Dart

Collections

Carmelo Villegas Cruz

Collections

Son un elemento muy importante a la hora de desarrollar interfaces en Flutter.

En esta presentación nos vamos a centrar y describir en:

- List
- Set
- Maps

Lists

Estructura de datos que contiene una colección de datos de forma ordenada.

Este orden se consigue a través de los índices, que van desde 0 hasta en número de elementos menos uno.

Veamos cómo usar List en la práctica.

Lists


```
//Creando una List, usamos []
var postres = [‘Galletas’, ‘Flan’, ‘Tarta’, ‘Fruta’];

//¿De qué tipo es la lista?
//Se ha creado una lista de tipo String, dado que todos los
//son de tipo String

//¿Y si hacemos lo siguiente, de qué tipo sería?
var snacks = [];

//La respuesta es dynamic. Esto causa lo conocido como lose
type safety que debemos evitar con las colecciones.
```

Lists


```
//Para evitar lose type safety, la declaración debe ser más
//explícita.
List<String> snacks = [];

//Equivalente a lo anterior
var snacks = <String>[];
```

Lists


```
//Imprimos una lista
print(postres);
//Resultado: [Galletas,Flan,Tarta,Fruta]

//Accediendo a los valores y obteniendo índices
final primerElemento = postres[0];
final index = postres.indexOf('Galletas');

//Asignando , añadiendo y borrando
postres[0] = 'Oreo';
postres.add('Helado');
postres.remove('Helado'); //postres.removeAt(4);
```

Lists - Mutable and immutable

¿Qué significa mutable and immutable?
Básicamente, que se puede y no se puede modificar.

```
//Definiendo una mutable List.
var postres = [‘Galletas’, ‘Flan’, ‘Tarta’, ‘Fruta’];
postres = [];
postres = [‘Galletas’, ‘Flan’, ‘Tarta’, ‘Fruta’];

//Al ser una lista mutable me permite volver a asignar un valor a la variable lista.
```

Lists - Mutable and immutable


```
//Si la definimos como immutable,
final postres = ['Galletas', 'Flan', 'Tarta', 'Fruta'];
postres = [];                  // 🚫 No permitido
postres = ['Galletas'];        // 🚫 No permitido
postres = otraLista;           // 🚫 No permitido


final postres = ['Galletas', 'Flan', 'Tarta', 'Fruta'];
postres.remove('Flan');        // ✅ permitido
postres.remove('Fruta');       // ✅ permitido
postres.add('Helado');         // ✅ permitido
```

Lists - Mutable and immutable


```
//Con final podemos modificar el contenido de nuestra Lista
//¿Pero y si queremos que no se pueda modificar nada?
const postres = [‘Galletas’, ‘Flan’, ‘Tarta’, ‘Fruta’];
postres.remove(‘Flan’);    // 🚫 No permitido

//Existe otra manera de crear lista immutable pero no
//profundizaremos en ella
List.unmodificable(postre)
```

Lists - Properties


```
var postres = [‘Galletas’, ‘Flan’, ‘Tarta’, ‘Fruta’];

postres.first; //Galletas
postres.last; //Fruta

postres.isEmpty; //False
postres.isNotEmpty; //true

postres.length; //4
```

Lists - Looping


```
var postres = [‘Galletas’, ‘Flan’, ‘Tarta’, ‘Fruta’];

for(var postre in postres){
    print(postre);
}

postres.forEach( (postre) => print(postre));
postres.forEach(print); //Tear-off
```

Lists - Spread Operator


```
var galletas = [‘Oreo’, ‘Chiquilin’, ‘Maria’];
var golosinas = [‘Caramelo’, ‘Regaliz’];

const postres = [‘Helado’, ...galletas, ...golosinas];
print(postres);

//[Helado, Oreo, Chiquilin, Maria, Caramelo, Regaliz];
```

Existe el operador ...? para evitar el error con los nulos.

Lists - Collection if


```
const cacahueteAlergia = true;

var galletas = [‘Chiquilin’,
                ‘Maria’,
                if(!cacahueteAlergia) ‘Oreo’];
```

Lists - Collection for


```
var galletas = [‘Oreo’, ‘Chiquilin’, ‘Maria’];
var upperGalletas = [‘CHIPS AHOY’,
    for(var galleta in galletas) galleta.toUpperCase
    ];
```

Sets

Colecciones de datos que no ordenadas que contienen valores únicos.

Para definir un Set usamos {}

```
final Set<int> ejemploSet = {};
final ejemploSet = <int>{};
//Siempre tenemos que definir el tipo cuando la inicializamos a vacío, más adelante veremos el porqué.

final ejemploSet = {1,2,3,4};
```

Sets - Operaciones


```dart
//Comprobando el contenido.
final ejemploSet = {1,2,3,4};
ejemploSet.contains(1); //true
ejemploSet.contains(99); //false

//Añadiendo elementos
final someSet = <int>{};
someSet.add(42);
someSet.add(2112);
someSet.add(42); //¿Se añade?
print(someSet);
//Resultado es: {42,2112}
```

Sets - Operaciones


```dart
//Borrando elementos
someSet.remove(2112);

//Añadiendo múltiples elementos
final someSet = {0};
someSet.addAll([1, 2, 3, 4]);
print(someSet);
//Resultado es: {0,1,2,3,4}
```

Sets - Intersección y Unión


```
final setA = {8,2,3,1,4};
final setB = {1,6,5,4};

//Intersección
final intersection = setA.intersection(setB);
//{1,4}

//Unión
final union = setA.union(setB);
//{8,2,3,1,4,6,5}
```

Sets - Otras operaciones

A igual que las listas, Set incluye:

- collection if
- collection for
- for-in
- forEach
- spread operators

Maps

Colecciones de datos que almacenan pares de datos asociados con clave y valor. Las claves son únicas y los valores pueden ser de cualquier tipo incluso otras colecciones.

Para definir un Map también usamos {}

```
final Map<String,int> ejemploMap = {};
final ejemploMap = <String,int>{};
final ejemploMap = {}; //¿Es un Set o un Map?
//En dart el uso de {} se asocia por defecto a Maps
```

Maps


```
final inventario = {
    'tarta' : 20,
    'helado': 14,
}; //¿De qué tipo se definiría?
//Map<String,int>

final digitToWord = {
    1: 'uno',
    2: 'dos'
}; //¿De qué tipo se definiría?
//Map<int,String>
```

Maps - Operaciones


```dart
//Accediendo a elementos
final numeroTartas = inventario['tarta'];

//Añadiendo elementos
inventario['fruta'] = 4;

//Actualizando elementos
inventario['tarta'] = 20;

//Eliminando
inventario.remove('tarta');
```

Maps - Propiedades


```
inventario.isEmpty; //False
inventario.isNotEmpty; //True
inventario.length; //Devuelve el num elementos

print(inventario.keys); //Devuelve una lista con las claves
print(inventario.values); //Devuelve una lista con los
valores

inventario.containsKey('tarta'); //True
inventario.containsValue(20); //Comprueba si existe un valor
//que sea 20
```

Maps - Bucles

Si queremos iterar un Map debemos hacerlo a través de sus claves.

```
for(var item in inventario.keys){
    print(inventario[item]);
}
```

Usando forEach

```
inventario.forEach((key,value) => print(inventario[key]));
```

Maps - Ejercicio

1. Crea un mapa con las siguientes claves: nombre, profesión, país y ciudad. Rellénalo con tus datos.
2. Cambia tu ciudad a Lisboa y cambia también el país.
3. Itera los elementos e imprime todas las claves y valores.

Métodos de alto nivel

Son métodos que realizan operaciones sobre colecciones incluyendo transformaciones, filtrado y consolidación de elementos.

- map
- filter
- reduce

Métodos de alto nivel - **Map**

Devuelve un nuevo iterable, con el resultado que le asignemos en el retorno de la función que recibe como argumento (callback).

```
const numeros = [1,2,3,4];
final nCuadrados = numeros.map( (num) => num * num
).toList();

//El resultado será [1,4,9,16]
```

Where y firstWhere

Devuelve un iterable que cumple los criterios de filtrado pasados como parámetro en forma de función.

```
const numeros = [1,2,3,4];
final nCuadrados = numeros.where( (int num) => num > 2  );
//El resultado será [3,4]

const numeros = [1,2,3,4];
final nCuadrados = numeros.firstWhere(
    (int num) => num > 3, orElse: () => 0);
//El resultado será 4, en caso de que no se cumpliera el
resultado sería el devuelto en orElse.
```

Reduce

Combina todos los elementos aplicando la función que se pasa como parámetro.

```
const numeros = [1,2,3,4];
print(numeros.reduce( (sum, element) => sum + element  ));
//Resultado es: 10

const nombre = <String>['Anakin', 'Skywalker'];
String fullName = nombre.reduce(
    (fullName, value) => '$fullName $value');
print(fullName); //Anakin Skywalker
```

Fold

Igual que reduce pero nos permite definir un valor inicial.

```
const numeros = [1,2,3,4];
print(numeros.fold<int>(0,(int sum, element) => sum +
element ));

//Resultado es: 10
```

Reversed

Cambia el orden de los elementos de la colección

```
const numeros = [1,2,3,4];
print(numeros.reversed);

//Resultado es: [4,3,2,1]
```

Sort

Nos permite ordenar para ello utiliza Comparator por defecto y también se le puede pasar como parámetros una función que compare.

```
List<int> nums = [13, 2, -11];
nums.sort();
print(nums);  // [-11, 2, 13]
```

Sort


```
List<User> users =
    [User(name: 'Anakin', age: 6),
     User(name: 'Yoda', age: 900),
     User(name: 'Otro', age: 25)];

users.sort(
    (userA, userB) => userA.age >= userB.age ? 0 : 1);
print(users); // [Yoda, Otro, Anakin]
```

Any


```
List<User> users =
    [User(name: 'Anakin', age: 6),
     User(name: 'Yoda', age: 900),
     User(name: 'Otro', age: 25)];

users.any((user) => user.age > 18)
//Devuelve un true
```

RemoveWhere


```
List<User> users =
    [User(name: 'Anakin', age: 6),
     User(name: 'Yoda', age: 900),
     User(name: 'Otro', age: 25)];

users.removeWhere((user) => user.name == 'Yoda');
//[User(name: 'Anakin', age: 6), User(name: 'Otro', age: 25)]
```

Ejercicios

1. Escribe una función que tome como parámetro de entrada un texto y devuelva una colección con aquellos caracteres únicos que el texto contiene.

2. Función que tome como parámetro de entrada un párrafo y devuelva la frecuencia de veces que aparece cada carácter único.

Ejercicios

3. Realizar las siguientes instrucciones:
 - Crear una clase llamada User con dos atributos: id y name.
- Crear una lista con tres usuarios con datos.
- Crear una función que convierta la lista de objetos en una lista de mapas donde las keys serán id y name el valor.