

---

# Dart Collection

Eko Kurniawan Khannedy

# Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 11+ years experiences
- [www.programmerzamannow.com](http://www.programmerzamannow.com)
- [youtube.com/c/ProgrammerZamanNow](https://youtube.com/c/ProgrammerZamanNow)



---

# Eko Kurniawan Khannedy

- Telegram : [@khannedy](https://t.me/khannedy)
- Facebook : [fb.com/ProgrammerZamanNow](https://www.facebook.com/ProgrammerZamanNow)
- Instagram : [instagram.com/programmerzamannow](https://www.instagram.com/programmerzamannow)
- Youtube : [youtube.com/c/ProgrammerZamanNow](https://www.youtube.com/c/ProgrammerZamanNow)
- Telegram Channel : [t.me/ProgrammerZamanNow](https://t.me/ProgrammerZamanNow)
- Email : echo.khannedy@gmail.com

---

# Sebelum Belajar

- Dart Dasar
- Dart OOP
- Dart Generic
- Dart Packages
- Sudah Mengikuti Kelas Git dari Programmer Zaman Now

---

# Agenda

- Pengenalan Dart Collection
- Iterable, Iterator
- List, Linked List
- Set, Hash Set, Tree Set, Linked Set
- Map, Hash Map, Linked Map, Tree Map
- Dan Lain-Lain

---

# Pengenalan Collection

---

# Pengenalan Collection

- Sebelumnya di materi Dart Dasar, kita sudah berkenalan dengan tipe data seperti List, Set dan Map
- Semua tipe data tersebut disebut Collection, atau bisa dibilang adalah kumpulan data
- Di Dart sendiri, terdapat package yang khusus menyediakan class-class untuk tipe data Collection
- Pada kelas ini, kita akan bahas tuntas tentang tipe data Collection tersebut
- <https://api.dart.dev/stable/2.17.6/dart-collection/dart-collection-library.html>

---

# Membuat Project

---

# Membuat Dart Project

```
dart create --template=console-simple belajar_dart_collection
```

---

# Iterable

---

# Iterable

- Sebelum kita bahas tentang Dart Collection, ada satu class yang teman-teman wajib tau, yaitu Iterable
- Iterable adalah parent class dari class-class Collection di Dart
- Sederhananya, Iterable adalah kumpulan data yang bisa diakses secara sequential atau satu per satu
- <https://api.dart.dev/stable/2.17.6/dart-core/Iterable-class.html>
- List dan Set adalah class turunan dari Iterable

---

# Iterator

---

# Iterator

- Salah satu fitur di Dart yang bisa digunakan untuk melakukan iterasi data, atau mengakses data di Iterable satu persatu adalah menggunakan for in
- Jika tipe data memiliki property dengan nama iterator dan tipe Iterator, secara otomatis kita bisa menggunakan perulangan for in
- Contohnya di Iterable terdapat property iterator, oleh karena itu kita bisa mengakses data di Iterable menggunakan for in
- <https://api.dart.dev/stable/2.17.6/dart-core/Iterator-class.html>

## Kode : Iterator For In



```
 iterable.dart
1 void main() {
2     var names = ['Seth', 'Logan', 'Mack'];
3
4     for (var value in names) {
5         print(value);
6     }
7 }
8
```

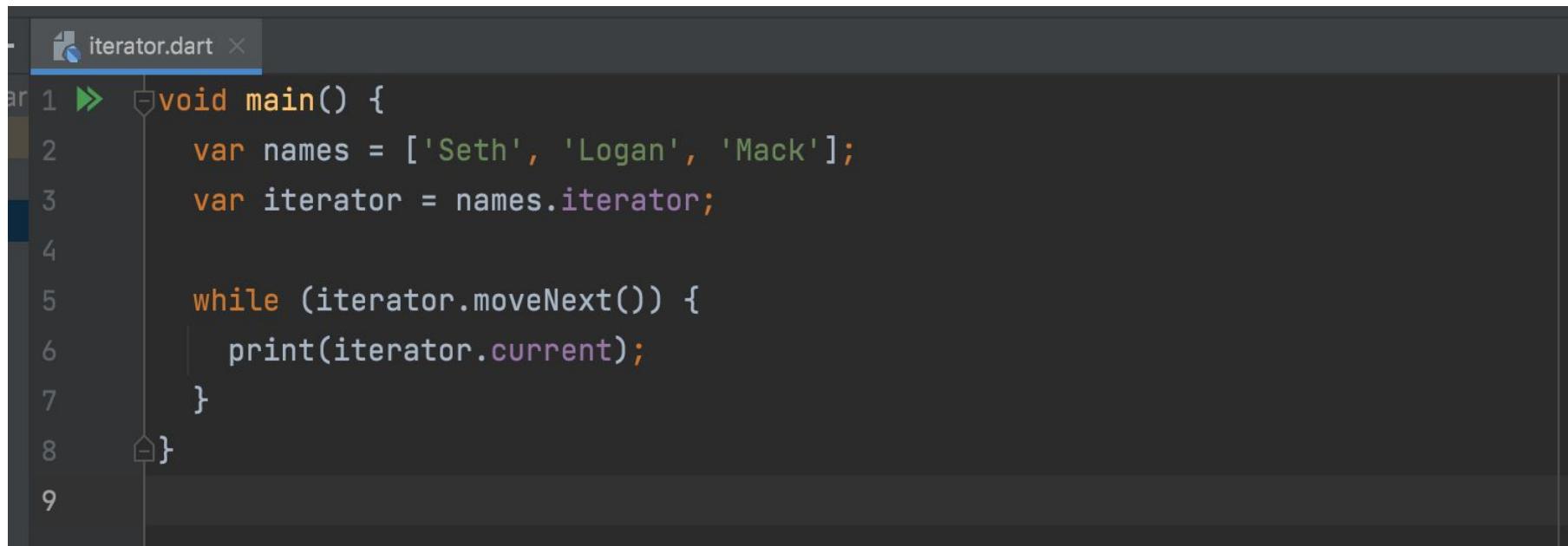
The image shows a screenshot of a code editor with a dark theme. A file named 'iterable.dart' is open. The code contains a main function that prints three names: 'Seth', 'Logan', and 'Mack'. The code is numbered from 1 to 8. A vertical line with a small square icon at the top is positioned to the left of the first line of code, likely indicating the current line or a break point.

---

## Iterasi Manual

- Sebenarnya, ketika kita menggunakan perulangan for in, secara tidak langsung kita melakukan perulangan terhadap Iterator menggunakan method-method yang tersedia di Iterator
- Iterator.MoveNext() digunakan untuk berpindah ke data selanjutnya
- Iterator.Current digunakan untuk mendapatkan data saat ini

## Kode : Iterasi Manual



A screenshot of a code editor displaying a Dart file named `iterator.dart`. The code implements manual iteration over a list of names.

```
1 void main() {
2     var names = ['Seth', 'Logan', 'Mack'];
3     var iterator = names.iterator;
4
5     while (iterator.moveNext()) {
6         print(iterator.current);
7     }
8 }
```

---

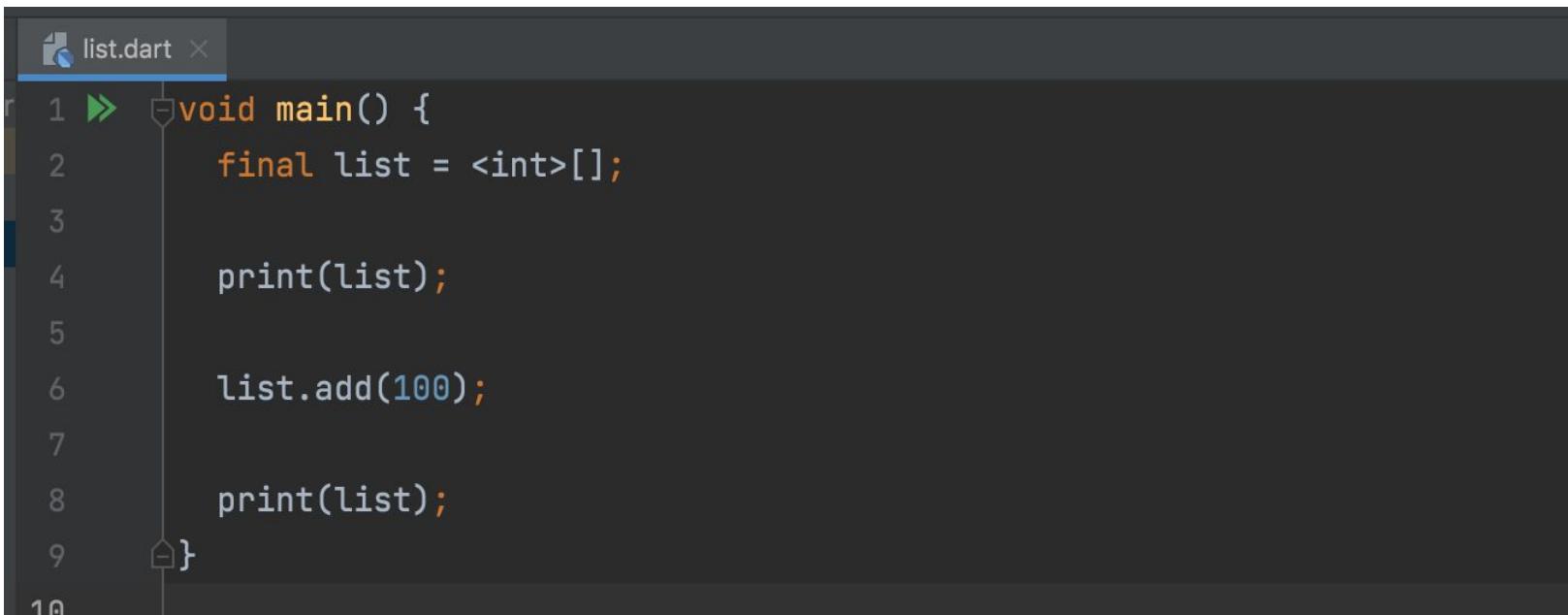
# List

---

# List

- List adalah tipe data yang berisikan kumpulan data yang memiliki index angka
- Tipe data List sudah kita bahas di materi Dart Dasar
- Saat membuat List, kita bisa buat dengan ukuran fix (tidak bisa berubah), atau bisa bertambah secara otomatis seiring penambahan data ke List
- <https://api.dart.dev/stable/2.17.6/dart-core/List-class.html>

## Kode : Growable List



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "list.dart". The code itself is as follows:

```
list.dart
1 ➤ void main() {
2     final list = <int>[];
3
4     print(list);
5
6     list.add(100);
7
8     print(list);
9 }
10
```

The code defines a main function that creates an empty list of integers, prints it, adds the number 100 to it, and then prints it again.

## Kode : Fixed List



The screenshot shows a code editor window titled "list.dart". The code is a Dart program with the following content:

```
list.dart
1 > void main() {
2     final list = List<int>.filled(10, 0);
3
4     print(list);
5
6     list.add(100); // error
7 }
8
```

The file is currently running, as indicated by the play button icon in the top left corner. The code defines a main function that creates a list of 10 integers filled with 0, prints the list, and then attempts to add 100 to the list, which would result in an error due to the list being fixed.

---

# Linked List

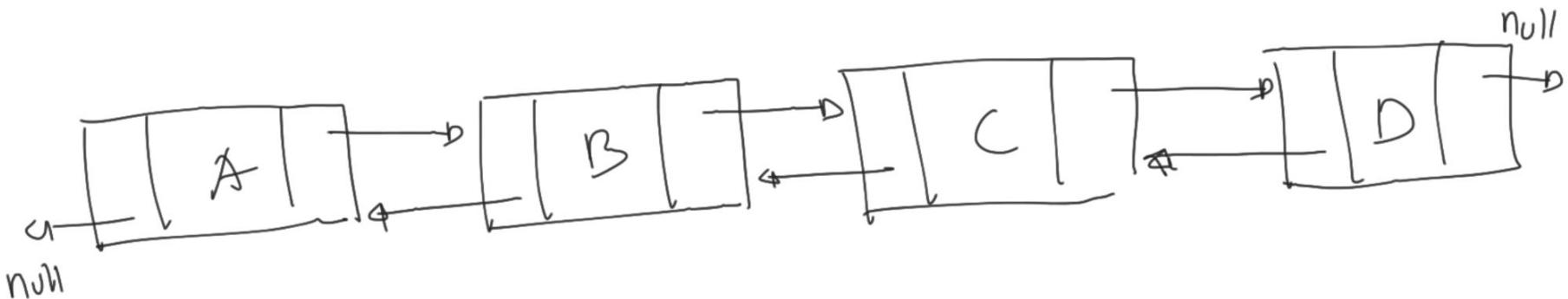
---

## Linked List

- List di Dart mirip seperti Array di bahasa pemrograman lain, untuk mengubah atau mengakses datanya kita menggunakan Index
- Di Dart juga tersedia collection bernama Linked List, ini adalah implementasi dari struktur data Double Linked List
- <https://api.dart.dev/stable/2.17.6/dart-collection/LinkedList-class.html>

---

## Diagram : Double Linked List



---

## List vs Linked List

Operasi	List	Linked List
tambah data	Cepat jika kapasitas Array masih cukup, lambat jika sudah penuh	Cepat karena hanya menambah node di akhir
ambil data	Cepat karena tinggal gunakan index array	Lambat karena harus di cek dari node awal sampai ketemu index nya
ubah data	Cepat karena tinggal gunakan index array	Lambat karena harus di cek dari node awal sampai ketemu
hapus data	Lambat karena harus menggeser data di belakang yang dihapus	Cepat karena tinggal ubah prev dan next di node sebelah yang dihapus

---

## Fokus Linked List

- Fokus Linked List adalah pada performa penambahan data dan penghapusan data yang cepat, oleh karena ini di Linked List tidak terdapat operasi pengubahan data menggunakan Index seperti di List
- Linked List juga turunan langsung dari Iterable, bukan List

---

# Linked List Entry

- Node di Linked List direpresentasikan dalam class LinkedListEntry
- Oleh karena itu, jika kita ingin membuat Node, kita perlu membuat class turunan LinkedListEntry
- Pada class tersebut sudah disediakan implementasi property next dan prev nya
- <https://api.dart.dev/stable/2.17.6/dart-collection/LinkedListEntry-class.html>

---

## Kode : Class String Entry

```
class StringEntry extends LinkedListEntry<StringEntry> {  
    String value;  
    StringEntry(this.value);  
}
```

---

## Kode : Linked List

```
void main() {  
    var linkedList = LinkedList<StringEntry>();  
    linkedList.addAll(  
        [StringEntry('Eko'), StringEntry('Kurniawan'), StringEntry('Khannedy')]);  
  
    for (var value in linkedList) {  
        print(value.value);  
    }  
}
```

---

# Unmodifiable List

---

## Unmodifiable List

- Collection List, walaupun kita buat dalam bentuk Fix atau Growable, data di dalam List, tetap bisa kita modifikasi
- Dart menyediakan collection bernama Unmodifiable List, yaitu List yang setelah dibuat, data di dalamnya tidak bisa diubah lagi
- Cara menggunakan Unmodifiable List adalah dengan cara membungkus List yang sudah kita buat sebelumnya
- <https://api.dart.dev/stable/2.17.6/dart-collection/UnmodifiableListView-class.html>

## Kode : Unmodifiable List



The screenshot shows a code editor window with a dark theme. The file is named "unmodifiable\_list.dart". The code demonstrates creating an unmodifiable list from a regular list and attempting to add an element to it.

```
1 import 'dart:collection';
2
3 void main(){
4     final list = [1, 2, 3];
5     final unmodifiableList = UnmodifiableListView(list);
6
7     unmodifiableList.add(100); // error
8 }
```

---

# Linked Hash Set

---

# Set

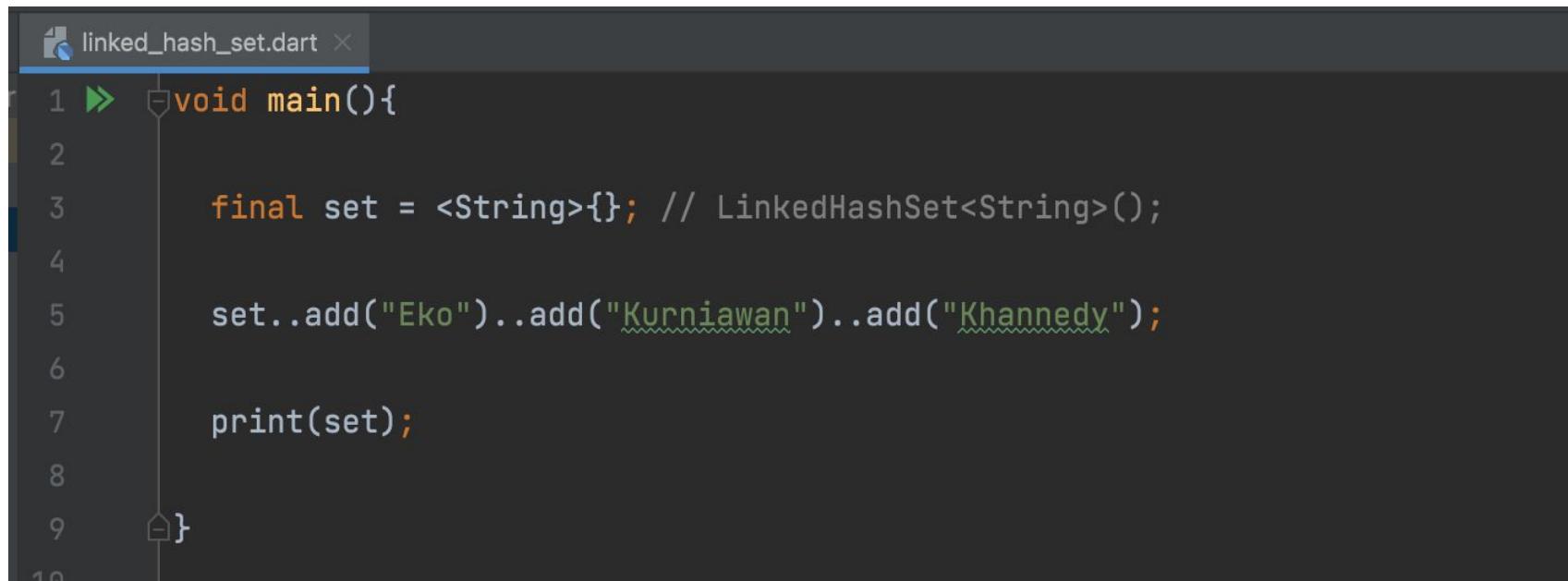
- Set sudah pernah dibahas di materi Dart Dasar
- Set adalah collection yang berisikan kumpulan data unique, ketika kita menambahkan data yang sudah ada, maka otomatis data tersebut akan diabaikan
- <https://api.dart.dev/stable/2.17.6/dart-core/Set-class.html>

---

# Linked Hash Set

- Saat kita membuat Set, implementasi default dari Set sendiri adalah class LinkedHashSet
- Seperti terlihat dari namanya, LinkedHashSet menggunakan struktur data double linked list sebagai implementasinya
- Hal ini menjadikan, urutan data di Set sesuai dengan urutan ketika kita memasukkan data ke Set
- <https://api.dart.dev/stable/2.17.6/dart-collection/LinkedHashSet-class.html>

## Kode : Linked Hash Set



A screenshot of a code editor showing a Dart file named `linked_hash_set.dart`. The code defines a main function that creates a `LinkedHashSet`, adds three strings to it, and then prints the set.

```
linked_hash_set.dart
void main(){
    final set = <String>{}; // LinkedHashSet<String>();
    set..add("Eko")..add("Kurniawan")..add("Khannedy");
    print(set);
}
```

---

# Hash Set

---

# Hash Set

- Hash Set adalah implementasi Set yang tidak menggunakan struktur data double linked list
- Hal ini menyebabkan urutan di Hash Set tidak tentu, karena tergantung dari hash code data yang kita masukkan
- Hash Set membuat proses insert data menjadi cepat karena tidak perlu melakukan pengecekan satu per satu di double linked list, cukup langsung menggunakan hash code
- <https://api.dart.dev/stable/2.17.6/dart-collection/HashSet-class.html>

# Kode : Hash Set



A screenshot of a code editor showing a Dart file named `hash_set.dart`. The code defines a main function that creates a `HashSet<String>`, adds three strings to it, and then prints the set. The code is numbered from 1 to 12.

```
1 import 'dart:collection';
2
3 void main(){
4
5     final set = HashSet<String>();
6
7     set..add("Eko")..add("Kurniawan")..add("Khannedy");
8
9     print(set);
10
11 }
12
```

---

# Splay Tree Set

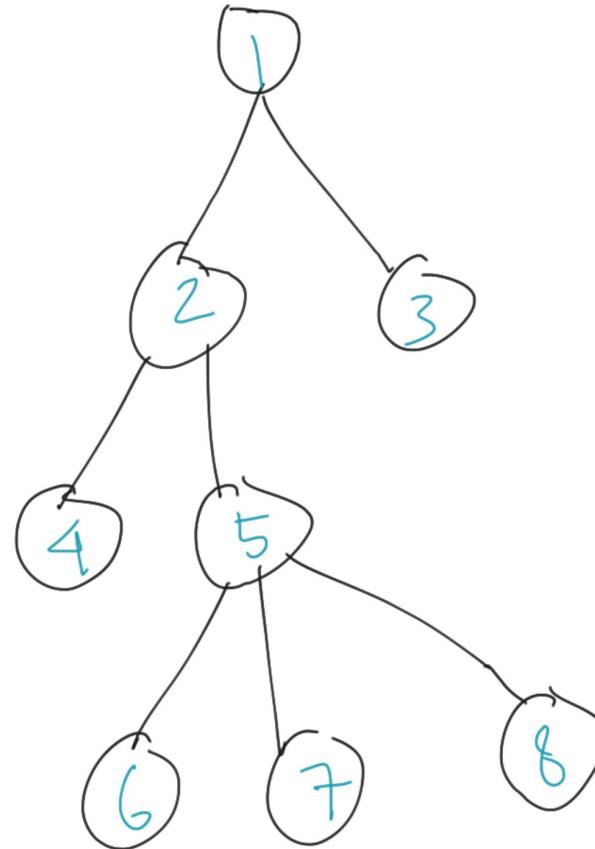
---

## Splay Tree Set

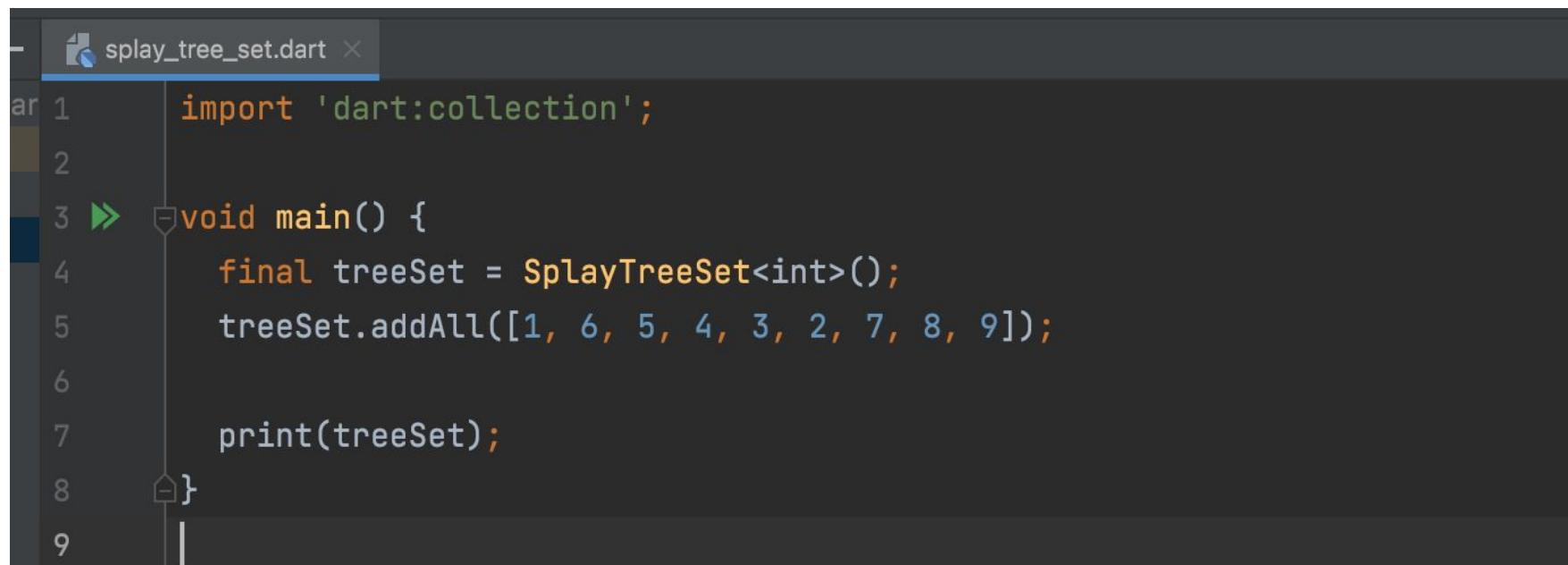
- Splay Tree Set merupakan implementasi dari Set yang menggunakan struktur data Tree
- Hal ini menyebabkan data di Splay Tree Set akan secara otomatis berurut sesuai comparator nya, defaultnya adalah ascending
- <https://api.dart.dev/stable/2.17.6/dart-collection/SplayTreeSet-class.html>

---

## Diagram : Tree



# Kode : Splay Tree Set



A screenshot of a code editor showing a Dart file named `splay_tree_set.dart`. The code defines a main function that creates a `SplayTreeSet<int>`, adds integers from 1 to 9 to it, and then prints the set.

```
1 import 'dart:collection';
2
3 void main() {
4     final treeSet = SplayTreeSet<int>();
5     treeSet.addAll([1, 6, 5, 4, 3, 2, 7, 8, 9]);
6
7     print(treeSet);
8 }
9
```

---

# Comparable

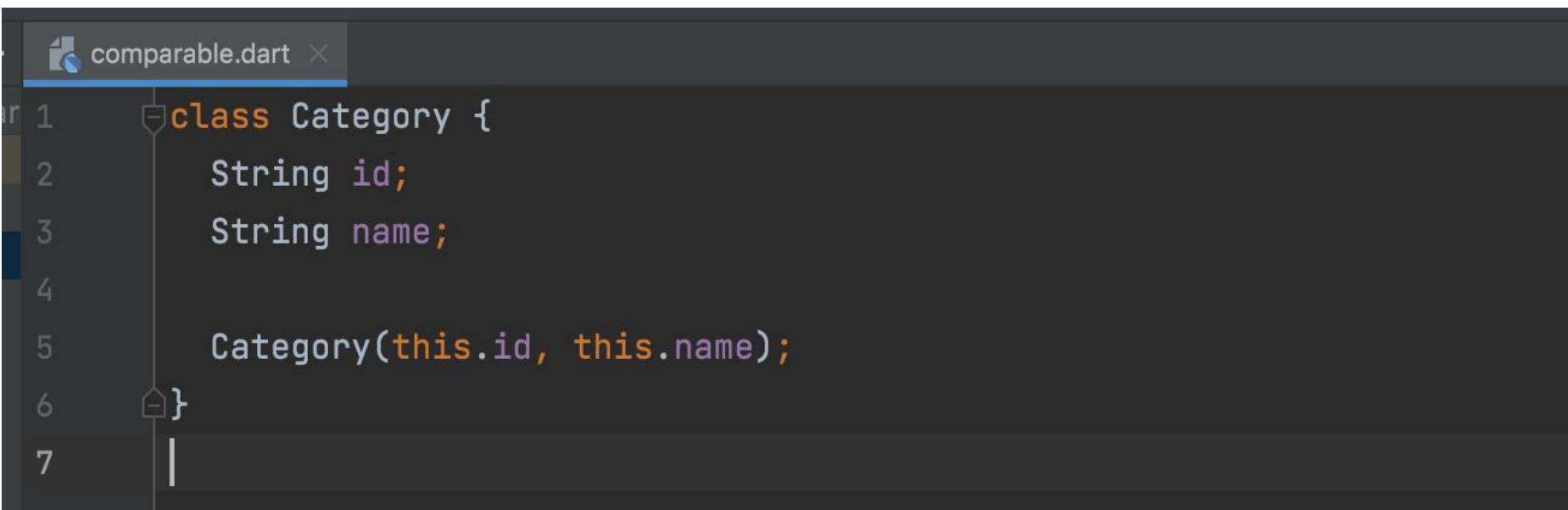
---

# Comparable

- Comparable adalah sebuah kontrak yang digunakan untuk membuat tipe data yang bisa diurutkan
- Hampir semua tipe data di Dart implement kontrak Comparable, seperti number, string, boolean, dan lain-lain, oleh karena itu, data-data tersebut bisa diurutkan secara otomatis ketika menggunakan SplayTreeSet misalnya
- Bagaimana jika kita ingin membuat class sendiri? Secara default, class kita tidak bisa diurutkan datanya oleh SplayTreeSet, kita wajib implement kontrak Comparable
- <https://api.dart.dev/stable/2.17.6/dart-core/Comparable-class.html>

---

## Kode : Class Category



```
comparable.dart ✘
1 class Category {
2     String id;
3     String name;
4
5     Category(this.id, this.name);
6 }
7 |
```

The image shows a screenshot of a code editor with a dark theme. A file named "comparable.dart" is open. The code defines a class named "Category" with two properties: "id" and "name", both of type "String". It also includes a constructor "Category(this.id, this.name)". Lines 1 through 7 are numbered on the left side.

---

## Kode : Splay Tree Set Error

```
void main(){
    final treeSet = SplayTreeSet<Category>();
    treeSet.add(Category("2", "Category 2"));
    treeSet.add(Category("1", "Category 1"));
    treeSet.add(Category("3", "Category 3"));

    print(treeSet);
}
```

---

## Kode : Implement Comparable

```
class Category implements Comparable<Category> {
    String id;
    String name;

    Category(this.id, this.name);

    @Override
    int compareTo(Category other) {
        return id.compareTo(other.id);
    }
}
```

---

# Comparator

---

# Comparator

- Secara default, ketika mengurutkan data, SplayTreeSet akan menggunakan Comparable yang terdapat pada data nya
- Bagaimana jika kita ingin memodifikasi cara melakukan pengurutan data nya? Tapi tidak mau mengubah class data tersebut? Atau bahkan tidak bisa mengubahnya, seperti tipe data number, boolean, String dan lain-lain
- Pada kasus ini, kita bisa membuat Comparable, yaitu function yang bisa kita gunakan untuk menentukan cara melakukan pengurutan data
- <https://api.dart.dev/stable/2.17.6/dart-core/Comparator.html>

# Kode : Comparator



A screenshot of a code editor showing a Dart file named `splay_tree_set.dart`. The code defines a main function that creates a `SplayTreeSet` with a custom comparator and adds elements [1, 6, 5, 4, 3, 2, 7, 8, 9] to it, then prints the set.

```
1 import 'dart:collection';
2
3 void main() {
4     final treeSet = SplayTreeSet<int>((first, second) => second.compareTo(first));
5     treeSet.addAll([1, 6, 5, 4, 3, 2, 7, 8, 9]);
6
7     print(treeSet);
8 }
9
```

---

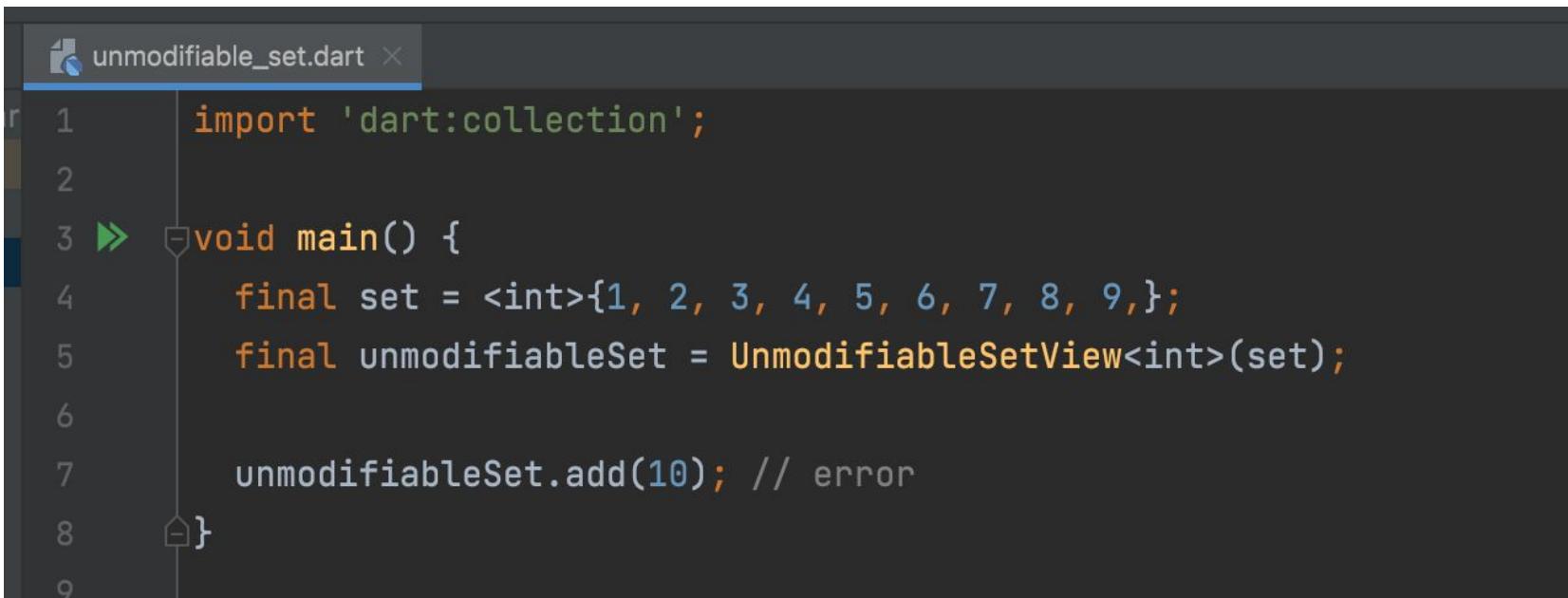
# Unmodifiable Set

---

## Unmodifiable Set

- Sama seperti List, di Set pun terdapat class Unmodifiable Set, yang digunakan untuk membungkus Set agar tidak bisa dimodifikasi lagi
- <https://api.dart.dev/stable/2.17.6/dart-collection/UnmodifiableSetView-class.html>

## Kode : Unmodifiable Set



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "unmodifiable\_set.dart". The code itself is as follows:

```
1 import 'dart:collection';
2
3 void main() {
4     final set = <int>{1, 2, 3, 4, 5, 6, 7, 8, 9,};
5     final unmodifiableSet = UnmodifiableSetView<int>(set);
6
7     unmodifiableSet.add(10); // error
8 }
9
```

The code imports the `'dart:collection'` library. It defines a `main` function that creates a mutable `set` containing integers from 1 to 9. It then creates an `unmodifiableSet` view of this set using the `UnmodifiableSetView` class. Finally, it attempts to add the integer 10 to the `unmodifiableSet`, which results in a runtime error.

---

# List Queue



# Queue

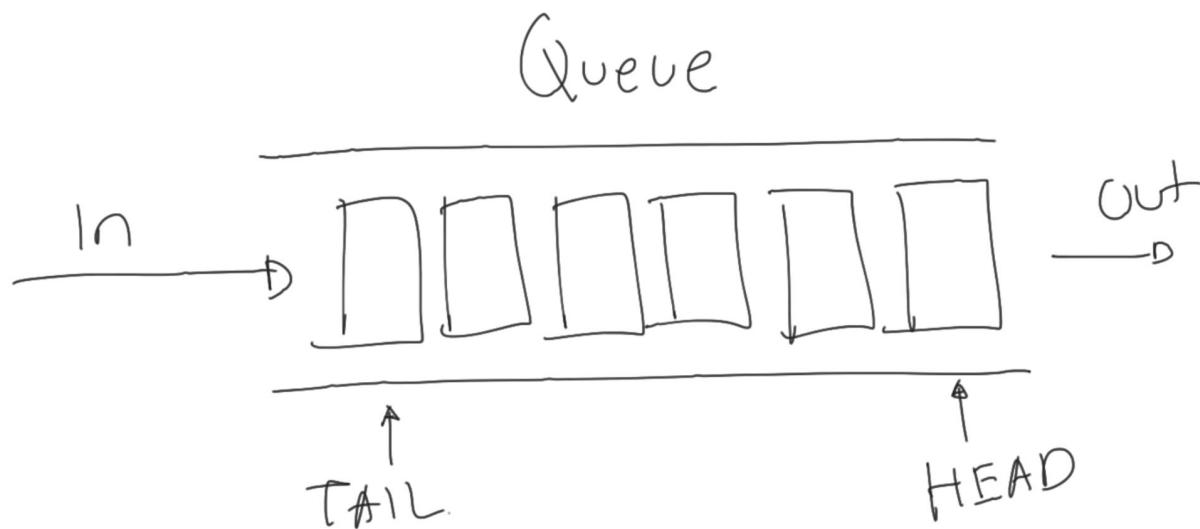
- Queue adalah collection implementasi dari struktur data Stack (tumpukan) atau Queue (antrian)
- Queue mirip seperti List, namun yang membedakan, pada Queue, modifikasi data bisa dilakukan di depan (HEAD) atau di belakang (TAIL)
- <https://api.dart.dev/stable/2.17.6/dart-collection/Queue-class.html>

---

## List Queue

- List Queue merupakan implementasi default dari Queue di Dart
- Saat kita membuat object Queue, sebenarnya kita membuat List Queue
- <https://api.dart.dev/stable/2.17.6/dart-collection/ListQueue-class.html>

## Diagram : Queue



# Kode : Queue

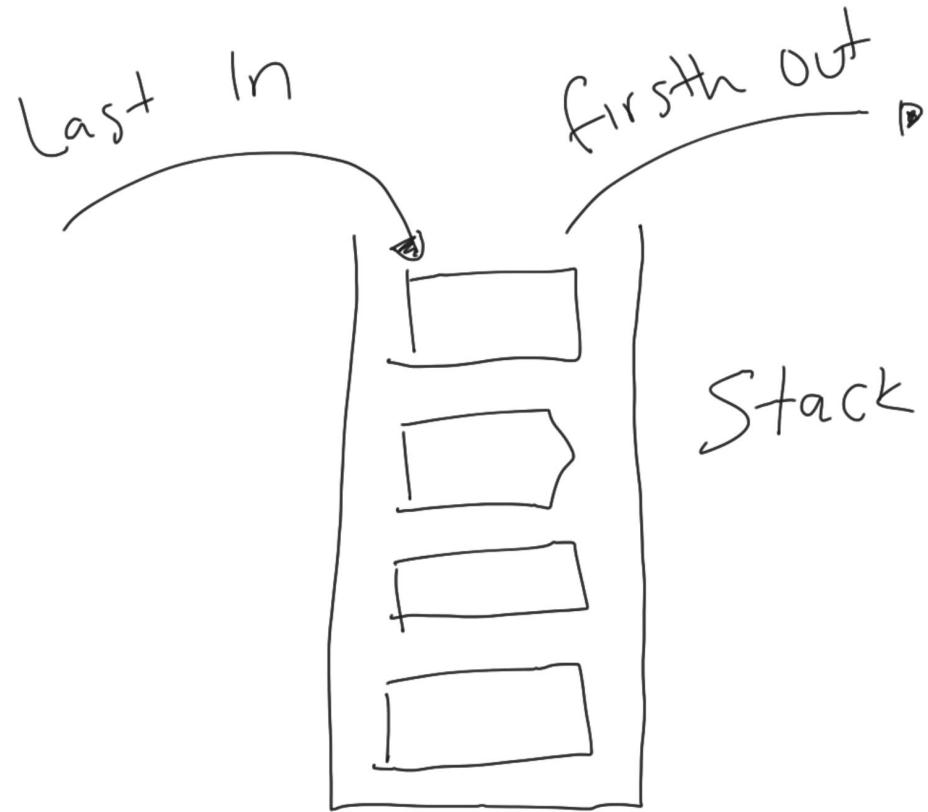


A screenshot of a code editor showing a Dart file named `queue.dart`. The code demonstrates the use of the `Queue` class from the `dart:collection` library. It creates a queue, adds three elements ("Eko", "Kurniawan", and "Khannedy") using `addLast`, and then prints each element removed from the front of the queue using `removeFirst` three times.

```
queue.dart
1 import 'dart:collection';
2
3 void main(){
4     final queue = Queue<String>();
5
6     queue.addLast("Eko");
7     queue.addLast("Kurniawan");
8     queue.addLast("Khannedy");
9
10    print(queue.removeFirst());
11    print(queue.removeFirst());
12    print(queue.removeFirst());
13}
14
```

---

## Diagram : Stack



# Kode : Stack



A screenshot of a code editor showing a Dart file named `stack.dart`. The code implements a stack using a `Queue` from the `dart:collection` package. It adds three names to the stack and then removes them in a last-in, first-out (LIFO) manner.

```
1 import 'dart:collection';
2
3 void main(){
4     final stack = Queue<String>();
5
6     stack.addLast("Eko");
7     stack.addLast("Kurniawan");
8     stack.addLast("Khannedy");
9
10    print(stack.removeLast());
11    print(stack.removeLast());
12    print(stack.removeLast());
13}
14
```

---

# Double Linked Queue

---

## Double Linked Queue

- Double Linked Queue merupakan implementasi dari Queue dengan struktur data Double Linked List
- Sebenarnya penggunaan Double Linked Queue sangat cocok untuk queue, karena struktur data Double Linked List sangat cepat untuk modifikasi data di awal dan akhir, sehingga cocok untuk Queue ataupun Stack
- <https://api.dart.dev/stable/2.17.6/dart-collection/DoubleLinkedQueue-class.html>

# Kode : Double Linked Queue

```
double_linked_queue.dart
import 'dart:collection';

void main(){
    final stack = DoubleLinkedQueue<String>();

    stack.addLast("Eko");
    stack.addLast("Kurniawan");
    stack.addLast("Khannedy");

    print(stack.removeLast());
    print(stack.removeLast());
    print(stack.removeLast());
}
```

---

# Iterable Method

---

# Iterable Method

- Sampai saat ini, kita hanya membahas tentang class-class yang terdapat di Dart Collection, namun belum membahas tentang fitur method apa saja yang dimiliki oleh Dart Collection
- Sebenarnya di dalam class Iterable, sudah banyak method yang tersedia untuk bisa kita gunakan ketika butuh melakukan operasi terhadap data collection
- Di materi-materi selanjutnya, kita akan coba bahas lebih detail tentang Iterable Method tersebut
- <https://api.dart.dev/stable/2.17.6/dart-core/Iterable-class.html>

---

# Check Method

---

## Check Method

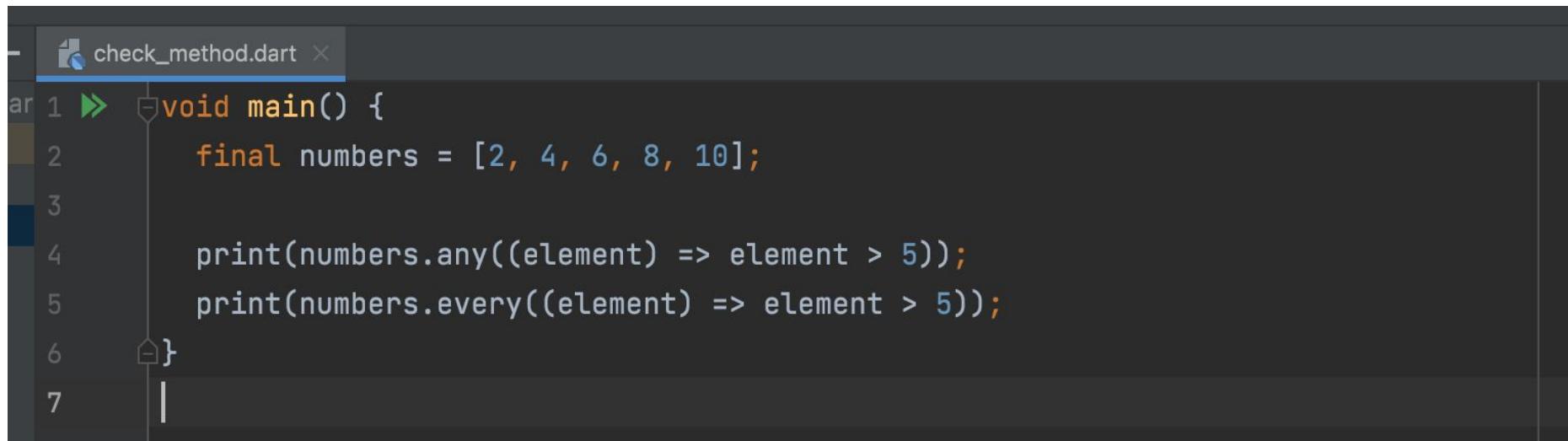
- Salah satu yang biasa kita lakukan saat menggunakan collection (List, Set, Queue dan lain-lain), adalah melakukan pengecekan data
- Iterable memiliki beberapa method untuk melakukan pengecekan data

---

## Check Method

Method	Keterangan
any(bool test(E)) : bool	Mengecek apakah terdapat data yang sesuai dengan test function
every(bool test(E)) : bool	Mengecek apakah semua data sesuai dengan test function
contains(E) : bool	Mengecek apakah data E terdapat di iterable

## Kode : Check Method



A screenshot of a code editor showing a Dart file named "check\_method.dart". The code defines a main function that prints whether any element in the array is greater than 5 and whether all elements are greater than 5.

```
check_method.dart
void main() {
    final numbers = [2, 4, 6, 8, 10];
    print(numbers.any((element) => element > 5));
    print(numbers.every((element) => element > 5));
}
```

---

# Filter Method

---

## Filter Method

- Iterable juga memiliki banyak sekali method untuk melakukan filtering data yang terdapat di Iterable

---

## Filter Method (1)

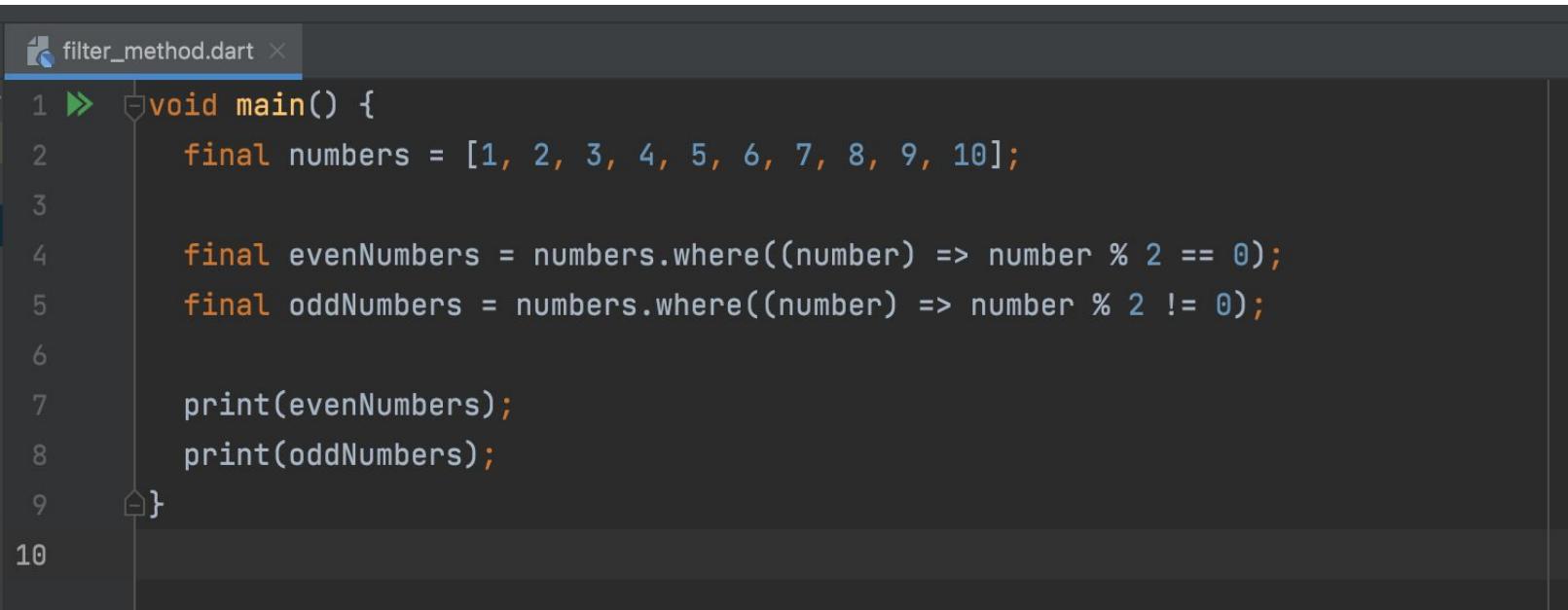
Method	Keterangan
firstWhere(bool test(E), E orElse): E	Mengambil data pertama yang sesuai dengan kondisi test, jika tidak ada, maka hasilnya data orElse
lastWhere(bool test(E), E orElse): E	Mengambil data terakhir yang sesuai dengan kondisi test, jika tidak ada, maka hasilnya data orElse
singleWhere(bool test(E), E orElse): E	Memastikan hanya ada satu data yang sesuai kondisi test, jika tidak ada, maka hasilnya data orElse, jika lebih dari satu, maka akan throw error

## Filter Method (2)

---

Method	Keterangan
skip(count) : Iterable<E>	Membuat iterable baru dengan menghapus data di awal sejumlah count
skipWhile(bool test(E)) : Iterable<E>	Membuat iterable baru dengan menghapus data di awal selama kondisi test terpenuhi
take(count) : Iterable<E>	Membuat iterable baru dengan hanya mengambil sejumlah count di awal
takeWhile(bool test(E)) : Iterable<E>	Membuat iterable baru dengan mengambil data di awal selama kondisi test terpenuhi
where(bool test(E))	Membuat iterable baru dengan semua data yang sesuai kondisi test

## Kode : Filter Method



A screenshot of a code editor showing a Dart file named `filter_method.dart`. The code demonstrates the use of the `where` method to filter elements from a list based on a condition. The code is as follows:

```
filter_method.dart
1 void main() {
2     final numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
3
4     final evenNumbers = numbers.where((number) => number % 2 == 0);
5     final oddNumbers = numbers.where((number) => number % 2 != 0);
6
7     print(evenNumbers);
8     print(oddNumbers);
9 }
10
```

---

# Transform Method

---

## Transform Method

- Iterable juga memiliki method untuk melakukan transformasi (mengubah) data menjadi data baru

# Transform Method (1)

---

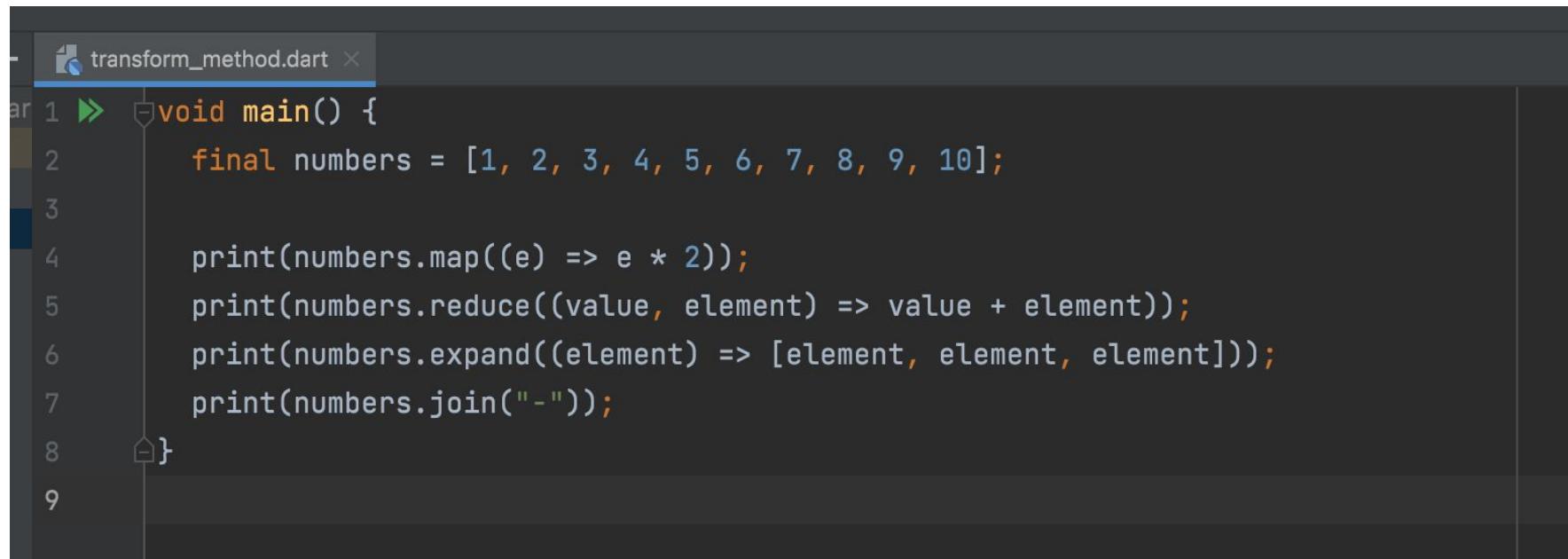
Method	Keterangan
expand(Iterable<T> toElements(E)) : Iterable<t>	Mengubah tiap element menjadi Iterable<T> dan menggabungkan semuanya
map(T toElement(E)) : Iterable<T>	Mengubah tiap element menggunakan function toElement
join(separator) : String	Mengubah element menjadi string, lalu menggabungkan dengan separator

## Transform Method (2)

---

Method	Keterangan
<code>fold(T initial, T combine(T, E)) : T</code>	Mengubah Iterable dengan cara melakukan iterasi satu persatu element dari mulai data initial, lalu hasil iterasi dikirim ke iterasi selanjutnya
<code>reduce(E combine(E, E)): E</code>	Sama dengan fold, namun hasilnya tetap tipe data yang sama dengan element

## Kode : Transform Method



A screenshot of a code editor showing a Dart file named `transform_method.dart`. The code demonstrates various transform methods on a list of integers. The code is as follows:

```
transform_method.dart
1 void main() {
2     final numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
3
4     print(numbers.map((e) => e * 2));
5     print(numbers.reduce((value, element) => value + element));
6     print(numbers.expand((element) => [element, element, element]));
7     print(numbers.join("-"));
8 }
9
```

---

# Convert Method

---

## Convert Method

- Iterable juga memiliki method yang bisa digunakan untuk melakukan konversi tipe data Collection menjadi tipe data lain

# Convert Method (1)

---

Method	Keterangan
toSet() : Set<E>	Mengubah Iterable menjadi Set
toList({growable: bool}) : List<E>	Mengubah Iterable menjadi List
toString() : String	Mengubah Iterable menjadi String

## Kode : Convert Method

```
convert_method.dart ×
1 ➤ void main() {
2     var numbers = [1, 2, 2, 3, 4, 4, 5, 6, 7, 8, 9, 10];
3     var numberSet = numbers.toSet();
4
5     print(numbers);
6     print(numberSet);
7 }
|
```

---

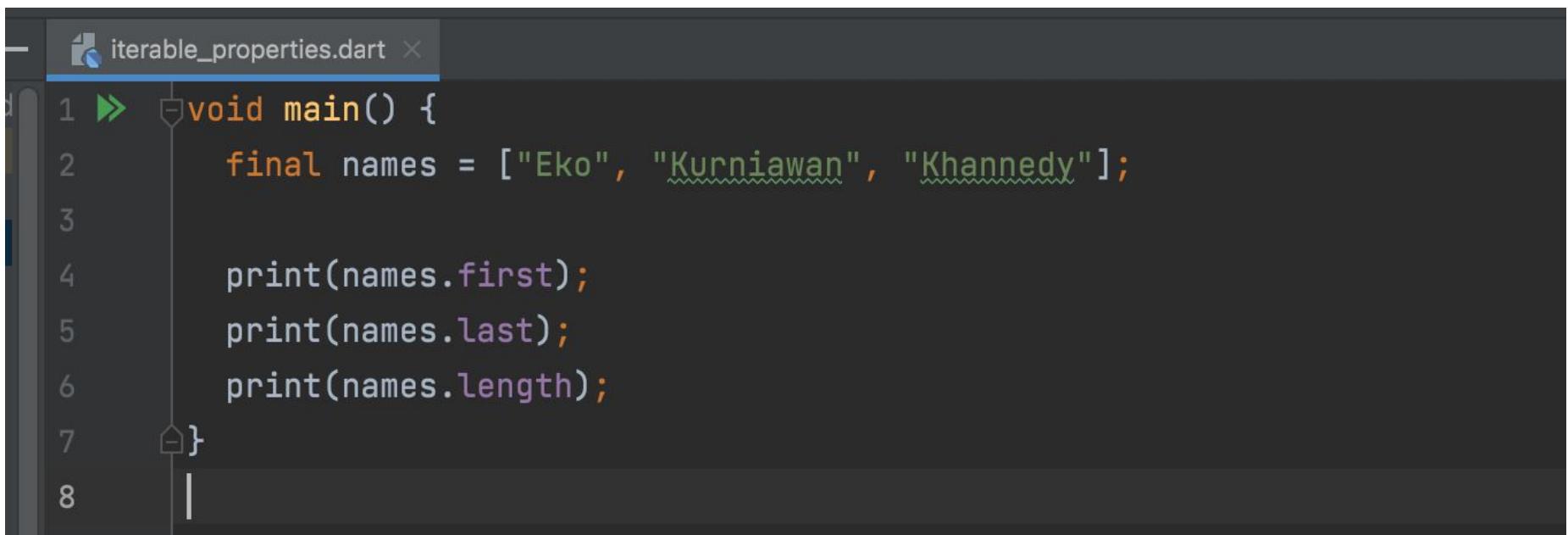
# Iterable Properties

---

# Iterable Properties

- Selain banyak sekali method yang dimiliki oleh Iterable
- Iterable juga memiliki banyak sekali property yang bisa kita gunakan untuk mendapatkan informasi dan data dari Iterable
- <https://api.dart.dev/stable/2.17.6/dart-core/Iterable-class.html#instance-properties>

## Kode : Iterable Properties



A screenshot of a code editor showing a Dart file named `iterable_properties.dart`. The code defines a `main` function that prints the first, last, and length of a list of names.

```
1 >> void main() {
2     final names = ["Eko", "Kurniawan", "Khannedy"];
3
4     print(names.first);
5     print(names.last);
6     print(names.length);
7 }
|
```

---

# List Method

---

## List Method

- List dan Set karena turunan dari Iterable, secara otomatis dapat menggunakan semua method di Iterable
- Namun karena sifat dari List dan Set itu berbeda, List sendiri memiliki method lain yang khusus untuk List
- <https://api.dart.dev/stable/2.17.6/dart-core/List-class.html#instance-methods>

## Kode : List Method

```
list_method.dart
1 void main() {
2     final names = ["Eko", "Khannedy"];
3     names.insert(1, "Kurniawan");
4
5     print(names);
6 }
```

---

## List Operator

- Selain method, List juga memiliki banyak Operator
- <https://api.dart.dev/stable/2.17.6/dart-core/List-class.html#operators>

## Kode : List Operator



A screenshot of a code editor showing a Dart file named `list_operator.dart`. The code demonstrates the addition operator for lists. It defines two final lists: `names` containing "Eko" and "Khannedy", and `authors` containing "Programmer", "Zaman", and "Now". It then creates a new list `combine` by adding `names` and `authors` together. Finally, it prints the combined list.

```
list_operator.dart
void main(){
    final names = ["Eko", "Khannedy"];
    final authors = ["Programmer", "Zaman", "Now"];

    final combine = names + authors;

    print(combine);
}
```

---

# Set Method

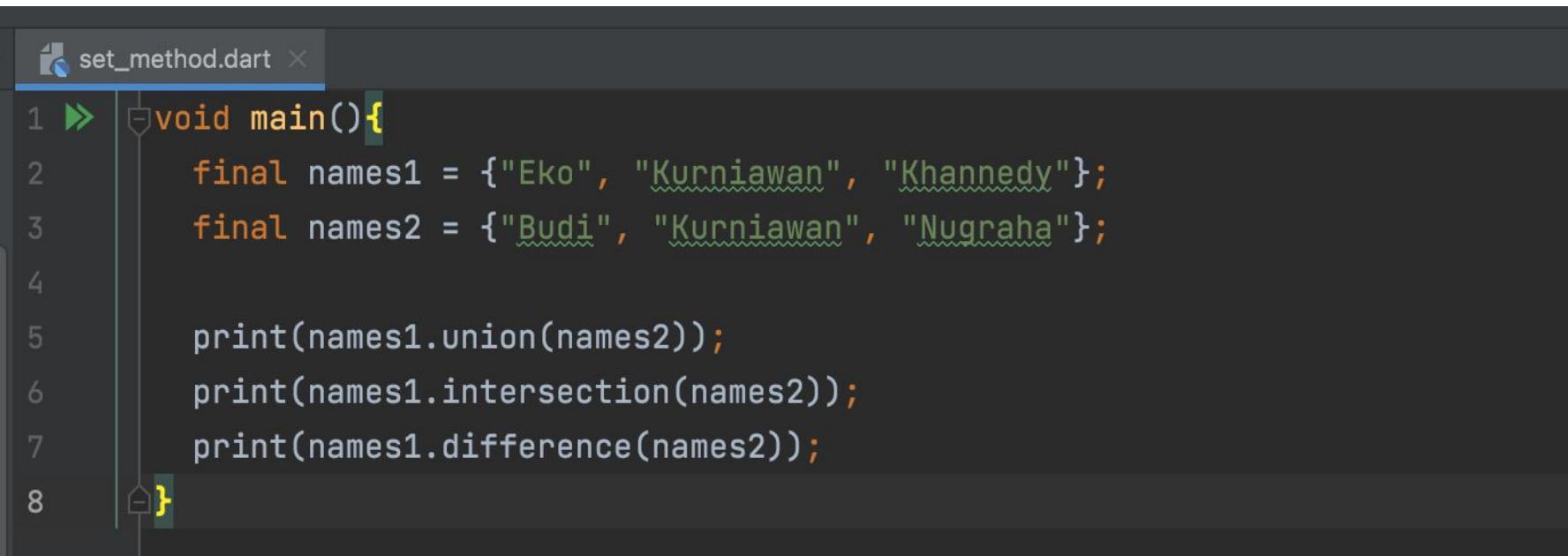
---

## Set Method

- Selain List, Set juga memiliki method yang spesial terdapat di Set
- <https://api.dart.dev/stable/2.17.6/dart-core/Set-class.html#instance-methods>

---

## Kode : Set Method



```
set_method.dart
1 void main(){
2     final names1 = {"Eko", "Kurniawan", "Khannedy"};
3     final names2 = {"Budi", "Kurniawan", "Nugraha"};
4
5     print(names1.union(names2));
6     print(names1.intersection(names2));
7     print(names1.difference(names2));
8 }
```

---

# Map

---

# Map

- Map sebenarnya mirip dengan tipe data List, dimana memiliki index dan value
- Hanya saja, berbeda dengan List, pada Map, kita bisa menentukan data index dengan tipe data dan data index sesuai yang kita mau
- Di Map, index disebut dengan key
- Detail tentang Map sudah kita bahas di materi Dart Dasar
- <https://api.dart.dev/stable/2.17.6/dart-core/Map-class.html>

## Kode : Map

```
map.dart x
1 ► void main() {
2   final Map<String, String> person = {
3     'firstName': 'Eko',
4     'lastName': 'Khannedy',
5   };
6
7   person['middleName'] = 'Kurniawan';
8
9   print(person);
10}
11
```

---

# Map Entry

---

## Map Entry

- Map sendiri bukanlah turunan dari Iterable, oleh karena itu secara default tidak bisa di iterasi menggunakan perulangan for
- Namun, Map memiliki property bernama entries, yang mengembalikan Iterable berisi MapEntry
- MapEntry adalah gabungan antara satu buah Key + Value
- <https://api.dart.dev/stable/2.17.6/dart-core/MapEntry-class.html>

## Kode : Map Entry



A screenshot of a code editor showing a Dart file named `map_entry.dart`. The code defines a `Map` named `person` with `String` keys and values. It prints each entry of the map.

```
1 >> void main() {
2     final Map<String, String> person = {
3         'firstName': 'Eko',
4         'lastName': 'Khannedy',
5     };
6
7     for (var entry in person.entries) {
8         print('${entry.key}: ${entry.value}');
9     }
10    }
11 }
```

---

# Hash Map

---

# Hash Map

- Hash Map merupakan implementasi dari Map yang tidak menggunakan struktur data Double Linked List
- Sama seperti Hash Set, urutan key pada Hash Map tidak bisa di jamin berurut
- <https://api.dart.dev/stable/2.17.6/dart-collection/HashMap-class.html>

# Kode : Hash Map



```
hash_map.dart ×
1 import 'dart:collection';
2
3 void main() {
4     final scores = HashMap<String, int>();
5
6     scores["Eko"] = 100;
7     scores["Budi"] = 100;
8     scores["Joko"] = 100;
9     scores["Dimas"] = 100;
10    scores["Donis"] = 100;
11
12    print(scores);
13}
14
```

---

# Linked Hash Map

---

# Linked Hash Map

- Linked Hash Map merupakan implementasi dari Map yang menggunakan struktur data Double Linked List
- Hal ini menjadikan Linked Hash Map datanya terurut sesuai dengan urutan kita memasukkan data ke Linked Hash Map
- Linked Hash Map merupakan default implementasi untuk Map
- <https://api.dart.dev/stable/2.17.6/dart-collection/LinkedHashMap-class.html>

# Kode : Linked Hash Map



A screenshot of a code editor showing a Dart file named `linked_hash_map.dart`. The code demonstrates the use of a `LinkedHashMap` to store names and scores. The code is as follows:

```
1 import 'dart:collection';
2
3 void main() {
4     final scores = LinkedHashMap<String, int>();
5
6     scores["Eko"] = 100;
7     scores["Budi"] = 100;
8     scores["Joko"] = 100;
9     scores["Dimas"] = 100;
10    scores["Donis"] = 100;
11
12    print(scores);
13}
```

---

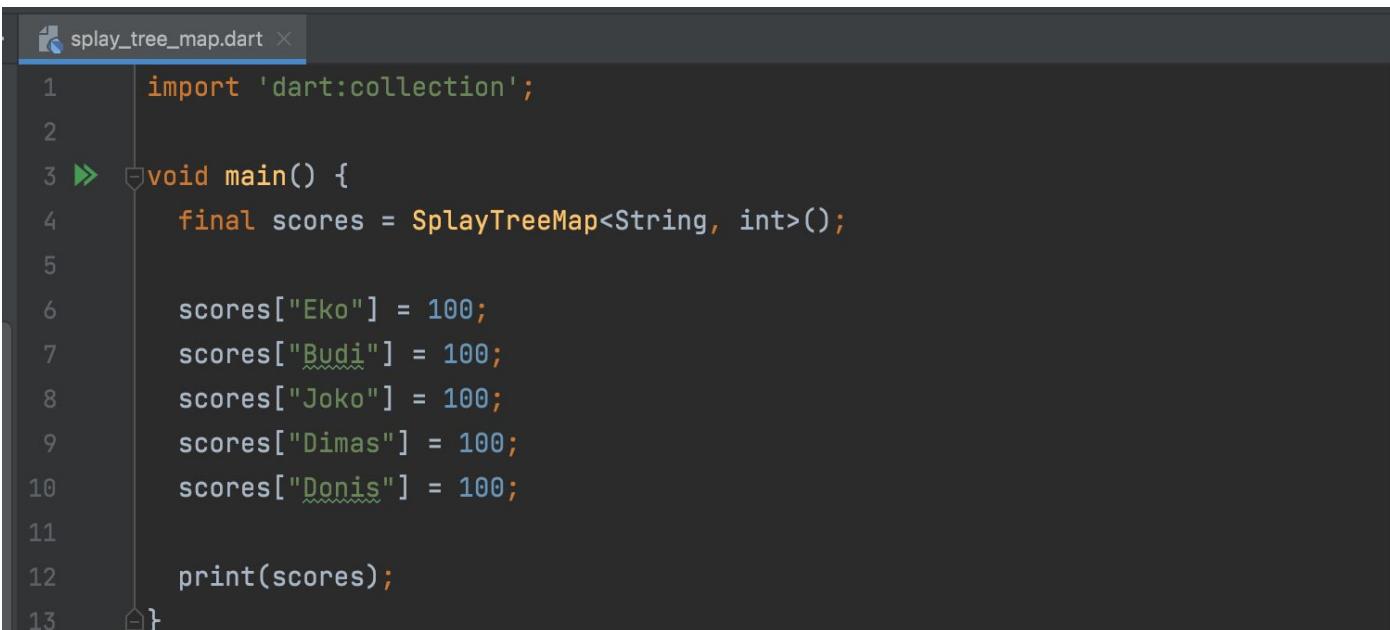
# Splay Tree Map

---

## Splay Tree Map

- Splay Tree Map merupakan implementasi dari Map yang menggunakan struktur data Tree
- Data di Splay Tree Map secara otomatis akan berurut sesuai dengan data nya, atau bisa menggunakan Comparator, mirip dengan Splay Tree Set
- <https://api.dart.dev/stable/2.17.6/dart-collection/SplayTreeMap-class.html>

# Kode : Splay Tree Map



A screenshot of a code editor window titled "splay\_tree\_map.dart". The code is written in Dart and uses a SplayTreeMap to store names and scores. The code is as follows:

```
1 import 'dart:collection';
2
3 void main() {
4     final scores = SplayTreeMap<String, int>();
5
6     scores["Eko"] = 100;
7     scores["Budi"] = 100;
8     scores["Joko"] = 100;
9     scores["Dimas"] = 100;
10    scores["Donis"] = 100;
11
12    print(scores);
13}
```

---

# Unmodifiable Map

---

## Unmodifiable Map

- Unmodifiable Map merupakan implementasi Map yang tidak bisa diubah lagi
- Cara penggunaannya adalah membungkus Map yang sudah ada, dengan Unmodifiable Map, sehingga tidak bisa dimodifikasi lagi
- <https://api.dart.dev/stable/2.17.6/dart-collection/UnmodifiableMapView-class.html>

# Kode : Unmodifiable Map



The screenshot shows a code editor window with a dark theme. The file tab at the top is labeled "unmodifiable\_map.dart". The code itself is as follows:

```
1 import 'dart:collection';
2
3 void main() {
4     final Map<String, String> person = {
5         'firstName': 'Eko',
6         'lastName': 'Khannedy',
7     };
8
9     final finalPerson = UnmodifiableMapView(person);
10
11     finalPerson['middleName'] = 'Kurniawan'; // error
12 }
```

The code demonstrates creating a map and then attempting to modify it using the `UnmodifiableMapView` wrapper, which prevents changes to the original map.

---

# Materi Selanjutnya

---

## Materi Selanjutnya

- Dart Unit Test
- Dart Async