Kotlin is here

Life is great and everything will be OK

+Christina Lee
@RunChristinaRun

+Jake Wharton
@JakeWharton

MainActivity.java

```java
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

MainActivity.java

```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

MainActivity.java

```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

MainActivity.java

```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

MainActivity.java

```java
public class MainActivity extends Activity {
  @Override
  protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

MainActivity.kt

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

MainActivity.kt

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

MainActivity.kt

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

TextView.java

```java
public float getAlpha() {
  // Retrieve value...
}
```

TextView.java

```
public float getAlpha() {
  // Retrieve value...
}

public void setAlpha(float alpha) {
  // Set value...
}
```

TextView.java

```
public float getAlpha() {
  // Retrieve value...
}

public void setAlpha(float alpha) {
  // Set value...
}
```

MainActivity.java

```
TextView tv = // ...
Log.d("MainActivity", "Alpha: " + tv.getAlpha());
tv.setAlpha(0f);
```

TextView.java

```java
public float getAlpha() {
  // Retrieve value...
}

public void setAlpha(float alpha) {
  // Set value...
}
```

MainActivity.kt

```kotlin
val tv = // ...
Log.d("MainActivity", "Alpha: " + tv.alpha)
tv.alpha = 0f
```

TextView.java

```java
public float getAlpha() {
  // Retrieve value...
}

public void setAlpha(float alpha) {
  // Set value...
}
```

MainActivity.kt

```kotlin
val tv = // ...
Log.d("MainActivity", "Alpha: " + tv.alpha)
tv.alpha = 0f
```

TextView.java

```java
public float getAlpha() {
  // Retrieve value...
}

public void setAlpha(float alpha) {
  // Set value...
}
```

MainActivity.kt

```kotlin
val tv = // ...
Log.d("MainActivity", "Alpha: " + tv.alpha)
tv.alpha = 0f
```

TextView.java

```java
public float getAlpha() {
  // Retrieve value...
}

public void setAlpha(float alpha) {
  // Set value...
}
```

MainActivity.kt

```kotlin
val tv = // ...
Log.d("MainActivity", "Alpha: " + tv.alpha)
tv.alpha = 0f
```

MainActivity.java

```java
LinearLayout views = // ...
for (int i = 0; i < views.getChildCount(); i++) {
  View view = views.getChildAt(i);
  // TODO do something with view
}
```

MainActivity.kt

```kotlin
val views = // ...
for (index in 0 until views.childCount) {
  val view = views.getChildAt(index)
  // TODO do something with view
}
```

MainActivity.kt

```kotlin
val views = // ...
for (index in 0 until views.childCount) {
  val view = views.getChildAt(index)
  // TODO do something with view
}
```

MainActivity.kt

```
val views = // ...
for (index in 0 until views.childCount) {
  val view = views.getChildAt(index)
  // TODO do something with view
}
```

MainActivity.kt

```kotlin
val views = // ...
for (index in 0 until views.childCount) {
  val view = views.getChildAt(index)
  // TODO do something with view
}
```

MainActivity.kt

```kotlin
val views = // ...
for (index in 0 until views.childCount) {
  val view = views.getChildAt(index)
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view ->
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view ->
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view ->
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view ->
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view ->
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view ->
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
inline fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view ->
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
inline fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view -> /* ... */ }
views.forEachIndexed { index, view -> /* ... */ }
```

ViewGroups.kt

```kotlin
inline fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
  for (index in 0 until childCount) {
    action(index, getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
views.forEach { view -> /* ... */ }
views.forEachIndexed { index, view -> /* ... */ }
```

ViewGroups.kt

```kotlin
inline fun ViewGroup.forEach(action: (View) -> Unit) {
  for (index in 0 until childCount) {
    action(getChildAt(index))
  }
}
inline fun ViewGroup.forEachIndexed(action: (Int, View) -> Unit) {
  for (index in 0 until childCount) {
    action(index, getChildAt(index))
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1
```

MainActivity.kt

```kotlin
val views = // ...
val first = views.get(0)
views.minusAssign(first)
views.plusAssign(first)
if (views.contains(first)) doSomething()
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
Log.d("MainActivity", "View count: ${views.size}")
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1

val ViewGroup.size: Int
  get() = childCount
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
Log.d("MainActivity", "View count: ${views.size}")
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1

val ViewGroup.size: Int
  get() = childCount
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
Log.d("MainActivity", "View count: ${views.size}")
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1

val ViewGroup.size: Int
  get() = childCount
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
Log.d("MainActivity", "View count: ${views.size}")
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1

val ViewGroup.size: Int
  get() = childCount
```

MainActivity.kt

```kotlin
val views = // ...
val first = views[0]
views -= first
views += first
if (first in views) doSomething()
Log.d("MainActivity", "View count: ${views.size}")
```

ViewGroups.kt

```kotlin
operator fun ViewGroup.get(index: Int): View? = getChildAt(index)
operator fun ViewGroup.minusAssign(child: View) = removeView(child)
operator fun ViewGroup.plusAssign(child: View) = addView(child)
operator fun ViewGroup.contains(child: View) = indexOfChild(child) != -1

val ViewGroup.size: Int
  get() = childCount
```

MainActivity.kt

```kotlin
val views = // ...
```

ViewGroups.kt

```kotlin
fun ViewGroup.children() = object : Iterable<View> {
  override fun iterator() = object : Iterator<View> {
    var index = 0
    override fun hasNext() = index < childCount
    override fun next() = getChildAt(index++)
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
for (view in views.children()) {
  // TODO do something with view
}
```

ViewGroups.kt

```kotlin
fun ViewGroup.children() = object : Iterable<View> {
  override fun iterator() = object : Iterator<View> {
    var index = 0
    override fun hasNext() = index < childCount
    override fun next() = getChildAt(index++)
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
for (view in views.children()) {
  // TODO do something with view
}
val visibleHeight = views.children()
  .filter { it.visibility == View.VISIBLE }
  .sumBy { it.measuredHeight }
```

ViewGroups.kt

```kotlin
fun ViewGroup.children() = object : Iterable<View> {
  override fun iterator() = object : Iterator<View> {
    var index = 0
    override fun hasNext() = index < childCount
    override fun next() = getChildAt(index++)
  }
}
```

MainActivity.kt

```kotlin
val views = // ...
for (view in views.children()) {
  // TODO do something with view
}
val visibleHeight = views.children()
  .filter { it.visibility == View.VISIBLE }
  .sumBy { it.measuredHeight }
```

ViewGroups.kt

```kotlin
fun ViewGroup.children() = object : Iterable<View> {
  override fun iterator() = object : Iterator<View> {
    var index = 0
    override fun hasNext() = index < childCount
    override fun next() = getChildAt(index++)
  }
}
```

MainActivity.java

```java
Trace.beginSection(sectionName);
expensiveCalculation();
Trace.endSection();
```

MainActivity.java

```java
Trace.beginSection(sectionName);
expensiveCalculation();
Trace.endSection();
```

Traces.kt

```kotlin
inline fun trace(sectionName: String, body: () -> Unit) {
  Trace.beginSection(sectionName)
  try {
    body()
  } finally {
    Trace.endSection()
  }
}
```

MainActivity.kt

```kotlin
trace("foo") {
  expensiveCalculation()
}
```

Traces.kt

```kotlin
inline fun trace(sectionName: String, body: () -> Unit) {
  Trace.beginSection(sectionName)
  try {
    body()
  } finally {
    Trace.endSection()
  }
}
```

MainActivity.kt

```kotlin
trace("foo") {
  expensiveCalculation()
}
```

Traces.kt

```kotlin
inline fun <T> trace(sectionName: String, body: () -> T): T {
  Trace.beginSection(sectionName)
  try {
    return body()
  } finally {
    Trace.endSection()
  }
}
```

MainActivity.kt

```kotlin
val result = trace("foo") {
  expensiveCalculation()
}
```

Traces.kt

```kotlin
inline fun <T> trace(sectionName: String, body: () -> T): T {
  Trace.beginSection(sectionName)
  try {
    return body()
  } finally {
    Trace.endSection()
  }
}
```

MainActivity.kt

```kotlin
val result = trace("foo") {
  expensiveCalculation()
}
```

Traces.kt

```kotlin
inline fun <T> trace(sectionName: String, body: () -> T): T {
  Trace.beginSection(sectionName)
  try {
    return body()
  } finally {
    Trace.endSection()
  }
}
```

MainActivity.java

```
SQLiteDatabase db = // ...
db.beginTransaction();
try {
  db.delete("users", "first_name = ?", new String[] { "jake" });
} finally {
  db.endTransaction();
}
```

MainActivity.java

```java
SQLiteDatabase db = // ...
db.beginTransaction();
try {
  db.delete("users", "first_name = ?", new String[] { "jake" });
  db.setTransactionSuccessful();
} finally {
  db.endTransaction();
}
```

```
SQLiteDatabase db = // ...
db.beginTransaction();
try {
  db.delete("users", "first_name = ?", new String[] { "jake" });
  db.setTransactionSuccessful();
} finally {
  db.endTransaction();
}
```

Databases.kt

```
inline fun SQLiteDatabase.transaction(body: () -> Unit) {
  beginTransaction()
  try {
    body()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  db.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: () -> Unit) {
  beginTransaction()
  try {
    body()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```
val db = // ...
db.transaction {
  db.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```
inline fun SQLiteDatabase.transaction(body: () -> Unit) {
  beginTransaction()
  try {
    body()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  db.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: () -> Unit) {
  beginTransaction()
  try {
    body()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  db.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: (SQLiteDatabase) -> Unit) {
  beginTransaction()
  try {
    body()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  db.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: (SQLiteDatabase) -> Unit) {
  beginTransaction()
  try {
    body(this)
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  it.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: (SQLiteDatabase) -> Unit) {
  beginTransaction()
  try {
    body(this)
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```
val db = // ...
db.transaction {
  it.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```
inline fun SQLiteDatabase.transaction(body: (SQLiteDatabase) -> Unit) {
  beginTransaction()
  try {
    body(this)
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  it.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: (SQLiteDatabase) -> Unit) {
  beginTransaction()
  try {
    body(this)
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```
val db = // ...
db.transaction {
  it.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```
inline fun SQLiteDatabase.transaction(body: (SQLiteDatabase) -> Unit) {
  beginTransaction()
  try {
    body(this)
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  it.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: SQLiteDatabase.() -> Unit) {
  beginTransaction()
  try {
    body(this)
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  it.delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: SQLiteDatabase.() -> Unit) {
  beginTransaction()
  try {
    body()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: SQLiteDatabase.() -> Unit) {
  beginTransaction()
  try {
    body()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

MainActivity.kt

```kotlin
val db = // ...
db.transaction {
  delete("users", "first_name = ?", arrayOf("jake"))
}
```

Databases.kt

```kotlin
inline fun SQLiteDatabase.transaction(body: SQLiteDatabase.() -> Unit) {
  beginTransaction()
  try {
    body()
    setTransactionSuccessful()
  } finally {
    endTransaction()
  }
}
```

UserPersistence.kt

```kotlin
class UserPersistence(private val db: SQLiteDatabase) {
  fun deleteByFirstName(name: String) {
    db.transaction {
      delete("users", "first_name = ?", arrayOf(name))
    }
  }
}
```

UserPersistence.kt

```
class UserPersistence(private val db: SQLiteDatabase) {
  fun deleteByFirstName(name: String) {
    db.transaction {
      delete("users", "first_name = ?", arrayOf(name))
    }
  }
}
```

UserPersistence.kt

```kotlin
class UserPersistence(private val db: SQLiteDatabase) {
  fun deleteByFirstName(name: String) {
    db.transaction {
      delete("users", "first_name = ?", arrayOf(name))
    }
  }
}
```

UserPersistence.kt

```kotlin
class UserPersistence(private val db: SQLiteDatabase) {
  private val deleteByFirstName = db.compileStatement(
      "DELETE FROM users WHERE first_name = ?")

  fun deleteByFirstName(name: String) {
    db.transaction {
      deleteByFirstName.bindString(1, name)
      deleteByFirstName.execute()
    }
  }
}
```

UserPersistence.kt

```
class UserPersistence(private val db: SQLiteDatabase) {
  private val deleteByFirstName = db.compileStatement(
      "DELETE FROM users WHERE first_name = ?")

  fun deleteByFirstName(name: String) {
    db.transaction {
      deleteByFirstName.bindString(1, name)
      deleteByFirstName.execute()
    }
  }
}
```

UserPersistence.kt

```kotlin
class UserPersistence(private val db: SQLiteDatabase) {
  private val deleteByFirstName by lazy {
    db.compileStatement("DELETE FROM users WHERE first_name = ?")
  }

  fun deleteByFirstName(name: String) {
    db.transaction {
      deleteByFirstName.bindString(1, name)
      deleteByFirstName.execute()
    }
  }
}
```

UserPersistence.kt

```kotlin
class UserPersistence(private val db: SQLiteDatabase) {
  private val deleteByFirstName by lazy {
    db.compileStatement("DELETE FROM users WHERE first_name = ?")
  }

  fun deleteByFirstName(name: String) {
    db.transaction {
      deleteByFirstName.bindString(1, name)
      deleteByFirstName.execute()
    }
  }
}
```

```kotlin
private val deleteByFirstName by lazy {
  db.compileStatement("DELETE FROM users WHERE first_name = ?")
}
```

```kotlin
private val deleteByFirstName by lazy {
  db.compileStatement("DELETE FROM users WHERE first_name = ?")
}

private val name by Delegates.observable("jane") { old, new, prop ->
  println("Name changed from $old to $new")
}
```

```
private val deleteByFirstName by lazy {
  db.compileStatement("DELETE FROM users WHERE first_name = ?")
}

private val name by Delegates.observable("jane") { old, new, prop ->
  println("Name changed from $old to $new")
}

private val address by Delegates.notNull<String>()
```

```kotlin
private val deleteByFirstName by lazy {
  db.compileStatement("DELETE FROM users WHERE first_name = ?")
}

private val name by Delegates.observable("jane") { old, new, prop ->
  println("Name changed from $old to $new")
}

private val address by Delegates.notNull<String>()

private val nameView by bindView<TextView>(R.id.name)
private val deleteByFirstName by lazy {
  db.compileStatement("DELETE FROM users WHERE first_name = ?")
}

private val name by Delegates.observable("jane") { old, new, prop ->
  println("Name changed from $old to $new")
}

private val address by Delegates.notNull<String>()

private val nameView by bindView<TextView>(R.id.name)
```

```kotlin
class MyListener : TransitionListener {
  override fun onTransitionEnd(transition: Transition) {
  }

  override fun onTransitionResume(transition: Transition) {
  }

  override fun onTransitionPause(transition: Transition) {
  }

  override fun onTransitionCancel(transition: Transition) {
  }

  override fun onTransitionStart(transition: Transition) {
  }
}
```

```
class MyListener : TransitionListener {
  override fun onTransitionStart(transition: Transition) {
  }
}
```

```
class MyListener : TransitionListener {
  override fun onTransitionStart(transition: Transition) {
  }
}

object EmptyTransitionListener : TransitionListener {
  override fun onTransitionEnd(transition: Transition) {}
  override fun onTransitionResume(transition: Transition) {}
  override fun onTransitionPause(transition: Transition) {}
  override fun onTransitionCancel(transition: Transition) {}
  override fun onTransitionStart(transition: Transition) {}
}
```

```
class MyListener : TransitionListener {
  override fun onTransitionStart(transition: Transition) {
  }
}

object EmptyTransitionListener : TransitionListener {
  override fun onTransitionEnd(transition: Transition) {}
  override fun onTransitionResume(transition: Transition) {}
  override fun onTransitionPause(transition: Transition) {}
  override fun onTransitionCancel(transition: Transition) {}
  override fun onTransitionStart(transition: Transition) {}
}
```

```
class MyListener : TransitionListener {
  override fun onTransitionStart(transition: Transition) {
  }
}

object EmptyTransitionListener : TransitionListener {
  override fun onTransitionEnd(transition: Transition) {}
  override fun onTransitionResume(transition: Transition) {}
  override fun onTransitionPause(transition: Transition) {}
  override fun onTransitionCancel(transition: Transition) {}
  override fun onTransitionStart(transition: Transition) {}
}
```

```kotlin
class MyListener : TransitionListener by EmptyTransitionListener {
  override fun onTransitionStart(transition: Transition) {
  }
}

object EmptyTransitionListener : TransitionListener {
  override fun onTransitionEnd(transition: Transition) {}
  override fun onTransitionResume(transition: Transition) {}
  override fun onTransitionPause(transition: Transition) {}
  override fun onTransitionCancel(transition: Transition) {}
  override fun onTransitionStart(transition: Transition) {}
}
```

```kotlin
class MyListener : TransitionListener by EmptyTransitionListener {
  override fun onTransitionStart(transition: Transition) {
  }
}

object EmptyTransitionListener : TransitionListener {
  override fun onTransitionEnd(transition: Transition) {}
  override fun onTransitionResume(transition: Transition) {}
  override fun onTransitionPause(transition: Transition) {}
  override fun onTransitionCancel(transition: Transition) {}
  override fun onTransitionStart(transition: Transition) {}
}
```

PaymentRobot.kt

```kotlin
class PaymentRobot {
  fun amount(value: Long) {
    // TODO Espresso interactions
  }

  fun recipient(value: String) {
    // TODO Espresso interactions
  }

  fun send() {
    // TODO Espresso interactions
  }
}
```

PaymentTest.kt

```kotlin
@Test fun sendMoney() {
  PaymentRobot().apply {
    amount(4_00)
    recipient("foo@example.com")
    send()
  }
}
```

PaymentRobot.kt

```kotlin
class PaymentRobot {
  fun amount(value: Long) {
    // TODO Espresso interactions
  }

  fun recipient(value: String) {
    // TODO Espresso interactions
  }

  fun send() {
    // TODO Espresso interactions
  }
}
```

PaymentRobot.kt

```kotlin
class PaymentRobot {
  fun amount(value: Long) {
    // TODO Espresso interactions
  }

  fun recipient(value: String) {
    // TODO Espresso interactions
  }

  fun send() {
    // TODO Espresso interactions
  }
}

fun payment(body: PaymentRobot.() -> Unit) = PaymentRobot().apply(body)
```

PaymentRobot.kt

```kotlin
class PaymentRobot {
  fun amount(value: Long) {
    // TODO Espresso interactions
  }

  fun recipient(value: String) {
    // TODO Espresso interactions
  }

  fun send() {
    // TODO Espresso interactions
  }
}

fun payment(body: PaymentRobot.() -> Unit) = PaymentRobot().apply(body)
```

PaymentTest.kt

```kotlin
@Test fun sendMoney() {
  PaymentRobot().apply {
    amount(4_00)
    recipient("foo@example.com")
    send()
  }
}
```

PaymentTest.kt

```kotlin
@Test fun sendMoney() {
  payment {
    amount(4_00)
    recipient("foo@example.com")
    send()
  }
}
```

```kotlin
sealed class Payloads {
}
```

```kotlin
sealed class Payloads {
  data class Favorite(val favorited: Boolean) : Payloads()
}
```

```kotlin
sealed class Payloads {
  data class Favorite(val favorited: Boolean) : Payloads()
  data class Retweet(val retweeted: Boolean) : Payloads()
}
```

```kotlin
sealed class Payloads {
  data class Favorite(val favorited: Boolean) : Payloads()
  data class Retweet(val retweeted: Boolean) : Payloads()
  data class CountUpdate(
      val favorites: Long,
      val retweets: Long,
      val replies: Long) : Payloads()
}
```

```kotlin
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
}
```

```kotlin
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
  payloads.forEach {
  }
}
```

```kotlin
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
  payloads.forEach {
    when (it) {
    }
  }
}
```

```kotlin
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
  payloads.forEach {
    when (it) {
      is Favorite -> holder.favoriteIcon.isActivated = it.favorited
    }
  }
}
```

```kotlin
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
  payloads.forEach {
    when (it) {
      is Favorite -> holder.favoriteIcon.isActivated = it.favorited
    }
  }
}
```

```
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
  payloads.forEach {
    when (it) {
      is Favorite -> holder.favoriteIcon.isActivated = it.favorited
    }
  }
}

sealed class Payloads {
  data class Favorite(val favorited: Boolean) : Payloads()
  data class Retweet(val retweeted: Boolean) : Payloads()
  data class CountUpdate(
      val favorites: Long,
      val retweets: Long,
      val replies: Long) : Payloads()
}
```

```kotlin
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
  payloads.forEach {
    when (it) {
      is Favorite -> holder.favoriteIcon.isActivated = it.favorited
    }
  }
}
```

```kotlin
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
  payloads.forEach {
    when (it) {
      is Favorite -> holder.favoriteIcon.isActivated = it.favorited
      is Retweet -> holder.retweetIcon.isActivated = it.retweeted
    }
  }
}
```

```
override fun onBindViewHolder(holder: TweetViewHolder,
      position: Int, payloads: List<Any>) {
  payloads.forEach {
    when (it) {
      is Favorite -> holder.favoriteIcon.isActivated = it.favorited
      is Retweet -> holder.retweetIcon.isActivated = it.retweeted
      is CountUpdate -> {
        holder.apply {
          favoriteCount.text = it.favorites.toString()
          retweetCount.text = it.retweets.toString()
          replyCount.text = it.replies.toString()
        }
      }
    }
  }
}
```

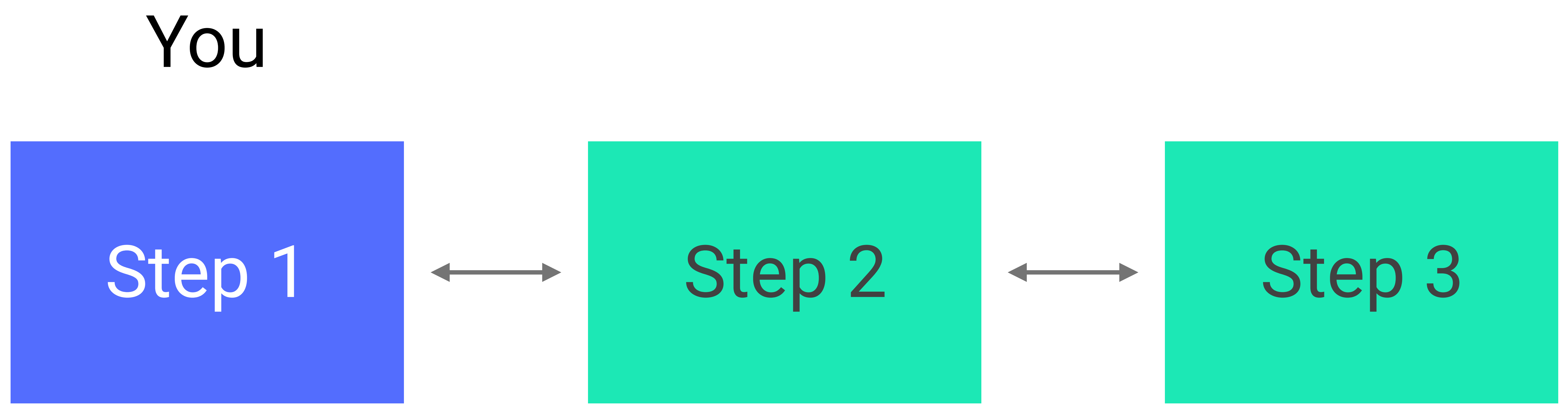
It takes great salesmanship to convince a customer to buy something from you that isn't built or isn't finished.

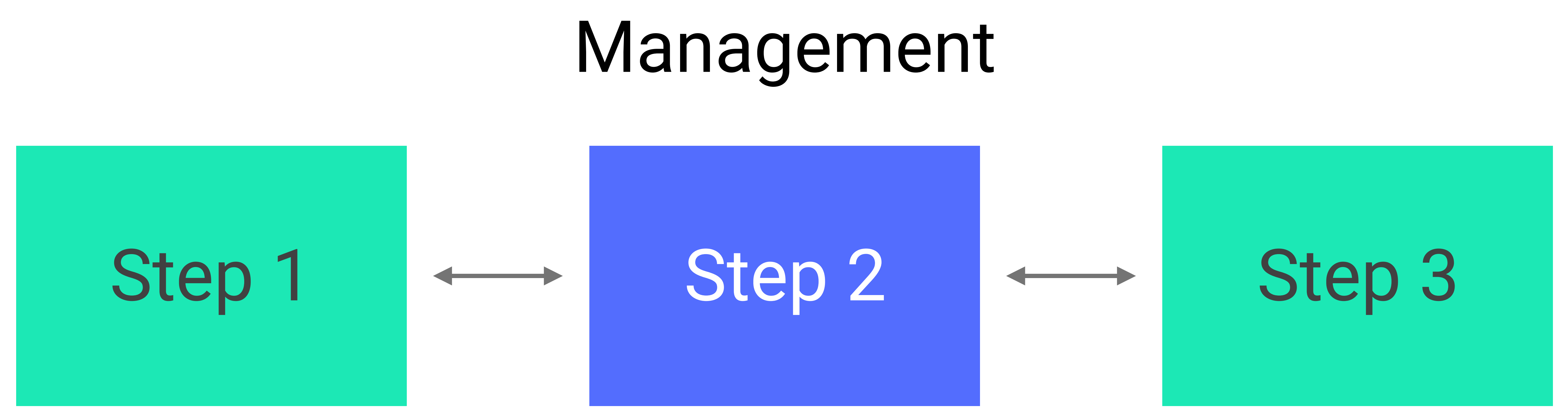
/ Fred Wilson

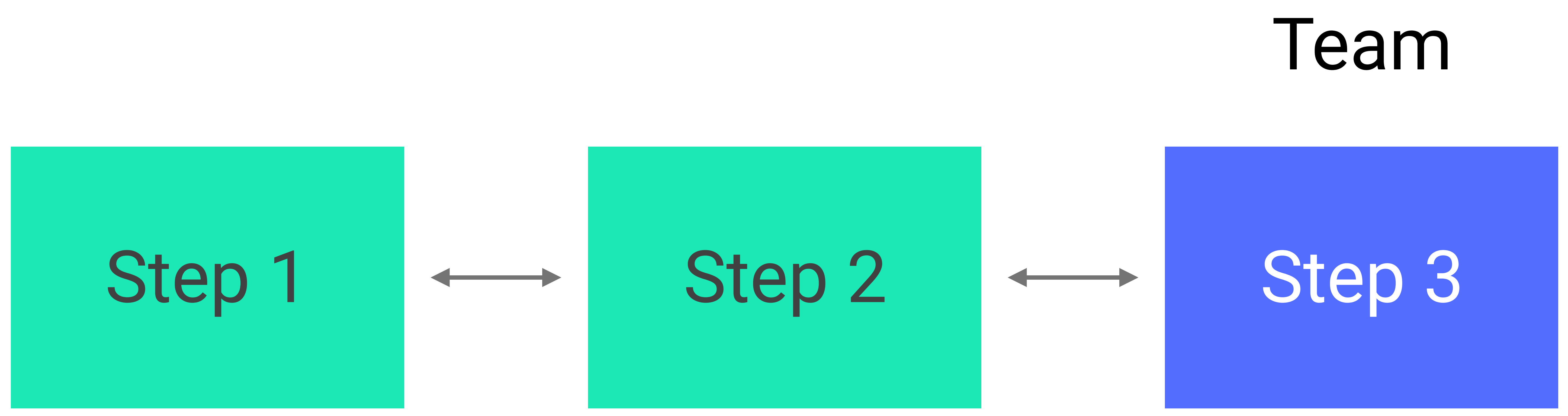
#io17

Google

Step 1
Step 2
Step 3


You
Step 1
Step 2
Step 3


Management
Step 1
Step 2
Step 3


Team
Step 1
Step 2
Step 3

You

Get excited.

Never doubt that a small group of thoughtful, committed citizens can change the world; indeed, it's the only thing that ever has.

Margaret Mead (Def. not talking about tech)

Be enthusiastic

Adoption is work. If you want it, you need to earn it.

Management

Be persuasive.

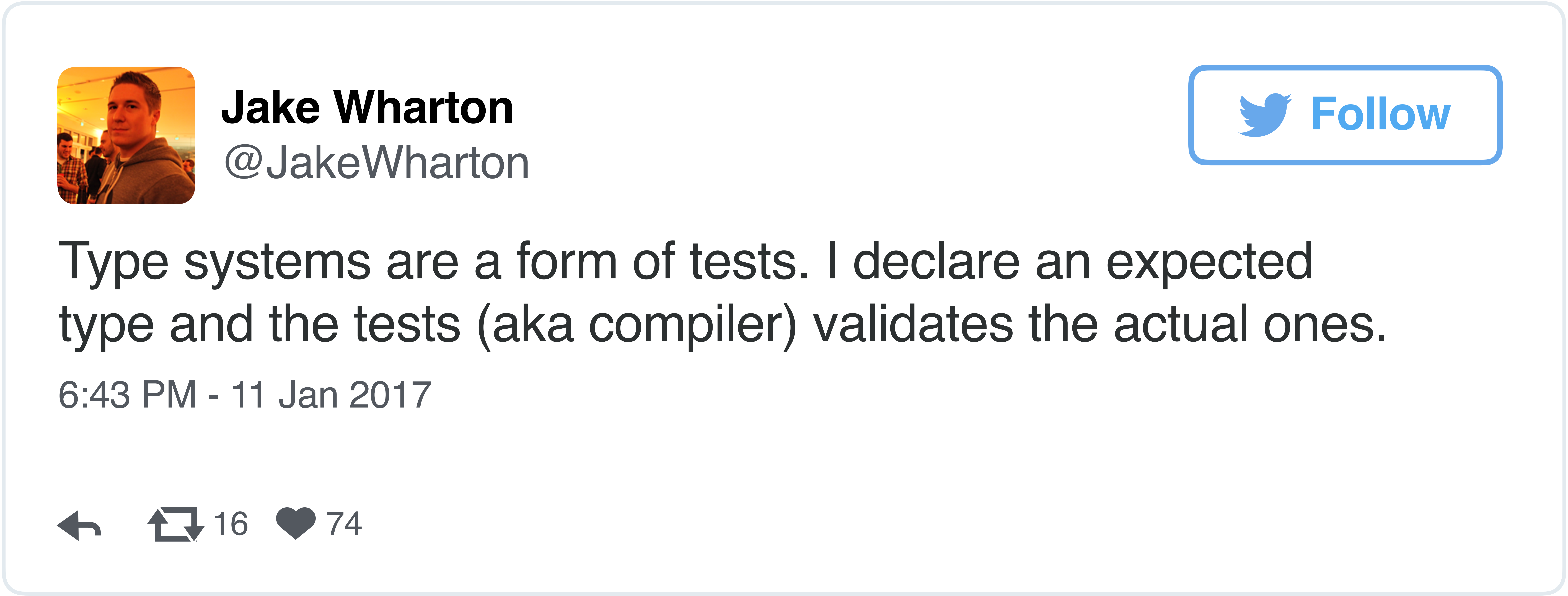
I call it my billion-dollar mistake.

Sir Charles Antony Richard Hoare

One of the most feared expressions in modern times is 'The computer is down.'

/ Norman Ralph Augustine

15
CHROME VANADIUM
15
17
CHROME VANADIUM
17
13
CHROME VANADIUM
8
CHROME VANADIUM
8



Jake Wharton
@JakeWharton

Follow

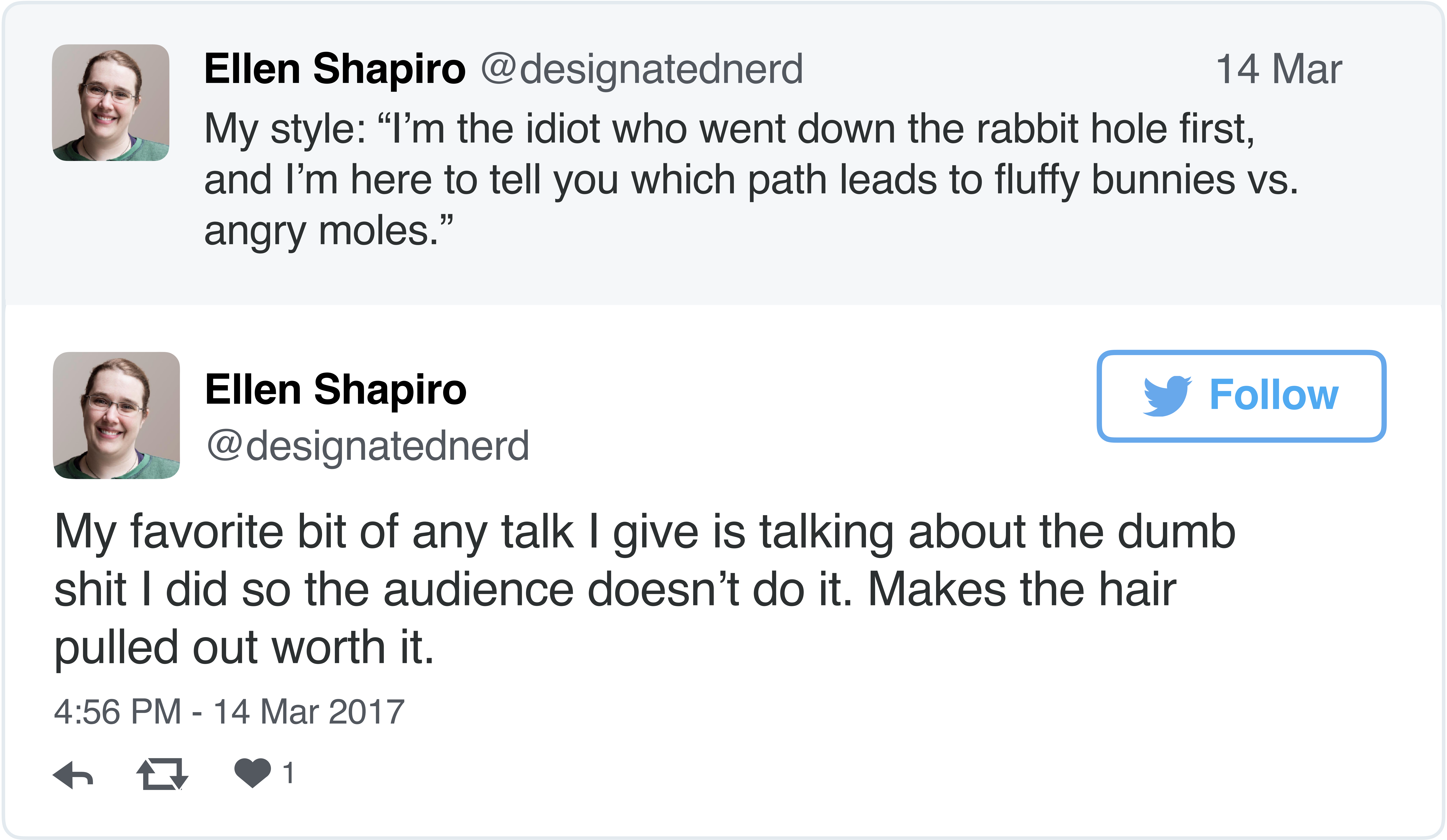
Type systems are a form of tests. I declare an expected type and the tests (aka compiler) validates the actual ones.

6:43 PM - 11 Jan 2017

16 74

Team

Do the work.



Ellen Shapiro @designatednerd 14 Mar

My style: “I’m the idiot who went down the rabbit hole first, and I’m here to tell you which path leads to fluffy bunnies vs. angry moles.”

Ellen Shapiro
@designatednerd

My favorite bit of any talk I give is talking about the dumb shit I did so the audience doesn’t do it. Makes the hair pulled out worth it.

4:56 PM - 14 Mar 2017

Define success

http://blog.danlew.net/

≡ MENU

Kotlin Puzzler: Whose Line Is It Anyways?

16 MARCH 2017 on android, kotlin

Here's a small Kotlin puzzler. What's wrong with the following code (when used on Android)?

```
val map = mapOf("hello" to "goodbye")
map.forEach { t, u -> Log.i("tag", t + u) }
```

I'll give you a hint: on older versions of Android, the above code crashes due to `java.lang.NoClassDefFoundError`.

Here's another hint: it has to do with destructuring declarations (or a lack thereof).

On-boarding should be a first class citizen

Show up

What's next?

Kotlin documentation and koans

https://kotlinlang.org/

Android Kotlin documentation

https://developer.android.com/kotlin/index.html

Kotlin In Action

Dimitry Jemerov, Svetlana Isakova

Thank you!

+Christina Lee
@RunChristinaRun

+Jake Wharton
@JakeWharton

#KotlinIsHere

#io17

#io17