

# Dart For Java Developers

Part 2

# How to create Dart app

# Dart Scripts

# Dart Scripts

- Any Dart file can be launched with **dart** command
- File **must** contain **main** top-level function
- It can import SDK libraries

IDEA: 01\_dart\_script/01\_main.dart

# Dart Scripts with Imports

- *Normally* cannot contain package imports.
- Why? Package import paths resolved relatively to the current file's directory.
- Workaround - explicitly specify path to a package root directory.
- Package Spec should improve the way package imports are resolved.

# dart2js

- Not only for Dart-to-JavaScript compilation
- `--output-type` allows producing single Dart file
- Resulting Dart file contains all dependencies, tree-shaken and (optionally) minified.
- Can be run with just **dart**
- Still portable across other machines

# Snapshots

- Binary-serialized Dart program
- Format is close to the way Dart objects represented in memory
- Improve application startup time
- Can be created manually for command-line apps
- Browsers with Dart VM (not anticipated) create snapshots automatically
- Snapshot is executed by **dart**, hence still portable

IDEA: 01\_dart\_script/04\_snapshot.sh

# Dump Info

- dart2js allows generating dump info files
- Use --dump-info flag to generate dump into JSON file
- Dump info files can be used to analyze generated JavaScript
- Use Dump Info Visualizer to inspect generated JS code
- Allows to inspect size of individual code elements (methods, classes, etc.)
- Allows to diff two dump info files

IDEA: compare 05\_dump.info.json and 05\_dump\_with\_mirrors.info.json in visualiser



# Summary

- Standalone Dart scripts can be run with **dart**
- Dart program can be compiled into a single Dart file
- dart2js allows to optimize and inspect generated JS code
- All of the above is applicable to Dart Web apps

# Dart Packages

# What is Pub

- Main responsibilities:
  - Manages dependencies
  - Builds applications
  - Publishes packages to [pub.dartlang.org](https://pub.dartlang.org)
  - Manages *global* packages
  - Project management tool
- Pub is part of SDK
- Pub is written in Dart

# Dart Packages

- Dart package is a directory with `pubspec.yaml`
- `pubspec.yaml` contains metadata about the package
- Sample `pubspec.yaml`:

```
name: 02_dart_package
version: 0.0.1
author: Farata Systems LLC.
description: Demo package

dependencies:
  logging: any

environment:
  sdk: ">=1.9.0"
```

# Pub Dependencies

# Types of Dependencies

- There are 3 types of dependencies:
  - `dependencies` - regular dependencies
  - `dev_dependencies` - do not become transitive for other packages (e.g. `unittest` package)
  - `dependency_overrides` - force particular dependency versions, for development only
- Single dependency - key/value pair where key is the package *name* and value is the version constraint. Example:

```
dependencies:  
  logging: ">=0.9.3 <0.10.0"
```

# Version Constraints

- Allow to specify both boundaries or only one boundary:

`dependencies:`

`di: ">=3.3.4 <4.0.0"`

`logging: ">=0.9.0"`

`observe: "<0.13.0"`

- Caret syntax, similar to bower and npm:

`# These are two equivalent entries:`

`di: ">=3.3.4 <4.0.0"`

`di: "^3.3.4"`

- Special value `any` instructs to pick the latest stable version of the package that satisfies all immediate and transitive dependencies:

`unittest: any`

# Dependency Sources

- Hosted packages - hosted by [pub.dartlang.org](https://pub.dartlang.org) or custom pub server

```
angular: ">=1.1.0 <2.0.0"
```

- Git packages - any Git repository

```
angular:
```

```
  git: git@github.com:angular/angular.dart.git
```

```
angular:
```

```
  git:
```

```
    url: git@github.com:angular/angular.dart.git
```

```
    ref: dev-branch
```

```
    # ref can be a revision number, e.g.:
```

```
    # ref: "011d65f00d7a7d5f825b4337cf594baf20e6625f"
```

- Path packages - on a local file system

```
surancebay_api:
```

```
  path: "../surancebay_api"
```

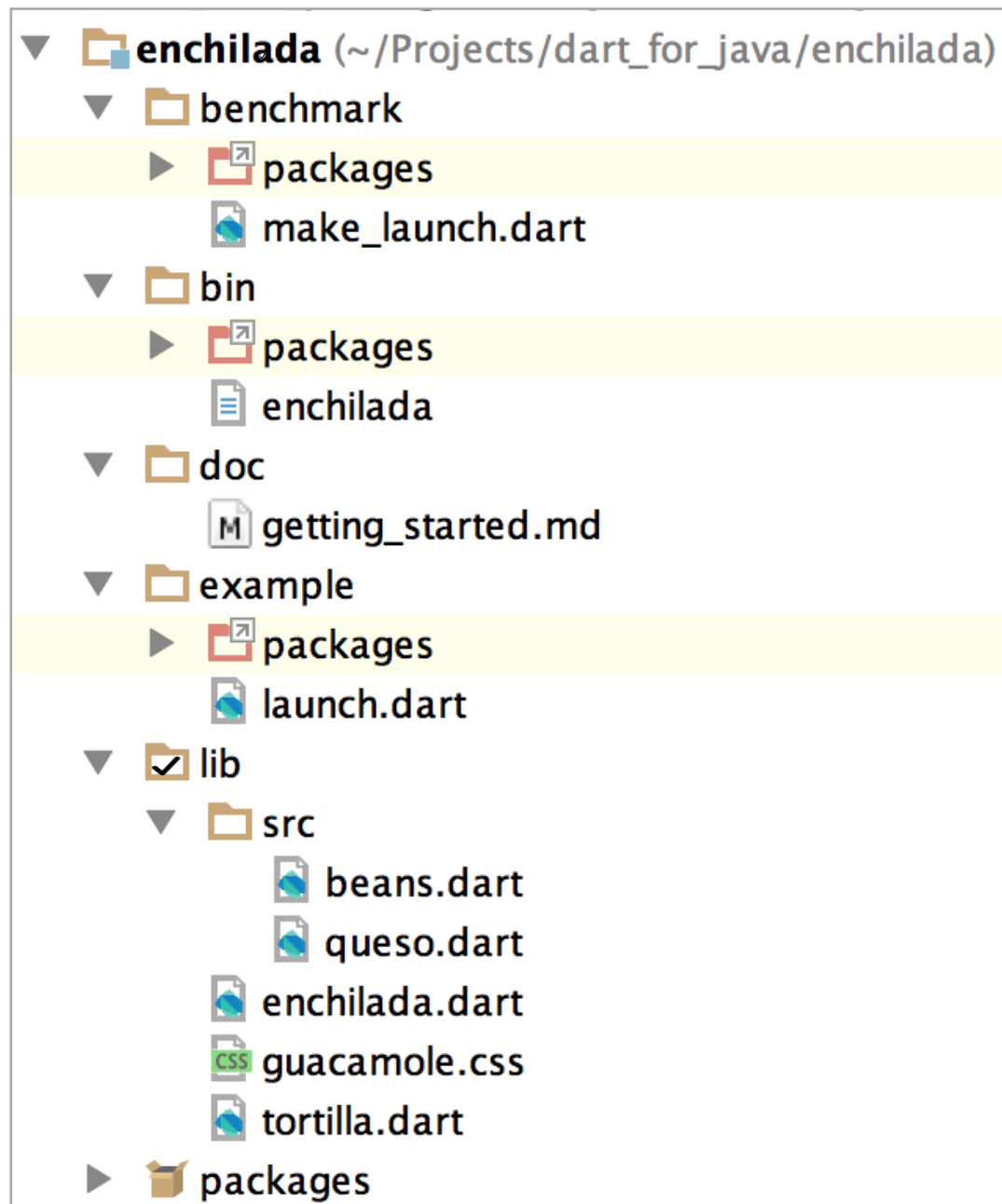


# Pub Commands

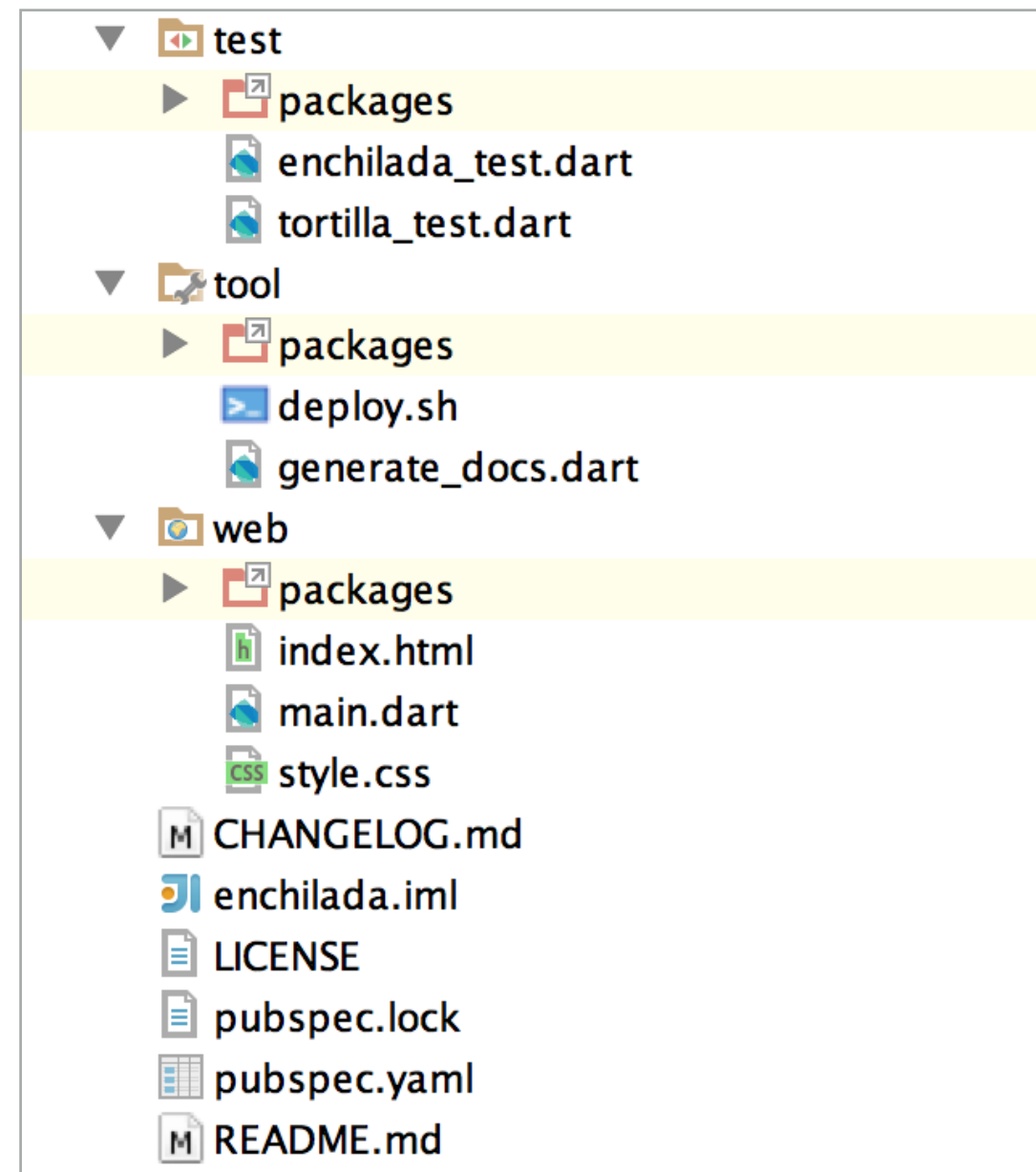
- `pub get` - gets dependencies listed either in `pubspec.lock` or `pubspec.yaml`
- `pub upgrade` - ignores `pubspec.lock`, attempts to get the most recent dependencies satisfying version constraints.
- `pub get` doesn't update versions of existing dependencies in lock file
- `pub upgrade` updates lock file
- Both commands create local packages/ dir containing symbolic links to the real packages store in `~/.pub-cache`

# Package Layout Conventions

First part:



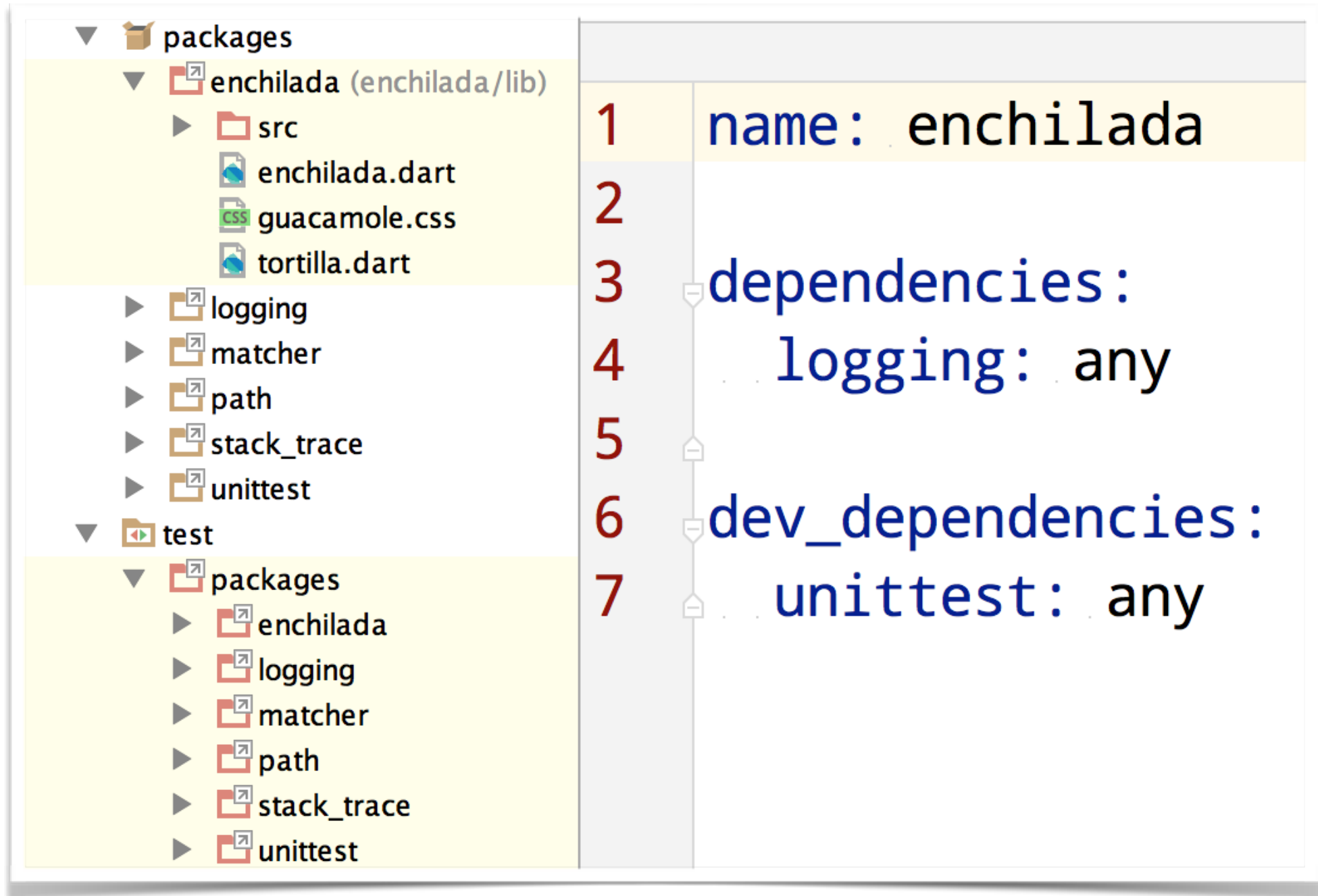
Second part:



Example project from: <https://www.dartlang.org/tools/pub/package-layout.html>

# Packages Directory

Only *entrypoint* directories (and their sub-directories) contain packages/



The screenshot displays a project structure in an IDE. The left pane shows a tree view of the project:

- ▼ packages
  - ▼ enchilada (enchilada/lib)
    - src
      - enchilada.dart
      - guacamole.css
      - tortilla.dart
    - logging
    - matcher
    - path
    - stack\_trace
    - unittest
  - test
    - packages
      - enchilada
      - logging
      - matcher
      - path
      - stack\_trace
      - unittest

The right pane shows the `pubspec.yaml` file for the `enchilada` package, with line numbers 1 through 7 on the left:

```
1 name: enchilada
2
3 dependencies:
4   logging: any
5
6 dev_dependencies:
7   unittest: any
```

# Types of Dart Packages

- Library packages:
  - Dart libraries placed in **lib/** dir are publicly exposed
  - Do not check lockfile in source control
- Application packages:
  - Usually checks in lockfile in source control
  - Command-line app entrypoint in **bin/**
  - Web app entrypoint in **web/**

# Revisiting Dart Language

# async/await

IDEA: StockQuoteAsyncAwait

# Deferred Libraries

# Mirrors