Noop

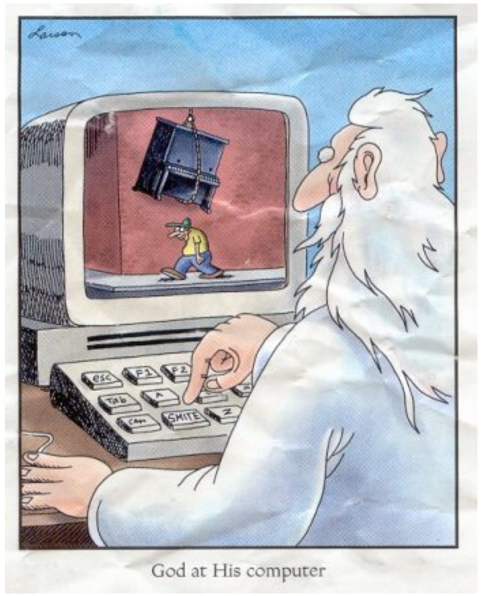
A language for teams of 2 or more

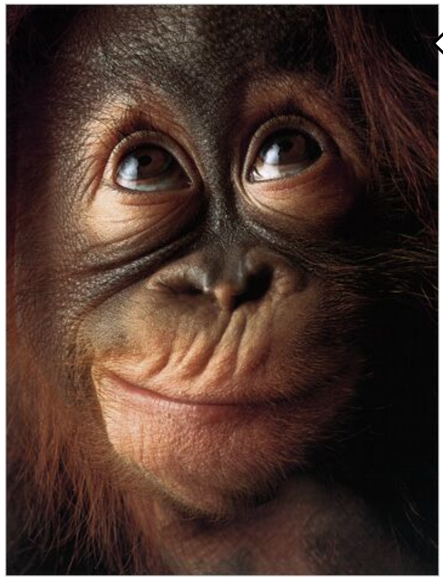
`http://noop.googlecode.com`

Alex Eagle
Jérémie Lenfant-Engelmann
Google

Questions? http://bit.ly/noop-jls

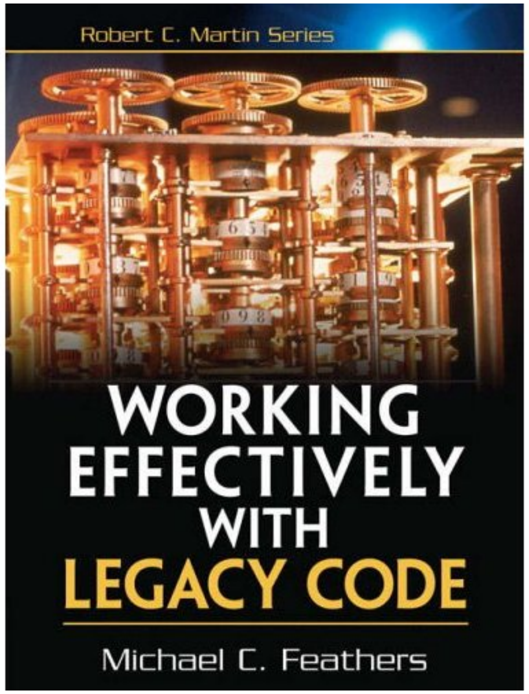
Why another language?





Now the developer stays involved

What is legacy code?
(Most code is legacy code)



What is non-legacy code?
(Most languages are focused on it)

Noop will improve the effectiveness of working on legacy code

Noop's mission

Help teams develop software that is easier to understand and maintain.

Meet the team:

.... Sloth

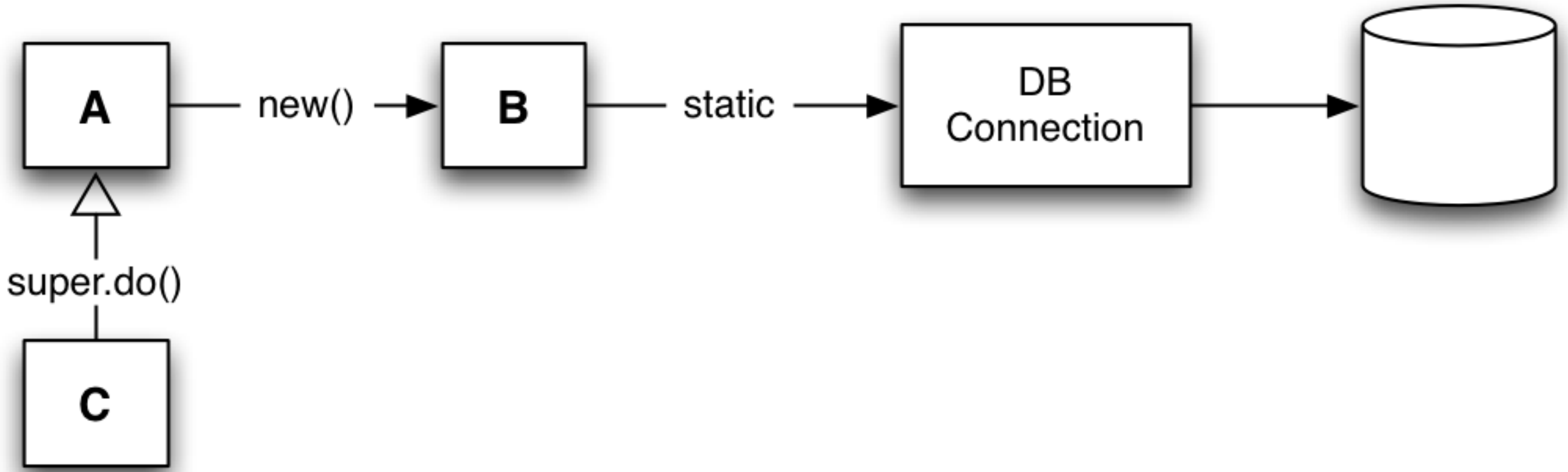
.... All of you

.... Us

Theorem 1

Better unit testing leads to better software

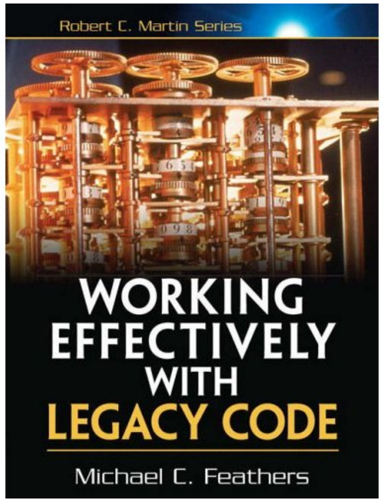
What is hard to test?



```
class A {
 void do(String s) {new B().getData(s);}
}
class B {
 public B() { DBConnFactory.init(); }
 void getData(String s) {...}
}
class C extends A {
 void getUser() {super.do("select usr");}
}
```

Seams

"A seam is a place where you can alter behaviour in your program without editing in that place"



Seams are critical for unit testing

Every place where we had no seam was due to a language feature!

Decision 1

In Noop, there will be a seam between every pair of classes

Dependency Injection in Noop

Instead of	We will
`new Foo();`	Request injection of an instance of Foo (or a subtype)
extends	Composition, via delegation with implied method forwarding
`static (method or field)`	Inject a dependency that is bound as a Singleton
`final (class or method)`	we don't have subclasses!

Dependency Injection Example

```
// [this is *proposed* syntax]

// BankServiceImpl.noop
class BankServiceImpl(DbConnection c)
 implements BankService { /* ... */ }

// Mine.noop
class Mine() implements Application {
 Int main(List<String> args) {
  scope(BankService -> BankServiceImpl,
        ConnectString -> "jdbc:mydb:") {
   BankService service = BankService();
  } // exits the scope, back to parent injector
 }
}
```

Named Scope

```
// MyScope.noop
scope MyScope {
 MyInterface -> MyImplementation;
}

// Someclass.noop
String helloWorld() scope MyScope {
 // some code
}
String goodbyeWorld() {
 scope MyScope {
  // some code
 }
}
```

Aliased types

In Guice:

```
@Named("port") Integer port;
@Inject class Server(@Named("port") Integer port);
```

We provide an alias keyword, associating another identifier with the same type:

```
alias Int Port;
```

Now you can inject the port number:

```
Port -> 9876;
// ...
class Server(Port port) {}
```

Composition Example

```
// Foo.noop
class Foo() {
 String doFoo() { return "foo"; }
 String doBar() { return "bar"; }
}
// Delegator.noop
class Delegator(delegate Foo foo) {
 override String doBar() { return "BAR"; }
}

// calling code can then do the following...
Foo foo = Foo();
Delegator d = Delegator(foo);
assertEquals("foo", d.doFoo());
assertEquals("BAR", d.doBar());
// and Delegator is assignable to a Foo variable
Foo otherFoo = d;
```

What do teams do?

Teams that write large-scale software systems communicate in code.

This means, they:

- Have to write their code
- Have to fix their code
- Have to extend their code
- Have to read their code
- Have to stay sane

Theorem 2

Code is read much more than it is written

Readability

Always Be Consistent - no optional syntax

- $_
- object.methodCall

Documentation to be checked as much as possible, including

- parameter names
- markup
- example code
- links

Non-noop coders should be able to review the code.

Testing DSL

Why should tests be methods in a class?

```
// in any file, including Interpreter.noop
test ("interpreter") {
  test ("it should parse hello world") {
    assertEquals(new Parser.parseAST("class Hello(){}"),
      "(CLASS Hello)");
  }
}
```

Leaves are tests, non-leaves are suites

Like-named suites are grouped together

Choices, choices

- No primitive types, everything is an Object
- `final` fields/variables by default, **mutable** keyword otherwise
- Strong static typing (possibly implied)
- StdLib: reflect best options out there, ie. JodaTime
- Exceptions: only unchecked
- Properties: a lot like C#
- No `new` keyword - just Python-style Type()
- Type appears to the left of the variable (like Java)
- One class defined per file, must match filename
- Namespace determined by relative path from source root
- Closures/lambdas/blocks: sounds great
- Parameterized types: sounds great

But what if I want to do X?

X: "code to 200 columns wide"

X: "name my methods silly things"

X: "create static methods"

What's next?

Writing the interpreter now. It runs "Hello World", you can download a snapshot from our continuous build.

Haven't thought much about the compiler yet, but we plan to emit Java bytecodes.

If you'd like to get involved, talk to us after, or join
`http://groups.google.com/group/noop-dev`

If you'd like to stay tuned to announcements, just join
http://groups.google.com/group/noop-announce

Topics for discussion

- Do you use Dependency Injection? Would you want language-level support?
- Is there a case where you feel you really want implementation inheritance?
- Partial injection
- Newable / Injectable
- Errors vs. Exceptions