



Dart

**Team: Ashish Kupsad, Biju Amruta Dathan, Hemanth Reddy,
Paritosh Gavali**

Challenges/Issues Dart tries to solve

- Large scale application development in JavaScript requires a lot of effort, if not impossible.
- JavaScript lacks structuring mechanisms, tools, editors, code analyzers.
- JavaScript Frameworks and Libraries – jQuery, Backbone, Knockout, Angular, React, Ember, Aurelia, Bootstrap etc, are so vast and different in their architectural style.
- Different languages compiling to JavaScript like GWT (compiles Java to JS), Pyjamas (Python to JS). Dart is such an example

Released in 2013, Dart was meant as a replacement to the traditional JavaScript in browsers

Goal of Dart



Help app developers
write complex, high
fidelity client apps for
the modern web.

Some history

- Dart was first unveiled at the GOTO conference in Aarhus, Denmark, October 10–12, 2011. The project was founded by Lars Bak and Kasper Lund of Google.
- Apparently, Google engineers were reportedly frustrated with maintaining massive JavaScript code bases for Gmail and Google Maps, and so they began working on an alternative language
- Dart 1.0 was released on November 14th, 2013. In August 2018, Dart 2.0 was released (featured improvements to client-side development, including strong typing and “UI as code”).
- Dart 2.6 introduced a new extension, `dart2native`. The feature extends native compilation to the Linux, macOS, and Windows desktop platforms.

What is Dart

- Dart is an **open-source, scalable programming language**, with **robust libraries and runtimes**, for building **web, server, and mobile apps for multiple platforms..** It is developed by **Google**
- Dart is class based, **purely object oriented**, (similar to Smalltalk, Ruby and Scala) - so even basic types (int, float) are objects.
- It is a **dynamic language** like JavaScript.
- Dart supports **optional static typing** and type checks
- Dart supports **single inheritance** with support for **mixins**
- Dart supports **Real lexical scoping** and **closures**

What is Dart (cont.)

- Dart is heavily influenced by JavaScript, Java and C#
- It has familiar syntax of Java/C# and incorporates ideas from JavaScript.

Dart is comprised of the following:

- Language and libraries
- Tools
- Virtual Machine
- Compiler to JavaScript

What is Dart (Cont..)

- Influenced by Strongly typed languages like Java, C# and loosely typed dynamic language like JavaScript

Feature	Dart	Java / C#	JavaScript
Type system	Optional, dynamic	Strong, static	Weak, dynamic
First class functions	Yes	Can simulate with anonymous functions	Yes
Closures	Yes	Yes, with anonymous classes	Yes
Classes	Yes, single inheritance	Yes, single inheritance	Prototypal
Interfaces	Yes, multiple inheritance	Yes, multiple inheritance	No
Concurrency	Yes, with isolates	Yes, with threads	Yes, with HTML5 web workers

Popularity of Dart

Worldwide, Nov 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
20	↑↑↑↑	Lua	0.58 %	+0.2 %
21	↑	Dart	0.57 %	+0.2 %
22	↓↓↓	Perl	0.47 %	-0.1 %

Table taken from PYPL (Popularity of Programming Language)

Dart-lang SDK contributions graph

dart-lang / sdk

Watch

265

Star

6.1k

Fork

890

Code

Issues 5k+

Pull requests 7

Actions

Projects 10

Wiki

Security 1

Insights

Pulse

Contributors

Community

Commits

Code frequency

Dependency graph

Network

Forks

Oct 2, 2011 – Nov 10, 2020

Contributions: Commits

Contributions to master, excluding merge commits



<https://github.com/dart-lang/sdk/pulse>

 schealov

#1

 stereotype441

#2

Advantages of Dart:

1. Fast performance
2. Ease of learning
3. Availability of good documentation
4. Very stable and has robust libraries
5. AOT and JIT compilation
6. write Dart program without any installation or configuration via DartPad

Basic Concepts

- Dart parses all your code before running it. You can provide tips to Dart—for example, by using types or compile-time constants—to catch errors or help your code run faster.
- Dart supports top-level functions (such as `main()`), as well as functions tied to a class or object (static and instance methods, respectively). You can also create functions within functions (nested or local functions).
- Similarly, Dart supports top-level variables, as well as variables tied to a class or object (static and instance variables).

Basic Concepts

- Unlike Java and some other languages, Dart doesn't have the keywords `public`, `protected`, and `private`. If an identifier starts with an underscore (`_`), it's private to its library.
- Identifiers can start with a letter or `'_'`, followed by any combination of those characters plus digits.
- Dart tools can report two kinds of problems: warnings and errors. Warnings are just indications that your code might not work, but they don't prevent your program from executing. Errors can be either compile-time or run-time. A compile-time error prevents the code from executing at all; a run-time error results in an exception being raised while the code executes.

Modes

- Dart VM has two runtime modes: **production** and **checked**. It is recommended that you develop and debug in checked mode, and deploy in production mode.
- **Production mode** is the default runtime mode of a Dart program, optimized for speed. Production mode ignores assert statements and static types.
- **Checked mode** is a developer-friendly mode that helps you catch some type errors during runtime. For example, if you assign a non-number to a variable declared as a num, then checked mode throws an exception.

Creating native mobile apps via Flutter

- Google introduced **Flutter** for native mobile app development on both **Android** and **iOS**.
- Flutter is powered by Dart, which allows compilation to native 32-bit and 64-bit ARM code for both iOS and Android.
- Flutter is easy to program in, and developers generally like the idea of developing cross-platform with a single code-base
- Flutter/Dart supports IDEs such as Visual Studio Code, Android Studio, and IntelliJ Idea.

Basics - First Dart Program

```
// Entry point to Dart program
main() {
    print('Hello from Dart');
}
```

- main() - The special, required, top-level function where app execution starts.
- Every app must have a top-level main() function, which serves as the entry point to the app.
- Returns void and has an optional List<String> parameter for arguments.

```
void main(List<string> args) {
    print('Hello from Dart');
    print(args[0] + ", " + args[1]);
}
```

```
dartplay.dart arg1 arg2 // Command to run dart
Hello from Dart
arg1, arg2
```

Comments

- Dart supports both single line and multi line comments similar to C,C++,Java,..

```
// Single line comment
```

```
/* This is an  
example  
of multi line comment  
*/
```

```
/*  
This is also an example of  
multi line comment  
*/
```


Variables

- Variables are declared using `var` keyword similar to JavaScript.

```
var name = 'Bob';
```

- Variables are references.
- Uninitialized variables have an initial value of null. Even variables with numeric types are initially null, because numbers are objects.

Built-in types

- number
 - int - Integer values, which generally should be in the range -253 to 253
 - double - 64-bit (double-precision) floating-point numbers, as specified by the IEEE 754 standard
- string
- boolean – true and false
- symbol
- Collections
 - list (arrays)
 - map
 - queue
 - set

String Interpolation

- Identifiers could be added within a string literal using **\$identifier** or **\$variable_name** syntax.

```
var user = 'Bill';  
var city = 'Bangalore';  
print("Hello $user. Are you from $city?");  
// prints Hello Bill. Are you from Bangalore?
```

- You can put the value of an expression inside a string by using **\${expression}**

```
print('3 + 5 = ${3 + 5}'); // prints 3 + 5 = 8
```

List

- Perhaps the most common collection in nearly every programming language is the array, or ordered group of objects.
- In Dart, arrays are List objects, so we usually just call them lists.

```
var numbers = [1,2,3,4,5];  
var cities = ['Bangalore', 'Kolkata', 'Chennai'];
```

Control flow statements

- if and else
- for loops (for and for in)
- while and do while loops
- break and continue
- switch and case

if and else

- if and else

```
var age = 17;
if(age >= 18){
    print('you can vote');
}
else{
    print('you can not vote');
}
```

- curly braces { } could be omitted when the blocks have a single line of code

```
var age = 17;
if(age >= 18)
    print('you can vote');
else
    print('you can not vote');
```

else if

- Supports else if as expected

```
var income = 75;
if (income <= 50) {
  print('tax rate is 10%');
}
else if (income > 50 && income < 80) {
  print('tax rate is 20%');
}
else {
  print('tax rate is 30%');
}
```

- curly braces { } could be omitted when the blocks have a single line of code

```
if (income <= 50)
  print('tax rate is 10%');
else if (income > 50 && income < 80)
  print('tax rate is 20%');
else
  print('tax rate is 30%');
```

for loops

- Supports standard for loop (as supported by other languages that follow C like syntax)

```
for(int ctr=0; ctr<5; ctr++){  
    print(ctr);  
}
```

- Iterable classes such as List and Set also support the for-in form of iteration

```
var cities = ['Kolkata', 'Bangalore', 'Chennai', 'Delhi'];  
  
for(var city in cities){  
    print(city);  
}
```

- Iterable classes also support forEach method

```
var cities = ['Kolkata', 'Bangalore', 'Chennai', 'Delhi'];  
cities.forEach((city) => print(city));
```


switch case

- Switch statements compare integer, string, or compile-time constants using ==
- Enumerated types work well in switch statements
- Supports empty case clauses, allowing a form of fall-through

```
var window_state = 'Closing';
switch(window_state){
    case 'Opening':
        print('Window is opening');
        break;
    case 'Opened':
        print('Window is opened');
        break;
    case 'Closing':
        print('Window is closing');
        break;
    case 'Closed':
        print('Window is closed');
        break;
    case 'Terminating':
    case 'Terminated':
        print('Window is terminating or terminated');
        break;
}
```

Object Oriented Features

- Supports single inheritance and multiple interfaces.
- Dart's OO model is similar to Java/C# and not similar to JavaScript. Dart supports class based inheritance, and not prototypal inheritance supported by JavaScript.

Class

- Dart is an object-oriented language with classes and mixin-based inheritance.
 - Every object is an instance of a class, and all classes descend from Object
- Instance Variables:

```
class Employee{
    String firstName;
    String lastName;
    int age;
    double salary;
}

main(){
    var emp = new Employee();

    emp.firstName = "Lars";
    emp.lastName = "Bak";
    print(emp.firstName);
    print(emp.lastName);
}
```

Class constructor

- The pattern of assigning a constructor argument to an instance variable is so common, Dart has syntactic sugar to make it easy.
- If you don't declare a constructor, a default constructor is provided for you. It has no arguments and invokes the no-argument constructor in the superclass.

```
class Employee{
  String firstName;
  String lastName;
  int age;
  double salary;
  Employee(this.firstName, this.lastName, this.age, this.salary);
}
```

```
//this is syntactic sugar of above class
Employee(firstName, lastName, age, salary){
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
  this.salary = salary;
}
```

Class constructor (cont.)

```
main() {  
    var emp =  
    new Employee('Lars', 'Bak', 45, 550.67);  
    print(emp.firstName);  
    print(emp.lastName);  
    print(emp.age);  
    print(emp.salary);  
}
```

First Class Functions

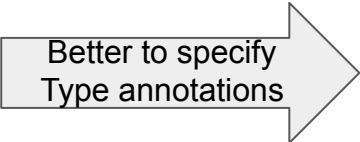
- Dart is similar in many ways to languages such as Java and C#, but its function syntax is more similar to that found in JavaScript than in more strongly typed languages.
- Everything is an object, including functions, which means you can store a function in a variable and pass it around your application the same way that you might pass a String, an int, or any other object. This is called first-class functions, because they're treated as equivalent to other types.

Functions

```
display() {  
    print('Hello from Dart');  
}
```

```
add(num1, num2) {  
    return  
    num1+num2;  
}
```

Better to specify
Type annotations



```
int add(int num1, int num2) {  
    return num1+num2;  
}
```

Declaring functions with => syntax

- For functions that contain just one expression, you can use a shorthand syntax
- The **=> expr;** syntax is a shorthand for **{ return expr; }**
- Only an expression, not a statement, can appear between arrow (=>) and semicolon (;). For example, you can't put an if statement there, but you can use a conditional expression.

```
void display(){
    print('Hello from Dart');
}
var display = () => print('Hello from Dart');
```

```
int add(int num1, int num2){
    return num1+num2;
}
var add = (x,y) => x+y;
```


Optional named parameters

```
int add(int num1, [int num2 = 5]) {           // num2 is optional with default value 5
    return num1 + num2;
}

print(add(20,10));
print(add(20));
```

Optional positional parameter

- Wrapping function parameters in `[]` marks them as optional positional parameters

```
void display(String message, [string user]) {  
    if(user == null)  
        print(message);  
    else  
        print("Hello $user. $message");  
}  
  
display("Welcome to Dart", "Ani");           // Hello Ani. Welcome to Dart  
display("Welcome to Dart");                 // Welcome to Dart
```

- Optional positional parameters can have default value

```
void display(String message, [string user = "User"]) {  
    print("Hello $user. $message");  
}  
  
display("Welcome to Dart", "Ani");           // Hello Ani. Welcome to Dart  
display("Welcome to Dart");                 // Hello User. Welcome to Dart
```

Method Cascades

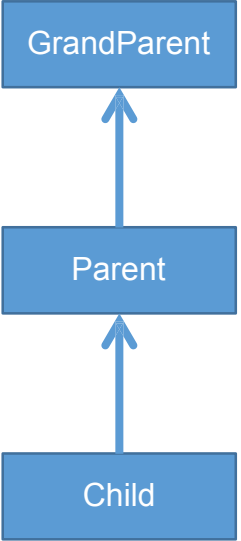
- Inspired from Smalltalk, Basic/VB also supports this style using `with` keyword
- `..` is the cascaded method invocation operation. The `..` syntax invokes a method (or setter or getter) but discards the result, and returns the original receiver instead
- Helps writing code in Fluent API style

```
class Employee{
    var name;
    var age;
    var designation;
    var salary;

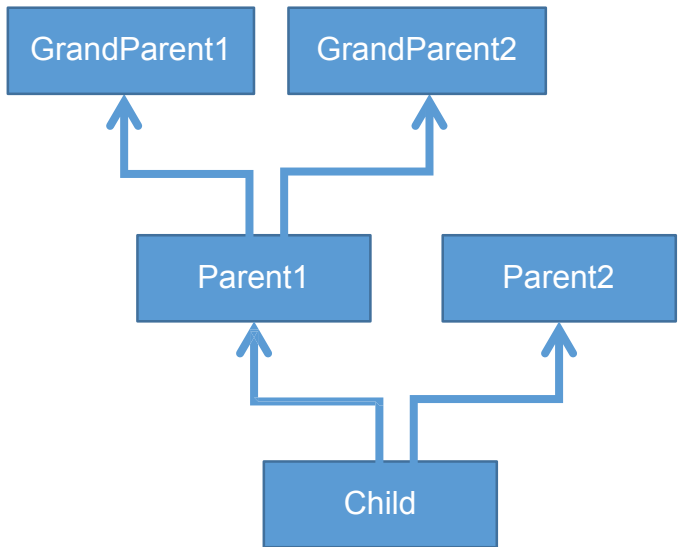
    Employee(this.name,this.age);
}

var emp = new Employee('XYZ',30)
    .. designation = "CEO"
    .. salary = 100.50;
```

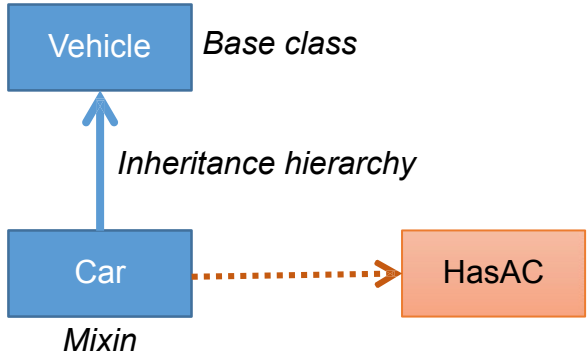
Mixins



Single Inheritance



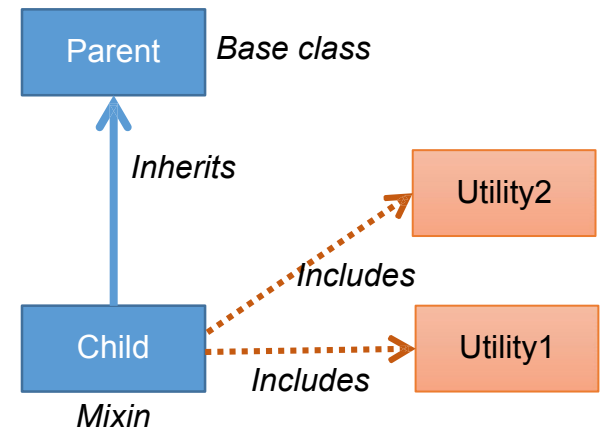
Multiple Inheritance using Inheritance hierarchy



Mixins

- A mixin is a class that contains a combination of methods from other classes. How such a combination is done depends on the language.
- Described as being "included" rather than "inherited".
- Mixins encourage code reuse and can be used to avoid the inheritance ambiguity that multiple inheritance can cause. Mixins are a way of reusing a class's code in multiple class hierarchies.
- Originated in LISP. Variants found in Ruby, Scala, Newspeak.
- Restrictions on mixin definitions in Dart include:
 - Must not declare a constructor
 - Superclass is Object
 - Contains no calls to super
- You can use the mixin with the **with** keyword

```
class Child extends Parent with Utility1, Utility1 {  
}
```

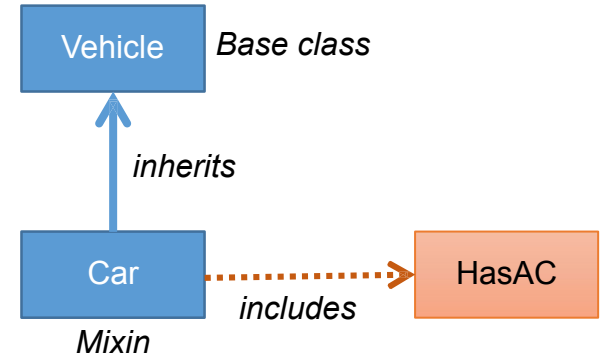


Mixins

```
// Base class
class Vehicle{
    int noOfWheels;
    drive(){
        print('I can move');
    }
}

// Mixin - implemented as abstract class
abstract class HasAC{
    double temperature = 20;
    void increaseTemperature(double by){
        temperature += by;
    }
    void decreaseTemperature(double by){
        temperature -= by;
    }
}

class Car extends Vehicle with HasAC{
    int noOfWheels = 4;
}
```



Single inheritance hierarchy

```
main(){
    var car = new Car();

    car.drive();
    // prints I can move

    car.decreaseTemperature(5);

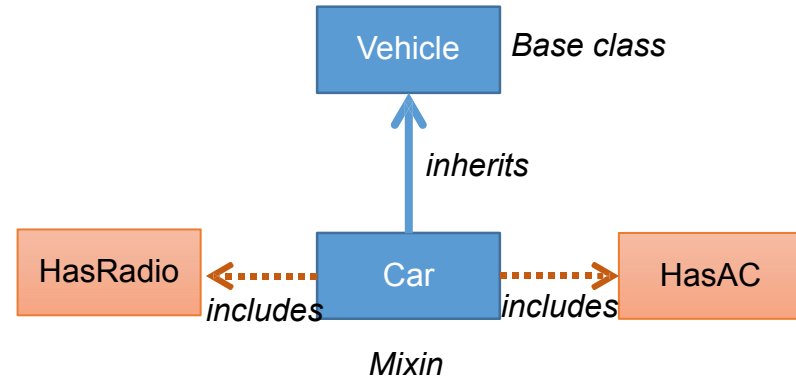
    print(car.temperature);
    // prints 15
}
```

Mixins (Cont'd)

```
// Base class
class Vehicle{
    int noOfWheels;
    var drive = () => print('I can move');
}

// Mixin - implemented as abstract class
abstract class HasAC{
    double temperature = 20;
    void increaseTemperature(double by){
        temperature += by;
    }
    void decreaseTemperature(double by){
        temperature -= by;
    }
}

abstract class HasRadio{
    String channel = 'bbc';
    void setChannel(channel){
        this.channel = channel;
    }
    void play(){
        print('Playing $channel');
    }
}
```



```
class Car extends Vehicle with HasAC, HasRadio{
    int noOfWheels = 4;
}

main(){
    var car = new Car();
    car.drive();
    car.decreaseTemperature(5);
    print(car.temperature);
}
```

Metadata (@)

- Use metadata to give additional information about your code. A metadata annotation begins with the character @, followed by either a reference to a compile-time constant (such as deprecated) or a call to a constant constructor.
- Two annotations are available to all Dart code: @deprecated and @override. For examples of using @override, see Extending a class. Here's an example of using the @deprecated annotation:

@deprecated	@override
<pre>class Television { /// _Deprecated: Use [turnOn] instead._ @deprecated void activate() { turnOn(); } /// Turns the TV's power on. void turnOn() {...} }</pre>	<pre>class SmartTelevision extends Television { @override void turnOn() {...} // ... }</pre>

Isolate and Asynchronous Operations

- Inspired by Actor Model of solving concurrency issues in Erlang, Scala and other languages.
- Most computers, even on mobile platforms, have multi-core CPUs. To take advantage of all those cores, developers traditionally use shared-memory threads running concurrently. However, shared-state concurrency is error prone and can lead to complicated code.
- Instead of threads, all Dart code runs inside of isolates. Each isolate has its own memory heap, ensuring that no isolate's state is accessible from any other isolate.

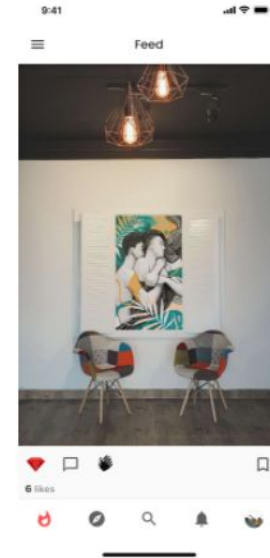
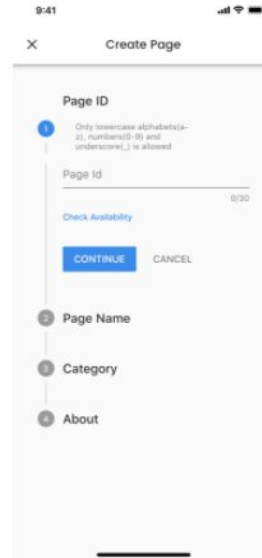
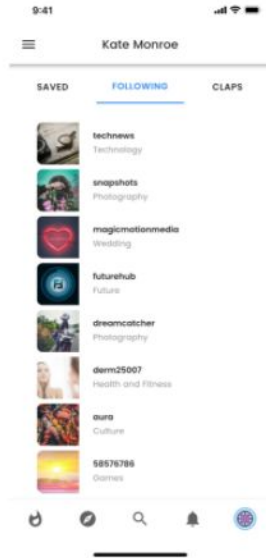
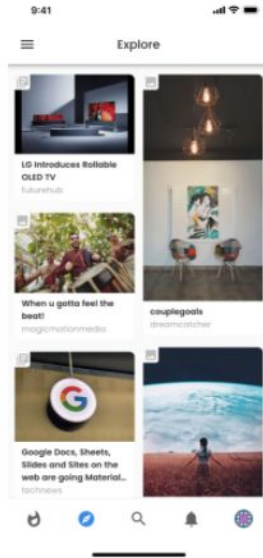
Where is Dart used?

- Mainly Flutter apps
 - **Google Ads:** helps customers keep their ad campaigns running smoothly on the go.



Where is Dart used? Cont.

- **KlasterMe:** an app for creating, sharing, and discovering different forms of content from images to articles.



Where is Dart used? Cont.

- **Xianyu, Alibaba:** has 50M+ downloads and more than 10 millions users use this app daily.



Where is Dart used? Cont.

- **Fuchsia** (an open-source operating system built by Google) is also built with Dart and the recently introduced **Google Home Hub** will possibly run Fuchsia in the future.
- **Google Assistant:** The Assistant team at Google uses Dart for features in Smart Displays.
- Google uses Dart for building critical Web apps, utilised with **AngularDart**.

Join the growing list of organizations working with Flutter.

Google

GROUPON

 Alibaba Group
阿里巴巴集团

 Capital One

Tencent 腾讯

 Square

ebay



 DREAM11

SONOS

 nyu bank

EMAAR

THANK YOU!

Ashish Kupsad

Biju Amruta Dathan

Hemanth Reddy

Paritosh Gavali