

# 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 Development with Flex**

**Enterprise Software Without the BS**

**Java 2 Enterprise Edition 1st Bible**

**Java Programming 24-Hour Trainer**

**ADOBE FLEX & JAVA**

**Java Tutorial for the real world**

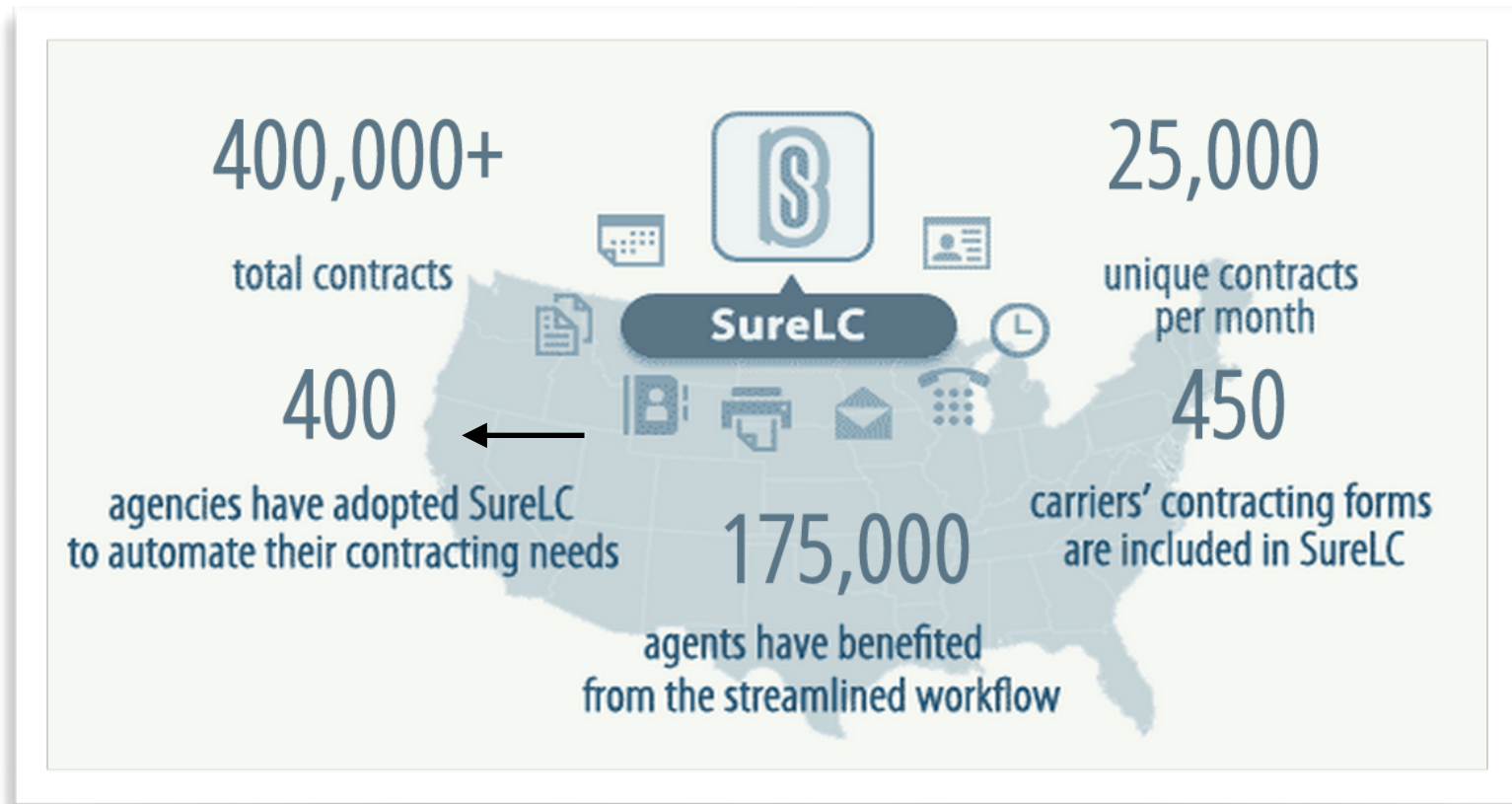
**Программирование на JAVA**

**Мы создаем приложения. Каждое приложение уникально. Мы создаем его. Вы владеете им.**

[www.faratasystems.com](http://www.faratasystems.com)

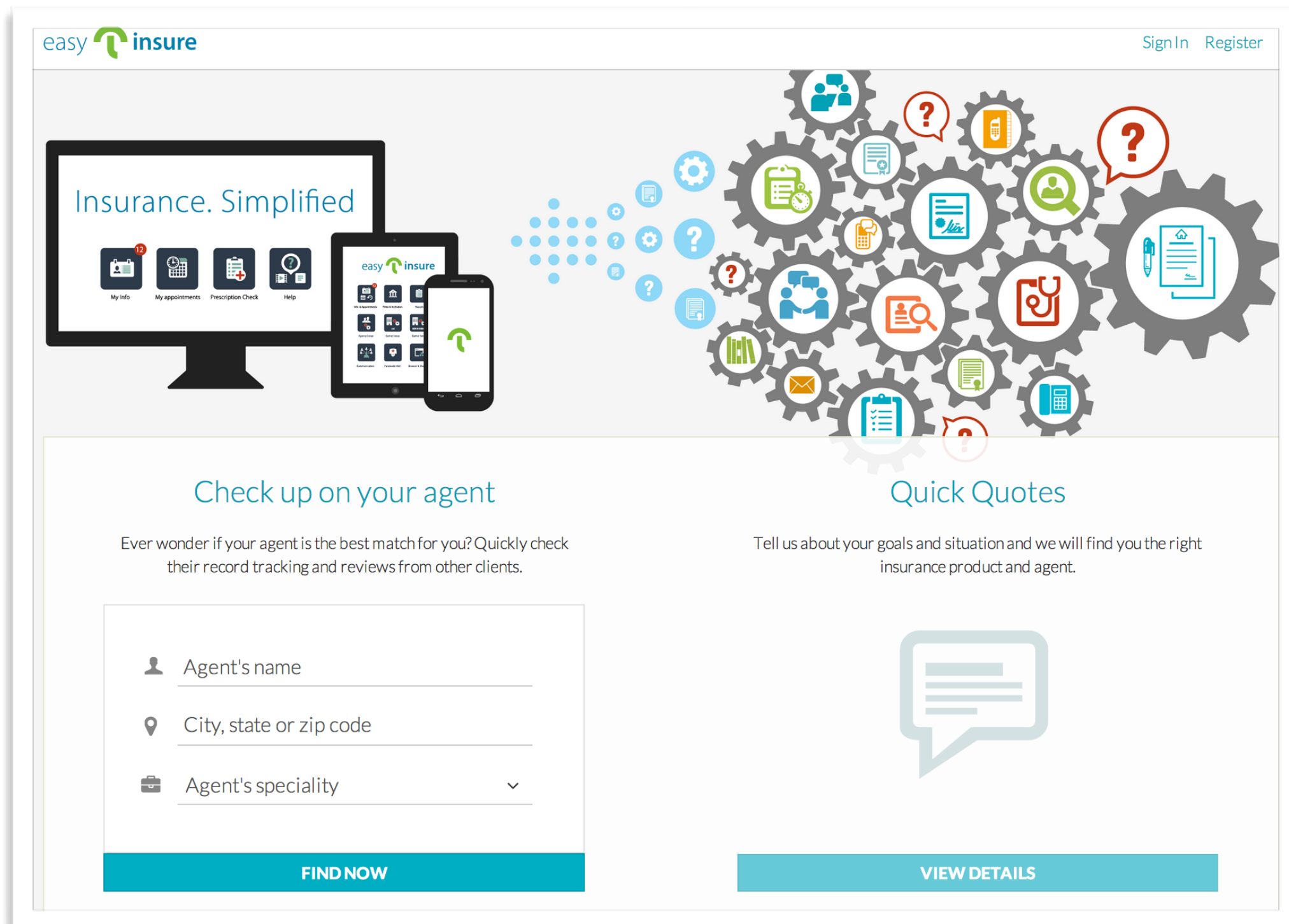
faratasystems.com

[faratasystems.com](http://faratasystems.com)



[surancebay.com](http://surancebay.com)

# Our First Dart App: Easy Insure



<http://easy.insure>

# Why Learn Google Dart

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

# Dart: Java Simplified

- Program with classes and/or with functions
- You can, but don't have to declare variable types.
- Any class is an interface
- No data access qualifiers
- Single-threaded but supports concurrency
- 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

# Why Developing with Dart VM ?

- Developing in Dart is more productive than in JavaScript. Static code analyser shows errors as you type.
- Instantaneous feedback from the VM.
- No bytecode is generated for running in Dart VM.
- During development there's no need to transpile code to JS as in GWT.
- Your program can run standalone in Dart VM.
- Production deployment is done in JavaScript.

# 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!' );  
}
```

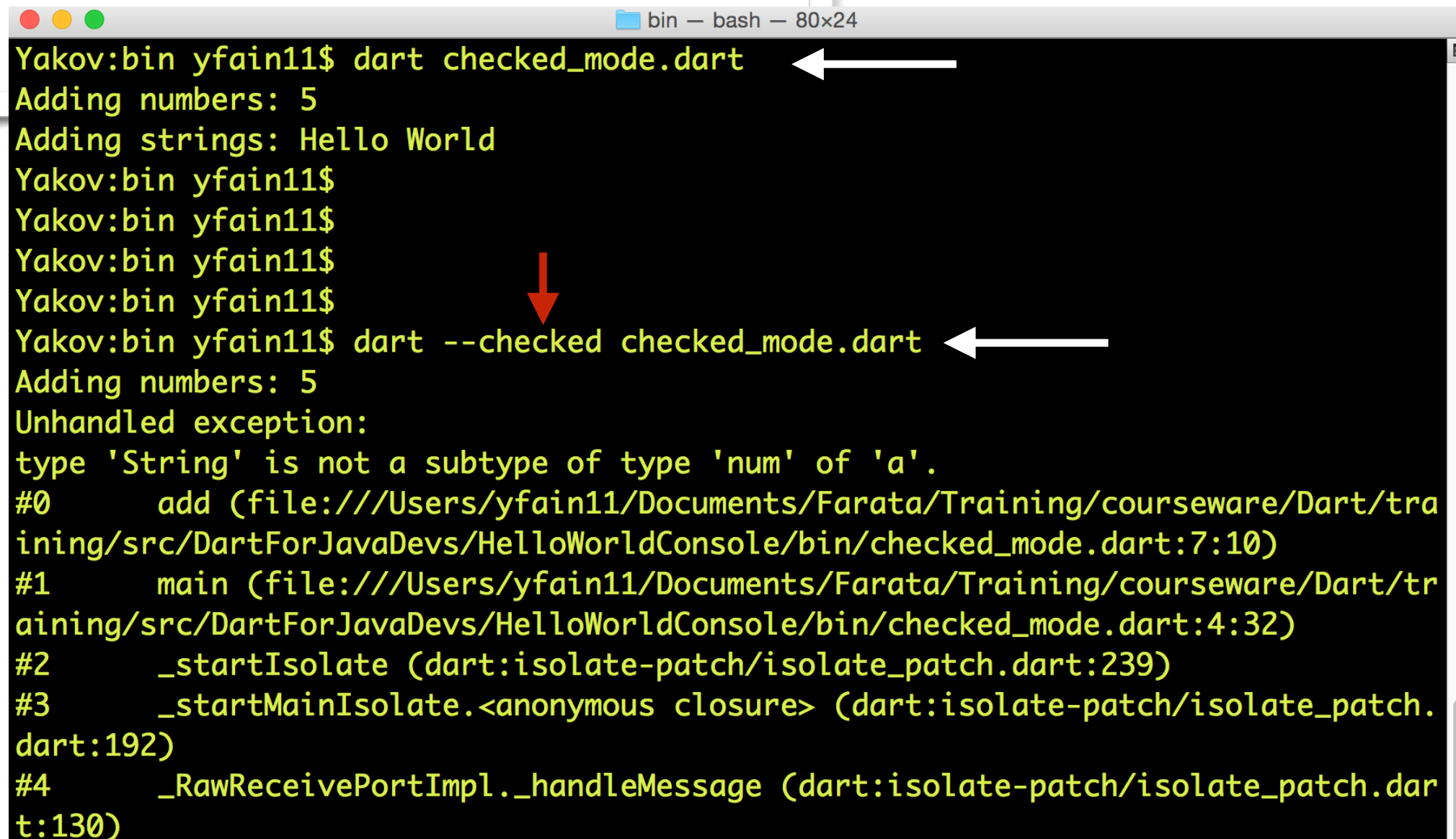
Or with command-line arguments:

```
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]}' );  
  
}
```



# Checked Mode

```
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 runs successfully and outputs "Adding numbers: 5" and "Adding strings: Hello World". The second command is `dart --checked checked_mode.dart`, which fails with an "Unhandled exception: type 'String' is not a subtype of type 'num' of 'a'". A red arrow points to the `--checked` flag in the second command. White arrows point to the command lines. The stack trace shows the error originates from the `add` function in `checked_mode.dart` at line 7:10, which is called from `main` at line 4:32.

```
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 libraries whose names start with `dart`:

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

- You can add packages in the directory *packages* of your project are app dependencies. Imports start with `package`:

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

# Dart Libraries

- `dart:core`
- `dart:async`
- `dart:math`
- `dart:io`
- `dart:html`
- `dart:mirrors`
- `dart:convert`

# 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: stock_quote_console
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](http://pub.dartlang.org) has 1000+ 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](https://pub.dartlang.org), or GitHub

# Demo

Functions-only app

Using pub to get dependencies

Command-line arguments

# Dart Classes

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

# Class Stock

**private vars** → `class Stock {  
    String _symbol;  
    double _price;`

**constructor** → `Stock (this._symbol);`

**lazy getter** → `double get price => _price==null?  
                                _price=getFromYahoo():_price;`

**setter** → `set price(double value){  
        _price = value;  
    }`

`String get symbol => _symbol;`

`set symbol(String value){  
        // some other processing may go here  
        _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("IBM");  
var price = stock.price;
```

# Constructors

short form →

named →

optional param →

factory →

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

# Cascade Operator ..

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

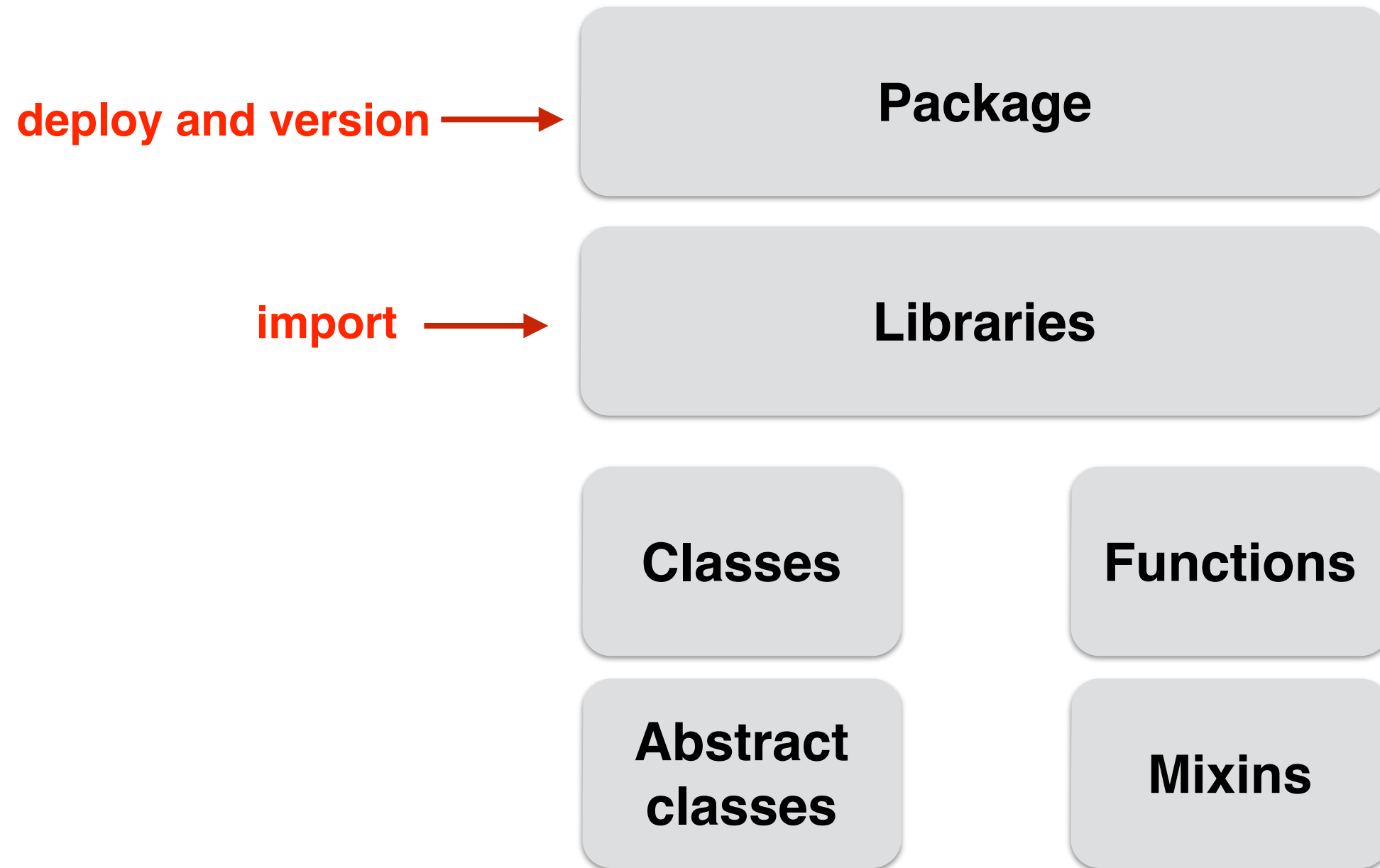
```
querySelector('#abutton') // Get an object
  ..text = 'Confirm'      // Use its members
  ..classes.add('important')
  ..onClick.listen((e) => window.alert('Confirmed!'));
```

# 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');  
}
```

# Code Structure



# Libraries

Combine 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 {
  ...
}
```

# Demo

- Classes
- Getters
- Setters
- Libraries

# Mixins

- Mixin is a class with functionality that can be added to another class using the `with` keyword
- You can add multiple mixins to the class declaration

```
class Bond extends Security with TradeReporter{  
}
```

Java 8 interfaces are not mixins. They can't have state.



# Mixin Sample

```
class Security{  
    String name;  
    Security(this.name);  
}
```

```
class Stock extends Security{  
    String symbol;  
    double previousClosingPrice;  
    Stock(String name): super(name);  
}
```

```
class Bond extends Security  
    → with TradeReporter{  
    double faceValue;  
    DateTime maturityDate;  
    Bond(String name): super(name);  
}
```

# Mixin Sample

```
class Security{  
    String name;  
    Security(this.name);  
}
```

```
class Stock extends Security{  
    String symbol;  
    double previousClosingPrice;  
    Stock(String name): super(name);  
}
```

```
class Bond extends Security  
    → with TradeReporter{  
    double faceValue;  
    DateTime maturityDate;  
    Bond(String name): super(name);  
}
```

```
class TradeReporter{  
    String whereToReport;  
    reportMuniBondTrade(name){  
        print('Trade for municipal bond $name has been reported to $whereToReport');  
    }  
    reportCorpBondTrade(name){  
        print('Trade for corporate bond $name has been reported to $whereToReport');  
    }  
}
```

# A Mixin Sample

```
class Security{  
    String name;  
    Security(this.name);  
}
```

```
class Stock extends Security{  
    String symbol;  
    double previousClosingPrice;  
    Stock(String name): super(name);  
}
```

```
class Bond extends Security  
    with TradeReporter{  
    double faceValue;  
    DateTime maturityDate;  
    Bond(String name): super(name);  
}
```

```
class TradeReporter{  
    String whereToReport;  
    reportMuniBondTrade(name){  
        print('Trade for municipal bond $name has been reported to $whereToReport');  
    }  
    reportCorpBondTrade(name){  
        print('Trade for corporate bond $name has been reported to $whereToReport');  
    }  
}
```

```
Bond bond = new Bond('Metropolitan Transportation');  
bond.whereToReport = "SEC"; // from mixin  
bond.reportMuniBondTrade('Metropolitan Transportation'); // from mixin
```

# Demo

## Mixins

# Web Apps Development

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

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

```
</head>
```

```
<body>
```

Your HTML content goes here

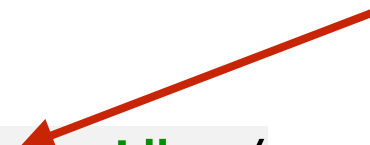
```
<script type="application/dart" src="main.dart"></script>
```

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

```
</body>
```

```
</html>
```

For development with Dartium



# Web Apps Deployment

```
<!DOCTYPE html>

<html>
<head>
  <title>My Web App</title>
</head>

<body>

  Your HTML content goes here

  <script src="main.dart.js"></script>

</body>
</html>
```

**For deployment, manually precompile you code with dart2js  
or add a transformer to pubspec.yaml**

```
dependencies:
  browser: any
  dart_to_js_script_rewriter: any
transformers:
  - dart_to_js_script_rewriter
```

# Running Dart Web App

## 1. **From a command line:**

Run `pub serve` and refresh the Web page

## 2. **From IDEA:**

Right-click on your `index.html` file and 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
```



# 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';

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");
```

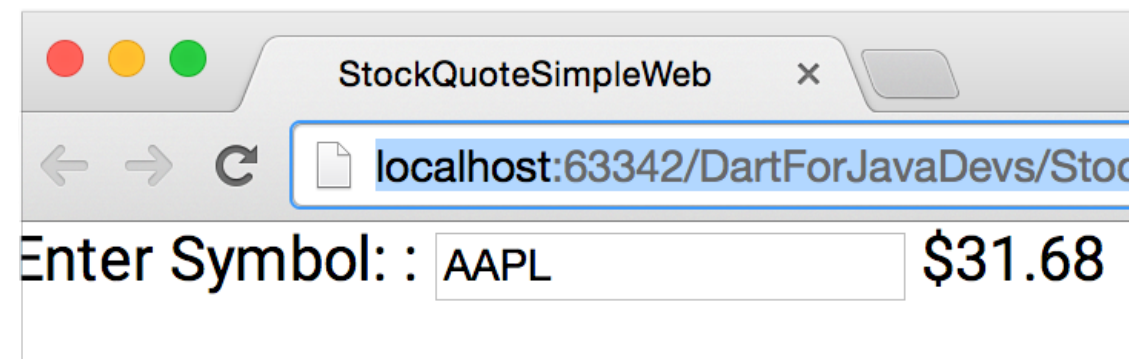
```
  enteredSymbol.onChange.listen(showPrice);
```

```
}
```

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

```
  Stock stock = generator.getQuote(enteredSymbol.value);
  priceQuote.text = "\${stock.price.toStringAsFixed(2)}";
```

```
}
```



DOM search

Event listener

Event handler

# Demo

Web app

Working with DOM

Event Handling

pubspec.yaml

```
name: stock_quote_simple_web
version: 0.0.1
description: >
  An absolute bare-bones web app.
environment:
  sdk: '>=1.0.0 <2.0.0'
dependencies:
  browser: any
transformers:
- $dart2js:
    commandLineOptions: [--minify, --show-package-warnings]
```

# Debugging Dart Web App

- In Chromium:
  - Open Chrome Development Tools, enable JavaScript sourcemaps
  - Set breakpoints, refresh.
- In IntelliJ IDEA:
  - Install the extension JetBrains IDE Support in Chromium.
  - Set the breakpoint in IDEA in Dart code and run your HTML file in the debug mode.

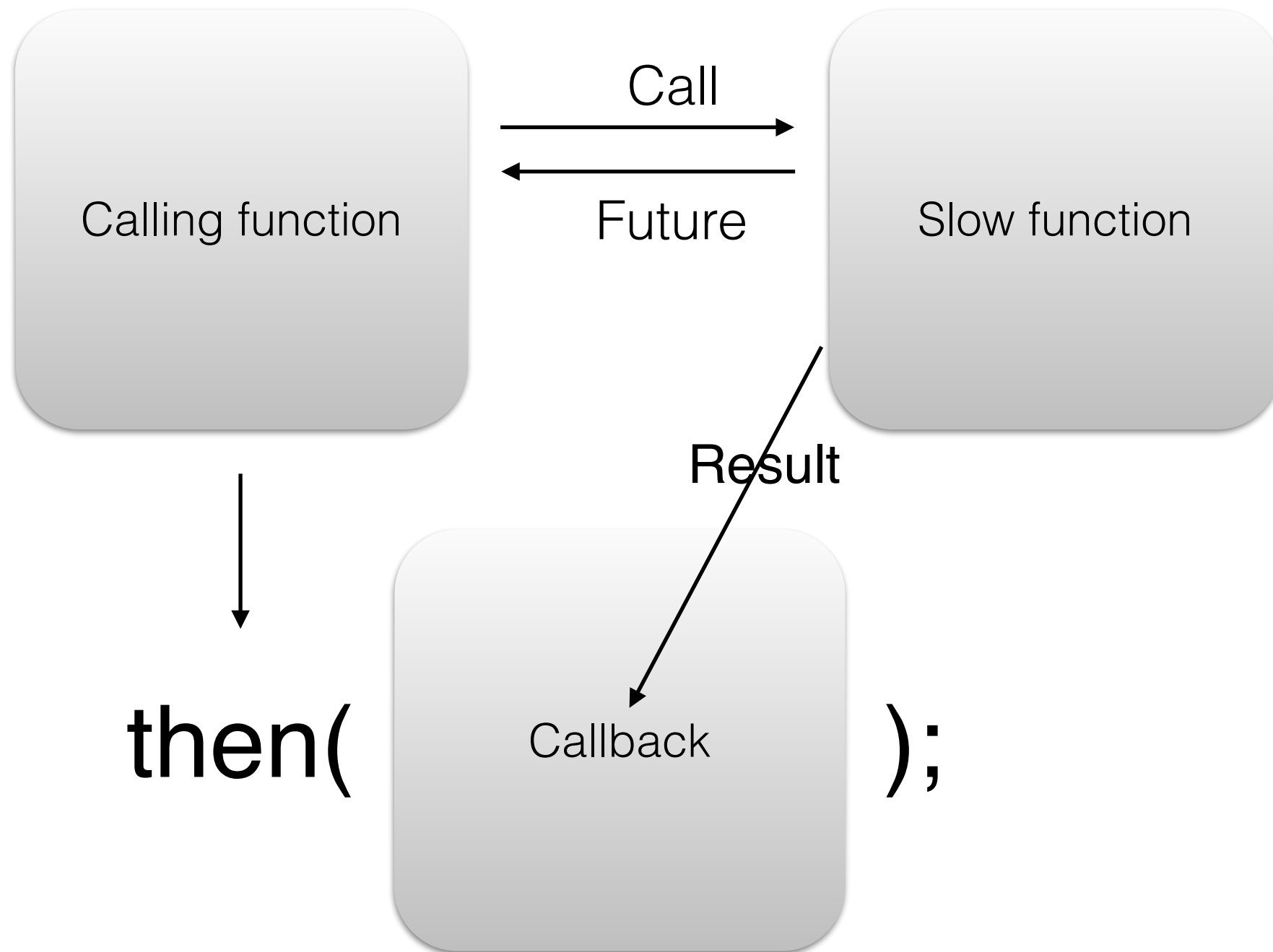
# Async Programming

- Use `Future` objects
- Use `async` and `await` keywords

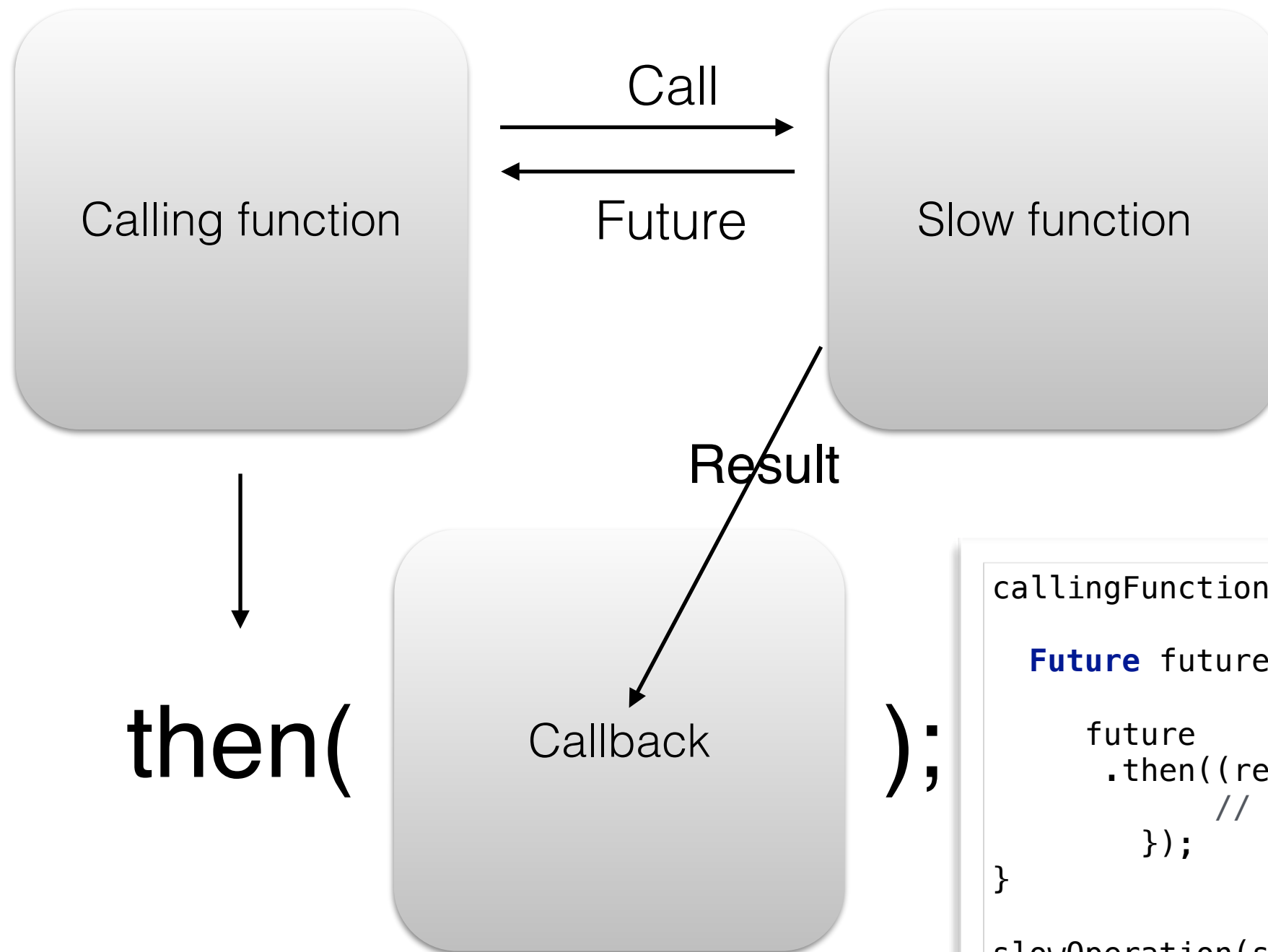
`Future`, `async`, and `await` are for **asynchronous** execution.

For **parallel** execution use `isolates`.

# Async Processing: Futures



# Async Processing: Futures



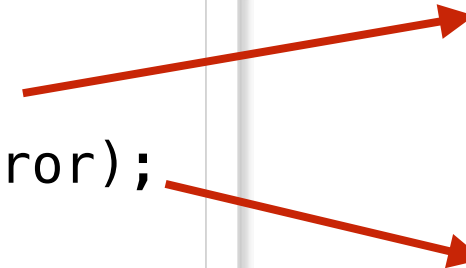
```
callingFunction(){  
    Future future = slowOperation("IBM");  
    future  
        .then((result){  
            // handle result here  
        });  
}  
  
slowOperation(stockSymbol){  
    return new Future.delayed(  
        const Duration(seconds: 10), () {  
            return "$stockSymbol is a great investment!";  
        });  
}
```



# Error Handling in Future Objects

- A `Future` represents 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){  
  // ...  
}
```

# Demo

Web app

Asynchronous processing using Future

# Asynchronous Processing with `async` and `await`

- A function with `async` modifier immediately returns a `Future`
- The code after `await` will wait for the result
- You can have several sequential operations with `await`
- The code flow is easier to read comparing to callbacks, e.g. you can place all `awaits` in the `try/catch` block if need be.

```
callingFunction() async{  
  
    var result = await slowOperation(arg);  
  
    // the code will be suspended until  
    // the result is returned  
  
    // Handle the result here  
}  
  
slowOperation(stockSymbol){  
  
    return new Future.delayed(  
        const Duration(seconds: 10), () {  
            return "$stockSymbol is a great investment!";  
        });  
}
```

# Demo

Asynchronous processing using `async`  
and `await`

# AJAX:HttpRequest

HttpRequest class from dart:html is used for AJAX operations from a Web browser:

```
var path = 'myData.json';

HttpRequest.getString(path)
  .then((data) {
    // do something with data
  })
  .catchError((Error error) {
    print(error.toString());
  });
```

The package http.dart can be used on both the client and server.

# Demo

## AJAX and JSON

# Concurrency with Isolates

- Isolates are units of secure execution
- The code can only access classes/values located in the same isolate
- Each isolate has its own heap - no shared memory
- The code in separate isolates can execute in parallel
- Isolates can communicate by sending each other messages via **send** and **receive** ports

# 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 dynamically load code

# Demo

Parallel execution with isolates

Using multiple CPU cores

# Demo

1. Reading a large JSON file with HttpRequest - GUI doesn't freeze
2. Running a long Dart loop with async - GUI freezes
3. Running a long Dart loop in isolate - GUI doesn't freeze

# Dart-WebSocket-Java

Dart client's getting data pushed by the GlassFish server:

```
main() {  
    var output = querySelector('#output');  
    WebSocket ws = new WebSocket(  
        'ws://localhost:8080/GlassfishWebsocketDart_war_exploded/clock');  
    ws.onOpen.listen((event){  
        output.text = "Connected";  
    });  
    ws.onMessage.listen((event){  
        output.text = event.data;  
    });  
}
```

# Java EE WebSocket Endpoint

```
@ServerEndpoint("/clock")
public class WebSocketClock {

    static ScheduledExecutorService timer =
        Executors.newSingleThreadScheduledExecutor();

    private static Set<Session> allSessions;

    DateTimeFormatter timeFormatter =
        DateTimeFormatter.ofPattern("HH:mm:ss");
    @OnOpen
    public void showTime(Session session){
        allSessions = session.getOpenSessions();

        // start the scheduler on the very first connection
        // to call sendTimeToAll every second
        if (allSessions.size()==1){
            timer.scheduleAtFixedRate(
                () -> sendTimeToAll(session),0,1,TimeUnit.SECONDS);
        }
    }

    private void sendTimeToAll(Session session){
        allSessions = session.getOpenSessions();
        for (Session sess: allSessions){
            try{
                sess.getBasicRemote().sendText("Local time: " +
                    LocalTime.now().format(timeFormatter));
            } catch (IOException ioe) {
                System.out.println(ioe.getMessage());
            }
        }
    }
}
```

# Demo

Pushing data from a Java EE server to the Dart client using WebSocket protocol

The detailed description of this demo is here: <http://bit.ly/1DeulKX>

# Dart Tools

1. **IDEs:** Dart Editor, and plugins for all major IDEs
2. **dart2js** is a Dart-to-JavaScript transpiler and a tree-shaker
3. **pub** is a dependency management, development server and build tool.
4. **gulp** is a task manager. It's an analog of Grunt or Gradle.
5. **Dartium** is a Web Browser for developers.
6. **Dump-Info Visualizer** allows to inspect the generated JavaScript.
7. **Observatory** is Dart profiler.
8. **AngularDart** is a port of AngularJS framework.

# Links

- Code samples from this demo: <https://github.com/Farata/dart>
- Dart Style Guide <https://www.dartlang.org/articles/style-guide>
- Dart API docs: <https://api.dartlang.org>
- Try Dart: <https://dartpad.dartlang.org>
- Hands-on labs: [dartlang.org/codelabs](http://dartlang.org/codelabs)
- List of languages that compile to JS: <http://bit.ly/1FGHtPT>
- NYC Dart meetup: <http://www.meetup.com/Dart-Users-Group>



# More Links

- Our Dart/Java app: <https://easy.insure>
- Farata Systems: [faratasystems.com](http://faratasystems.com)
- Twitter: @yfain
- email: [yfain@faratasystems.com](mailto:yfain@faratasystems.com)
- Personal blog: [yakovfain.com](http://yakovfain.com)

