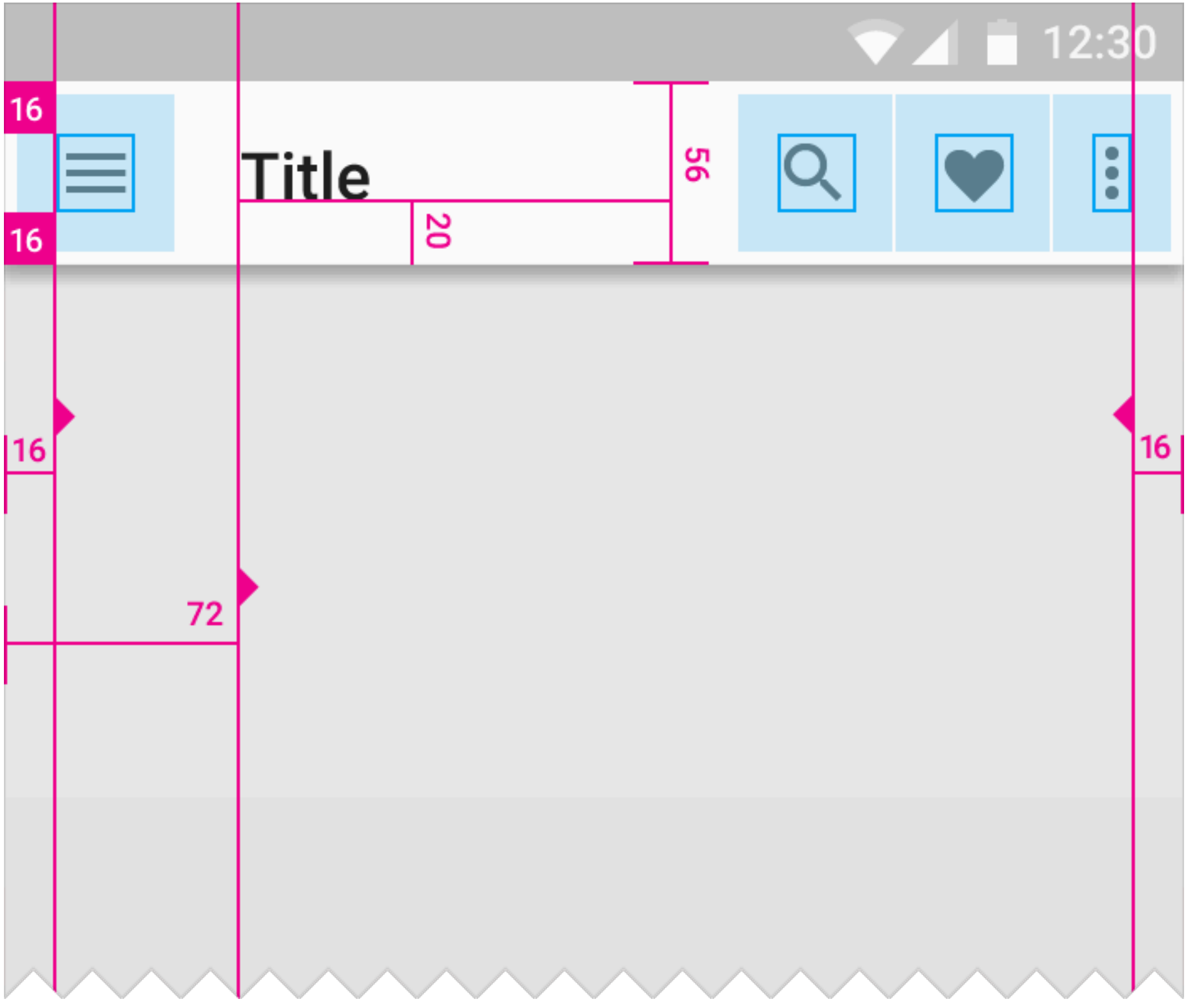
Layouts DSL is the only way

```
textView("") {
    textSize = 45f
    font = Fonts.MENLO
}.lparams {
    gravity = Gravity.CENTER
}
```

```
textView("Nov 14, 2017")
```


12:30
16
Title
56
20
16
16
16
72

XML is ...

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:title="Hello, I am a Toolbar"
        />

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

XML is verbose

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:title="Hello, I am a Toolbar"
        />

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

nonsense

Where’s the views, XML

```
View view = layoutInflater.inflate(R.layout.activity_main, null);
```

Where’s the views, XML

```
View view = layoutInflater.inflate(R.layout.activity_main, null);
```

```
<TextView
    android:id="@+id/first_name"/>

<TextView
    android:id="@+id/middle_name"/>

<TextView
    android:id="@+id/last_name"/>

<TextView
    android:id="@+id/email"/>

<TextView
    android:id="@+id/city"/>
```

too much sense

And then …

```
public class MainActivity extends Activity {
    TextView firstName;
    TextView middleName;
    TextView lastName;
    TextView email;
    TextView city;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        firstName = (TextView) findViewById(R.id.first_name);
        middleName = (TextView) findViewById(R.id.middle_name);
        lastName = (TextView) findViewById(R.id.last_name);
        email = (TextView) findViewById(R.id.email);
        city = (TextView) findViewById(R.id.city);
    }

    public void userUpdated(User user) {
        firstName.setText(user.firstName);
        middleName.setText(user.middleName);
        lastName.setText(user.lastName);
        email.setText(user.email);
        city.setText(user.city);
    }
}
```

And then findViewById

```
public class MainActivity extends Activity {
    TextView firstName;
    TextView middleName;
    TextView lastName;
    TextView email;
    TextView city;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        firstName = (TextView) findViewById(R.id.first_name);
        middleName = (TextView) findViewById(R.id.middle_name);
        lastName = (TextView) findViewById(R.id.last_name);
        email = (TextView) findViewById(R.id.email);
        city = (TextView) findViewById(R.id.city);
    }

    public void userUpdated(User user) {
        firstName.setText(user.firstName);
        middleName.setText(user.middleName);
        lastName.setText(user.lastName);
        email.setText(user.email);
        city.setText(user.city);
    }
}
```

boilerplate

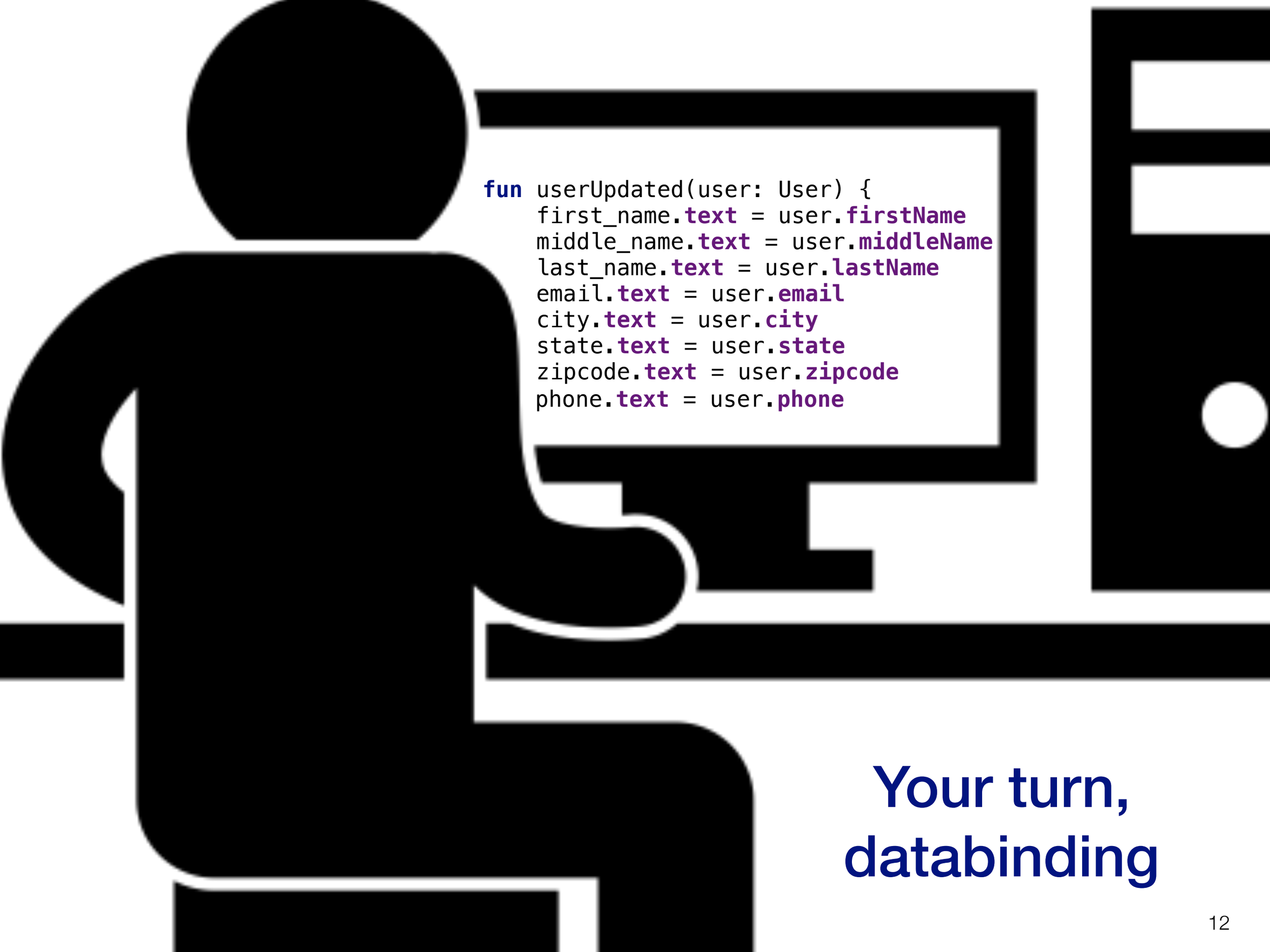
ButterKnife - bit better

```
public class MainActivity extends Activity {
    @BindView(R.id.first_name) TextView firstName;
    @BindView(R.id.middle_name) TextView middleName;
    @BindView(R.id.last_name) TextView lastName;
    @BindView(R.id.email) TextView email;
    @BindView(R.id.city) TextView city;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.bind(this);
    }

    public void userUpdated(User user) {
        firstName.setText(user.firstName);
        middleName.setText(user.middleName);
        lastName.setText(user.lastName);
        email.setText(user.email);
        city.setText(user.city);
    }
}
```

Kotlin and Android Extensions - like a magic

```
apply plugin: ‘kotlin-android-extensions’
```

```
import kotlinx.android.synthetic.activity_main.*

class MainActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    fun userUpdated(user: User) {
        first_name.text = user.firstName
        middle_name.text = user.middleName
        last_name.text = user.lastName
        email.text = user.email
        city.text = user.city
    }
}
```



Your turn, databinding

android.databindinG

```kotlin
class MainActivity : Activity() {
    lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = DataBindingUtil.setContentView(this, R.layout.activity_main)
    }

    fun userUpdated(user: User) {
        binding.user = user
    }
}
```

```xml
<layout>
    <data>
        <variable
            name=“user" type="com.example.User"/>
    </data>
</layout>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{user.firstName}" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@{user.lastName}" />
```

Wat

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <import type="com.example.MyStringUtils"/>
        <variable name="user" type="User" />
        <variable name="presenter" type="Presenter" />
    </data>

    <Button
        android:onClick="@{() -> presenter.userClicked(user)}"
        android:text="@{MyStringUtils.capitalize(user.lastName)}"
        android:visibility="@{user.old ? View.VISIBLE : View.GONE}" />
</layout>
```

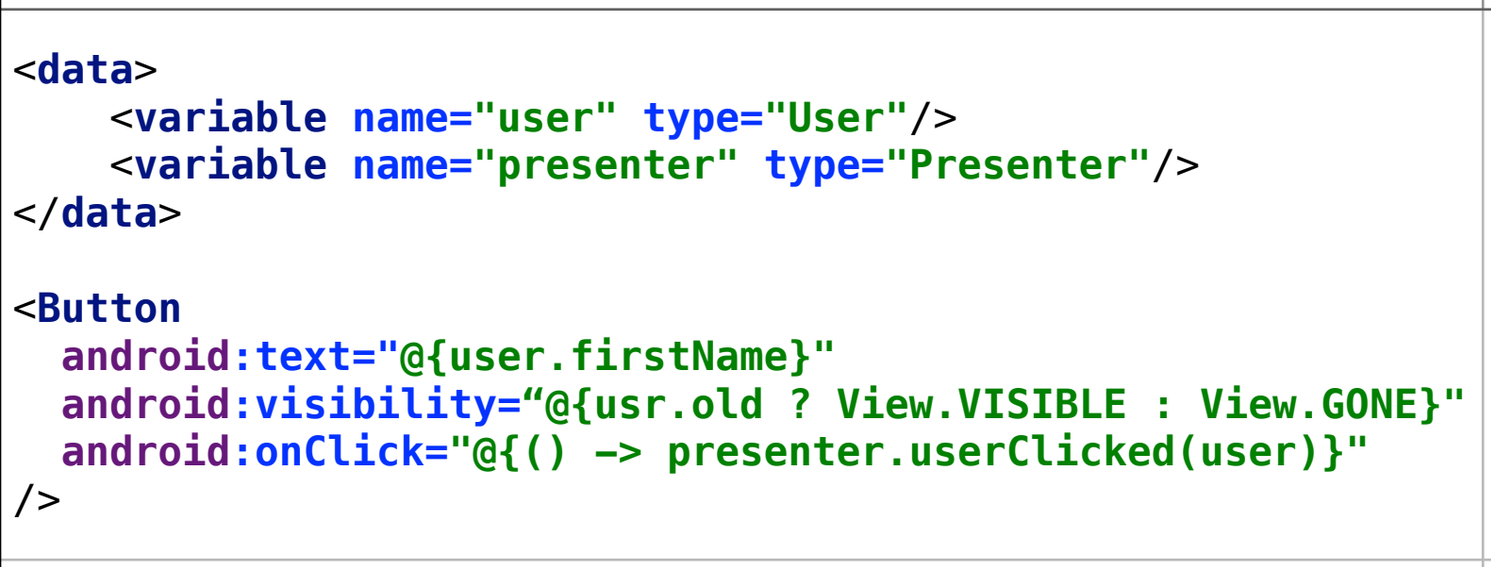

Anko

Anko Layouts DSL

```
compile "org.jetbrains.anko:anko-sdk25:0.10.2"
compile "org.jetbrains.anko:anko-support-v4:0.10.2"
compile "org.jetbrains.anko:anko-appcompat-v7:0.10.2"
compile "org.jetbrains.anko:anko-recyclerview-v7:0.10.2"
compile "org.jetbrains.anko:anko-cardview-v7:0.10.2"
compile "org.jetbrains.anko:anko-percent:0.10.2"
compile "org.jetbrains.anko:anko-coroutines:0.10.2"
                          . . .
fun Context.scrollView(init: ScrollView.() -> Unit)
fun ViewManager.button(init: Button.() -> Unit)
```

```
button {
  button.text = "A Button"
  button.textSize = 14f
  button.textColor = Color.BLACK
  button.setOnClickListener {  }
}

scrollView {
    verticalLinearLayout {
        relativeLayout {
            button {
                text = "A Button"
            }
        }.lparams(matchParent, height = dip(300))
    }.lparams(matchParent, wrapContent)
}.lparams(width = matchParent)
```

```
frameLayout {  }
linearLayout {  }
relativeLayout {  }
tableLayout {  }
toolbar {  }
gridLayout {  }
absoluteLayout {  }
scrollView {  }
      . . .
button {  }
textView {  }
imageView {  }
radioButton {  }
toggleButton {  }
zoomButton {  }
imageButton {  }
listView {  }
gridView {  }
searchView {  }
surfaceView {  }
editText {  }
progressBar {  }
ratingBar {  }
checkBox {  }
seekBar {  }
spinner {  }
switch {  }
      . . .
```

android.databinding	Anko Layouts DSL
	

```
<data>
    <variable name="user" type="User"/>
    <variable name="presenter" type="Presenter"/>
</data>

<Button
  android:text="@{user.firstName}"
  android:visibility="@{usr.old ? View.VISIBLE : View.GONE}"
  android:onClick="@{() -> presenter.userClicked(user)}"
/>
```

android.databinding

```
<data>
    <variable name="user" type="User"/>
    <variable name="presenter" type="Presenter"/>
</data>

<Button
  android:text="@{user.firstName}"
  android:visibility="@{usr.old ? View.VISIBLE : View.GONE}"
  android:onClick="@{() -> presenter.userClicked(user)}"
/>
```

Anko Layouts DSL

```
val user: User
val presenter: Presenter

button {
    text = user.firstName
    visibility =
        if (user.old)
            View.VISIBLE
        else View.GONE
    onClick { presenter.userClicked(user) }
}
```

android.databinding

```
<data>
    <variable name="user" type="User"/>
    <variable name="presenter" type="Presenter"/>
</data>

<Button
  android:text="@{user.firstName}"
  android:visibility="@{usr.old ? View.VISIBLE : View.GONE}"
  android:onClick="@{() -> presenter.userClicked(user)}"
/>
```

```
<data>
    <variable name="user" type="User" />
    <import type="com.example.MyStringUtils" />
</data>

<Button
   android:text="@{user.displayName ?? user.lastName}"
   android:hint="@{MyStringUtils.capitalize(user.lastName)}"
   android:transitionName='@{"image_" + user.id}'
/>
```

Anko Layouts DSL

```
val user: User
val presenter: Presenter

button {
    text = user.firstName
    visibility =
        if (user.old)
            View.VISIBLE
        else View.GONE
    onClick { presenter.userClicked(user) }
}
```

android.databinding

```
<data>
    <variable name="user" type="User"/>
    <variable name="presenter" type="Presenter"/>
</data>

<Button
  android:text="@{user.firstName}"
  android:visibility="@{usr.old ? View.VISIBLE : View.GONE}"
  android:onClick="@{() -> presenter.userClicked(user)}"
/>
```

```
<data>
    <variable name="user" type="User" />
    <import type="com.example.MyStringUtils" />
</data>

<Button
   android:text="@{user.displayName ?? user.lastName}"
   android:hint="@{MyStringUtils.capitalize(user.lastName)}"
   android:transitionName='@{"image_" + user.id}'
/>
```

Anko Layouts DSL

```
val user: User
val presenter: Presenter

button {
    text = user.firstName
    visibility =
        if (user.old)
            View.VISIBLE
        else View.GONE
    onClick { presenter.userClicked(user) }
}
```

```
import com.example.MyStringUtils.capitalize

val user: User

button {
   text = user.displayName ?: user.firstName
   hint = capitalize(user.lastName)
   transitionName = "image_${user.id}"
}
```

android.databinding	Anko Layouts DSL

`<data>`
`  <variable name="user" type="User"/>`
`  <variable name="presenter" type="Presenter"/>`
`</data>`
 	
`<Button`
`  android:text="@{user.firstName}"`
`  android:visibility="@{usr.old ? View.VISIBLE : View.GONE}"`
`  android:onClick="@{() -> presenter.userClicked(user)}"`
`/>`
	`val user: User`
	`val presenter: Presenter`
	
	`button {`
	`  text = user.firstName`
	`  visibility =`
	`  if (user.old)`
	`  View.VISIBLE`
	`  else View.GONE`
	`  onClick { presenter.userClicked(user) }`
	`}`

`<data>`
`  <variable name="user" type="User" />`
`  <import type="com.example.MyStringUtils" />`
`</data>`
 	
`<Button`
`  android:text="@{user.displayName ?? user.lastName}"`
`  android:hint="@{MyStringUtils.capitalize(user.lastName)}"`
`  android:transitionName='@{"image_" + user.id}'`
`/>`
	`import com.example.MyStringUtils.capitalize`
	
	`val user: User`
	
	`button {`
	`  text = user.displayName ?: user.firstName`
	`  hint = capitalize(user.lastName)`
	`  transitionName = "image_${user.id}"`
	`}`

`@BindingAdapter({"bind:loadImage"})`
`public static void loadImage(ImageView view, String url) {`
`  Picasso.with(view.getContext()).load(url).into(view);`
`}`
 	
`<ImageView`
`  bind:loadImage="@{user.imageUrl}"/>`	

android.databinding

```
<data>
    <variable name="user" type="User"/>
    <variable name="presenter" type="Presenter"/>
</data>

<Button
  android:text="@{user.firstName}"
  android:visibility="@{usr.old ? View.VISIBLE : View.GONE}"
  android:onClick="@{() -> presenter.userClicked(user)}"
/>
```

```
<data>
    <variable name="user" type="User" />
    <import type="com.example.MyStringUtils" />
</data>

<Button
   android:text="@{user.displayName ?? user.lastName}"
   android:hint="@{MyStringUtils.capitalize(user.lastName)}"
   android:transitionName='@{"image_" + user.id}'
/>
```

```
@BindingAdapter({"bind:loadImage"})
public static void loadImage(ImageView view, String url) {
    Picasso.with(view.getContext()).load(url).into(view);
}

<ImageView
    bind:loadImage="@{user.imageUrl}"/>
```

Anko Layouts DSL

```
val user: User
val presenter: Presenter

button {
    text = user.firstName
    visibility =
        if (user.old)
            View.VISIBLE
        else View.GONE
    onClick { presenter.userClicked(user) }
}
```

```
import com.example.MyStringUtils.capitalize

val user: User

button {
   text = user.displayName ?: user.firstName
   hint = capitalize(user.lastName)
   transitionName = "image_${user.id}"
}
```

```
fun ImageView.loadImage(url: String) {
  Picasso.with(context).load(url).into(this)
}

imageView {
    loadImage(user.imageUrl)
}
```

android.databinding

```
<data>
    <variable name="user" type="User"/>
    <variable name="presenter" type="Presenter"/>
</data>

<Button
  android:text="@{user.firstName}"
  android:visibility="@{usr.old ? View.VISIBLE : View.GONE}"
  android:onClick="@{() -> presenter.userClicked(user)}"
/>
```

```
<data>
    <variable name="user" type="User" />
    <import type="com.example.MyStringUtils" />
</data>

<Button
   android:text="@{user.displayName ?? user.lastName}"
   android:hint="@{MyStringUtils.capitalize(user.lastName)}"
   android:transitionName='@{"image_" + user.id}'
/>
```

```
@BindingAdapter({"bind:loadImage"})
public static void loadImage(ImageView view, String url) {
    Picasso.with(view.getContext()).load(url).into(view);
}

<ImageView
    bind:loadImage="@{user.imageUrl}"/>
```

Anko Layouts DSL

```
val user: User
val presenter: Presenter

button {
    text = user.firstName
    visibility =
        if (user.old)
            View.VISIBLE
        else View.GONE
    onClick { presenter.userClicked(user) }
}
```

```
import com.example.MyStringUtils.capitalize

val user: User

button {
   text = user.displayName ?: user.firstName
   hint = capitalize(user.lastName)
   transitionName = "image_${user.id}"
}
```

```
fun ImageView.loadImage(url: String) {
  Picasso.with(context).load(url).into(this)
}

imageView {
    loadImage(user.imageUrl)
}
```

Ok, Anko Layouts, how to use

```
val user: User
val presenter: Presenter

button {
    text = user.firstName
    visibility =
        if (user.old)
            View.VISIBLE
        else View.GONE
    onClick { presenter.userClicked(user) }
}
```

All you need is ...

```
View(Context context)
```

Simple constructor to use when creating a view from code.

```
View(Context context, AttributeSet attrs)
```

Constructor that is called when inflating a view from XML.

```
View(Context context, AttributeSet attrs, int defStyleAttr)
```

Perform inflation from XML and apply a class-specific base style from a theme attribute.

```
View(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes)
```

Perform inflation from XML and apply a class-specific base style from a theme attribute or style resource.

All you need is context

```
View(Context context)
```

Simple constructor to use when creating a view from code.

```
View(Context context, AttributeSet attrs)
```

Constructor that is called when inflating a view from XML.

```
View(Context context, AttributeSet attrs, int defStyleAttr)
```

Perform inflation from XML and apply a class-specific base style from a theme attribute.

```
View(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes)
```

Perform inflation from XML and apply a class-specific base style from a theme attribute or style resource.

View Factory

```
typealias ViewFactory = (Context) -> View

val buttonFactory: ViewFactory = ::Button
val frameLayoutFactory: ViewFactory = ::FrameLayout

val userProfileFactory: ViewFactory = { context: Context ->
    with(context) {
        verticalLinearLayout {
            gravity = Gravity.CENTER

            textView("Foo") {
                textSize = 20f
                textColor = Color.BLUE
            }

            textView("Bar") {
                textSize = 16f
                textColor = Color.RED
            }.lparams {
                topMargin = dip(10)
            }
        }

    }
}
```

View Factory

```
typealias ViewFactory = (Context) -> View

fun viewFactory(factory: Context.() -> View): ViewFactory
        = { context -> context.factory() }

val userProfile = viewFactory {
    verticalLinearLayout {
        gravity = Gravity.CENTER

        textView("Foo") {
            textSize = 20f
            textColor = Color.BLUE
        }

        textView("Bar") {
            textSize = 16f
            textColor = Color.RED
        }.lparams {
            topMargin = dip(10)
        }
    }
}
```

How to use

```
class MainActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(user_profile)
    }
}

fun Activity.setContentView(factory: ViewFactory) = setContentView(factory(this))

val user_profile = viewFactory {
    verticalLinearLayout {
        gravity = Gravity.CENTER

        textView("Foo") {
            textSize = 20f
            textColor = Color.BLUE
        }

        textView("Bar") {
            textSize = 16f
            textColor = Color.RED
        }.lparams {
            topMargin = dip(10)
        }
    }
}
```

Dex count matters

Usage	Method count	Field count
`val user_profile = viewFactory { }`	4	1
`val user_profile inline get() = viewFactory { }`	4	0
`val user_profile = fun(user: User) = viewFactory { }`	8	2
`fun userProfile() = viewFactory { }`	4	0
`fun userProfile(user: User = User()) = viewFactory { }`	5	0

Parameterization

```
val user_profile = fun(user: User) = viewFactory {
    verticalLinearLayout {
        gravity = Gravity.CENTER

        textView(user.firstName) {
            textSize = 20f
            textColor = Color.BLUE
        }

        textView(user.lastName) {
            textSize = 16f
            textColor = Color.RED
        }.lparams {
            topMargin = dip(10)
        }
    }
}

setContentView(user_profile(User("Foo", "Bar")))
```

Data binding on the spot

```
val user_profile = fun(user: rx.Observable<User>) = viewFactory {
    verticalLinearLayout {
        gravity = Gravity.CENTER

        textView {
            user.map(User::firstName).subscribe(::setText)
            textSize = 20f
            textColor = Color.BLUE
        }

        textView {
            user.map(User::lastName).subscribe(::setText)
            textSize = 16f
            textColor = Color.RED
        }.lparams {
            topMargin = dip(10)
        }
    }
}
```

Data binding even simpler

```
val user_profile = fun(user: rx.Observable<User>) = viewFactory {
    verticalLinearLayout {
        gravity = Gravity.CENTER

        textView(user.map(User::firstName)) {
            textSize = 20f
            textColor = Color.BLUE
        }

        textView(user.map(User::lastName)) {
            textSize = 16f
            textColor = Color.RED
        }.lparams {
            topMargin = dip(10)
        }
    }
}

inline fun ViewManager.textView(obsText: Observable<CharSequence>,
                                init: TextView.() -> Unit)
      = textView {
            init()
            obsText.subscribe(::setText)
        }
```

Data binding two way

```
import com.jakewharton.rxbinding.*

val user_profile = fun(user: rx.Observable<User>,
                       saveUser: (User) -> Unit) = viewFactory {
    verticalLinearLayout {
        gravity = Gravity.CENTER

        editText(user.map(User::firstName)) {
            textSize = 20f
            textColor = Color.BLUE
            textChanges().withLatestFrom(user)
                    .map { (name, user) -> user.copy(firstName = name) }
                    .subscribe(saveUser)
        }

        editText(user.map(User::lastName)) {
            textSize = 16f
            textColor = Color.RED
            textChanges().withLatestFrom(user)
                    .map { (name, user) -> user.copy(lastName = name) }
                    .subscribe(saveUser)

        }.lparams {
            topMargin = dip(10)
        }
    }
}
```

Parameterize them all

```
val screen_main = fun(user: Observable<User>,
                      hisFriends: Observable<List<Friend>>,
                      hisPhotos: Observable<List<Photo>>,
                      hisComments: Observable<List<Comment>>,
                      girlHiLikes: Observable<Girl>,
                      answerToTheQuestionOfLife: Int) = viewFactory { }
```

Parameterize them all

```
val screen_main = fun(user: Observable<User>,
                      hisFriends: Observable<List<Friend>>,
                      hisPhotos: Observable<List<Photo>>,
                      hisComments: Observable<List<Comment>>,
                      girlHiLikes: Observable<Girl>,
                      answerToTheQuestionOfLife: Int) = viewFactory { }
```

Parameterize them all

```
val screen_main = fun(user: Observable<User>,
                      hisFriends: Observable<List<Friend>>,
                      hisPhotos: Observable<List<Photo>>,
                      hisComments: Observable<List<Comment>>,
                      girlHiLikes: Observable<Girl>,
                      answerToTheQuestionOfLife: Int) = viewFactory { }
```

View Model

```
class MainScreenViewModel(source: Source) {
    val user: Observable<User> = source.getUser()
    val hisFriends: Observable<List<Friend>> = user.flatMap(source::getFriendsOfUser)
    val hisPhotos: Observable<List<Photo>> = user.flatMap(source::getPhotosOfUser)
    val hisComments: Observable<List<Comment>> = source.getHisComments()
    val girlHiLikes: Observable<Girl> = source.getGirlHiLikes()
}

val screen_main = fun(viewModel: MainScreenViewModel) = viewFactory {
    verticalLinearLayout {
        addView(user_profile(viewModel.user))

        viewPager {
            viewModel.hisPhotos
                    .map(::userPhotosViewPagerAdapter)
                    .subscribe(::setAdapter)
        }.lparams(width = matchParent, height = dip(300))

        textView {
            viewModel.hisComments
                    .map { "User has ${it.size} comments" }
                    .subscribe(::setText)
        }

        imageView {
            viewModel.girlHiLikes
                    .map(Girl::avatarUrl)
                    .subscribe(::loadUrl)
        }.lparams(dip(150), dip(150))
    }
}
```

Don’t forget about lifecycle

```
import com.jakewharton.rxbinding.*

fun <T> Observable<T>.manageLifecycle(view: View) = this
        .observeOn(AndroidSchedulers.mainThread())
        .delaySubscription(view.attaches())
        .takeUntil(view.detaches())
        .repeatWhen { view.attaches() }
```

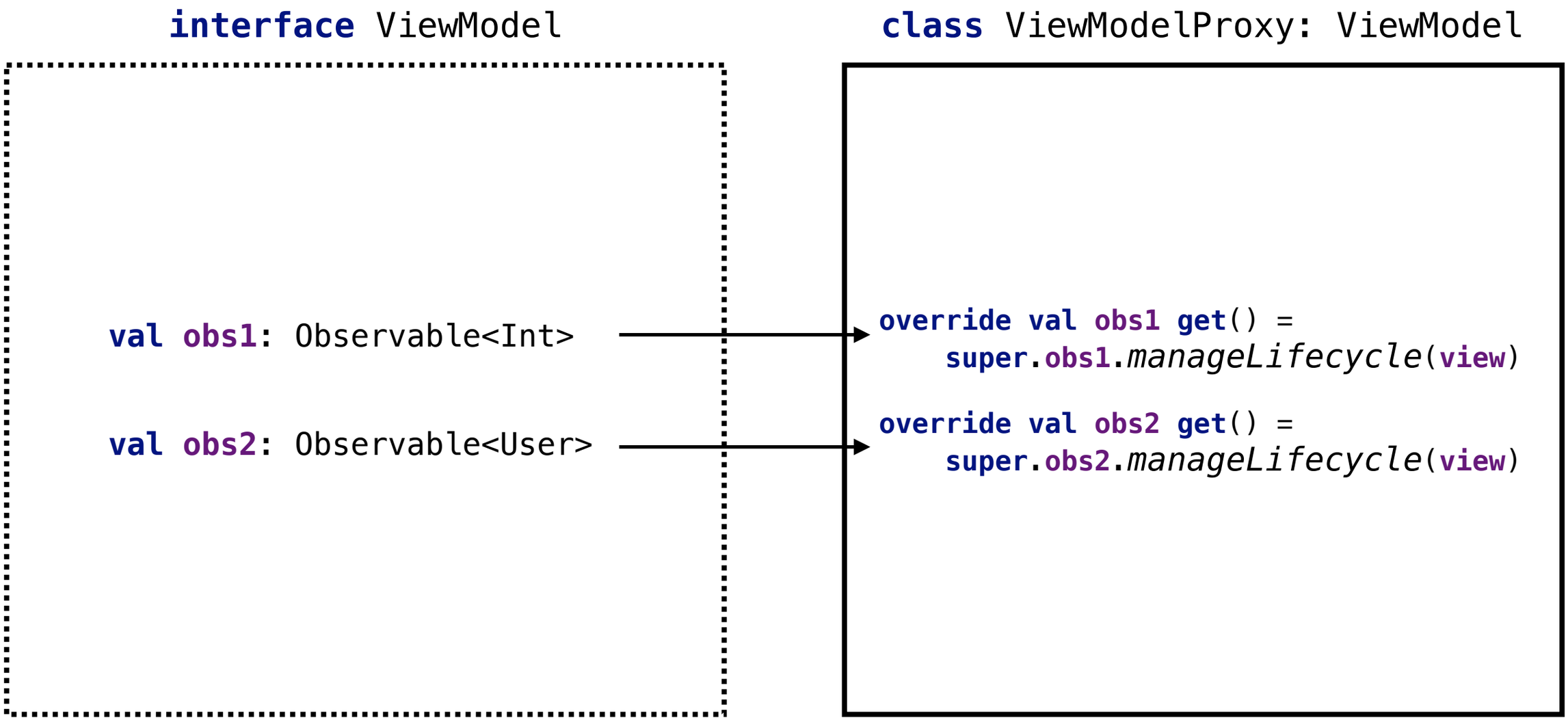
```
val screen_main = fun(viewModel: MainScreenViewModel) = viewFactory {
    verticalLinearLayout {

        viewPager {
            viewModel.hisPhotos
                    .manageLifecycle(view = this)
                    .map(::userPhotosViewPagerAdapter)
                    .subscribe(::setAdapter)
        }.lparams(width = matchParent, height = dip(300))

        textView {
            viewModel.hisComments
                    .manageLifecycle(view = this)
                    .map { "User has ${it.size} comments" }
                    .subscribe(::setText)
        }
    }
}
```

Lifecycle management is error-prone

```
val screen_main = fun(viewModel: MainScreenViewModel) = viewFactory {
    scrollView {
        verticalLinearLayout {

            viewPager {
                viewModel.hisPhotos
                        .manageLifecycle(view = this)
                        .map(::userPhotosViewPagerAdapter)
                        .subscribe(::setAdapter)
            }.lparams(width = matchParent, height = dip(300))

            textView {
                viewModel.hisComments
                        .manageLifecycle(view = this)
                        .map { "User has ${it.size} comments" }
                        .subscribe(::setText)
            }

            imageView {
                viewModel.girlHiLikes
                        .map(Girl::avatarUrl)
                        .subscribe(::loadUrl)
            }.lparams(dip(150), dip(150))
        }
    }
}
```

Leak

View Model Lifecycle Proxy

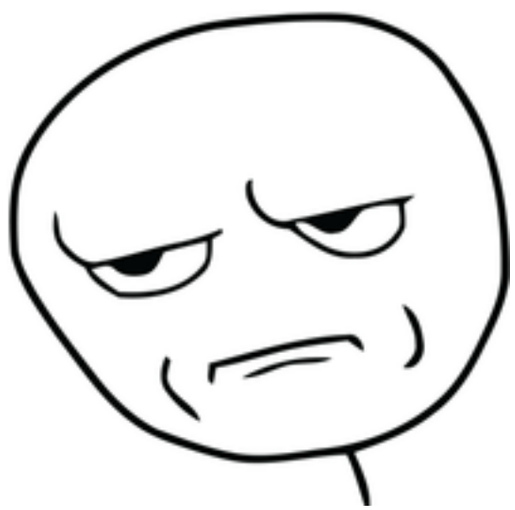


Lifecycle Proxy Implemenation

```
interface MyViewModel {
    val answer get() = Observable.just(42)

    class Impl : MyViewModel
}

class MyViewModelLifecycleProxy(val view: View): MyViewModel by MyViewModel.Impl() {
    override val answer: Observable<Int>

        get() = super.answer
                .observeOn(AndroidSchedulers.mainThread())
                .delaySubscription(view.attaches())
                .takeUntil(view.detaches())
                .repeatWhen { view.attaches() }
}
```

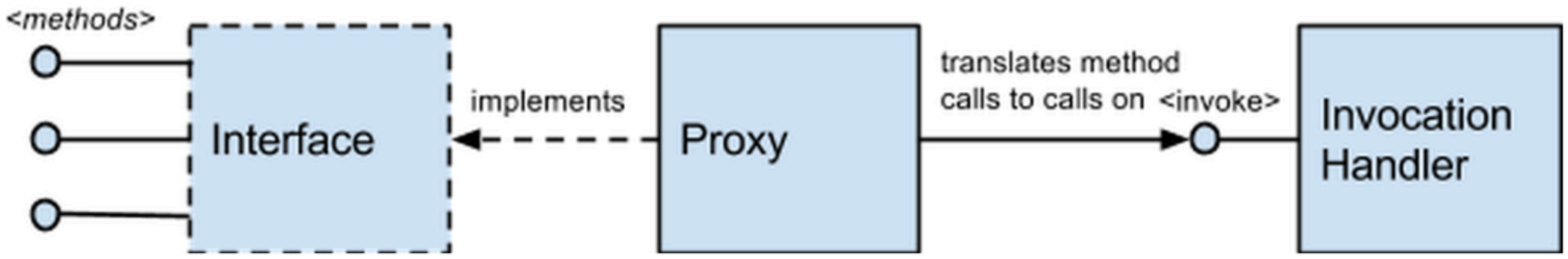
Lifecycle Proxy Implemenation

```
interface MyViewModel {
    val answer get() = Observable.just(42)

    class Impl : MyViewModel
}

class MyViewModelLifecycleProxy(val view: View): MyViewModel by MyViewModel.Impl() {
    override val answer: Observable<Int>

        get() = super.answer
                .observeOn(AndroidSchedulers.mainThread())
                .delaySubscription(view.attaches())
                .takeUntil(view.detaches())
                .repeatWhen { view.attaches() }
}
```



Java Dynamic Proxy

The class `java.lang.reflect.Proxy` allows you to implement interfaces dynamically by handling method calls in an `InvocationHandler`. It is considered part of Java's reflection facility, has nothing to do with bytecode generation.



For example Retrofit

```
public interface GitHubService {
    @GET("users/{user}/repos")
    Call<List<Repo>> listRepos(@Path("user") String user);
}
```

```
GitHubService service = retrofit.create(GitHubService.class);
Call<List<Repo>> repos = service.listRepos("octocat");
```

```
public <T> Retrofit.create(final Class<T> service) {
    return (T) Proxy.newProxyInstance(service.getClassLoader(), new Class<?>[]{service},
            new InvocationHandler() {
                @Override
                public Object invoke(Object proxy, Method method, @Nullable Object[] args)
                        throws Throwable {
                    ServiceMethod<Object, Object> serviceMethod =
                            (ServiceMethod<Object, Object>) loadServiceMethod(method);
                    OkHttpCall<Object> okHttpCall = new OkHttpCall<>(serviceMethod, args);
                    return serviceMethod.callAdapter.adapt(okHttpCall);
                }
            });
}
```

View Model Dynamic Lifecycle Proxy

```
interface ViewModelMarker

fun <VM : ViewModelMarker> dynamicLifecycleProxy(viewModel: VM,
                                                 vmInterface: Class<*>,
                                                 view: Observable<View>): VM {
    val handler = InvocationHandler { proxy, method, args ->

        if (method.returnType == rx.Observable::class.java) {
            (method.invoke(viewModel, args) as Observable<*>)
                    .observeOn(AndroidSchedulers.mainThread())
                    .delaySubscription(view.flatMap(View::attaches))
                    .takeUntil(view.flatMap(View::detaches))
                    .repeatWhen { view.flatMap(View::attaches) }
        } else {
            method.invoke(viewModel, args)
        }
    }

    return Proxy.newProxyInstance(
            classLoader = vmInterface.classLoader,
            interfaces = arrayOf(vmInterface),
            invocationHandler = handler
    ) as VM
}
```

How to use

```
interface MainScreenViewModel : ViewModelMarker {
    val userName get() = Observable.just("Foobar")

    class Impl: MainScreenViewModel
}

fun usage(context: Context) {
    1) val viewModel = MainScreenViewModel.Impl()

    2) val viewBus = PublishSubject.create<View>()

    3) val viewModelProxy = createLifecycleProxy(
            viewModel,
            MainScreenViewModel::class.java,
            viewBus)

    4) val mainScreenViewFactory = screen_main(viewModelProxy)
       val mainScreenView = mainScreenViewFactory(context)

    5) viewBus.onNext(mainScreenView)
}

val screen_main = fun(viewModel: MainScreenViewModel) = viewFactory {
    frameLayout {
        textView(viewModel.userName)
    }
}
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View

typealias ViewWithModelFactory<VM> = (Context, VM) -> View
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View

typealias ViewWithModelFactory<VM> = (Context, VM) -> View


fun viewFactory(factory: Context.() -> View)
        : ViewFactory = { ctx -> ctx.factory() }
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View

typealias ViewWithModelFactory<VM> = (Context, VM) -> View


fun viewFactory(factory: Context.() -> View)
        : ViewFactory = { ctx -> ctx.factory() }

fun <VM : ViewModelMarker> viewWithModelFactory(factory: Context.(VM) -> View)
        : ViewWithModelFactory<VM> = { ctx, vm -> ctx.factory(vm) }
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View

typealias ViewWithModelFactory<VM> = (Context, VM) -> View


fun viewFactory(factory: Context.() -> View)
        : ViewFactory = { ctx -> ctx.factory() }

fun <VM : ViewModelMarker> viewWithModelFactory(factory: Context.(VM) -> View)
        : ViewWithModelFactory<VM> = { ctx, vm -> ctx.factory(vm) }
```

```
fun <VM : ViewModelMarker> ViewWithModelFactory<VM>.bind(vmFactory: () -> VM)
        : ViewFactory = { ctx -> ... createLifecycleProxy ... }
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View

typealias ViewWithModelFactory<VM> = (Context, VM) -> View


fun viewFactory(factory: Context.() -> View)
        : ViewFactory = { ctx -> ctx.factory() }

fun <VM : ViewModelMarker> viewWithModelFactory(factory: Context.(VM) -> View)
        : ViewWithModelFactory<VM> = { ctx, vm -> ctx.factory(vm) }
```

```
fun <VM : ViewModelMarker> ViewWithModelFactory<VM>.bind(vmFactory: () -> VM)
        : ViewFactory = { ctx -> ... createLifecycleProxy ... }
```

```
setContentView(screen_main.bind(MainScreenViewModel::Impl))

val screen_main = viewWithModelFactory<MainScreenViewModel> {
    frameLayout {
        textView(it.userName)
    }
}
```

it ??

Context With View Model

ContextWrapper	Proxying implementation of Context that simply delegates all of its calls to another Context.

```
class ContextWithViewModel<VM : ViewModelMarker>(
        val viewModel: VM,
        context: Context
) : ContextWrapper(context)
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View

typealias ViewWithModelFactory<VM> = (ContextWithViewModel<VM>) -> View
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View

typealias ViewWithModelFactory<VM> = (ContextWithViewModel<VM>) -> View


fun viewFactory(factory: Context.() -> View)
        : ViewFactory = { ctx -> ctx.factory() }


fun <VM : ViewModelMarker> viewWithModelFactory(fct: ContextWithViewModel<VM>.() -> View)
        : ViewWithModelFactory<VM> = { ctx -> ctx.fct() }
```

Let’s simplify

```
typealias ViewFactory = (Context) -> View

typealias ViewWithModelFactory<VM> = (ContextWithViewModel<VM>) -> View


fun viewFactory(factory: Context.() -> View)
        : ViewFactory = { ctx -> ctx.factory() }


fun <VM : ViewModelMarker> viewWithModelFactory(fct: ContextWithViewModel<VM>.() -> View)
        : ViewWithModelFactory<VM> = { ctx -> ctx.fct() }
```

```
fun <VM : ViewModelMarker> ViewWithModelFactory<VM>.bind(vmFactory: () -> VM)
        : ViewFactory = { ctx -> ... createLifecycleProxy ... }



setContentView(screen_main.bind(MainScreenViewModel::Impl))

val screen_main = viewWithModelFactory<MainScreenViewModel> {
    frameLayout {
        textView(viewModel.userName)
    }
}
```

An example

```
data class Listing(
        val id: String,
        val make: String?,
        val model: String?,
        val price: Int?,
        ...
)

interface ListingViewModel : ViewModelMarker {
    val listing: Observable<Listing>

    fun bookmark() {
        id.take(1).subscribe(Logic::bookmarkListing)
    }

    fun smsToSeller(message: String) { ... }
    fun callToSeller() { ... }
    fun report(reason: String) { ... }

    class Impl(override val listing: Observable<Listing>) : ListingViewModel

    companion object {
        fun byId(listingId: String) = Impl(Logic.getListingById(listingId))
    }
}

val ListingViewModel.id      get() = listing.map(Listing::id)
val ListingViewModel.make    get() = listing.map(Listing::make)
val ListingViewModel.model   get() = listing.map(Listing::model)
val ListingViewModel.price   get() = listing.map(Listing::price)
```

viewWithModel<ListingViewModel>

```
...
val listing_recommended_grid_item = viewWithModel<ListingViewModel> {  }
val listing_discovery_feed_item   = viewWithModel<ListingViewModel> {  }
val listing_horizontal_view       = viewWithModel<ListingViewModel> {  }
val listing_profile_screen        = viewWithModel<ListingViewModel> {  }
…
```

37 matches in 5 files
thousands of lines of DSL
several ID for testing purpose
several ID for view state saving
0 lines of lifecycle management

Dynamic Proxy Performance

- Nexus 5X (mid 2015)
- Android 8.0.0

Just proxy instantiation

Methods in interface	First (ms)	Subsequent (ms)
0	0.1	0.15
10	1.3	0.15
100	6.4	0.15

Method invocation

Without any modification

Calls	Time (ms)
100	0.6
1000	6.4
10000	69

With rx.Observable modification

Calls	Time (ms)
100	2.5
1000	13.3
10000	146

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative
  - Control-flow: if, when, for, while

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative
  - Control-flow: if, when, for, while
  - Async programming: Promise, RxJava, Coroutines

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative
  - Control-flow: if, when, for, while
  - Async programming: Promise, RxJava, Coroutines
  - …
  - ALL THE POWER OF KOTLIN in your layouts

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative
  - Control-flow: if, when, for, while
  - Async programming: Promise, RxJava, Coroutines
  - …
  - ALL THE POWER OF KOTLIN in your layouts
- Easy to use

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative
  - Control-flow: if, when, for, while
  - Async programming: Promise, RxJava, Coroutines
  - …
  - ALL THE POWER OF KOTLIN in your layouts
- Easy to use
- A bit faster at runtime than XML

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative
  - Control-flow: if, when, for, while
  - Async programming: Promise, RxJava, Coroutines
  - …
  - ALL THE POWER OF KOTLIN in your layouts
- Easy to use
- A bit faster at runtime than XML

Drawbacks:

- Tooling is not so good yet
  - No preview
  - No drag-n-drop layout development
  - No layout qualifiers: layout-xlarge-land, layout-sw720dp

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative
  - Control-flow: if, when, for, while
  - Async programming: Promise, RxJava, Coroutines
  - …
  - ALL THE POWER OF KOTLIN in your layouts
- Easy to use
- A bit faster at runtime than XML

Drawbacks:

- Tooling is not so good yet
  - No preview
  - No drag-n-drop layout development
  - No layout qualifiers: layout-xlarge-land, layout-sw720dp
- DSL is code
  - More time to compile
  - More DEX methods

Ok, Layouts DSL

Benefits:

- DSL is imperative, but declarative, but imperative
  - Control-flow: if, when, for, while
  - Async programming: Promise, RxJava, Coroutines
  - …
  - ALL THE POWER OF KOTLIN in your layouts
- Easy to use
- A bit faster at runtime than XML

Drawbacks:

- Tooling is not so good yet
  - No preview
  - No drag-n-drop layout development
  - No layout qualifiers: layout-xlarge-land, layout-sw720dp
- DSL is code
  - More time to compile
  - More DEX methods
- Yet another hipster thing

- Contact: stas@instamotor.com
- Code: bit.ly/dsl-the-only-way

Q&A

```
textView("Q&A") {
    textSize = 100f
    font = Fonts.HELVETICA
}.lparams {
    gravity = Gravity.CENTER
}
```

WANTED

Kotlin Android Developer

instamotor.com/careers