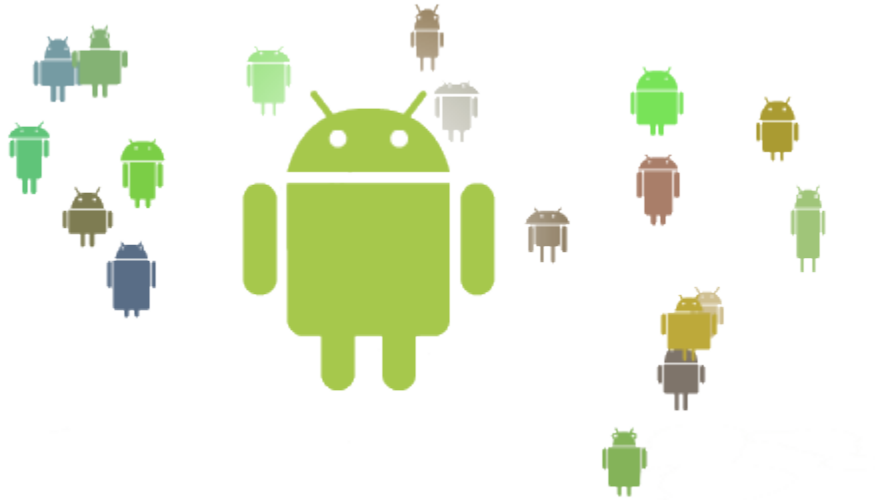
Deep dive into RoboGuice

beyond "Hello World apps"



about("Konrad Malawski")

lunar logic polska

Konrad Malawski
github: ktoso
twitter: ktosopl

[sckrk]

Does your Activity look like this?

```
 1 public class CracowMobiActivity extends Activity {
 2
 3   Twitter twitter;
 4
 5   EditText msg;
 6   ListView tweets;
 7   Button send;
 8   TextView hello;
 9
10   LayoutInflater inflater;
11
12   @Override
13   public void onCreate(Bundle savedInstanceState) {
14     super.onCreate(savedInstanceState);
15
16     twitter = new FastTwitter();
17
18     setContentView(R.layout.main);
19
20     msg = (EditText) findViewById(R.id.msg);
21     tweets = (ListView) findViewById(R.id.tweets);
22     send = (Button) findViewById(R.id.send);
23     hello = (TextView) findViewById(R.id.hello);
24
25     // magic string alert!
26     inflater = (LayoutInflater) getSystemService(LAYOUT_INFLATER_SERVICE);
27   }
28 }
```

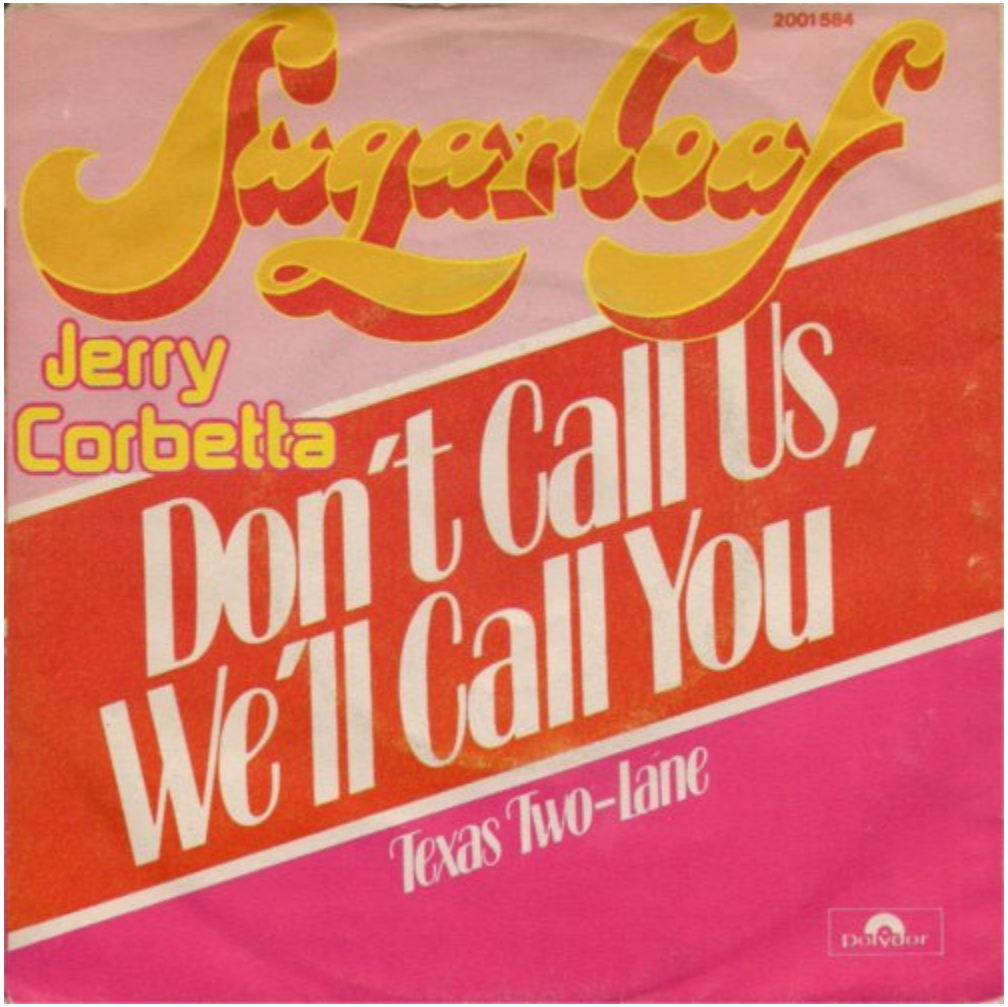
Me.sad(true)

IoC && DI

- Inversion of Control
- Dependency Injection

Inversion of Control

The hollywood principle



Dependency Injection in Java

JSR-330

Dependency Injection in Java

See: JSR-330 Specification

@Inject

A simple example

Bad:

```
1 class Clazz {
2   Twitter twitter;
3
4   public Clazz() {
5     twitter = new Twitter();
6   }
7 }
```

A simple example (field injection)

Bad:

```
1 class Clazz {
2   Twitter twitter;
3
4   public Clazz() {
5     twitter = new Twitter();
6   }
7 }
```

Better:

```
1 class Clazz {
2
3   @Inject
4   Twitter twitter;
5
6   public Clazz() {}
7 }
```

A simple example (constructor injection)

Better:

```
1 class Clazz {
2
3   @Inject
4   Twitter twitter;
5
6   public Clazz() {}
7 }
```

Better (cleaner) (but has boilerplate):

```
1 class Clazz {
2
3   Twitter twitter;
4
5   @Inject
6   public Clazz(Twitter twitter) {
7     this.twitter = twitter;
8   }
9 }
```

The "cleanest solution"

A simple example (setter injection)

Better:

```
1 class Clazz {
2
3   @Inject
4   Twitter twitter;
5
6   public Clazz() {}
7 }
```

Also ok (but has boilerplate):

```
 1 class Clazz {
 2
 3   Twitter twitter;
 4
 5   public Clazz() {    }
 6
 7   @Inject
 8   public setTwitter(Twitter twitter) {
 9     this.twitter = twitter;
10   }
11 }
```

Google Guice

Google Guice

- **Dependency Injection** Framework
- JSR-330 compatible (mostly)

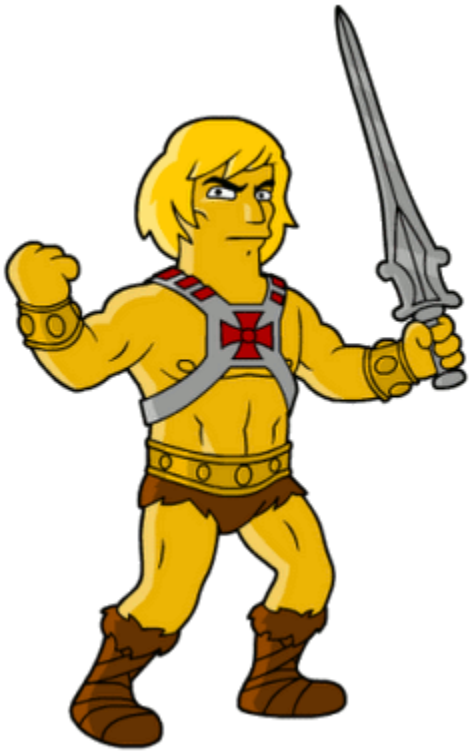
JSR-330 vs. Guice

JSR-330 javax.inject	**Guice** com.google.inject	
@Inject	@Inject	Interchangeable (almost).
@Named	@Named	Interchangeable.
@Qualifier	@BindingAnnotation	Interchangeable.
@Scope	@ScopeAnnotation	Interchangeable.
@Singleton	@Singleton	Interchangeable.
Provider	Provider	Guice's Provider extends JSR-330's Provider. Use Providers.guicify() to convert a JSR-330 provider into a Guice provider.

JSR-330 Annotations vs. Guice Annotations

Injector

He's the one who does all the heavy lifting of creating instances.



Injector Man

Injector

Here's what it does:

```
1 class InjectStuffIntoMe {
2   @Inject
3   Stuff stuff;
4
5   {
6     Injector injector = /*...*/.getInjector()
7     injector.injectMembers(someInstance);
8   }
9 }
```

Android + Guice = RoboGuice

RoboGuice - version disclaimer

! I'm using RoboGuice 2.0 beta 3 here !

It's fairly new - released in December 2011.

A migration guide for those still using 1.x.

Get Robo Guice

- Guice 3.0 no_aop
  - Why no aop?
- Robo Guice 2.x
- *(optional)* javax.inject

Configure RoboGuice (2.+)

res/values/roboguice.xml

```
1 <resources>
2   <string-array name="roboguice_modules">
3     <item>pl.project13.hello.guice.CracowMobiModule</item>
4   </string-array>
5 </resources>
```

Guice Module

```
 1 package pl.project13.hello.guice;
 2
 3 import com.google.inject.AbstractModule;
 4 import com.google.inject.name.Names;
 5 import pl.project13.hello.CracowMobiActivity;
 6 import pl.project13.hello.twitter.SlowTwitter;
 7 import pl.project13.hello.twitter.Twitter;
 8
 9 public class CracowMobiModule extends AbstractModule {
10
11   @Override
12   protected void configure() {
13     // wow, nothing?
14   }
15 }
```

Hello @Inject-ion World!

```
 1 class Twitter {
 2   // ...
 3 }
 4
 5 class MyActivity extends RoboActivity {
 6
 7   @Inject
 8   Twitter twitter;
 9
10 }
```

Why Module#configure matters

```
 1 interface Twitter { /**/ }
 2
 3 class SlowTwitter implements Twitter { /**/ }
 4 class FastTwitter implements Twitter { /**/ }
 5
 6 class MyActivity extends RoboActivity {
 7
 8   @Inject Twitter twitter;
 9
10 }
```

Whoops! Which Twitter?

Why Module#configure matters

In the **CracowMobiModule**:

```
1 public class CracowMobiModule extends AbstractModule {
2
3   @Override
4   protected void configure() {
5
6     bind(Twitter.class).to(FastTwitter.class);
7
8   }
9 }
```

Back in my Activity:

```
1 class MyActivity extends RoboActivity {
2
3   @Inject Twitter twitter;
4
5 }
```

Ok, that'll work :-)

Another way to do this...

```
1 @ImplementedBy(FastTwitter.class)
2 interface Twitter { }
```

No other configuration needed

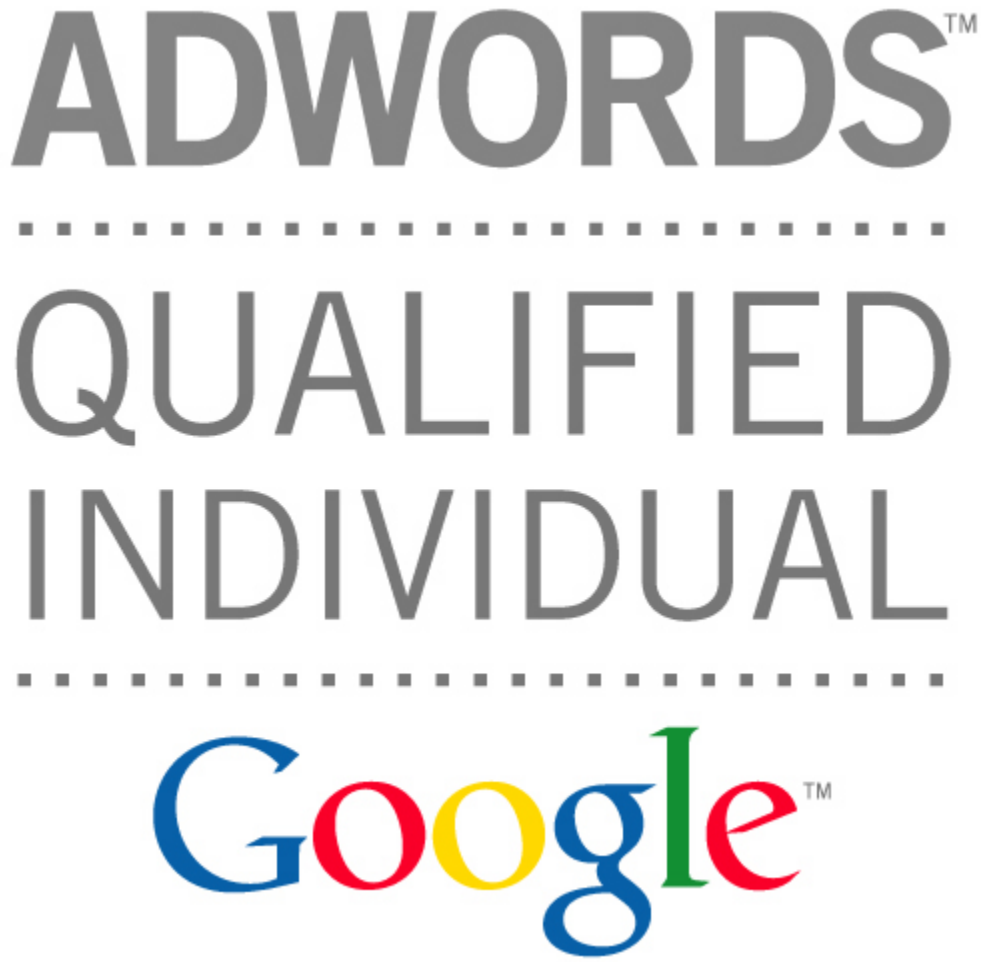
Another way to do this...

```
1 @ImplementedBy(FastTwitter.class)
2 interface Twitter { }
```

No other configuration needed

But it suck's to maintain such @ImplementedBy annotations.

@Qualifier-s



@Qualifier - @Named

In the **CracowMobiModule**:

```
 1 public class CracowMobiModule extends AbstractModule {
 2
 3   @Override
 4   protected void configure() {
 5
 6     bind(Twitter.class).to(FastTwitter.class);
 7
 8     bind(Twitter.class).annotatedWith(Names.named("slow")
 9                        .to(SlowTwitter.class);
10
11   }
12 }
```

Back in my Activity:

```
1 class MyActivity extends RoboActivity {
2
3   @Inject @Named("slow")
4   Twitter twitter;
5
6 }
```

Hmmm... but I don't like magic strings!

@Qualifier - roll your own!

Create an **@Interface**, using JSR-330:

```
1 @Documented
2 @Qualifier
3 @Retention(RetentionPolicy.RUNTIME)
4 @Target({ElementType.TYPE, ElementType.FIELD, ElementType.ANNOTATION_TYPE})
5 public @interface Slow {   }
```

The same in "plain Guice":

```
1 @Documented
2 @BindingAnnotation
3 @Retention(RetentionPolicy.RUNTIME)
4 @Target({ElementType.TYPE, ElementType.FIELD, ElementType.ANNOTATION_TYPE})
5 public @interface Slow {   }
```

Let's use it!

@Qualifier - Roll your own!

Let's change our Module:

```
1 bind(Twitter.class).annotatedWith(Slow.class)
2                    .to(SlowTwitter.class);
```

Back in my Activity:

```
1 class MyActivity extends RoboActivity {
2
3   @Inject @Slow Twitter client;
4
5 }
```

Whoa! That reads like a sentence!

Injection Scopes

This creates new instances:

```
1 class FastTwitter implements Twitter {}
2
3 bind(Twitter.class).to(FastTwitter.class);
```

Injection Scopes

This creates new instances:

```
1 class FastTwitter implements Twitter {}
2
3 bind(Twitter.class).to(FastTwitter.class);
```

And this does not:

```
1 bind(Twitter.class).to(FastTwitter.class)
2                     .asEagerSingleton();
```

Injection Scopes

This creates new instances:

```
1 class FastTwitter implements Twitter {}
2
3 bind(Twitter.class).to(FastTwitter.class);
```

And this does not:

```
1 bind(Twitter.class).to(FastTwitter.class)
2                     .asEagerSingleton();
```

This one too!

```
1 @Singleton
2 class FastTwitter implements Twitter {}
3
4 bind(Twitter.class).to(FastTwitter.class);
```

@Inject Constants too

In the Module:

```
1 bindConstant().annotatedWith(Author.class)
2                .to("ktoso");
```

In the app:

```
1 @Inject @Author
2 String author;
3
4 // ...
5
6 assert author == "ktoso";
```

toInstance()

You sometimes need to build the injected instance by hand...

```
1 bind(Twitter.class).toInstance(new FastTwitter());
```

But that mixes logic into the module :-(

Providers

In the module:

```
1 bind(Twitter.class).annotatedWith(Slow.class)
2                    .toProvider(TwitterProvider.class);
```

Provider<Twitter>

In the module:

```
1 bind(Twitter.class).annotatedWith(Slow.class)
2                    .toProvider(TwitterProvider.class);
```

Implement the **Provider**:

```
 1 public class TwitterProvider implements Provider<Twitter> {
 2
 3   @Inject @Author
 4   String username;
 5
 6   @Override
 7   public Twitter get() {
 8     return FastTwitter.forUser(username);
 9   }
10 }
```

Notice that it can have injected values too!

@InjectView

Now for something new in RoboGuice:

```
1 class MyActivity extends RoboActivity {
2
3   @InjectView(R.id.send)
4   Button send;
5
6 }
```

@InjectView

Now for something new in RoboGuice:

```
1 class MyActivity extends RoboActivity {
2
3   @InjectView(R.id.send)
4   Button send;
5
6 }
```

- No explicit casting
- No duplication

@InjectResource

Inject other things too:

```
1 @InjectResource(R.string.hello_message)
2 String helloMessage;
```

instead of:

```
1 String helloMessage;
2
3 protected void onCreate(Bundle savedInstance) {
4   helloMessage = getString(R.string.hello_message)
5 }
```

@Inject... everything!!!

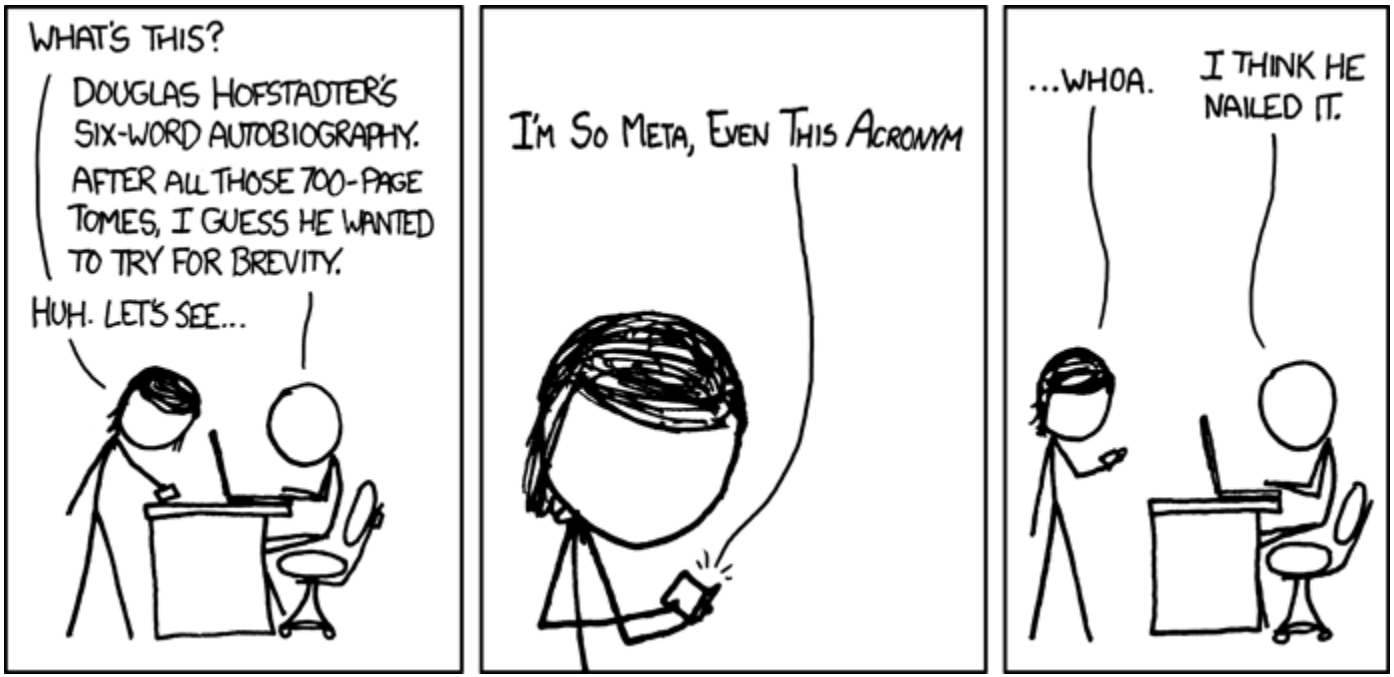
```
 1 public class CracowMobiActivity extends RoboActivity {
 2
 3   @Inject Twitter twitter;
 4
 5   @InjectView(R.id.msg) EditText msg;
 6   @InjectView(R.id.tweets) ListView tweets;
 7   @InjectView(R.id.send) Button send;
 8   @InjectView(R.id.hello) TextView hello;
 9
10   @Inject LayoutInflater inflater;
11
12   @Override
13   public void onCreate(Bundle savedInstanceState) {
14     super.onCreate(savedInstanceState);
15
16     setContentView(R.layout.main);
17   }
18 }
```

@Inject... EVERYTHING!!!

@ContentView()

```
 1 @ContentView(R.layout.main)
 2 public class CracowMobiActivity extends RoboActivity {
 3
 4   @Inject Twitter twitter;
 5
 6   @InjectView(R.id.msg) EditText msg;
 7   @InjectView(R.id.tweets) ListView tweets;
 8   @InjectView(R.id.send) Button send;
 9   @InjectView(R.id.hello) TextView hello;
10
11   @InjectResource(R.string.hello_message) String helloMessage;
12
13   @Inject LayoutInflater inflater;
14   @Inject ContactManager contacts;
15
16   @Override
17   public void onCreate(Bundle savedInstanceState) {
18     super.onCreate(savedInstanceState);
19   }
20 }
```

I'm sooo meta...


WHAT'S THIS?
DOUGLAS HOFSTADTER'S SIX-WORD AUTOBIOGRAPHY. AFTER ALL THOSE 700-PAGE TOMES, I GUESS HE WANTED TO TRY FOR BREVITY.
HUH. LET'S SEE...
I'M SO META, EVEN THIS ACRONYM
...WHOA.
I THINK HE NAILED IT.

QQ == Quick quiz...

Is this ok?

```
 1 class Anything {}
 2
 3 class Nothing {
 4   @Inject Something sth;
 5 }
 6
 7 class Something {
 8   @Inject Anything anything;
 9
10   public Something() {
11     It it = anything.get();
12   }
13 }
```

QQ == Quick quiz...

Is this ok?

```
 1 class Anything {}
 2
 3 class Something {
 4   @Inject Anything anything;
 5
 6   public Something() {
 7     It it = anything.get(); // null!!!
 8   }
 9 }
10
11 @Inject Something sth;
```

NullPointerException!

Use @Inject for "PostInitialized"

```
 1 class Anything {}
 2
 3 class Something {
 4   @Inject Anything anything;
 5
 6   public Something() {}
 7
 8   @Inject
 9   public void init() {
10     It it = anything.get();
11   }
12 }
13
14 @Inject Something sth;
```

This is ok.

Light! Camera! ~~Action!~~ Events!

It @Observes an Event

RoboGuice supplies us with an EventManager:

```
1 @Inject
2 EventManager rambo;
```

which can **fire** events:

```
1 class ShootingEvent{ /**/ }
2
3 rambo.fire(new ShootingEvent("Bam bam bam!"));
```

and someone may get hit by it:

```
1 public void onScrollEvent(@Observes ShootingEvent shot) {
2   if(this.wasHitBy(shot)) {
3     this.explode();
4   }
5 }
```

It's a very nice way to have **loosely coulped** listeners on events.

Unit Testing Android :-)

It's dangerous to go alone...

Let's test! Yaaaaay...

```
1 public class Test {
2   // ...
3 }
```

It's dangerous to go alone...

Let's test! Yaaaaay...

```
1 public class Test {
2   // ...
3 }
```

Woooot?!

```
1 java.lang.RuntimeException: Unable to instantiate activity ComponentInfo{com.someapp.myapp/c
2                     java.lang.RuntimeException: stub!
3     at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:1569)
4     at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:1663)
```

java.lang.RuntimeException: stub!

It's dangerous to go alone...

Let's test! Yaaaaay...

```
1 public class Test {
2   // ...
3 }
```

Woooot?!

```
1 java.lang.RuntimeException: Unable to instantiate activity ComponentInfo{com.someapp.myapp/c
2                     java.lang.RuntimeException: stub!
3     at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:1569)
4     at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:1663)
```

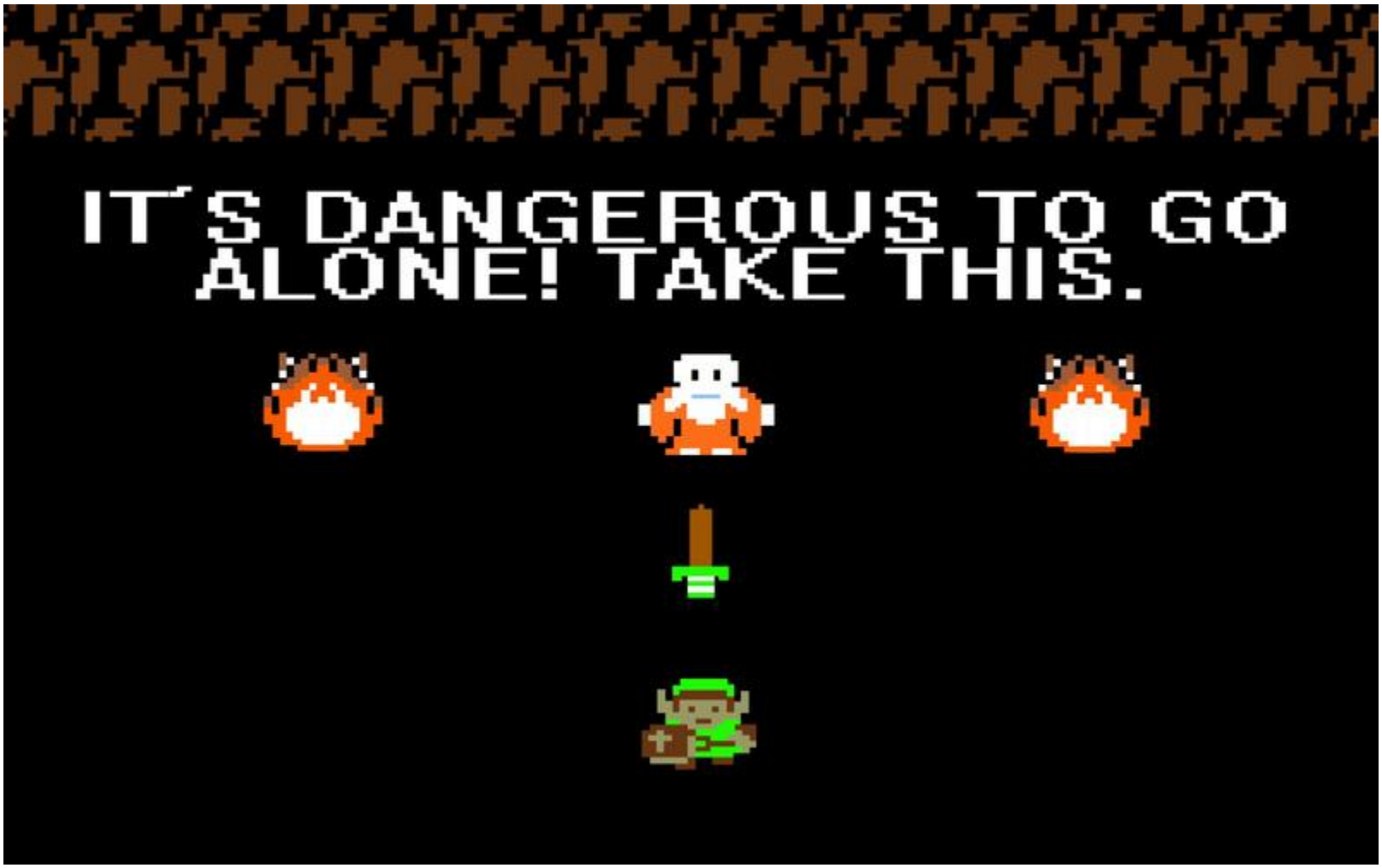
java.lang.RuntimeException: stub!

Android API:

```
1 public void doSomething() {
2   throw new RuntimeException("Stub!");
3 }
```

No joke.

… Take this!



Robolectric


```
1 public void doSomething() {
2   throw new RuntimeException("Stub!")
3 }
4                 |
5                 V
6 public void doSomething() {
7 }
```

Robolectric


```
1 public void doSomething() {
2   throw new RuntimeException("Stub!")
3 }
4                 |
5                 V
6 public void doSomething() {
7 }
```

or:

```
1 public String doSomething() {
2   throw new RuntimeException("Stub!")
3 }
4                 |
5                 V
6 public String doSomething() {
7   return null;
8 }
```

Visit their homepage!

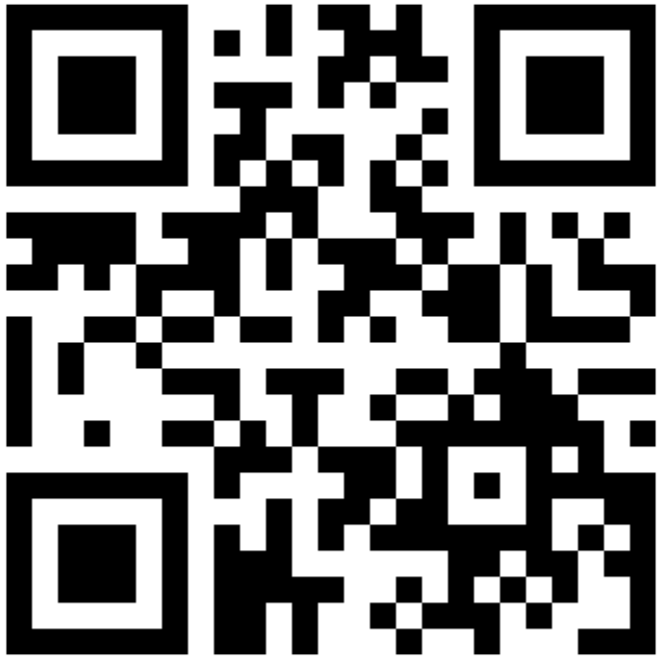
How do both help in testing?

Using my custom JUnit Runner, you can:

```
 1 @RunWith(GuiceRobolectricTestRunner.class)
 2 @GuiceModules({CracowTestModule.class})
 3 public class MyActivityTest {
 4
 5   @Inject
 6   MyActivity activity;
 7
 8   @Test
 9   public void shouldHaveBeenInjected() {
10     assertThat(activity).isNotNull(); // our own rolled FEST
11   }
12
13   @Test
14   public void shouldShowOffRobolectric() {
15
16     // button
17     activity.button.performClick();
18     assertThat(activity.button.getText()).hasText("awesome");
19
20     // shadows
21     ShadowImageView shadowPivotalLogo = Robolectric.shadowOf(pivotalLogo);
22     assertThat(shadowPivotalLogo.resourceId, equalTo(R.drawable.pivotallabs_logo)); // hamc
23   }
24 }
```

Thanks! Dziękuję! ありがとう~!

Slides && code will be blogged:



blog.project13.pl