Chapter-2:

Code: 2.1

//class

class Student {

//variable

String name;

int age;

//private variable

int \_score;

//constructor

Student(this.name, this.age, this.\_score);

//getter

int getScore() {

return \_score;

}

//setter

void setScore(int newScore) {

\_score = newScore;

}

}

Code: 2.2

// class

class Employee {

var name = 'Tom';

var designataion = 'Developer';

}

void main() {

// Object of Employee

var employee1 = Employee();

print('${employee1.name}, ${employee1.designataion}');

// Object of Employee

var employee2 = Employee();

employee2.name = 'Arnav';

employee2.designataion = 'Team Lead';

print('${employee2.name}, ${employee2.designataion}');

}

Code: 2.3:

class Human {

void walk() {

print('Humans walk!');

}

}

// Inheriting the parent class i.e. Human

class Man extends Human {

void speak() {

print('That man can speak');

}

}

void main() {

var man = Man();

man.speak();

man.walk(); // invoking the parent class method

}

Code: 2.4:

class Employee {

void salary() {

print("Employee salary is £3000.");

}

}

class Manager extends Employee {

@override

void salary() {

print("Manager salary is £4000.");

}

}

class Developer extends Employee {

@override

void salary() {

print("Developer salary is £5000.");

}

}

void main() {

Manager manager = Manager();

Developer developer = Developer();

manager.salary();

developer.salary();

}

Code 2.5:

//class Example1 (Interface1)

class Example1 {

void printdata1() {

print("Hello Example1!!");

}

}

//class Example2 (Interface2)

class Example2 {

void printdata2() {

print("Hello Example2!!");

}

}

//class Example implementing Example1, Example2.

class Example implements Example1, Example2 {

@override

void printdata1() {

print("Hello Example1, welcome!");

}

@override

void printdata2() {

print("Hello Example2, welcome!");

}

}

void main() {

// Creating object of the class Example

var example = Example();

// Calling method (After Implemetation)

example.printdata1();

example.printdata2();

}

Code: 2.6:

//abstract class

abstract class Language {

//abstract methods

void write();

void read();

//concrete method

void understand() {

print('I can understand');

}

}

class English extends Language {

@override

void write() {

print('I can write English!');

}

@override

void read() {

print('I can read English!');

}

}

void main() {

//object of English class

var english = English();

//Accessing property using object

english.write();

english.read();

}

Code 2.7:

class YearGroup {

//public property

var students = ['Arnav', 'Vansh', 'Lily'];

//private property

var \_teacher = 'Mr. Oscar';

//final property

final String topper = 'Freya';

}

void main() {

//object of YearGroup class

var year1 = YearGroup();

year1.\_teacher = 'Mr Sanat';

//Accessing property using object

print(year1.\_teacher);

print(year1.topper);

}

Code: 2.8:

//creating a Laugh Mixin

mixin Laugh {

void laugh() => print("Laughing");

}

//creating a Talk Mixin

mixin Talk {

void talk() => print('Talking');

}

//creating super class

class Human {

void think() => print('Thinking');

}

//creating sub class

class Boy extends Human with Laugh, Talk {

@override

void think() => print('No more thinking');

@override

void laugh() => print('Smiling');

}

void main() {

var boy = Boy();

boy.laugh();

boy.talk();

boy.think();

}

Code: 2.9:

// Creating enum UserState

enum UserState {

//Inserting data

authorised,

pending,

rejected

}

void main() {

// Printing the values present in the UserState

for (UserState state in UserState.values) {

print(state);

}

}

Code 2.10:

// Creating enum UserState

enum UserState {

//Inserting data

authorised,

pending,

rejected

}

void main() {

var state = UserState.authorised;

//switch case

switch (state) {

case UserState.authorised:

print("This is an authorsed user.");

break;

case UserState.pending:

print("This is an authorsed user.");

break;

case UserState.rejected:

print("This is an authorsed user.");

break;

}

}

Code: 2.11:

//This generic function prints and returns the itemat the required index of a list

T getElement<T>(List<T> genericList, int index) {

T element = genericList[index];

print(element);

return element;

}

void main() {

//Lists of three different datat types declared

List<int> ageList = [9, 3, 16];

List<double> heightList = [4.4, 3.6, 5.8];

List<String> nameList = ['Arnav', 'Oscar', 'Sanat'];

getElement(ageList, ageList.length - 1);

getElement(heightList, heightList.length - 1);

getElement(nameList, nameList.length - 1);

}

Code: 2.12:

void main() {

//Assigning an anonymous function to a variable

var addNumbers = (int x, int y) {

return x + y;

};

//Calling the anonymous function using the variable

var result = addNumbers(3, 4);

print(result);

//Passing an anonymous function as an argument to another function

var multiplyNumbers = (int x, int y) {

return x \* y;

};

doCalculation(5, 6, multiplyNumbers);

}

void doCalculation(int x, int y, Function operation) {

var result = operation(x, y);

print(result);

}

Code 2.13:

void main() {

for (int i = 0; i < 5; i++) {

print('i is $i');

}

}

Code 2.14:

void main() {

List<String> namesList = ['Arnav', 'Oscar', 'Sanat'];

for (String name in namesList) {

print(name);

}

}

Code: 2.15:

void main() {

var count = 4;

int i = 1;

while (i <= count) {

print('Hello Ipsi');

i++;

}

}

Code:2.16:

void main() {

var count = 4;

int i = 1;

do {

print('Hello Ipsi!');

i++;

} while (i <= count);

}

Code: 2.17:

import 'package:flutter/material.dart';  
  
class Count extends StatelessWidget {  
 final int count;  
 final VoidCallback onCountSelected;  
  
 const Count({  
 super.key,  
 required this.count,  
 required this.onCountSelected,  
 });  
  
 @override  
 Widget build(BuildContext context) {  
 return MaterialButton(  
 child: Text("$count"),  
 onPressed: () => onCountSelected(),  
 );  
 }  
}

import 'package:flutter/material.dart';  
import 'Count.dart';  
  
class CounterPage extends StatefulWidget {  
 @override  
 State<CounterPage> createState() => \_CounterPageState();  
}  
  
class \_CounterPageState extends State<CounterPage> {  
 int count = 0;  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: Text("Widget Communication")),  
 body: Center(  
 child: Count(  
 count: count,  
 onCountSelected: () {  
 print("Count was Selected");  
 },  
 ),  
 ),  
 );  
 }  
}

code: 2.18:

import 'package:flutter/material.dart';

class Count extends StatelessWidget {

final int count;

final VoidCallback onCountSelected;

final Function(int) onCountChanged;

const Count({

super.key,

required this.count,

required this.onCountChanged,

required this.onCountSelected,

});

@override

Widget build(BuildContext context) {

return Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

IconButton(

onPressed: () {

onCountChanged(1);

},

icon: const Icon(Icons.add)),

TextButton(onPressed: () => onCountSelected(), child: Text("$count")),

IconButton(

onPressed: () {

onCountChanged(-1);

},

icon: const Icon(Icons.remove)),

],

);

}

}

import 'package:flutter/material.dart';

import 'Count.dart';

class CounterPage extends StatefulWidget {

const CounterPage({super.key});

@override

State<CounterPage> createState() => \_CounterPageState();

}

class \_CounterPageState extends State<CounterPage> {

int count = 0;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: const Text("Widget Communication")),

body: Center(

child: Count(

count: count,

onCountSelected: () {

print("Count was Selected");

},

onCountChanged: (int value) {

setState(() {

count += value;

});

},

),

),

);

}

}

Code: 2.19:

void main() {

addNumbers(15, 20);

}

void addNumbers(int i, int j) {

var result = i + j;

print('The result of addition is $result');

}

Code: 2.20:

void main() {

addNumbers(15, 20);

}

void addNumbers(int i, int j) => print('The result of addition is ${i + j}');

code 2.21:

void main() {

var age = 20;

age >= 18 ? print('Eligible') : print('Not Eligible');

}

Code: 2.22:

void main() {

dynamic age;

print(age);

age ?? print('age is null');

}

code: 2.23:

// This example shows how \*not\* to write asynchronous Dart code.

String getEmoployeeDetails() {

var employee = fetchEmployeeName();

return 'Employee name is: $employee';

}

Future<String> fetchEmployeeName() =>

//Imagine that this function is more complex and slow.ArgumentError

Future.delayed(

const Duration(seconds: 2),

() => 'Tapsi',

);

void main() {

print(getEmoployeeDetails());

}

Code: 2.24:

Future<String> getEmoployeeDetails() async {

var employee = await fetchEmployeeName();

return 'Employee name is: $employee';

}

Future<String> fetchEmployeeName() =>

//Imagine that this function is more complex and slow.ArgumentError

Future.delayed(

const Duration(seconds: 2),

() => 'Tapsi',

);

Future<void> main() async {

print(await getEmoployeeDetails());

}