with the following details:

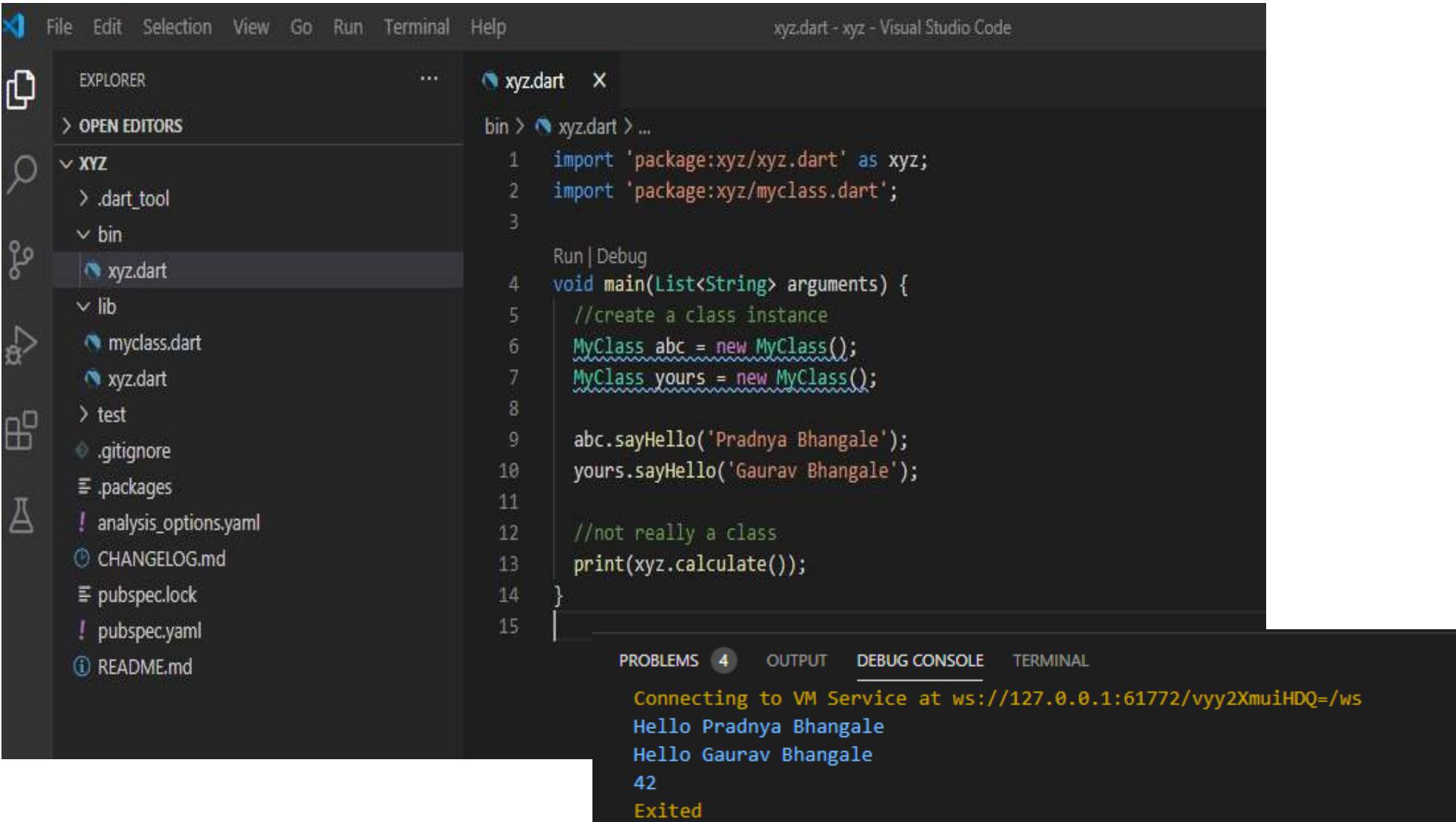
- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** xyz.dart - xyz - Visual Studio Code
- Explorer:** Shows the project structure:
 - XYZ folder
 - .dart_tool
 - bin
 - xyz.dart (selected)
 - lib
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Editor:** The xyz.dart file is open, showing the following code:

```
bin > xyz.dart > main
1 import 'package:xyz/xyz.dart' as xyz;
2 import 'package:xyz/myclass.dart';
3
4 void main(List<String> arguments) {
5     //create a class instance
6     MyClass abc = new MyClass();
7     abc.sayHello('Pradnya Bhangale');
8
9     //not really a class
10    print(xyz.calculate());
11 }
12
```

The terminal window shows the following output:

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
Hello Pradnya Bhangale
42
Exited
```

Classes(4){Multiple objects}



The screenshot shows a Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** xyz.dart - xyz - Visual Studio Code
- Explorer View:** Shows the project structure:
 - XYZ folder
 - .dart_tool
 - bin
 - xyz.dart (selected)
 - lib
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Editor View:** xyz.dart file content:

```
1 import 'package:xyz/xyz.dart' as xyz;
2 import 'package:xyz/myclass.dart';
3
4 void main(List<String> arguments) {
5     //create a class instance
6     MyClass abc = new MyClass();
7     MyClass yours = new MyClass();
8
9     abc.sayHello('Pradnya Bhangale');
10    yours.sayHello('Gaurav Bhangale');
11
12    //not really a class
13    print(xyz.calculate());
14 }
15
```
- Terminal View:** Output of the run command:

```
Connecting to VM Service at ws://127.0.0.1:61772/vyy2XmuiHDQ=/ws
Hello Pradnya Bhangale
Hello Gaurav Bhangale
42
Exited
```

Class Constructor

- Special method that is used to **initialize an object when it is created**
- Automatically called when an object is instantiated; to set initial values for instance variables
- **Has same name as of class it belongs to and has no explicit return type**
- Not mandatory to write constructor for class

Every class has default constructor which compiles and assigns default values to all member variables

- If you define your own constructor then default constructor will be ignored

Class Constructor(contd..)

Creating Constructors in Dart

- A constructor has same name as that of class and doesn't return any value

Syntax:-

```
class ClassName {  
    ClassName() {  
        //constructor body  
    }  
}
```

Example:-

```
void main() {  
    Employee emp = new Employee('EMP001');  
}  
class Employee{  
    Employee(String empCode) {  
        print(empCode);  
    }  
}
```

Output:-

EMP001

Types of Constructor

- Following are constructor type available in dart:
 1. **Default Constructor or no-arg Constructor**
 2. **Parameterized Constructor**
 3. **Named Constructor**

Default Constructor or no-arg Constructor

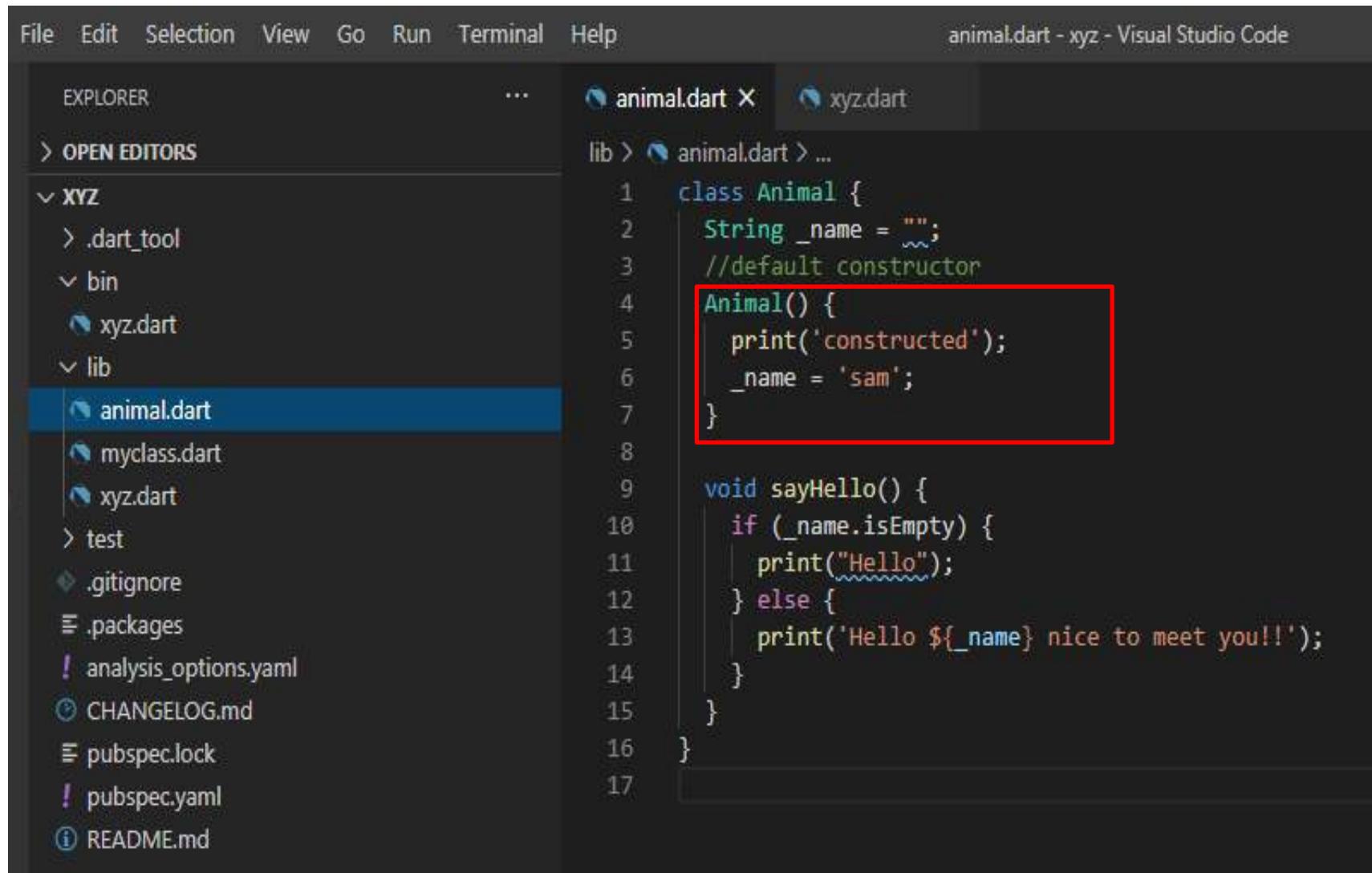
- As name specifies the constructor that has no parameters is known as default constructor
- If you don't create constructor for a class, **compiler will automatically create default constructor for the class**
- If we create a constructor with no-arguments or arguments then compiler will not create a default constructor
- Default constructor provide default values to member variables

Syntax:-

```
class ClassName {  
    ClassName() {  
        //constructor body  
    }  
}
```

```
void main() {  
    Employee emp = new Employee();  
}  
class Employee{  
    Employee() {  
        print("Default Constructor of Employee class ");  
    }  
}
```

Default Constructor or no-arg Constructor(1)



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** animal.dart - xyz - Visual Studio Code
- Explorer:** Shows the project structure under 'XYZ'. The 'lib' folder contains 'animal.dart', which is currently selected and highlighted with a blue background.
- Editor:** Displays the contents of 'animal.dart'. The code defines a class 'Animal' with a constructor and a method 'sayHello'. A red box highlights the constructor definition:

```
1 class Animal {  
2     String _name = "";  
3     //default constructor  
4     Animal() {  
5         print('constructed');  
6         _name = 'sam';  
7     }  
8  
9     void sayHello() {  
10        if (_name.isEmpty) {  
11            print("Hello");  
12        } else {  
13            print('Hello ${_name} nice to meet you!!');  
14        }  
15    }  
16}  
17
```

Default Constructor or no-arg Constructor(2)

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under the 'XYZ' folder:
 - bin
 - lib
 - animal.dart
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Editor:** The file 'xyz.dart' is open, showing the following code:

```
import 'package:xyz/animal.dart';
void main() {
    Animal dog = new Animal();
    dog.sayHello();
}
```

The line `Animal dog = new Animal();` is highlighted with a red box.
- Terminal:** Shows the output of the application:

```
Connecting to VM Service at ws://127.0.0.1:62727/LYZeRT9SKOM=/ws
constructed
Hello sam nice to meet you!!
Exited
```

Parameterized Constructor

- Constructors can also take parameters which is used to initialize instance variables
- A constructor that accepts parameter is called parameterized constructor
- If we want **to initialize instance variables with our own values**, then we are required to use parameterized constructor

Syntax:-

```
class ClassName {  
    ClassName(parameter_list) {  
        //constructor body  
    }  
}
```

```
void main() {  
    print("Dart Parameterized Constructor");  
    Employee emp = new Employee('EMP001');  
}  
  
class Employee{  
    Employee(String empCode) {  
        print(empCode);  
    }  
}
```

Parameterized Constructor(1)

The screenshot shows a Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** animal.dart - xyz - Visual Studio Code
- Explorer View:** Shows the project structure:
 - XYZ folder: .dart_tool, bin, xyz.dart
 - lib folder: animal.dart (selected), myclass.dart, xyz.dart
 - test folder
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Code Editor:** The animal.dart file is open, displaying the following code:

```
1 class Animal {  
2     String _name = "";  
3  
4     //parameterized constructor  
5     Animal(String name) {  
6         _name = name;  
7     }  
8  
9     void sayHello() {  
10        if (_name.isEmpty) {  
11            print("Hello");  
12        } else {  
13            print('Hello ${_name} nice to meet you!!');  
14        }  
15    }  
16}  
17
```

Parameterized Constructor(2)

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure with files like `animal.dart`, `xyz.dart`, `myclass.dart`, and `CHANGELOG.md`.
- Editor:** The `xyz.dart` file is open, displaying the following code:

```
1 import 'package:xyz/animal.dart';
2
3 void main() {
4     Animal dog = new Animal('Pradnya');
5
6     dog.sayHello();
7 }
```
- Terminal:** Shows the output of running the code, including connecting to a VM service and printing "Hello Pradnya nice to meet you!".

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL
Connecting to VM Service at ws://127.0.0.1:62795/pHdSHfpT_MQ=/ws
Hello Pradnya nice to meet you!
Exited
```

Named Constructors

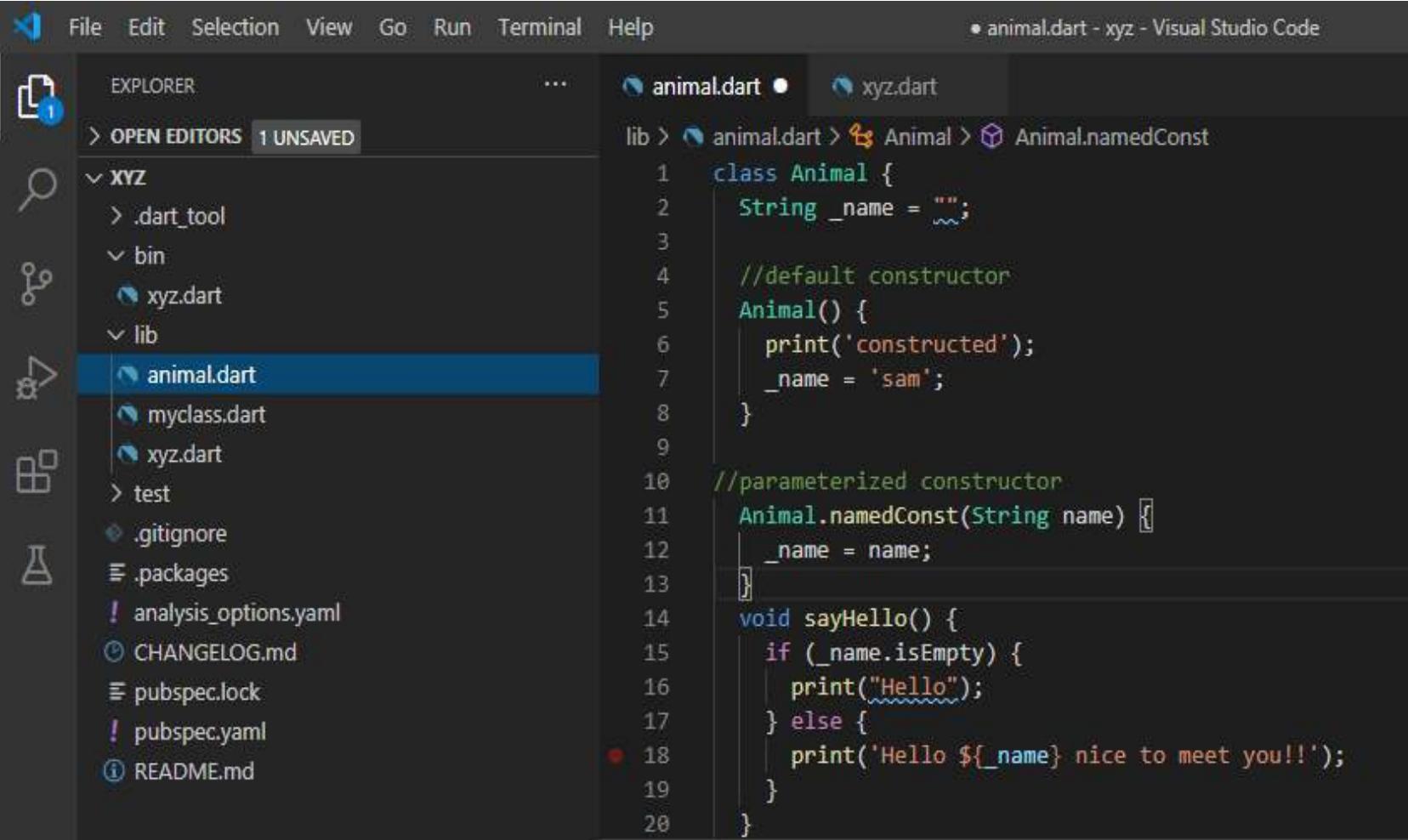
- In Dart, named constructor allows a class to define **multiple constructors**

Syntax:-

ClassName.constructor_name(param_list)

```
void main() {  
  
    Employee emp1 = new Employee();  
    Employee emp2 = new  
    Employee.namedConst('EMP001');  
}  
  
class Employee{  
    Employee() {  
        print("Default Constructor Invoked");  
    }  
    Employee.namedConst(String empCode) {  
        print("Named Constructor Invoked");  
        print(empCode);  
    }  
}
```

Named Constructor(1)



File Edit Selection View Go Run Terminal Help

animal.dart - xyz - Visual Studio Code

EXPLORER OPEN EDITORS 1 UNSAVED

XYZ

bin

lib

animal.dart

myclass.dart

xyz.dart

test

.gitignore

.packages

analysis_options.yaml

CHANGELOG.md

pubspec.lock

pubspec.yaml

README.md

animal.dart xyz.dart

lib > animal.dart > Animal > Animal.namedConst

```
1 class Animal {  
2     String _name = "";  
3  
4     //default constructor  
5     Animal() {  
6         print('constructed');  
7         _name = 'sam';  
8     }  
9  
10    //parameterized constructor  
11    Animal.namedConst(String name) {  
12        _name = name;  
13    }  
14    void sayHello() {  
15        if (_name.isEmpty) {  
16            print("Hello");  
17        } else {  
18            print('Hello ${_name} nice to meet you!!');  
19        }  
20    }  
21}
```

Named Constructor(2)

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under the 'xyz' folder:
 - bin
 - lib
 - animal.dart
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Editor:** The 'xyz.dart' file is open, showing the following code:

```
import 'package:xyz/animal.dart';

void main() {
    Animal cat = new Animal();
    Animal dog = new Animal.namedConst('Pradnya');

    cat.sayHello();
    dog.sayHello();
}
```
- Terminal:** Shows the output of the application:

```
Connecting to VM Service at ws://127.0.0.1:62820/2P0wkiaY0m4=/ws
constructed
Hello sam nice to meet you!!
Hello Pradnya nice to meet you!!
Exited
```

Dart **this** keyword

- **this** keyword represents an implicit object pointing to current class object
- Refers to current instance of the class in a method or constructor
- Mainly used to eliminate ambiguity between class attributes and parameters with same name

Ambiguity is avoided by prefixing class attributes with **this** keyword

- **this** keyword can be used to refer to any member of the current object from within an instance method or constructor

Use of this keyword

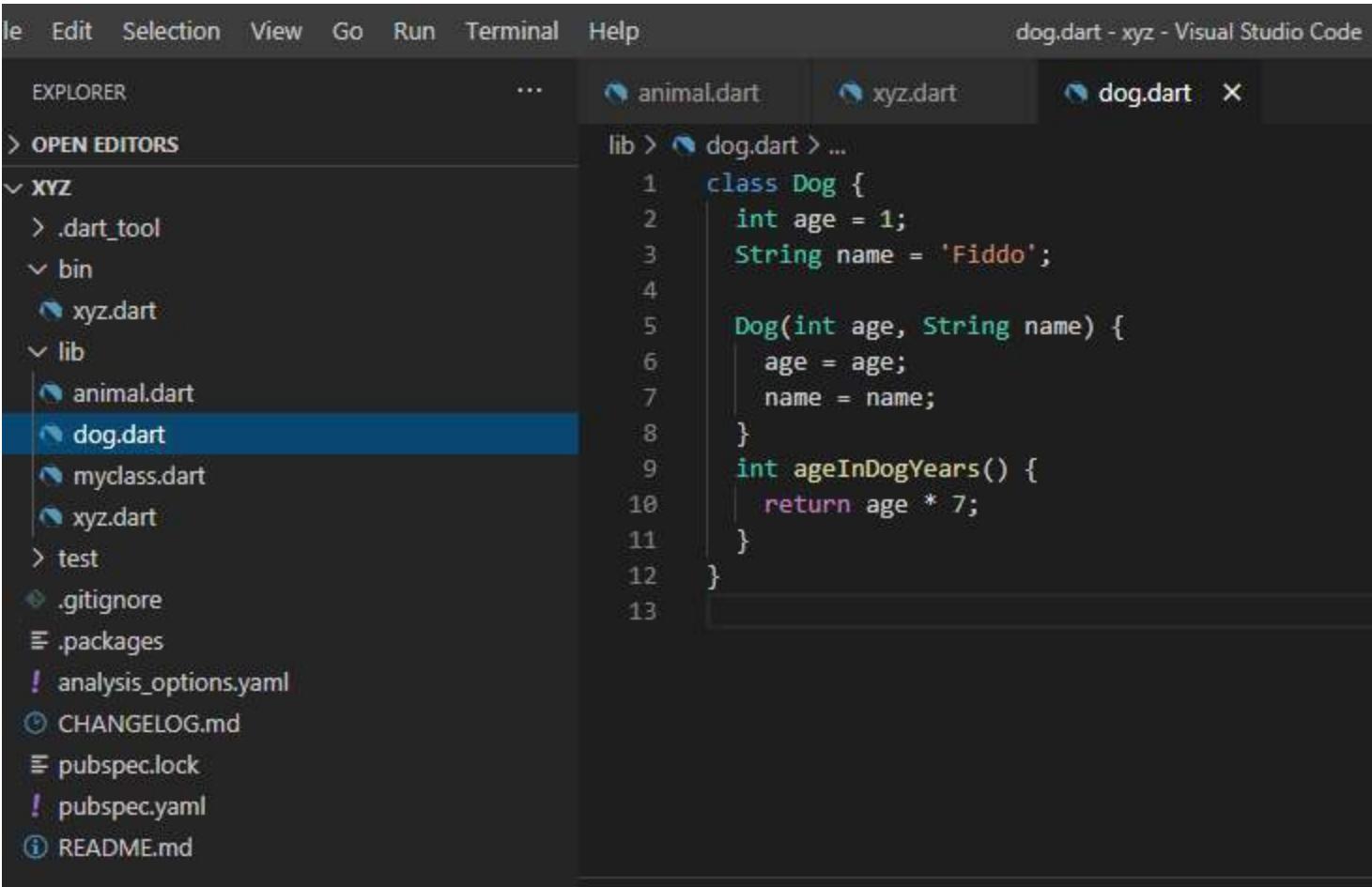
- ▶ It can be used to refer instance variable of current class
- ▶ It can be used to invoke or initiate current class constructor
- ▶ It can be passed as an argument in the method call
- ▶ It can be passed as argument in the constructor call
- ▶ It can be used to invoke current class method
- ▶ It can be used to return the current class instance

this keyword(contd..)

Example

```
void main() {  
    Employee emp = new Employee('EMP001');  
}  
  
class Employee {  
    String emp_code;  
    Employee(String emp_code) {  
        this.emp_code = emp_code;  
        print("Dart this Keyword Example.");  
        print("The Employee Code is : ${emp_code}");  
    }  
}
```

this keyword(contd..)

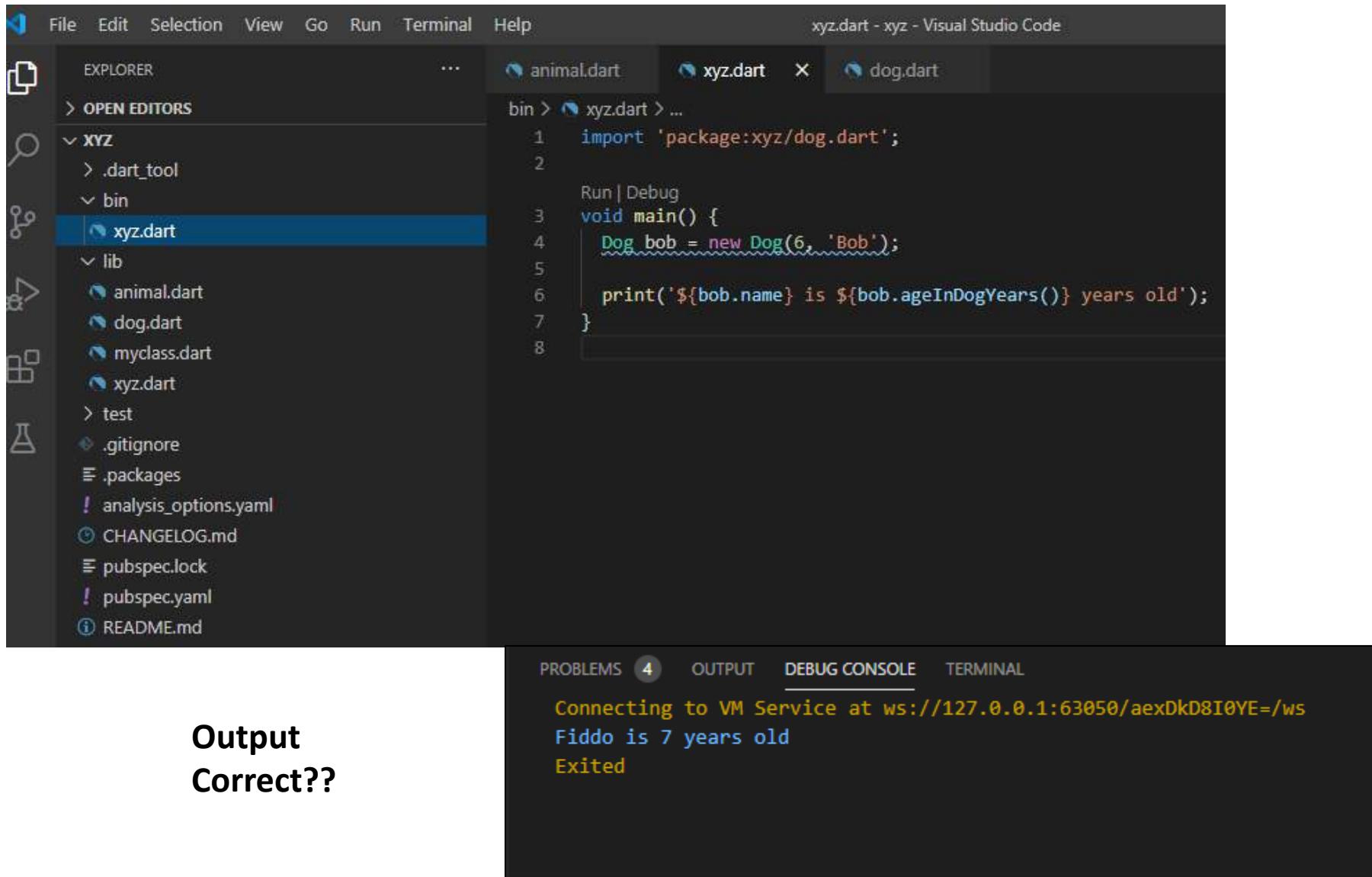


The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** dog.dart - xyz - Visual Studio Code.
- Explorer View:** Shows the project structure:
 - XYZ folder
 - .dart_tool
 - bin
 - xyz.dart
 - lib
 - animal.dart
 - dog.dart** (highlighted with a blue selection bar)
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Code Editor:** The dog.dart file is open, displaying the following Dart code:

```
1 class Dog {  
2     int age = 1;  
3     String name = 'Fido';  
4  
5     Dog(int age, String name) {  
6         this.age = age;  
7         this.name = name;  
8     }  
9     int ageInDogYears() {  
10        return age * 7;  
11    }  
12 }  
13 }
```

this keyword(contd..)



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure under the 'XYZ' folder:
 - bin
 - lib
 - animal.dart
 - dog.dart
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Editor:** The 'xyz.dart' file is open, containing the following Dart code:

```
import 'package:xyz/dog.dart';

void main() {
    Dog bob = new Dog(6, 'Bob');

    print('${bob.name} is ${bob.ageInDogYears()} years old');
}
```
- Terminal:** The terminal output shows:

```
Connecting to VM Service at ws://127.0.0.1:63050/aexDkD8I0YE=/ws
Fido is 7 years old
Exited
```
- Status Bar:** Shows 'PROBLEMS 4'.

Output
Correct??

this keyword(contd..)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under 'XYZ'. The 'lib' folder contains 'animal.dart', 'dog.dart', 'myclass.dart', and 'xyz.dart'. Other files like '.dart_tool', 'bin', 'test', '.gitignore', '.packages', 'CHANGELOG.md', 'pubspec.lock', 'pubspec.yaml', and 'README.md' are also listed.
- Editor (Right):** The 'dog.dart' file is open. The code is as follows:

```
class Dog {  
    int age = 1;  
    String name = 'Fido';  
  
    Dog(int age, String name) {  
        this.age = age;  
        this.name = name;  
    }  
  
    int ageInDogYears() {  
        return age * 7;  
    }  
}
```

A yellow lightbulb icon is positioned above the line 'this.name = name;' in the constructor.
- Status Bar:** Shows 'dog.dart - xyz - Visual Studio Code'

this keyword(contd..)

The screenshot shows a Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** xyz.dart - xyz - Visual Studio Code
- Explorer:** Shows a tree view of files and folders:
 - XYZ
 - .dart_tool
 - bin
 - xyz.dart
 - lib
 - animal.dart
 - dog.dart
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Editor:** The file xyz.dart is open, showing the following code:

```
1 import 'package:xyz/dog.dart';
2
3 void main() {
4     Dog bob = new Dog(6, 'Bob');
5
6     print('${bob.name} is ${bob.ageInDogYears()} years old');
7 }
```
- Terminal:** Shows the output of the application:

```
Connecting to VM Service at ws://127.0.0.1:63059/U8Y5wsLPhNY=/ws
Bob is 42 years old
Exited
```

Dart Programming: Basics

Outline

- Introduction to Dart
- Variables
- Collections
- Dart operators
- Flow Control
- Functions
- Error Handling

Introduction to Dart

- Dart is an **open-source, general-purpose, object-oriented programming language** with C-style syntax developed by **Google** in **2011**
- Dart is productive, fast, portable, approachable, and most of all reactive.
- It supports most of the common concepts of programming languages like **classes, interfaces, functions**
- Dart language does not support arrays directly.
It supports collection, which is used to replicate the data structure such as arrays, generics, and optional typing.

Dart Programming

- **Pros of Dart**
 - Open-source
 - Backed by Google and runs easily on Google Cloud Platform
 - Dart is approximately two times faster than JavaScript
 - Dart is type-safe and compiled with both AOT and JIT compilers
- **Cons of Dart**
 - Dart is fairly new to the programmers and rarely used in the market.
 - Dart has very limited resources online and it's hard to find solutions to problems.

Dart Basics: Variables

- Comments
- Booleans
- Numbers
- Strings
- Const Variables

Comments

- Help the readability of the code
- Used to describe the logic and dependencies of the app
- Non executable lines of code
- There are three types of comments:
 - **Single-line** (add a short description) `//.....`
 - **Multiline** (long descriptions that span multiple lines) `/*....*/`
 - **Documentation** (fully document a piece of logic, usually giving detailed explanations and sample code in the comments `/// []`)

Comments

```
sample.dart
sample.dart > main
1  /*this is starting point
2   in the application*/
3
4  void main()
5  {
6    //this prints out to the screen
7    print(Hello World! );
8  }
9
```

Multiline comments

```
/*.....*/
```

Single Line

```
//
```

```
PS C:\Users\Gaurav\Desktop\dart_apps> dart sample.dart
Hello world!
```

Variables

- Variables store *references to a value*
- Some of the built-in variable types are **numbers, strings, Booleans, lists, maps** etc.
- You can use **var** to declare a variable without specifying the type.

Dart infers the type of variable automatically

- Declaring the variable type makes for better code readability, and it's easier to know which type of value is expected.

Instead of using var, use the variable type expected:
double, String, and so on.

- An uninitialized variable has a value of null
- Use final or const keywords when the variable is not intended to change the initial value

Declaring Variables

- In Dart, all variables are declared public (available to all) by default
- Starting the variable name with an underscore (_), you can declare it as private.

By declaring a variable private, you are saying it cannot be accessed from outside classes/functions

- **Note** some built-in Dart variable types are lowercase like double and some uppercase like String
- Use ***final keyword*** when the value is assigned at runtime
- Use ***const keyword*** when the value is known at compile time (in code) and will not change at runtime.

Declaring Variables Rules

- Special characters are not allowed(@,&,#). There is an exception for dollar sign(\$) and underscore (_)
- A keyword cannot be used as a variable name for example int , double cannot be used as a variable name.
- Variables are case sensitive
- First character of a variable should not be a digit and must always be an alphabet.
- Blank spaces cannot be used
- A combination of numbers and alphabet can be used to name a variable for example , name123 can be used as a variable name.

Booleans

- Dart provides an inbuilt support for the Boolean data type.
- The Boolean data type in DART supports only two values – ***true and false***
- The keyword **bool** is used to represent a Boolean literal in DART.
- The syntax for declaring a Boolean variable in DART is as given below –
 - `bool var_name = true`
 - `bool var_name = false`

Booleans

Terminal Help sample.dart - dart_apps - Visual Studio Code

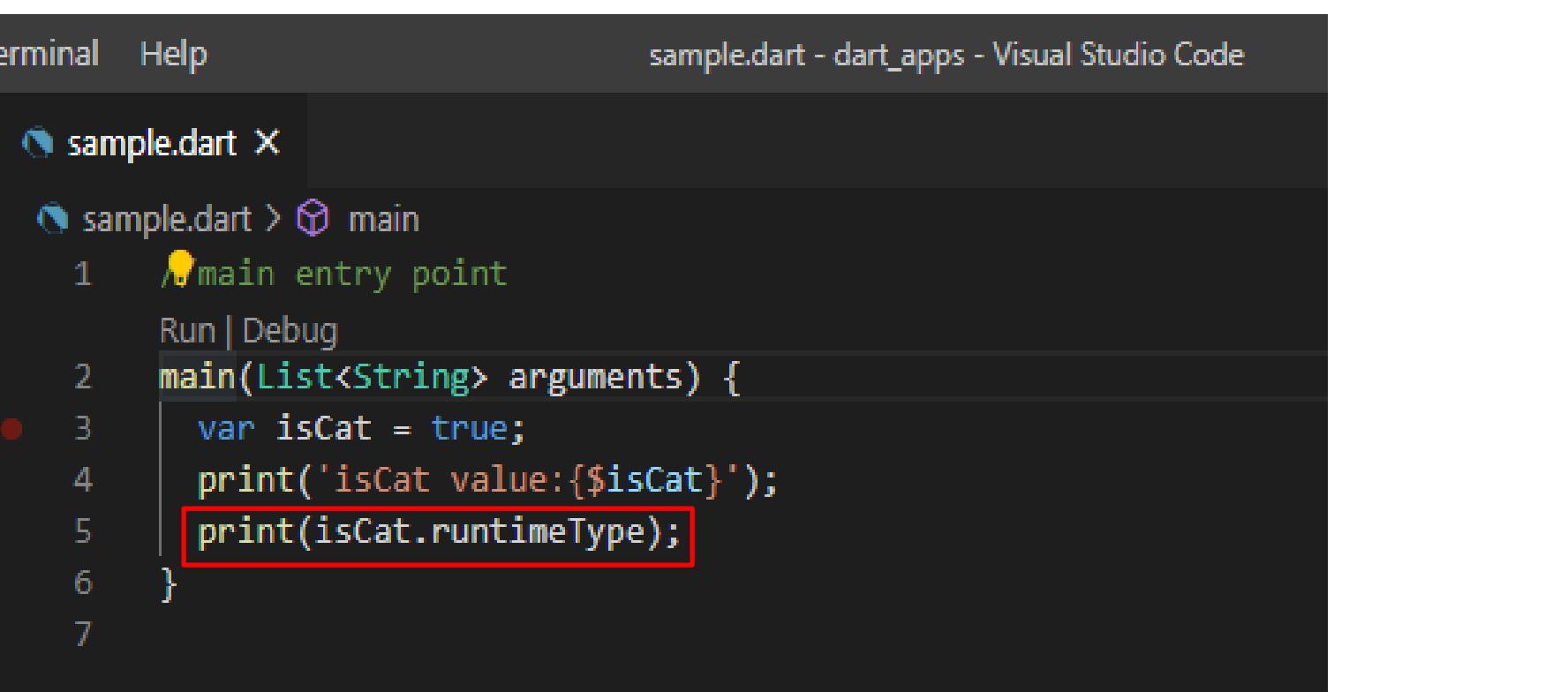
sample.dart X

sample.dart > ...

```
1 //main entry point
2 Run | Debug
3 main(List<String> arguments) {
4     bool isOn;
5     bool isDog = false;
6
7     print(isOn);
8     print(isDog);
9 }
```

```
PS C:\Users\Gaurav\Desktop\dart_apps> dart sample.dart
null
false
PS C:\Users\Gaurav\Desktop\dart_apps>
```

Booleans(contd..)



A screenshot of the Visual Studio Code interface. The title bar says "sample.dart - dart_apps - Visual Studio Code". The left sidebar shows a tree view with "sample.dart > main". The main editor area contains the following Dart code:

```
1 //main entry point
2
3 main(List<String> arguments) {
4     var isCat = true;
5     print('isCat value:${isCat}');
6     print(isCat.runtimeType);
7 }
```

The line `print(isCat.runtimeType);` is highlighted with a red box.

```
PS C:\Users\Gaurav\Desktop\dart_apps> dart sample.dart
isCat value:{true}
bool
PS C:\Users\Gaurav\Desktop\dart_apps>
```

To retrieve(interpolate) value stored in variable
{\$var_name} or \${var_name}

Numbers

- Dart numbers can be classified as –
 - **int** – Integer of arbitrary size used to represent whole numbers.
 - **double** – 64-bit (double-precision) floating-point numbers used to represent fractional numbers
- The **num** type is inherited by the **int** and **double** types.
- The **dart core library** allows numerous operations on numeric values.
- The syntax for declaring a number is as given below –
 - `int var_name; // declares an integer variable`
 - `double var_name; // declares a double variable`

Numbers(contd..)

Terminal Help sample.dart - dart_apps - Visual Studio Code

sample.dart X

sample.dart > ...

```
1 //main entry point
2 Run | Debug
3 main(List<String> arguments) {
4     //numbers
5     num age = 10;
6     //Int
7     int people = 8;
8     //Double
9     double temperature = 32.12;
10
11    print(age);
12    print(people);
13    print(temperature);
14
15    //parse an integer
16    int test = int.parse('12');
17    print(test);
18
19    //math
20    int dogyears = 34;
21    int dogage = age * dogyears;
22    print('dog age is:${dogage}');
23 }
```

```
PS C:\Users\Gaurav\Desktop\dart_apps> dart sample.dart
10
8
32.12
12
dog age is:340
```

Strings

- Declaring variables as String allows values to be entered as a sequence of text characters.
- To add a single line of characters, you can use single or double quotes like 'car' or "car".

Ex:

```
String defaultMenu = 'main';  
String defaultMenu = "main";
```

- To add multiline characters, use triple quotes, like ""car"".

Ex:

```
String multilineAddress = ""  
123 Any Street  
City, State, Zip  
"";
```

Strings(contd..)

- Strings can be concatenated (combined) by using the plus (+) operator or by using adjacent single or double quotes.

Ex:

```
String combinedName = 'main' + ' ' + 'function';
```

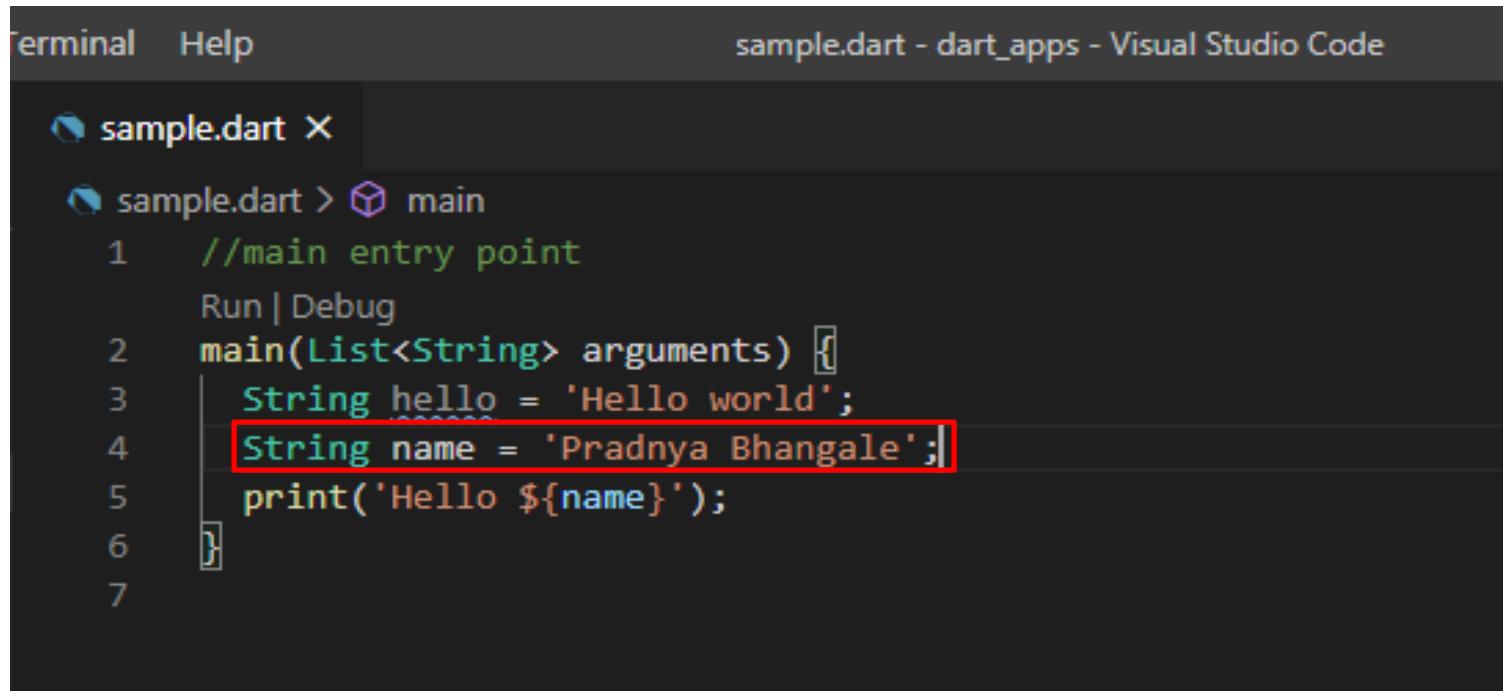
```
String combinedNameNoPlusSign = 'main' ' ' 'function';
```

Strings

```
sample.dart X
sample.dart > main
1 //main entry point
Run | Debug
2 main(List<String> arguments) {
3   String hello = 'Hello world';
4   print('Hello World');
5   print(hello);
6 }
7
```

```
PS C:\Users\Gaurav\Desktop\dart_apps> dart sample.dart
Hello World
Hello world
PS C:\Users\Gaurav\Desktop\dart_apps>
```

Strings(contd..)



Terminal Help sample.dart - dart_apps - Visual Studio Code

sample.dart X

sample.dart > main

```
1 //main entry point
2 Run | Debug
3 main(List<String> arguments) {
4   String hello = 'Hello world';
5   String name = 'Pradnya Bhangale';|
6   print('Hello ${name}');
7 }
```

```
PS C:\Users\Gaurav\Desktop\dart_apps> dart sample.dart
Hello Pradnya Bhangale
PS C:\Users\Gaurav\Desktop\dart_apps> 
```

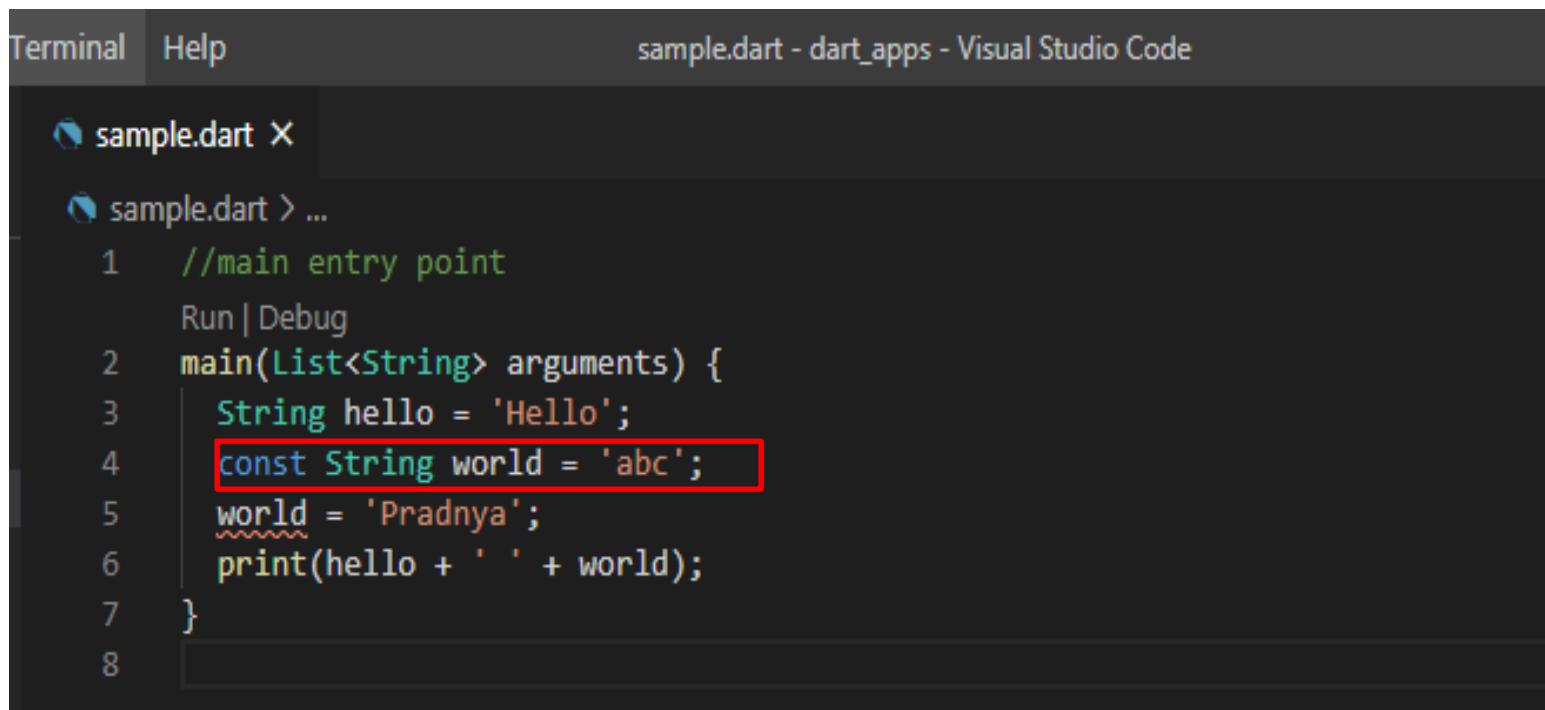
Strings(contd..)

The screenshot shows a Visual Studio Code interface with a dark theme. The title bar says "sample.dart - dart_apps - Visual Studio Code". The left sidebar shows a file tree with "sample.dart X" and "sample.dart > main". The main editor area contains Dart code. Several lines of code are highlighted with red boxes: line 5 ("String firstname = name.substring(0, 7);"), line 9 ("String lastname = name.substring(index).trim();"), line 12 ("print(name.length);"), and line 14 ("print(name.contains('nya'))").

```
1 //main entry point
2 main(List<String> arguments) {
3     String name = 'Pradnya Bhangale';
4     //get a substring
5     String firstname = name.substring(0, 7);
6     print('First Name is ${firstname} ');
7     //get the string index
8     int index = name.indexOf(' ');
9     String lastname = name.substring(index).trim();
10    print('Lastname is:${lastname}');
11    //length of string(includes space)
12    print(name.length);
13    //contains(search)
14    print(name.contains('nya'));
15 }
16
```

```
PS C:\Users\Gaurav\Desktop\dart_apps> dart sample.dart
First Name is Pradnya
Lastname is:Bhangale
16
true
PS C:\Users\Gaurav\Desktop\dart_apps>
```

Const Variables



```
Terminal Help sample.dart - dart_apps - Visual Studio Code

sample.dart X
sample.dart > ...

1 //main entry point
Run | Debug
2 main(List<String> arguments) {
3     String hello = 'Hello';
4     const String world = 'abc';
5     world = 'Pradnya';
6     print(hello + ' ' + world);
7 }
8
```

```
PS C:\Users\Gaurav\Desktop\dart_apps> dart sample.dart
sample.dart:5:3: Error: Can't assign to the const variable 'world'.
    world = 'Pradnya';
    ^^^^^^
```

```
PS C:\Users\Gaurav\Desktop\dart_apps>
```

Collections

- Enum
- List
- Set
- Queue
- Map

Introduction: Collection

- Dart **collections** can be used to replicate data structures like an array.
- The **dart:core** library and other classes enable **Collection** support in **Dart** scripts

Enum

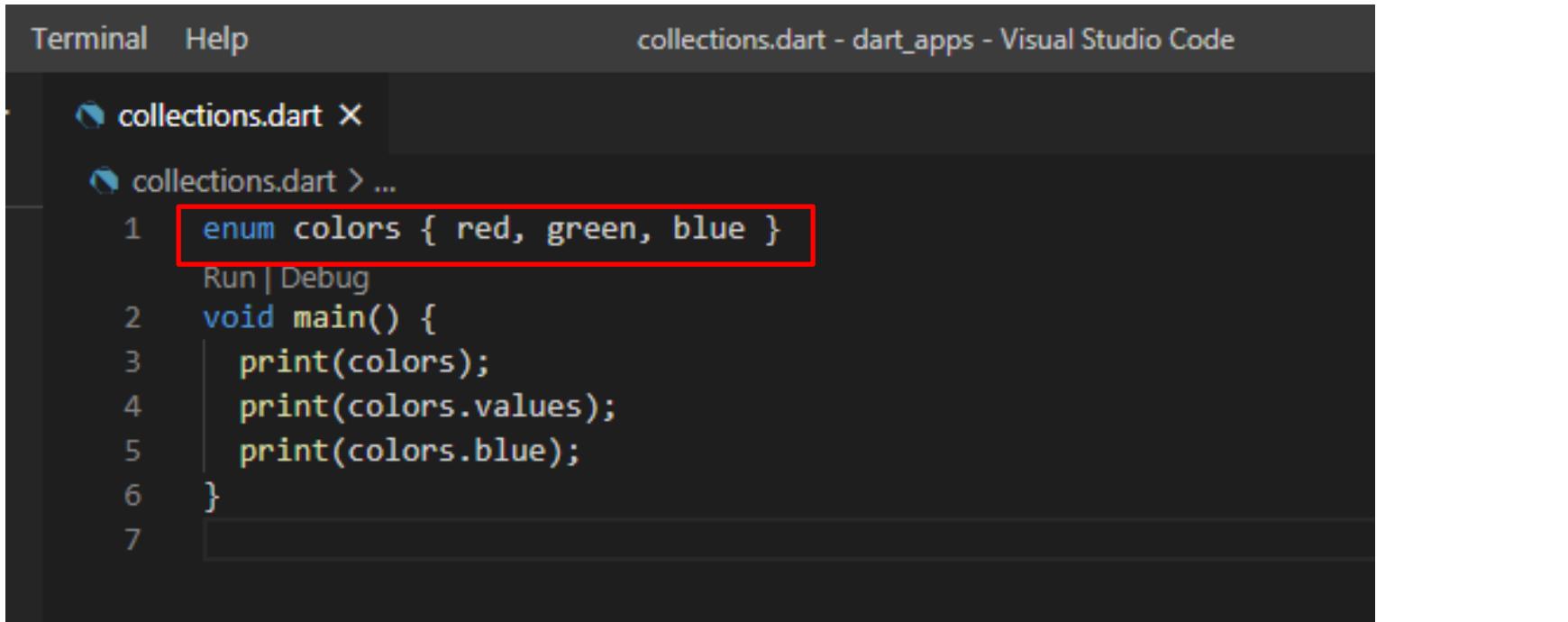
- An enumeration is used for defining named constant values
- An enumerated type is declared using the **enum** keyword
- Syntax

```
enum enum_name {  
    enumeration list  
}
```

enum_name specifies the enumeration type name

enumeration list is a comma-separated list of identifiers

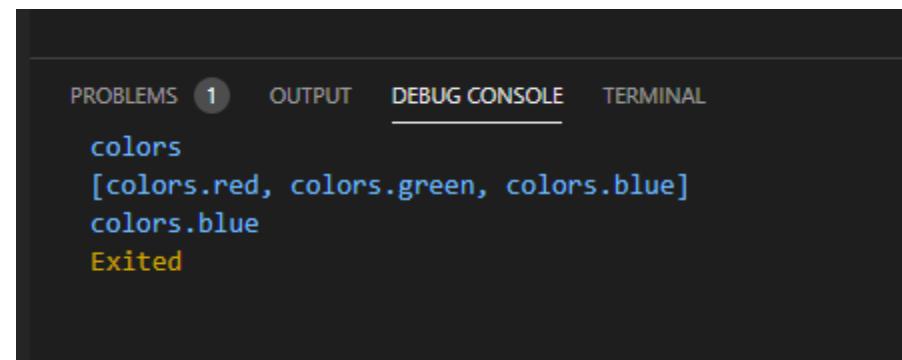
Enum



A screenshot of the Visual Studio Code interface. The title bar says "collections.dart - dart_apps - Visual Studio Code". The left sidebar shows two tabs: "collections.dart X" and "collections.dart > ...". The main editor area contains the following Dart code:

```
1 enum colors { red, green, blue }
2 void main() {
3     print(colors);
4     print(colors.values);
5     print(colors.blue);
6 }
7
```

The first line, "enum colors { red, green, blue }", is highlighted with a red rectangle.



The DEBUG CONSOLE tab is active, showing the following output:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
colors
[colors.red, colors.green, colors.blue]
colors.blue
Exited
```

List

- Declaring variables as List (comparable to arrays) allows multiple values to be entered
- **List is an ordered group of objects.**
- In programming, an array is an iterable (accessed sequentially) collection of objects, with each element accessible by the index position or a key.

To access elements, the List uses **zero-based indexing**, where the first element index is at 0, and the last element is at the List length (number of rows) minus 1

List(contd..)

- A List can be of **fixed length or growable**, depending on your needs.
- By default, a List is created as growable by using List() or []
- To create a fixed-length List, you add the number of rows required by using this format: List(25)
- Example:

// List Growable

```
List contacts = List();
```

// or

```
List contacts = [];
```

```
List contacts = ['Linda', 'John', 'Mary'];
```

// List fixed-length

```
List contact = List(25);
```

List(contd..)

Terminal Help collections.dart - dart_apps - Visual Studio Code

collections.dart X

collections.dart > main

Run | Debug

```
1 void main() {  
2     List test = [1, 2, 3, 4];  
3     print(test);  
4     print('length is ${test.length}'); //how many elements  
5     print('first element is:${test[0]}'); //zero based index  
6     print('element at specific index:${test.elementAt(3)}');  
7     print('element at specific index:${test.elementAt(300)}');  
8 }  
9
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
[1, 2, 3, 4]  
length is 4  
first element is:1  
element at specific index:4  
Unhandled exception:  
RangeError (index): Invalid value: Not in range 0..3, inclusive: 300  
#0      List.[]  (dart:core-patch/growable_array.dart:146:60)  
#1      List.elementAt (dart:core-patch/growable_array.dart:355:16)
```

Terminal Help

• collections.dart - dart_apps - Visual Studio Code

collections.dart ●

collections.dart > main

Run | Debug

```
1 void main() {  
2     List things = new List();  
3     things.add(1);  
4     things.add('cat');  
5     things.add(true);  
6     print(things);  
7  
8     List<int> numbers = new List<int>();  
9     numbers.add(1);  
10    numbers.add(2);  
11    numbers.add('cat');|  
12  
13 }  
14
```

The argument type 'String' can't be assigned to the parameter type 'int'. dart(argument_type_not_assignable)
Peek Problem (Alt+F8) No quick fixes available

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

Connecting to VM Service at ws://127.0.0.1:53620/Qe6Co6pxAVI=/ws
[1, cat, true]
Exited

Set

- Set is an unordered list of **distinct** values of same type
- Dart set are typed, once you declare type of set Dart infers all elements of set to be of same data type
- Used when you want to hold distinct values of single data type in a single variable and order of items is not important
- **Add element into set:**

```
set_name.add(<value>);
```

- **To retrieve element at specific index:**

```
set_name.elementAt(<index>);
```

Set (contd..)

- **Set elements count:**

```
set_name.length;
```

- **Finding element in Set:**

```
set_name.contains(<value>);
```

- **Remove Set element:**

```
set_name.remove(<value>);
```

- **Remove all elements in Set:**

```
set_name.clear();
```

Set(contd..)

- Similar to list but are unordered and do not contain duplicates

The screenshot shows a Visual Studio Code interface with the following details:

- Terminal:** Shows the command "collections.dart - dart_apps - Visual Studio Code".
- File Explorer:** Shows two files: "collections.dart X" and "collections.dart > main".
- Code Editor:** Displays the following Dart code in the "main" file:

```
void main() {
    //sets are unordered and do not contain duplicates
    Set<int> numbers = new Set<int>();
    numbers.add(1);
    numbers.add(2);
    numbers.add(3);
    numbers.add(1); //adding twice
    print(numbers);
}
```
- Output:** The "PROBLEMS" tab shows 1 error. The "OUTPUT" tab displays the output of the program:

```
{1, 2, 3}
Exited
```

Queue

- A Queue is a collection that can be manipulated at both ends.
- Queues are useful when you want to build a first-in, first-out collection.
- Queue inserts data from one end and deletes from another end.
- While using queue, it is required to import **“dart:collection”** package for performing operations on values stored in queue

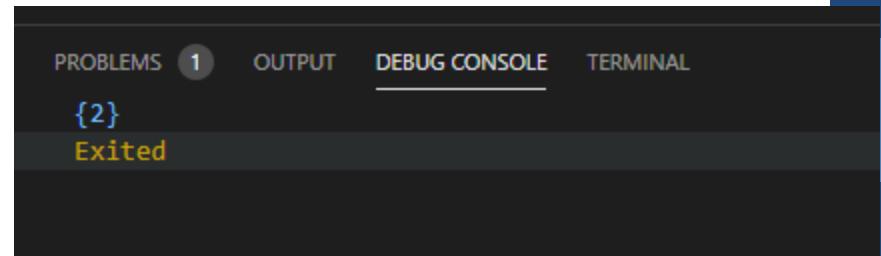
Queue (contd..)

Terminal Help collections.dart - dart_apps - Visual Studio Code

collections.dart X

collections.dart > ...

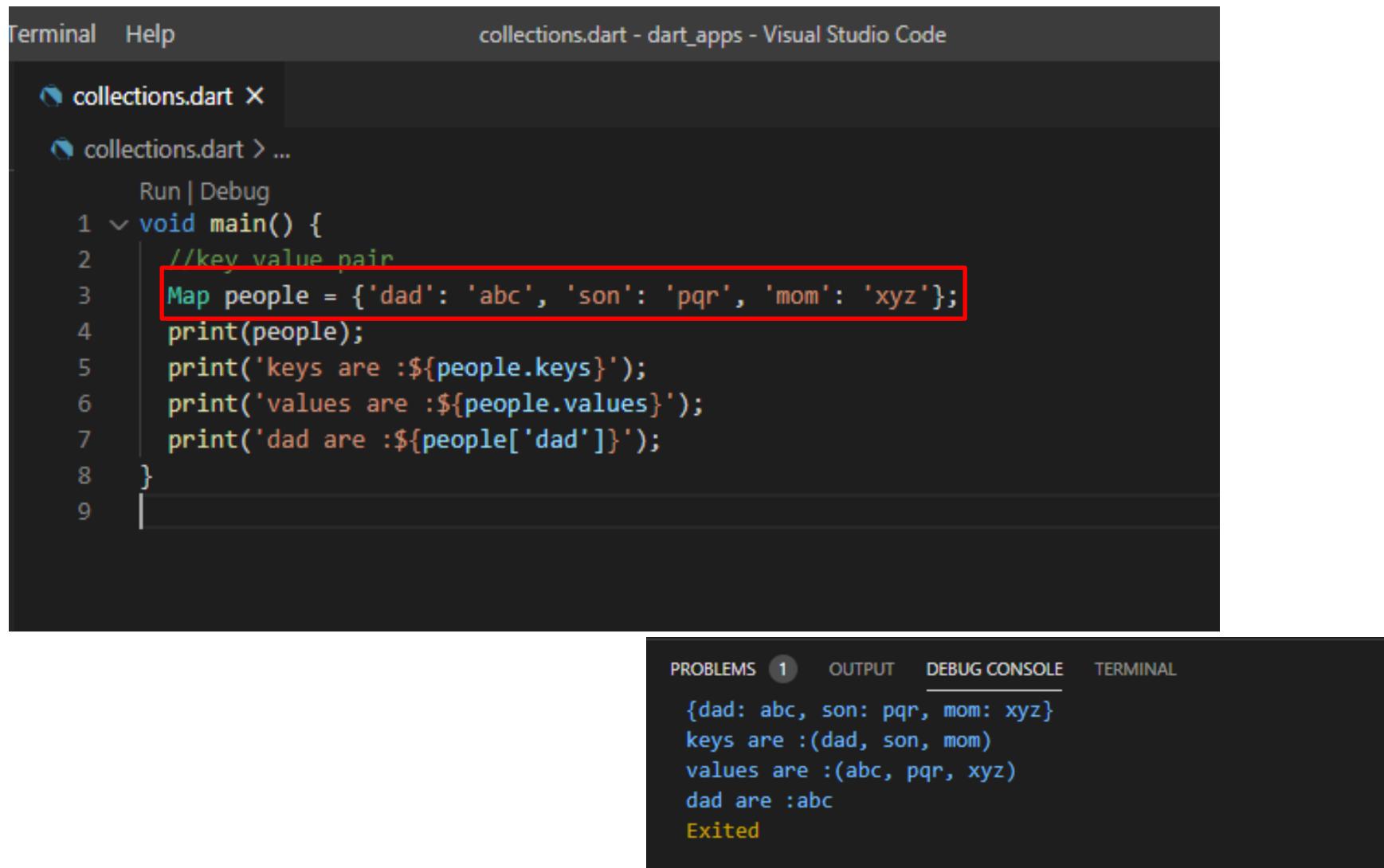
```
1 import 'dart:collection';
2
3 void main() {
4     //ordered, no index and remove element from start and end of queue
5
6     Queue items = new Queue();
7     items.add(1);
8     items.add(2);
9     items.add(3);
10    items.removeFirst();
11    items.removeLast();
12    print(items);
13 }
14
```



Map

- Map is an object that is used to represent a set of values as key-value pairs
- In Map, key and value can be of any type of object(need not be of same data type)
- In Map, each key can occur only once but same value can be used multiple times
- Map can be defined using ({}) and values can be assigned and accessed using ([])

Map (contd..)



The screenshot shows a Visual Studio Code interface with a dark theme. The top bar has 'Terminal' and 'Help' tabs, and the title 'collections.dart - dart_apps - Visual Studio Code'. In the editor area, there are two tabs: 'collections.dart X' and 'collections.dart > ...'. Below the tabs is a 'Run | Debug' button. The code in the editor is:

```
1 void main() {  
2     //key value pair  
3     Map people = {'dad': 'abc', 'son': 'pqr', 'mom': 'xyz'};  
4     print(people);  
5     print('keys are :${people.keys}');  
6     print('values are :${people.values}');  
7     print('dad are :${people['dad']}');  
8 }  
9
```

The line 'Map people = {'dad': 'abc', 'son': 'pqr', 'mom': 'xyz'};' is highlighted with a red box. The bottom part of the image shows the 'OUTPUT' tab of the terminal, which displays the execution results:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL  
{dad: abc, son: pqr, mom: xyz}  
keys are :(dad, son, mom)  
values are :(abc, pqr, xyz)  
dad are :abc  
Exited
```

Sr.No	Dart collection & Description
1	<p>List </p> <p>A List is simply an ordered group of objects. The dart:core library provides the List class that enables creation and manipulation of lists.</p> <ul style="list-style-type: none"> ■ Fixed Length List – The list's length cannot change at run-time. ■ Growable List – The list's length can change at run-time.
2	<p>Set </p> <p>Set represents a collection of objects in which each object can occur only once. The dart:core library provides the Set class to implement the same.</p>
3	<p>Maps </p> <p>The Map object is a simple key/value pair. Keys and values in a map may be of any type. A Map is a dynamic collection. In other words, Maps can grow and shrink at runtime. The Map class in the dart:core library provides support for the same.</p>
4	<p>Queue </p> <p>A Queue is a collection that can be manipulated at both ends. Queues are useful when you want to build a first-in, first-out collection. Simply put, a queue inserts data from one end and deletes from another end. The values are removed / read in the order of their insertion.</p>

Dart Operators(1)

Arithmetic Operator

- There are nine types of an arithmetic operator are there in a dart.

Operator	Meaning
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo
-expr	Unary Minus (reverse the sign of expression)
~/	Divide (Return Integer)
++	Increment Operator
--	Decrement Operator

Dart Operators(2)

Relational Operator

- Relational operator shows the relation between the two operands.
- The relational operator can have only two values either *true* or *false*.
- There are four relational operators are there.

Operator	Meaning
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To

Dart Operators(3)

Equality Operator

- Equality operator checks whether two objects are equal or not.
- Same as relational operator they contain Boolean value either *true* or *false*.
- There are two equality operators are there.

Operator	Meaning
<code>==</code>	Equal to
<code>!=</code>	Not Equal to

- Here remember that two strings are equivalent only if they contain the same character sequence. Here the sequence of character is important rather than the number of character.

Dart Operators(4)

Type Test Operator

- Type test operators are used to check the data type.

Operator	Meaning
is	True if the object has specified type
is!	True if the object has not specified type

Assignment Operator

- Dart has two assignment operators.

Operator	Meaning
=	Assign value from right operand to left operand
??=	Assign the value only if the variable is null

- Here note that `(+=, -=, *=, /=)` are also a type of assignment operator.
But as they are just a shortcut way. I didn't mention them.

Dart Operators(5)

Logical Operator

- Logical operators are used to combine two or more conditions.
- Logical operators return a Boolean value of *true* or *false*.
- There are three logical operators are there:

Operator	Meaning
&& (AND)	Returns true if all conditions are true
(OR)	Returns true if any one condition is true
! (NOT)	Returns the inverse of the result

Ternary operator

- Dart has two special operators which can help you to evaluate small expressions that might require an entire if-else statement.

Condition ? expr1 : expr2

- Here if the condition is true then the **expr1** is evaluated and returns its value. Otherwise, **expr2** is evaluated and returns the value.

Example: ternary operator

```
void main() {  
    var a = 25;  
    var res = a > 15 ? "value greater than 15" : "value lesser than or equal to 15";  
    print(res);  
}
```

Output

value greater than 15

Function(contd..)

- Complete function in action

```
void main(){  
    print("Dart Program To Add Two Numbers Using Function.");  
    var sum = add(10,20);  
    print("Sum Of Given No. Is : ${sum}");  
}
```

```
int add(int n1, int n2){  
    int result;  
    result = n1+n2;  
    return result;  
}
```

Basic Functions(contd..)

The screenshot shows a Visual Studio Code interface with a dark theme. The top bar includes 'Terminal' and 'Help' on the left, and the file path 'collections.dart - flutter_dart_apps - Visual Studio Code' on the right. The main editor area displays the following Dart code:

```
terminal Help collections.dart - flutter_dart_apps - Visual Studio Code

collections.dart X {} launch.json

collections.dart > main
  Run | Debug
1 main() {
2   int wall1 = squarefeet(10, 10);
3   int wall2 = squarefeet(10, 15);
4   int wall3 = squarefeet(10, 20);
5   int wall4 = squarefeet(10, 25);
6   int ceiling = squarefeet(20, 25);
7
8   double paint = paintNeeded(wall1, wall2, wall3, wall4, ceiling);
9   print('you ${paint} gallons of paint is required');
10 }
11
12 int squarefeet(int width, int length) {
13   return width * length;
14 }
15
16 double paintNeeded(int wall1, int wall2, int wall3, int wall4, int ceiling) {
17   int footage = wall1 + wall2 + wall3 + wall4 + ceiling;
18   return footage / 10;
19 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
you 120.0 gallons of paint is required
Exited

Optional Parameters

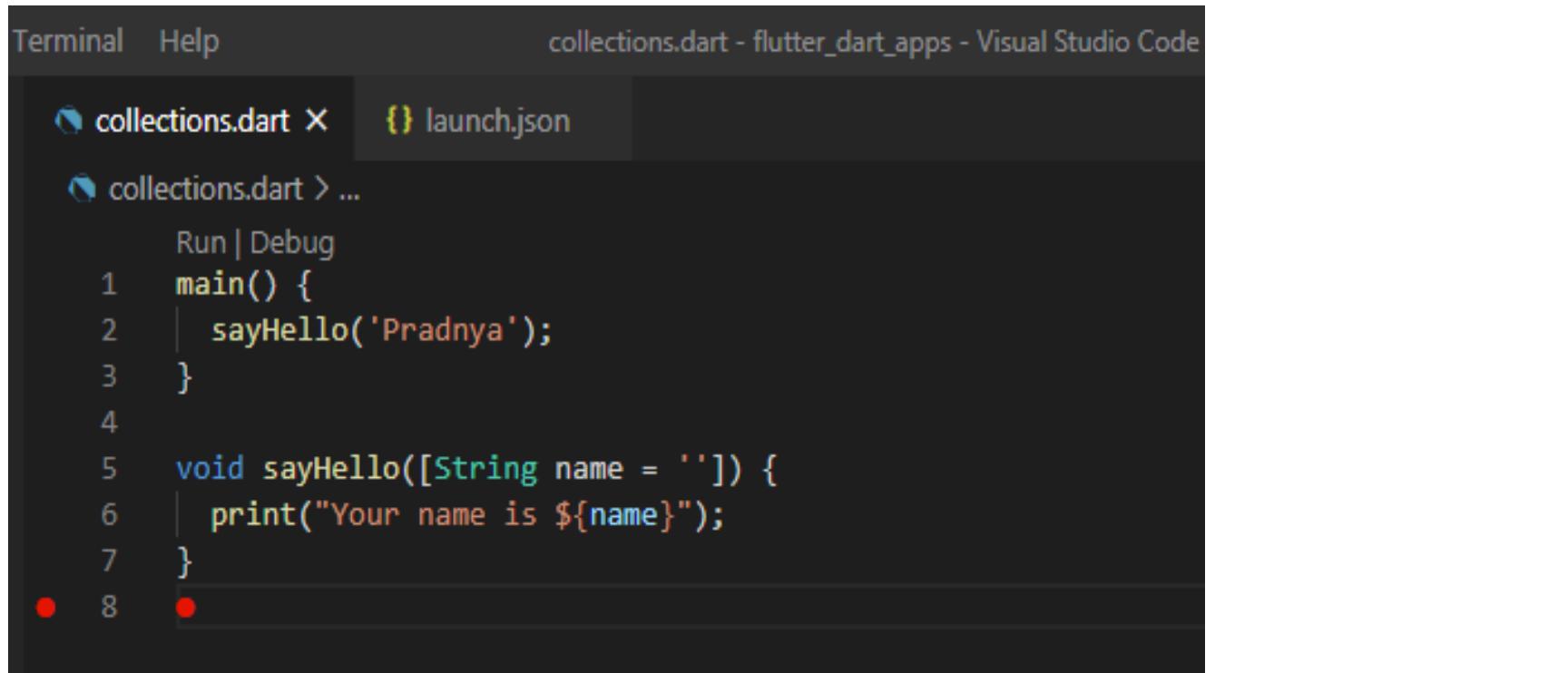
- In dart, we can set any parameter as optional; which allows function with optional parameter(if any) to be called/executed even if values for those parameters is not being provided
- Optional parameters are declared after required parameters of the function
- There are two types of optional parameters in dart:
 - **Optional positional parameter**
 - **Optional named parameter**

Optional positional parameter

- Can be specified by having individual parameter separated by commas and enclosed within square [] bracket
- Calling a function having optional positional parameter defined may specify variable number of arguments
- Syntax:

```
void function_name(param1, [optional_param_1,  
optional_param_2]) { }
```

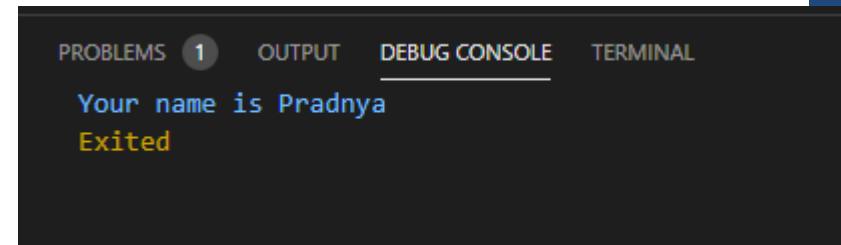
Optional positional parameter(contd..)



The screenshot shows a Visual Studio Code interface with a dark theme. In the top left, there are 'Terminal' and 'Help' tabs. The title bar says 'collections.dart - flutter_dart_apps - Visual Studio Code'. The file 'collections.dart' is open, showing the following code:

```
1 main() {
2   sayHello('Pradnya');
3 }
4
5 void sayHello([String name = '']) {
6   print("Your name is ${name}");
7 }
8
```

The code defines a main function that calls a sayHello function with the argument 'Pradnya'. The sayHello function has an optional positional parameter 'name' with a default value of an empty string. The print statement inside sayHello prints "Your name is \${name}".

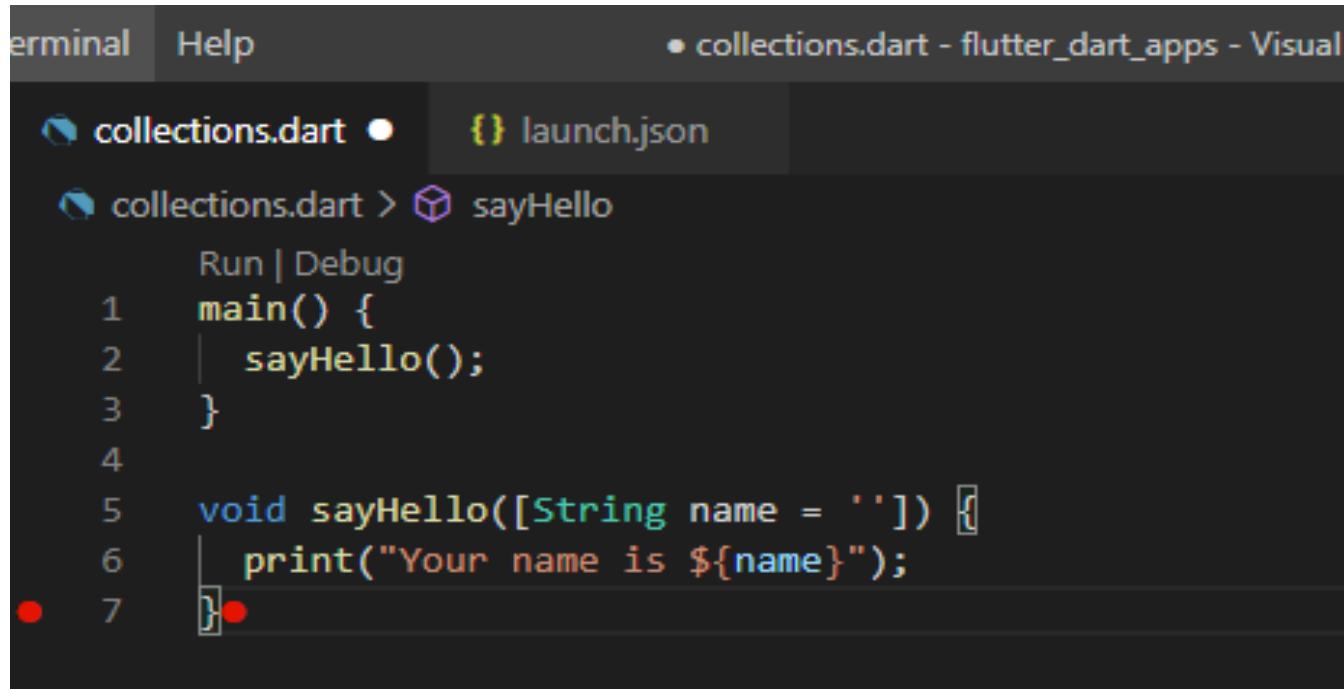


The screenshot shows the 'TERMINAL' tab in Visual Studio Code. The output is:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
Your name is Pradnya
Exited
```

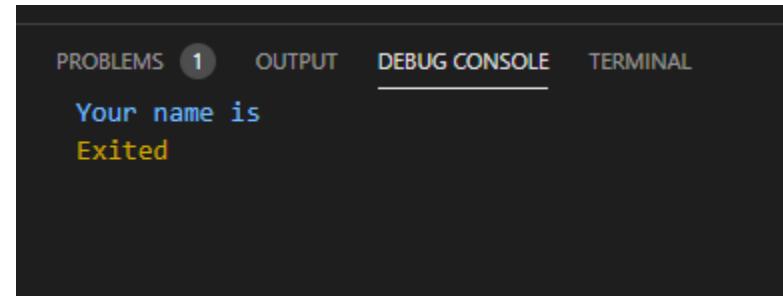
The terminal output shows the string "Your name is Pradnya" followed by the word "Exited", indicating the program has completed its execution.

Optional positional parameter(contd.)



A screenshot of the Visual Studio Code interface. The title bar shows "terminal" and "Help". The top right has a dropdown menu with items like "collections.dart - flutter_dart_apps - Visual Studio Code". The left sidebar shows a tree view with "collections.dart" selected, and "sayHello" is expanded, showing "Run | Debug". The main editor area contains the following Dart code:

```
1 main() {  
2   sayHello();  
3 }  
4  
5 void sayHello([String name = '']) {  
6   print("Your name is ${name}");  
7 }
```



The bottom right corner shows the VS Code terminal window. The tabs at the top are "PROBLEMS" (with a count of 1), "OUTPUT", "DEBUG CONSOLE", and "TERMINAL". The terminal output is:

```
Your name is  
Exited
```

Optional positional parameter(contd.)

The screenshot shows a Visual Studio Code interface with a dark theme. The top bar includes 'Terminal' and 'Help' on the left, and the title 'collections.dart - flutter_dart_apps - Visual Studio Code' on the right. The main area displays a Dart file named 'collections.dart'. The code contains a main function that calls a download function twice. The download function has a parameter 'file' and an optional parameter 'open' with a default value of false. A yellow lightbulb icon is shown above the download function, indicating a suggestion or error. The code is as follows:

```
1  main() {
2      download('myfile.txt');
3      download('myfile2.txt', true);
4  }
5
6  void download(String file, [bool open = false]) {
7      print("downloading ${file}");
8      if (open) print('Opening ${file}');
9  }
```

The bottom part of the screenshot shows the 'TERMINAL' tab in the VS Code interface. It displays the output of the program's execution. The output shows the program printing 'downloading myfile.txt', 'downloading myfile2.txt', and 'Opening myfile2.txt', followed by the message 'Exited'.

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
downloading myfile.txt
downloading myfile2.txt
Opening myfile2.txt
Exited
```

Optional Named Parameters

- Named positional parameters can be specified by having individual parameter name separated by commas and enclosed within curly brackets({})
- **Syntax:**

```
void function_name(param1, {optional_param1, optional_param2})  
{ }
```

- Calling a function having optional named parameter, individual parameter name must be specified while value is being passed
- **Syntax:**

```
function_name(optional_param1:value,...);
```

Optional Named Parameters(contd..)

The screenshot shows a code editor window with a dark theme. At the top, there are two tabs: 'collections.dart X' and '{} launch.json'. Below the tabs, the file 'collections.dart' is open, showing the following code:

```
collections.dart X {} launch.json

collections.dart > main
Run | Debug
1 main() {
2     int footage = squarefeet(length: 10, width: 20);
3     print('Your footage is ${footage}');
4 }
5
6 int squarefeet({int width, int length}) {
7     return width * length;
8 }
9
```

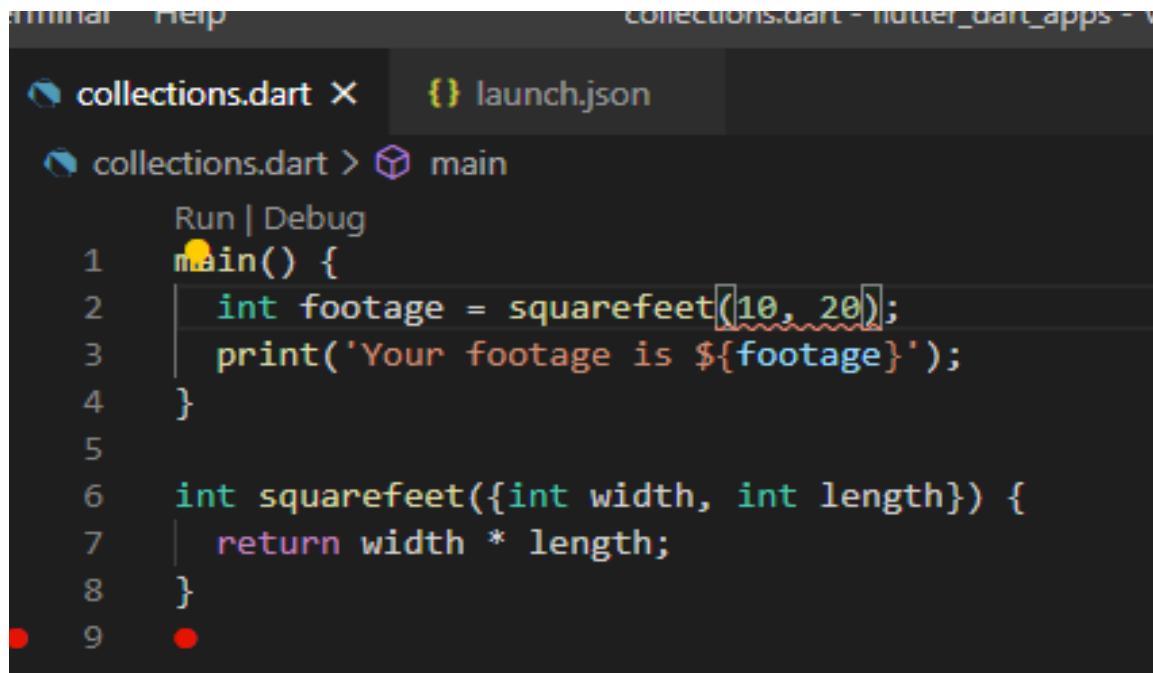
The code defines a main function that calls a squarefeet function with optional named parameters. The squarefeet function returns the product of its width and length parameters.

The screenshot shows the VS Code terminal window at the bottom of the interface. It displays the following output:

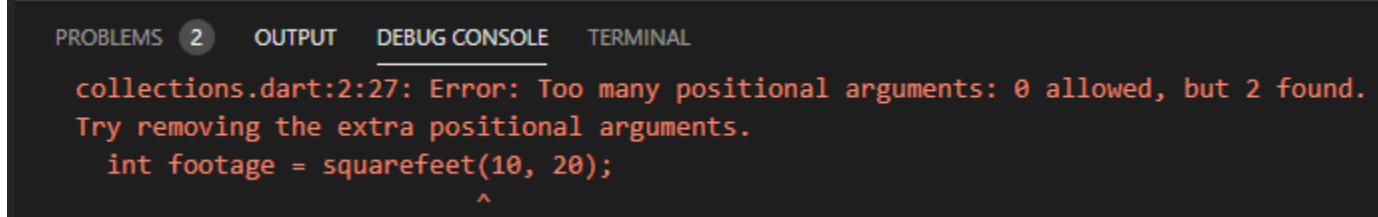
```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
Connecting to VM Service at ws://127.0.0.1:54565/agZWv1P7LPU=/ws
Your footage is 200
Exited
```

The terminal shows the connection to the VM service and the output of the print statement from the Dart code, which is 'Your footage is 200'. The process exits after printing the result.

Optional Named Parameters(contd..)



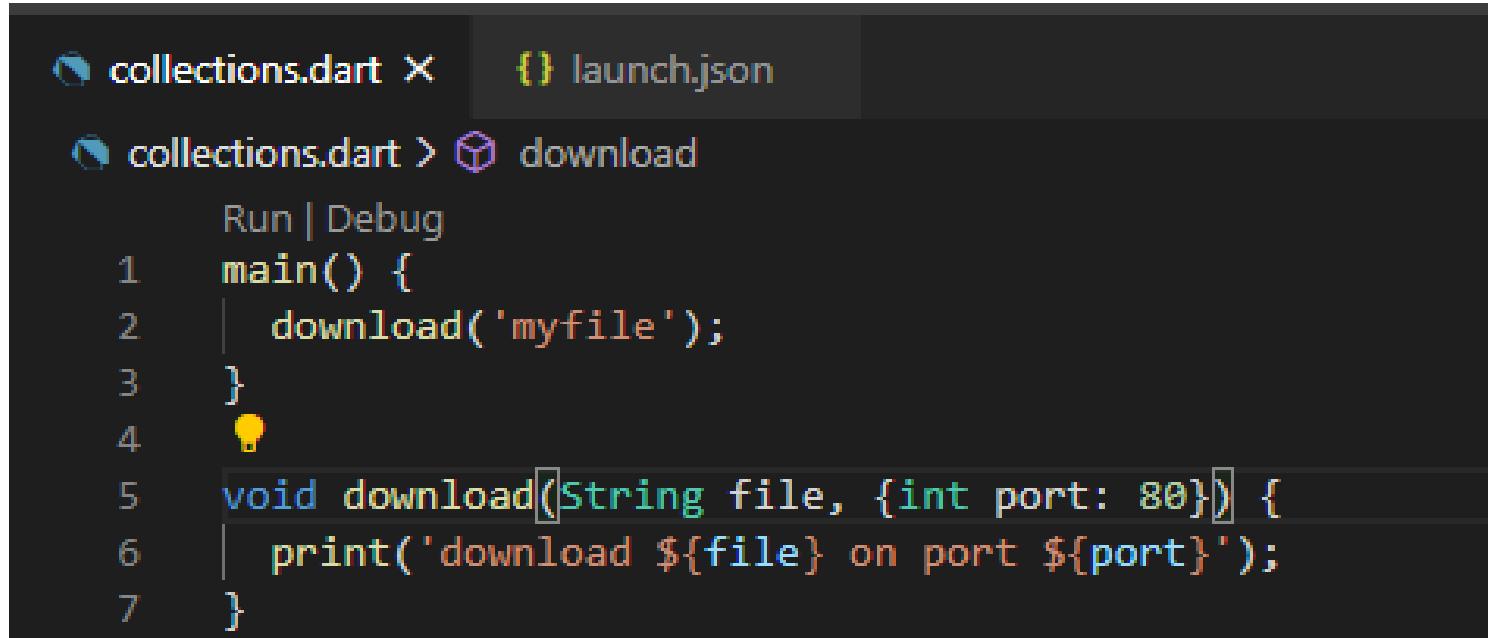
```
collections.dart X {} launch.json
collections.dart > main
Run | Debug
1 main() {
2     int footage = squarefeet(10, 20);
3     print('Your footage is ${footage}');
4 }
5
6 int squarefeet({int width, int length}) {
7     return width * length;
8 }
9 ●
```



PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
collections.dart:2:27: Error: Too many positional arguments: 0 allowed, but 2 found.
Try removing the extra positional arguments.
    int footage = squarefeet(10, 20);
                           ^
```

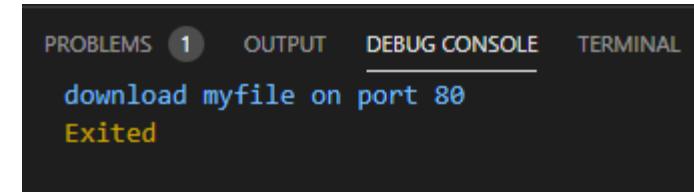
Optional Named Parameters with default values(contd..)



The screenshot shows a code editor interface with a dark theme. At the top, there are two tabs: 'collections.dart' and '{} launch.json'. Below the tabs, the file 'collections.dart' is open, showing the following code:

```
Run | Debug
1 main() {
2   | download('myfile');
3 }
4 🌟
5 void download(String file, {int port: 80}) {
6   | print('download ${file} on port ${port}');
7 }
```

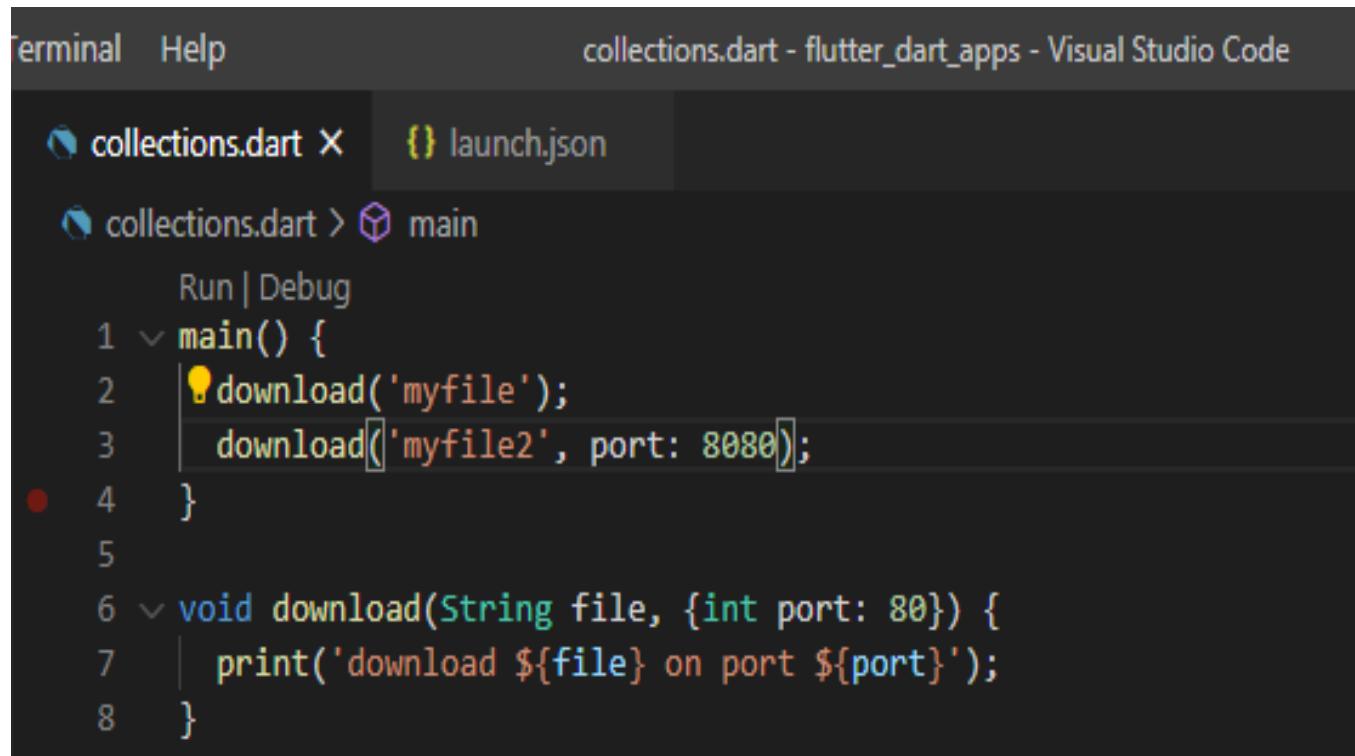
A yellow lightbulb icon is positioned between line 4 and 5, indicating a suggestion or warning.



The screenshot shows the 'DEBUG CONSOLE' tab in the VS Code interface. The output window displays the following text:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
download myfile on port 80
Exited
```

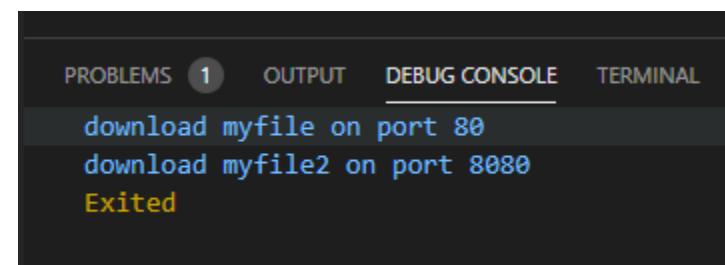
Optional Named Parameters with default values(contd..)



The screenshot shows a Visual Studio Code interface with the following details:

- Terminal** Help collections.dart - flutter_dart_apps - Visual Studio Code
- File Explorer**: Shows two files: collections.dart (selected) and launch.json.
- Code Editor**: Displays the following Dart code:

```
1  main() {  
2      download('myfile');  
3      download(['myfile2', port: 8080]);  
4  }  
5  
6  void download(String file, {int port: 80}) {  
7      print('download ${file} on port ${port}');  
8  }
```



The screenshot shows the DEBUG CONSOLE tab with the following output:

PROBLEMS	1	OUTPUT	DEBUG CONSOLE	TERMINAL
download myfile on port 80				
download myfile2 on port 8080				
Exited				

Single-Line Functions

- Dart makes it possible to declare short functions more compactly
- The syntax **=>** is used for separating the return type, name, and parameters of a single-line function from its implementation
- As with regular functions, specifying the return type of a single-line function is optional

Single-line functions do not use return statements.

Instead, the return value is implied

- The evaluated result of the expression on the right of the **=>** is the return value

Single-Line Functions(contd..)

Examples:

// just as valid as next version

- sayHello(String name) => print("Hello, \$name");

// same as previous but with void

- void sayHello(String name) => print("Hello, \$name");

// think of as return num1 * num2;

- int multiply(int num1, int num2) => num1 * num2;

// return is implied

- int luckyNumber() => 7;

- **Single-line functions are simply another way of defining the same thing which is provided by the language for convenience**

main function()

- Every dart program to have top-level main () function definition
- There can be only one main () function in dart program
- It serves as entry point for every dart program or app, execution of a dart program or app start with main() function
- main() function is responsible for execution of all user defined statements, functions and library functions
- main() function is further structured into variable declaration, function declaration and user defined executable statements
- main() function return void and can have an optional parameter List<String> as arguments
- **Syntax:**

```
void main() {  
    // main function body  
}
```

Exception Handling

Error Handling

- What is Exception and Exception Handling?
- How to handle Exceptions using:
 - TRY
 - CATCH
 - ON
 - FINALLY
- What is StackTrace?
- How to create our own exception handling class

Exception Handling

- **Exception** is a runtime unwanted event that disrupts the flow of code execution or application crashes
 - It can be occurred because of programmer's mistake or by wrong user input.
- To handle such events at runtime is called **Exception Handling**.

Types of Exceptions in Dart

N
o

Exception and Description

1

`IntegerDivisionByZeroException`

Thrown when a number is divided by zero.

2

`IOException`

Base class for all Input-Output related exceptions.

3

`DeferredLoadException`

Thrown when a deferred library fails to load.

4

`FormatException`

An exception is thrown when a string or some other data does not have an expected format and cannot be parsed or processed.

5

`IsolateSpawnException`

Thrown when an isolate cannot be created.

6

`Timeout`

Thrown when a scheduled timeout happens while waiting for an async result.

Exception

xyz.dart X

> xyz.dart > ...

Run | Debug

```
void main() {
  int result = 12 ~/ 4;
  print(result);
}
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

Connecting to VM Service at ws://127.0.0.1:53495/MvkuwiNztDg=/ws

3

Exited

a.dart X

.vscode > a.dart > main

Run | Debug

```
void main() {
  int result = 12 ~/ 0;
  print('result is ${result}');
}
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL

[Running] dart "c:\Users\Gaurav\Desktop\dart_basics\.vscode\a.dart"

Unhandled exception:

IntegerDivisionByZeroException

```
#0      int.~/ (dart:core-patch/integers.dart:24:7)
#1      main (file:///c:/Users/Gaurav/Desktop/dart_basics/.vscode/a.dart:2:19)
#2      _startIsolate.<anonymous closure> (dart:isolate-patch/isolate_patch.dart:301:19)
#3      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:168:12)
```

try..catch..finally

- In Dart Exceptions can be handle via

Try: In the try block, we write the logical code that can produce the exception

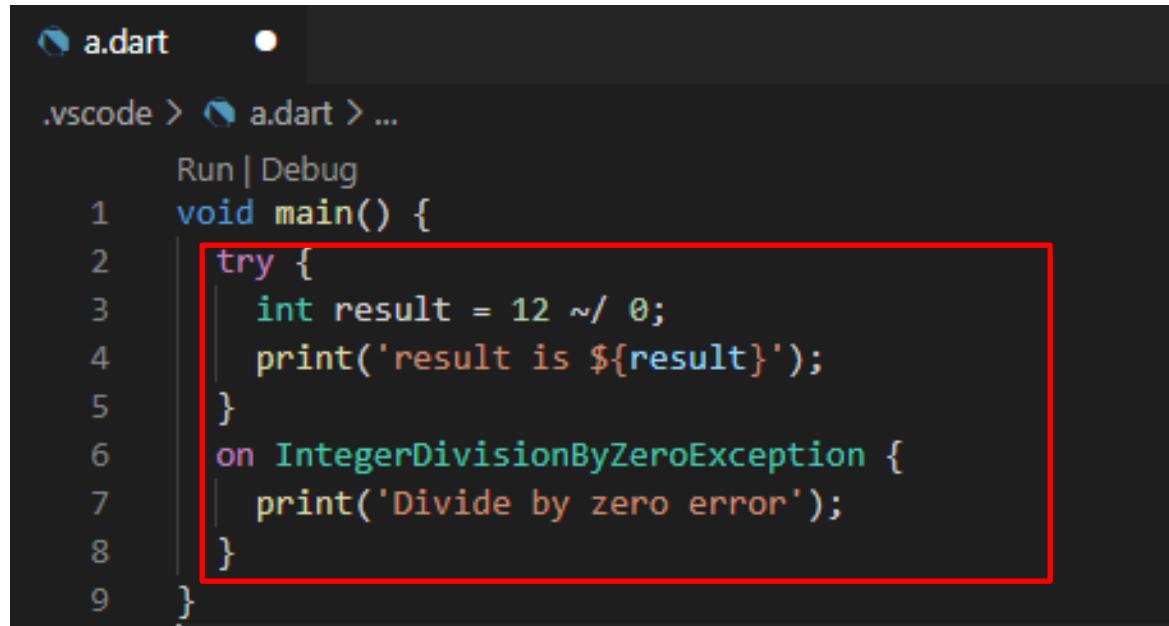
Catch: Catch block is written with try block to catch the general exceptions: In other words, if it is not clear what kind of exception will be produced. Catch block is used.

On: On block is used when it is 100% sure what kind of exception will be thrown.

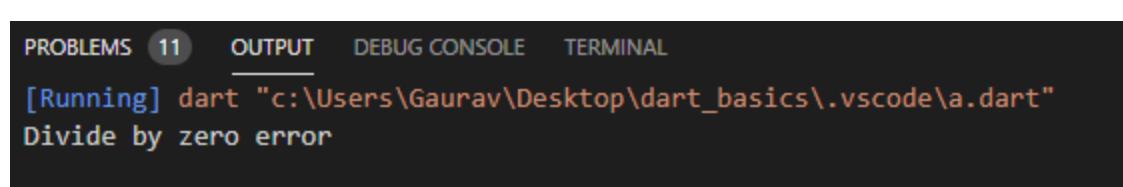
Finally: The finally part is always executed but it is not mandatory.

try clause

Case 1: when you know exception to be thrown use **On clause**



```
a.dart
.vscode > a.dart > ...
Run | Debug
1 void main() {
2   try {
3     int result = 12 ~/ 0;
4     print('result is ${result}');
5   }
6   on IntegerDivisionByZeroException {
7     print('Divide by zero error');
8   }
9 }
```

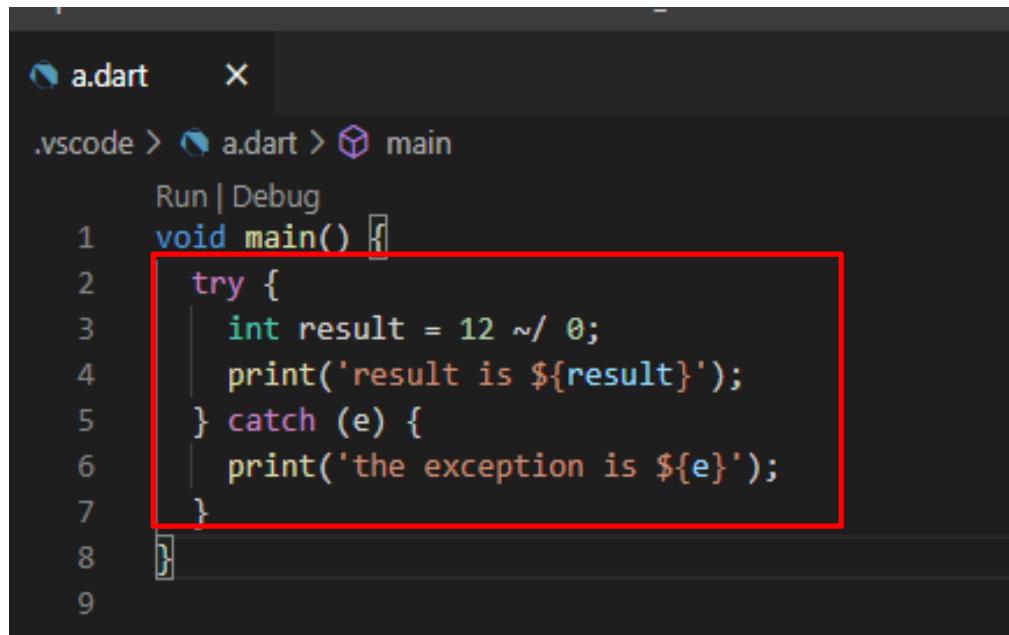
A screenshot of the VS Code interface showing a Dart file named 'a.dart'. The code contains a 'main' function with a 'try' block that attempts to divide 12 by 0. An 'on' clause handles the 'IntegerDivisionByZeroException' that is thrown. The entire try-catch block is highlighted with a red rectangular selection. The status bar at the bottom shows 'PROBLEMS 11'.

```
PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL
[Running] dart "c:\Users\Gaurav\Desktop\dart_basics\.vscode\a.dart"
Divide by zero error
```

The screenshot shows the 'OUTPUT' tab of the VS Code terminal. It displays the command 'dart "c:\Users\Gaurav\Desktop\dart_basics\.vscode\a.dart"' followed by the error message 'Divide by zero error'. The terminal also indicates that there are 11 problems in the project.

try...catch clause

- Case 2: when you do not exception, use **catch clause**



```
a.dart    x
.vscode > a.dart > main
Run | Debug
void main() {
  try {
    int result = 12 ~/ 0;
    print('result is ${result}');
  } catch (e) {
    print('the exception is ${e}');
  }
}
```

A screenshot of a VS Code editor window. The file is named 'a.dart'. The code in the main function contains a try...catch block. The entire try...catch block is highlighted with a red rectangular selection. The code itself is in a monospaced font, with keywords like 'try' and 'catch' in purple, and variable names like 'result' and 'e' in blue.

```
[Running] dart "c:\Users\Gaurav\Desktop\dart_basics\.vscode\a.dart"
the exception is IntegerDivisionByZeroException
```

try...catch clause

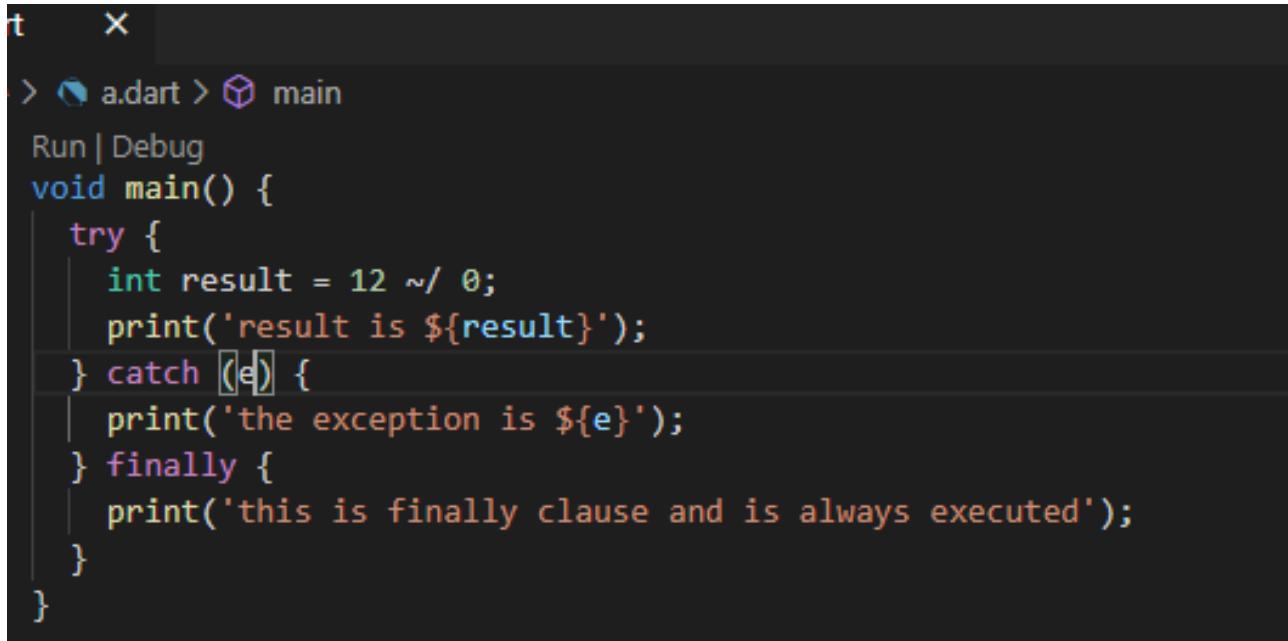
- Case 3: Using **Stack Trace** to know events occurred before exception was thrown

```
Run | Debug
1 void main() {
2     try {
3         int result = 12 ~/ 0;
4         print('result is ${result}');
5     } catch (e, s) {
6         print('the exception is ${e}');
7         print('the exception is ${s}');
8     }
9 }
```

```
[Running] dart "c:\Users\Gaurav\Desktop\dart_basics\.vscode\a.dart"
the exception is IntegerDivisionByZeroException
the exception is #0      int.~/ (dart:core-patch/integers.dart:24:7)
#1      main (file:///c:/Users/Gaurav/Desktop/dart_basics/.vscode/a.dart:3:21)
#2      _startIsolate.<anonymous closure> (dart:isolate-patch/isolate_patch.dart:301:19)
#3      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:168:12)
```

try..catch..finally

- Case 4: Whether there is an exception or not, **finally clause** is always executed

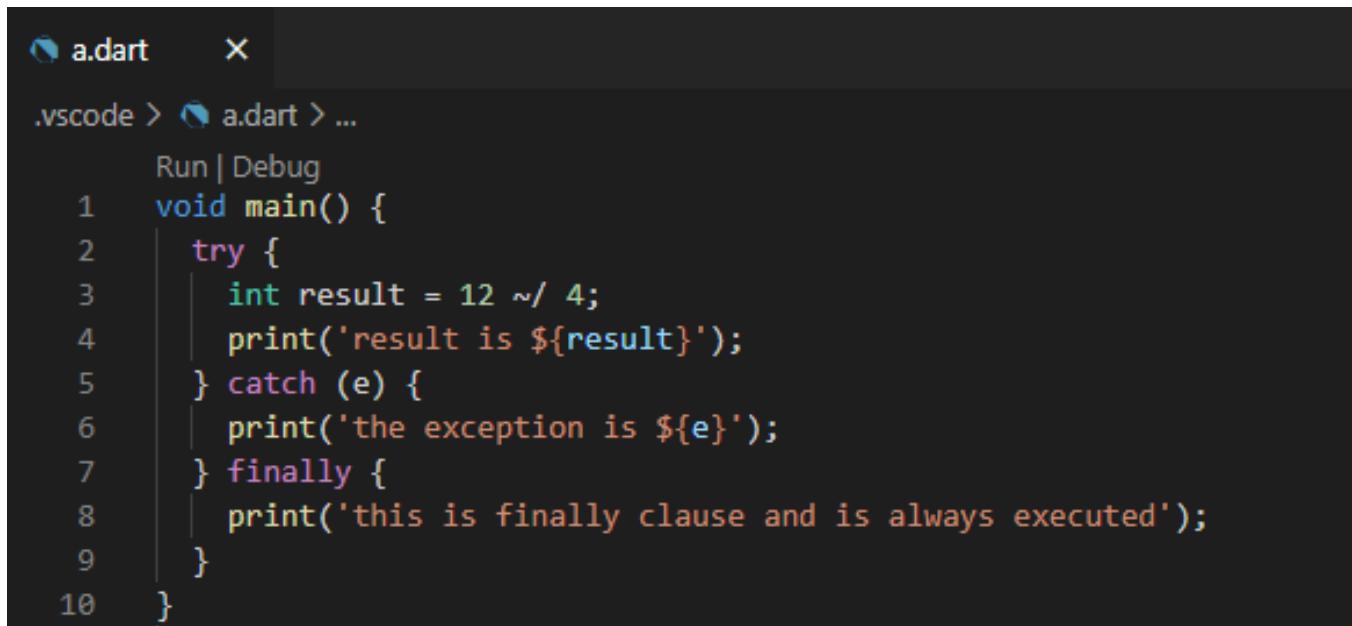


```
t  x
> a.dart > main
Run | Debug
void main() {
  try {
    int result = 12 ~/ 0;
    print('result is ${result}');
  } catch (e) {
    print('the exception is ${e}');
  } finally {
    print('this is finally clause and is always executed');
  }
}
```

```
[Running] dart "c:\Users\Gaurav\Desktop\dart_basics\.vscode\a.dart"
the exception is IntegerDivisionByZeroException
this is finally clause and is always executed
```

try..catch..finally

- Case 4: Whether there is an exception or not, **finally clause** is always executed



```
a.dart  X

.vscode > a.dart > ...

Run | Debug
1 void main() {
2   try {
3     int result = 12 ~/ 4;
4     print('result is ${result}');
5   } catch (e) {
6     print('the exception is ${e}');
7   } finally {
8     print('this is finally clause and is always executed');
9   }
10 }
```

```
[Running] dart "c:\Users\Gaurav\Desktop\dart_basics\.vscode\ a.dart"
result is 3
this is finally clause and is always executed
```

Throwing Exception

- The **throw** keyword is used to explicitly raise an exception.
A raised exception should be handled to prevent the program from exiting abruptly.
- The **syntax** for raising an exception explicitly is –

throw new Exception_name()

Throwing Exception

The screenshot shows a Visual Studio Code interface with the following details:

- Terminal:** Help
- File Explorer:** collections.dart - flutter_dart_apps - Visual Studio Code
- Editor:** The file "collections.dart" is open, showing Dart code for a main function. A specific line of code is highlighted with a red box:

```
1 main(List<String> arguments) {  
2     try {  
3         int age;  
4         int dogyears = 7;  
5         if (age == null) throw new NullThrownError();  
6         print(age * dogyears);  
7     }  
8     on NullThrownError {  
9         print("the value was null!!!");  
10    }  
11    on NoSuchMethodError {  
12        print("Sorry no such method");  
13    }  
14    catch (e) {  
15        print("Unknown error:${e.toString()}");  
16    }  
17    finally {  
18        print("complete");  
19    }  
}
```
- Output:** The terminal output shows the execution results:

```
the value was null!!!  
complete  
Exited
```
- Bottom Navigation:** PROBLEMS 1, OUTPUT, DEBUG CONSOLE, TERMINAL

Throwing Exception

The screenshot shows a Dart code editor interface with the following details:

- File Tabs:** collections.dart X, sample.dart, launch.json
- Current File:** collections.dart > main
- Code Content:**

```
1 main(List<String> arguments) {
2   try {
3     int age;
4     int dogyears = 8;

5
6     if (dogyears != 7) throw new Exception('Dog years must be 7');
7     if (age == null) throw new NullThrownError();
8     print(age * dogyears);
9   } on NullThrownError {
10     print("the value was null!!!!");
11   } on NoSuchMethodError {
12     print("Sorry no such method");
13   } catch (e) {
14     print("Unknown error:${e.toString()}");
15   } finally {
16     print("complete");
17   }
18 }
19 }
```
- Red Box Selection:** A red box highlights the following code block:

```
if (dogyears != 7) throw new Exception('Dog years must be 7');
if (age == null) throw new NullThrownError();
```
- Output Panel:** PROBLEMS 1, OUTPUT, DEBUG CONSOLE, TERMINAL
Unknown error:Exception: Dog years must be 7
complete
Exited

Custom Exceptions

- Every exception type in Dart is a subtype of the built-in class **Exception**.
- Dart enables creating custom exceptions by extending the existing ones.
- The syntax for defining a custom exception is as given below –

```
class Custom_exception_Name implements Exception  
{ // can contain constructors, variables and methods }
```

Custom Exceptions(contd..)

```
1 class AmtException implements Exception {  
2     String errMsg() => 'Amount should be greater than zero';  
3 }  
4  
Run | Debug  
5 void main() {  
6     try {  
7         withdraw_amt(-1);  
8     } catch (e) {  
9         print(eerrMsg());  
10    } finally {  
11        print('Ending requested operation.....');  
12    }  
13 }  
14  
15 void withdraw_amt(int amt) {  
16     if (amt <= 0) {  
17         throw new AmtException();  
18     }  
19 }
```

Amount should be greater than zero
Ending requested operation.....
PS C:\Users\Gaurav\Desktop\flutter_dart_apps>

Scope

Private and Public Variables

- **Public variables**

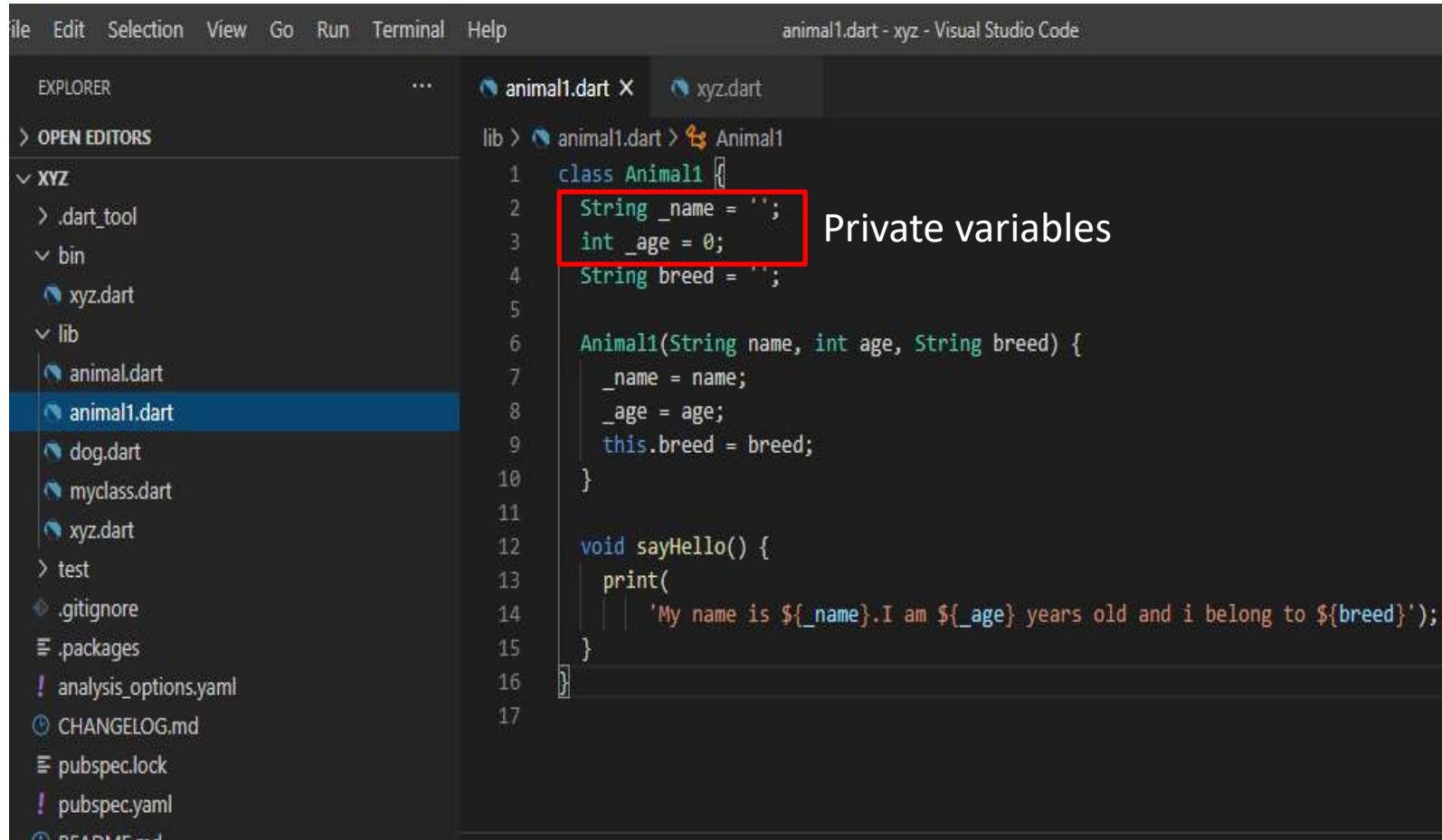
Variables that are visible to all classes

- **Private variables**

Variables that are visible only to the class to which they belong

They are defined using underscore symbol(eg: int _age)

Private and Public Scope



The screenshot shows a Visual Studio Code interface with the following details:

- File Menu:** file Edit Selection View Go Run Terminal Help
- Title Bar:** animal1.dart - xyz - Visual Studio Code
- Explorer:** Shows the project structure under XYZ:
 - .dart_tool
 - bin
 - xyz.dart
 - lib
 - animal.dart
 - animal1.dart
 - dog.dart
 - myclass.dart
 - xyz.dart
 - test
 - .gitignore
 - .packages
 - analysis_options.yaml
 - CHANGELOG.md
 - pubspec.lock
 - pubspec.yaml
 - README.md
- Open Editors:** animal1.dart (active) and xyz.dart
- Code Editor:** The code for the Animal1 class is displayed:

```
1 class Animal1 {
2     String _name = '';
3     int _age = 0;
4     String breed = '';
5
6     Animal1(String name, int age, String breed) {
7         _name = name;
8         _age = age;
9         this.breed = breed;
10    }
11
12    void sayHello() {
13        print(
14            'My name is ${_name}. I am ${_age} years old and i belong to ${breed}');
15    }
16 }
17
```

A red box highlights the private variables `_name` and `_age`, and a callout bubble labeled "Private variables" points to them.

Private and Public Scope(contd..)

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** xyz.dart - xyz - Visual Studio Code
- Explorer:** Shows the project structure under 'XYZ': .dart_tool, bin (with xyz.dart selected), lib (with animal.dart, animal1.dart, dog.dart, myclass.dart, xyz.dart), test, .gitignore, and .packages.
- Editor:** The file xyz.dart is open, showing the following code:

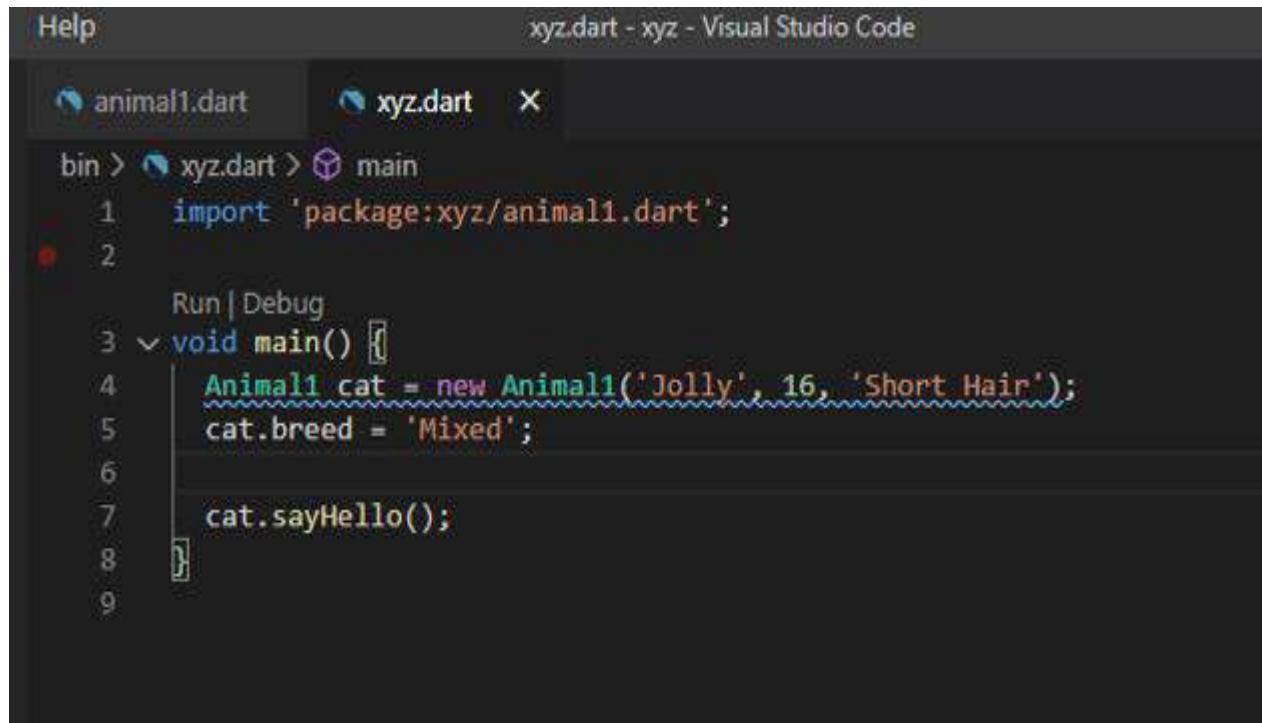
```
1 import 'package:xyz/animal1.dart';
2
3 void main() {
4     Animal1 cat = new Animal1('Jolly', 16, 'Short Hair');
5     cat.sayHello();
6 }
7
```
- Terminal:** At the bottom, the terminal window shows the output of the application:

```
Connecting to VM Service at ws://127.0.0.1:50818/f0kNgBgeT5I=/ws
My name is Jolly.I am 16 years old and i belong to Short Hair
Exited
```

The terminal window displays the following output:

```
Connecting to VM Service at ws://127.0.0.1:50818/f0kNgBgeT5I=/ws
My name is Jolly.I am 16 years old and i belong to Short Hair
Exited
```

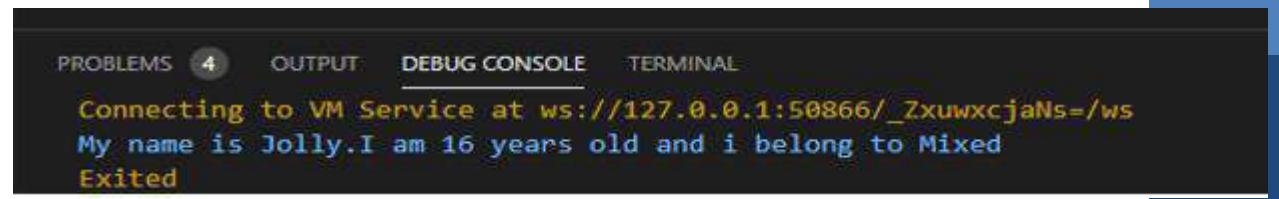
Private and Public Scope(contd..)



```
Help xyz.dart - xyz - Visual Studio Code

animal1.dart xyz.dart X

bin > xyz.dart > main
1 import 'package:xyz/animal1.dart';
2
3 Run | Debug
4 void main() {
5     Animal1 cat = new Animal1('Jolly', 16, 'Short Hair');
6     cat.breed = 'Mixed';
7
8     cat.sayHello();
9 }
```



PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

```
Connecting to VM Service at ws://127.0.0.1:50866/_ZxuwxcjaNs=/ws
My name is Jolly.I am 16 years old and i belong to Mixed
Exited
```

Private and Public Scope(private variable modified)

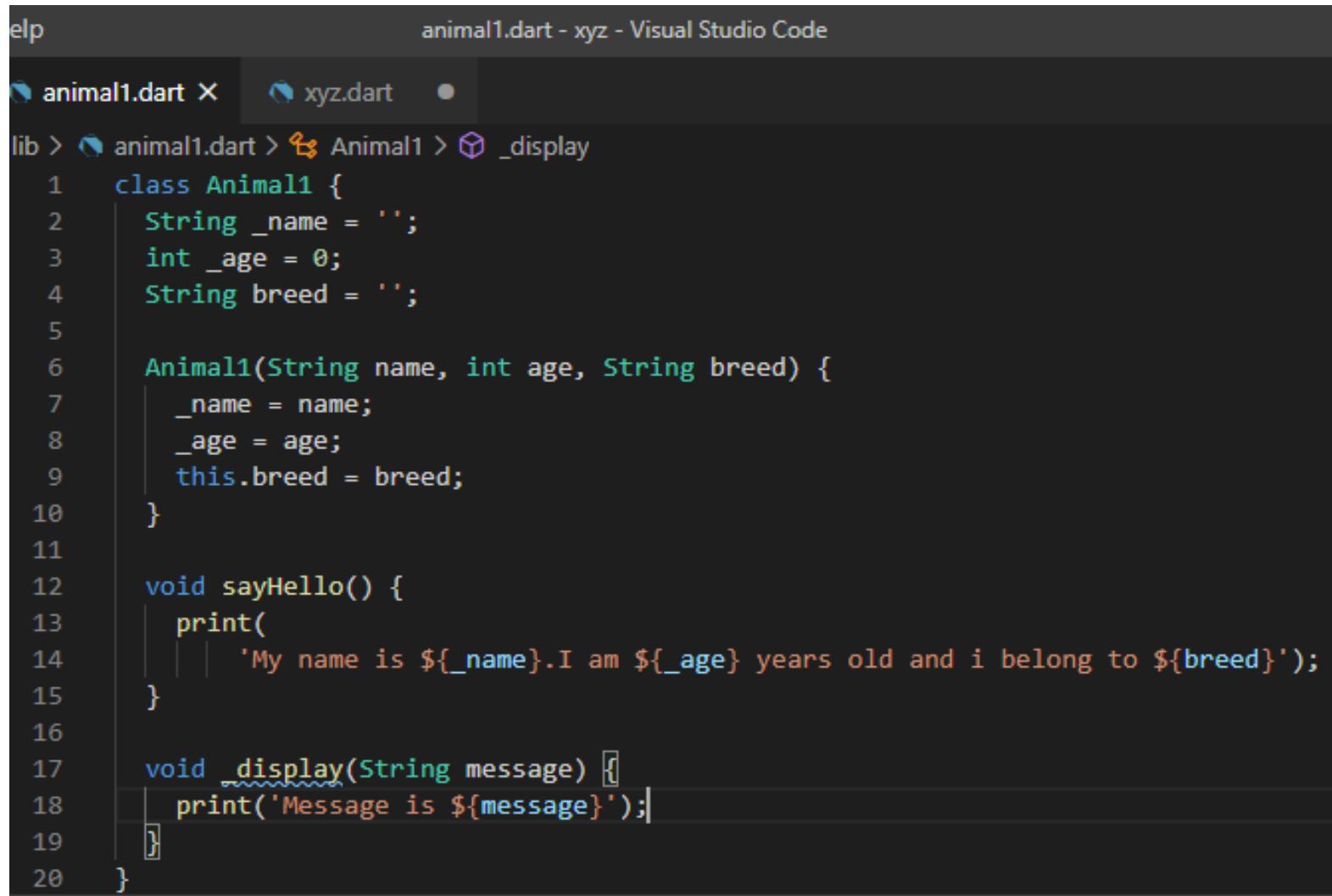
The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files `animal1.dart` and `xyz.dart`.
- Editor:** The `xyz.dart` file is open, showing the following code:

```
bin > xyz.dart > main
1   import 'package:xyz/animal1.dart';
2
3   void main() {
4     Animal1 cat = new Animal1('Jolly', 16, 'Short Hair');
5     cat.breed = 'Mixed';
6     cat._name = 'Muffin';
7
8     cat.sayHello();
9 }
10
```
- Terminal:** Displays error messages from the VM Service:

```
Connecting to VM Service at ws://127.0.0.1:50874/4oER14e0PBM=/ws
bin/xyz.dart:6:7: Error: The setter '_name' isn't defined for the class 'Animal1'.
- 'Animal1' is from 'package:xyz/animal1.dart' ('lib/animal1.dart').
Try correcting the name to the name of an existing setter, or defining a setter or field named '_name'.
  cat._name = 'Muffin';
          ^^^^^^
Exited (1)
```

Private and Public Scope(private method declared)



The screenshot shows a Visual Studio Code interface with the following details:

- Title Bar:** Help animal1.dart - xyz - Visual Studio Code
- File Explorer:** Shows two files: animal1.dart (selected) and xyz.dart.
- Code Editor:** Displays the code for the Animal1 class. The code includes private fields (_name, _age), a constructor, a public method sayHello(), and a private method _display().
- Code Navigation:** A sidebar on the right shows the navigation path: lib > animal1.dart > Animal1 > _display.

```
1  class Animal1 {
2      String _name = '';
3      int _age = 0;
4      String breed = '';
5
6      Animal1(String name, int age, String breed) {
7          _name = name;
8          _age = age;
9          this.breed = breed;
10     }
11
12     void sayHello() {
13         print(
14             'My name is ${_name}.I am ${_age} years old and i belong to ${breed}');
15     }
16
17     void _display(String message) {
18         print('Message is ${message}');
19     }
20 }
```

Private and Public Scope(private method accessed)

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files `animal1.dart` and `xyz.dart`. `xyz.dart` is the active file.
- Code Editor:** Displays the `main` function of `xyz.dart`. The line `cat._display('meow');` is highlighted with a yellow squiggle under `_display`, indicating a syntax error.
- Terminal:** Shows the output of the Dart VM service connecting and an error message about the undefined `_display` method.

```
Help xyz.dart - xyz - Visual Studio Code

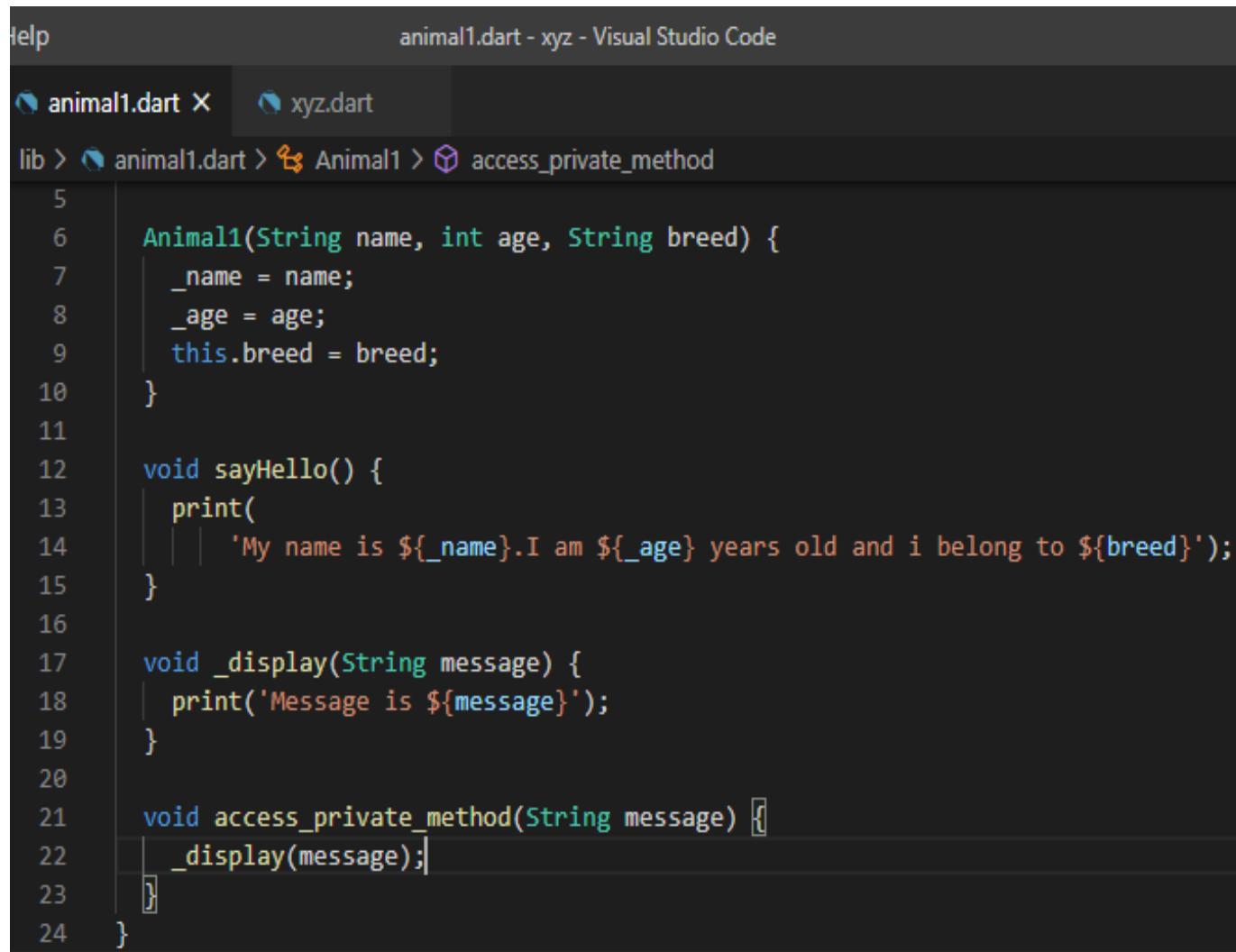
animal1.dart xyz.dart

bin > xyz.dart > main
1 import 'package:xyz/animal1.dart';
2
3 Run | Debug
4 void main() {
5   Animal1 cat = new Animal1('Jolly', 16, 'Short Hair');
6   cat.breed = 'Mixed';
7   cat._display('meow');
8   cat.sayHello();
9 }
10

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL

Connecting to VM Service at ws://127.0.0.1:50930/IEW41TKmpfI=/ws
bin/xyz.dart:6:7: Error: The method '_display' isn't defined for the class 'Animal1'.
- 'Animal1' is from 'package:xyz/animal1.dart' ('lib/animal1.dart').
Try correcting the name to the name of an existing method, or defining a method named '_display'.
  cat._display('meow');
  ^^^^^^
Exited (1)
```

Private and Public Scope(solution: private method accessed through encapsulation)



The screenshot shows a Visual Studio Code interface with the following details:

- Title Bar:** Help, animal1.dart - xyz - Visual Studio Code
- File Explorer:** Shows files: animal1.dart X, xyz.dart.
- Search Bar:** lib > animal1.dart > Animal1 > access_private_method
- Code Editor:** Displays the following Dart code for the Animal1 class:

```
5
6     Animal1(String name, int age, String breed) {
7         _name = name;
8         _age = age;
9         this.breed = breed;
10    }
11
12    void sayHello() {
13        print(
14            'My name is ${_name}. I am ${_age} years old and i belong to ${breed}');
15    }
16
17    void _display(String message) {
18        print('Message is ${message}');
19    }
20
21    void access_private_method(String message) {
22        _display(message);
23    }
24 }
```

Private and Public Scope(solution: private method accessed through encapsulation)

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files `animal1.dart` and `xyz.dart`.
- Editor:** The `xyz.dart` file contains the following code:

```
bin > xyz.dart > ...
1 import 'package:xyz/animal1.dart';
2
3 void main() {
4     Animal1 cat = new Animal1('Jolly', 16, 'Short Hair');
5     cat.breed = 'Mixed';
6     cat.access_private_method('meow');
7     cat.sayHello();
8 }
9
```
- Output Panel:** Shows the execution results:

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL
Connecting to VM Service at ws://127.0.0.1:51048/4JJfHr3en2I=/ws
Message is meow
My name is Jolly.I am 16 years old and i belong to Mixed
Exited
```

Getters and Setters

- Getters and Setters are special class methods that is used to initialize and retrieve the values of the class field respectively
- **Setter method** is used to initialize or set respective class fields
- **Getter method** is used to retrieve respective class fields
- All classes have default getter and setter method associated with it

However, you are free to override default ones by implementing getter and setter method explicitly

Default Getter and Setter

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure with files like pqr.dart, Student.dart, and various configuration files. The main editor window shows the Student.dart file with a class definition:

```
lib > Student.dart > Student
1   class Student{
2     String name;
3 }
```

The Output panel at the bottom right shows the VM service connection status and some log messages:

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

Connecting to VM Service at ws://127.0.0.1:58371/Byxz9
Peter
Exited

The screenshot shows the Visual Studio Code interface. The Explorer sidebar displays the project structure with files like pqr.dart, main.dart, and various configuration files. The main editor window shows the pqr.dart file with a main function:

```
bin > pqr.dart > main
1
2 import 'package:pqr/Student.dart';
3
4 void main(List<String> arguments) {
5   var s1= new Student();
6   s1.name="Peter"; //calling default setter to set value
7   print(s1.name); //calling default getter to get value
8 }
```

The Output panel at the bottom right shows the VM service connection status and some log messages:

pqr.dart - pqr - Visual Studio Code

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

Connecting to VM Service at ws://127.0.0.1:58371/Byxz9
Peter
Exited

Custom Getters and Setters(contd..)

Defining a getter

The getters are defined using the get keyword with no parameters and returns a value.

Syntax:-

```
return_type get field_name  
{  
}
```

Defining a setter

The setters are defined using the set keyword with one parameter and no return value.

Syntax:-

```
set field_name  
{  
}
```

Custom Getters and Setters(contd..)

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "PQR". The "lib" folder contains "Student.dart" and "pqr.dart". Other files like ".dart_tool", "bin", "test", ".gitignore", ".packages", "analysis_options.yaml", "CHANGELOG.md", and "pubspec.lock" are also listed.
- Code Editor (Right):** The file "Student.dart" is open. The code defines a class "Student" with instance variables "name" and "percent", and custom getter and setter methods for "percentage".

```
1 class Student{
2   String name; //instance variable with default getter and setter
3   double percent;
4
5   void set percentage(double marksObtained)
6   { //Instance variable with custom setter
7
8     percent= (marksObtained/500)*100;
9   }
10
11  double get percentage{ //Instance variable with custom getter
12    return percent;
13  }
14}
```
- Status Bar (Top):** Shows "Student.dart - pqr - Visual Studio Code".

Custom Getters and Setters(contd..)

The screenshot shows a Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** pqr.dart - pqr - Visual Studio Code
- Explorer View:** Shows the project structure:
 - > OPEN EDITORS
 - < PQR
 - .dart_tool
 - bin
 - pqr.dart
 - lib
 - pqr.dart
 - Student.dart
 - > test
 - .gitignore
 - .packages
 - ! analysis_options.yaml
- Code Editor:** The file pqr.dart is open, showing the following Dart code:

```
1 import 'package:pqr/Student.dart';
2
3 void main(List<String> arguments) {
4   var s1= new Student();
5   s1.name="Peter"; //calling default setter to set value
6   print(s1.name); //calling default getter to get value
7
8   s1.percentage=450;
9   print(s1.percentage);
10 }
11 }
```
- Terminal View:** Shows the output of the application:

```
Connecting to VM Service at ws://127.0.0.1:59406/_O3PyKBiABI=/ws
Peter
90.0
Exited
```

The terminal window displays the following output:

```
Connecting to VM Service at ws://127.0.0.1:59406/_O3PyKBiABI=/ws
Peter
90.0
Exited
```

Static members

- Static keyword is used for memory management of global data members
- Static keyword can be applied to fields and methods of a class
- Static variables and methods are part of class instead of specific instance
- Static data members can be accessed without creating an object of class

Simply put `class_name` before static variable or method name to use them

Static members(contd..)

Static Variables

- Belongs to class instead of specific instance(objects)
- It is common to all instances of class
- Memory allocation to static variable happen only once in class area at time of class loading

Important Points :-

- ▶ Static variables are also known as Class Variables.
- ▶ Static variables can be accessed directly in Static method
- ▶ Single copy of static variable is shared among all the instances of a class

Static members(contd..)

Declaring Static Variables

Static variables can be declared using the **static** keyword followed by data type then variable name.

Syntax:-

```
static [data_type] [variable_name];
```

Accessing Static Variable

Static variable can be accessed directly from the class name itself rather than creating an instance of it.

Syntax:-

```
ClassName.staticVariable;
```

Static Members(contd..)

The screenshot shows two tabs open in Visual Studio Code: 'xyz.dart' and 'stat.dart'. The 'xyz.dart' tab contains a main function that creates four instances of the 'Stat' class. The 'stat.dart' tab shows the definition of the 'Stat' class with a private counter and a constructor that increments it.

```
xyz.dart - xyz - Visual Studio Code
Help xyz.dart stat.dart
bin > xyz.dart > main
1 import 'package:xyz/stat.dart';
2
3 Run | Debug
4 main() {
5     Stat s1 = new Stat();
6     Stat s2 = new Stat();
7     Stat s3 = new Stat();
8     Stat s4 = new Stat();
9 }
```

```
Help stat.dart - xyz - Visual Studio Code
xyz.dart stat.dart
lib > stat.dart > ...
1 class Stat {
2     int _counter = 0;
3
4     Stat() {
5         _counter++;
6         print('there are ${_counter} values');
7     }
8 }
9
```

PROBLEMS 17 OUTPUT DEBUG CONSOLE TERMINAL

```
Connecting to VM Service at ws://127.0.0.1:52352/B82WyyYqW-4=/ws
there are 1 values
there are 1 values
there are 1 values
there are 1 values
Exited
```

Static Members(contd..)

The screenshot shows the Visual Studio Code interface with two files open:

- xyz.dart**: Contains the definition of the `Stat` class with a static counter.
- stat.dart**: Contains the `main` function which creates four instances of `Stat` and prints the current value of the static counter.

xyz.dart Content:

```
1 class Stat {  
2     static int _counter = 0;  
3  
4     Stat() {  
5         _counter++;  
6         print('there are ${_counter} values');  
7     }  
8 }  
9
```

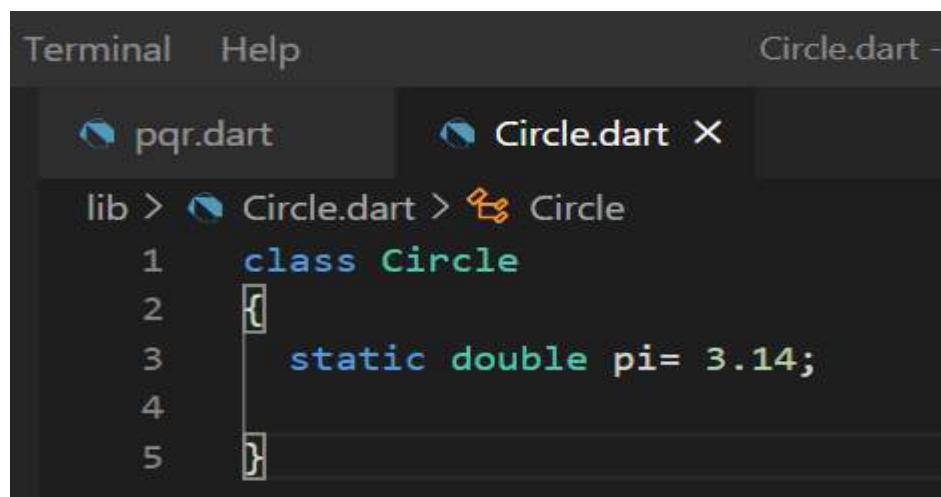
stat.dart Content:

```
1 import 'package:xyz/stat.dart';  
2  
3 main() {  
4     Stat s1 = new Stat();  
5     Stat s2 = new Stat();  
6     Stat s3 = new Stat();  
7     Stat s4 = new Stat();  
8 }
```

Output Console:

```
PROBLEMS 17 OUTPUT DEBUG CONSOLE TERMINAL  
Connecting to VM Service at ws://127.0.0.1:52378/-FeNVI3aPFQ=/w  
there are 1 values  
there are 2 values  
there are 3 values  
there are 4 values
```

Static Members(contd..)

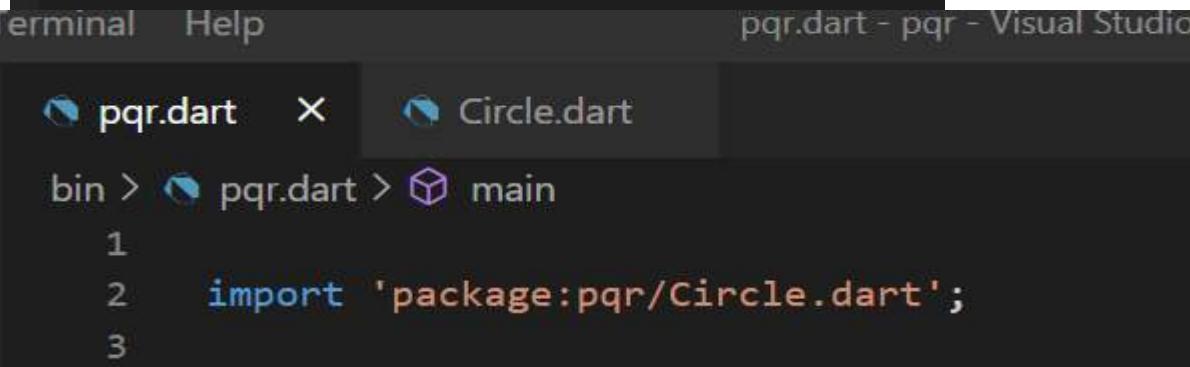


Terminal Help Circle.dart -

pqr.dart Circle.dart X

lib > Circle.dart > Circle

```
1 class Circle
2 {
3     static double pi= 3.14;
4 }
5 }
```



Terminal Help pqr.dart - pqr - Visual Studio Code

pqr.dart X Circle.dart

bin > pqr.dart > main

```
1
2 import 'package:pqr/Circle.dart';
3
4 Run | Debug
5 void main(List<String> arguments) {
6     print(Circle.pi);
7     Circle.pi=6.14;
8     print(Circle.pi);
9 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

Connecting to VM Service at ws://127.0.0.1:59906/r...

3.14

6.14

Exited

Static Members(contd..)

Terminal Help pqr.dart - pqr - Visual Studio Code

pqr.dart X Circle.dart

bin > pqr.dart > main

```
1
2 import 'package:pqr/Circle.dart';
3
4 void main(List<String> arguments) {
5   print(Circle.pi);
6   Circle.pi=6.14;
7   print(Circle.pi);
8 }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

Connecting to VM Service at ws://127.0.0.1:60013/P8_ZohCMpQo=/ws
bin/pqr.dart:6:9: Error: Setter not found: 'pi'.
Circle.pi=6.14;
 ^
Exited (1)

Terminal Help Circle.dart - pqr - Visual Studio Code

pqr.dart X Circle.dart X

lib > Circle.dart > Circle

```
1 class Circle
2 {
3   static const double pi= 3.14;
4 }
5 }
```

Static Members(contd..)

Static Method

- Static method belong to class instead of class instances
- **Static method is allowed to access static variables of class and can invoke only static methods of the class**

Important Points :-

- ▶ Static methods are also known as Class Methods.
- ▶ Static method is only allowed to access the static variables of class.
- ▶ Single copy of static method is shared among all the instances of a class
- ▶ Static methods can be accessed directly using class name.

Static Members(contd..)

Declaring Static Methods

Static method can be declared using static keyword followed by return type, followed by method name.

Syntax:-

```
static return_type method_name ()  
{  
    //Statement(s)  
}
```

Calling Static Method

Static methods can be invoked directly from the class name itself rather than creating an instance of it.

Syntax:-

```
ClassName.staticMethod();
```

Static Members(contd..)

```
stat.dart - xyz - Visual Studio Code
```

```
xyz.dart - xyz - Visual Studio Code
```

```
Help
```

```
xyz.dart  stat.dart
```

```
lib > stat.dart > ...
1  class Stat {
2    static int _counter = 0;
3
4    Stat() {
5      _counter++;
6      print('there are ${_counter} values');
7    }
8
9    static void run() {
10      print('running');
11    }
12 }
```

```
xyz.dart  stat.dart
```

```
bin > xyz.dart > main
1  import 'package:xyz/stat.dart';
2
3  Run | Debug
4  main() {
5    Stat s1 = new Stat();
6    Stat s2 = new Stat();
7    Stat s3 = new Stat();
8    Stat s4 = new Stat();
9    Stat.run();
10 }
```

PROBLEMS 17 OUTPUT DEBUG CONSOLE TERMINAL

```
Connecting to VM Service at ws://127.0.0.1:52404/0othF6XWpYc=/ws
there are 1 values
there are 2 values
there are 3 values
there are 4 values
running
```

Packages

Dart Packages

- Set of dart program organized in an independent, reusable unit
- Contain set of functions and classes for specific purpose or utility along with compiled code and sample data
- Dart has rich set of default packages, loaded automatically when dart console is started
- Any other package then default needs to be installed and loaded explicitly first in order to use it

Dart Package Manager

- Dart has an inbuilt package manager known as pub package manager
- Used to install, organize and manage third party libraries, tools and dependencies
- Every dart application has an **pubspec.yaml** file (track of all third party libraries and application dependencies along with metadata of application like application name, author, version and description)
- Most of dart IDEs have built in support for using pub that includes creating, downloading, updating and publishing packages

Sr.No	Description
pub get	This will get all packages your application is dependent upon.
pub upgrade	This will upgrade all your application dependencies to latest version.
pub build	This is used to create a built of your web application and it will create a build folder, with all related scripts in it.
pub help	This command is used to know about all different pub commands.

Dart Package Manager(contd..)

- Installing a package:
- Add package name in dependency section of your projects pubspec.yaml file.
- Run following command from your application directory to get package installed in your project

>> pub get

Example:-

```
name: DemoApp
version: 0.0.1
description: A simple dart application.
dependencies:
  xml:
```

Here, we have added `xml:` to the project dependencies. This will install Dart XML package in the project.

Dart Package Manager(contd..)

Importing libraries from package:

- Once package is installed in our project, we need to import required libraries from package in our project as:

```
import 'package:xml/xml.dart' as xml;
```

Polymorphism

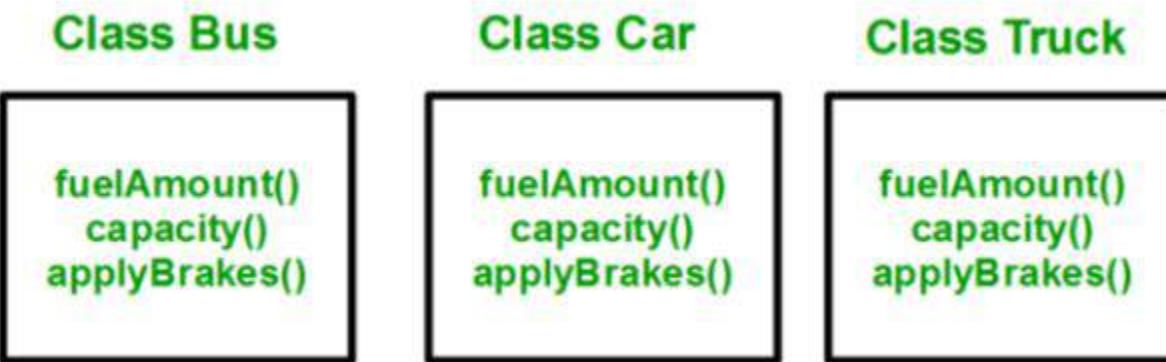
- Derived from 2 greek words “**poly**” and “**morph**” where poly means many and morph means form.
- Polymorphism means an object have different forms and each different form performs **same** action/task in multiple/different ways(speaking different languages)
 - Inheritance
 - Abstraction
 - Interfaces
 - Mixins

Inheritance

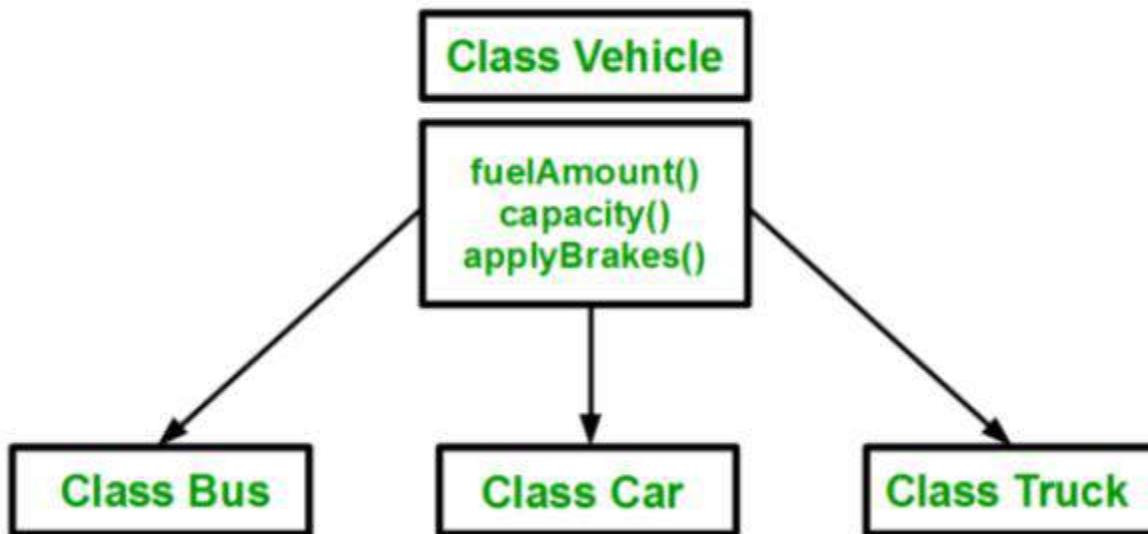
Inheritance

- The capability of a class to derive properties and characteristics from another class is called **Inheritance**
It is ability of a program to create new class from an existing class
- **Parent Class**
The class whose properties are inherited by child class is called Parent Class. Parent class is also known as **base class or super class.**
- **Child Class**
The class that inherits properties from another class is called child class. Child class is also known as **derived class or base class.**

Inheritance (contd..)



Inheritance



Inheritance (contd..)

- **Syntax:**

```
class child_class extends parent_class {  
    //body of child class  
}
```

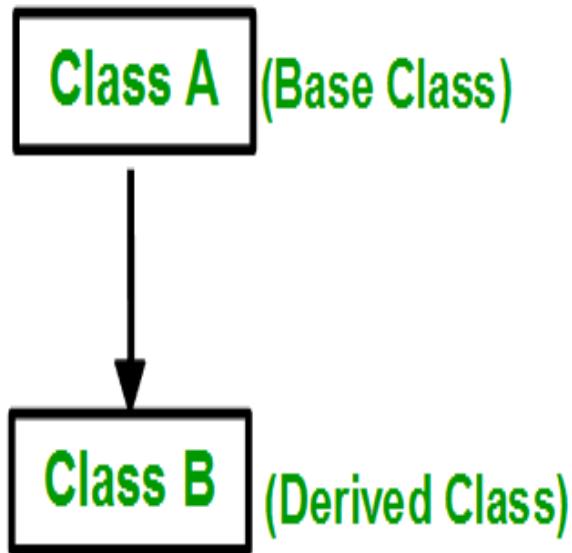
Child class inherits all the properties and methods except constructor from parent class

Types of Inheritance

Mainly there are three types of inheritance:

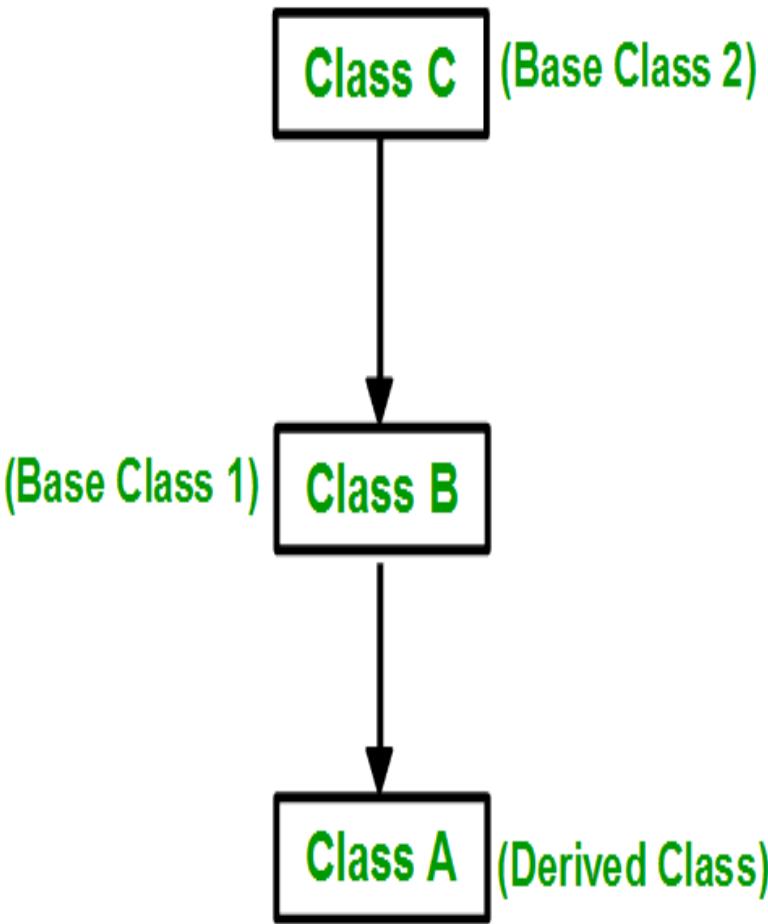
- **Single**
- **Multiple** - Dart doesn't support Multiple Inheritance
- **Multi-level**

Single Level Inheritance



```
class Person{  
    void showName(String name){  
        print(name);  
    }  
  
    void showAge(int age){  
        print(age);  
    }  
}  
  
class Jay extends Person {}  
  
main(){  
    var jay = new Jay();  
  
    jay.showName("JD");  
    jay.showAge(20);  
}  
  
Output  
JD  
20
```

Multi Level Inheritance



```
class Person {  
    void showName(String name) {  
        print(name);  
    }  
  
    void showAge(int age) {  
        print(age);  
    }  
}  
  
class Jay extends Person {  
    void showProfession(String profession) {  
        print(profession);  
    }  
  
    void showNationality(String nationality) {  
        print(nationality);  
    }  
}  
//Derived class created from another derived class.  
class Sanket extends Jay {}  
  
main() {  
    var sanket = new Sanket();  
  
    sanket.showName("Sanket");  
    sanket.showAge(20);  
    sanket.showNationality("Indian");  
    sanket.showProfession("Engineer");  
}
```

Output

Sanket
20
Indian
Engineer

Inheritance(contd..)

animalinherit.dart

```
lib > animalinherit.dart > ...
1  class Animalinherit {
2    bool isAlive = true;
3    void breathe() {
4      print("breathing");
5    }
6  }
7
```

feline.dart

```
lib > feline.dart > ...
1  import 'mammal.dart';
2
3  class Feline extends Mammal {
4    bool hasClaws = true;
5
6    void grow() {
7      print('growing');
8    }
9
10   @override
11   void test() {
12     print('testing feline class');
13     super.test();
14   }
15 }
```

Help

mammal.dart

mammal.dart

```
lib > mammal.dart > ...
1  import 'animalinherit.dart';
2
3  class Mammal extends Animalinherit {
4    bool hasHair = true;
5    bool hasBackbone = true;
6    bool isWarmBlooded = true;
7
8    void walking() {
9      print('walking');
10   }
11
12   void test() {
13     print('testing mammal class');
14   }
15 }
```

Inheritance

The screenshot shows the Visual Studio Code interface with the xyz.dart file open. The code imports 'package:xyz/feline.dart' and defines a main() function that creates a Feline object and calls its breathe() method. A tooltip for 'cat.breathe()' lists the method's signature and other available methods like grow(), hasBackbone(), and hashCode().

```
bin > xyz.dart > lib/main.dart
1 import 'package:xyz/feline.dart';
2
3 Run | Debug
4 main() {
5   Feline cat = new Feline();
6   cat.breathe();
7 }
```

xyz.dart

```
1 import 'package:xyz/feline.dart';
2
3 Run | Debug
4 main() {
5   Feline cat = new Feline();
6   cat.breathe();
7 }
```

The screenshot shows the Visual Studio Code interface with the xyz.dart file open. The code imports 'package:xyz/feline.dart' and defines a main() function that creates a Feline object and calls its breathe() method. The terminal at the bottom shows the output of the run command, which includes the word 'breathing' and 'Exited'.

```
xyz.dart - xyz - Visual Studio Code
```

```
File Edit Selection View Go Run Terminal Help xyz.dart - xyz - Visual Studio Code xyz.dart X animalinherit.dart mammal.dart xyz.dart ... bin > xyz.dart > ...
1 import 'package:xyz/feline.dart';
2
3 Run | Debug
4 main() {
5   Feline cat = new Feline();
6   cat.breathe();
7 }
```

xyz.dart

```
xyz.dart - xyz - Visual Studio Code xyz.dart X animalinherit.dart mammal.dart xyz.dart ... bin > xyz.dart > ...
1 import 'package:xyz/feline.dart';
2
3 Run | Debug
4 main() {
5   Feline cat = new Feline();
6   cat.breathe();
7 }
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL

```
Connecting to VM Service at ws://127.0.0.1:57408/_rl0ie-lm6g=/ws
breathing
Exited
```

Super keyword

- **super** keyword is a reference variable which is used to refer **immediate parent class object**
- Using super keyword, we **can access super class properties and methods**
- Instances of child class can refer to properties and methods of super class using super keyword
- **Use:** to eliminate ambiguity between super class and sub class that have variables and methods with same name

Super keyword(contd..)

Use super keyword to access parent class variables

When you have a variable in sub class which is already exists in its super class; then the super keyword can be used to access variable in super class.

Syntax:-

```
super.varName
```

Use super keyword to invoke parent class method

When a subclass contains a method already present in the superclass then it is called method overriding. In method overriding call to the method from subclass object will always invoke the subclass version of the method. However, using the super keyword we are allowed to invoke superclass version of the method.

Syntax:-

```
super.methodName
```

Method overriding

- Declaring a method in sub class which already exist in parent class with same name, same arguments, same return type is termed as method overriding

Sub class provides its own implementation for the method which already exist in super class

- Method defined in super class is called **overridden method** and the method in subclass is called as **overriding method**

Method overriding(contd..)

Rules for Method Overriding

- 1.) A method can only be written in Subclass, not in same class.
- 2.) The argument list should be exactly the same as that of the overridden method.
- 3.) The return type should be the same as declared in the original overridden method in the super class.
- 4.) A method declared final cannot be overridden.
- 5.) A method declared static cannot be overridden.
- 6.) If a method cannot be inherited then it cannot be overridden.
- 7.) Constructors cannot be overridden.

Inheritance(contd..)

animalinherit.dart

```
lib > animalinherit.dart > ...
1  class Animalinherit {
2    bool isAlive = true;
3    void breathe() {
4      print("breathing");
5    }
6  }
7
```

feline.dart

```
lib > feline.dart > ...
1  import 'mammal.dart';
2
3  class Feline extends Mammal {
4    bool hasClaws = true;
5
6    void grow() {
7      print('growing');
8    }
9
10   @override
11   void test() {
12     print('testing feline class');
13     super.test();
14   }
15 }
```

Help

mammal.dart

```
mammal.dart
lib > mammal.dart > ...
1  import 'animalinherit.dart';
2
3  class Mammal extends Animalinherit {
4    bool hasHair = true;
5    bool hasBackbone = true;
6    bool isWarmBlooded = true;
7
8    void walking() {
9      print('walking');
10   }
11
12   void test() {
13     print('testing mammal class');
14   }
15 }
```

Inheritance(contd..){Method overriding}

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files xyz.dart, animalinherit.dart, mammal.dart, and feline.dart.
- Code Editor:** The xyz.dart file contains the following code:

```
bin > xyz.dart > main
1 import 'package:xyz/feline.dart';
2
3 void main() {
4     Feline cat = new Feline();
5     cat.breathe();
6     cat.test();
7 }
8
```
- Terminal:** The terminal output shows the execution of the program:

```
Connecting to VM Service at ws://127.0.0.1:57683/mrcnOqS-_aM=/ws
breathing
testing feline class
testing mammal class
Exited
```

Abstraction

Abstraction

- Real life example: ATM Machine
- Any class that contains one or more abstract methods is known as abstract class which declared abstract using “abstract” keyword followed by class declaration
- A class declared as abstract may or may not include abstract methods
- **An abstract class cannot be instantiated**
- Abstract class only be extended where subclass needs to provide implementation for all abstract methods from super class

If subclass doesn't implement abstract methods
then it should also be declared as abstract

Abstraction(contd..)

Rules for Abstract Classes

- 1.) Abstract classes may or may not include abstract methods (methods without body).
- 2.) If a class has at least one abstract method, then the class must be declared abstract.
- 3.) An abstract class can not be instantiated, but can be extended .
- 4.) If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.
- 5.) An abstract class can be declared using **abstract** keyword.
- 6.) An abstract class may also have concrete (methods with body) methods.

Abstraction(contd..)

Declaring Abstract Class

An abstract class can be defined using **abstract** keyword followed by class declaration. An abstract class is mostly used to provide a base for subclasses to extend and implement the abstract methods.

Syntax:-

```
abstract class ClassName {  
    // Body of abstract class  
}
```

Abstraction(contd..)

```
car.dart X  
lib > car.dart > ...  
1 abstract class Car {  
2     bool hasWheels;  
3     bool hasHorn;  
4  
5     void honk();  
6 }
```

```
xyz.dart X  
bin > xyz.dart > ...  
1 import 'package:xyz/racecar.dart';  
2  
3 Run | Debug  
4  
5 main() {  
6     RaceCar r = new RaceCar();  
7     r.honk();  
8 }
```

```
racecar.dart X  
lib > racecar.dart > ...  
1 import 'car.dart';  
2  
3 class RaceCar extends Car {  
4     RaceCar() {  
5         this.hasHorn = true;  
6         this.hasWheels = true;  
7     }  
8     void honk() {  
9         print('beep beep');  
10    }  
11 }
```

PROBLEMS 15 OUTPUT DEBUG CONSOLE TERMINAL

```
Connecting to VM Service at ws://127.0.0.1:58301/61G9zJnpfdg=/ws  
beep beep  
Exited
```

Interfaces

Interfaces

- Blueprint of a class, that any class entity must adhere to
- Declares set of methods available on an object
- Contain only method signature, it does not provide implementation details of method
- Class that uses interface needs to use ***implements keyword*** to use interface method

Class must provide complete implementation details for all methods that belong to interface

Usage of Interfaces

In Dart, interfaces is a way to achieve full abstraction. Since interface methods do not have body, the class that implements interface must implement all the methods of that interface before it can be accessed.

- 1.) An Interface is used to achieve abstraction.
- 2.) Interface is a mechanism to achieve multiple inheritance in Dart.

Interface(contd..)

Declaring an Interface

In Dart, there is no way does for declaring interfaces directly. In Dart, class declarations themselves implicitly defines an interface containing all the instance members of the class and of any interfaces it implements. .

Implementing an Interface

In order to use interface methods, the interface must be implemented by another class. Class should use the **implements** keyword (instead of extends) to be able to use an interface method. A class implementing interface must provide a concrete implementation of all the methods belongs to the interface. In other words, a class must redefine every method of the interface it is implementing.

Syntax:-

```
class ClassName implements InterfaceName
```

Interface(contd..)

```
employee.dart X
lib > employee.dart > Employee > test
1   class Employee {
2     String name = '';
3     void test() {
4       print('test');
5     }
6   }
7
```

```
xyz.dart X
bin > xyz.dart > main
1 import 'package:xyz/manager.dart';
2
3 Run | Debug
4 main() {
5   Manager bob = new Manager();
6   bob.test();
7 }
```

```
manager.dart - xyz - Visual Studio Code
manager.dart X
lib > manager.dart > ...
1 import 'employee.dart';
2
3 class Manager implements Employee {
4   String name = 'Bob';
5   void test() {
6     print('I am a Manager');
7   }
8 }
```

Help

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL

Connecting to VM Service at ws://127.0.0.1:58122/3ou-HmOPeVc-/ws

I am a Manager

Exited

Interface(contd..)

```
employee.dart X  
lib > employee.dart > Employee > test  
1   class Employee {  
2     String name = '';  
3     void test() {  
4       print('test');}  
5     }  
6   }  
7
```

```
xyz.dart X  
bin > xyz.dart > main  
1   import 'package:xyz/manager.dart';  
2  
3   Run | Debug  
4   main() {  
5     Manager bob = new Manager();  
6     bob.test();  
7 }
```

```
Help manager.dart - xyz - Visual Studio Code  
manager.dart X  
lib > manager.dart > ...  
1   import 'employee.dart';  
2  
3   class Manager implements Employee {  
4     String name = 'Bob';  
5     void test() {  
6       print('test in manager class');  
7       print(super.toString());  
8     }  
9   }
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL
Connecting to VM Service at ws://127.0.0.1:58187/vkmaQWoBMT0=/ws
test in manager class
Instance of 'Manager'
Exited

Interface(contd..)

Implementing Multiple Interface

As we know the Dart does not support multiple inheritance, but a class can implement multiple interfaces. This way Interface can be used as a mechanism to achieve multiple inheritance in Dart.

Syntax:-

```
class ClassName implements interface1, interface2, interfa
```

Rules For Implementing An Interfaces

- 1.) Any class implementing an Interface must override every method and instance variable of an interface.
- 2.) In Dart, there is no syntax for declaring an interfaces, class declaration itself implicitly defines an interface.
- 3.) A class implementing interface must provide a concrete implementation of all the methods belongs to the interface.
- 4.) A class can implement more than one interface simultaneously.
- 5.) A class can extend only one class, but can implement multiple interfaces.

Interface(contd..)

The screenshot shows a Dart code editor interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar shows 'person.dart' and 'docs'. The left sidebar has sections for EXPLORER, OPEN EDITORS, ABCS, bin, abcs.dart, lib, abcs.dart, employee.dart, Myclass.dart, person.dart (which is selected), profession.dart, test, .gitignore, and .packages. The main editor area displays the following code:

```
lib > person.dart > Person
1 class Person {
2   String name;
3   int age;
4
5   void ShowName() {
6     print("My name is $name");
7   }
8
9   void ShowAge() {
10    print("My Age is $age");
11 }
```

The screenshot shows a Dart code editor interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar shows 'person.dart', 'profession.dart X', and 'employee.dart'. The left sidebar has sections for EXPLORER, OPEN EDITORS, ABCS, bin, abcs.dart, lib, abcs.dart, employee.dart, Myclass.dart, person.dart, profession.dart (which is selected), test, and .gitignore. The main editor area displays the following code:

```
lib > profession.dart > Profession
1 class Profession {
2   String prof;
3   int salary;
4
5   void ShowProfession() {
6     print("My Profession Is: $prof");
7   }
8
9   void ShowSalary() {
10    print("My Salary Is: $salary");
11 }
```

Interface(contd..)

The screenshot shows a Dart code editor interface with the following details:

- EXPLORER** sidebar:
 - > OPEN EDITORS
 - > ABCS
 - > .dart_tool
 - ✓ bin
 - abc.dart
 - ✓ lib
 - abc.dart
 - employee.dart
 - Myclass.dart
 - person.dart
 - profession.dart
 - > test
 - ▷ .gitignore
 - Ξ .packages
 - ! analysis_options.yaml
 - ⌚ CHANGELOG.md
 - Ξ pubspec.lock
 - ! pubspec.yaml
 - ⓘ README.md
- OPEN EDITORS** tab bar: person.dart X, profession.dart, employee.dart X
- employee.dart** file content:

```
lib > employee.dart > Employee > ShowAge
1 import 'person.dart';
2 import 'profession.dart';
3
4 class Employee implements Person, Profession { // Line 4
5   @override
6   String name;
7   @override
8   int age;
9
10  @override
11  void ShowName() {
12    print("Employee Name Is: $name");
13  }
14
15  @override
16  void ShowAge() {
17    print("Employee Age Is: $age");
18  }
19
20  @override
21  String prof;
22  @override
23  int salary;
```
- Output** panel:

```
24
25   @override
26   void ShowProfession() {
27     print("Employee Profession Is: $prof");
28   }
29
30   @override
31   void ShowSalary() {
32     print("Employee Salary Is: $salary");
33   }
34 }
```

Interface(contd..)

The screenshot shows a Dart development environment with several tabs open at the top: person.dart, profession.dart, employee.dart, and abcs.dart (which is currently selected). Below the tabs, the code for abcs.dart is displayed:

```
bin > abcs.dart > main
1 import 'package:abcs/employee.dart';
Run | Debug
2 main() {
3     Employee emp = new Employee();
4     emp.name = "Keith";
5     emp.age = 30;
6     emp.prof = "System Analyst";
7     emp.salary = 25000;
8     print("W3Adda - Dart implementing Multiple Interfaces Example");
9     emp.ShowName();
10    emp.ShowAge();
11    emp.ShowProfession();
12    emp.ShowSalary();
13 }
```

At the bottom of the screen, there is a terminal window showing the output of the program:

```
PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL
Connecting to VM Service at ws://127.0.0.1:53080/RXtHs4krhpA=/ws
W3Adda - Dart implementing Multiple Interfaces Example
Employee Name Is: Keith
Employee Age Is: 30
Employee Profession Is: System Analyst
Employee Salary Is: 25000
Exited
```

MODULE 2:

CREATING USER INTERFACE WITH

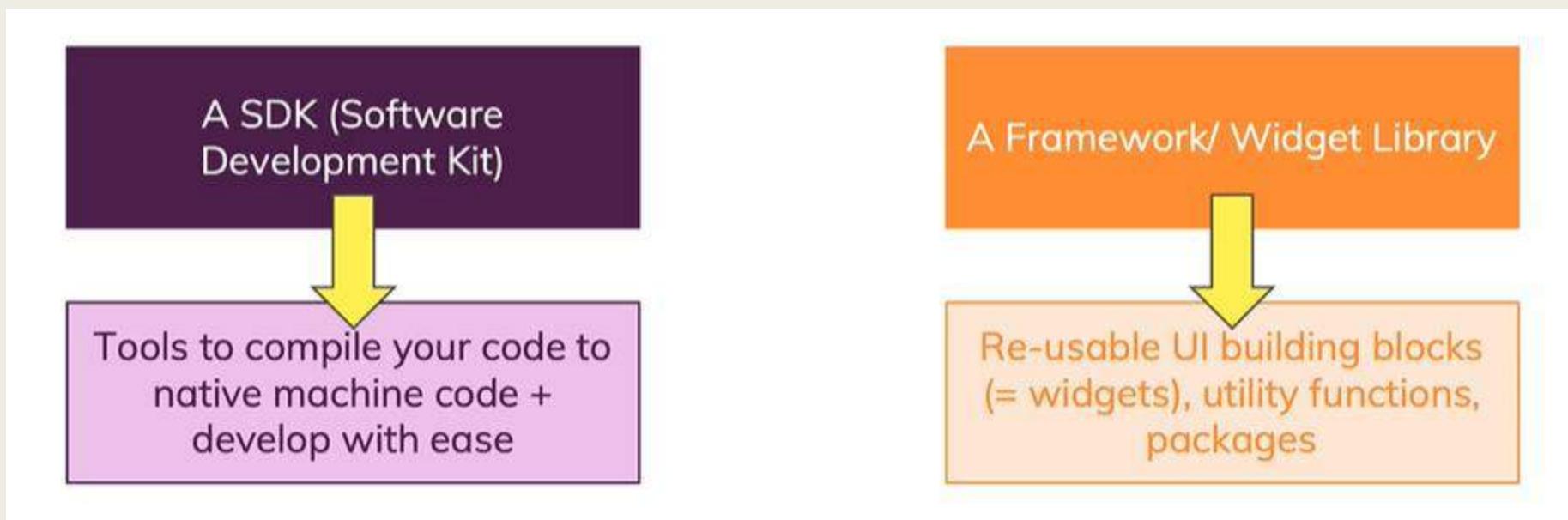
FLUTTER

Outline

- Exploring default project structure
- Flutter Basic Layouts
- Widgets in flutter
- Adding custom images, fonts

What is Flutter?

- A “tool” that allows you to build **native cross-platform** (iOS, Android) apps with **one programming language** and codebase



Dart??

Programming language developed by **Google**

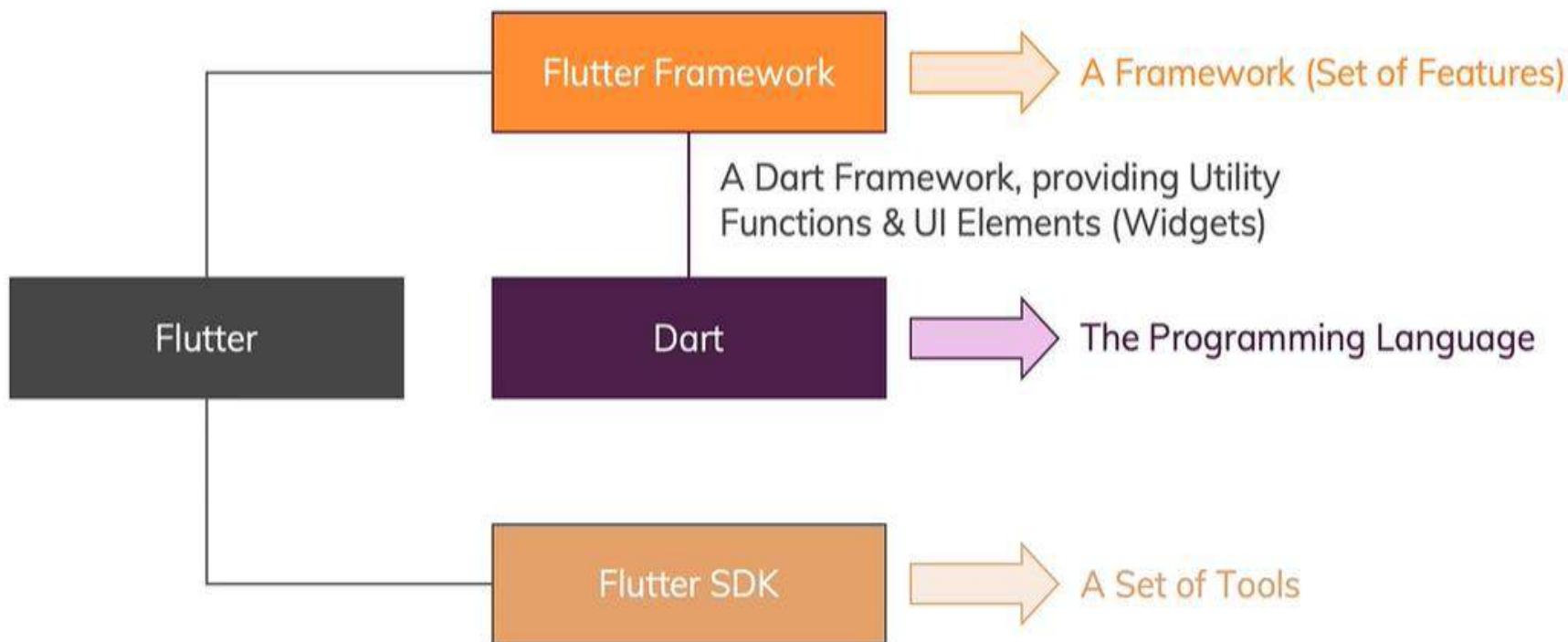
Object-oriented &
Strongly Typed



Syntax is like a mixture of
JavaScript, Java, C# (you don't
need to know these languages!)

Focused on **frontend** (mobile apps, web) **user interface** (UI)
development

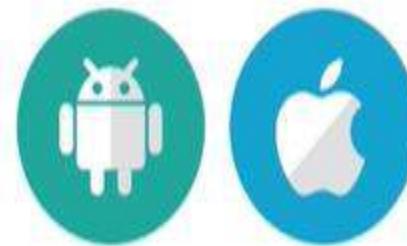
Flutter vs Dart



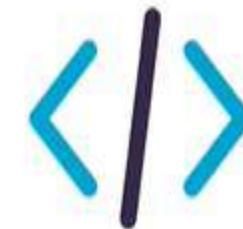
Flutter Architecture



UI as Code: Build a
Widget Tree



Embrace Platform Differences



One Codebase

Flutter Architecture(contd...)

UI as “code”

No Drag & Drop

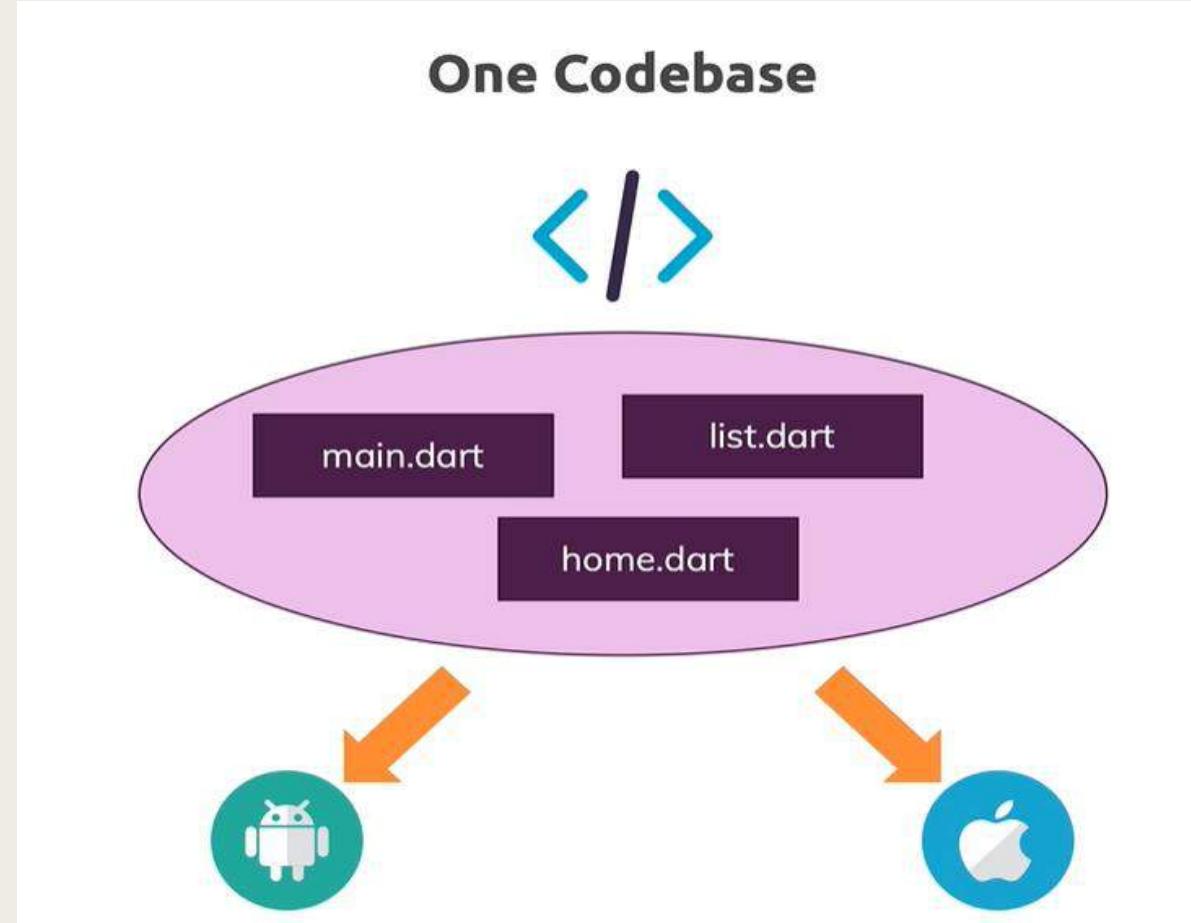
No Visual Editor

Code only

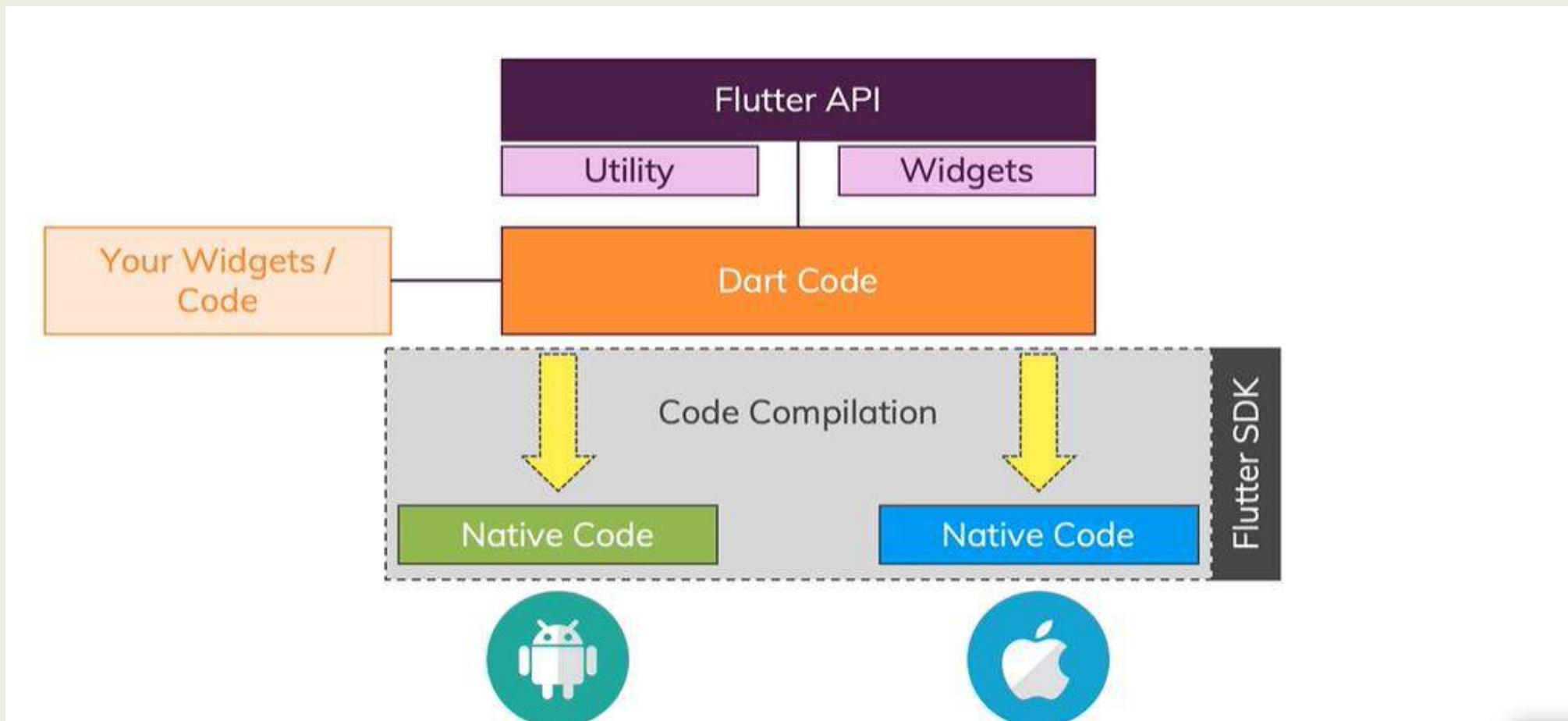
But extremely straightforward

```
body: Stack  
children: <Widget>[  
 Container(  
 decoration: BoxDecoration(  
 image: DecorationImage(  
 image: AssetImage('assets/images/store.jpg'),  
 fit: BoxFit.cover,  
 alignment: Alignment.center,  
 ), // DecorationImage  
, // BoxDecoration  
>, // Container  
 Container(  
 // width: double.infinity,  
 // height: double.infinity,  
 decoration: BoxDecoration(  
 gradient: LinearGradient(  
 colors: [  
 Color.fromRGBO(215, 117, 255, 1).withOpacity(0.5),  
 Color.fromRGBO(255, 188, 117, 1).withOpacity(0.9),  
 ],  
 begin: Alignment.topLeft,  
 end: Alignment.bottomRight,  
 stops: [0, 1],  
 ), // LinearGradient  
, // BoxDecoration  
>, // Container  
 SingleChildScrollView(<br>
```

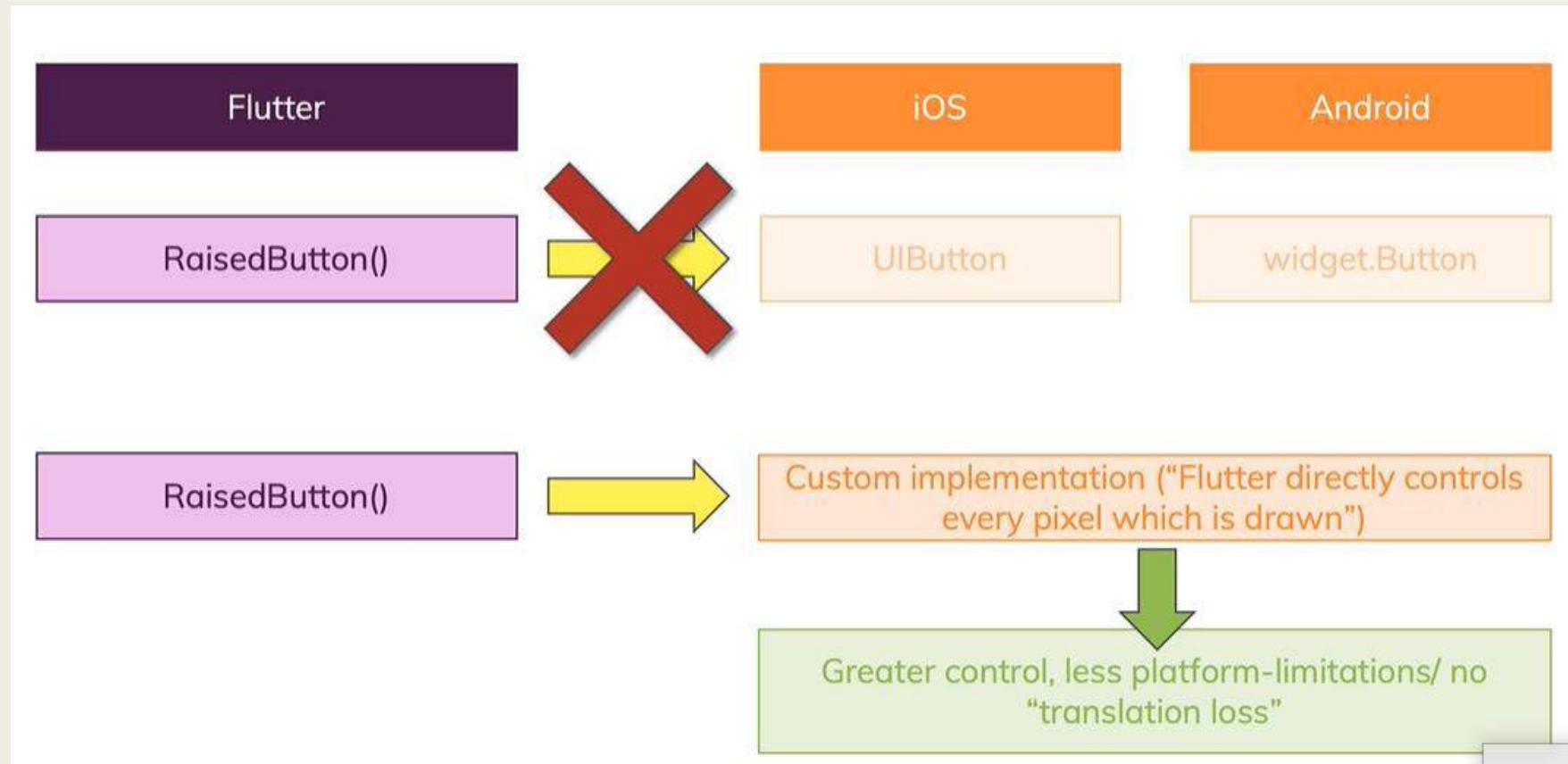
Flutter Architecture(contd...)



Flutter and Dart transformed to Native App



Flutter does NOT use Platform Primitives



Flutter Versions

Flutter Versions change frequently



This does NOT mean, that Flutter changes all
the time



Bugfixes, small improvements, “behind-the-
scenes” changes, new (niche) features, ...

Project Structure

The screenshot shows a code editor interface with the following details:

- EXPLORER** panel on the left, showing the project structure:
 - FIRST_APP (expanded): .dart_tool, .idea, android, build, ios, lib (expanded): main.dart, test, .gitignore, .metadata, .packages, first_app.iml, pubspec.lock, pubspec.yaml, README.md.
- main.dart** editor tab at the top center.
- Code Editor Content:**

```
lib > main.dart > main
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   // This widget is the root of your application.
9   @override
10  Widget build(BuildContext context) {
11    return MaterialApp(
12      title: 'Flutter Demo',
13      theme: ThemeData(
14        // This is the theme of your application.
15        //
16        // Try running your application with "flutter run". You'll see
17        // the application has a blue toolbar. Then,
```
- Toolbar icons:** Run, Stop, Refresh, Save, Undo, Redo, Copy, Paste, Find, Select All, etc.

Shortcuts in VSCode

5.2. Basic shortcuts

The following tables lists the most important shortcuts for using Visual Studio Code for Flutter development.

Table 1. Must known shortcuts

Shortcut	Description
<code>Ctrl + .</code>	Quick fix for solving issues
<code>Ctrl + Space</code>	Content assist/ code completion
<code>Ctrl + Shift + p</code>	Execute command
<code>F5</code>	Start Flutter application

5.3. Editing shortcuts

Shortcut	Description
<code>Ctrl + Shift + i</code>	Format source
<code>Ctrl + x</code>	Cut line
<code>Ctrl + k + c</code>	Comment selected code
<code>Ctrl + k + u</code>	Uncomment selected code
<code>Ctrl + Shift + k</code>	Delete line

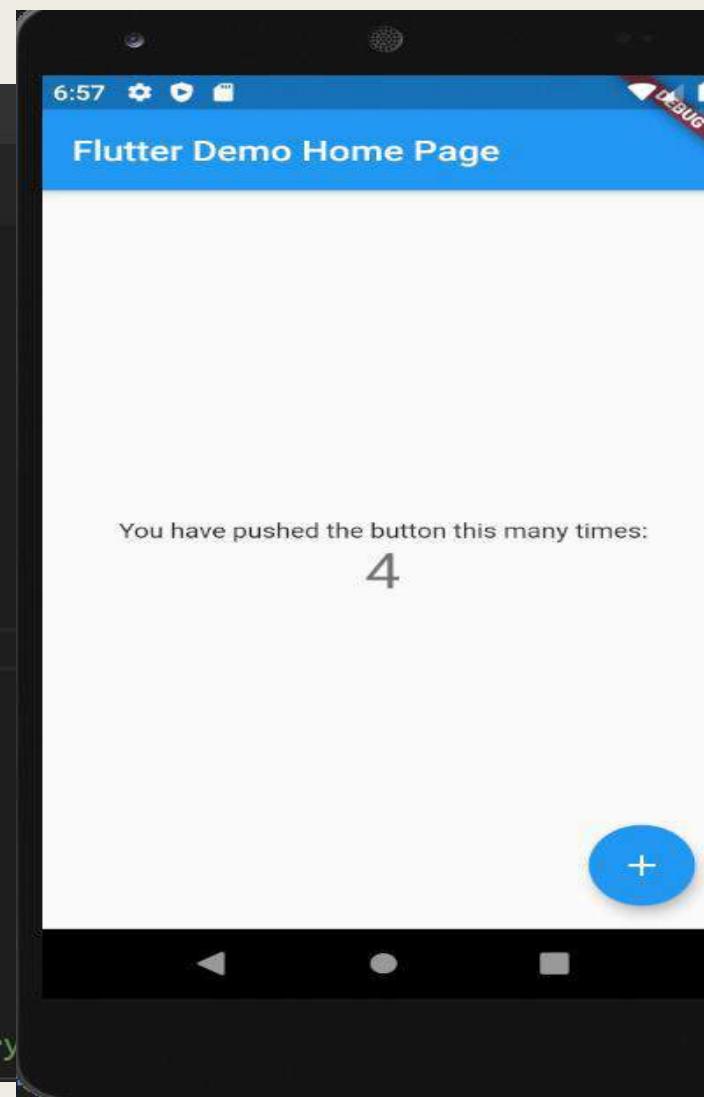
First Application

terminal Help main.dart - first_app - Visual Studio Code

main.dart X Dart

lib > main.dart > MyApp

```
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   // This widget is the root of your application.
9   @override
10  Widget build(BuildContext context) {
11    return MaterialApp(
12      title: 'Flutter Demo',
13      theme: ThemeData(
14        // This is the theme of your application.
15        //
16        // Try running your application with "flutter run". You'll see the
17        // application has a blue toolbar. Then, without quitting the app, try
```



First Application

```
Run | Debug
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
```

```
class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  // This widget is the home page of your application. It is stateful, meaning
  // that it has a State object (defined below) that contains fields that affect
  // how it looks.

  // This class is the configuration for the state. It holds the values (in this
  // case the title) provided by the parent (in this case the App widget) and
  // used by the build method of the State. Fields in a Widget subclass are
  // always marked "final".

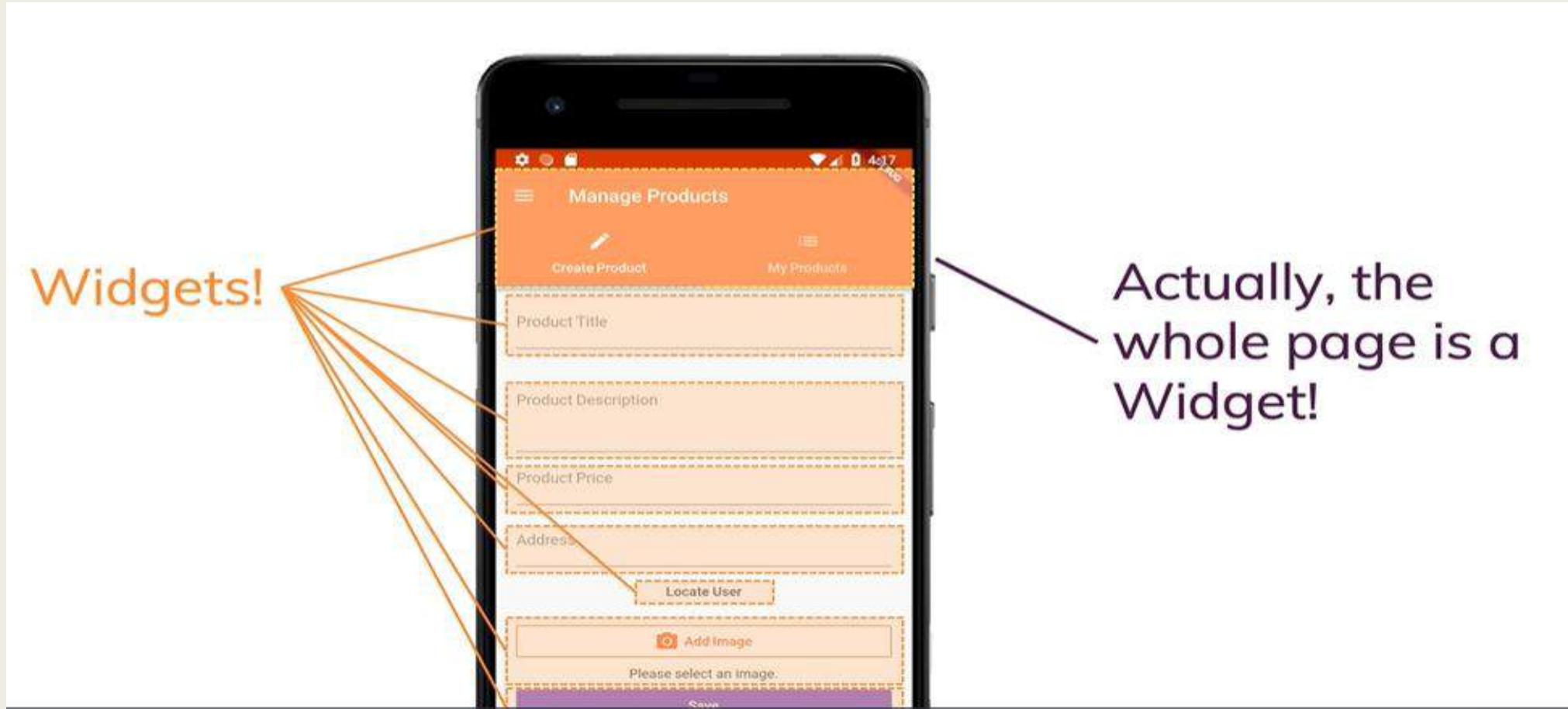
  final String title;
```

Widgets

- A widget is either stateful or stateless
- **A stateless widget** never changes
 - *Icon, IconButton, and Text are examples of stateless widgets*
 - *Stateless widgets subclass StatelessWidget*
- **A stateful widget** is dynamic: for example, it can change its appearance in response to events triggered by user interactions or when it receives data
 - *Checkbox, Radio, Slider, InkWell, Form, and TextField are examples of stateful widgets*
 - *Stateful widgets subclass StatefulWidget*
- Example: flutter basic app(counter)

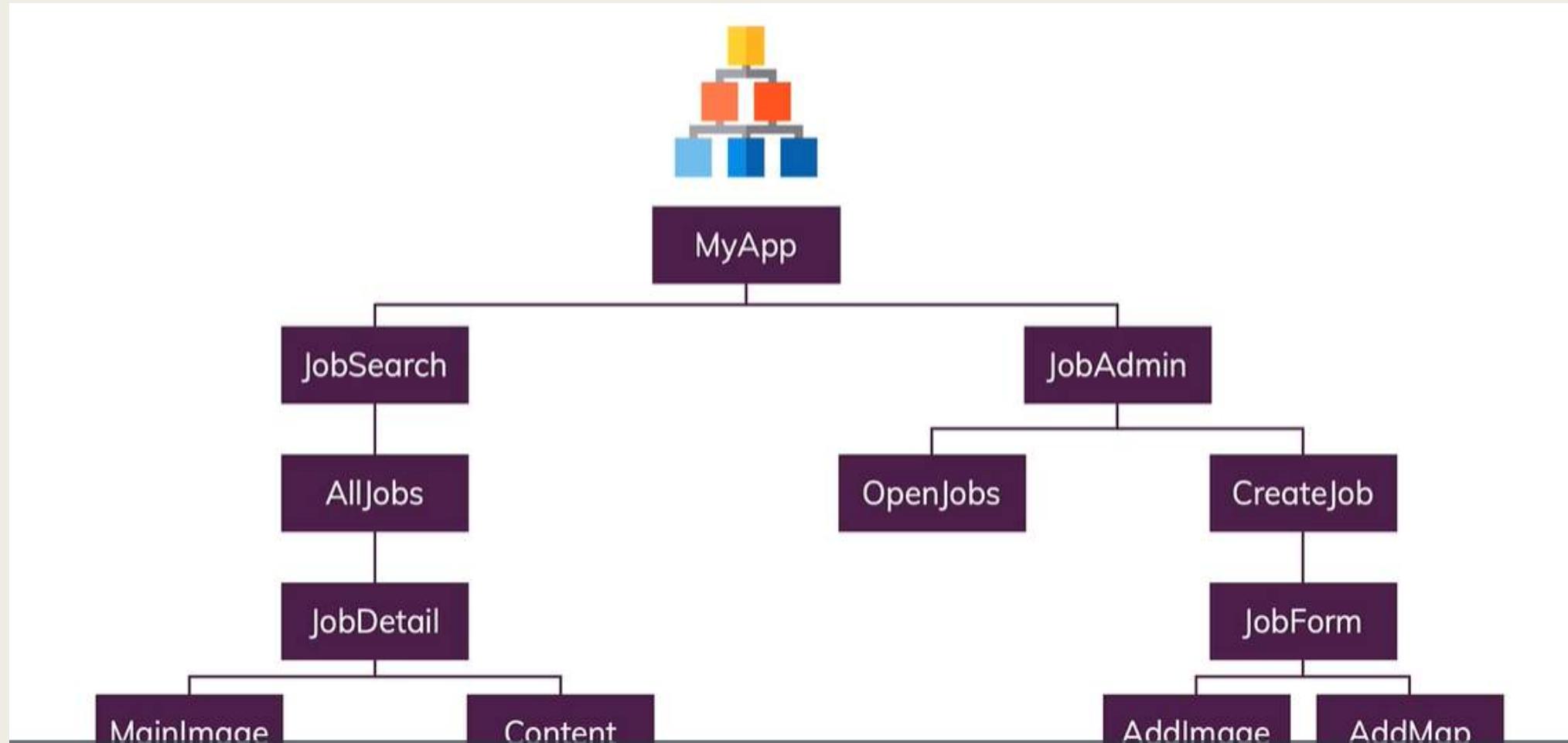
Flutter Architecture

Everything's is a Widget



Flutter Architecture(contd...)

Your App's UI is a widget tree



MaterialApp Widget

- MaterialApp widget is the means to follow the Material Design guidelines in Flutter(insert it as root of the widget tree to get started)
- Properties:
 - defines the theme (the ‘look and feel’) of app
 - MaterialApp object is used to specify the list of ‘routes’ to all the separate screens that will make up your app along with which will be the first screen
 - Using MaterialApp object where you give a title to your app
 - MaterialApp object internally first instantiate [MediaQuery](#) widget which handles particular model of phone on how the app will appear and function

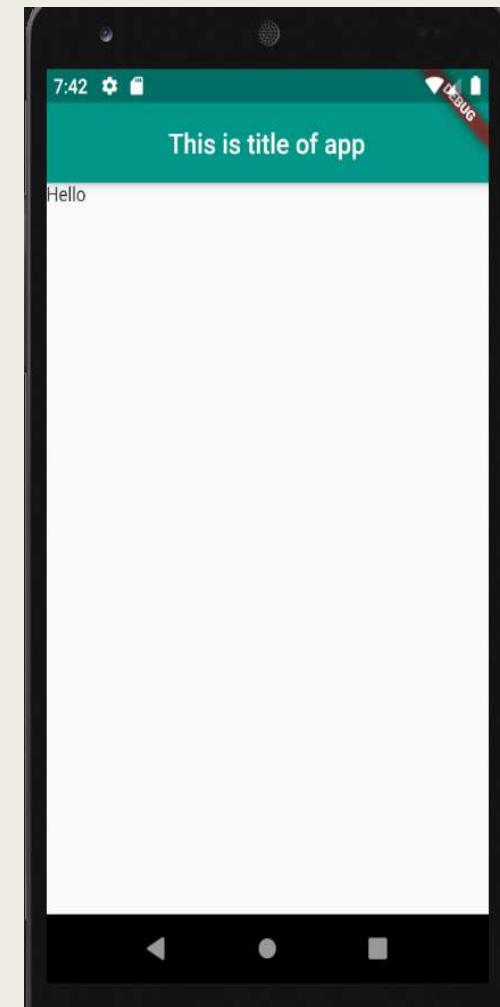
Scaffold

- Scaffold widget implements the basic Material Design visual layout, allowing you to easily add various widgets such as
 - *AppBar*
 - *BottomAppBar*
 - *FloatingActionButton*
 - *Drawer*
 - *SnackBar*
 - *BottomSheet*

Scaffold

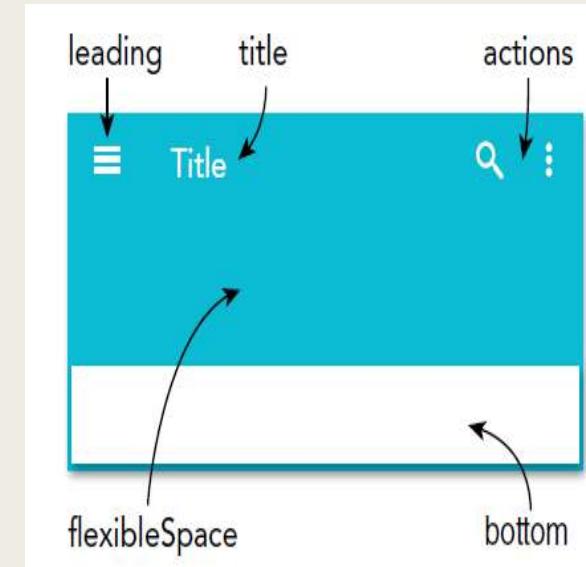
terminal Help • main.dart - scaffold_widget - Visual Studio Code

```
lib > main.dart > MyAppState > build
  1 import 'package:flutter/material.dart';
  2
  3 Run | Debug
  4 void main() {
  5   runApp(MyApp());
  6 }
  7 class MyApp extends StatefulWidget {
  8   @override
  9   MyAppState createState() => new MyAppState();
 10 }
 11
 12 class MyAppState extends State<MyApp> {
 13   @override
 14   Widget build(BuildContext context) {
 15     return MaterialApp(
 16       home: Scaffold(
 17         appBar: AppBar(
 18           title: Text('This is title of app'),
 19           backgroundColor: Colors.teal,
 20           centerTitle: true,
 21         ), // AppBar
 22         body: Text('Hello'),
 23       ), // Scaffold
 24     ); // MaterialApp
 25   }
 26 }
```



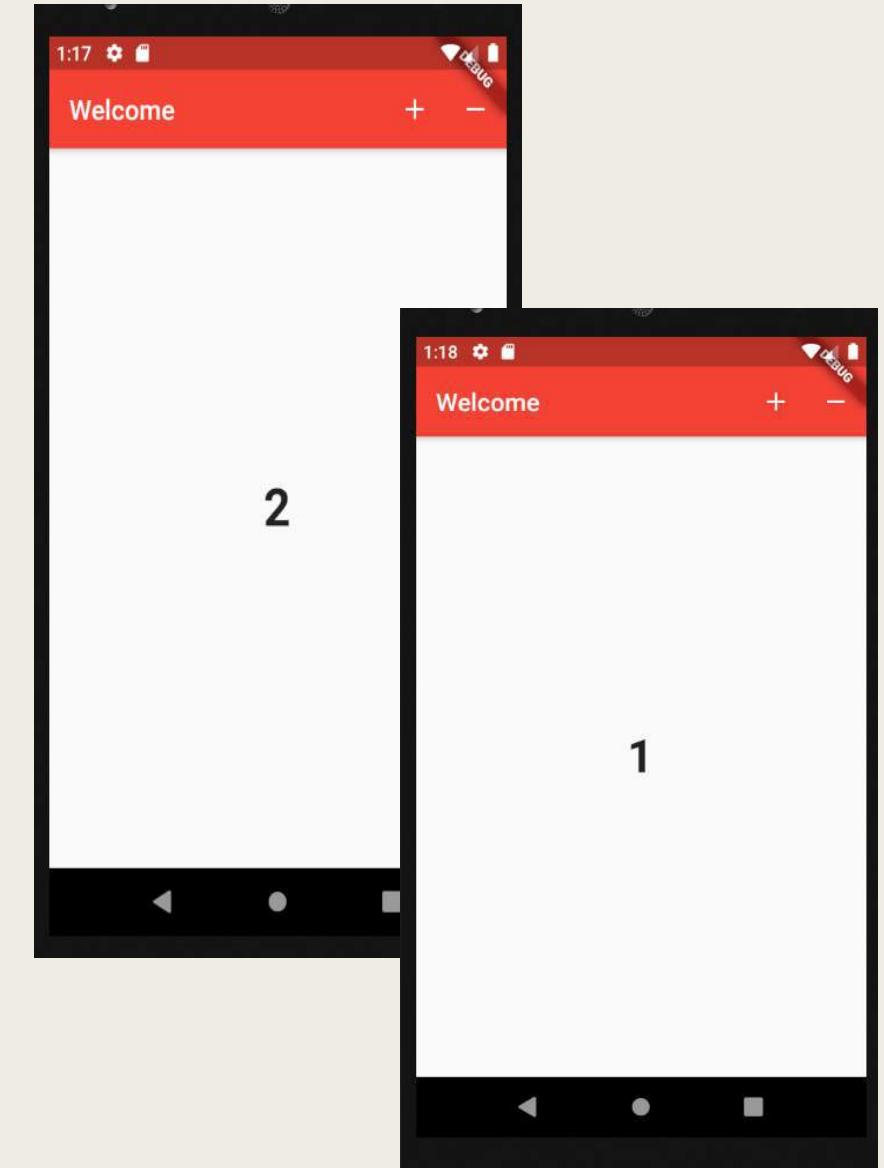
AppBar

- AppBar widget include following properties as well as many customization options
- **title** The title property is typically implemented with a Text widget. You can customize it with other widgets such as a DropdownButton widget.
- **leading** The leading property is displayed before the title property. Usually this is an IconButton or BackButton.
- **actions** The actions property is displayed to the right of the title property. It's a list of widgets aligned to the upper right of an AppBar widget usually with an IconButton or PopupMenuButton.
- **flexibleSpace** The flexibleSpace property is stacked behind the Toolbar or TabBar widget with height same as the AppBar widget's. A background image is commonly applied to the flexibleSpace property, but any widget, such as an Icon, could be used



AppBar

```
15   class _State extends State<MyApp>
16 {
17
18     int _value = 0;
19     void _add()=>setState(()=>_value++);
20     void _remove()=>setState(()=>_value--);
21
22   @override
23   Widget build(BuildContext context){
24
25     return new Scaffold(
26       appBar:new AppBar(
27         title: new Text('Welcome'),
28         backgroundColor: Colors.red,
29         actions:<Widget>[
30           new IconButton(icon: new Icon(Icons.add), onPressed: _add),
31           new IconButton(icon: new Icon(Icons.remove), onPressed: _remove),
32           ] // <Widget>[]
33     ), // AppBar
34     body: new Container(
35       padding: new EdgeInsets.all(32.0),
36       child: new Center(
37         child: new Text(_value.toString(),
38         style: new TextStyle(
39           fontWeight:FontWeight.bold,
40           fontSize:37.0)), // TextStyle // Text
41       ), // Center
42     ), // Container
43   ); // Scaffold
44 }
```



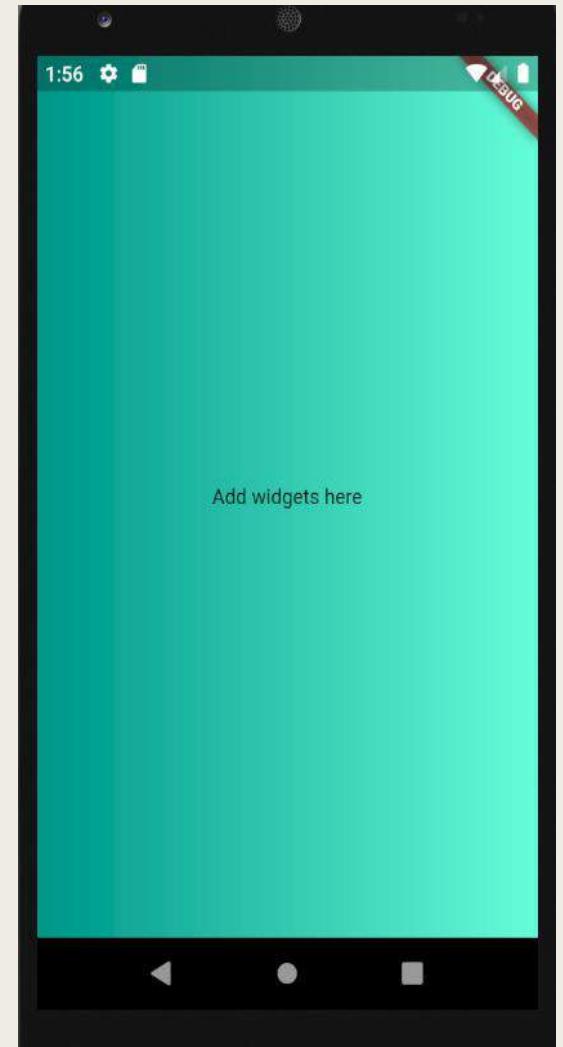
Container Widget

- The Container widget has an optional child widget property and can be used as a decorated widget with a custom border, color, constraint, alignment, transform (such as rotating the widget), and more

```
1 Container({  
2   Key key,  
3   this.alignment,  
4   this.padding,  
5   Color color,  
6   Decoration decoration,  
7   this.foregroundDecoration,  
8   double width,  
9   double height,  
10  BoxConstraints constraints,  
11  this.margin,  
12  this.transform,  
13  this.child,  
14})
```

Container

```
main.dart -> _State -> build
import 'package:flutter/material.dart';
Run | Debug
void main() {
  runApp(new MaterialApp(
    home: new MyApp(),
  )); // MaterialApp
}
class MyApp extends StatefulWidget {
  @override
  _State createState()=>new _State();
}
class _State extends State<MyApp>
{
@override
Widget build(BuildContext context){
  double height=MediaQuery.of(context).size.height;
  double width=MediaQuery.of(context).size.width;
  return Scaffold(
    body: Container(
      alignment:AlignmentDirectional.center,
      child: new Text('Add widgets here'),
      decoration:BoxDecoration(
        gradient:LinearGradient([colors: [Colors.teal,Colors.tealAccent]]),
      ), // BoxDecoration
    ), // Container
  ); // Scaffold
}
```

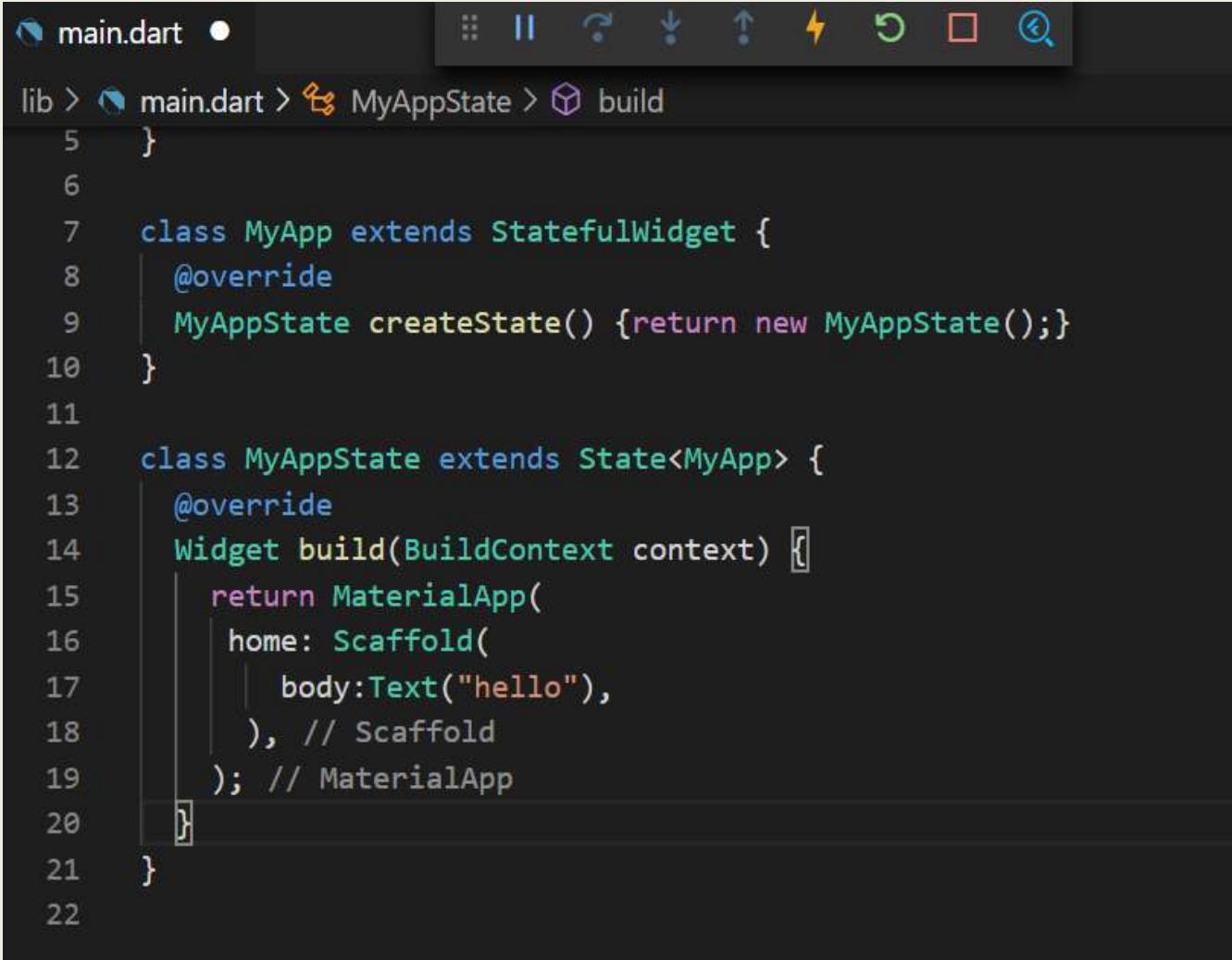


Text Widget

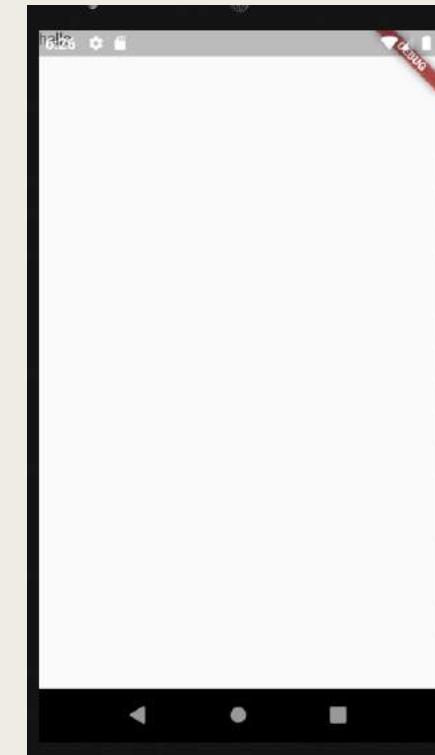
- It's an easy widget to use but also customizable.
- The Text constructor takes the arguments string, style, maxLines, overflow, textAlign, and others.

```
Text(  
  'Flutter World for Mobile',  
  style: TextStyle(  
    fontSize: 24.0,  
    color: Colors.deepPurple,  
    decoration: TextDecoration.underline,  
    decorationColor: Colors.deepPurpleAccent,  
    decorationStyle: TextDecorationStyle.dotted,  
    fontStyle: FontStyle.italic,  
    fontWeight: FontWeight.bold,  
  ),  
  maxLines: 4,  
  overflow: TextOverflow.ellipsis,  
  textAlign: TextAlign.justify,  
) ,
```

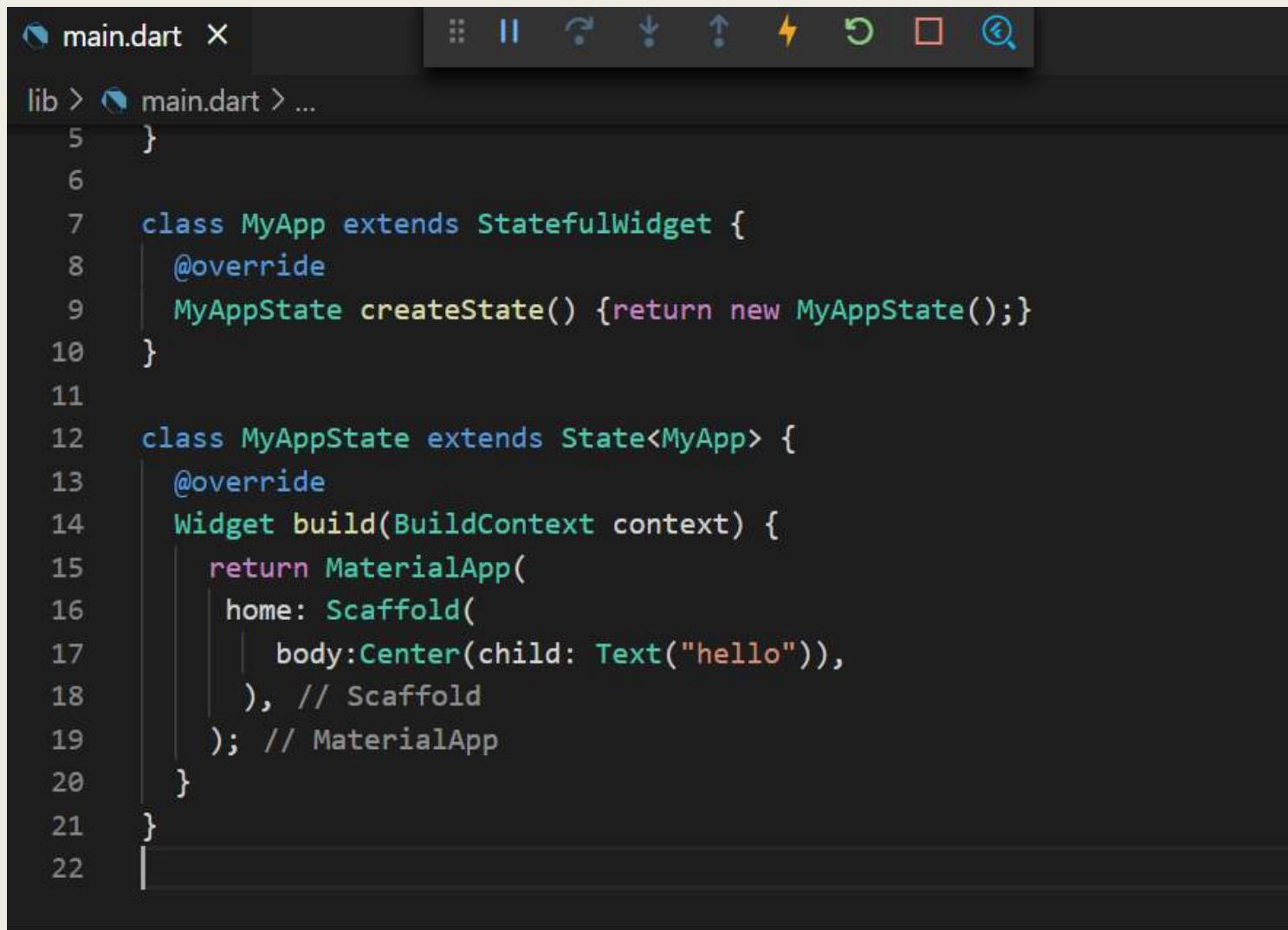
Text



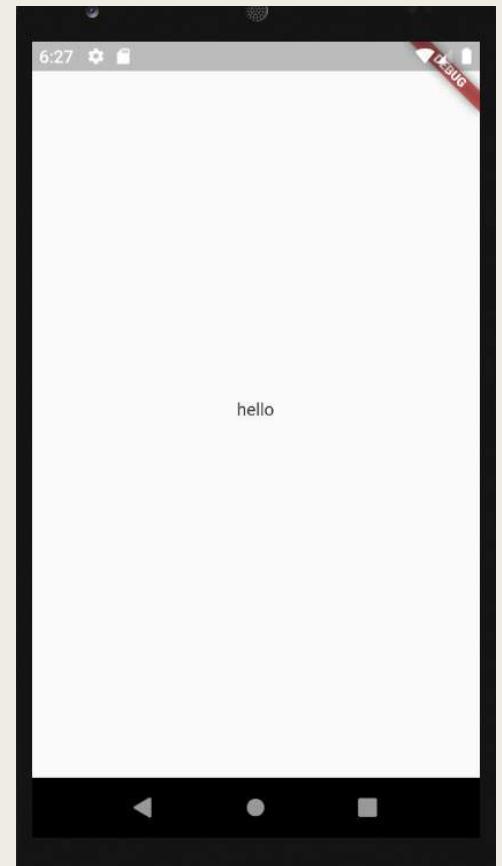
```
main.dart • lib > main.dart > MyAppState > build
5 }
6
7 class MyApp extends StatefulWidget {
8   @override
9   MyAppState createState() {return new MyAppState();}
10 }
11
12 class MyAppState extends State<MyApp> {
13   @override
14   Widget build(BuildContext context) {
15     return MaterialApp(
16       home: Scaffold(
17         body:Text("hello"),
18       ), // Scaffold
19     ); // MaterialApp
20   }
21 }
22
```



Text with Center Widget(Contd..)



```
main.dart X
lib > main.dart > ...
5 }
6
7 class MyApp extends StatefulWidget {
8   @override
9   MyAppState createState() => new MyAppState();
10 }
11
12 class MyAppState extends State<MyApp> {
13   @override
14   Widget build(BuildContext context) {
15     return MaterialApp(
16       home: Scaffold(
17         body: Center(child: Text("hello")),
18       ), // Scaffold
19     ); // MaterialApp
20   }
21 }
22
```



Text(Contd..)

```
main.dart
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatefulWidget {
8   @override
9   MyAppState createState() {return new MyAppState();}
10 }
11
12 class MyAppState extends State<MyApp> {
13   @override
14   Widget build(BuildContext context) {
15     return MaterialApp(
16       home: Scaffold(
17         body:Center(child: Text("hello",
18           style:TextStyle(color:Colors.red,fontWeight:FontWeight.bold),
19         ), // Text
20         ), // Center
21       ), // Scaffold
22     ); // MaterialApp
23   }
24 }
25
```



Rich Text

- The RichText widget is a great way to display text using multiple styles. The RichText widget takes
- TextSpan as children to style different parts of the strings

```
        child: Center(
            child: RichText(
                text: TextSpan(
                    text: 'Flutter World',
                    style: TextStyle(
                        fontSize: 24.0,
                        color: Colors.deepPurple,
                        decoration: TextDecoration.underline,
                        decorationColor: Colors.deepPurpleAccent,
                        decorationStyle: TextDecorationStyle.dotted,
                        fontStyle: FontStyle.italic,
                        fontWeight: FontWeight.normal,
                    ),
                    children: <TextSpan>[
                        TextSpan(
                            text: ' for',
                        ),
                        TextSpan(
                            text: ' Mobile',
                            style: TextStyle(
                                color: Colors.deepOrange,
                                fontStyle: FontStyle.normal,
                                fontWeight: FontWeight.bold),
                        ),
                    ],
                ),
            ),
        ),
```

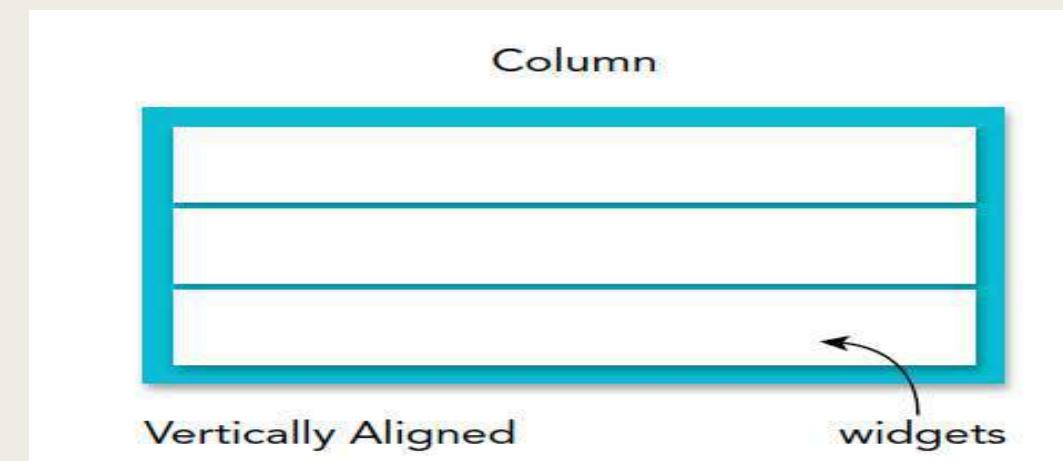


Flutter World for Mobile

FIGURE 6.1: RichText with TextSpan

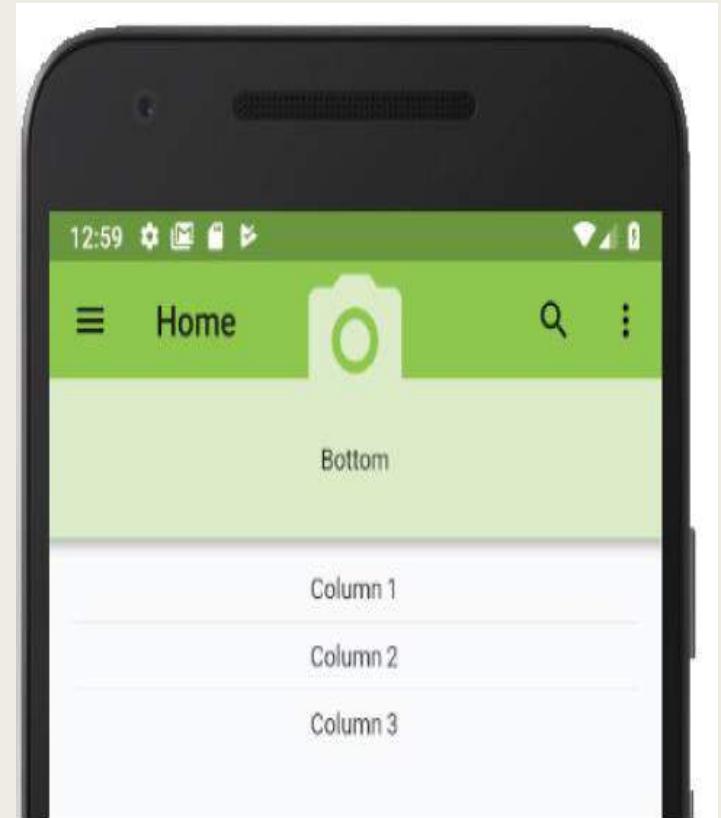
Column Widget

- Column widget displays its children vertically.
- It takes a children property containing an array of List<Widget>. The children align vertically without taking up the full height of the screen.
- Each child widget can be embedded in an Expanded widget to fill available space. You can use CrossAxisAlignment, MainAxisAlignment, and MainAxisSize to align and size how much space is occupied on the main axis



Column Widget

```
Column(  
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  mainAxisSize: MainAxisSize.max,  
  children: <Widget>[  
    Text('Column 1'),  
    Divider(),  
    Text('Column 2'),  
    Divider(),  
    Text('Column 3'),  
  ],  
) ,
```

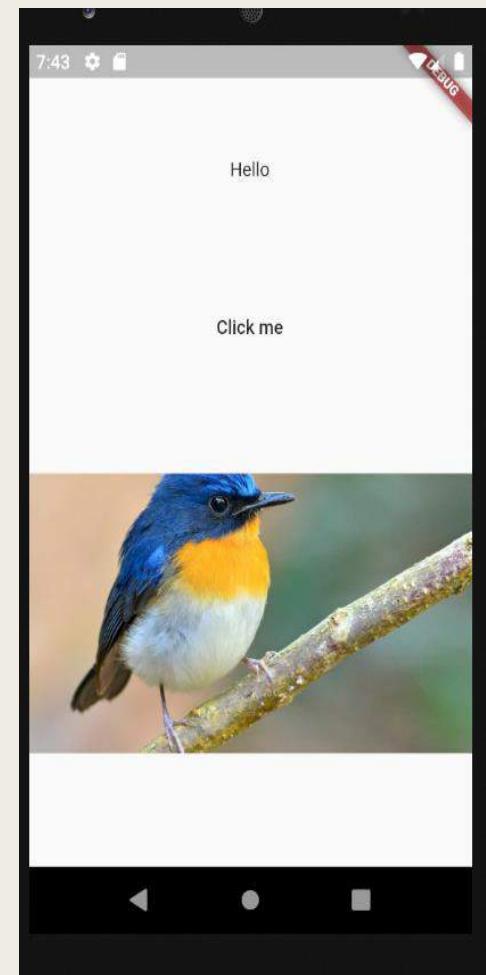


Column with Image Widget

File Edit Selection View Go Run Terminal Help

main.dart - column_widget - Visual Studio Code

```
lib > main.dart > MyAppState > build
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   @override
9   MyAppState createState() => new MyAppState();
10 }
11
12 class MyAppState extends State<MyApp> {
13   @override
14   Widget build(BuildContext context) {
15     return MaterialApp(
16       home: Scaffold(
17         body: Center(
18           child: Column(
19             mainAxisAlignment: MainAxisAlignment.spaceEvenly,
20             children: [
21               Text('Hello'),
22               FlatButton(
23                 child: Text("Click me"),
24                 onPressed: (){},
25               ), // FlatButton
26               Image.network("https://media.daysoftheyear.com/cdn-cgi/image/fit=cover,f=auto,onerror=redirect,width=1400,height=784/2017122311244"),
27             ],
28           ), // Column
29         ),
30       ),
31     );
32   }
33 }
```



Row Widget

- A Row widget displays its children horizontally.
- It takes a children property containing an array of List<Widget>. The same properties that the Column contains are applied to the Row widget with the exception that the **alignment is horizontal, not vertical**

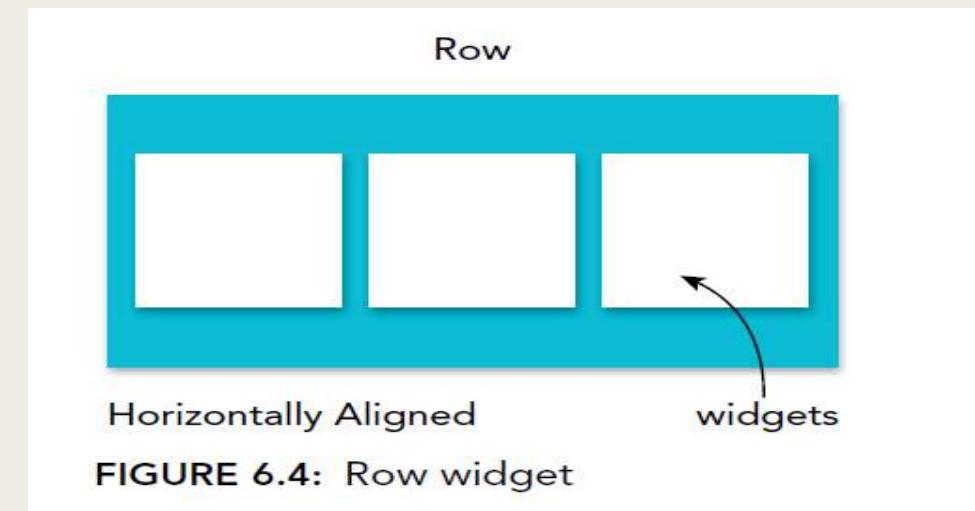
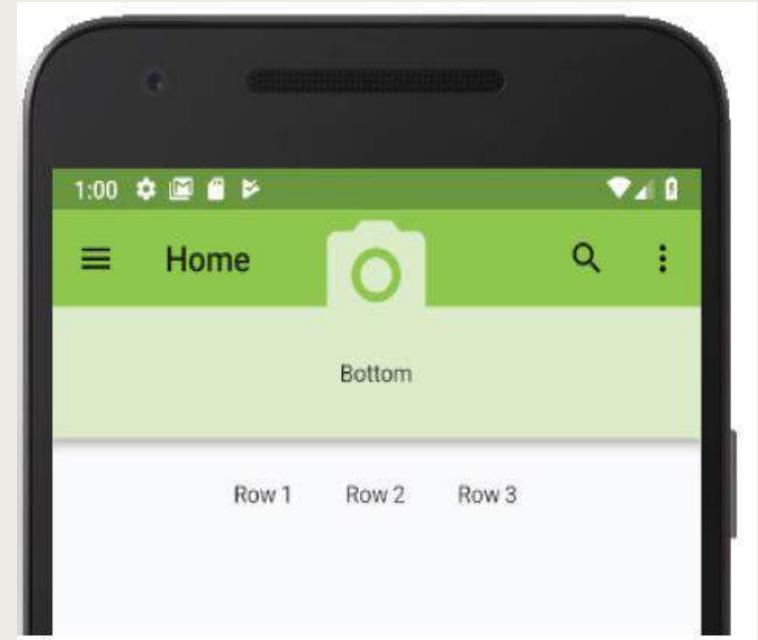


FIGURE 6.4: Row widget

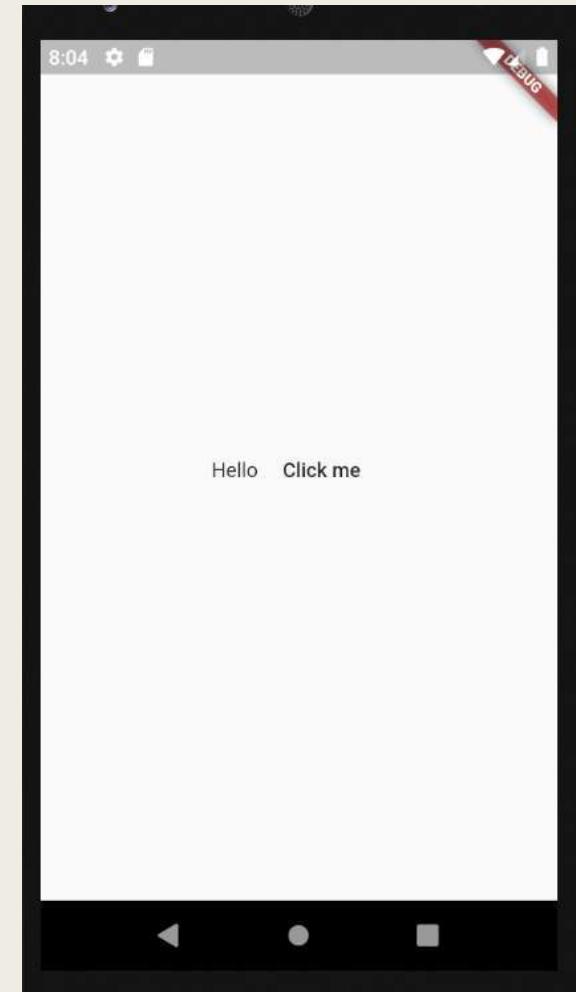
Row with Padding Widget

```
Row(  
    mainAxisAlignment: MainAxisAlignment.start,  
    mainAxisSize: MainAxisSize.max,  
    spaceEvenly:  
    children: <Widget>[  
        Row(  
            children: <Widget>[  
                Text('Row 1'),  
                Padding(padding: EdgeInsets.all(16.0),),  
                Text('Row 2'),  
                Padding(padding: EdgeInsets.all(16.0),),  
                Text('Row 3'),  
            ],  
        ),  
    ],  
)
```



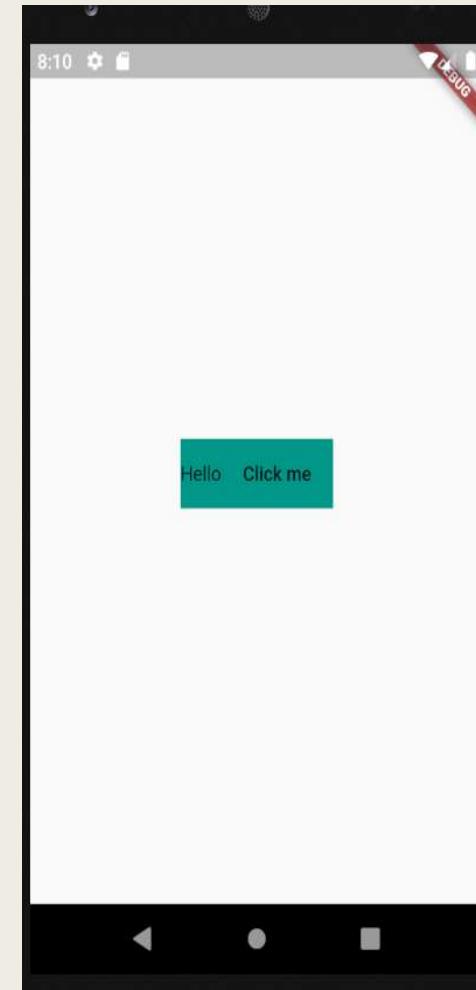
Row Widget

```
7   class MyApp extends StatefulWidget {
8     @override
9     MyAppState createState() {return new MyAppState();}
10    }
11
12    class MyAppState extends State<MyApp> {
13      @override
14      Widget build(BuildContext context) {
15        return MaterialApp(
16          home: Scaffold(
17            body: Center(
18              child: Row(
19                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
20                mainAxisSize: MainAxisSize.min,
21                children: [
22                  Text('Hello'),
23                  FlatButton(
24                    child: Text("Click me"),
25                    onPressed: (){},
26                  ), // FlatButton
27
28                ],
29              ), // Row
30
31            ), // Center
32          ), // Scaffold
33        ); // MaterialApp
34      }
35    }
```



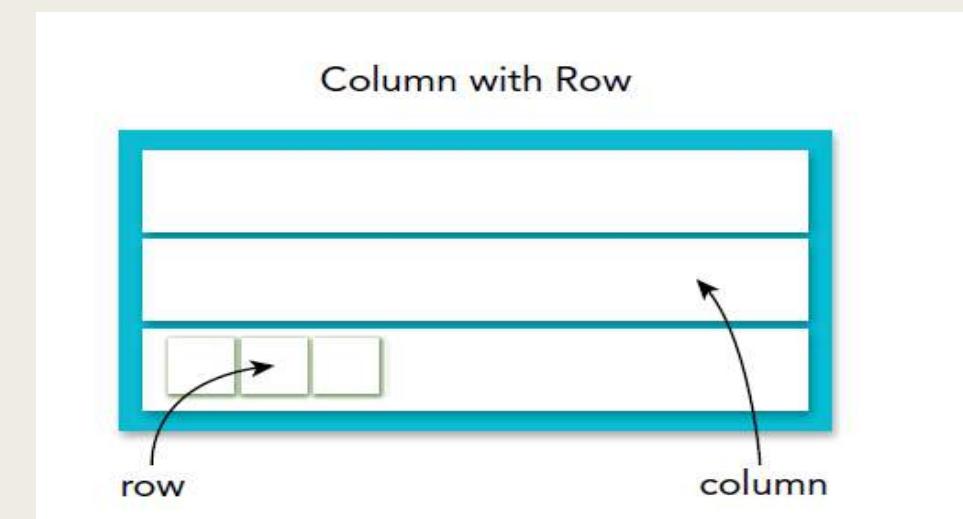
Row with Container Widget (Contd..)

```
11
12  class MyAppState extends State<MyApp> {
13    @override
14    Widget build(BuildContext context) {
15      return MaterialApp(
16        home: Scaffold(
17          body: Center(
18            child: Container(
19              color: Colors.teal,
20              child: Row(
21                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
22                mainAxisSize: MainAxisSize.min,
23                children: [
24                  Text('Hello'),
25                  FlatButton(
26                    child: Text("Click me"),
27                    onPressed: (){},
28                  ), // FlatButton
29                ],
30              ), // Row
31            ), // Container
32          ), // Center
33        ), // Scaffold
34      ); // MaterialApp
35    }
36 }
```



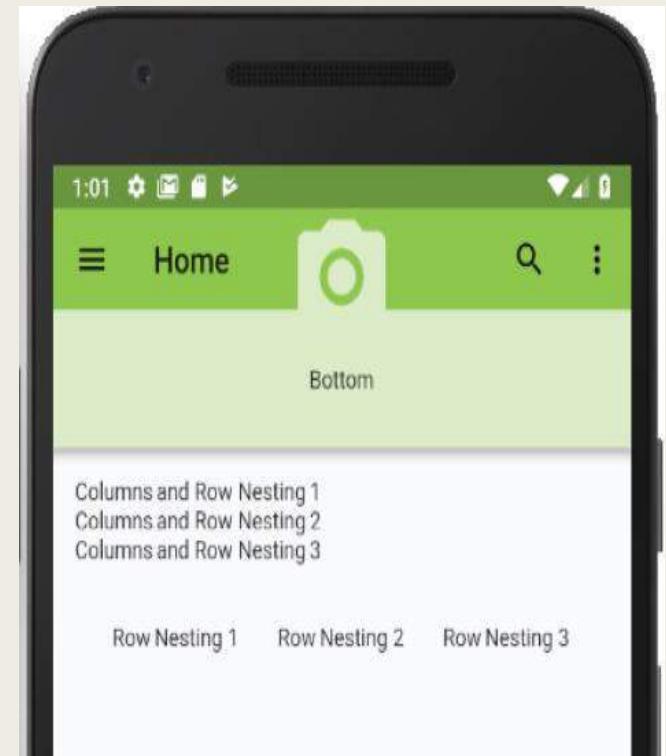
Rows and Columns

- A great way to create unique layouts is to combine Column and Row widgets for individual needs
- Imagine having a journal page with Text in a Column with a nested Row containing a list of images



Rows and Columns

```
Column(  
  mainAxisAlignment: MainAxisAlignment.start,  
  mainAxisSize: MainAxisSize.max,  
  children: <Widget>[  
    Text('Columns and Row Nesting 1'),  
    Text('Columns and Row Nesting 2'),  
    Text('Columns and Row Nesting 3'),  
    Padding(padding: EdgeInsets.all(16.0)),  
    Row(  
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
      children: <Widget>[  
        Text('Row Nesting 1'),  
        Text('Row Nesting 2'),  
        Text('Row Nesting 3'),  
      ],  
    ),  
  ],  
)
```



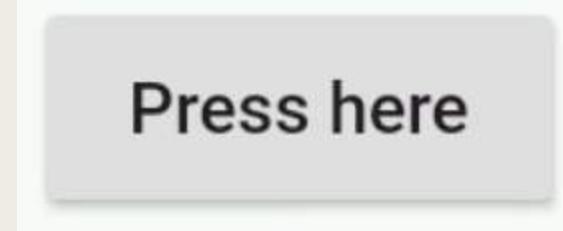
Button Widgets

- *Raised Button*
- *Raised Button with parameters*
- *Flat Button*
- *Icon Button*
- *Floating Action Button*
- *PopupMenu Button*
- *ButtonBar*

Raised Button

- RaisedButton is one of the most widely used widget in the flutter material library.
- It is actually a simple button which can handle normal click event.

```
const RaisedButton({  
    Key key,  
    @required this.onPressed,  
    this.color,  
    this.highlightColor,  
    this.splashColor,  
    this.disabledColor,  
    this.elevation: 2.0,  
    this.highlightElevation: 8.0,  
    this.disabledElevation: 0.0,  
    this.colorBrightness,  
    this.child  
}) : super(key: key);
```



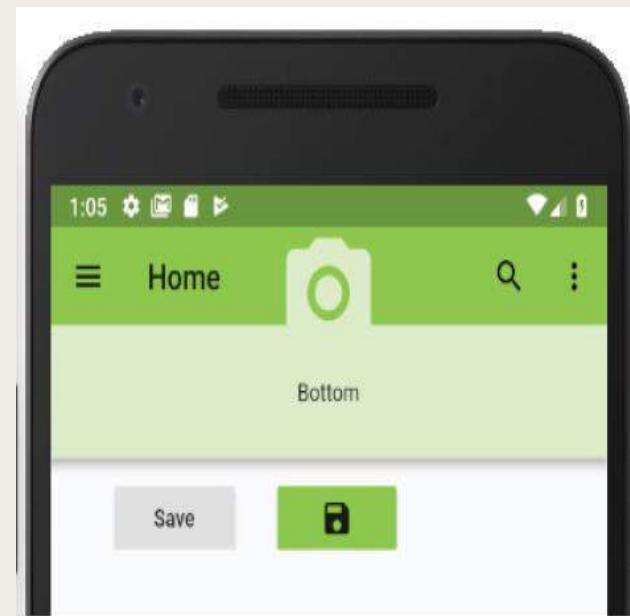
Press here

Raised Button

- The RaisedButton widget adds a dimension, and the elevation (shadow) increases when the user presses the button

```
// Default - left button
RaisedButton(
  onPressed: () {},
  child: Text('Save'),
),
```

```
// Customize - right button
RaisedButton(
  onPressed: () {},
  child: Icon(Icons.save),
  color: Colors.lightGreen,
),
```

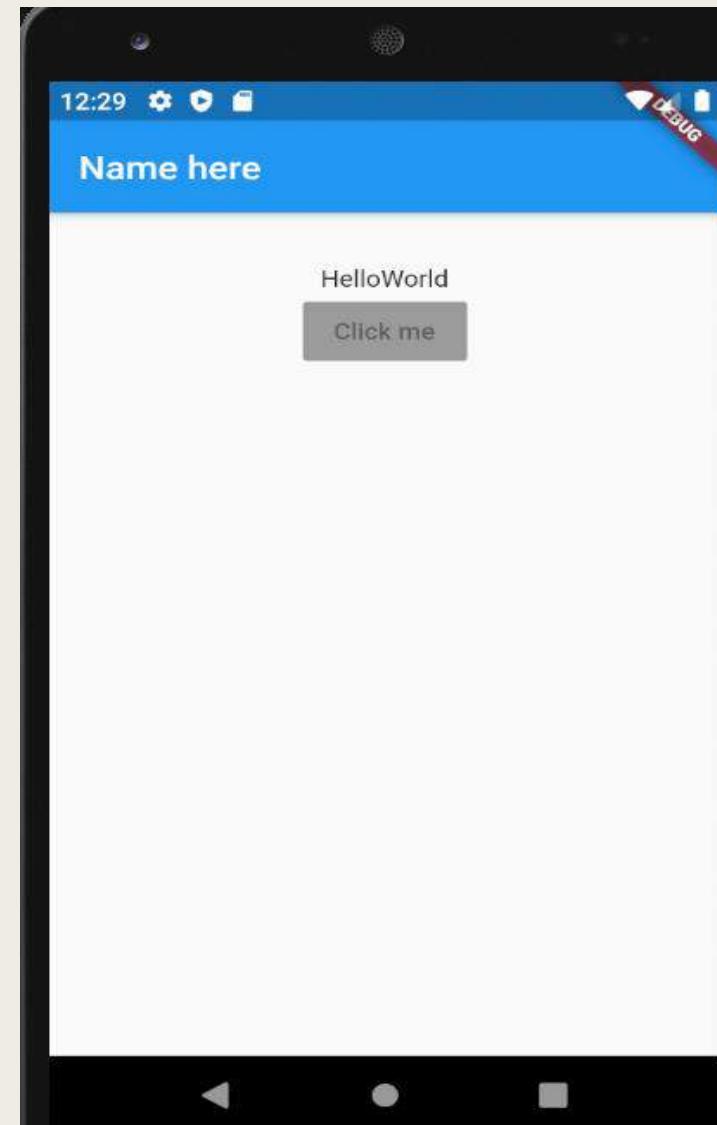


Raised Button

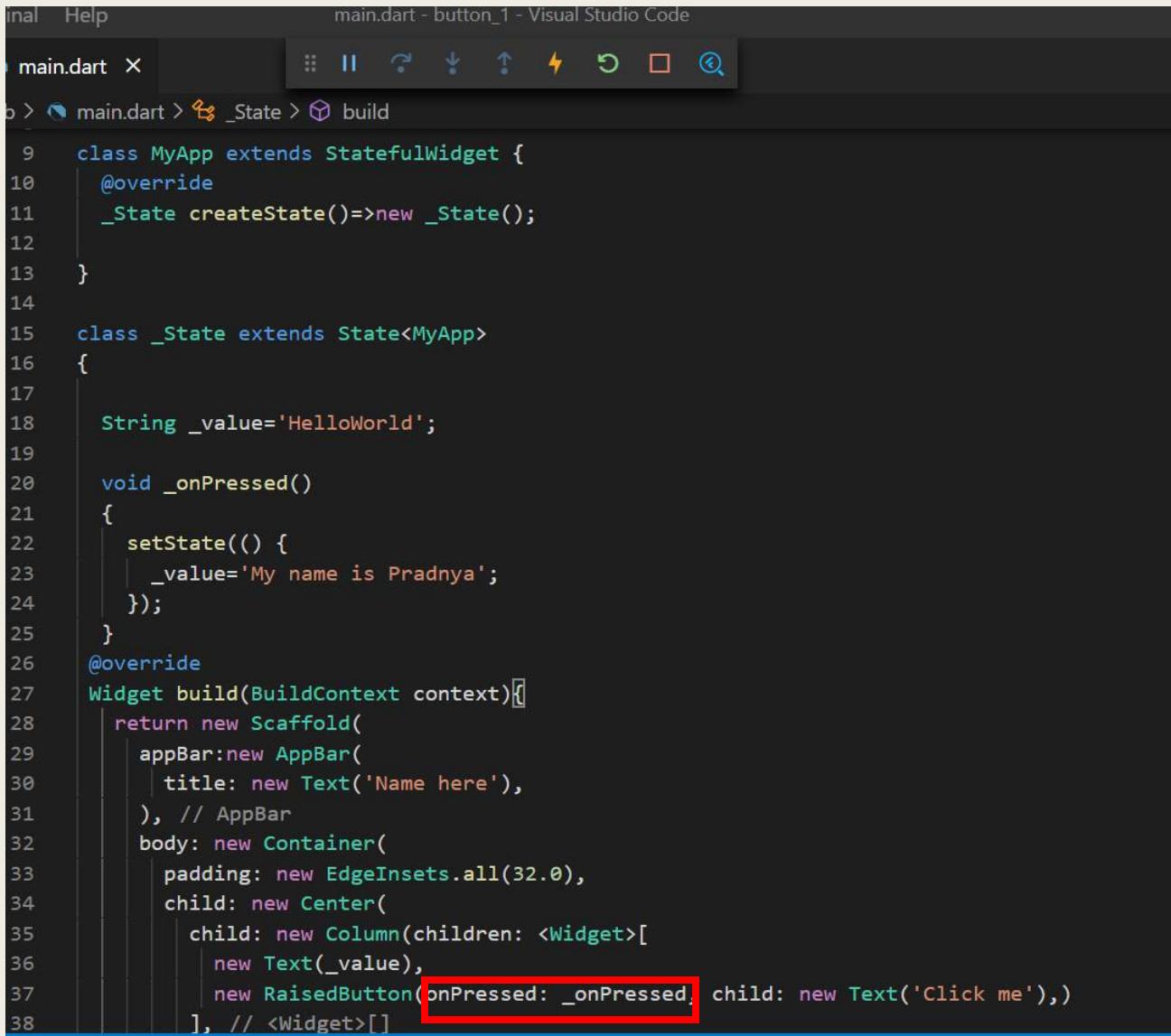
Terminal Help main.dart - button_1 - Visual Studio Code

main.dart X

```
lib > main.dart > _State > build
14
15     class _State extends State<MyApp>
16     {
17
18         String _value='HelloWorld';
19
20         void _onPressed()
21         {
22             setState(() {
23                 _value='My name is Pradnya';
24             });
25         }
26
27         @override
28         Widget build(BuildContext context){
29             return new Scaffold(
30                 appBar:new AppBar(
31                     title: new Text('Name here'),
32                 ), // AppBar
33                 body: new Container(
34                     padding: new EdgeInsets.all(32.0),
35                     child: new Center(
36                         child: new Column(children: <Widget>[
37                             new Text(_value),
38                             new RaisedButton(onPressed: null, child: new Text('Click me'),)
39                         ], // <Widget>[]
40                     ) // Column
41                 ), // Container
42             ); // Scaffold
43 }
```



Raised Button

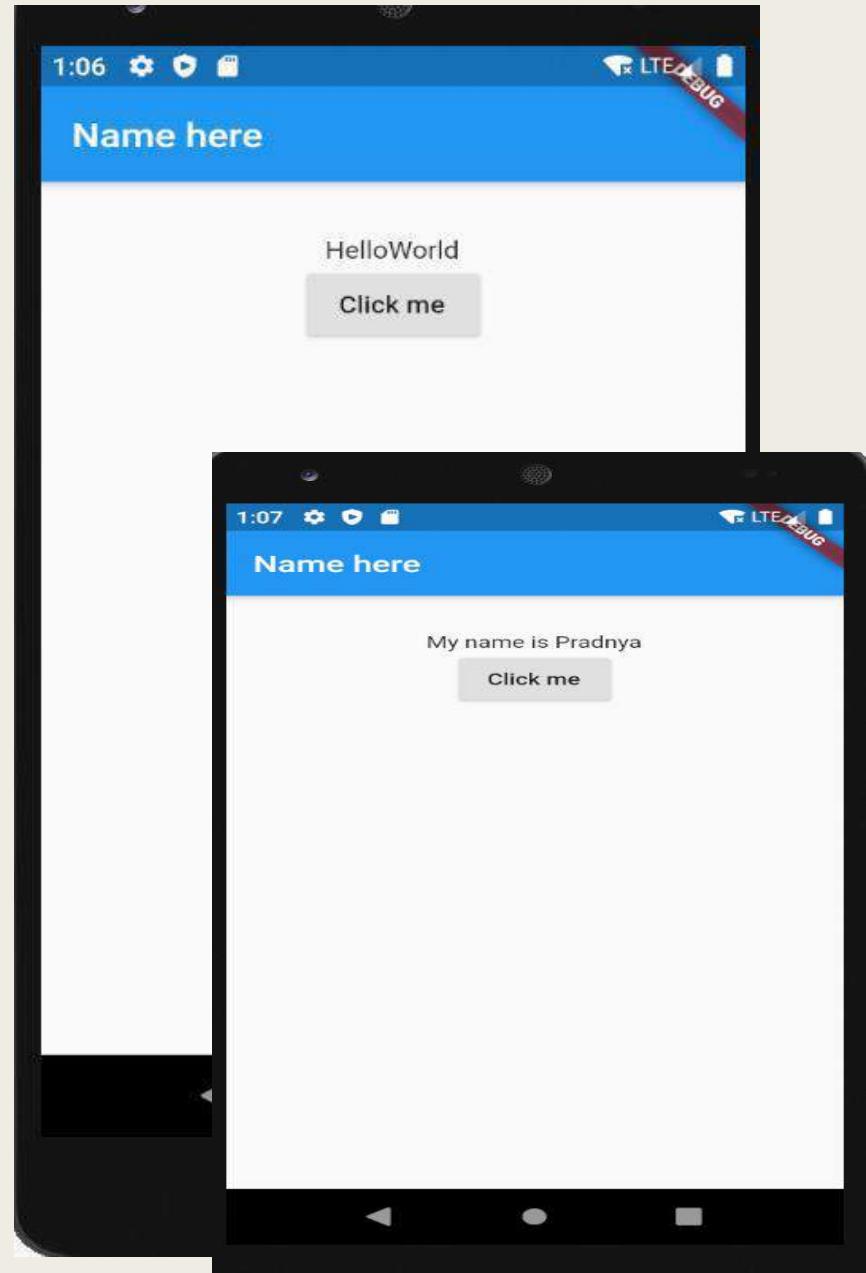


The screenshot shows the Visual Studio Code interface with the file `main.dart` open. The code implements a `RaisedButton` widget that changes its text when clicked.

```
final Help
main.dart - button_1 - Visual Studio Code
main.dart X
main.dart > _State > build
1:06 LTE DEBUG
Name here
HelloWorld
Click me
1:07 LTE DEBUG
Name here
My name is Pradnya
Click me

main.dart > _State > build
1:06 LTE DEBUG
Name here
HelloWorld
Click me
1:07 LTE DEBUG
Name here
My name is Pradnya
Click me

9   class MyApp extends StatelessWidget {
10     @override
11     _State createState() => new _State();
12   }
13 }
14
15 class _State extends State<MyApp>
16 {
17
18   String _value = 'HelloWorld';
19
20   void _onPressed()
21   {
22     setState(() {
23       _value = 'My name is Pradnya';
24     });
25   }
26   @override
27   Widget build(BuildContext context) {
28     return new Scaffold(
29       appBar: new AppBar(
30         title: new Text('Name here'),
31       ), // AppBar
32       body: new Container(
33         padding: new EdgeInsets.all(32.0),
34         child: new Center(
35           child: new Column(children: <Widget>[
36             new Text(_value),
37             new RaisedButton(onPressed: _onPressed, child: new Text('Click me'),)
38           ], // <Widget>[])
39         )
40       )
41     );
42   }
43 }
```

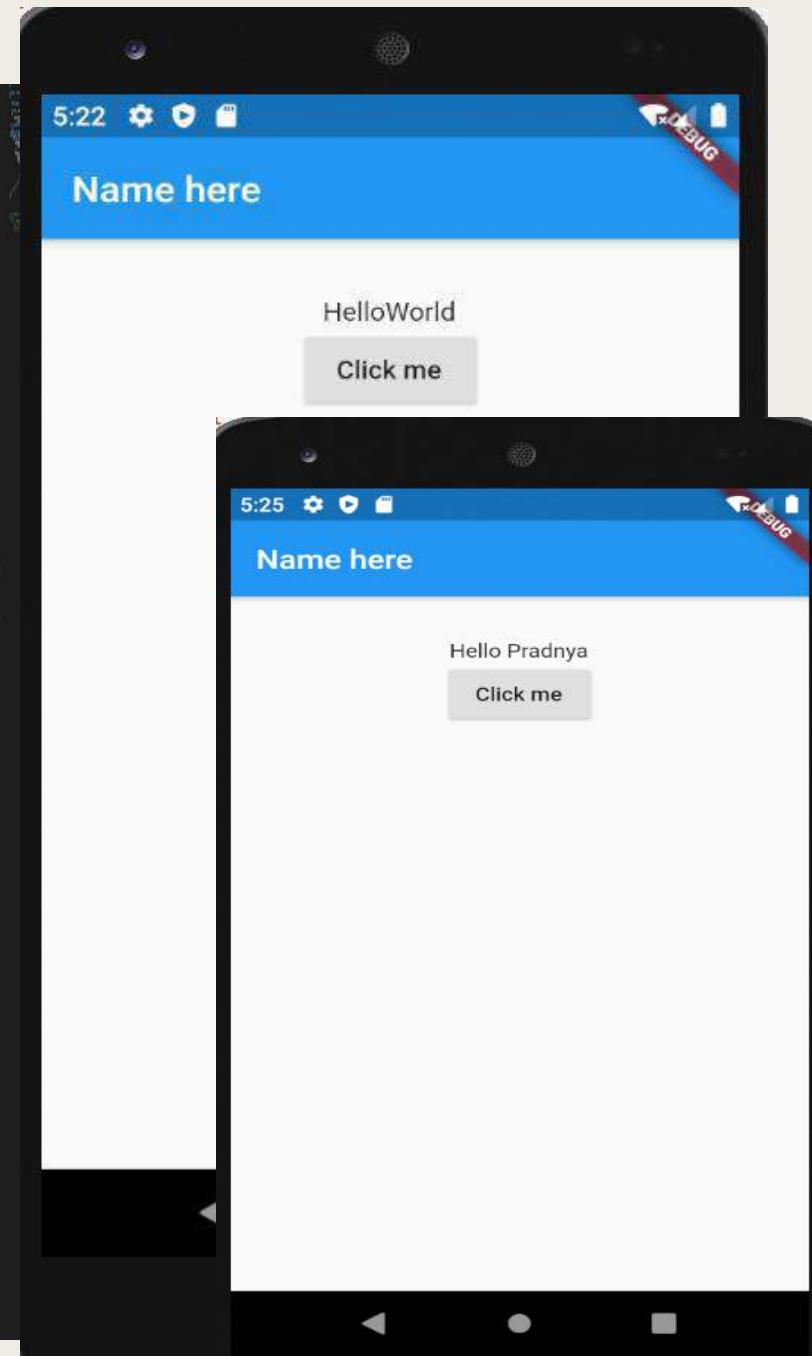


Raised Button with Parameters

```
class _State extends State<MyApp>
{
    String _value='HelloWorld';

    void _onPressed(String value)
    {
        setState(() {
            _value=value;
        });
    }

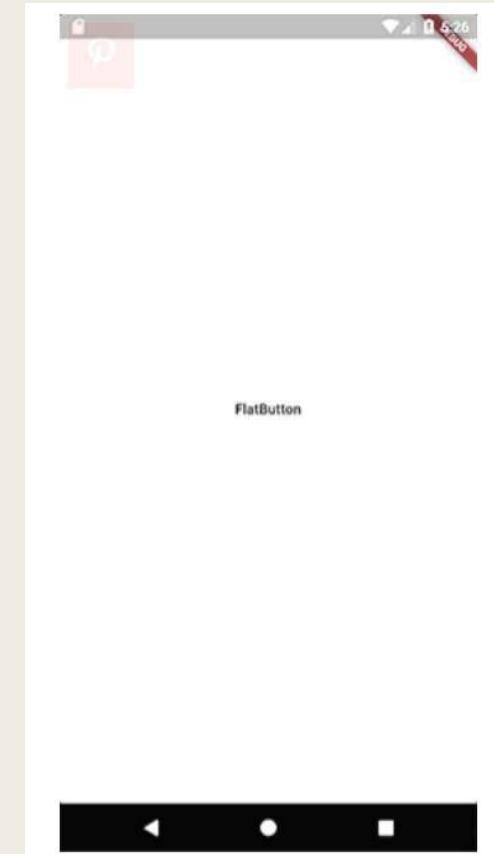
    @override
    Widget build(BuildContext context){
        return new Scaffold(
            appBar:new AppBar(
                title: new Text('Name here'),
            ), // AppBar
            body: new Container(
                padding: new EdgeInsets.all(32.0),
                child: new Center(
                    child: new Column(children: <Widget>[
                        new Text( value),
                        new RaisedButton(onPressed:()=> _onPressed('Hello Pradnya'), child: new Text('Click me'),),
                    ], // <Widget>[]
                ),
            ),
        );
    }
}
```



Flat Button

Simple button that has not much highlighted decoration, Mostly use on toolbar, dialog and etc.

```
Key key,  
@required VoidCallback onPressed,  
ValueChanged<bool> onHighlightChanged,  
ButtonTextTheme textTheme,  
Color textColor,  
Color disabledTextColor,  
Color color,  
Color disabledColor,  
Color highlightColor,  
Color splashColor,  
Brightness colorBrightness,  
EdgeInsetsGeometry padding,  
ShapeBorder shape,  
Clip clipBehavior = Clip.none,  
MaterialTapTargetSize materialTapTargetSize,  
@required Widget child,
```

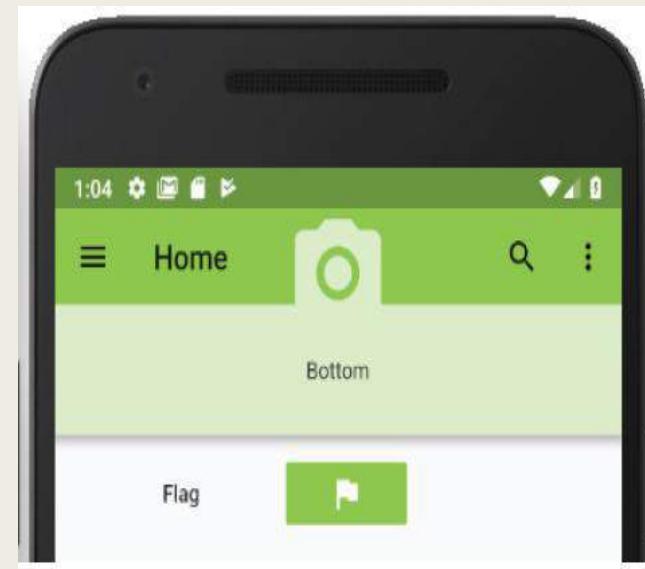


FlatButton

- The FlatButton widget is the **most minimalist button used**; it displays a text label without any borders or elevation (shadow).
- Since the text label is a widget, you could use an Icon widget instead or another widget to customize the button.
- Color, highlightColor, splashColor, textColor, and other properties can be customized.

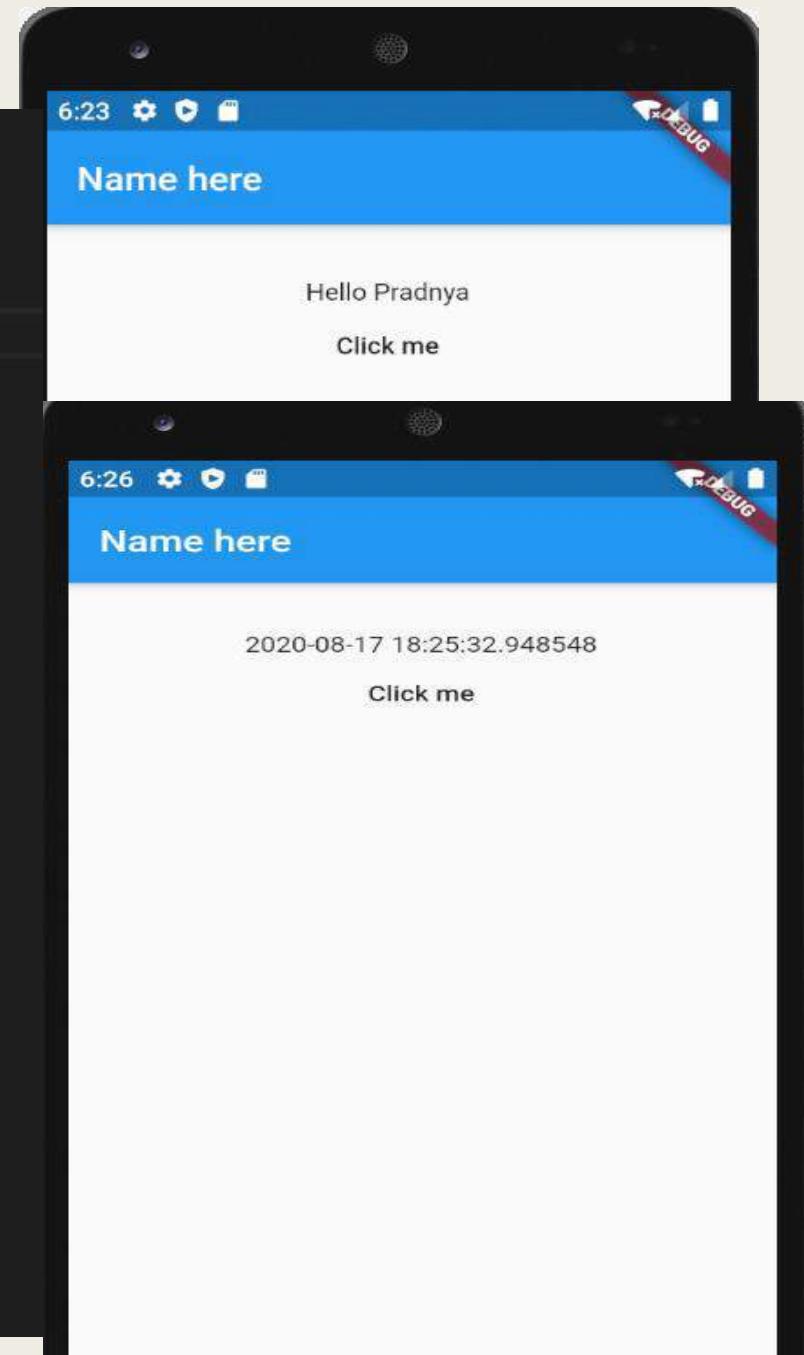
```
// Default - left button
FlatButton(
  onPressed: () {},
  child: Text('Flag'),
),

// Customize - right button
FlatButton(
  onPressed: () {},
  child: Icon(Icons.flag),
  color: Colors.lightGreen,
  textColor: Colors.white,
),
```



Flat Button

```
class _State extends State<MyApp>
{
  String _value='Hello Pradnya';
  void _onPressed()
  {
    setState(() {
      _value= new DateTime.now().toString();
    });
  }
  @override
  Widget build(BuildContext context){
    return new Scaffold(
      appBar: new AppBar(
        title: new Text('Name here'),
      ), // AppBar
      body: new Container(
        padding: new EdgeInsets.all(32.0),
        child: new Center(
          child: new Column(children: <Widget>[
            new Text( value),
            new FlatButton(onPressed:_onPressed, child: new Text('Click me'),)
          ], // <Widget>[])
    
```



Icon Button

- Simple display touchable icon, Trigger action when clicked.
- Mostly use with AppBar widget with actions property.

```
IconButton({  
    Key key,  
    this.iconSize = 24.0,  
    this.padding = const EdgeInsets.all(8.0),  
    this.alignment = Alignment.center,  
    @required this.icon,  
    this.color,  
    this.focusColor,  
    this.hoverColor,  
    this.highlightColor,  
    this.splashColor,  
    this.disabledColor,  
    @required this.onPressed,  
    this.focusNode,  
    this.autofocus = false,  
    this.tooltip,  
    this.enableFeedback = true,  
})
```



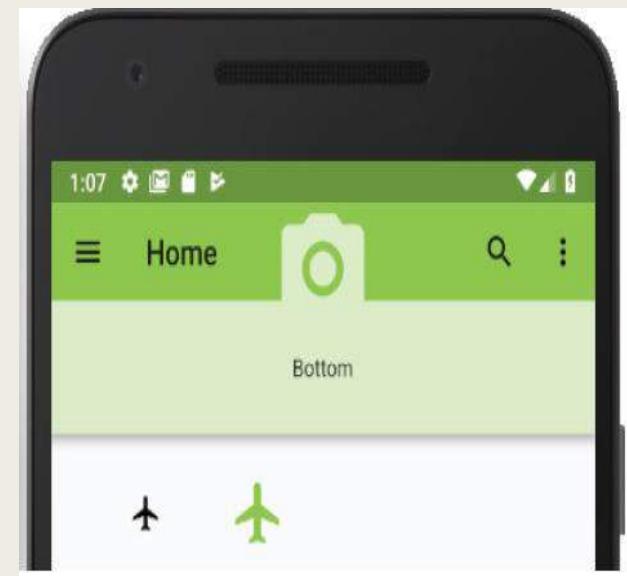
IconButton

IconButton

- The IconButton widget uses an Icon widget on a Material Component widget that reacts to touches by filling with color (ink).
- The combination creates a nice tap effect, giving the user feedback that an action has started.

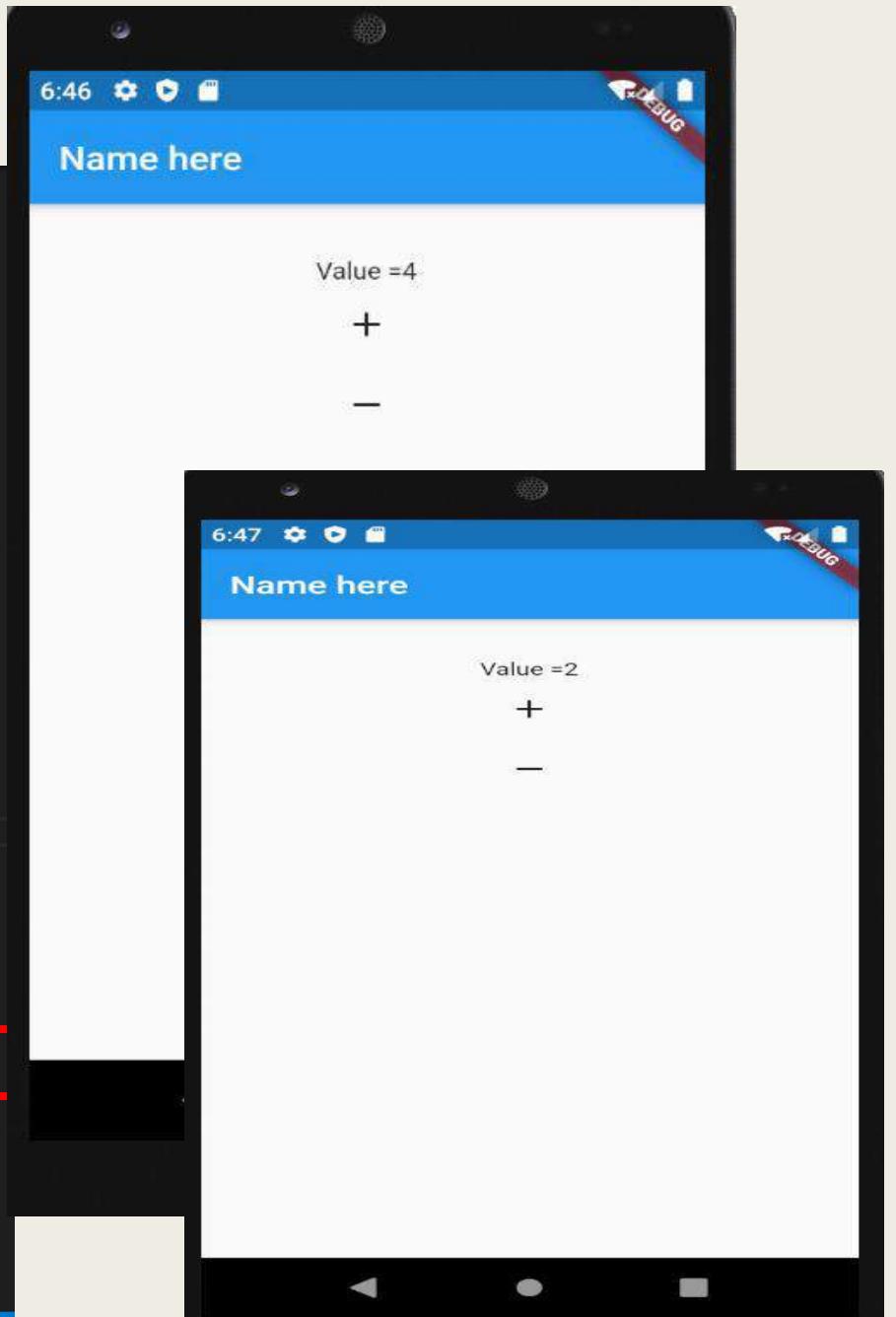
```
// Default - left button
IconButton(
  onPressed: () {},
  icon: Icon(Icons.flight),
),

// Customize - right button
IconButton(
  onPressed: () {},
  icon: Icon(Icons.flight),
  iconSize: 42.0,
  color: Colors.white,
  tooltip: 'Flight',
),
```



Icon Button

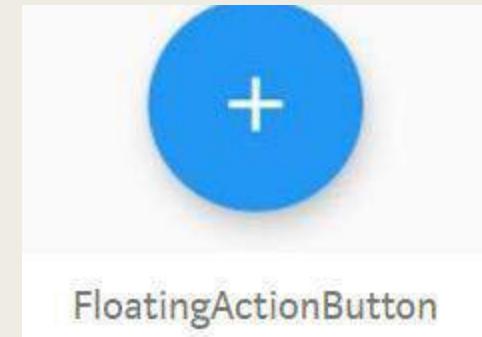
```
15 class _State extends State<MyApp>
16 {
17     int _value=0;
18
19     void _add()
20     {
21         setState(() {
22             _value++;
23         });
24     }
25
26     void _subtract()
27     {
28         setState(() {
29             _value--;
30         });
31     }
32     @override
33     Widget build(BuildContext context){
34         return new Scaffold(
35             appBar:new AppBar(
36                 title: new Text('Name here'),
37             ), // AppBar
38             body: new Container(
39                 padding: new EdgeInsets.all(32.0),
40                 child: new Center(
41                     child: new Column(children: <Widget>[
42                         new Text('Value =$ value'),
43                         new IconButton(icon: new Icon(Icons.add), onPressed: _add),
44                         new IconButton(icon: new Icon(Icons.remove), onPressed: _subtract)
45                     ], // <Widget>[]
46                 ) // Column
47             ), // Center
48         ), // Container
49     ); // Scaffold
50 }
51 }
```



Floating Action Button

- The FloatingActionButton widget is usually placed on the bottom right or center of the main screen in the Scaffold floatingActionButton property

```
FloatingActionButton({  
    Key key,  
    this.child,  
    this.tooltip,  
    this.foregroundColor,  
    this.backgroundColor,  
    this.focusColor,  
    this.hoverColor,  
    this.splashColor,  
    this.heroTag = const _DefaultHeroTag(),  
    this.elevation,  
    this.focusElevation,  
    this.hoverElevation,  
    this.highlightElevation,  
    this.disabledElevation,  
    @required this.onPressed,  
    this.mini = false,  
    this.shape,  
    this.clipBehavior = Clip.none,  
    this.focusNode,  
    this.autofocus = false,  
    this.materialTapTargetSize,  
    this.isExtended = false,  
})
```



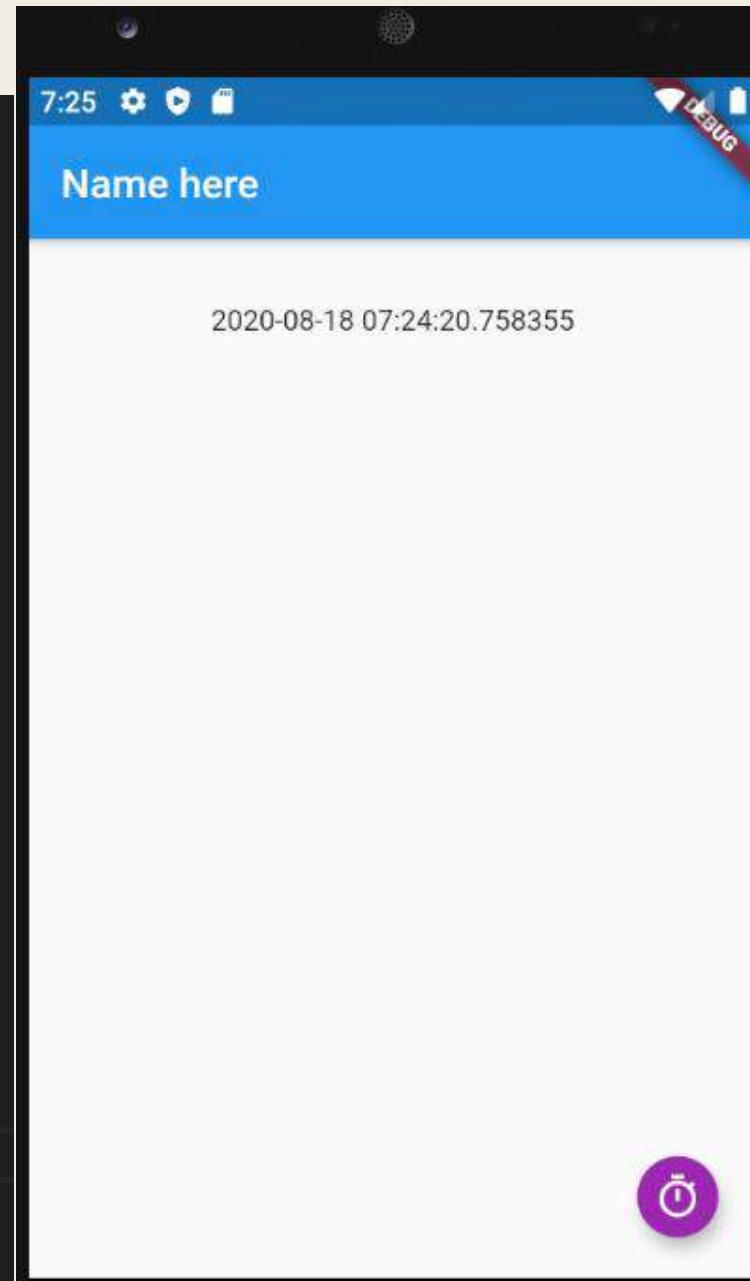
Floating Action Button

```
floatingActionButton: FloatingActionButton(  
    onPressed: () {},  
    child: Icon(Icons.play_arrow),  
    backgroundColor: Colors.lightGreen.shade100,  
,
```



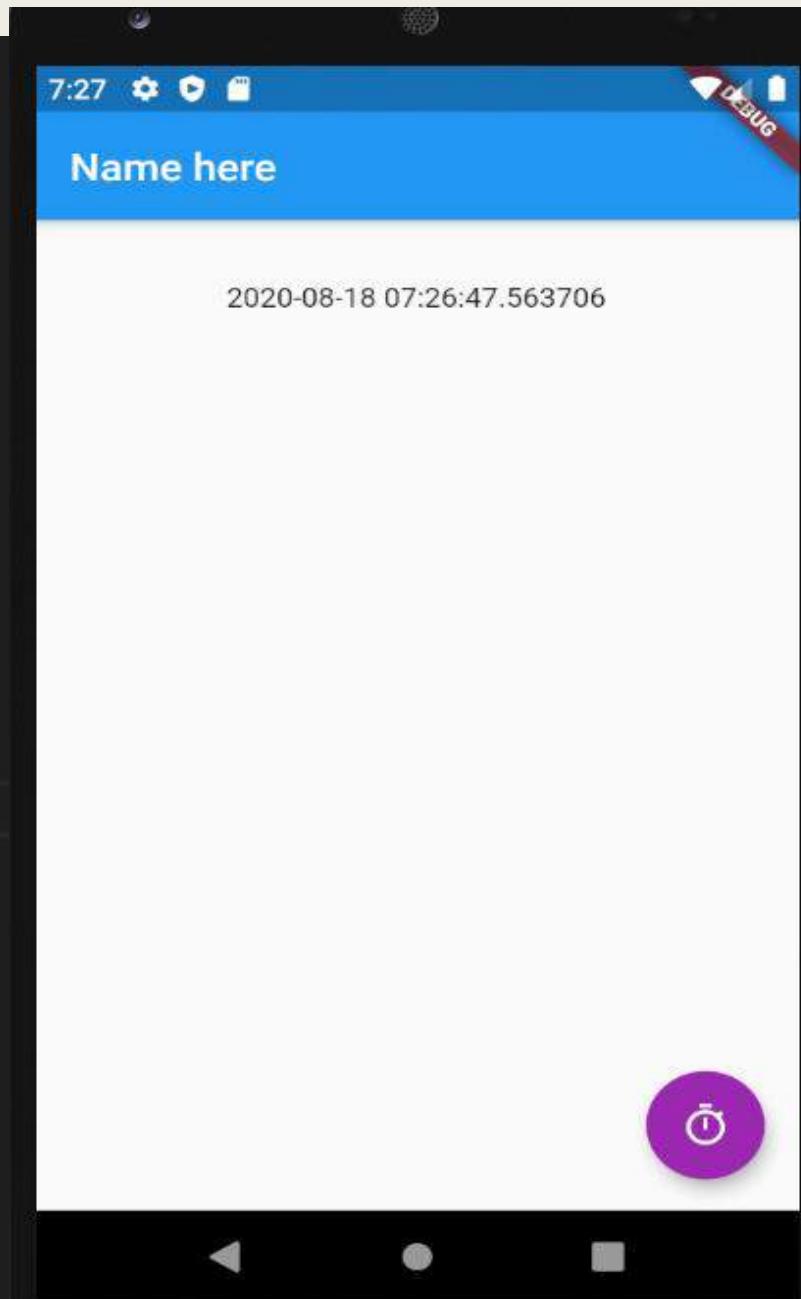
Floating Action Button

```
15 class _State extends State<MyApp>
16 {
17     String _value = '';
18     void _onClicked() => setState(()=> _value = new DateTime.now().toString());
19     @override
20     Widget build(BuildContext context){
21         return new Scaffold(
22             appBar: new AppBar(
23                 title: new Text('Name here'),
24             ), // AppBar
25             floatingActionButton: new FloatingActionButton(
26                 onPressed: _onClicked,
27                 backgroundColor: Colors.purple,
28                 mini: true,
29                 child: new Icon(Icons.timer),
30             ), // FloatingActionButton
31             body: new Container(
32                 padding: new EdgeInsets.all(32.0),
33                 child: new Center(
34                     child: new Column(children: <Widget>[
35                         new Text(_value),
36                     ],
37                 ),
38             ),
39         );
40     }
41 }
```



Floating Action Button(contd..)

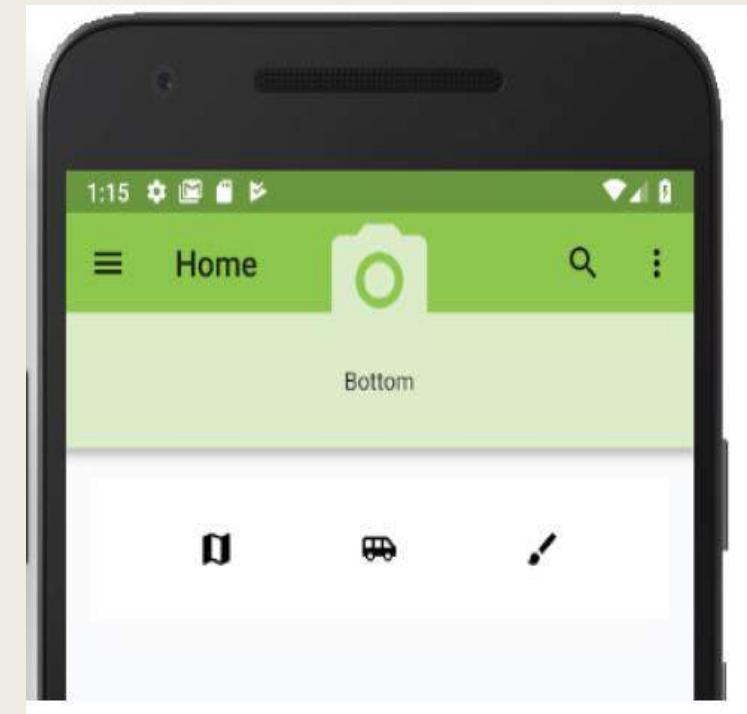
```
15 class _State extends State<MyApp>
16 {
17     String _value = '';
18     void _onClicked() => setState(()=> _value = new DateTime.now().toString());
19     @override
20     Widget build(BuildContext context){
21         return new Scaffold(
22             appBar: new AppBar(
23                 title: new Text('Name here'),
24             ), // AppBar
25             floatingActionButton: new FloatingActionButton(
26                 onPressed: _onClicked,
27                 backgroundColor: Colors.purple,
28                 mini: false, // Line 28
29                 child: new Icon(Icons.timer),
30             ), // FloatingActionButton
31             body: new Container(
32                 padding: new EdgeInsets.all(32.0),
33                 child: new Center(
34                     child: new Column(children: <Widget>[
35                         new Text(_value),
36                     ], // <Widget>[]
37                 ),
38             ),
39         );
40     }
41 }
```



ButtonBar

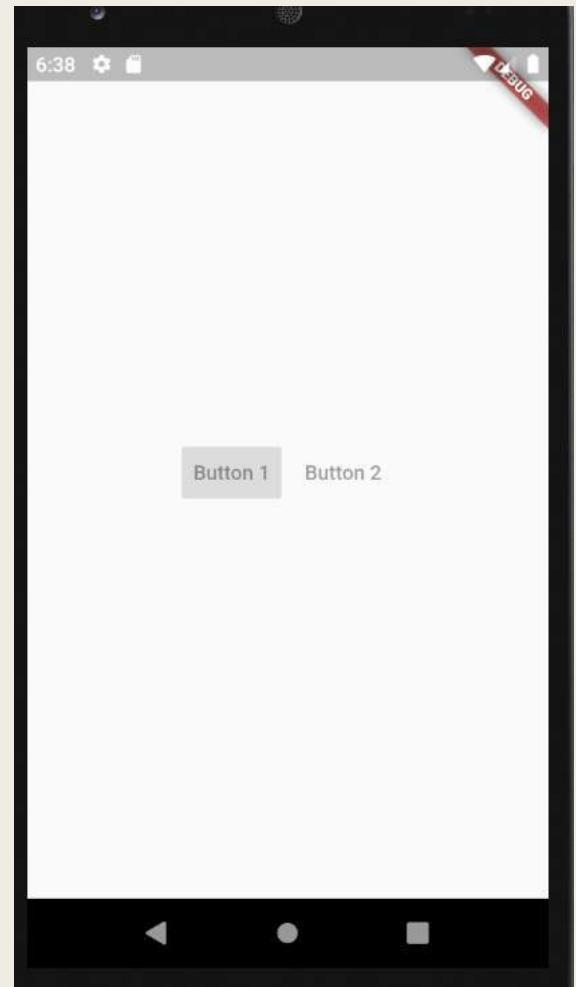
- The ButtonBar widget aligns buttons horizontally.
- In this example, the ButtonBar widget is a child of a Container widget to give it a background color.

```
Container(  
    color: Colors.white70,  
    child: ButtonBar(  
        alignment: MainAxisAlignment.spaceEvenly,  
        children: <Widget>[  
            IconButton(  
                icon: Icon(Icons.map),  
                onPressed: () {},  
            ),  
            IconButton(  
                icon: Icon(Icons.airport_shuttle),  
                onPressed: () {},  
            ),  
            IconButton(  
                icon: Icon(Icons.brush),  
                onPressed: () {},  
            ),  
        ],  
    ),  
) ,
```



Button Bar

```
lib > main.dart > main
1 import 'package:flutter/material.dart';
2
3     Run | Debug
4 void main() {
5     runApp(new MaterialApp(
6         title: 'Flutter Tutorial',
7         home: new MyApp(),
8     )); // MaterialApp
9 }
10 class MyApp extends StatelessWidget {
11     @override
12     Widget build(BuildContext context) {
13         return new Scaffold(
14             body: new Center(
15                 child: new ButtonBar(
16                     mainAxisAlignment: MainAxisAlignment.min,
17                     children: <Widget>[
18                         new RaisedButton(
19                             child: new Text('Button 1'),
20                             onPressed: null,
21                         ), // RaisedButton
22                         new FlatButton(
23                             child: new Text('Button 2'),
24                             onPressed: null,
25                         ), // FlatButton
26                     ], // <Widget>[]
27                 ), // ButtonBar
28             ), // Center
29         ); // Scaffold
30 }
```

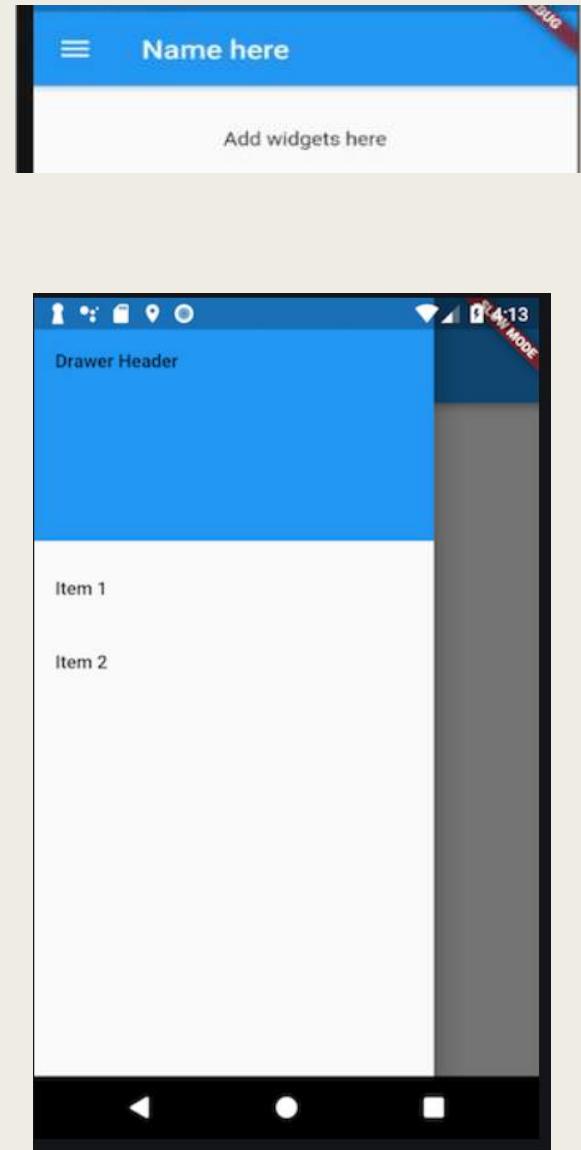


Scaffold Widget(contd..)

- Scaffold widget implements the basic Material Design visual layout, allowing you to easily add various widgets such as
 - *AppBar*
 - *FloatingActionButton*
 - *Drawer*
 - *BottomNavigationBar*
 - *BottomAppBar*
 - *Footer Buttons*

Drawer Widget

- A panel displayed to the side of the body, often hidden on mobile devices. Swipes in from either left-to-right or right-to-left
- Drawer can be customized for each individual need but usually has a header to show an image or fixed information
- Drawer is used when the **navigation list** has many items
- To set the Drawer header, you have two built-in options, the **UserAccountsDrawerHeader** or the **DrawerHeader**



Drawer Widget

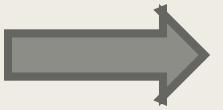
AppDrawer is a stateless widget, it's child is a ListView in which children attribute we will nest the following elements:

- **_createHeader()**: A function that returns a DrawerHeader widget.
- **_createDrawerItem()**: This function will return a new DrawerItem each time it is called.
- **Divider()**: We will use this widget between two DrawerItems whenever we want to separate items simulating sections.
- **ListTile()**: Last but not least, this attribute represents the Drawer's footer or its last element to be shown.



Drawer Widget Syntax

```
Scaffold(  
    drawer:  
)
```



```
drawer: Drawer(  
    child: ListView(  
        padding: EdgeInsets.zero,  
        children: [  
            DrawerHeader(  
                child: Text("Header"),  
            ),  
            ListTile(  
                title: Text("Home"),  
            ),  
        ],  
    ),  
)
```

Drawer Widget Syntax

```
drawer: Drawer(  
    child: ListView(  
        padding: EdgeInsets.zero,  
        children: [  
            DrawerHeader(  
                decoration: BoxDecoration(  
                    image: DecorationImage(  
                        image: AssetImage("images/header.jpeg"),  
                        fit: BoxFit.cover  
                    )  
                ),  
                child: Text("Header"),  
            ),  
            ListTile(  
                title: Text("Home"),  
            )  
        ],  
    ),  
)
```



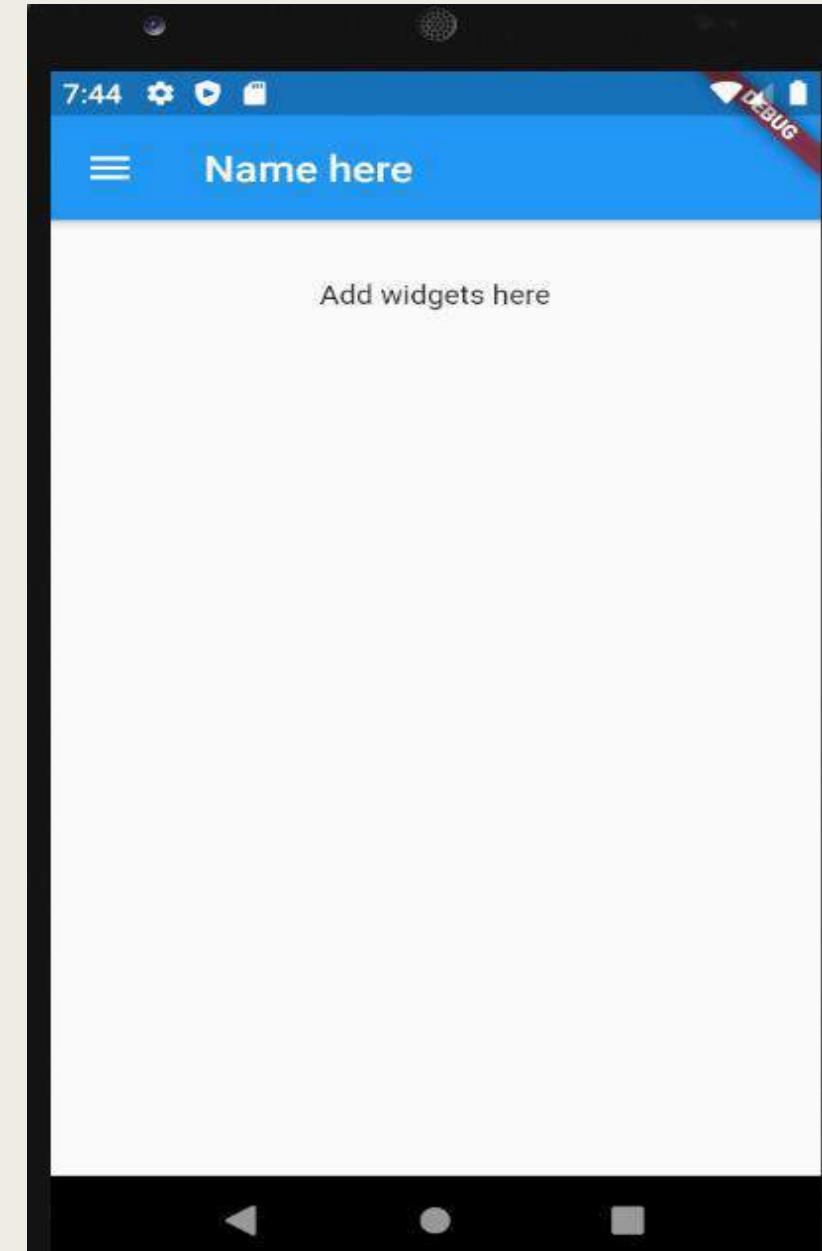
```
child: ListView(children: [  
    ListTile(  
        title: Text("Home"),  
        onTap: () {  
            Navigator.of(context).pop();  
        },  
    ),  
    ListTile(  
        title: Text("Home"),  
        onTap: () {  
            Navigator.of(context).pop();  
        },  
    ),  
],
```

Navigation Drawer in the right side

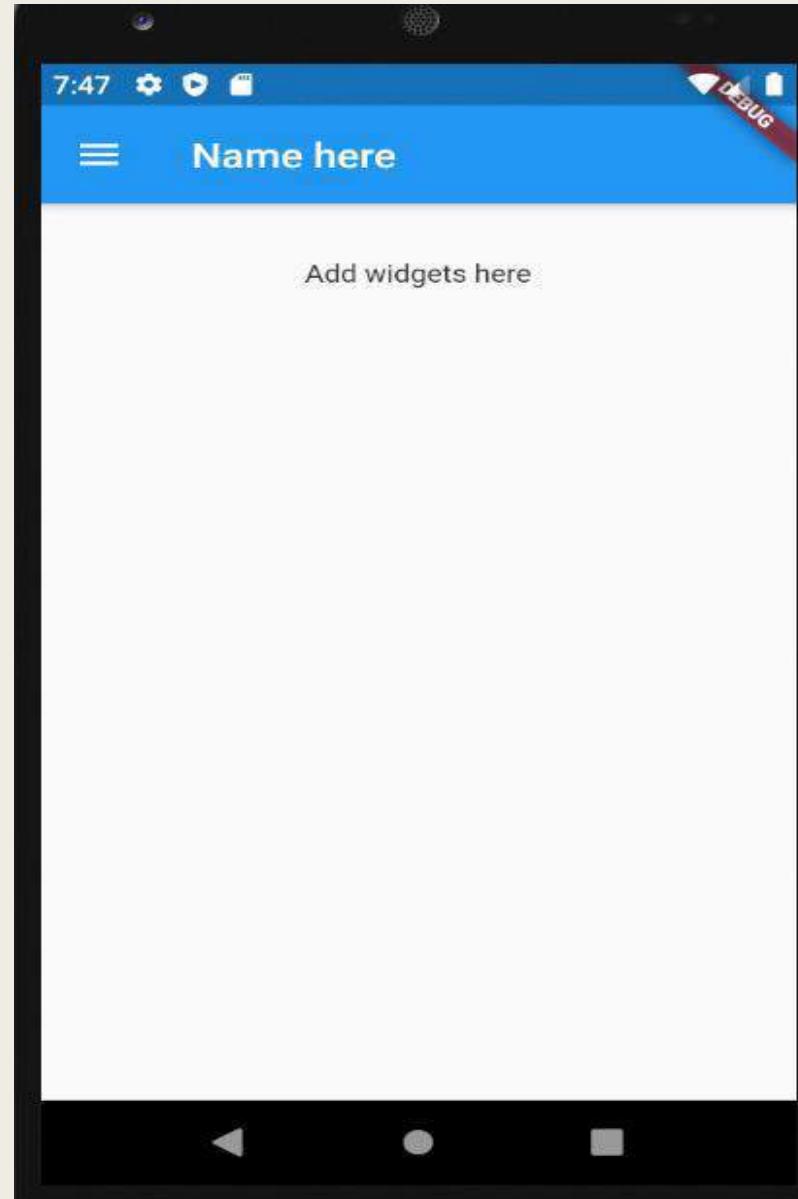
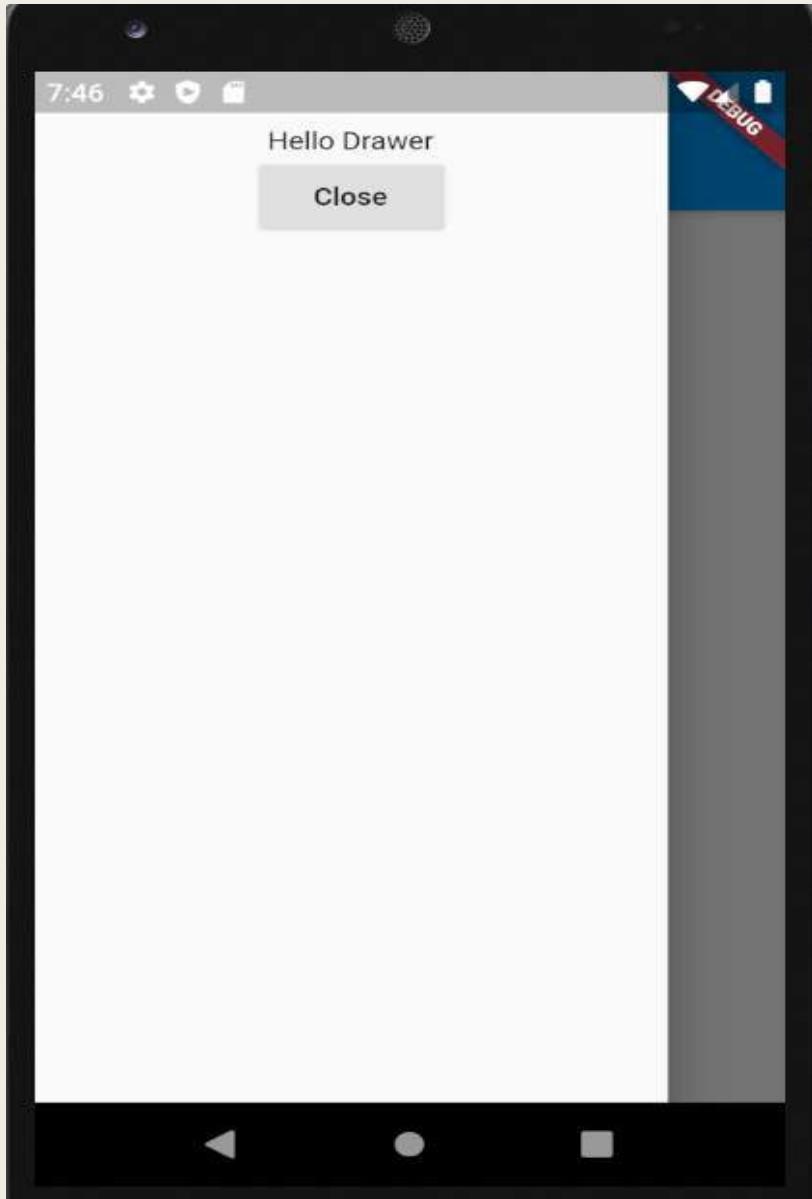
```
return Scaffold(  
    key: _scaffoldKey,  
    endDrawer: Drawer(  
        child: ListView(  
            padding: EdgeInsets.zero,  
            children: [  
                DrawerHeader(  
                    decoration: BoxDecoration(  
                        image: DecorationImage(  
                            image: AssetImage("images/header.jpeg"),  
                            fit: BoxFit.cover  
                        )  
                    ),  
                child: Text("Header"),  
            ),  
                ListTile(  
                    title: Text("Home"),  
                    onTap: (){  
                        Navigator.of(context).pop();  
                    },  
                ),  
            ],  
        ),  
    ),  
,
```

Drawer

```
15 class _State extends State<MyApp>
16 {
17     @override
18     Widget build(BuildContext context){
19         return new Scaffold(
20             appBar:new AppBar(
21                 title: new Text('Name here'),
22             ), // AppBar
23             drawer: new Drawer(
24                 child: new Container(
25                     padding: EdgeInsets.all(32.0),
26                     child: new Column(
27                         children: <Widget>[
28                             new Text('Hello Drawer'),
29                             new RaisedButton(onPressed:(){Navigator.pop(context)},child: new Text('Close'),)
30                         ],), // <Widget>[] // Column
31                 ), // Container
32
33             ), // Drawer
34             body: new Container(
35                 padding: new EdgeInsets.all(32.0),
36                 child: new Center(
37                     child: new Column(children: <Widget>[
38                         new Text('Add widgets here'),
39                     ], // <Widget>[]
39             
```



Drawer



BottomNavigation Bar

- A bottom navigation bar to display at the bottom of the scaffold
- BottomNavigationBar is used to provide a Navigation-like bar in the bottom of the App

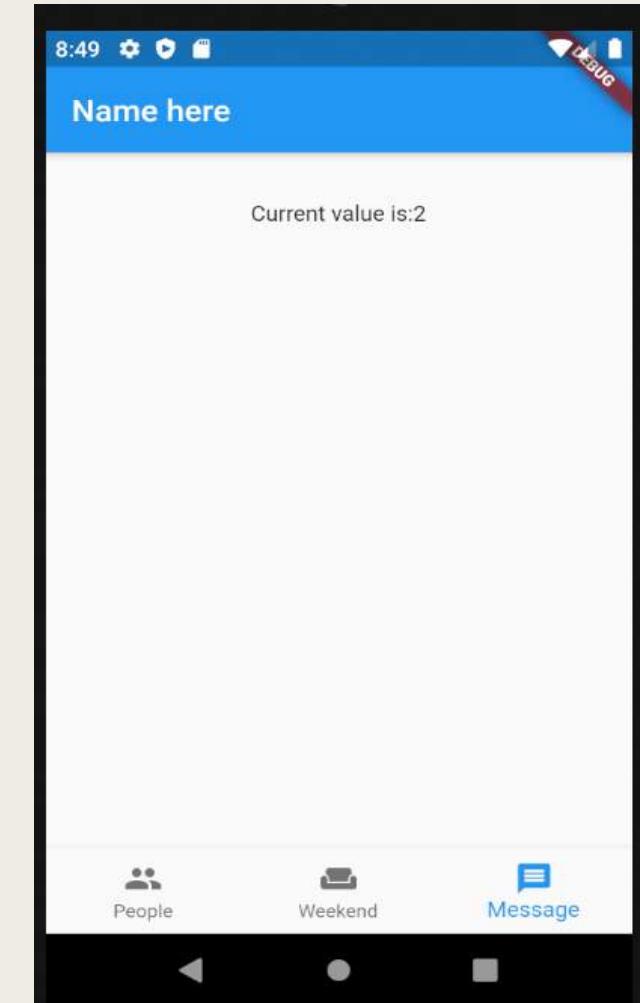
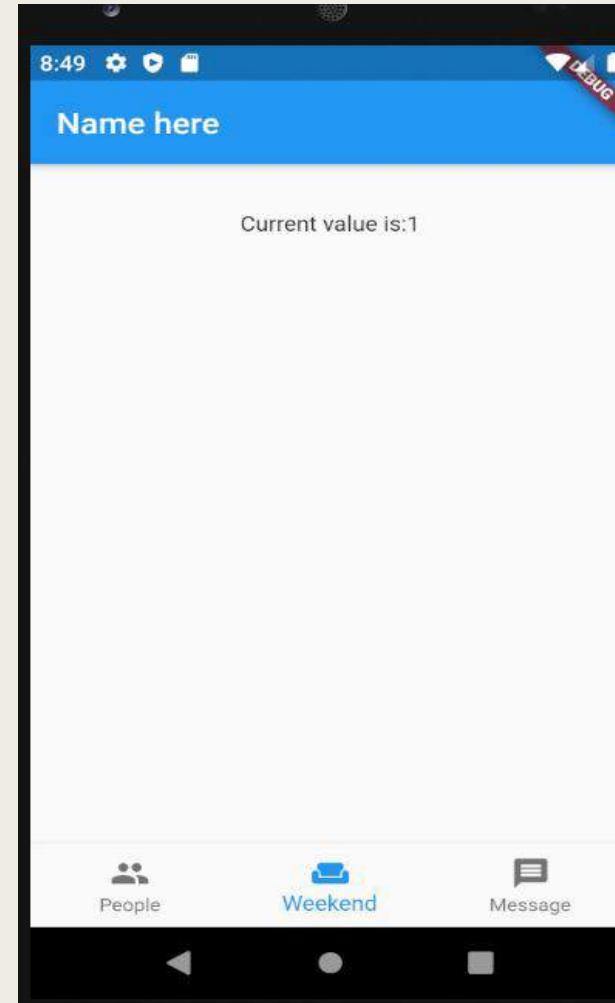
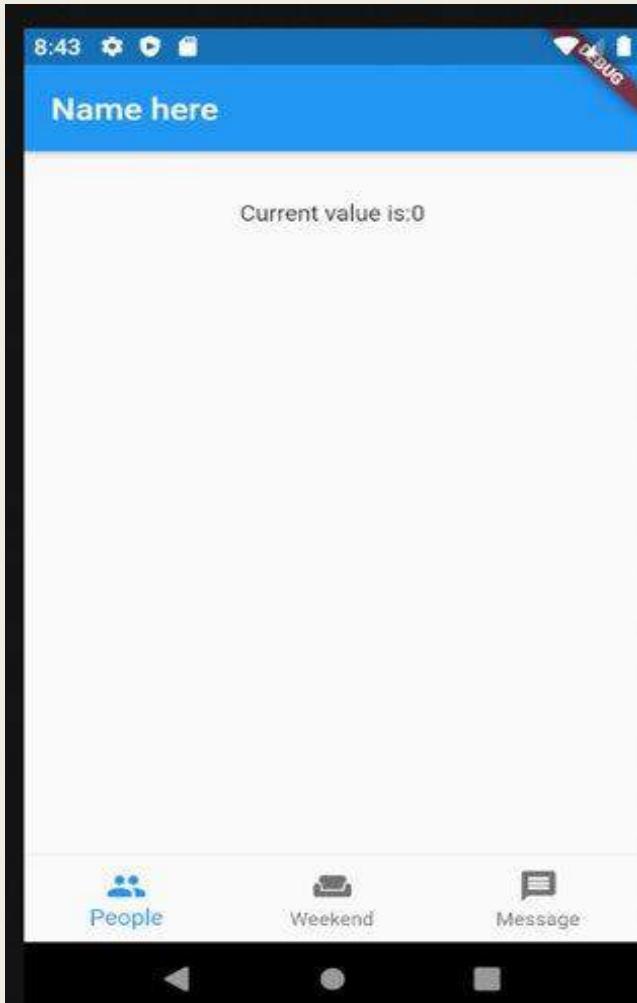


```
1  BottomNavigationBar({  
2      Key key,  
3      @required this.items,  
4      this.onTap,  
5      this.currentIndex = 0,  
6      BottomNavigationBarType type,  
7      this.fixedColor,  
8      this.iconSize = 24.0,  
9  })  
  
const BottomNavigationBarItem({  
    @required this.icon,  
    this.title,  
    Widget activeIcon,  
    this.backgroundColor,  
})
```

Bottom Navigation Bar

```
15 class _State extends State<MyApp>
16 {
17     List<BottomNavigationBarItem> _items;
18     String _value = '';
19     int _index = 0;
20
21     @override
22     void initState()
23     {
24         _items = new List();
25         _items.add(new BottomNavigationBarItem(icon: new Icon(Icons.people), title : new Text('People')));
26         _items.add(new BottomNavigationBarItem(icon: new Icon(Icons.weekend), title : new Text('Weekend')));
27         _items.add(new BottomNavigationBarItem(icon: new Icon(Icons.message), title : new Text('Message')));
28     }
29     @override
30     Widget build(BuildContext context){
31         return new Scaffold(
32             appBar:new AppBar(
33                 title: new Text('Name here'),
34             ), // AppBar
35             body: new Container(
36                 padding: new EdgeInsets.all(32.0),
37                 child: new Center(
38                     child: new Column(children: <Widget>[
39                         new Text(_value),
40                     ], // <Widget>[]
41                 ) // Column
45             bottomNavigationBar: new BottomNavigationBar(
46                 items: _items,
47                 fixedColor: Colors.blue,
48                 currentIndex: _index,
49                 onTap:(int item)
50                 {
51                     setState(() {
52                         _index = item;
53                         _value = 'Current value is:${_index.toString()}';
54                     });
55                 },
56             ), // BottomNavigationBar
57         ); // Scaffold
58     }
59 }
```

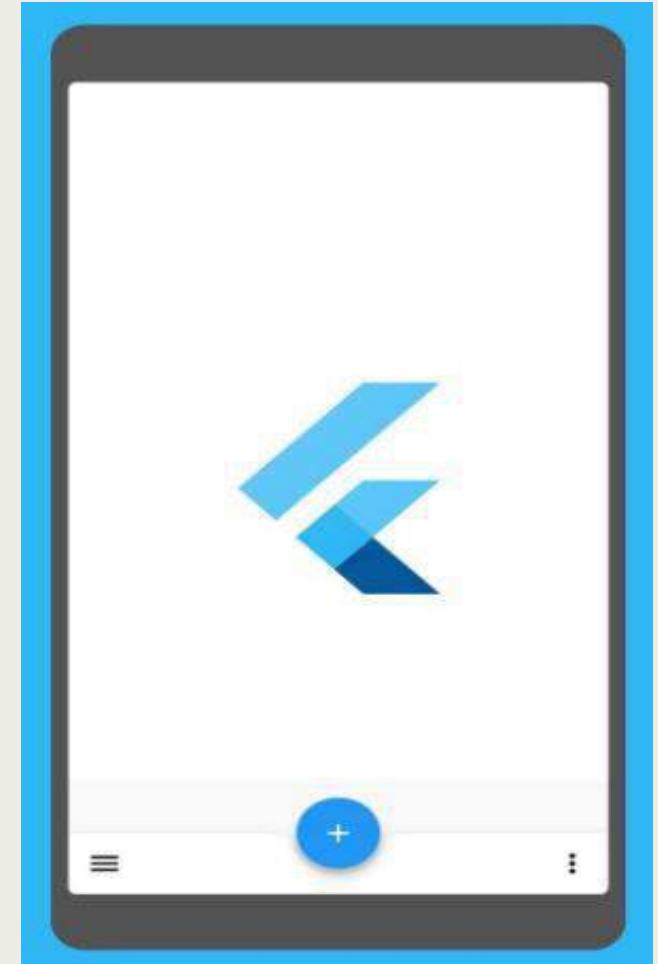
Bottom Navigation Bar(contd..)



BottomApp Bar

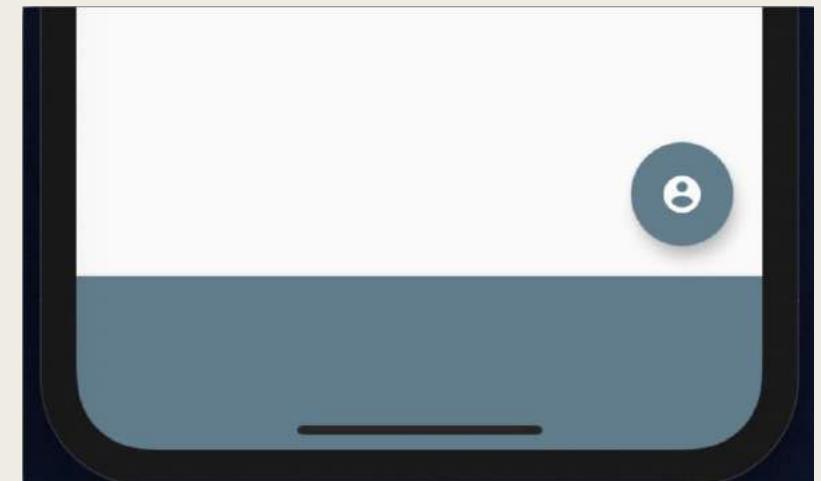
- BottomApp Bar widget allows BottomNavigation Bar to be combined with Floating Action Button

```
const BottomAppBar(  
    {Key key,  
     Color color,  
     double elevation,  
     NotchedShape shape,  
     Clip clipBehavior: Clip.none,  
     double notchMargin: 4.0,  
     Widget child}  
)
```



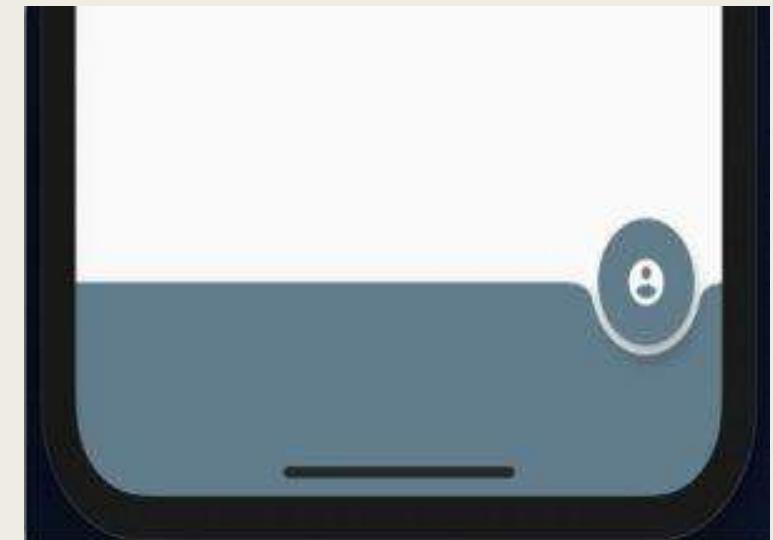
BottomApp Bar Versions(1)

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blueGrey,
        ), // AppBar
        body: MainPage(),
        bottomNavigationBar: BottomAppBar(
          color: Colors.blueGrey,
          shape: const CircularNotchedRectangle(),
          child: Container(
            height: 60.0,
          ), // Container
        ), // BottomAppBar
        floatingActionButton: FloatingActionButton(
          backgroundColor: Colors.blueGrey,
          onPressed: null,
          child: Icon(Icons.account_circle),
        ), // FloatingActionButton
      ), // Scaffold
    ); // MaterialApp
  }
}
```

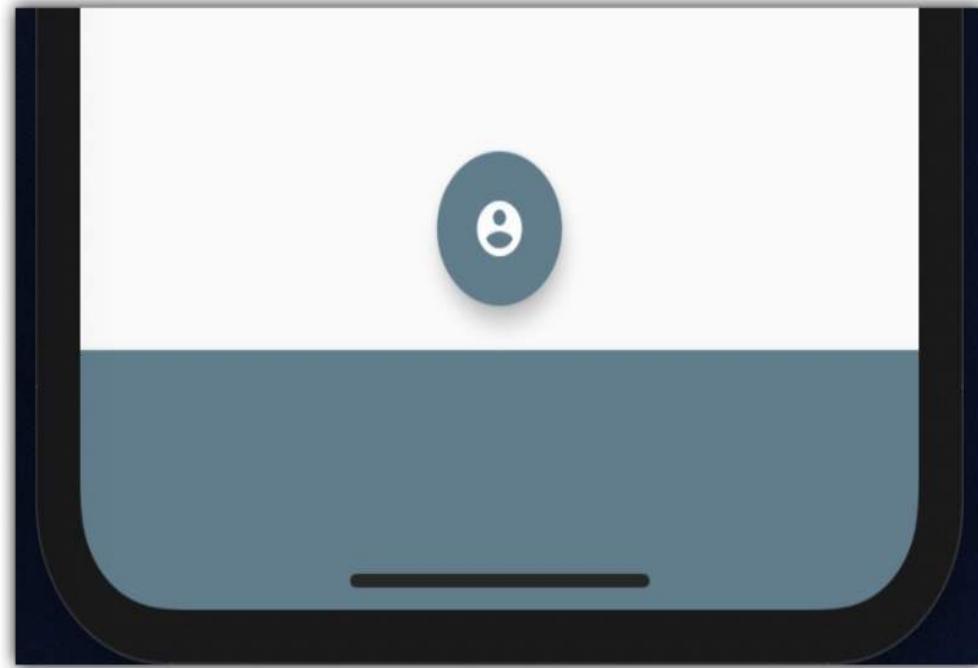


BottomApp Bar Versions(2)

```
bottomNavigationBar: BottomAppBar(  
    color: Colors.blueGrey,  
    shape: const CircularNotchedRectangle(),  
    child: Container(  
        height: 60.0,  
    ),  
,  
floatingActionButton: FloatingActionButton(  
    backgroundColor: Colors.blueGrey,  
    onPressed: null,  
    child: Icon(Icons.account_circle),  
,  
floatingActionButtonLocation: FloatingActionButtonLocation.endDocked,
```

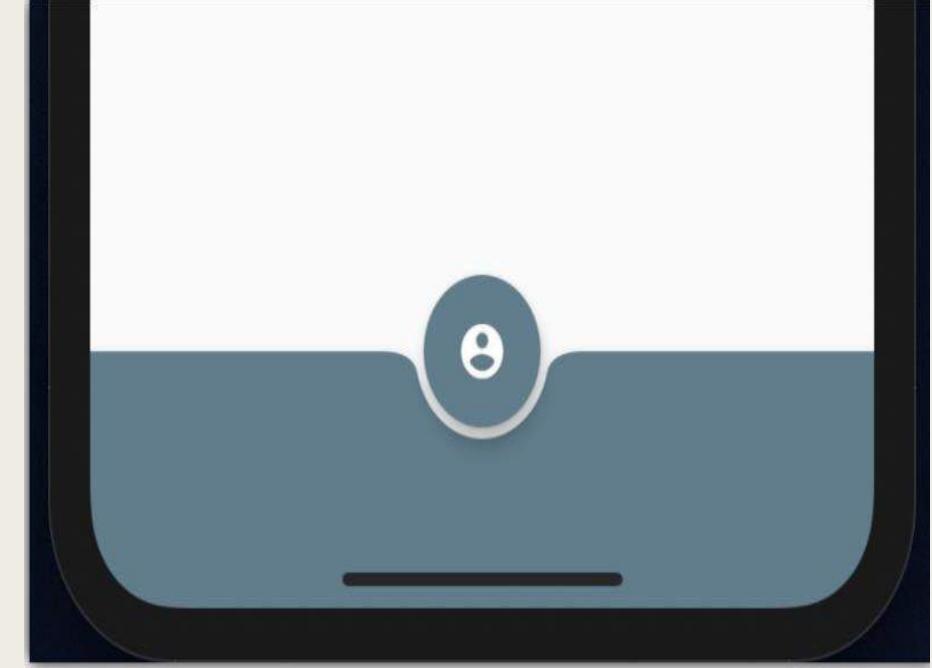


BottomApp Bar Versions(3)



Changing only `floatingActionButtonLocation` property

`FloatingActionButtonLocation.centerFloat`

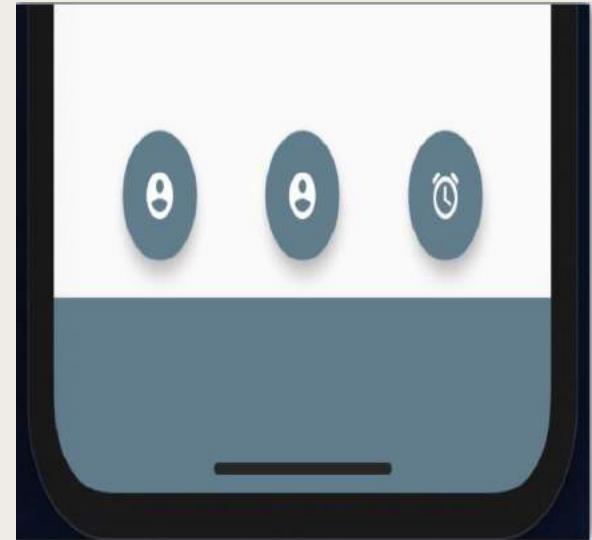


Changing only `floatingActionButtonLocation` property

`FloatingActionButtonLocation.centerDocked`

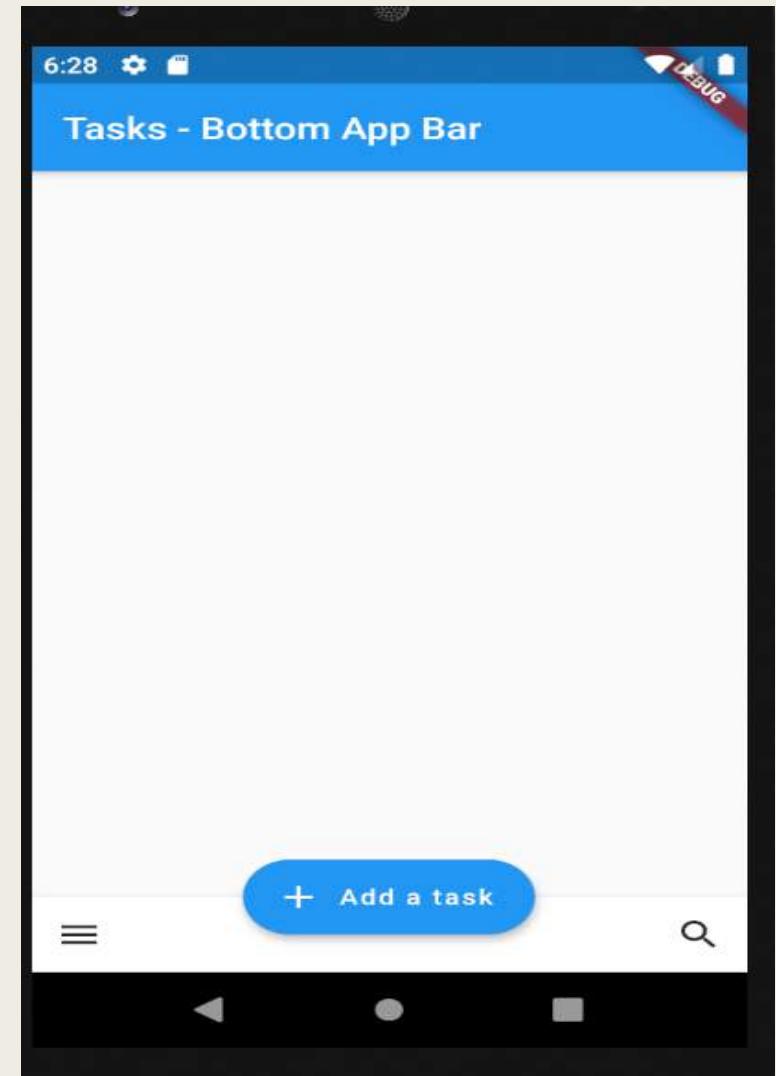
BottomApp Bar Versions(4)

```
bottomNavigationBar: BottomAppBar(  
    color: Colors.blueGrey,  
    shape: CircularNotchedRectangle(),  
    child: Container(  
        height: 50.0,  
    ),  
,  
floatingActionButton: Row(  
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
    children: <Widget>[  
        FloatingActionButton(  
            backgroundColor: Colors.blueGrey,  
            onPressed: () {},  
            child: Icon(Icons.account_circle),  
            heroTag: null,  
        ),  
        FloatingActionButton(  
            backgroundColor: Colors.blueGrey,  
            onPressed: () {},  
            child: Icon(Icons.account_circle),  
            heroTag: null,  
        ),  
        FloatingActionButton(  
            backgroundColor: Colors.blueGrey,  
            onPressed: () {},  
            child: Icon(Icons.access_alarm),  
            heroTag: null,  
        ),  
    ],  
,  
floatingActionButtonLocation:  
FloatingActionButtonLocation.centerFloat,
```



BottomApp Bar Versions(5)

```
@override  
Widget build(BuildContext context) {  
  return new Scaffold(  
    appBar: AppBar(title: const Text('Tasks - Bottom App Bar')),  
    floatingActionButton: FloatingActionButton.extended(  
      elevation: 4.0,  
      icon: const Icon(Icons.add),  
      label: const Text('Add a task'),  
      onPressed: () {},  
    ), // FloatingActionButton.extended  
    floatingActionButtonLocation:  
      FloatingActionButtonLocation.centerDocked,  
    bottomNavigationBar: BottomAppBar(  
      child: new Row(  
        mainAxisAlignment: MainAxisAlignment.spaceBetween,  
        mainAxisSize: MainAxisSize.max,  
        children: <Widget>[  
          IconButton(  
            icon: Icon(Icons.menu),  
            onPressed: () {},  
          ), // IconButton  
          IconButton(  
            icon: Icon(Icons.search),  
            onPressed: () {},  
          ), // IconButton  
        ], // <Widget>[]  
      ), // Row  
    ), // BottomAppBar  
  ); // Scaffold
```



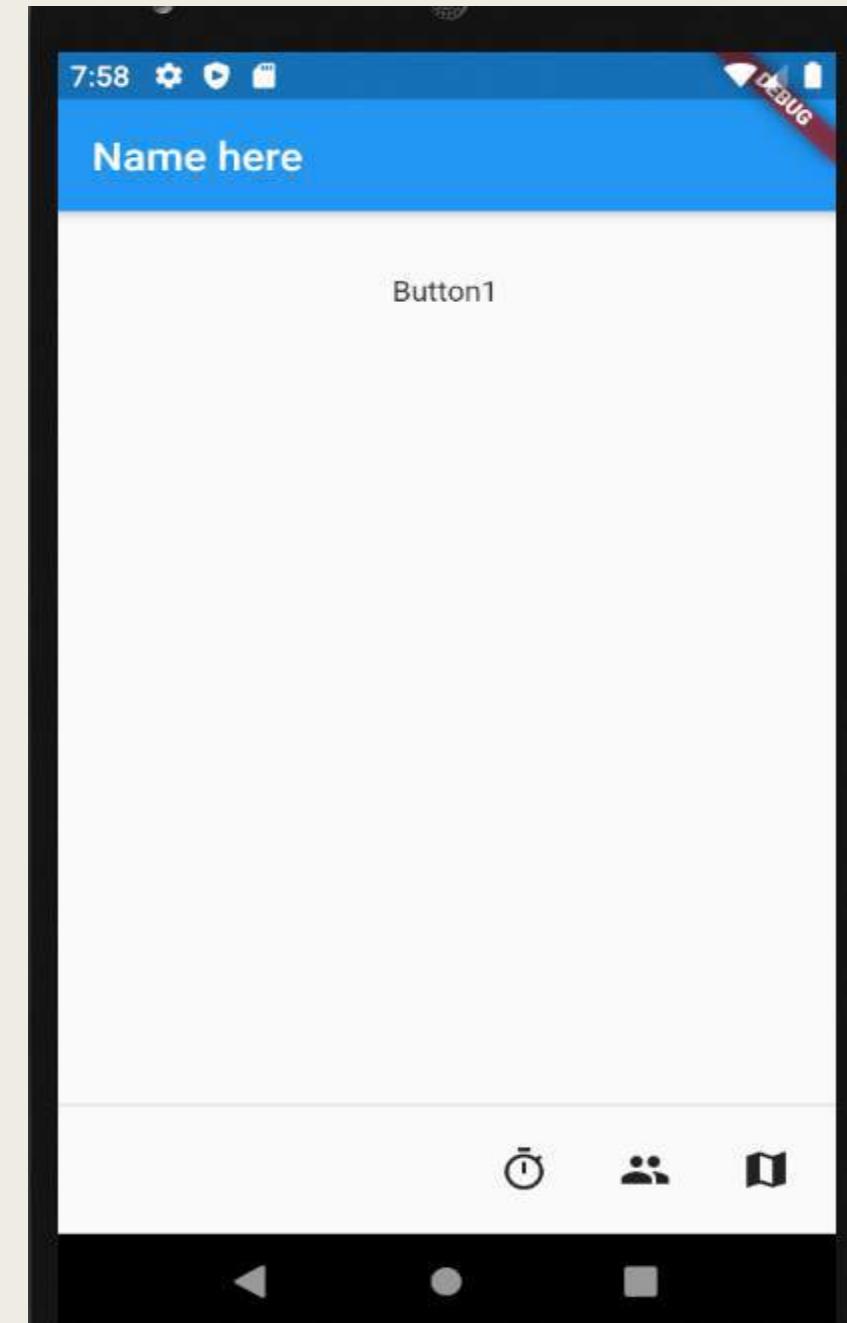
Footer Buttons

`persistentFooterButtons – List<Widget>`

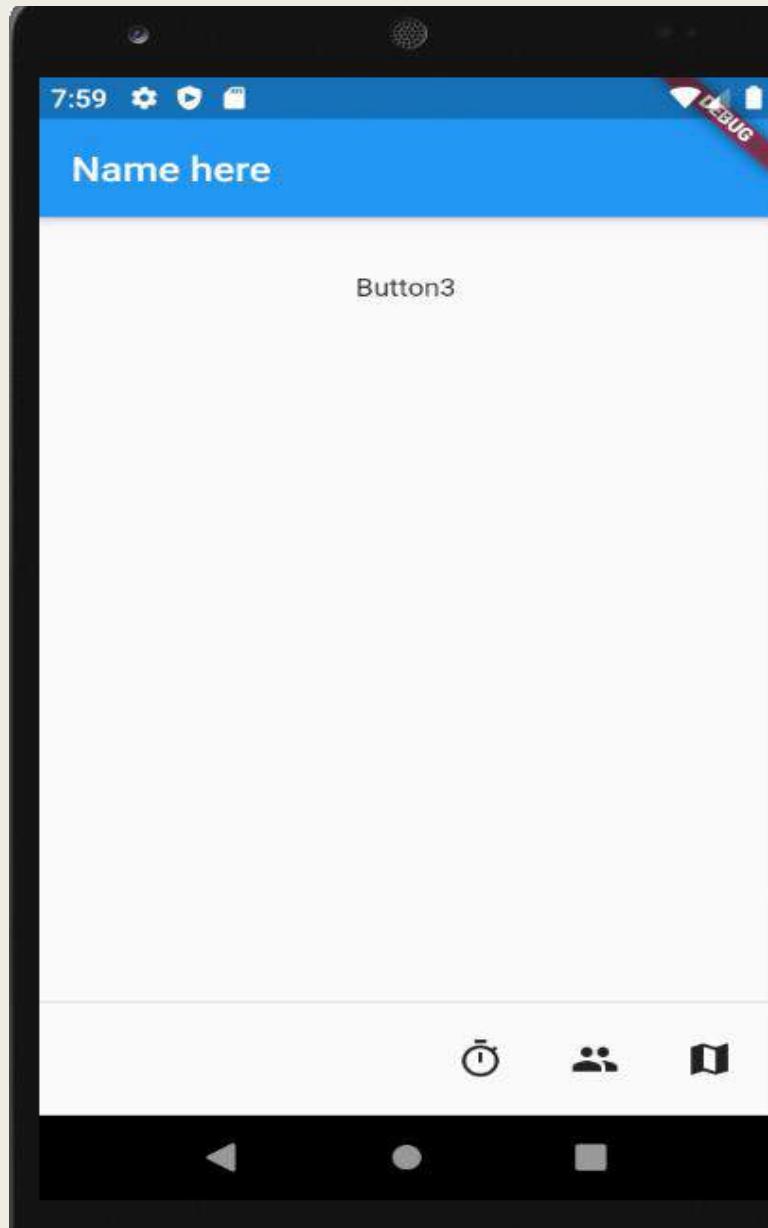
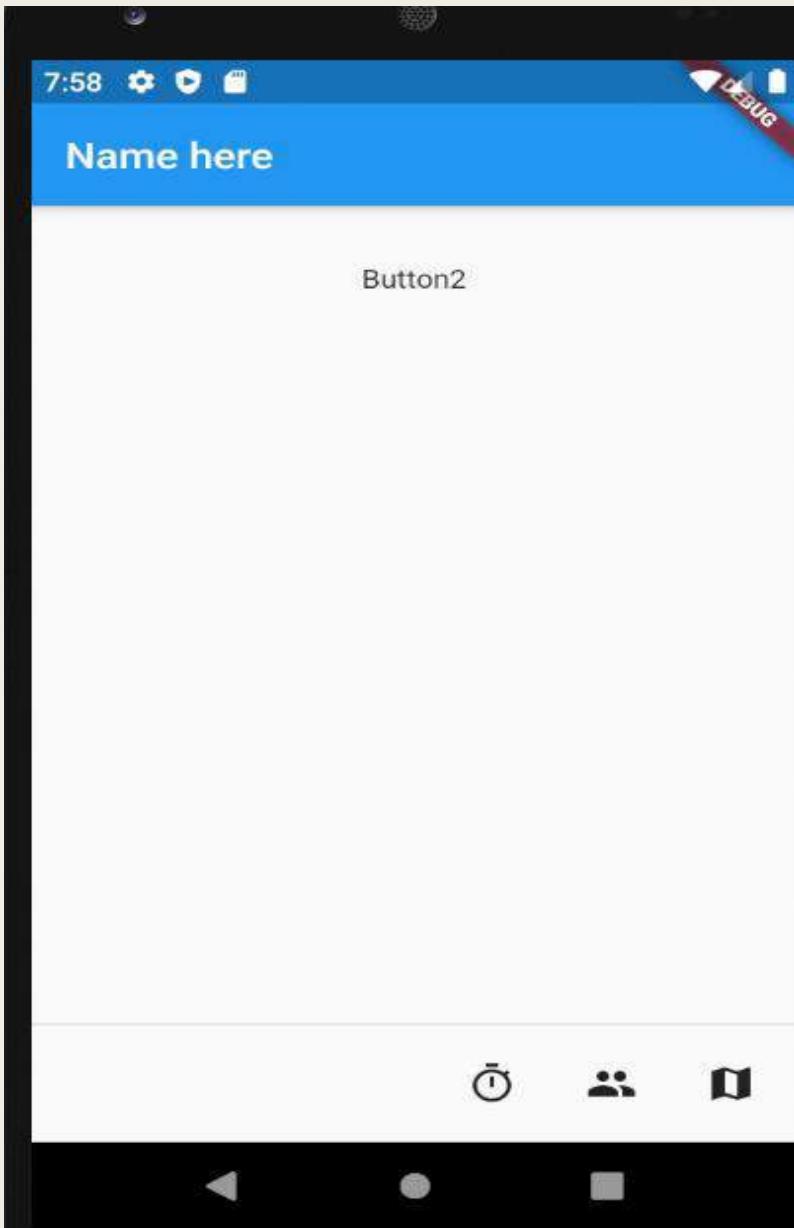
- A set of buttons that are displayed at the bottom of the scaffold. This is a collection of FloatingActionButtonButtons that are present in the bottom of the Application
- Displays a set of buttons which remains persistent on the screen, even if the body of Scaffold scrolls.

Footer Buttons

```
15 class _State extends State<MyApp>
16 {
17     String _value = '';
18     void _onClick(String value)=> setState(()=>_value=value);
19     @override
20     Widget build(BuildContext context){
21         return new Scaffold(
22             appBar:new AppBar(
23                 title: new Text('Name here'),
24             ), // AppBar
25             persistentFooterButtons:<Widget>[
26                 new IconButton(icon: new Icon(Icons.timer), onPressed:()=>_onClick('Button1')),
27                 new IconButton(icon: new Icon(Icons.people), onPressed:()=>_onClick('Button2')),
28                 new IconButton(icon: new Icon(Icons.map), onPressed:()=>_onClick('Button3')),
29
30
31             ], // <Widget>[]
32             body: new Container(
33                 padding: new EdgeInsets.all(32.0),
34                 child: new Center(
35                     child: new Column(children: <Widget>[
36                         new Text(_value),
37                     ], // <Widget>[]
38                 ),
39             ),
40         );
41     }
42 }
```



Footer Buttons(contd..)



Input Widgets

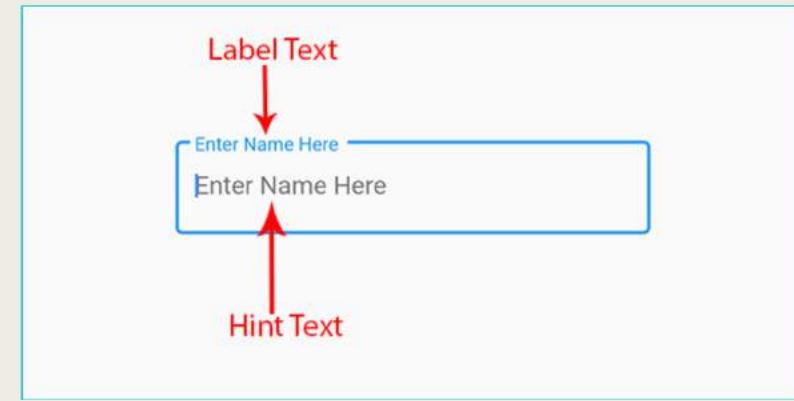
- TextField
- Checkbox
- Radios
- Switches
- Slider
- Date Picker (Self learning topic)

Text Field

A text field lets the user enter text, either with hardware keyboard or with an onscreen keyboard.

- **decoration** : To show decoration around Text Field.
- **border** : Create default rounded rectangle border around Text Field.
- **labelText** : To show label text on selection of Text Field.
- **hintText** : To show Hint text inside Text field.

```
1 Container(  
2   width: 300,  
3   child: TextField(  
4     decoration: InputDecoration(  
5       border: OutlineInputBorder(),  
6       labelText: 'Enter Name Here',  
7       hintText: 'Enter Name Here',  
8     ),  
9     autofocus: false,  
10    )  
11  )
```



TextField

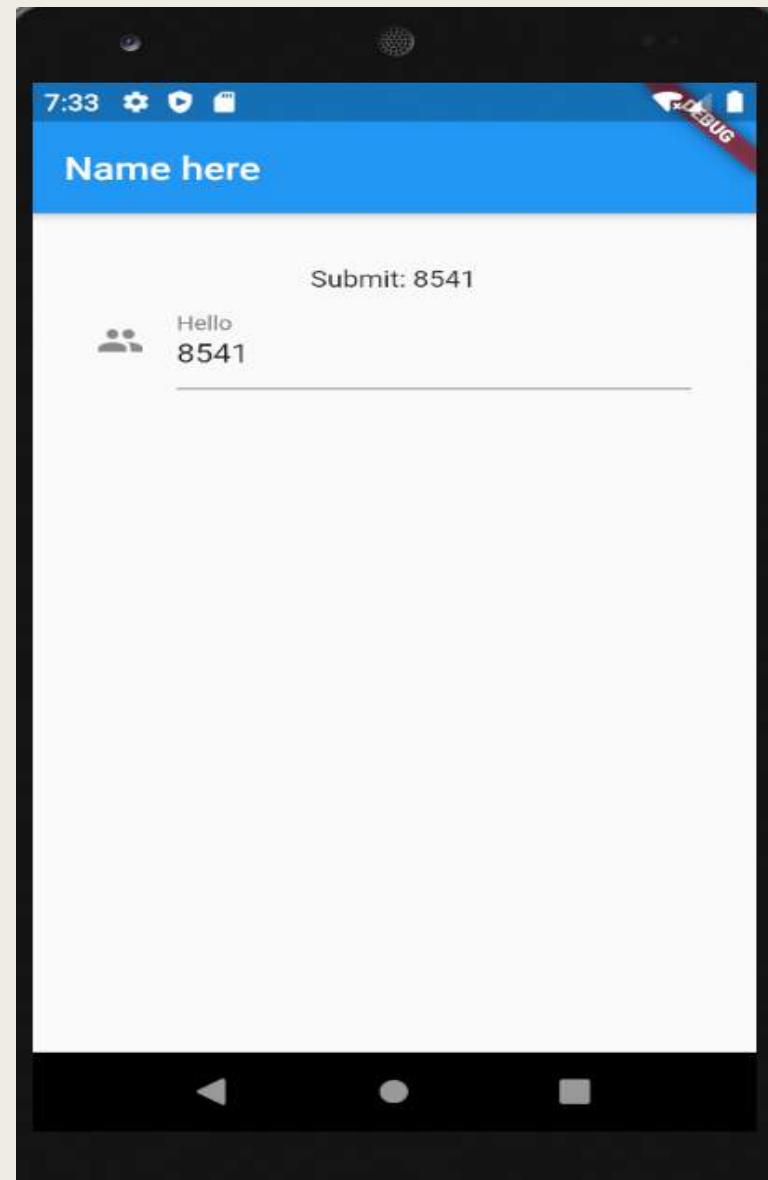
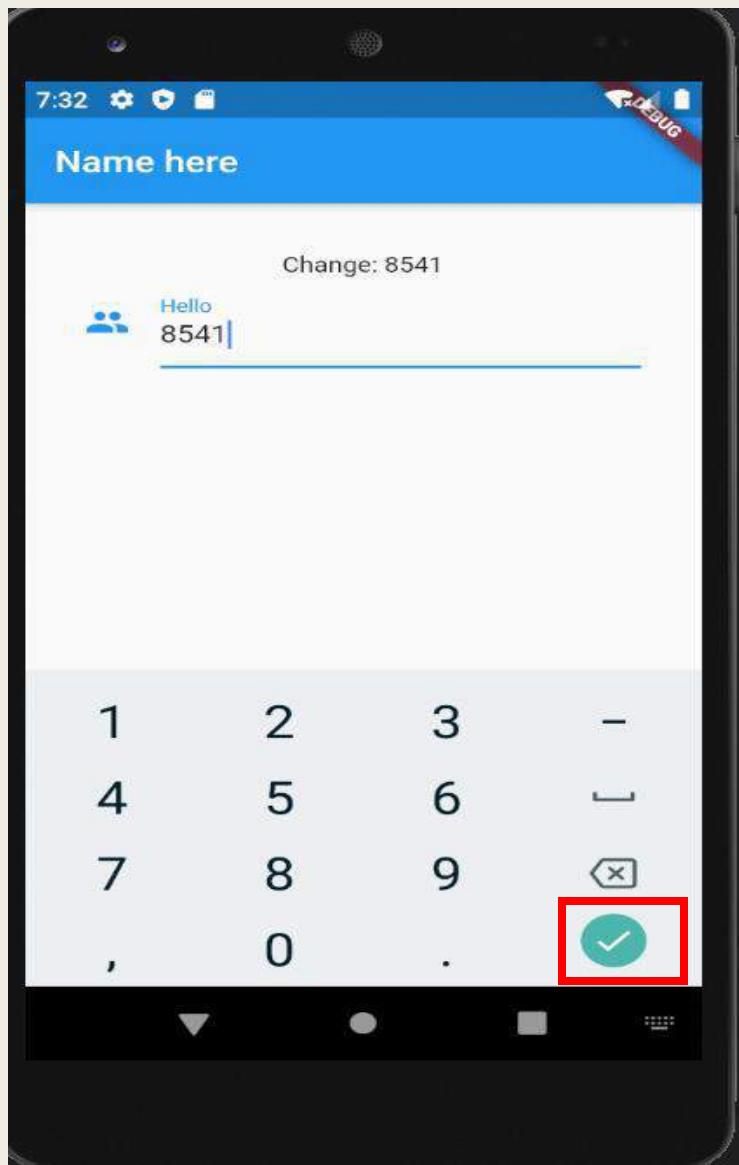
```
class _State extends State<MyApp>
{
  String _value = '';

  void _onChange(String value)
  {
    setState(() => _value='Change: ${value}');
  }

  void _onSubmit(String value)
  {
    setState(() => _value='Submit: ${value}');
  }
}
```

```
@override
Widget build(BuildContext context){
  return new Scaffold(
    appBar: new AppBar(
      title: new Text('Name here'),
    ), // AppBar
    body: new Container(
      padding: new EdgeInsets.all(32.0),
      child: new Center(
        child: new Column(children: <Widget>[
          new Text(_value),
          new TextField(
            decoration: new InputDecoration(
              labelText: 'Hello',
              hintText: 'Hint',
              icon: new Icon(Icons.people),
            ), // InputDecoration
            autocorrect: true,
            autofocus: true,
            keyboardType: TextInputType.number,
            onChanged:_onChange,
            onSubmitted: _onSubmit,
          ) // TextField
        ], // <Widget>[]
      ) // Column
    ), // Center
  ). // Container
```

TextField (contd..)



CheckBox Widget

For example, there is a state variable `_isChecked` which is passed as `value` property.

```
bool _isChecked = false;
```

Below is the basic example of how to create a `CheckboxListTile`.

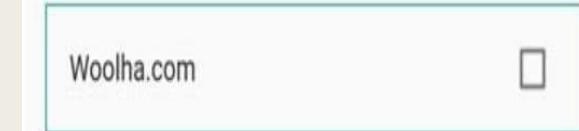
```
1 CheckboxListTile(  
2   value: _isChecked,  
3   onChanged: (bool value) {  
4     setState(() {  
5       _isChecked = value;  
6     });  
7   },  
8 )
```

Output:



```
1 Container(  
2   decoration: BoxDecoration(border: Border.all(color: Colors.teal)),  
3   child: CheckboxListTile(  
4     title: const Text('Woolha.com'),  
5     value: _isChecked,  
6     onChanged: (bool value) {  
7       setState(() {  
8         _isChecked = value;  
9       });  
10    },  
11  ),  
12 ),
```

Output:



CheckBox Widget(contd..)

```
1 Container(  
2   decoration: BoxDecoration(border: Border.all(color: Colors.teal)),  
3   child: CheckboxListTile(  
4     title: const Text('Woolha.com'),  
5     subtitle: const Text('A programming blog'),{1}  
6     value: _isChecked,  
7     onChanged: (bool value) {  
8       setState(() {  
9         _isChecked = value;  
10      });  
11    },  
12  ),  
13),
```

Output:

Woolha.com
A programming blog



```
1 Container(  
2   decoration: BoxDecoration(border: Border.all(color: Colors.teal)),  
3   child: CheckboxListTile(  
4     title: const Text('Woolha.com'),  
5     subtitle: const Text('A programming blog'),  
6     secondary: const Icon(Icons.web),{1}  
7     value: _isChecked,  
8     onChanged: (bool value) {  
9       setState(() {  
10         _isChecked = value;  
11      });  
12    },  
13  ),  
14),
```

Output:



Woolha.com
A programming blog

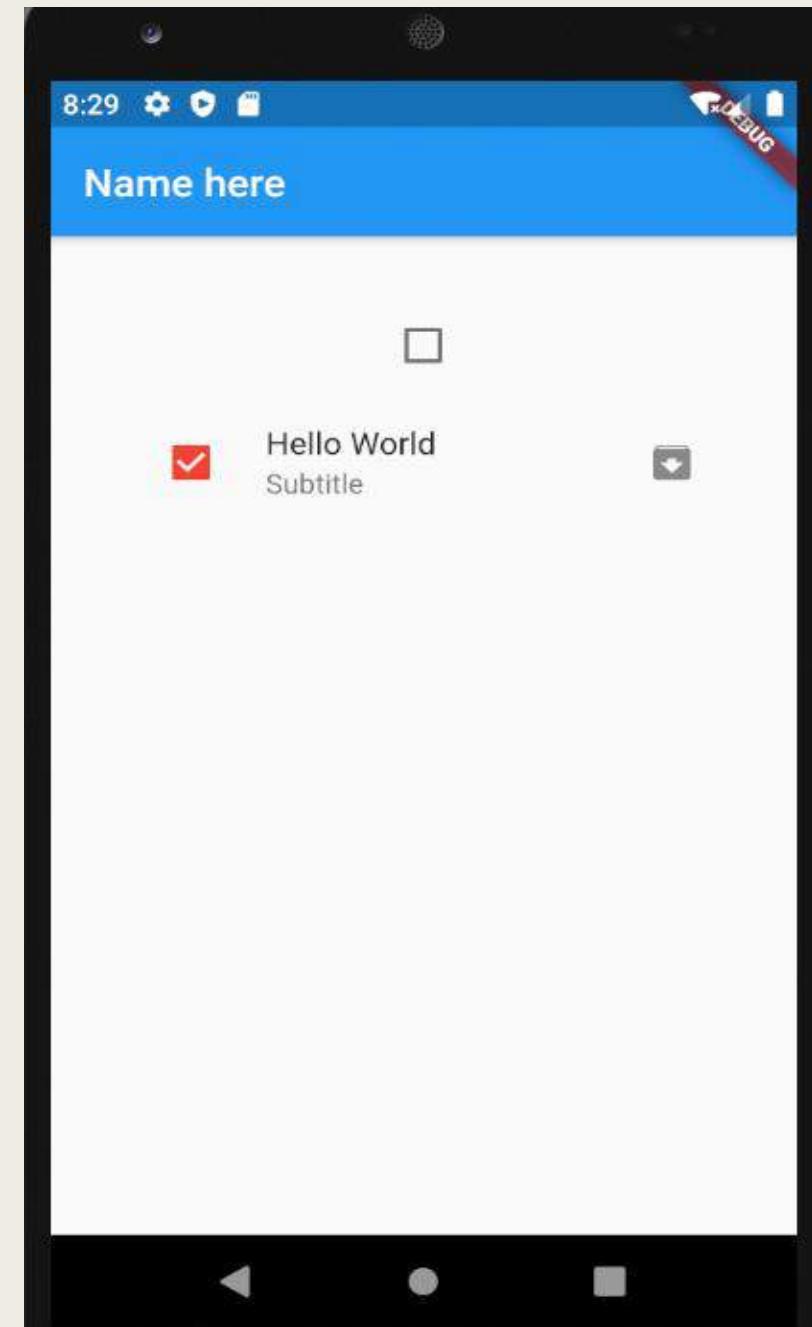


Checkbox

```
class _State extends State<MyApp>
{
  bool _value1=false;
  bool _value2=false;

  void _value1Changed(bool value) => setState(()=>_value1=value);
  void _value2Changed(bool value) => setState(()=>_value2=value);

  @override
  Widget build(BuildContext context){
    return new Scaffold(
      appBar:new AppBar(
        title: new Text('Name here'),
      ), // AppBar
      body: new Container(
        padding: new EdgeInsets.all(32.0),
        child: new Center(
          child: new Column(children: <Widget>[
            new Checkbox(value: _value1, onChanged: _value1Changed),
            new CheckboxListTile(value: _value2,
                onChanged: _value2Changed,
                title: new Text('Hello World'),
                controlAffinity: ListTileControlAffinity.leading,
                subtitle: new Text(['Subtitle']),
                secondary: new Icon(Icons.archive),
                activeColor: Colors.red,
              ), // CheckboxListTile
          ], // <Widget>[]
        ),
      ),
    );
}
```



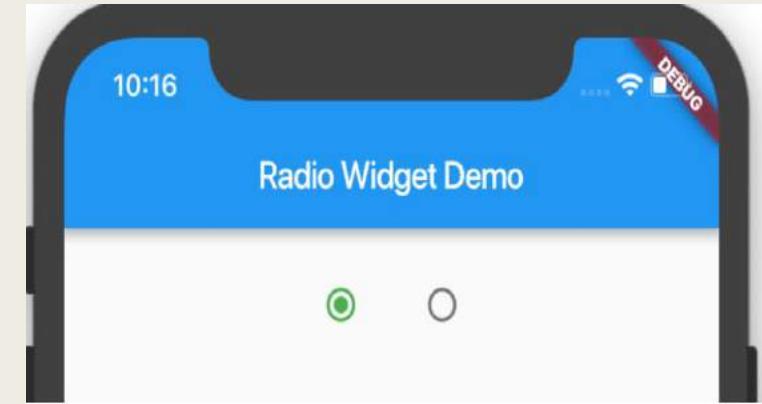
Radio Button

```
// Declare this variable
int selectedRadio;

@Override
void initState() {
    super.initState();
    selectedRadio = 0;
}

// Changes the selected value on 'onChanged' click on each radio button
setSelectedRadio(int val) {
    setState(() {
        selectedRadio = val;
    });
}

// This goes to the build method
ButtonBar(
    alignment: MainAxisAlignment.center,
    children: <Widget>[
        Radio(
            value: 1,
            groupValue: selectedRadio,
            activeColor: Colors.green,
            onChanged: (val) {
                print("Radio $val");
                setSelectedRadio(val);
            },
        ),
        Radio(
            value: 2,
            groupValue: selectedRadio,
            activeColor: Colors.blue,
            onChanged: (val) {
                print("Radio $val");
                setSelectedRadio(val);
            },
        ),
    ],
),
```



Radio Button

```
// Declare this variable
int selectedRadioTile;

@Override
void initState() {
    super.initState();
    selectedRadio = 0;
    selectedRadioTile = 0;
}

setSelectedRadioTile(int val) {
    setState(() {
        selectedRadioTile = val;
    });
}

// This goes to the build method
RadioListTile(
    value: 1,
    groupValue: selectedRadioTile,
    title: Text("Radio 1"),
    subtitle: Text("Radio 1 Subtitle"),
    onChanged: (val) {
        print("Radio Tile pressed $val");
        setSelectedRadioTile(val);
    },
    activeColor: Colors.red,
    secondary: OutlineButton(
        child: Text("Say Hi"),
        onPressed: () {
            print("Say Hello");
        },
    ),
    selected: true,
),
```

```
RadioListTile(
    value: 2,
    groupValue: selectedRadioTile,
    title: Text("Radio 2"),
    subtitle: Text("Radio 2 Subtitle"),
    onChanged: (val) {
        print("Radio Tile pressed $val");
        setSelectedRadioTile(val);
    },
    activeColor: Colors.red,
    secondary: OutlineButton(
        child: Text("Say Hi"),
        onPressed: () {
            print("Say Hello");
        },
    ),
    selected: false,
),
```



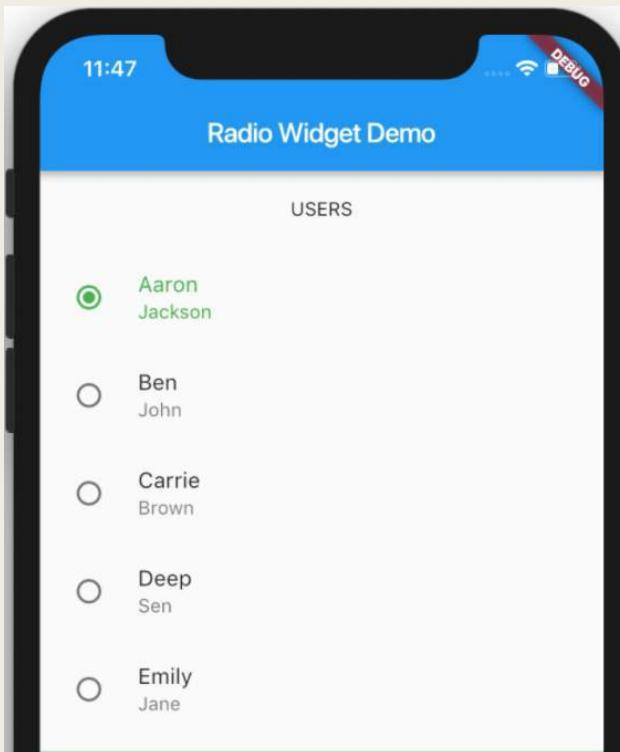
Radio Button

user.dart

```
class User {  
    int userId;  
    String firstName;  
    String lastName;  
  
    User({this.userId, this.firstName, this.lastName});  
  
    static List<User> getUsers() {  
        return <User>[  
            User(userId: 1, firstName: "Aaron", lastName: "Jackson"),  
            User(userId: 2, firstName: "Ben", lastName: "John"),  
            User(userId: 3, firstName: "Carrie", lastName: "Brown"),  
            User(userId: 4, firstName: "Deep", lastName: "Sen"),  
            User(userId: 5, firstName: "Emily", lastName: "Jane"),  
        ];  
    }  
}
```

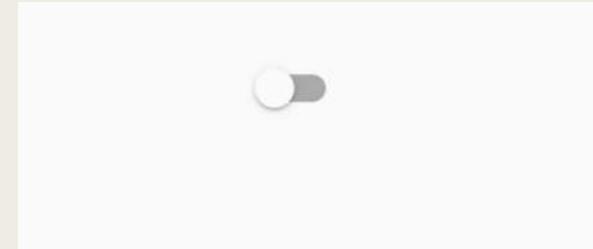
```
List<User> users;  
  
@override  
void initState() {  
    super.initState();  
    users = User.getUsers();  
}  
  
setSelectedUser(User user) {  
    setState(() {  
        selectedUser = user;  
    });  
}  
  
List<Widget> createRadioListUsers() {  
    List<Widget> widgets = [];  
    for (User user in users) {  
        widgets.add(  
            RadioListTile(  
                value: user,  
                groupValue: selectedUser,  
                title: Text(user.firstName),  
                subtitle: Text(user.lastName),  
                onChanged: (currentUser) {  
                    print("Current User ${currentUser.firstName}");  
                    setSelectedUser(currentUser);  
                },  
                selected: selectedUser == user,  
                activeColor: Colors.green,  
            ),  
        );  
    }  
    return widgets;  
}  
  
// In the build method  
Column(  
    children: createRadioListUsers(),  
,  
....
```

Radio Button



Switch Widget

- Flutter **Switch** is used to **toggle** a setting between on/off which is true/false respectively. When the **switch** is on, the value returned by the **Switch onChanged** property is true, while the **switch** is off, **onChanged** property returns false.

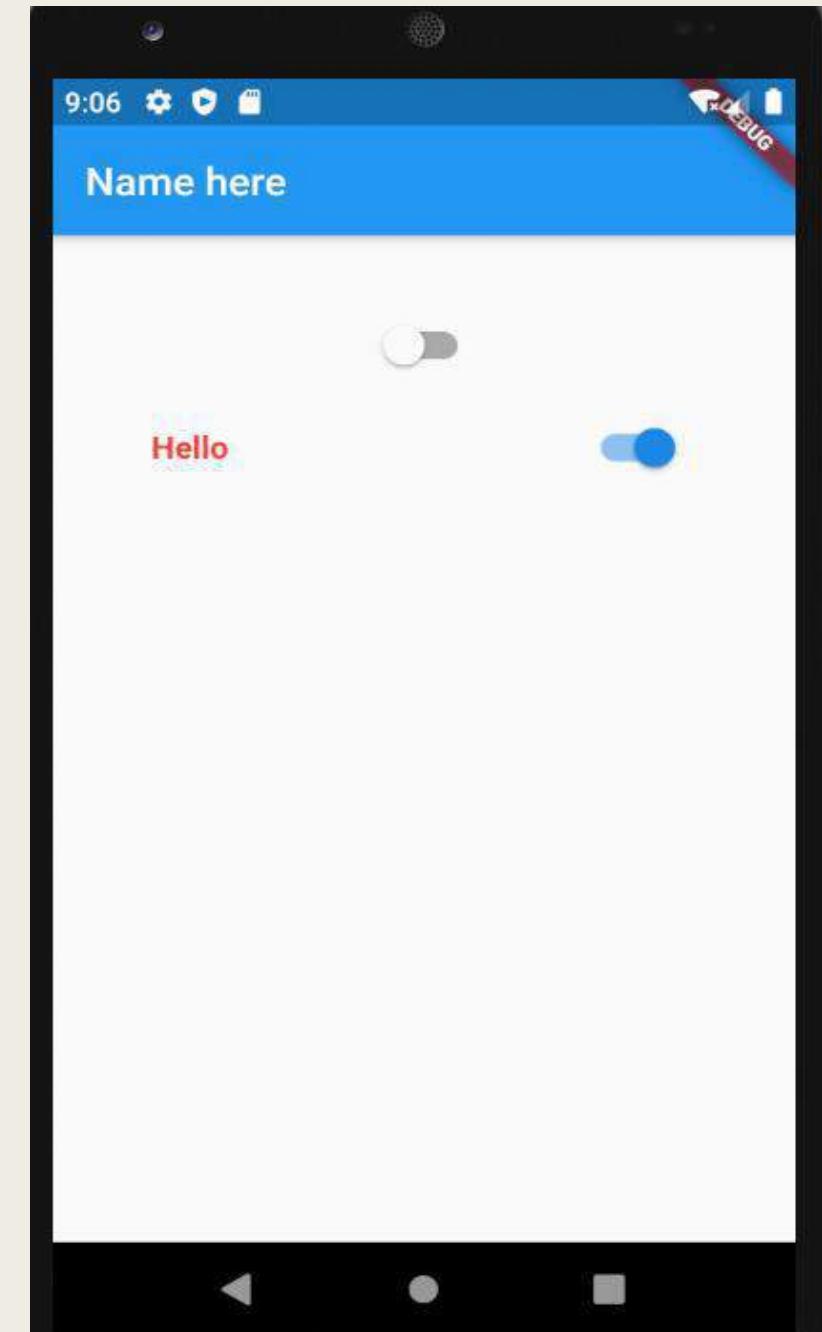


Switches

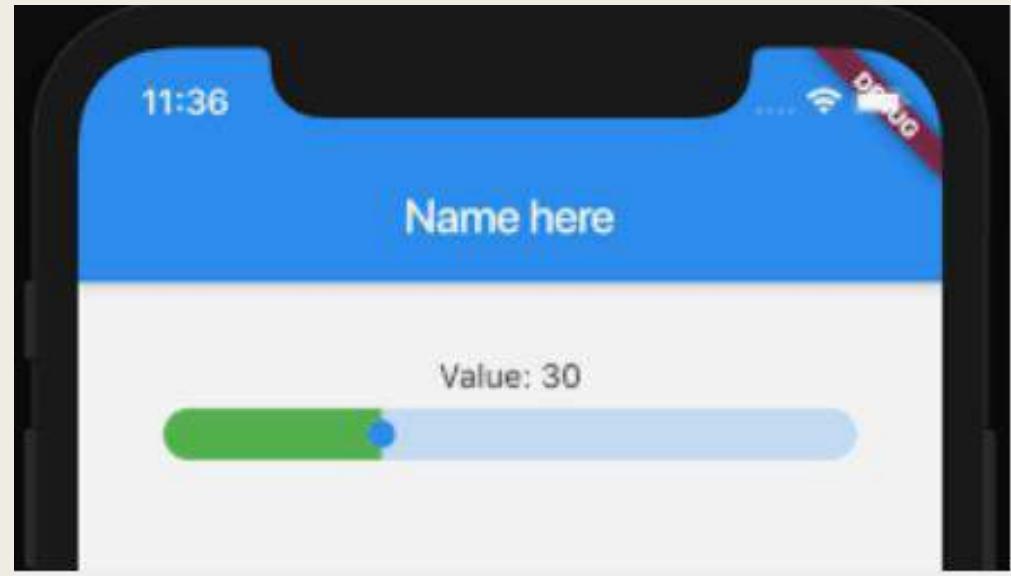
```
class _State extends State<MyApp>
{
  bool _value1 = false;
  bool _value2 = false;

  void _onChanged1(bool value)=> setState(()=> _value1 = value);
  void _onChanged2(bool value)=> setState(()=> _value2 = value);

  @override
  Widget build(BuildContext context){
    return new Scaffold(
      appBar:new AppBar(
        title: new Text('Name here'),
      ), // AppBar
      body: new Container(
        padding: new EdgeInsets.all(32.0),
        child: new Center(
          child: new Column(children: <Widget>[
            new Switch(value: _value1, onChanged:_onChanged1),
            new SwitchListTile(
              value: _value2,
              onChanged:_onChanged2,
              title: new Text('Hello',
                style:new TextStyle(
                  fontWeight: FontWeight.bold,
                  color: Colors.red
                )), // TextStyle // Text
            ) // SwitchListTile
          ],
        ),
      ),
    );
  }
}
```

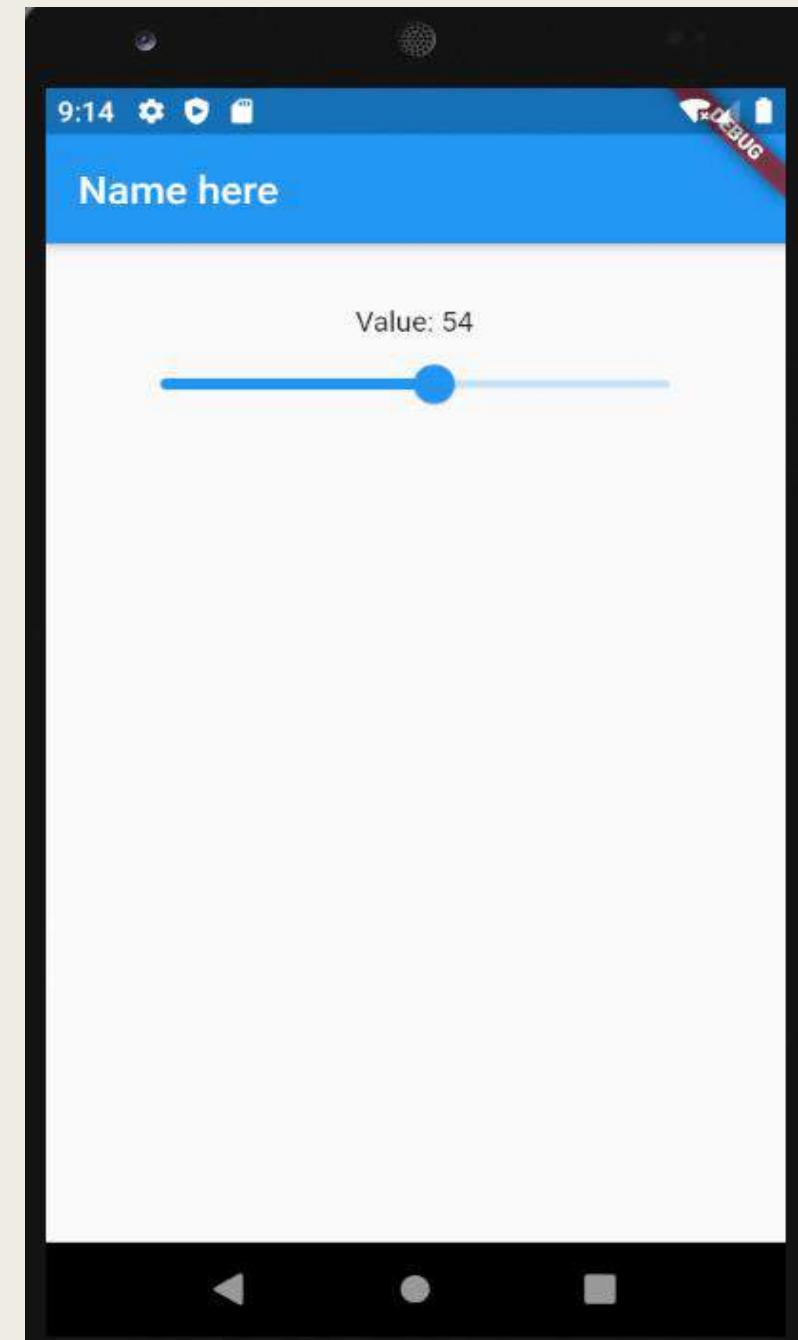


Slider Widget



Slider

```
class MyApp extends StatefulWidget {  
  @override  
  _State createState() => new _State();  
  
}  
  
class _State extends State<MyApp>  
{  
  
  double _value = 0.0;  
  void _setValue(double value) => setState(() => _value = value);  
  
  @override  
  Widget build(BuildContext context){  
    return new Scaffold(  
      appBar: new AppBar(  
        title: new Text('Name here'),  
      ), // AppBar  
      body: new Container(  
        padding: new EdgeInsets.all(32.0),  
        child: new Center(  
          child: new Column(children: <Widget>[]  
            new Text('Value: ${(_value*100).round()}'),  
            new Slider(value: _value, onChanged: _setValue),  
          ], // <Widget>[]  
        ) // Column  
    );  
}
```



Customized Slider

```
SliderTheme(  
    data: SliderTheme.of(context).copyWith(  
        activeTrackColor: Colors.red,  
        inactiveTrackColor: Colors.black,  
        trackHeight: 3.0,  
        thumbColor: Colors.yellow,  
        thumbShape: RoundSliderThumbShape(enabledThumbRadius: 8.0),  
        overlayColor: Colors.purple.withOpacity(0.32),  
        overlayShape: RoundSliderOverlayShape(overlayRadius: 14.0),  
    ),  
    child: slider(  
        value: _value,  
        onChanged: (value) {  
            setState(() {  
                _value = value;  
            });  
        },  
    ),  
,
```

