`deject`: Writing Testable Code in D

Ben Gertzfield <beng@fb.com>

If you take away one thing from this talk, make it this:

Writing tests can be easy, if you separate out your dependencies.

Why write testable code?

Making your code testable lets us make new and interesting mistakes

(instead of those same old boring mistakes)

Don't live in fear of touching code you didn't write

And don't live in fear of touching code you did write…

(6 months ago)

This talk covers (just) one technique for writing testable code: dependency injection.

Code speaks louder than words...

Let's take a look at some code that's hard to test.

```
import std.socket;
import std.stdio;

void checkHost(string host) {
  auto ih = new InternetHost;
  if (!ih.getHostByName(host)) {
    logError("No DNS: " ~ host);
    throw new Exception("Host down: " ~ host);
  }
}

void logError(string err) {
  auto f = File("log.txt", "a");
  f.write(err);
}

unittest {
  assertNotThrown(checkHost("dlang.org"));
  assertThrown(checkHost("qlang.org"));
}
```

When will these tests fail?

When the network is down

If someone registers `qlang.org`

When the filesystem is full…
or read-only

When I hit Control-C after waiting 30 seconds for the test to complete

```
% time rdmd --main -unittest counterexample.d
30.838 secs
```

Let's make this testable

```
import std.log; // Wishful thinking..
import std.socket;

class HostChecker {
  private InternetHost ih;
  private Logger logger;

  this(InternetHost ih, Logger logger) {
    this.ih = ih;
    this.logger = logger;
  }

  void checkHost(string host) {
    if (!ih.getHostByName(host)) {
      logger.error("No DNS: " ~ host);
      throw new Exception("Host down: " ~ host);
    }
  }
}
```

What changed here?

- Pass dependencies from above to constructor
- Clearly define seams to stitch together logic and dependencies
- Avoid static methods and globals

Let's Test This Puppy

```
unittest {
  import dmocks.Mocks;
  auto host = "a.b.c";
  auto mocker = new Mocker;
  auto ih = mocker.mock!InternetHost;
  auto logger = mocker.mock!Logger;

  mocker.expect(ih.getHostByName(host));
  mocker.replay();

  auto checker = new HostChecker(ih, logger);
  checker.checkHost(host);

  mocker.verify();
}
```

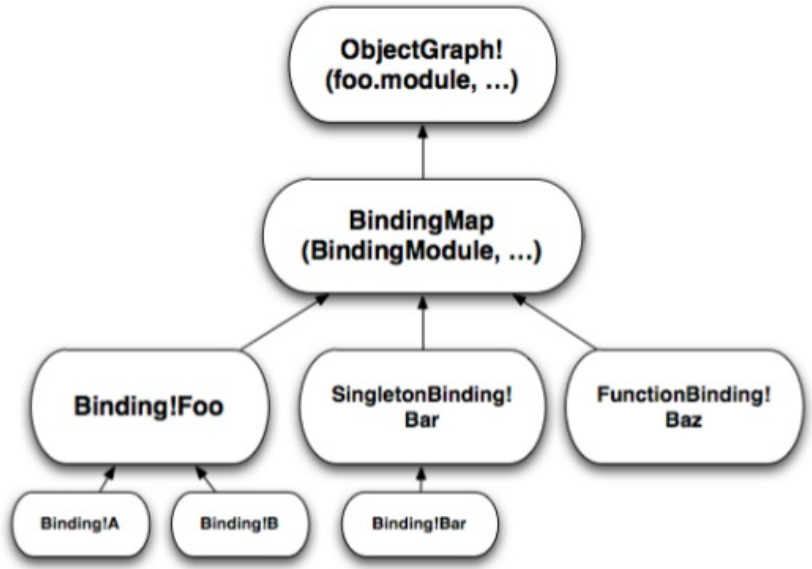
Testing failure is now an option

```
unittest {
  import dmocks.Mocks;
  auto host = "a.b.c";
  auto mocker = new Mocker;
  auto ih = mocker.mock!InternetHost;
  auto logger = mocker.mock!Logger;

  mocker
    .expect(ih.getHostByName(host))
    .returns(false);
  mocker
    .expect(logger.error(""))
    .repeatAny()
    .ignoreArgs;
  mocker.replay();

  auto checker = new HostChecker(ih, logger);
  assertThrown(checker.checkHost(host));

  mocker.verify();
}
```

The Good, The Bad, and the Dejected

- The good
  - Writing code this way makes it simple to test
- The bad
  - Tying every client to a (possibly changing) list of dependencies is heinously bad
- The dejected
  - D is a modern programming language: can't it take care of these details?
- Introducing `deject`, dependency injection for D

The library: `deject`



Runtime dependency injection

`BindingModule`s define map of (`TypeInfo` → `Binding`) pairs

`ObjectGraph` analyzes graph of dependencies and uses `Linker` to find and cache dependencies of `Binding`s

Manages object lifetime for singletons and other objects with scoped lifetime

Compile-time dependency injection

D's compile-time introspection lets us build the object graph at compile time

Looks for `@Inject` attribute to denote classes with dependencies managed by object graph

Emits D mixins to define `ObjectGraph.get!T` to construct injected type `T`

One small change to make that code injected

```
import deject.attributes;

@Inject
class HostChecker {
  private InternetHost ih;
  private Logger logger;

  this(InternetHost ih, Logger logger) {
    this.ih = ih;
    this.logger = logger;
  }

  // ...
}
```

Wrapping it up

Use pass from above to separate dependencies from your code

Use mocks to control behavior of dependencies in tests

Grab the source: **github.com/bgertzfield/deject**

Questions?