DESIGN ISSUES OF LANGUAGE'S BASIC COMPONENTS DART A

Nishat Quyoum, Giselle Avila, Micheal O'Neill,
Christian Anderson, Devin Khun

NAME AND CONVENTIONS

Identifier Rules

- Must start with a letter (any Unicode letter) or an underscore
- Subsequent characters can be letters, digits (0–9), or underscores
- Case sensitive
- Cannot start with a digit
- Cannot use Reserved words (keywords)

Keywords

- abstract
- as
- assert
- async
- await
- base
- break
- etc.

Valid identifier

_privateValue

myVariable

Δx

Invalid identifier

9lives

my-var

NAME AND CONVENTIONS

Guidelines for Variables

Dart's documentation provides Effective Style Guide for naming variables: https://dart.dev/effective-dart/style

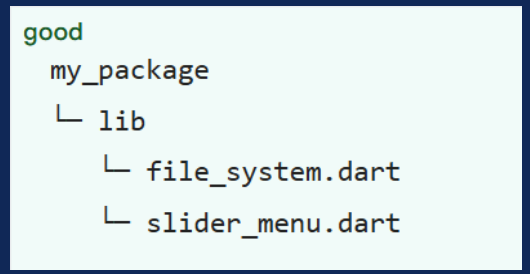
- Types should be named using UpperCamelCase
- Other identifiers (including variables) should be named using lowerCamelCase
- Packages, directories, and source files should use lowercase_with_underscores

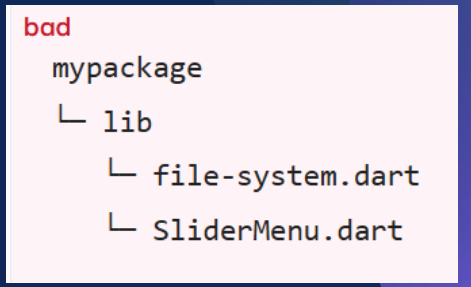
Compiler/Interpreter

- Does not enforce Style guidelines (stylistic recommendations rather than syntactic requirements)
- Tools like Dart analyzer can flag deviations as warnings to follow the guidelines consistently

Follow? Yes

- Readability
- Maintainability
- Community best practices





DATA TYPES

Java

```java
int x = 5, y = 10;

// Comparison operators
boolean isGreater = x > y;
boolean isNotEqual = x != y;

// Logical operators
boolean isTrue = true;
boolean isFalse = false;
boolean andResult = isTrue && isFalse;
boolean orResult = isTrue || isFalse;
boolean notResult = !isTrue;

// Compound expressions
boolean complexExpression = (x > 0) && (y < 20) || (x == 5);
```

Dart

```dart
int x = 5, y = 10;

// Comparison operators
bool isGreater = x > y;
bool isNotEqual = x != y;

// Logical operators
bool isTrue = true;
bool isFalse = false;
bool andResult = isTrue && isFalse;
bool orResult = isTrue || isFalse;
bool notResult = !isTrue;

// Compound expressions
bool complexExpression = (x > 0) && (y < 20) || (x == 5);
```

Converting Java to Dart

- Only real difference between dart and java is *bool* instead of *boolean*
- Can also use *var* for all type declarations

Output

```
x: 5
y: 10
isGreater: false
isNotEqual: true
isTrue: true
isFalse: false
andResult: false
orResult: true
notResult: false
complexExpression: true
```

DATA TYPES

- Type declaration is not strictly required due to *var*
- Type errors are mostly caught at compile time, but some are caught at run time
  - *dynamic* types can be caught at run time
  - Type casting using *as* operator
- Dart supports a boolean type, *bool*, with *true* and *false* values

Type Inference with *var*

```
var x = 5, y = 10;

// Comparison operators
var isGreater = x > y;
var isNotEqual = x != y;
```

Boolean data

```
bool isTrue = true;
bool isFalse = false;
```

Run-time type checking

```
String z = "Hello";
print(z is bool);
```

```
int number = "hello" as int;
```

Dynamic data

```
dynamic v = "hello";
print(v.length);
v = 5;
print(v.length);
```

MAP/DIRECTORY TYPES

- Dart has a built-in dictionary-like data type called Map. It allows key-value pair storage and retrieval.
- **Key-Value Structure**: Each entry in a Map consists of a unique key and an associated value.
- **Key Uniqueness**: No two entries in a Map can have the same key. (If you try to attempt this, the latest entry overwrites the previous one without throwing an error.)
- **Dynamic Sizing**: A Map can grow and shrink dynamically as elements are added or removed.
- **Fast Lookups**: Maps provide fast access to values using keys.

MAP/DIRECTORY TYPES

```dart
Map<String, int> ages = {'Alice': 25, 'Bob': 30};
print(ages['Alice']); // Output: 25
```

- Here's an example of how to define and use a **Map** in Dart:
- Map<String, int>: This declares a Map where keys are of type String and values are of type int.
- ages: The variable name holding the map.
- {'Alice': 25, 'Bob': 30}: This initializes the map with two key-value pairs:
  - Key 'Alice' maps to the value 25.
  - Key 'Bob' maps to the value 30.
- ages['Alice']: This retrieves the value associated with the key 'Alice', which is 25.

ARRAY TYPES

```dart
// Fixed Heap-Dynamic
var fixedList = List<int>.filled(5, 0); // Fixed length of 5, all elements initialized to 0

// Heap-Dynamic
var dynamicList = <int>[]; // Empty list, can grow or shrink
dynamicList.add(10); // Adding an element
var names = ["John", "Robert", "James"];
List<String> names = ["John", "Robert", "James"];
names.add("Michael");
names.remove("John");
```

Allocation: Dart arrays (List objects or Collections) are allocated on the heap

- All objects are heap-allocated in Dart
- Fixed Heap-Dynamic: fixed length is set at initialization
- Heap-Dynamic: Lists can dynamically grow or shrink throughout the program

- Homogenous: Strong typing and all elements must be of specified type
  - List<int> or List<String>
- Heterogenous: By default, Lists can hold elements of different data types and when a type is not specified
  - List<dynamic>

```dart
// Homogeneous List
List<int> numbers = [1, 2, 3];

// Heterogeneous List
var mixedList = [1, "hello", true];
```

ARRAY TYPES

- Elements of arrays can be accessed through their index
  - Dart does not allow negative indices for array indexing
  - Attempting to use negative subscripts will result in runtime error
- 2-D arrays can be represented as a List of Lists

```dart
// Accessing elements
List<int> numbers = [10, 20, 30];
print(numbers[0]); // Output: 10
print(numbers[-1]); // Error: invalid value for index

// 2-D Array
var matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
]; // 3x3 2-D array

print(matrix[1][2]); // Accessing element at 2nd row, 3rd column => Output: 6
print(matrix[0]); // Accessing 1st row => Output: [1, 2, 3]
```

SCOPING

Does Dart allow nested scope for function definitions?

- Dart supports nested scopes by employing lexical scoping
- Lexical Scoping:
  - Variables are resolved based on their location in the code source
  - An inner function can use outer function variables, but outer functions cannot use inner variables
- Visibility:
  - This is because variables in nested functions are not visible to the parent scope. So accessing them outside their scope causes an error

```dart
void main() {
  for (int i = 1; i <= 3; i++) { // Outer loop
    print("Outer loop iteration: $i");

    for (int j = 1; j <= 2; j++) { // Inner loop
      print("  Inner loop iteration: $j");
    }

    // print(j); // Error: j is not accessible outside the inner loop
  }
}
```

Example:

- Variable 'i' is declared in the outer loop and is accessible in both loops.
- Variable 'j' is only accessible in the inner loop
- Attempting to access variable 'j' outside the inner loop it will result in a runtime error since it is out of scope.

SCOPING

How does Dart define/distinguish global and local variables?

- Dart does not use keywords like global or non-local keywords

Top-Level Variables (Global):

- Declared outside any function or class.
- Accessible throughout the entire program.

Local Variables:

- Declared inside a function, block, or loop
- Accessible only within that scope

```dart
int globalVar = 100; // Top-level (global) variable

void myFunction() {
  int localVar = 50; // Local variable
  print(globalVar + localVar); // Access global and local variables
}

void main() {
  myFunction(); // Output: 150
}
```

globalVar is a top-level variable and is accessible everywhere
localVar is a local variable and only is accessible inside its function

RESOURCES

1. https://dart.dev/language/functions#lexical-scope
2. https://how.dev/answers/how-to-create-a-nested-function-in-dart
3. https://medium.com/@prashannagc11/1-types-of-dart-variables-6c7228baf36c
4. https://dart.dev/language/collections
5. https://dart.dev/effective-dart/style
6. https://dart.dev/language/keywords

QUESTIONS

Thank you for listening!