Flutter Projects files & Folders :-

.idea :- projects related settings

✓ android - android specific code / settings (for android platforms)

✓ web -

✓ ios - ios specific code (specific changes for iphone)

build - o/p generated when we run the flutter projects

lib - "library" main folder where flutter app related codes.
Contains 'main.dart' file.

test - testing related we will write here.

.gitignore - adjust the .gitignore file to include or exclude any files as our need. in remote repository.

.metadata - project related metadata.

.package - predefined libraries.

.pubspec.yaml - font, images changes but firstly lock by

.pubspec.lock - font, images lock to next changes

README.md - what will in the overall project. ("git repo")

Dart

= Launch in 2011. Developed by Google. (2.0) 2018

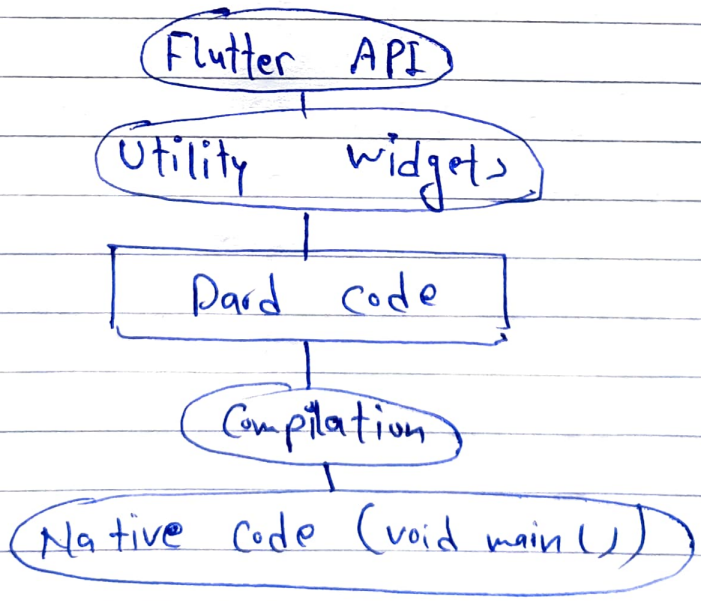
= Focused on frontend.

= object Oriented & strongly typed.

= Mixture of Java, JavaScript.

= Use of 'async' & 'await' for asynchronous programing.

= Compiled → AOT (ahead of time)
→ JIT (just in time)
for hot reload.



= Dart file always write in small letter.

Important :-

1) Dart code syntax :-

```
void main() {
  print('Hello world');
  print("Hello world");
  print(44/5);          // 8.8
}
```

2) User I/O :- 'string'

```
import 'dart:io';
void main() {
  print('Enter');          // stdout.write('Enter');
  var name = stdin.readLineSync();
  stdout.write('welcome, $name');
}
```

3) Printing 2 way :

① print('');

② stdout.write('');

★ integer input :-

can't be null value.

use for null safty

```
int? n = int.parse(stdin.readLineSync()!);
var
```

call to compiler its not null.

print(a); ✓
print('$a'); ✓

print('$a $b');
print(a b); ✗

① ⇒ Data type :- 'var' is universal ... function ✗
dynamic is " " ... function ✓

- Number (int, double, num), BigInt
- String (String)
- Boolean (Bool)
- Lists (List) > Dynamic
- Maps (Map)
- Runes "Unicode characters in a String"

✓ all data types in Dart are objects so default is null.

✓ Ex: var isValid = true // automatically change
(or)
bool isValid = true

✓ Ex: int a = 5;
var b = 5; // automatically change int
var q = num.parse("5"); // string to int
num q = 5

print("sum is ${a+b}"); // 10

✓ Ex: String st = "1012";
var st = 'Pawan';

String st = "'This is it'";

✓ Ex: List name = ["Han", "Rohan", "Kapil", 17, 1];
var name = [2, 21, "The", false];
✗ var List<int> name = [2, 8, 3];

✓ Ex: Map name = {2: "Pawan", 8: 'Han',};
var name = {'p': true, true: "82"};
✓ Map<int, String = {1: "true", 8: 'Pawan'};
✗ Var

Important :-

① print("My name is " + name); ✗
print("My name is $name"); ✓

② print("sum is " + (a+b).toString()); ✗
print("sum is ${a+b}") ✓

③ print("length of string ${st.length}");

④ print("$"); ✗

⑤ print(8+2); //10 ✓
print("P" + "awan"); //Pawan ✓

⑥ var a = 10; "change into integer" but var a; bcz
a = 10.2; //error ✗ a = 10; now
a = 10.11; it's univer
a = true; sal
dynamic a = 10; ✓ (dynamic)
a = 'Pawan'; ✓ } change in all types
a = true; ✓

⑦ int a = 5;
double b = a; ✗
double b = a.toDouble(); ✓

⑧ null → Nothing no value
Null → Type Eg: var x;
print(x.runtimeType);

(9) int fun(int a) {
return a*a;
}

(or) int fun(int a) => a*a;

(10) int a; "can't be null"
int? a; "can be null"

(11) BigInt a; // for long integer
a = BigInt.parse('28797677896569'); // take it as string

(12) int a = 10; → public
variable: int _a = 10; → private
funct°: void _fun() { } → //

(13) var a = x / b; // fractional
var a = x ~/ b; // integer

(14) Not function Overloading supports on same class.
Ex: void fun(){}
void fun(int x) { } // error

(15) class Demo {
}

(or) class Demo extends Object{
}

(16) we can write function outside the main function either class' and 'simply outside main'.

(2) # 'Final' & 'const'

⇒ "both never change a value during the execution"

⇒ Syntax: final name = "Pari"; (or) final String name = "Pari";
const PI = 3.14; (or) const double PI = 3.14;

final	const
• final variable can only be set once when it's initialized.	• implicitly final but initialize during compilation.
• Instance variable can be final.	• Can't const at class level, but make it 'static const'.

Ex: class Demo {
final color = 'red';
}
	Ex: class Demo {
	static const color = 'red';
	const color = 'red'; ✗
	}

• int a = 10; final b = a; ✓	• int a = 10; const b = a; ✗ const b = 10; ✓
• final var name; ✗ (var can't use)	
List ⇒ • final a = [1, 2, 3]; a.add(4); ✓	• const a = [1, 2, 3]; a.add(4); ✗ can't changes at compile time

.runtimeType();

(3) Casting :-

- => string to int : var a = int.parse('10');
  a += 10; // 20
- => int to string : var a = 10.toString();
  a += 'pawan'; // 10pawan

(4) Operator :-

- (Arithmetic) +, -, *, %, / (OR) ~/
  - / → Fractional
  - ~/ → integer
- (Assignment) =, +=, %=
- (Relational) >, <, <=, ==, !=
- as | is | is! Eg: int a = 5 (Type test)
  print (a is int); // true
  double b = a as double // convertion
- ++, --
- &&, ||, ! true
- ?: (condition ? ex1 : ex2;)
- &, |, ~, ^, >>, << (bitwise) (0,1)
  - << → Left shift
- .. (cascade function) Eg: obj..fun1()..fun2();
  (OR)
  obj.fun1();
  obj.fun2();

- ... (Spared operator) Ex: List li2 = [12, 13, ...li1];
- ?? (Default op.) Ex: int a;
  int b = 5;
  var p = a??b; // 5 = o/p
  " if a is null than p = 5
  otherwise p value same as a value. "
- ??= (fall back assig.) Ex = int a;
  a ??= 3; //op → 3
  " if a is null than a = 3
  otherwise assign value. "
- => (fat arrow) Eg:- int getArea (int a) => a*a;
  void area() => print(a);
  " It's use instant of { } for function. "

(5) Loop :-

(+) for Loop:

Ex: for (int i=0; i<a.length; i++) { }

var a = [2, 4, 'P', true];

- for-in loop: Ex: for (var x in a)
  print(x);
  }
- for-each loop: Ex: var a = [2, 12, 'pawan' true];
  a.forEach((x) { //'(x)' anony-
  print(x); my fun
  });
  (OR)
  a.forEach((x) => print(x));

- return type is not mandatory

⑥ function : - "written reusable code"

Requested

```
void function (int a) {
  print ($a);
}
```

(or)

```
void funct (int a) =>
    print ($a);
```

→ no need to write

```
int fun (int a) {
  return a*a;
}

void main() {
print ( ${fun (3)});
}
```

(or)

```
int fun(int a) => a*a;

void main () {
print ( ${fun(3)});
}
```

Optional Parameter :-

① optional parameter :- [ ]

Eg:- `void fun ( string a, [int? b=10, bool? c ]){ }`

`fun ('Pan');` (or) `fun('Pan', 2);`

② named parameter :- { } (order doesn't matter)

Eg: `void fun ( int a, {int? b, string? c})`

```
{ ——
}

fun( 10, c: 'pawan', b: 5);   // order no
fun ( 10, b: 20);                 metter
```

Default parameter: Ex: `int fun ( int a, {int b=8})`

```
{ ——
}

fun( 2); or fun( 2, b: 3);   // 16
```

overide

Functional Programming :-

anonymous / Lambda function :-

Syntax: `(parameter_list) {` // use in forEach

`};`

Eg:- `void main ( ) {`

Function `var x = (int a) {` (or)

`print ( $ a );` // also return

`};`

`x ( 3);` // Call

`}`

Return 'function as parameter' :- (higher order function)

```
void main () {
    fun ('HEL', x );
}

Function x = (string s) {
        retun s;
      };

void fun ( string a, Function b) {
    print ('$a, ${b('oh')}' );
}
```

o Store fun, another function :-

```
void main() {
  var x = marks();
  print(x(10, 20, 30));
}

Function marks() {
    Function n = (int a, int b, int b){
      return a+b+c;
      };
return n;
}
```

Collection :- List, Map, set

(1) List :- "array" (ordered)

```
var n = [1, 2, true, 2];
List n = [1, 1, 'Pan'];

var m = [1, 1, 'Pan', ... n];
var<int> m = [1, 2, 2, 3];      // only integer
```

(generic)

- store duplicates
- index based values (index starts → 0)
- 2 types → i) Fixed-length list
  ii) Variable-length list

i) Fixed-length :

```
Syntax  List<int> val = const [8, 2, 13, 4];
                      = List(5);
                      = List.filled(4, 6);
val[0] = 55;
val[1] = 21;
val[2] = 9;
val[4] = 51;
print(val[1]);   // 21
```

```
obj.remove(value);
obj.insert(value);
// .add(value);
// .clear();
```

(Can't use in fixed-length bcz variable size function)

ii) Growable list :

```
List val = new List();
Syntax: List<int> val = [1, 12, 8, 2];
  (or) List val = [];
       a.add(1);
       a.add(12);
       a.add('Nepal');
```

Method: also use in "set" and "map"

i) obj.length
ii) obj.isEmpty
iii) obj.add(value)
iv) obj.insert(index, value)
v) obj.addAll(obj2);
vi) obj.remove(value);
vii) obj.removeAt(index);
viii) obj.sort()
ix) obj.sublist(fromindex, toindex)
x) obj.indexOf(value)
xi) obj.reversed;
* xii) obj.whereType<int>(); // only return int valu
xiii) obj.toSet(); // change List → set ✗
obj.asMap(); // " List → map ✗
xiv) a[index] = value; //update/replace
xv) obj.toUpperCase();

(8) Set: 'unorderd'

- Not store duplicate value.
- index is not here.
- updation is not posible. Eg: a[0] = 51; ✗
- sort, shuffle, reversed method can't use

Syntax:-

→ var n = {10, 'He', 8, 71.62};
→ Set<int> n = {7, 8, 13};
Set n = <int>{7, 8, 13};

→ var a = <String>{}; // set bcz type written
a.add('Hello');
a.add('17');
a.addAll({2, a, "we"});

✗ → var a = {}; // map "if no type written"

→ Set a = const{10, 20, 30};
→ set a = {2, 8, "He", ...b};

- Not use method:-

i) insert(); first; last; removeAt(), removeLast();
sort(); shuffle(); reversed;

- use for-in & for-each loop but not for loop bcz indexing funda is not there.

(9) Map:

- "key - value" associated (unique key)
- unordered
-

Syntax:

```
var a = { 2: 'He', 31: True, 'She': false};
var <int, int> a = {};
    a[11] = 212;
    a[2] = 12;
    a[0] = 77;
    a[9] = 20;

map a = const{ 'A': 'Pawan', 12:12 };
map a = <int, dynamic>{ };   // Map a = map();
```

- print (a.values); only values print
- print (a['key']); print only specific keys value
- Map p = {1: 'Pawan', 2: 'Harry'};
  Map p2 = Map.from(p);

a.keys ⟶ // only print keys

- for (var i in a.values) { ⟶ // only values prints
    print (i);
  }
- a.forEach ((k, v) => print ('$k : $v'));
- update : obj.update ('old_val' value => 'new_val');
- Method (extra) :

i) .removeWhere ((k, v) { if() }); // specific

ii) obj.containsKey ('keyname'); // boolean
    obj.containsValue (value);

iii) obj.map ((k, v) { ═══ });

iv) obj.update ('keyn', (value) => changes);
    obj.updateAll ((k, v) { ═══ });

```
if (k == 7)
  return ++v;
```

Class - Object :

```
void main() {
  var x = new Demo();   (or)  var x = Demo();
  x..fun()..fun2()..fun3();      // call function
```

instance variable :

```
class Demo {                    // 4 way instance variable
  int a = -1                    (or)  late int a;
  String s = "";                      late String s;
  (or)
  Demo(this.a, this.s);
  (or)
  int? a;
  String? s;        -> null
```

instance variable

(~) assume simble Consider

```
void fun() {
  print($a);
}
dynamic fun2() => a;
String fun3() => s;
}
```

int (gives error)

Support 'Public' and 'private' → instance variable & fun()

Syntax :

```
int? x;          //public
void fun() {}

int? _y;         //private
int _fun() {_}
```

Super keyword :

- use in clild or inherited class to get immediate parent class data.

Eg: print(super.a);
print(super.fun());

this keyword :

- Resolve the name conflict. 'this' denotes the globle variable.

Eg:
```
class Demo {
  int x;
  void fun(int x) {
    this.x = x;
  }
```

Important :-

(1) we can import other file in our main file. just like java by import keyword.

" import 'file_name.dart' ; "

Static /method

(2) • Static variable, are class variable, so get the static variable by class name.
• local variable, globle variable can't static but instance variable.
• static method access only static variable. ~~but~~ but ~~Cannot static~~ non-static access both.
• single location shared to all objects.

Abstract / Interface : abstract class Demo { }

(3) • Abstract class contains abstract function and normal function both & constructor also.
• abstract function creates only abstract class.
• abstract function have no body.
• after extends (Inheritance) abstract class must define abstract function.
• no Interface keyword but achieve by class. and must again define class (super class) funitions.
"implements" keyword are here.
Ex:- class C implements A, B { ___ }
• abstract class can't be initiated.

(11) Constructor :

(1) Default (2) Parameterised (3) Named (own constru)

• at a time 'Default' and 'Parameterised' both of us can't use. Use only one. (Not Support Const-Overl padig)
• But can create more than one named Constructor at a time.

Syntax:

```
class Demo {
  Demo.namedConst() {          // named const
    print("Hi");
  }
  Demo.newConst(int a){
    print("His $a");
  }
}
```

✓ • Constant Constructor:

```
class Demo {
  final int x ;
  const Demo(this.x);
}
```

- instance variabe are not final. not allow.
- for. const constructor variable must final.

✓ • Constructor can be redirecting : can't have body

syntax:

```
Demo(this.x, this.y) { };
Demo.namedConstr() : this(10, 20);
```

Named Constructor

- Constructor are not inherited. but can 'super' key

Ex: class Parent {
    Parent(int a) { }
}

class Son extends Father {
    Son() { }
} // error

Son(): super(14) { }

(12) Getter & setter :-
↳ get it
↳ set data

- special type of method.
- should be same name.
- more than one getter and setter.

getter	Setter
• int get fun { return a; }	• void set fun (int a) { this.a = a; }
(or) int get fun => a;	(or) void set fun(int a) => this.a =
• No parameter list	• should be atleast one parameter
• Must return value.	• No return value.

Ex:

```
void main() {
  var obj = Demo();
  obj.fun = 77;
  print(obj.fun);
}
```

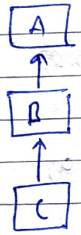
```
class Demo {
  int _a ;
  Demo(this._a);

  void set fun(int ass) {
    _a = ass;
  }

  int get fun => _a;
}
```

o/p : 77

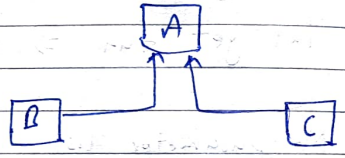
(13) Inheritance :

1) single inheritance :

2) Multilevel Inheritance:



3) Hierarchical inheritance:



- Not support multiple inheritance. Implement interfaces but interface keyword not exists.
- Private variable can access on same file, different class. but different class, file inherited private variable not access.
- when we override the function in child class it is recomended to use "@override".

(14) Abstract :-

- Abs. method not a body. Eg. void fun();
- Abs. method only in abst. class. Eg. abstract class A{ void fun(); }
- abs. method contains abs. func & normal fun.
- abs. class can't object.
- ~~ab~~ extended class must be define all abstract fun. on parent abs. class.
- abs. class have constructor.

(15) Interface :-

- no Interface keyword
- Eg: class A { int a=10; void fun() { } } // all data & functions are must redefine in implemented class.

class B implements A { int a = 99; void fun() { } } // after writting implements 'A' is interface so must all function are define in child class.

- multiple inheritance achieve i.e. multiple class implements at atime.

- Here in A class we can't write abstract function bcz for abst. function we have must a abstract class.
- static data & function redefining is optional
- Here B class not use super keyword bcz A is not class it's interface.
- implements also abstract class and interface.
- class D implements A, B, C { }

class D extends A implements B, C { }

(16) Mixing :- 'with', 'On'

- mixing A { void speaking () {} }

class Man with A { }

class Dog with A { }

- multiple mixing is possible.
  Eg: class Demo with A, B, C { }
- can't write constructor.
- mixing can't extents any class, but implements class, mixing.
- can't create object.
- By super keyword access mixing data.
- Can write abstract method.
- overide the mixing method's.

(17) Enum (Enumeration):

- represent fix no of constant values.
- It's class.
- print by loop.

Syntax: enum name-enum { val1, val2, --- valn }