Google Developer Student Clubs
Institut Teknologi Telkom Surabaya

Dart Decoded: Unveiling the Basics with Control Flow, Functions, and OOP

Ahmad Mu’min Faisal
Core Team GDSC ITTS

Outline

1. **Dart Overview**
2. **Variable** and **Data Type**
3. **Control Flow** (conditional and loop)
4. **Functions** (function)
5. **Object Oriented Programming** (class and object)

Slido

bit.ly/Tanya-MobDev-GDSCItts

Discord

bit.ly/discord-gdsc-itts

Google Developer Student Clubs

Dart Overview

```
lookup.KeyValue
f.constant(['em
=tf.constant([0
.lookup.StaticV

_buckets=5)
```

Dart is a **client-optimized and type-safe language** for developing **fast apps** on **any platforms**.

Dart forms the **foundation of Flutter**

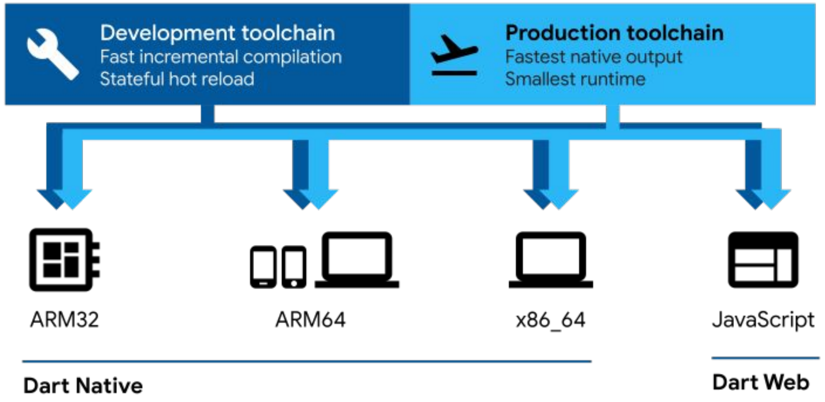
Dart’s Compiler Technology

Native Platform:

- **Development**: Using just-in-time (JIT) compiler for faster development.
- **Production**: Using ahead-of-time (AOT) compiler to compile to native ARM or x86_64 machine code.

Dart Web:

Compile Dart code into Javascript code which can run in a browser, such as Chromium and Firefox.



Dart's Compiler Technology

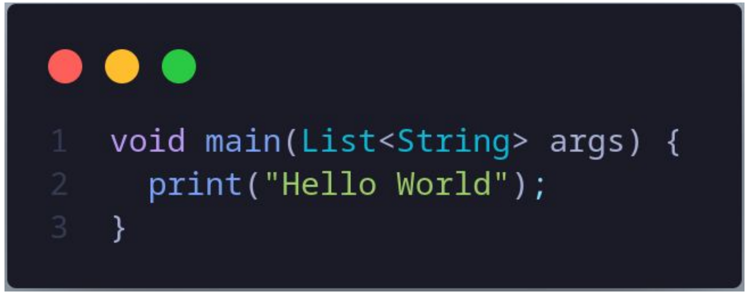
Curious about Dart sophisticated compiler? Let's do some experiment…

Please write the codes in the right side to the file named **`hello_world.dart`**.

Open new terminal, then run these command to compile dart to executable native machine code and to Javascript:

```
dart compile exe hello_world.dart
```

```
dart compile js hello_world.dart
```



```dart
void main(List<String> args) {
  print("Hello World");
}
```

Then, you can execute the compiled code with these command:

```
./hello_world.exe
```

`node hello_world.js` (if nodejs is installed)

But, for convenience reason, we will use

```
dart run hello_world.dart
```

to run Dart file with JIT compiler.

Google Developer Student Clubs
Institut Teknologi Telkom Surabaya

Variable and Data Types

```
lookup.KeyValue
f.constant(['em
=tf.constant([0
.lookup.StaticV

_buckets=5)
```

Variable and Data Types

Variable is a **container to store a value** in a computer program.

Data types **define** what **kind of value** a variable can store.

```
String communityName = "GDSC ITTelkom Surabaya" ;
int currentMember = 100;
int maxMember = 200;
bool isActive = true;
List<String> availableDivisions = [
     "Event Organizer",
     "Public Relations",
     "Media and Creative",
     "Tech and Curriculum",
];
```

Dart language has a special supports for the following types:

- Numbers (`int`, `double`)
- Strings (`String`)
- Booleans (`bool`)
- Records ((`value1`, `value2`))
- Lists (`List`, also known as arrays)
- Sets (`Set`)
- Maps (`Map`)
- Runes (`Runes`; often replaced by the characters API)
- Symbols (`Symbol`)
- The value null (`Null`)

Type Inference and Dynamic Type

Variable types can also **inferred** by Dart runtime using `var` keyword:

```
var str = "This is type inference" ;
```

Even though Dart is a typesafe language, we can also **disable type-checking** by using `dynamic` keyword:

```
dynamic str = "This is string";
str = 2023; // can store value with different type at runtime
```

Null Safety Feature

Null safety means that a variable **can't be null**, **unless we defined for it to be nullable**. This feature **prevents errors** that result from **unintentional access** of variables set to **null**.

```
void main(List<String> args) {
      List<String> divisions; // this is not nullable
      print(divisions); // gives compile-time error
}
```

If you really want to have a **nullable variable**, just **define it** :)

```
void main(List<String> args) {
      List<String>? greetings; // define as nullable
      print(greetings ?? ["Hello", "Hi"]); // do a null check
}
```

Note: This feature is a largest change from Dart 2 to Dart 3

Google Developer Student Clubs
Institut Teknologi Telkom Surabaya

Control Flow

```
lookup.KeyValue
f.constant(['em
=tf.constant([0
.lookup.StaticV

_buckets=5)
```

Control Flow

Control flow **decides the order** your program takes. It makes your program “smarter”, by **making decisions** or **looping** through certain instructions. There are 2 types of control flow:

- Branches
- Loops

Branches

Branches is used to **execute** your code **conditionally**. In Dart, use `if`, `else if`, and `else` statements to **execute specific code blocks** based on whether a **condition evaluates to true or false**.

```
if (isActive && currentMember >= maxMember) {
      print("$communityName is active but not accepting new members");
} else if (isActive && currentMember < maxMember) {
      print("$communityName is active and still accepting new members");
} else {
      print("$communityName is not active");
}
```

Alternatively, you can define branches with ternary operation if not really complex.

```
print(isActive ? "$communityName is active" : "$communityName is not active");
```

Loops (*for-loop*)

Loops let you **repeat code until a condition is met**. This kind of control flow is mainly used to **performs repetitive task**.

```
// iterate from i = 0; while i < availableDivisions, with 1 step
each iteration
for(int i = 0; i < availableDivisions.length; i++){
      // for each iteration, print division
      print("${availableDivisions[i]} is available");
}
```

But for more **readability**, you can run above code as follow (depends on the use case).

```
for (String division in availableDivisions) {
      print("$division is available");
}
```

There are generally 2 types of loops, that is *for-loop* and *while-loop*.

For-loop is used when we are certain when will our loops stop iterating. It is used to iterate over a range of an object.

While-loop is used when the looping condition is not certain, usually if the condition is quite complex. It is used to iterate while boolean condition is met.

Loops (*while-loop*)

While-loop is another type of loops where it is **iterating by boolean condition**. While condition is evaluated as **true**, it will **continue iterating**. Otherwise, it will stop the iterations.

Note: while-loop is **more flexible** than for-loop, because it is not limited in a range of an objects. But for-loop is **easier to use**.

```
String searchedDivision = "Tech";
bool foundDivision = false;
int index = 0;

while(index < availableDivisions.length) {
      String division = availableDivisions[index];
      if(division.contains(searchedDivision)){
            foundDivision = true;
            break;
      }
      index++;
}

print(foundDivision ? "Division found" : "Division
not found");
```

Google Developer Student Clubs
Institut Teknologi Telkom Surabaya

Functions

```
lookup.KeyValue
f.constant(['em
=tf.constant([0
.lookup.StaticV

_buckets=5)
```

Function (void function)

A function is a **block** of **organized**, **reusable code** that is used to **perform a single, related action**. For the most basic function, we can make a single void function with no parameter.

A function can take no parameters. But, in this example we take a single parameter, the list of divisions.

```
// declare a function outside the main function
void printAllDivisions(List<String> divisions) {
      for (String division in divisions) {
            print("$division is available");
      }
}

void main(List<String> arguments) {
      /**
      * other codes...
      */

      // call the function
      printAllDivisions(availableDivisions);
}
```

Function (non-void function & named parameter)

```
String getRecruitmentStatus({
      required String communityName,
      required bool isActive,
      required int currentMember,
      required int maxMember,
}) {
      if (isActive && currentMember >= maxMember) {
            return "$communityName is active but not 
accepting new members";
      } else if (isActive && currentMember < maxMember) {
            return "$communityName is active and still 
accepting new members";
      }
      return "$communityName is not active";
}
```

```
 // returned value can be stored in a 
variable
String recruitmentStatus = 
getRecruitmentStatus(
      communityName: communityName,
      isActive: isActive,
      currentMember: currentMember,
      maxMember: maxMember,
);
```

Note: required keyword in the function parameter means that the argument must be defined in the function call. You can’t remove those requirements unless you provides the default/fallback value.

Is there **any other task** in our main function that can be **defined into a function**?

Is your code **getting cleaner**?

But, **can we get cleaner code**?

But, a **new problem** arised... What if we have **more community data**?

Google Developer Student Clubs

Object-Oriented Programming

```
lookup.KeyValue
f.constant(['em
=tf.constant([0
.lookup.StaticV

_buckets=5)
```

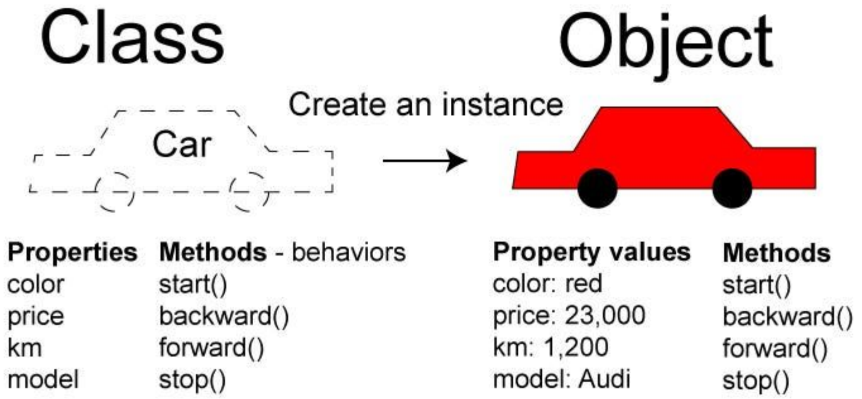
Object-Oriented Programming

Object-Oriented Programming is a **programming paradigm** based on the **concept of object**.

An object can contain data and code, where:

- Data in the form of **properties**
- Code in the form of **methods**

These object is instantiated from a **object blueprint** called **class**.



So, let’s define **community as an object**

Define Community Class (properties and constructor)

Class is created as a **blueprint to an object**.

Here, the blueprint define that **every object instantiated** from Community class, **will have such properties**.

Properties defines "**What is this object have**".

Then, we can define a **constructor** to **create an object** from given arguments assigned to its properties.

```
class Community {
	// define the properties of the object created by
Community class
	final String name;
	final int currentMember;
	final int maxMember;
	final bool isActive;
	final List<String> divisions;

	// constructor: used to instantiate the object
from specified arguments/data
	const Community({
		required this.name,
		required this.currentMember,
		required this.maxMember,
		required this.isActive,
		required this.divisions,
	});
}
```

Define Community Class (methods)

Methods is a functions that defines “**What this object can do**”. These functions can be executed by object instantiated from this class.

A method of an object **can access its object properties**.

```
// method to print all divisions
void printAllDivisions() {
  for (String division in divisions) {
    print("$division is available");
  }
}

// Dart built-in getter method
String get recruitmentStatus {
  if (isActive && currentMember >= maxMember) {
    return "$name is active but not accepting new members";
  } else if (isActive && currentMember < maxMember) {
    return "$name is active and still accepting new members";
  }

  return "$name is not active";
}
```

So, let's define **the remaining method**

Instantiate Object from Class

We can instantiate object to “realize” your class. Those object will have your assigned properties and defined methods!

And, we can instantiate another Community from our class easily!

```
void main(List<String> arguments) {
 Community gdsc = Community(
   name: "GDSC Institut Teknologi Telkom Surabaya",
   currentMember: 150,
   maxMember: 200,
   isActive: true,
   divisions: <String>[
     "Event Organizer",
     "Public Relations",
     "Media and Creative",
     "Tech and Curriculum",
   ],
 );

 gdsc.printAllDivisions();
 print(gdsc.recruitmentStatus);
 gdsc.searchDivision(searchedDivision: "Tech");
}
```

Yay, we have **solved our problems**!

Now, our **code** getting **even cleaner**!

Let's **experiment**!

Absensi

bitly

```
ild: Column(
 crossAxisAlignment: CrossAxis
 children: [
   /*2*/
   Conta
     pad
     chi
       '
       s

       )
     ),
   ),
   Text(
     'Ka
     sty
       c
     ),
   ),
```

Thank you

Google Developer Student Clubs

Google Developer Student Clubs

Institut Teknologi Telkom Surabaya