Dart – Day4

Emp-id : 4781

- **Anonymous Function**

In Dart, an anonymous function is a function without a name, often used as a short inline function.

Example:

```
void main()
{
  var greet = (String name)
  {
    print("Hello, $name!");
  };
  greet("Chandini");   // Hello, Chandini!
}
```

- **Closure**

A closure is a function that can use variables from its surrounding scope even after the outer function has finished.

Example:

```
Function multiplier(int factor)
{
  return (int number) => number * factor; // closure
}

void main()
{
  var doubleIt = multiplier(2);
  print(doubleIt(5)); // 10
```

```
}
```

- ## Lambda Function

A lambda function is just a short function written using the arrow syntax (=>).

Example:

```
void main()
 {
  var square = (int x) => x * x;
  print(square(4));   // 16
}
```

- ## Object-Oriented Programming (OOP)

OOP is a way of programming where you model real-world things as objects (data + behavior). Dart supports OOP with classes and objects.

Example:

```
class Car
{
  String brand = "Tesla";
  void drive()
  {
    print("$brand is driving");
  }
}

void main()
{
  var myCar = Car();
  myCar.drive(); // Tesla is driving
}
```

- ## Class

A class is a blueprint for creating objects. It contains fields (variables) and methods (functions).

Example:

```
class Student
{
  String name = "Chandini";
  void display()
  {
    print("Student: $name");
  }
}

void main()
{
  var s1 = Student();
  s1.display(); // Student: Chandini
}
```

- ## Constructor

A constructor is a special method in a class that is called automatically when an object is created.

Example:

```
class Person
{
  String name;
  Person(this.name); // constructor
}

void main()
{
  var p = Person("Sneha");
  print(p.name); // Sneha
}
```

- ## Types of Constructor

1. **Default Constructor** → created automatically if not defined.
2. **Parameterized Constructor** → takes arguments.
3. **Named Constructor** → gives multiple ways to create objects.

1. Default Constructor

If no constructor is defined, Dart provides a default constructor automatically.

Example:

```
class Student
{
  String name = "Chandini";
  int age = 21;
}

void main()
{
  var s = Student(); // default constructor
  print("${s.name}, ${s.age}"); // Chandini, 21
}
```

2. Parameterized Constructor

A constructor that accepts arguments to initialize variables.

Example:

```
class Student
{
  String name;
  int age;

  Student(this.name, this.age); // parameterized constructor
```

```
}

void main()
{
  var s = Student("Sneha", 22);
  print("${s.name}, ${s.age}"); // Sneha, 22
}
```

3. Named Constructor

Used when you want multiple ways to create objects.

Example:

```
class Student
{
  String name;
  int age;

  Student(this.name, this.age); // parameterized constructor
  Student.guest()
{  // named constructor
    name = "Guest";
    age = 0;
  }
}

void main()
{
  var s1 = Student("Neeha", 23);
  print("${s1.name}, ${s1.age}"); // Neeha, 23

  var s2 = Student.guest();
  print("${s2.name}, ${s2.age}"); // Guest, 0
}
```

- **this Keyword**

this refers to the current object of the class.

Example:

```
class Dog
{
  String name;
  Dog(this.name);

  void display()
{
    print("Dog's name is ${this.name}");
  }
}

void main()
{
  var d = Dog("Bruno");
  d.display(); // Dog's name is Bruno
}
```

- **Method Chaining**

Method chaining allows calling multiple methods on the same object in a single statement.

Example:

```
class Calculator
{
  int value = 0;

  Calculator add(int n)
  {
    value += n;
    return this;
  }
```

```
  Calculator multiply(int n)
  {
    value *= n;
    return this;
  }
}

void main()
{
  var calc = Calculator();
  calc.add(5).multiply(3);
  print(calc.value); // 15
}
```

- **List**

A List in Dart is an ordered collection of items (like an array).

Example:

```
void main()
{
  var numbers = [10, 20, 30];
  print(numbers);      // [10, 20, 30]
  print(numbers[1]);   // 20
}
```

- **List Methods in Dart**

Dart provides many methods to work with lists.

```
void main()
{
  var fruits = ["Apple", "Banana", "Mango"];

  // 1. Add elements
 fruits.add("Orange");
```

```
fruits.addAll(["Grapes", "Pineapple"]);
print(fruits); // [Apple, Banana, Mango, Orange, Grapes, Pineapple]

// 2. Insert elements
fruits.insert(1, "Cherry");          // at index 1
fruits.insertAll(2, ["Kiwi", "Papaya"]);
print(fruits);

// 3. Remove elements
fruits.remove("Banana");             // removes first match
fruits.removeAt(0);                  // removes index 0
fruits.removeLast();                 // removes last element
fruits.removeRange(1, 3);            // removes index 1 & 2
print(fruits);

// 4. Search/Check
print(fruits.contains("Mango"));      // true
print(fruits.indexOf("Mango"));       // index of Mango
print(fruits.lastIndexOf("Mango"));   // last index of Mango

// 5. Replace
fruits[0] = "Strawberry";            // replace by index
fruits.replaceRange(0, 2, ["Peach", "Plum"]);
print(fruits);

// 6. Sort & Reverse
fruits.sort();                       // sort alphabetically
print(fruits);
var reversed = fruits.reversed.toList();
print(reversed);

// 7. Sublist
print(fruits.sublist(0, 2));         // first 2 items

// 8. Properties
print(fruits.length);                // number of items
print(fruits.isEmpty);               // false
print(fruits.isNotEmpty);            // true
print(fruits.first);                 // first element
print(fruits.last);                  // last element
```

```
  // 9. Clear
  fruits.clear();                  // remove all
  print(fruits);                   // []
}
```

- ## Iterating a List

In Dart, you can loop through a list in multiple ways to access each element.

Example:

```
void main()
{
  var fruits = ["Apple", "Banana", "Mango", "Orange"];

  // 1. For loop (using index)
  for (int i = 0; i < fruits.length; i++) {
    print("Fruit at index $i is ${fruits[i]}");
  }

  // 2. For-in loop
  for (var fruit in fruits) {
    print("Fruit: $fruit");
  }

  // 3. forEach method
  fruits.forEach((fruit) {
    print("I like $fruit");
  });
```

- ## Fixed-length List

A fixed-length list has a constant size. You cannot add or remove elements, but you can change existing ones.

Example:

```
void main()
{
  var fixedList = List<int>.filled(5, 0);
  // length = 5, all initialized with 0

  print(fixedList);   // [0, 0, 0, 0, 0]

  fixedList[2] = 10;
  print(fixedList);   // [0, 0, 10, 0, 0]

  // fixedList.add(20);    //  Error
  // fixedList.removeAt(1);  //  Error
}
```

- ## Growable List

A growable list can increase or decrease in size. You must set growable: true when creating with List constructor.

Example:

```
void main() {
  var growList = List<int>.filled(3, 1, growable: true);

  print(growList);   // [1, 1, 1]

  growList.add(5);
  growList.add(10);
  print(growList);   // [1, 1, 1, 5, 10]

  growList.removeAt(2);
  print(growList);   // [1, 1, 5, 10]

  growList.insert(1, 99);
  print(growList);   // [1, 99, 1, 5, 10]
}
```

Summary :

- List.filled(size, value, growable: false) → Fixed-length list.
- List.filled(size, value, growable: true) → Growable list.
- Growable lists can add, insert, and remove elements, while fixed-length cannot.

- ## map() - transforming

The map() method is used to transform each element of a list (or any iterable) into something new.
It returns a new iterable with the transformed values.

Example :

```
void main() {

  var numbers = [1, 2, 3, 4, 5];

  // 1. Square each number

  var squares = numbers.map((n) => n * n);

  print(squares.toList()); // [1, 4, 9, 16, 25]

  // 2. Convert numbers to strings

  var texts = numbers.map((n) => "Value: $n");

  print(texts.toList());   // [Value: 1, Value: 2, Value: 3, Value: 4, Value: 5]

  // 3. Convert numbers to boolean (true if even)

  var isEven = numbers.map((n) => n % 2 == 0);

  print(isEven.toList());  // [false, true, false, true, false]

}
```

- ## where() - filtering

The where() method is used to filter elements of a list based on a condition.
It returns a new iterable containing only the elements that match the condition.

Example :

```
void main() {
  var numbers = [5, 10, 15, 20, 25, 30];
  // 1. Get even numbers
  var evens = numbers.where((n) => n % 2 == 0);
  print(evens.toList());  // [10, 20, 30]
  // 2. Get numbers greater than 15
  var greater = numbers.where((n) => n > 15);
  print(greater.toList()); // [20, 25, 30]
  // 3. Get numbers less than or equal to 10
  var small = numbers.where((n) => n <= 10);
  print(small.toList());  // [5, 10]
}
```

- ## Spread Operator (...)

The spread operator is used to insert all elements of one list into another list.

Example:

```
void main() {
  var list1 = [1, 2, 3];
  var list2 = [4, 5, 6];

  var combined = [...list1, ...list2];
  print(combined); // [1, 2, 3, 4, 5, 6]
}
```

... takes all elements of list1 and list2 and spreads them into combined.

- **Null-aware Spread Operator (...?)**

When the list might be null, use ...? to avoid errors.
If the list is null, it simply adds nothing instead of throwing an exception.

Example:

```
void main() {
  var list1 = [10, 20, 30];
  List<int>? list2 = null;  // nullable list

  var combined = [...list1, ...?list2];
  print(combined); // [10, 20, 30]
}
```

...?list2 → safely handles null (does nothing if list2 is null)