Kotlin for Android

Experiences writing an app entirely in Kotlin

Agenda

1. About Me
2. Why Kotlin?
3. TL;DR
4. ATC Connect
5. Patterns
6. Pitfalls
7. Smells
8. Odds and Ends

About Me

- Android developer (1.5 years)
  - WWT Asynchrony Labs
- C/C++ (11 years)
  - Mostly flight simulator graphics

Why Kotlin?

- Lambda expressions
- Inline functions
- Extension functions
- Range expressions
- Operator overloading
- No wildcard types
- Null-safety
- Smart casts
- String templates
- Properties
- Primary constructors
- Type inference
- Companion objects
- Data classes
- Read-only collections
- No checked exceptions
- Singletons
- First-class delegation

TL;DR

Less boiling of plates

More composition over inheritance

Kotlin is easy compared to Java

You find yourself writing Java style

Don’t try too hard to make it Kotlinesque

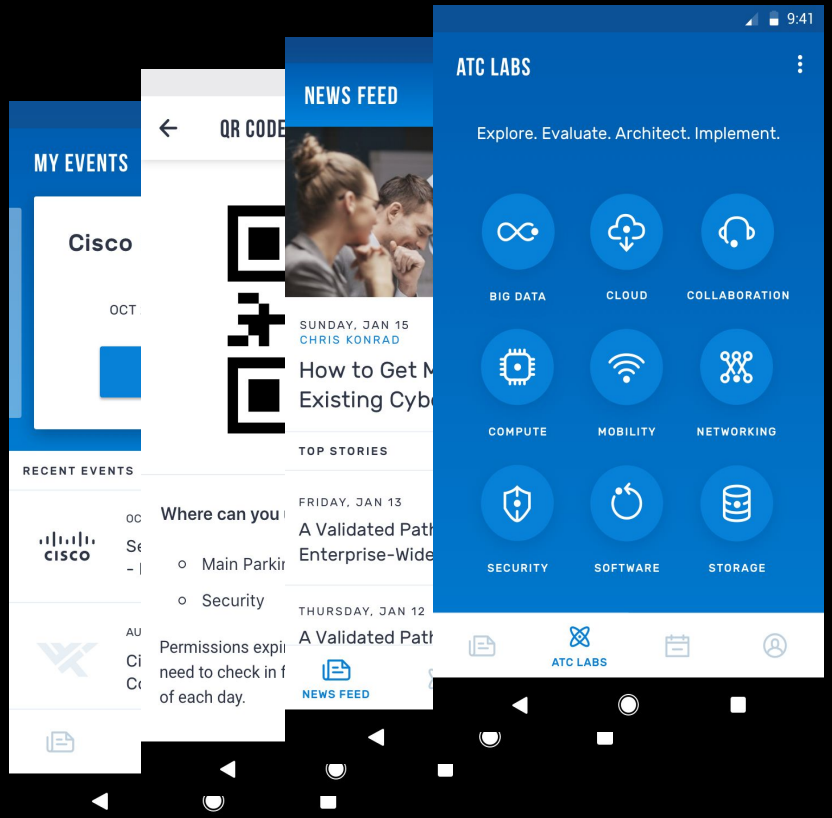
Enables functional-style, declarative development

Doesn’t push you toward functional programming

An excellent replacement for Java

ATC Connect

- Executive Briefings
  - Agenda
  - Attendees
  - Automatic check-in
  - GHQ access
- WWT News Article Feed
- ATC Practices
  - Labs
  - Partners
  - Capabilities



Patterns

Named & Default Parameters

- Synergy!
- Reduces need for builders

```
class AgendaItemPresenter(
   private val eventId: String,
   private val agendaItemId: String,
   ...
   private val mapsUrlBuilder:
      MapsUrlBuilder = MapsUrlBuilder()
) : EbcPresenter<AgendaItemView>() {
...
}
```

```
AgendaItemPresenter(
       eventId,
       agendaItemId,
       mapsUrlBuilder = MapsUrlBuilder()
)
```

Extension Methods

- Custom use cases for classes you don’t control
- Can replace Util classes
- Hard to unit test their use

```
fun Activity.requestLocationPermissions(requestCode: Int)
= ActivityCompat.requestPermissions(this,
        arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
        requestCode)
```

```
class ProximityHandler(...) {
  fun DateTime.sameDate(date: Date): Boolean {
    val now = DateTime(date)
    return (this.dayOfYear() == now.dayOfYear()) and
      (this.year() == now.year())
  }

  fun checkedInToday(checkInDtos: List<CheckInDto>)
    : Boolean {
    val today = DateTime.now()
    return checkInDtos.filter { today.sameDate(it.date) }
                      .isNotEmpty()
  }
}
```

Extension Methods

- Test helper methods

```
data class VisitDto(
        val id: String,
        ...
        val checkIns: List<CheckInDto>? = null
) {
    companion object
}
```

```
fun VisitDto.Companion.test(
        id: String = "id".appendRandom(),
        ...
        checkIns: List<CheckInDto>? = emptyList()
) = VisitDto(id, ..., checkIns)
```

```
val visit = VisitDto.test(id = "testId",
    checkIns = listOf(CheckInDto.test()))
```

UAT Robot pattern

- Screen “robot” hides action and assertion details
- Kotlin DSL syntax for specifying UAT steps

```
ProfileEditScreen.assertShowing()
ProfileStepScreen.setFirstName("Bob")
Espresso.pressBack()
ProfileEditScreen.clickYesonPrompt()
```

```
onProfileEdit {
   assertShowing()
   onProfileStep { setFirstName("Bob") }
   Espresso.pressBack()
   onPrompt { clickYes() }
}
```

Constructor Dependency Injection

- Inject via default parameters
- Reduces need for Dagger
- Initialization order is a challenge for UI tests

Constructor Dependency Injection

```
object EventRepositoryFactory {
   var eventRepository =
      EventRepository()
}

class EventRepository(
   private val userRepository: UserRepository =
      UserRepositoryFactory.userRepository,
   private val eventDataStorage: EventDataStorage = EventDataStorage(),
   private val sessionManager: SessionManager =
      SessionManagerFactory.sessionManager,
   private val requestFactory: RequestFactory = RequestFactory()
)
```

Kotlin Android Extensions

- Replace Butter Knife
- Autocasts!
- Autocompletes!

```
<EditText
    android:id="@+id/firstName"
    ... />
```

```
import
kotlinx.android.synthetic.main.fragment_profile_step.*

class ProfileStepFragment : ... {

    override fun setFirstName(
        name: String?
    ) {
        firstName.setText(name)
    }
}
```

Generics Instead of Class Params

- Concise
- Type safe
- No overhead

```
inline fun <reified T>
Context.intentFor() =
   Intent(this, T::class.java)

startActivity(
  intentFor<AgendaActivity>()
)

inline fun <reified T> assertType(
   value: Any?
) = assertTrue(
  value is T,
  "Expected type ${T::class.simpleName}
but was ${value?.javaClass?.simpleName}"
)
```

Pitfalls

Java Interoperability

- Watch out for nullability when using Java libraries!
- Synthesized properties don’t always match up

```
override fun onCreateView(
    inflater: LayoutInflater?,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View?
```

```
override fun setTitleDescription(titleDescription: String?) {
    this.roleDescription.setText(titleDescription)
}

override fun setPhon
    this.phone.setTe
}

override fun showLin
    showFailure("Acc
}

override fun onActiv
    when (requestCod
```

```
setText(resid: Int)
setText(resid: Int, type: TextView.Buffer
setText(text: CharSequence!, type: TextVie
setText(text: CharArray!, start: Int, len
text (from getText()/setText())
setTextColor(color: Int)
setTextAppearance(resId: Int)
setTextColor(colors: ColorStateList!)
setTextIsSelectable(selectable: Boolean)
setTextKeepState(text: CharSequence!)
```

Retrofit

- Gson converter bypasses initializers!

```
@Keep
data class NewsPostDto(
   val id: Int,
   val type: String,
   val url: String,
   val title: String,
   ...
) : Parcelable {

   val displayTitle: String by lazy {
StringEscapeUtils.unescapeHtml4(title)
   }

   val link: Uri get() = Uri.parse(url)
}
```

Mockito

- Easy to fill non-nullable types with `null`
- Use *anyOrNull()* just to be sure

```
interface AgendaView : RequestBaseView {
   fun setAgendaItems (agendaItems: List<AgendaItemDto>)
   val eventId: String
   ...
}
```

```
class AgendaItemRepository(...) {
   fun getAgendaItemsRequest (eventId: String)
     : Request<Set<AgendaItemDto>> {...}
}
```

```
class AgendaPresenterTest {
  @Test fun whenDataReturnedDisplayIt() {
    val view = mock<AgendaView>()
    val agendaListRequest = mock<Request<Set<AgendaItemDto>>>()
    val expectedAgendaItems = setOf(AgendaItemDto.test())
    val agendaItemRepository = mock<AgendaItemRepository>{
      on { getAgendaItemsRequest(any()) } doReturn agendaListRequest
    }
    AgendaPresenter(agendaItemRepository).attachView(view, true)

    argumentCaptor<(Set<AgendaItemDto>) -> Unit>().apply {
      verify(agendaListRequest).onDataReceived(capture())
      lastValue(expectedAgendaItems)
    }

    verify(view).setAgendaItems(expectedAgendaItems.toList())
  }
}
```

Language Peculiarities

- Data class `copy()` is shallow
- Non-local `return` from lambdas

```
val presenter = UserDto(0, profile =
   ProfileDto(pictureUrl = null)).copy()
```

```
inline fun <T, R> T.inlineLet(
   block: (T) -> R
): R = block(this)

fun <T, R> T.myLet(block: (T) -> R): R
       = block(this)

fun returnFromLet(
request: IRequest<CurrentUserEditModel>?
) : IRequest<CurrentUserEditModel>? {
   request?.inlineLet { return it }
   request?.myLet { return it }
   return request
}
```

Readability

- People can't always infer types
- Expect `val` to be immutable

```
var eventsRequestObservables =
  createEventsRequestObservables()
```

```
val session: Session
  get() {
    return Session(
      preferences.getString(ACCESS_TOKEN,
          ""))
   }
```

Tool Support

- Debugger can get confused
- Incremental compilation is suspect
- Refactoring support disparity compared to Java
- Auto-conversion from Java to Kotlin

Smells

Kotlin Style

Don't	Do
`nullable!!` `if (nullable != null) { }`	`nullable?.let { }`
`if () {} else if {} else if{}...`	`when (it) { }`
`fun getFoo(): Foo { }`	`val foo: Foo = ...`

`val something = Thing()`
`something.a()`
`something.b()`
`return something`
	`return Thing().apply {`
	`  a()`
	`  b()`
	`}`

Kotlin Style

Don’t	Do
`val value: Int` `  get() = Random().nextInt()`	`fun getValue(): Int = Random().nextInt()`

Kotlin Rough Edges

- No *checkstyle* yet
- Extension methods can be hard to discover
- Generic parameter syntax is inconsistent
- Bitwise `or` reads poorly on flags
- Secondary constructors must initialize properties in args

```
class PracticeDetailsPresenter :
EbcPresenter<PracticeDetailsView>()
```

```
fun <T : Activity> rotateScreen(
   activityRule: ActivityTestRule<T>
)
```

```
wwtNotification.setDefaults(
   Notification.DEFAULT_SOUND or
   Notification.DEFAULT_VIBRATE
)
```

```
constructor(source: Parcel) : this(
   source.readString(),
   source.readString()
)
```

Odds and Ends

Object Delegates

- Haven’t found a valid use
- Android APIs don’t help
- Problem use cases:
  - Delegate needs reference to delegator
  - Delegator needs to call methods on delegate that aren’t part of the interface

Generics: Variance

- Analogue for Java wildcard types
- Type-safe support for
  - Producing objects (**`out T`**)
  - Consuming objects (**`in T`**)

```
interface GenericProducer <out T> {
    fun create(): T
}

interface GenericConsumer <in T> {
    fun doSomething(value: T)
}
```

Filenames and Packages

- No need to match contents
- Any combination of classes and/or functions in a file
- Limits refactoring in Android Studio

Thanks!