Object- Oriented Principles,Functional Concepts

Module -3 Chapter 2

Core OOP Concepts

- Dart is an object-oriented programming language.
- OOP in Dart helps create **scalable, maintainable, and reusable software**.

OOP

- Defining classes and creating objects
- properties, methods, and constructors
- Pillars of OOP: Inheritance, Polymorphism, Abstraction, Encapsulation

Class and Object

- **Class** is a **blueprint** for creating objects.
- **Object** is an instance of a class — used to access its properties and methods.
- They have **properties (attributes)** and **methods (actions)**.

Class Members

- **Fields** – variables inside a class that hold data.
- **Getters/Setters** – control access to private fields.
- **Constructors** – special functions used to initialize objects.
- **Functions (Methods)** – define behaviors of an object.

```
Run | Debug
1  void main()
2  {
3    var n=college();
4    n.names();
5  }
6  class college
7  {
8    String c='GPTC kkm';
9    void names()
10   {
11     print('college name : $c');
12   }
13 }
```

```
PROBLEMS 16   OUTPUT   DEBUG CONSOLE   TERMI
college name : GPTC kkm

Exited.
```

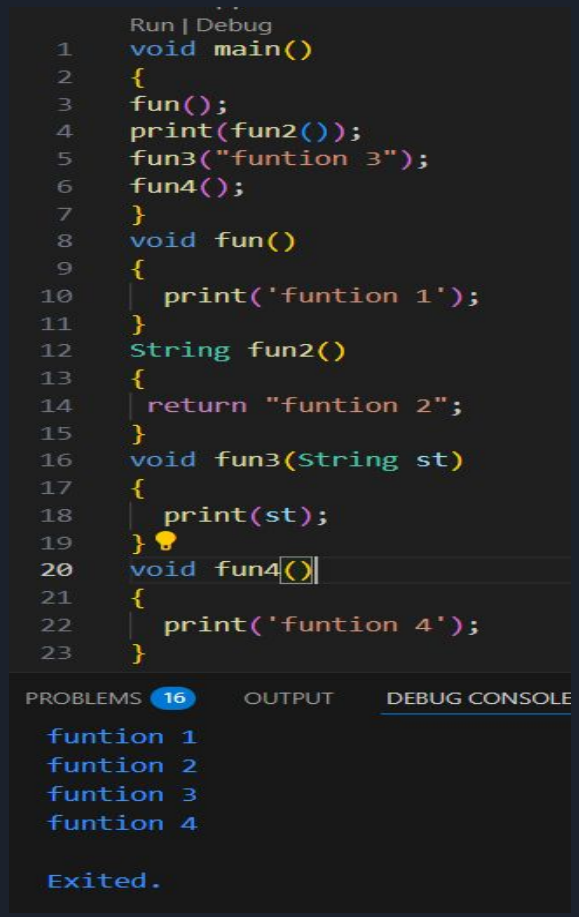
Functions

Functions are the **building blocks** of readable and reusable code.

Concepts:

- **Defining a function:** specifies when & how a task is done.
- **Calling a function:** executes it.
- **Returning functions:** send back a value.

Parameterized **functions:** use parameters to pass values.



Parameter Function Types

Optional Positional Parameter: use [ ]

Optional Named Parameter: use { }

- Must specify parameter name when passing value.

Optional Parameter with Default Value:

- Assign default values that can be overridden.

```
Run | Debug
1  void main()
2  {
3  position('one','two');
4  position('three');
5  print('');
6  Name(two:'password',one: 'username');
7  print('');
8  def('para 1');
9  def('para 3',two: 'para 4');
10 }
11 void position(String one, [String? two])
12 {
13   print(one);
14   print(two);
15 }
16 void Name({String? one,required String two}){
17   print('parameter1 $one');
18   print('parameter2 $two');
19 }
20 void def(String one, {String two ='admin'})
21 {
22   print('parameter1 $one');
23   print('parameter2 $two');
24 }
```

```
one
two
three
null

parameter1 username
parameter2 password

parameter1 para 1
parameter2 admin
parameter1 para 3
parameter2 para 4

Exited.
```

Run | Debug

```dart
void main()
{
position('one','two');
position('three');
print('');
Name(two:'password',one: 'username');
print('');
def('para 1');
def('para 3',two: 'para 4');
}
void position(String one, [String? two])
{
  print(one);
  print(two);
}
void Name({String? one,required String two}){
  print('parameter1 $one');
  print('parameter2 $two');
}
void def(String one, {String two ='admin'})
{
  print('parameter1 $one');
  print('parameter2 $two');
}
```

Constructors

Special functions with the **same name as the class**.

Execute **automatically** when an object is created.

Named constructors allow you to create and call custom constructors.

```
Run | Debug
1  void main()
2  {
3   var obj=cars();
4   obj.toyota();
5   var obj2=cars.suzuki();
6   obj2.bmw();
7  }
8  class cars
9  {
10   cars()
11   {
12     print('bmw,benz,toyota,suzuki,tata');
13   }
14   void toyota()
15   {
16     print('Toyota Supra');
17   }
18   cars.suzuki()
19   {
20     print('Suzuki cars:Alto K10, Swift,Baleno');
21   }
22   void bmw()
23   {
24     print('i7');
25   }
26 }
```

PROBLEMS 16 OUTPUT DEBUG CONSOLE TERMINAL PORTS

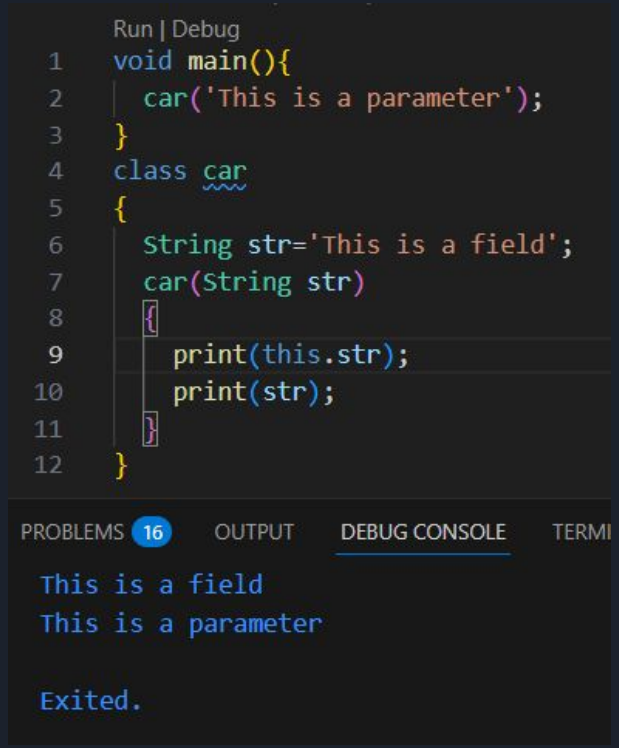
```
bmw,benz,toyota,suzuki,tata
Toyota Supra
Suzuki cars:Alto K10, Swift,Baleno
i7

Exited.
```

this Keyword

Refers to the **current instance** of a class.

Useful when **instance variables** share names with parameters.

Example: two strings named `str` (one class, one parameter);
`this.str` accesses the class variable.



Setters / Getters

Defined using `get` and `set` keywords.

Setter must run first; otherwise, getter will return `null`.

Getter
To read data
`print(obj.name)`
Setter
To assign data
`obj.name = 'value'`

```
Run | Debug
1  void main(){
2      var obj =car();
3      obj.Name = 'shift';
4      print(obj.Name);
5  }
6  class car{
7    late String suzuki;
8
9    set Name(String s){
10
11     this.suzuki = s;
12
13   }
14   String get Name{
15
16     return suzuki;
17   }
18 }
```

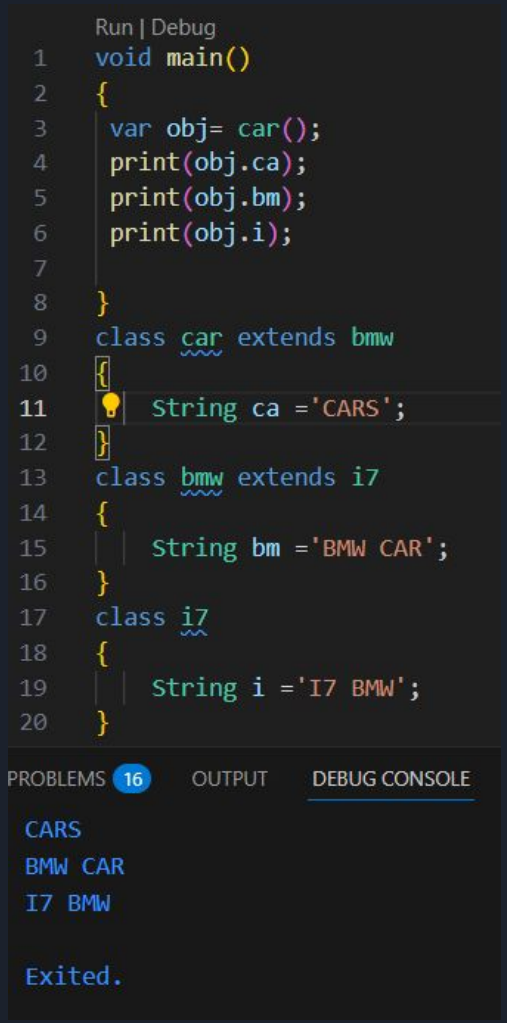
```
PROBLEMS 16   OUTPUT   DEBUG CONSOLE
shift

Exited.
```

Class Inheritance

- Allows one class (child) to access another class (parent).
- Create derived class objects to inherit parent class properties.

Types of Inheritance:

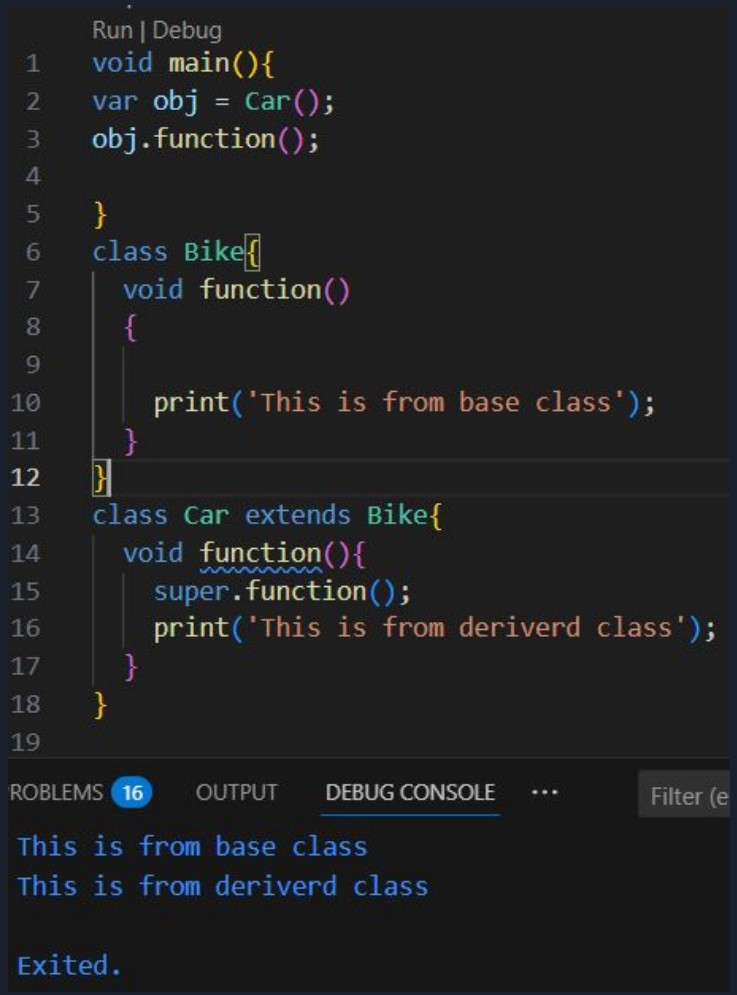
1. Single
2. Multilevel
3. Hierarchical



Super Keyword

`super.function()` calls the parent's version of the function.

Determines order of execution based on where it appears in the child function.



Abstract Class and Method

- Use `abstract` keyword for abstract class.
- **Abstract method:** use semicolon (;) instead of body.
- When a subclass inherits from the abstract class, it must write the actual code for this method. This process is called overriding.
- The Abstract Class cannot create objects/instances of it. You can only inherit (extend) it with a concrete subclass.

```
Run | Debug
1  void main(){
2  var x= 20; var y=15;
3  Ans().val(x, y);
4  Ans().add(x, y);
5  Ans().sub(x, y);
6  }
7  abstract class Cal{
8    void val(int x,int y);
9
10   void add(int x,int y){
11     print('add: ${x+y}');
12   }
13   void sub(int x,int y){
14     print('sub: ${x-y}');
15   }
16 }
17 class Ans extends Cal{
18   @override
19   void val(int x,int y){
20     print('number1: $x , number2: $y');
21   }
22 }
```

```
PROBLEMS 16   OUTPUT   DEBUG CONSOLE   ...   Filter (
number1: 20 , number2: 15
add: 35
sub: 5

Exited.
```

Polymorphism

Ability of an object,function or method to take **many forms**.

Types:

- **Compile-time** (method overloading)

  Same method name, different parameters.

- **Run-time** (method overriding)

Example: animal(class) sub - cat ,dog = breed (function)

```
Run | Debug
1  void main()
2  {
3    Animal cat=Cat();
4    Animal dog=Dog();
5    cat.breeds();
6    dog.breeds();
7  }
8  class Animal
9  {
10   void breeds()
11   {
12    print('animals');
13   }
14 }
15 class Cat extends Animal
16 {
17   void breeds()
18   {
19    print('cat: pershian');
20   }
21 }
22 class Dog extends Animal
23 {
24   void breeds()
25   {
26    print('dog : beagle');
27   }
28 }
```

```
PROBLEMS 16   OUTPUT   DEBUG CONSOLE   ...
cat: pershian
dog : beagle

Exited.
```

Interface

An interface is like a rulebook or contract between classes.

It defines what methods a class must have, but not how they work.

A **normal class** can serve as an **interface** using `implements`.

Must **override all methods** of the implemented class.

Multiple classes can be implemented, but **only one class** can be extended.

Interfaces are **important in Flutter** for cross-platform behavior.

```
Run | Debug
void main()
{
  Vehicles().bmw();
  print('');
  Vehicles().suzuki();
  print('');
  Vehicles().hero();
  print('');
  Vehicles().tvs();
}
abstract class car {
  void bmw();
  void suzuki();
}
abstract class bike{
  void hero();
  void tvs();
}
class Vehicles implements car,bike{
  Vehicles(){
    print('vehicle........');
  }
  @override
  void  bmw(){
    print('BMW - CAR');
  }
  @override
  void suzuki() {
    print('suzuki - CAR');
  }
  @override
  void hero(){
    print('hero bike');
  }
  @override
  void tvs() {
    print('tvs bike');
  }
}
```

```
vehicle........
BMW - CAR

vehicle........
suzuki - CAR

vehicle........
hero bike

vehicle........
tvs bike

Exited.
```

Exercise 1

Palindrome Number Check

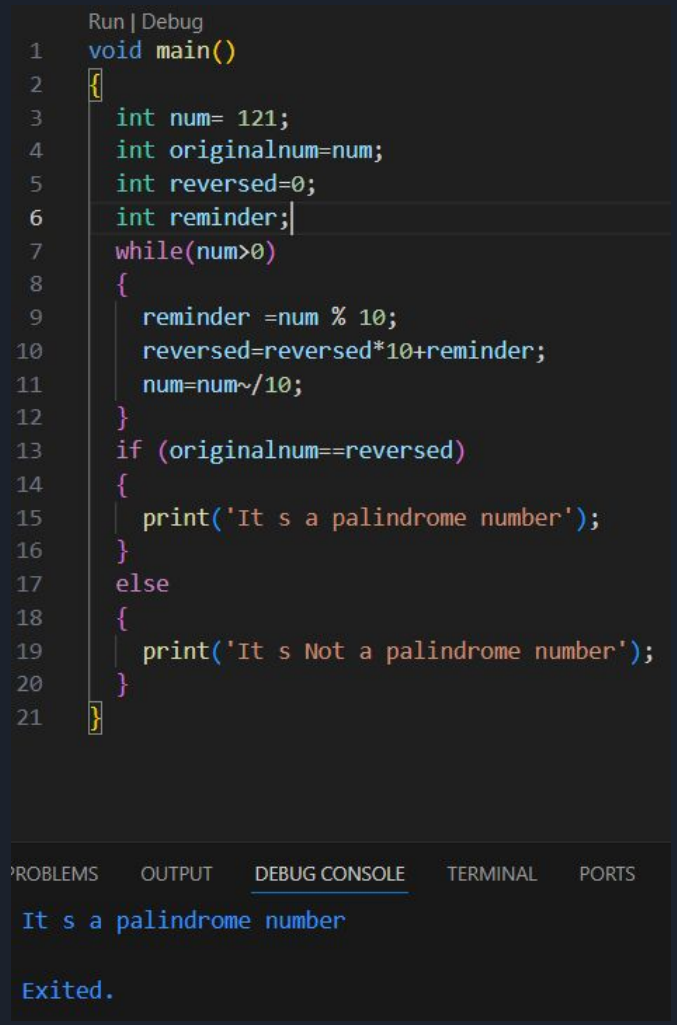
`while(num > 0)`
→ Loop runs until `num` becomes 0.

`reminder = num % 10;`
→ Gets the last digit of `num`.
Example: if `num = 121`, then `reminder = 1`.

`reversed = reversed * 10 + reminder;`
→ Builds the reversed number digit by digit.
Example: `0 * 10 + 1 = 1`, then next loop → `1 * 10 + 2 = 12`,
,12 *10+2 =121

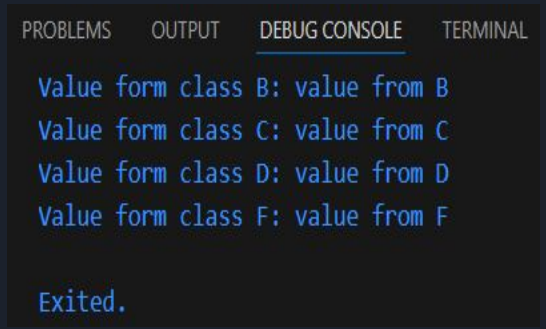
`num = num ~/ 10;`
→ Removes the last digit of `num`.
Example: `121 ~/ 10 = 12`.
12/ 10 = 1 , 1/10 =0

After loop ends
→ `reversed` holds the reversed version of the original number.



Exercise 2

Multiple Hierarchy




```
Run | Debug
1  void main() {
2    A a = A();
3    a.printval();
4  }
5
6  class A with B, C, D, F {
7    void printval() {
8      print('Value from class B: $bstr');
9      print('Value from class C: $cstr');
10     print('Value from class D: $dstr');
11     print('Value from class F: $fstr');
12   }
13 }
14
15 mixin B {
16   String bstr = 'value from B';
17 }
18
19 mixin C {
20   String cstr = 'value from C';
21 }
22
23 mixin D {
24   String dstr = 'value from D';
25 }
26
27 mixin F {
28   String fstr = 'value from F';
29 }
```