

Flutter & Dart Complete Guide

Prepared by Eng. Yousef Salah Kassem

DAFL 1
Collections & null safety

Lecture 4



Table Of Content

- Introduction
- List (fixed & growable)
- Set
- Map
- Null Safety
- Functions
- Problem Solving



Introduction

Collections

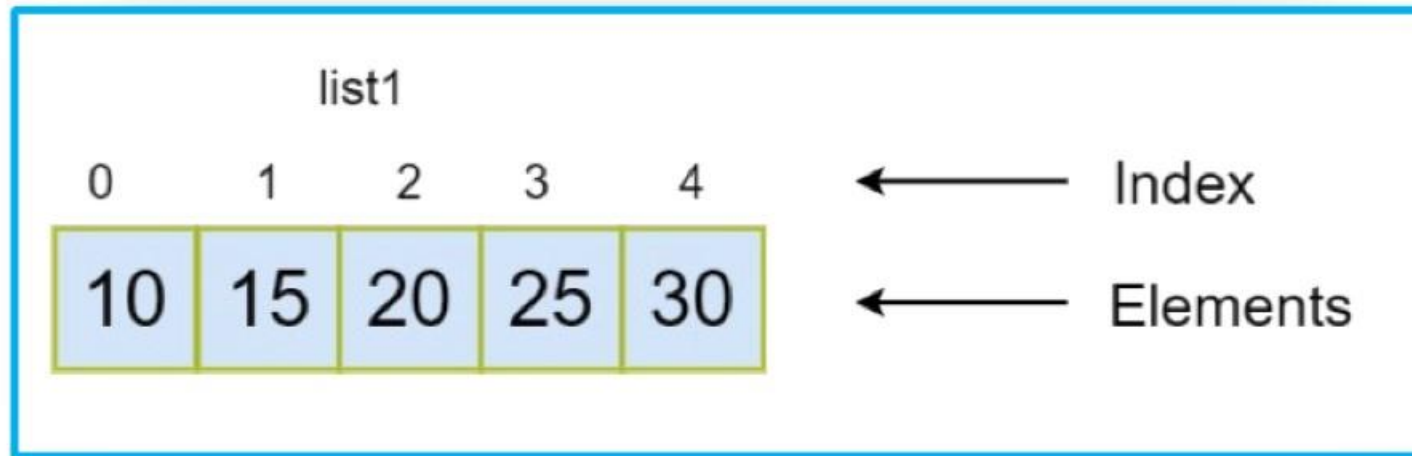
- ◆ In Dart, a collection is an ordered group of items, unlike unordered collections in some other programming languages. Typically, they can contain elements, each of which can hold either simple variables or complex objects.



- Dart List is similar to an array, which is the ordered collection of the objects.
- The array is the most popular and commonly used collection in any other programming language. The Dart list looks like the JavaScript array literals.
- The syntax of declaring the list is given below. `var list1 = [10, 15, 20,25,25]`
- The Dart list is defined by storing all elements inside the square bracket ([]) and separated by commas (,).

Chapter
04

List



Fixed Length List

- The fixed-length lists are defined with the specified length. We cannot change the size at runtime. The syntax is given below.
- Syntax - Create the list of fixed-size
- `var list_name = new List(size)`
 - The above syntax is used to create the list of the fixed size. We cannot add or delete an element at runtime. It will throw an exception if any try to modify its size.

Chapter
04**Fixed Length List**

- Syntax - Initialize the fixed size list element `list_name[index] = value;`
- Let's understand the following example.
- Example-

```
void main() {  
var list1 = new List(5);
```

```
list1[1] = 11;
```

```
list1[2] = 12;
```

```
list1[3] = 13;
```

```
list1[4] = 14;
```

```
}
```

```
list1[0] = 10;
```

```
print(list1);
```

Growable List

- The list is declared without specifying size is known as a Growable list. The size of the Growable list can be modified at the runtime.
- The syntax of the declaring Growable list is given below.

- Syntax - Declaring a List

// creates a list with values

```
var list_name = [val1, val2, val3]
```

Or

// creates a list of the size zero

```
var list_name = new List()
```

Growable List

- Syntax - Initializing a List

```
list_name[index] = value;
```

- Consider the following example -
- Example - 1

```
void main() {  
    var list1 = [10,11,12,13,14,15];  
    print(list1);  
}
```

Chapter

04

Growable List

- In the following example, we are creating a list using the empty list or List() constructor. The add() method is used to add element dynamically in the given list.
- Example

```
void main() {  
    var list1 = new List();  
  
    list1.add(10);  
  
    list1.add(11);  
  
    list1.add(12);  
  
    list1.add(13);  
  
    print(list1);  
}
```

Chapter 04

List Properties

first	It returns the first element case.
isEmpty	It returns true if the list is empty.
isNotEmpty	It returns true if the list has at least one element.
length	It returns the length of the list.
last	It returns the last element of the list.
reversed	It returns a list in reverse order.
Single	It checks if the list has only one element and returns it.

Chapter
04**Inserting Element into List**

- Dart provides four methods which are used to insert the elements into the lists. These methods are given below.
 - add()
 - addAll()
 - insert()
 - insertAll()

Removing Element from List

- The Dart provides following functions to remove the list elements.
 - `remove()`
 - `removeAt()`
 - `removeLast()`
 - `removeRange()`

Set

- Set is the unordered collection of the different values of the same type. It has much functionality, which is the same as an array, but it is unordered.
- Set doesn't allow storing the duplicate values. The set must contain unique values.
- It plays an essential role when we want to store the distinct data of the same type into the single variable.
- Once we declare the type of the Set, then we can have an only value of the same type. The set cannot keep the order of the elements.

- Dart provides the two methods to declare/initialize an empty set.
- The set can be declared by using the {} curly braces preceded by a type argument, or declare the variable type Set with curly braces {}. The syntax of declaring set is given below.

- Syntax -

```
var setName = <type>{};
```

Or

```
Set<type> setname = {};
```

```
void main(){  
    print("Initializing the Set");  
    var names = <String>{"Rima","Mohan","Babu","Ram"};  
    print(names);  
}
```

Chapter

04

Set - Add Elements

```
// Declaring empty set
```

```
var emp = <String>{};
```

```
emp.add("Jay");
```

```
print(emp);
```

```
// Adding multiple elements
```

```
emp.addAll(names);
```

```
print(emp);
```

Accessing Elements

- Dart provides the `elementAt()` method, which is used to access the item by passing its specified index position.
- The set indexing starts from the 0 and goes up to `size - 1`, where `size` is the number of the element exist in the Set.
- It will throw an error if we enter the bigger index number than its size. The syntax is given below.

– `Set_name.elementAt(index)`

Other methods

`set_name.contains(value);`

`set_name.remove(value);`

`set_name.clear();`

`x.union(y)`

`x.intersection(y)`

`y.difference(z)`

Map

- Map is an object that stores data in the form of a key- value pair. Each value is associated with its key, and it is used to access its corresponding value. Both keys and values can be any type.
- In Dart Map, each key must be unique, but the same value can occur multiple times. The Map representation is quite similar to Python Dictionary.
- The Map can be declared by using curly braces {}, and each key-value pair is separated by the commas(,).
- The value of the key can be accessed by using a square bracket([]).

```
void main() {  
var student = {'name':'Raju','age':'21'}; print(student);  
}  
student['course'] = 'B.tech';
```

Chapter

04

Map

```
void main() {  
var student = new Map();  
  
student['name'] = 'Raju';  
student['age'] = 21;  
student['course'] = 'B.tech';  
  
student['Branch'] = 'Computer Science';  
print(student);  
}
```

Keys	It is used to get all keys as an iterable object.
values	It is used to get all values as an iterable object.
Length	It returns the length of the Map object.
isEmpty	If the Map object contains no value, it returns true.
isEmpty	If the Map object contains at least one value, it returns true.

```
student.addAll({'dept':'Civil','email':'yousef@google.com'});  
student.clear();  
student.remove('age');
```

Null Safety

In the Dart programming language, the null value is used to indicate the absence of a value or the absence of a reference to any object. This allows you to explicitly indicate that a variable does not hold any valid value.

Chapter

04

Null Safety

- “Null safety” is a feature in the Dart language that helps prevent many errors related to null values, thereby enhancing program reliability and safety.
- In the Dart language prior to the introduction of “Null safety,” variables could hold a null value (indicating the absence of a value), which could lead to unpredictable errors during program execution.

Chapter

04

Null Safety

For Example:

```
String name;  
print(name.length); // Error! Attempt to access a property of a null value.
```

With “Null safety,” new data types and annotations have been introduced, which allow you to specify whether a variable can accept a null value or not.

Nullable variables

- Variables that can have a null value are marked with a question mark ? after the data type.

For example: `String? name;`

Now the variable “name” can hold a string or null.

Non Nullable variables

- Variables that cannot have a null value are declared without a question mark.

For example: `String name = 'yousef';`

Now the variable “name” can hold a string.

Safe access operators

To work safely with nullable variables, the operators `?.` and `??` are introduced. The `?.` operator allows you to invoke methods and access properties only if the variable is not equal to null. The `??` operator allows you to specify a default value that will be used if the variable is null

Chapter

04

Safe access operators

Example :

```
String? name;
```

```
int nameLength = name?.length ?? 0; // If "name" is not null, its length will be returned;
```

otherwise, 0 will be returned.

Functions

- Function is a set of codes that together perform a specific task.
- It is used to break the large code into smaller modules and reuse it when needed.
- Functions make the program more readable and easy to debug.
- It improves the modular approach and enhances the code reusability.

- Suppose, we write a simple calculator program where we need to perform operations number of times when the user enters the values.
- We can use the functions multiple times by calling.
- We can create different functions for each calculator operator. By using the functions, we don't need to write code for adding, subtracting, multiplying, and divide again and again.

- It increases the module approach to solve the problems.
- It enhances the re-usability of the program. We can do the coupling of the programs.
- It optimizes the code.
- It makes debugging easier.
- It makes development easy and creates less complexity.

A function can be defined by providing the name of the function with the appropriate parameter and return type. A function contains a set of statements which are called function body. The syntax is given below.

Syntax:

```
return_type func_name (parameter_list): {  
    //statement(s) return value;  
}
```

```
int mul(int a, int b){  
    int c;  
    c=a+b;  
    print("The sum is:${c}");  
}
```

Chapter
04**Functions - Parameters**

- When a function is called, it may have some information as per the function prototype is known as a parameter (argument).
- The number of parameters passed and data type while the function call must be matched with the number of parameters during function declaration. Otherwise, it will throw an error.
- Parameter passing is also optional, which means it is not compulsory to pass during function declaration. The parameter can be two types.
- Actual Parameter- A parameter which is passed during a function definition is called the actual parameter.
- Formal Parameter- A parameter which is passed during a function call is called the formal parameter.

- Dart also provides the facility to specify a nameless function or function without a name. This type of function is known as an anonymous function, lambda, or closure.
- An anonymous function behaves the same as a regular function, but it does not have a name with it. It can have zero or any number of arguments with an optional type annotation.
- We can assign the anonymous function to a variable, and then we can retrieve or access the value of the closure based on our requirement.

An Anonymous function contains an independent block of the code, and that can be passed around in our code as function parameters. The syntax is as follows.

```
Syntax: (parameter_list) {  
statement(s)  
}
```

Functions

```
void main() {  
    var list = ["Ram","Raj","Rani","Ritu"];  
    print("Example of anonymous function");  
    list.forEach((item) {  
        print('${list.indexOf(item)}: $item');  
    });  
}
```

Building a Restaurant Management System in Dart

You're hired to develop a small management system for a local restaurant. The restaurant needs to handle menu items, customer orders, and ratings. The system should make use of Dart's collection manipulation features, null safety, and functions to fulfill these requirements. Each part of the system represents a different problem, which you'll solve step by step.

Chapter 04

Problem Solving

Requirements

1. **Menu Management:**
 - The system must manage the restaurant's menu using a list of items and their prices.
 - You need to provide the ability to calculate the total cost of an order.
2. **Set Operations:**
 - The system should track the restaurant's unique dishes using a set to ensure no duplicate entries are allowed.
3. **Customer Orders:**
 - Store customer orders using a map, where the keys are customer names, and the values are the items they ordered.
 - The system should also handle customer orders, displaying a customer's current order and allowing updates.
4. **Nested Collections for Ratings:**
 - The restaurant collects ratings from customers for different dishes. Store ratings in a nested collection format, with each dish having a list of ratings.
5. **Null Safety:**
 - Ensure the system handles potential null values, especially for scenarios where no orders or ratings are available yet.
6. **Pizza Ordering using Maps and Functions:**
 - A customer should be able to place an order for a pizza, and the system should use a map to track the ordered items and calculate the total.
7. **Spread Operators and Collections:**
 - Build a shopping list using the spread operator, allowing additional items to be easily added or removed.
8. **Handling Null Values:**
 - The system must handle cases where orders or ratings are null, providing default behaviors.

What is Next ?



Thank You