By: Eng. Elias Al-Showaiter

[Dart Programming Course] - Lecture [5]

[Lists and Maps]

1. Introduction

One of the most common tasks a programmer has to do in a daily basis, is handling data, and most of the times data comes in collection, so creating a single variable of type string or int is not enough anymore we need a variable that can store any amount of values we want, not only store them, but also performs multiple operations on them. And there are two major kinds of these variables, Lists and Maps. We will take a look at both of them in this lecture.

Learning objectives:

1. Understand the basics of Lists and Maps.
2. Understand the difference between Lists and Maps.
3. Analyze when to use Maps and when to use Lists.

2. Context

Technical Background

In mobile applications dealing with data is a constant continuous task, and especially with databases. The most optimal way to deal with this data is to store them in a single place. And the uses of Lists and Maps are not only connected to storing data from databases, but also storing normal values.

Key Terms and Definitions:

- List: an ordered collection of objects. Each element in a List can be accessed by its index.
- Maps: an unordered collection of key-value pairs, where each key is unique.
- Database: a large source of data (very simple explanation).
- Index: a variable that is used to locate a value.
- Key: a variable that is used to located a value and its value can be anything.

3. Main Content

1. Lists:

Let’s start by taking a look how to create a List in dart:

List numbers = [ ];

In this simple instruction, we created a variable called numbers, the data type of this variable is List, its initial value is empty.

I can define some values to the list in the beginning or I can add it to them later:

List numbers = [1,2,3];

Or

Using a built-in behavior of Lists called add(),
numbers.add(1);
numbers.add(2);
numbers.add(3);

A List can contain any data type we want, for example
List numbersAndTexts = [1 , 2, 3, "Ahmed" , "Ali"];

We can see that this List contains numbers and texts, sometimes this is useful but in most cases this is not recommended, if we want to create a list with a single data type we do the following:
List<int> numbersOnly = [1,2,3];
List<String> stringsOnly = ["Ahmed" , "Ali"];

In the first List only numbers are allowed, and in the second List, only Strings are allowed.

Now that we understood how lists store data, let's take a look how do we read this data from the list.

Since Lists are ordered, we can use the index of that value to get is, so the syntax of getting an element from a List is as follows:

listName[indexOfValue];

we use the name of the list, then use [ ] and choose what position I want to get from this list.
this works because lists are ordered.

We retrieve the data by using the index of the value we want. For example, in this list, List<int> numbers = [1,2,3,4,5,6,7,8,9] if I want to get the number 1 we need to read the index with the value 0 (Lists start counting from 0) so to show this result

we write
print(numbers[0]).

The true power of lists comes when we use them with its built-in behaviors (functions) and attributes. For this reason, let's take a look at some of the most use behaviors.

All of the attributes and behaviors are applied on the values [1,2,3]

Attribute/Method	Description	Example	Result
length	Returns the number of elements in the list.	print(list.length);	3
isEmpty	Checks if the list is empty.	print(list.isEmpty);	false
isNotEmpty	Checks if the list is not empty.	print(list.isNotEmpty);	true
add(element)	Adds a single element to the end of the list.	list. add(4);	list becomes [1, 2, 3, 4]
insert(index, element)	Inserts an element at the specified index.	list.insert(1,8);	list becomes [1, 8, 2, 3]
addAll(iterable)	Adds all elements of another iterable to the list.	list.addAll([4, 5, 6]);	list becomes [1, 2, 3, 4, 5, 6]
remove(element)	Removes the first occurrence of the element.	list.remove(3);	list becomes [1, 2, 4]
removeAt(index)	Removes the element at the specified index.	list.removeAt(0);	list becomes [2, 3]
clear()	Removes all elements from the list.	list.clear();	list becomes []
reversed	Returns an iterable with the elements in reverse order.	print(list.reversed);	reversed is (3, 2, 1)
sort()	Sorts the list in place.	list.sort();	list becomes [1, 2, 3]
contains(element)	Checks if the list contains a specific element.	print(list.contains(5));	false

2. Maps:

Let’s start by looking how to create a map:

Map ages = { };

This creates a variable named ages of the data type Map, its initial value is empty. before we start looking at how maps work we should see the syntax of defining an entry in a map.

Map mapName = {key: value, key: value};

We can see that a single entry in a map consists of two parts a key and a value. The key is used to identify the value. It is very similar to an index, but a key we can change its value as we want. So Lists and Maps are very similar, the only important difference is that lists have a constant index that starts counting from 0, but maps we can change the key as we want.

So we use lists when the data is ordered, but we use maps when we want to associate data with a specific key.

Now Let’s see how to store data in a map.

We have three methods to add entries into a Map

1. Giving it some initial values:
   Map ages = {“Ali” : 18 , “Ahmed” : 20 };
2. Using a new key:
   ages[“Omar”] = 21;
3. Using built-in behavior (function):
   ages.addAll({“Omar” : 21});

Now Let’s see how to get data from Maps.

In Maps if we want to get a specific value we need to use its key, for example, in previous map if I want to know what is the age of Omar I will do the following:
int omarAge = ages[“Omar”];
this will result on the value of omarAge to be 21.

The power of Maps is also when we use attributes and behaviors (functions), so let’s take a look at some of the most used ones.

All of the attributes and functions are applied on a simple map with these entries:
{“a” : 1 , “b” : 2 , “c” : 3 }

Attribute/Method	Description	Example	Result
length	Returns the number of key-value pairs in the map.	print(map.length);	3
isEmpty	Checks if the map is empty.	print(map.isEmpty);	false
isNotEmpty	Checks if the map is not empty.	print(map.isNotEmpty);	true
keys	Returns an iterable of all the keys in the map.	print(map.keys);	('a', 'b', 'c')
values	Returns an iterable of all the values in the map.	print(map.values);	(1, 2, 3)
addAll(otherMap)	Adds all key-value pairs from another map to this map.	map.addAll({'d': 4, 'e': 5});	map becomes {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
remove(key)	Removes the key and its associated value from the map.	map.remove('a');	map becomes {'b': 2, 'c': 3}
clear()	Removes all key-value pairs from the map.	map.clear();	map becomes {}
containsKey(key)	Checks if the map contains the specified key.	print(map.containsKey('a'));	true
containsValue(value)	Checks if the map contains the specified value.	print(map.containsValue(2));	true

3. Examples:

Checking if a list contains an even number or not

```
void main() {
 List<int> numbers = [1, 3, 5, 7, 8, 11];
 int evenCount = 0;
 for (int number in numbers)
        {
          if (number % 2 == 0)
            {
            evenCount++;
           } }
}
```

Example 2:

```
void main() {
Map<String, List<int>> studentMarks = { 'Alice': [85, 90, 78, 92], 'Bob': [70, 60, 88, 75],
'Charlie': [95, 100, 85], 'David': [80, 70, 65, 90] };
 for (String studentName in studentMarks.keys)
        {
              List<int> marks = studentMarks[studentName]!;
              int sum = 0;
              for (int mark in marks)
                  {
                       sum += mark;
                   }
              int average = sum / marks.length;
             print('$ studentName: Average Mark = $ {average.toStringAsFixed(2) } );
         }
 }
```

Example 3:

```
void main() {
 List<String> words = ['apple', 'banana', 'apple', 'orange', 'banana', 'apple'];
 Map<String, int> wordCount = {};
 for (String word in words)
       {
        if (wordCount.containsKey(word)) {
           wordCount[word] = wordCount[word]! + 1;
                                            }
       else
           {
              wordCount[word] = 1;
          }
        }
print(wordCount);

}
```