

Dart For Java Developers

Yakov Fain, Farata Systems



Farata Systems and SuranceBay



FARATA THE EXPERT CONSULTANCY

Technologies supported:

- iOS
- Java
- ADOBE AIR
- open source
- android
- apache flex
- HTML5
- YUI

Enterprise Web Development

Enterprise Software Without the BS

Enterprise Development with Flex

Java Programming 24-Hour Trainer

Java 2 Enterprise Edition 1st Bible

ADOBE FLEX & JAVA

Java Tutorial for the real world

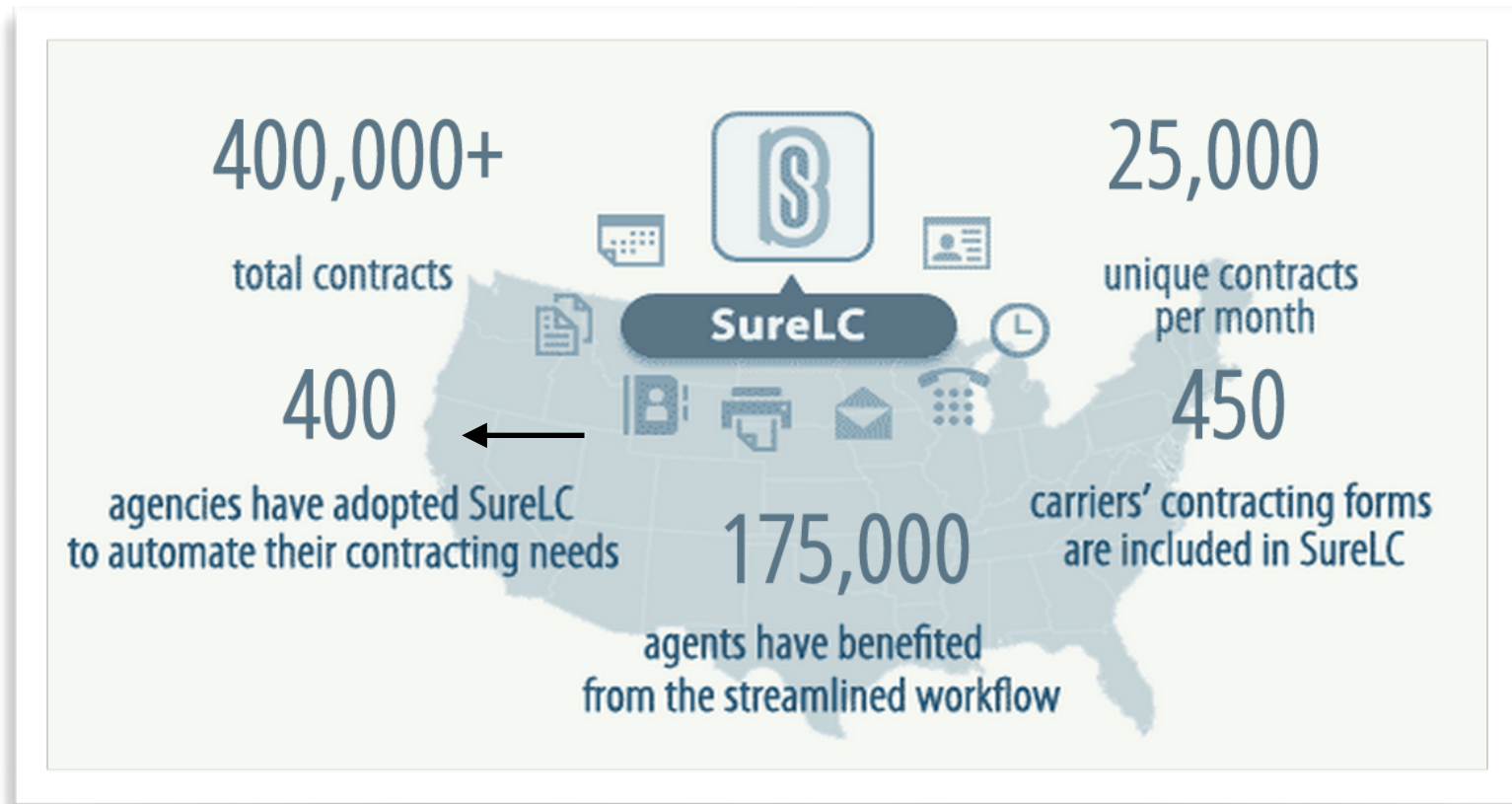
Программирование на JAVA

**We build applications. Every app is unique.
We create it. You own it.**

www.faratasystems.com

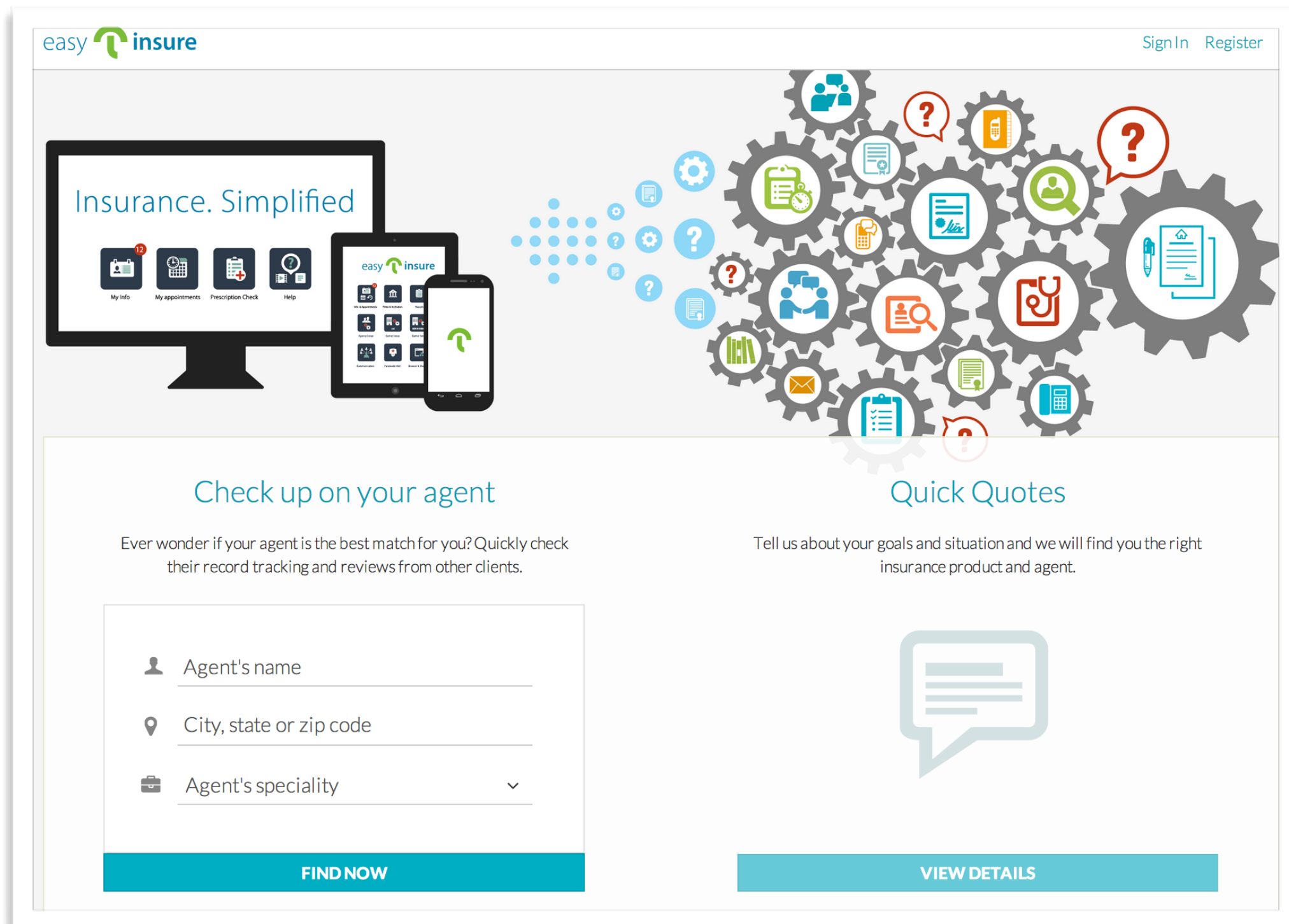
faratasystems.com

faratasystems.com



surancebay.com

Our First Dart App: Easy Insure



<http://easy.insure>

Why Learn Google Dart

- Its a productive way to develop JavaScript apps today
- Comes with a complete set of dev tools
- Will help you to ease into the EcmaScript 6 developement in 2016

Dart: Java Simplified

- Program with classes and/or with functions
- You can, but don't have to declare var types.
- Any class is an interface
- No data access qualifiers
- No checked exceptions
- No autoboxing/unboxing
- IDEs: Dart Editor, IntelliJ IDEA, WebStorm, Eclipse Sublime Text...
- Write in Dart and run either in Dart VM or as JavaScript on any browser

Dart VM

- You can run standalone apps in Dart VM
- Dartium, a special version of Chrome browser includes Dart VM
- Dart VM is created by engineers that used to work on Java VM

Why Dart VM if you'll
deploy JavaScript
anyway?



Why not just use
GWT?



Why Dart VM

- **Productive development.** Instantaneous feedback.
- No need to compile to JS to run the program in a browser as in GWT.
- Dart VM runs Dart, not the bytecode
- Your program can run on the server in Dart VM.

Demo

HelloWorld as a console and Web app

Generating and running the app in Dart Editor and IDEA

The Dart App Entry Point

A single entry point to any Dart app is a function `main()`:

```
main() {  
  print( 'Hello world!' );  
}
```

The Dart App Entry Point

A single entry point to any Dart app is a function `main()`:

```
main() {  
    print( 'Hello world!' );  
}
```

Or with command-line arguments:

```
import 'package:args/args.dart';  
  
main(List<String> args) {  
  
    final parser = new ArgParser();  
    argResults = parser.parse(args);  
  
    List<String> someArgs = argResults.rest;  
  
    print( 'Got the argument ${someArgs[0]}' );  
  
}
```

What this code will print?

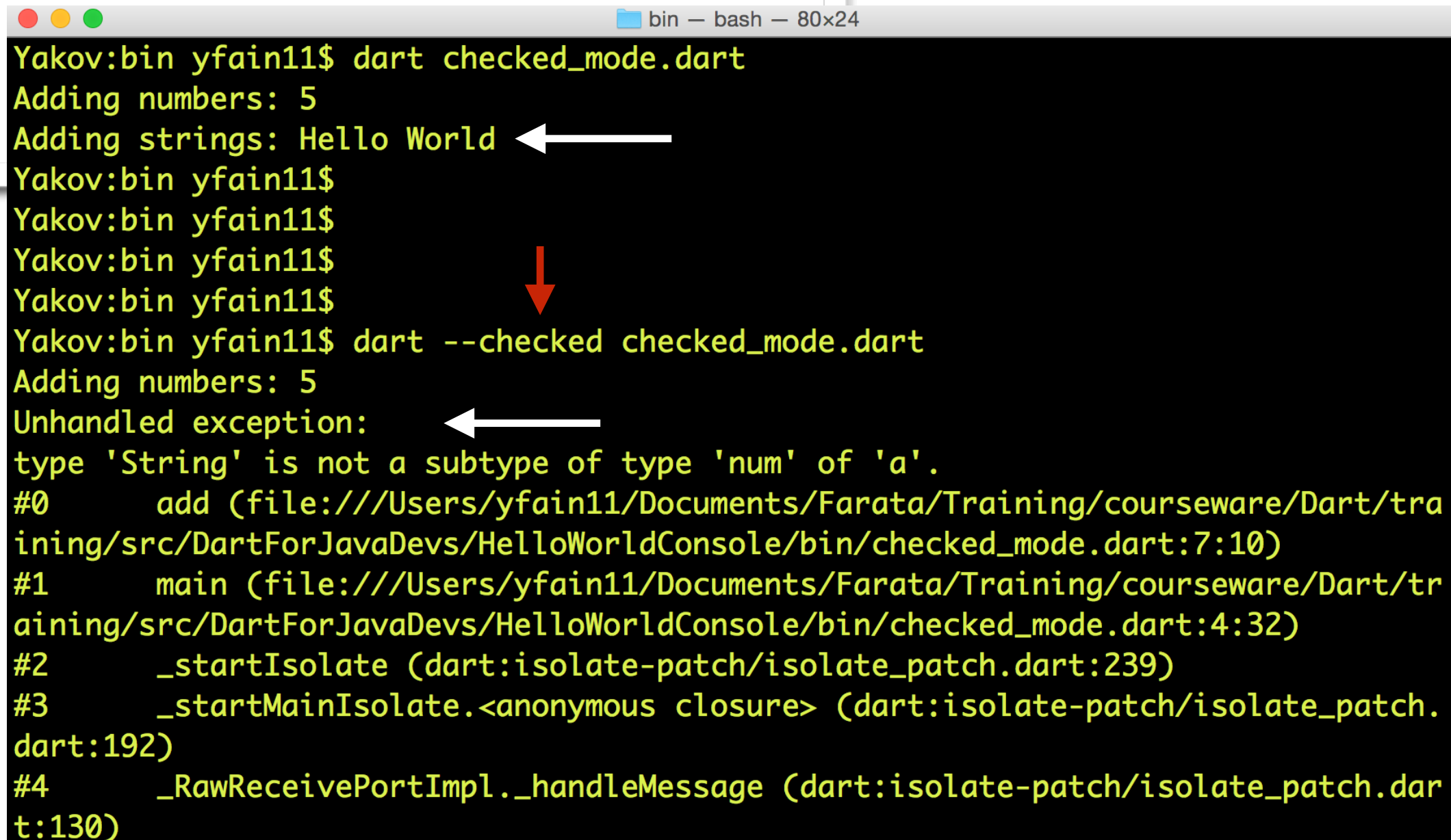
```
main() {  
    print('Adding numbers: ${add (2,3)}');  
    print('Adding strings: ${add ("Hello ", "World")}');  
}  
  
add (num a, num b){  
    return a+b;  
}
```

`$variableName (or ${expression})`

String interpolation: including a variable or expression's string equivalent inside of a string literal.

In Checked Mode: runtime error

```
main() {  
  print('Adding numbers: ${add (2,3)}');  
  print('Adding strings: ${add ("Hello ", "World")}');  
}  
  
add (num a, num b){  
  return a+b;  
}
```



A terminal window titled 'bin — bash — 80x24' showing the execution of Dart code. The first command is 'dart checked_mode.dart', which outputs 'Adding numbers: 5' and 'Adding strings: Hello World'. A white arrow points to the second output. The second command is 'dart --checked checked_mode.dart', which outputs 'Adding numbers: 5' and then an 'Unhandled exception:'. A white arrow points to the exception message. A red arrow points from the first command to the second. The stack trace shows the error originates from the 'add' function in 'checked_mode.dart' at line 7:10, where a 'String' is passed to a function expecting a 'num'.

```
Yakov:bin yfain11$ dart checked_mode.dart  
Adding numbers: 5  
Adding strings: Hello World  
Yakov:bin yfain11$  
Yakov:bin yfain11$  
Yakov:bin yfain11$  
Yakov:bin yfain11$  
Yakov:bin yfain11$ dart --checked checked_mode.dart  
Adding numbers: 5  
Unhandled exception:  
type 'String' is not a subtype of type 'num' of 'a'.  
#0      add (file:///Users/yfain11/Documents/Farata/Training/courseware/Dart/training/src/DartForJavaDevs/HelloWorldConsole/bin/checked_mode.dart:7:10)  
#1      main (file:///Users/yfain11/Documents/Farata/Training/courseware/Dart/training/src/DartForJavaDevs/HelloWorldConsole/bin/checked_mode.dart:4:32)  
#2      _startIsolate (dart:isolate-patch/isolate_patch.dart:239)  
#3      _startMainIsolate.<anonymous closure> (dart:isolate-patch/isolate_patch.dart:192)  
#4      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:130)
```

Importing Packages

- SDK comes with packages whose names start with `dart`:

```
import 'dart:math';
```

- Other packages located in the directory *packages* of your project. can be included as app dependencies. Imports start with `package`:

```
import 'package:args/args.dart';
```

Package Dependencies

- Dart package manager is called **pub**
- Dependencies are specified in the file **pubspec.yaml**.
- Package versions are locked in the file **pubspec.lock**.

pubspec.yaml

```
name: StockQuoteConsole
version: 0.0.1
description: A stock quote app
environment:
  sdk: '>=1.0.0 <2.0.0'
dependencies:
  args: any
dev_dependencies:
  unittest: any
```

The central repo pub.dartlang.org has 1500+ packages

Selected pub commands

- `pub get` - retrieves packages (dependencies)
- `pub upgrade` - upgrades packages and regenerates `pubspec.lock`
- `pub serve` - starts dev http server
- `pub run` - runs Dart scripts using dependencies from `pubspec.yaml`
- `pub build` - generates and copies assets into the build dir
- `pub global` - run Dart packages installed anywhere on the computer,
pub.dartlang.org or GitHub

More on pub at <http://bit.ly/1wDMhTi>

Demo

Stock Quote Generator. Take 1.

Functions only. Using pub to get dependencies. Command-line arguments.

Dart Classes

- Files names are in small letters with `_` as word separator
- Constructors support short syntax for variable initializations
- No keywords for `public`, `protected`, `private`.
- If a var name start wit `_` it's private on a library level
- No method overloading
- Getters and setters specified with `get` and `set` keywords

Class Stock

private vars

constructor

lazy getter

setter

```
class Stock {  
  String _symbol;  
  double _price;  
  
  Stock (this._symbol);  
  
  double get price => _price==null?  
    _price=getFromYahoo():_price;  
  
  set price(double value){  
    _price = value;  
  }  
  
  String get symbol => _symbol;  
  
  set symbol(String value){  
    _symbol = value;  
  }  
}
```

Class Stock

private vars

constructor

lazy getter

setter

```
class Stock {  
    String _symbol;  
    double _price;  
  
    Stock (this._symbol);  
  
    double get price => _price==null?  
        _price=getFromYahoo():_price;  
  
    set price(double value){  
        _price = value;  
    }  
  
    String get symbol => _symbol;  
  
    set symbol(String value){  
        _symbol = value;  
    }  
}
```

```
Stock stock = new Stock();  
var price = stock.price;
```

Constructors

- Short form with `this`
- Optional parameters
- Named constructors
- factory constructors

Constructors

short form →

optional param →

named →

factory →

```
class Customer {  
  
    int _id;  
    String name;  
  
    Customer(this._id, this.name);  
  
    Customer.optName(this._id, {this.name});  
  
    Customer.taxExempt(int id, String name){  
        // Do something  
    }  
  
    factory Customer.mafia(int id, String name){  
        if (name == "Don Carleone")  
            return new Customer.taxExempt(id, name);  
        else  
            return new Customer(id, name);  
    }  
}
```

Code Structure

deploy/version →

Package

import →

Libraries

Classes




Functions

Interfaces

Mixins

Libraries

You can encapsulate classes and top-level functions into libraries.




```
library stock_quote;

import 'dart:math';
import 'dart:io';
import 'package:args/args.dart';


part "stock.dart";
part "stock_quote_generator.dart";

main(List<String> args) {
  ...
}
```



```
part of stock_quote;

class Stock {
  ...
}
```



```
part of stock_quote;

class StockQuoteGenerator {
  ...
}
```

Dart Libraries

- `dart:core`
- `dart:async`
- `dart:io`
- `dart:html`

Demo

Stock Quote Generator. Take 2.

Classes, getters, setters, library.

Cascade Operator ..

You can use method cascades `..` on any object. Every `..` refers to the original object, not to the result of the previous method.

Exceptions

- All exceptions are unchecked
- You can throw any objects:
`throw "Something happened";`

```
try {  
    // Do stuff here  
} on NoSuchMethodError catch (e) {  
    print('Error: ${e.stackTrace}');  
} on RangeError catch (e) {  
    print('Error: ${e.stackTrace}');  
} on TypeError catch (e) {  
    print('Error: ${e.stackTrace}');  
} catch (e) {  
    print('$e');  
}
```

Web Apps

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>My Web App</title>
```

```
</head>
```

```
<body>
```

Your HTML content goes here

```
<script type="application/dart" src="main.dart"></script>
```

```
<script data-pub-inline src="packages/browser/dart.js"></script>
```

```
</body>
```

```
</html>
```

For browsers with Dart VM



JavaScript generation for browsers without Dart VM



Running Dart Web App

1. **From a command line:**

`pub serve` and refresh the Web page

2. **From IDEA:**

Right-click on your `index.html` file and open (or run) it in any Web browser

Running Web app with pub serve

Running
pub serve

Visiting
localhost:8080
in Dartium

Visiting
localhost:8080
in Chrome

```
StockQuoteSimpleWeb — dart — 80x29
Yakov:StockQuoteSimpleWeb yfain11$ pub serve
Loading source assets...
Serving stock_quote_simple_web web on http://localhost:8080
Build completed successfully
  GET / → stock_quote_simple_web/web/index.html
  GET /main.dart → stock_quote_simple_web/web/main.dart
  GET /packages/browser/dart.js → browserlib/dart.js
  GET /styles/main.css → stock_quote_simple_web/web/styles/main.css
  GET /packages/stock_quote_simple_web/stock.dart → stock_quote_simple_weblib/stock.dart
  GET /packages/stock_quote_simple_web/stock_quote_generator.dart → stock_quote_simple_weblib/stock_quote_generator.dart
  GET /favicon.ico → Could not find asset stock_quote_simple_web/web/favicon.ico.
  GET / → stock_quote_simple_web/web/index.html
  GET /packages/browser/dart.js → browserlib/dart.js
  GET /styles/main.css → stock_quote_simple_web/web/styles/main.css
[Info from Dart2JS]:
→ Compiling stock_quote_simple_web/web/main.dart...
[Info from Dart2JS]:
Took 0:00:05.553774 to compile stock_quote_simple_web/web/main.dart.
Build completed successfully
  GET /main.dart.js → stock_quote_simple_web/web/main.dart.js
  GET /main.dart.js.map → stock_quote_simple_web/web/main.dart.js.map
  GET / → stock_quote_simple_web/web/index.html
  GET /styles/main.css → stock_quote_simple_web/web/styles/main.css
  GET /packages/browser/dart.js → browserlib/dart.js
  GET /main.dart.js → stock_quote_simple_web/web/main.dart.js
  GET /main.dart.js.map → stock_quote_simple_web/web/main.dart.js.map
```


Debugging Dart

1. Run **pub serve** from the dir containing *pubspec.yaml*.
2. It'll run **pub build** and will deploy the app at <http://localhost:8080>
3. Debug the app in the browser or in IDE, for example:

- In Chrome:

- Enter the URL <http://localhost:8080>
- Open Chrome Development Tools, enable JavaScript sourcemaps
- Set breakpoints, refresh.

- In IntelliJ IDEA:

- In Chrome install the extension JetBrains IDE Support
- In IDEA go to the menu Run | Edit Configuration | + | JavaScript Debug, give it a name and the URL <http://localhost:8080>
- Set the breakpoint in IDEA in Dart code and click on Debug.

Demo

Stock Quote Generator. Take 3.

Web app. Working with DOM. Event Handling.

Working with DOM in a Browser

```
<body>
  Enter Symbol: :
  <input id="enteredSymbol" type="text">

<script type="application/dart" src="main.dart"></script>
<script data-pub-inline src="packages/browser/dart.js"></script>
</body>
```

```
import 'dart:html';

InputElement enteredSymbol;

void main() {
  InputElement enteredSymbol = querySelector("#enteredSymbol");
}
```

Event Handling

```
Element myHtmlElement = querySelector("#myElementID");
```

```
myHtmlElement.onChange.listen(myEventHandler);
```



```
void myEventHandler(Event e){  
    // Handle event here  
}
```

A Simple Stock Quote Web App

```
<body>
  Enter Symbol: :
  <input id="enteredSymbol" type="text" placeholder="AAPL, IBM, or MSFT">

  <span id="priceQuote"></span>

  <script type="application/dart" src="main.dart"></script>
  <script data-pub-inline src="packages/browser/dart.js"></script>
</body>
```

```
import 'dart:html';
import 'package:stock_quote_simple_web/stock.dart';
import 'package:stock_quote_simple_web/stock_quote_generat
```

```
StockQuoteGenerator generator = new StockQuoteGenerator();
```

```
InputElement enteredSymbol;
SpanElement priceQuote;
```

```
void main() {
```

```
  enteredSymbol = querySelector("#enteredSymbol");
  priceQuote = querySelector("#priceQuote");
```

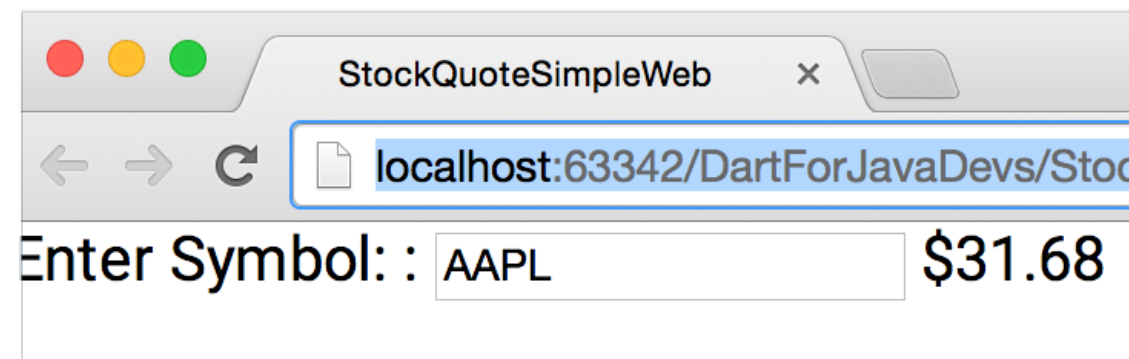
```
  enteredSymbol.onChange.listen(showPrice);
```

```
}
```

```
void showPrice(Event e){
```

```
  Stock stock = generator.getQuote(enteredSymbol.value);
  priceQuote.text = stock.price.toString();
```

```
}
```

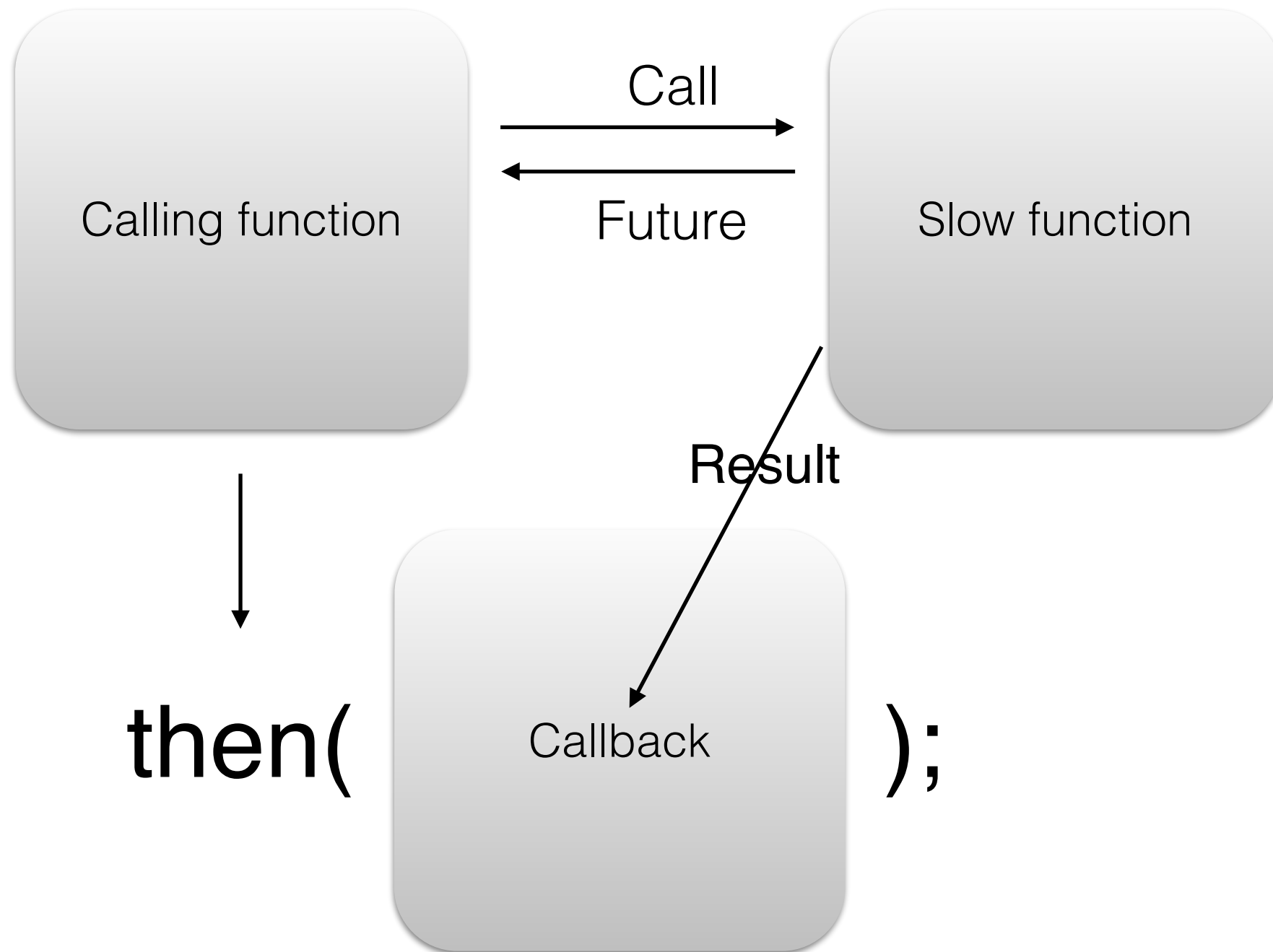


DOM search

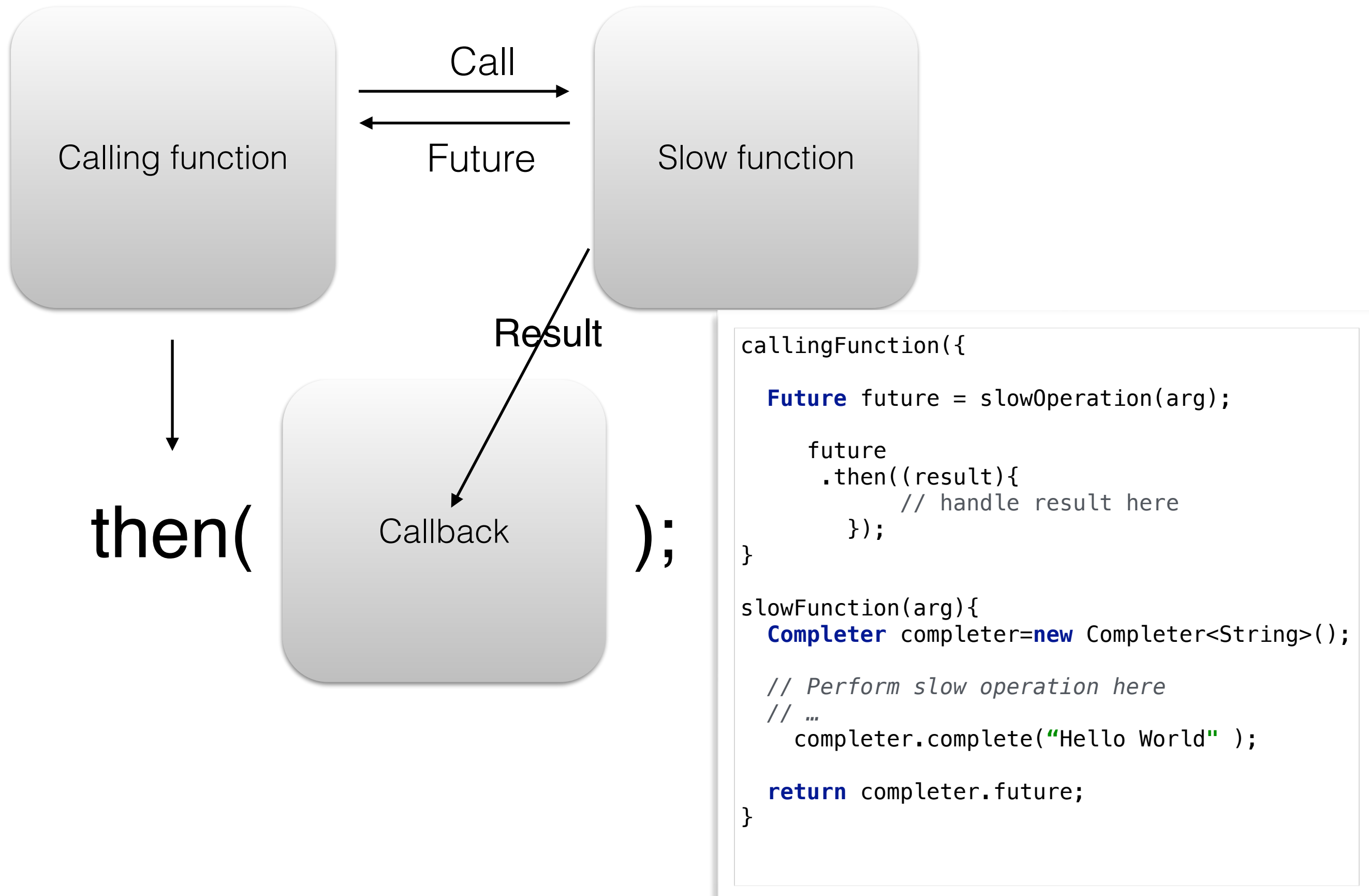
Event listener

Event handler

Async Processing: Futures



Async Processing: Futures



Demo

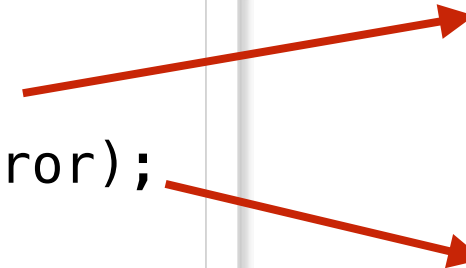
Stock Quote Generator. Take 4.

Web app. Calling a slow operation asynchronously using Future.

Futures and Error Handling

- A `Future` represent a deferred result of a function call
- Register callbacks for success and errors

```
doStuff()  
  .then(callbackForSuccess)  
  .catchError(callBackForError);
```

Two red arrows originate from the code on the left. One arrow points from `callbackForSuccess` to the definition of `callbackForSuccess` in the code block on the right. The other arrow points from `callBackForError` to the definition of `callbackForError` in the same code block on the right.

```
void callbackForSuccess() {  
  // ...  
}  
  
void callbackForError(Error error){  
  // ...  
}
```

AJAX:HttpRequest

Demo

Stock Quote Generator. Take 5.

Ajax + JSON.

Concurrency with Isolates

- Isolates are units of security
- Each isolate has its own heap - no shared memory
- Isolates communicate with each other via ports by sending messages

Isolates: Standalone vs Web Browser

Standalone Apps

- run isolates in parallel using available CPU cores
- isolates can be created by invoking `spawn()` or `spawnUri()`

Web Browser Apps

- run isolates in Dart VM or as JavaScript Web workers
- isolates can be created by invoking `spawnUri()`

Isolates: Standalone vs Web Browser

Standalone Apps

- run isolates in parallel using available CPU cores
- isolates can be created by invoking `spawn()` or `spawnUri()`

Web Browser Apps

- run isolates in Dart VM or as JavaScript Web workers
- isolates can be created by invoking `spawnUri()`

Use `spawnUri()` to load Dart code dynamically

Demo

Using isolates with `spawn()` and `spawnUri()`

Mixins

Generics

Streams

Functions

- Top-level functions
- You can pass a funct as a arg to another func
- You can return a function from another func
- Single-line functions =>
- functions with optional params (positionals and named):
 myFunct (String a, [int b], [int c])
 myFunc("Mary, 2");
myFunc (String a, {int b, int c})
 myFunc("Mary", c:3);

dart2js Transpiler

- Tree shaking
- File Watcher in IDEA <https://www.jetbrains.com/idea/help/transpiling-dart-to-javascript.html>

Dart Ecosystem

Links

- Style Guide <https://www.dartlang.org/articles/style-guide>
- Dart API docs: <https://api.dartlang.org>
- Try Dart: <try.dartlang.org>
- Hands-on labs: dartlang.org/codelabs
- List of languages that compile to JS:
<https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS>
- Dart Language Specification:
- <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-408.pdf>

More Links

- Our Dart app: <https://easy.insure>
- Farata Systems: faratasystems.com
- Twitter: @yfain
- Personal blog: yakovfain.com

