



real life with

**Dart**

# **Dart** story

- originally developed by Google
- appeared: November 2013
- idea: replacement of JavaScript in browsers
- these plans were cancelled
- turned into general-purpose programming language for building web, server, mobile applications and IoT devices (fletch and flutter).

# **Dart** tools

- originally: Dart Editor (based on Eclipse)
- currently: WebStorm (plugin)
- other: Sublime, Atom
- pub - the package manager
- chromium - chrome with dart vm
- dart2js - compiler Dart -> JavaScript

# Dart pubspec.yaml

```
name: 'my-project'  
version: 0.0.1  
description: A web app built using polymer.dart.  
  
environment:  
  sdk: '>=1.9.0 <2.0.0'  
  
dependencies:  
  browser: ^0.10.0  
  polymer_elements: ^1.0.0-rc.1  
  polymer: ^1.0.0-rc.2  
  reflectable: ^0.3.1  
  web_components: ^0.12.0  
  
transformers:  
- web_components:  
  entry_points: web/index.html
```

# **Dart** language

- influenced by Java, C#, JavaScript, Smalltalk, Ruby, Erlang
- supports interfaces, mixins, abstract classes, generics, exceptions
- standard by Ecma (ECMA-408)

# **Dart** language

- functions are objects

```
[0, 1, 2, 3].where((n) => n.isEven).forEach(print);
```

# Dart language

- properties (getters & setters)

```
class Car {  
  
    // private  
    int _wheels = 4;  
  
    // getter  
    int get wheels => this._wheels;  
  
    // setter  
    void set wheels(int count) {  
        _wheels = count;  
    }  
}
```

# Dart language

- cascade operator (..)

```
var person = new Person()  
  ..name = "John"  
  ..surname = "Doe"  
  ..age = 26;
```

- if null operator (??)

```
String personName = person.name ?? "default name";
```

- null-aware operators (??=, ?.)

```
actor?.sing();  
actor?.name = "John";  
actor.name ??= "John";
```



# Dart language

- constructors

```
// regular constructor
Person(this.name, this.surname);
var person = new Person("John", "Doe");

// named constructor
Person.fromJson(String json) {
    ""
}
var person = new Person.fromJson(json);

// factory pattern
factory Person.create(String type) {
    switch(type) {
        case 'ACTOR':
            return new Actor();
        case 'SINGER':
            return new Singer();
    }
}
```

# Dart language

- implicit interfaces

```
class Vehicle {  
  bool hasEngine() {  
    return true;  
  }  
}
```

```
class Car implements Vehicle {  
  @override  
  bool hasEngine() {  
    return true;  
  }  
}
```

```
class Bike extends Vehicle {  
  
}
```

# Dart language

- mixins

```
abstract class SingerMixin {  
    String songName();  
    void sing() => print("singing ${songName()}");  
}
```

```
class Person {  
}
```

```
class Actor extends Person with SingerMixin {  
    @override  
    String songName() => "some song";  
}
```

# Dart async

- callback hell

```
connectToDb((connection) {  
  runDatabaseQuery(connection, (data) {  
    renderTable(data, () {  
      print("done");  
    });  
  });  
});
```

# Dart async

- Future (a.k.a Promise) represents a means for getting a value sometime in the future.
- Completer: a way to produce Future objects and to complete them later with a value or error.

```
class AsyncOperation {  
  
    Completer<int> _completer = new Completer<int>();  
  
    Future<int> doOperation() {  
        return _completer.future;  
    }  
  
    finishOperation(int result) {  
        _completer.complete(result);  
    }  
  
}
```

# Dart async

- old style (before 1.9 release)

```
Future<DbConnection> connectToDb()  
Future<Data> runDatabaseQuery(DbConnection connection)  
Future renderTable(Data data)
```

```
connectToDb()  
  .then((connection) {  
    return runDatabaseQuery(connection);  
  })  
  .then((data) {  
    renderTable(data);  
  })  
  .catchError((e) {  
    // process error  
  });
```

# Dart async

- async/await (since 1.9 release)

```
DbConnection connectToDb() async  
Data runDatabaseQuery(DbConnection connection) async  
void renderTable(Data data) async
```

```
doOperation() async {  
  try {  
    var connection = await connectToDb();  
    var data = await runDatabaseQuery(connection);  
    renderTable(data);  
  } catch (e) {  
    // process error  
  }  
}
```

# **Dart** mirrors

- The `dart:mirrors` library provide basic reflection abilities to Dart
- work for web apps and command-line apps
- causing `dart2js` generate very large JavaScript files
- frameworks & libraries they are using transformers for metadata
- status: unstable
- `reflectable`: promissing static and dynamic reflection



# **Dart** project

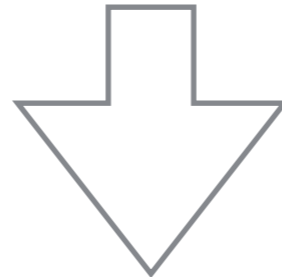
- limited resources (2-3 people, no JavaScript experience)
- special component (spreadsheet) needed
- backend in C#, REST API (no frontend rendering)
- single-page application
- time (4 months)
- we started with version 0.3, current version is 1.13



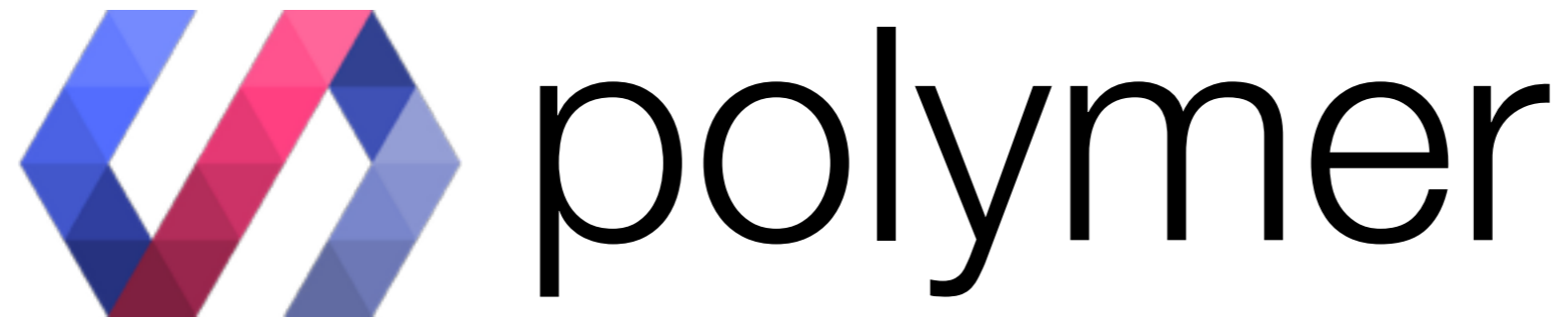
```
<div class="...">
  <div class="...">
    <div id="...">
      <div class="...">
        ...
      </div>
    </div>
  </div>
</div>
```



```
<div class="...">
  <div class="...">
    <div id="...">
      <div class="...">
        ...
      </div>
    </div>
  </div>
</div>
```



```
<spreadsheet>
  <cell row="A" column="1"></cell>
  <cell row="B" column="2"></cell>
</spreadsheet>
```



- library for creating Web Components (by W3C standard)
- Shadow DOM encapsulates and hides the innards of a custom element inside a nested document
- redesigned version 1.0

# **Dart** polymer

- originally pure dart library
- Shadow DOM: Hard to polyfill it, confusing, weak standard
- repeaters and 2-way binding didn't work
- missing some set of components like buttons, input fields, routing components etc.
- tricky styling (CSS) because Shadow DOM

# **Dart** pros & cons

- + learning curve
- + productivity
- + syntax / readability
- + package manager and build process
- Polymer/Shadow DOM
- very basic JSON library / missing set of components
- private properties
- mirrors (reflection) for web applications