

# KOTLIN

## LESS CODE, MORE FUN



Speaker notes

+



# Central Wisconsin CONFERENCE



Speaker notes

+



# SKYLINE

Speaker notes

- ▶ I work here!



Speaker notes

+



Speaker notes

+



Microsoft®  
.NET

Speaker notes

+



Speaker notes

+



# WHAT IS...



# Kotlin

## Speaker notes

- ▶ First unveiled in 2011 (after being in dev for a year)
- ▶ Open-sourced in 2012
- ▶ Version 1.0 released on February 15th, 2016
- ▶ JetBrains said most languages were lacking the features they wanted
  - ▶ Except Scala, but it was slow to compile
- ▶ JetBrains hopes to drive IntelliJ sales

# "STATICALLY TYPED PROGRAMMING LANGUAGE FOR MODERN MULTIPLATFORM APPLICATIONS"



# STATICALLY TYPED

```
val numbers: List<Int> = listOf(1, 4, 7, 10, 23)
```

```
enum class Sport {  
    Baseball,  
    Football,  
    Soccer,  
    Basketball  
}
```

```
data class Team(val city: String, val teamName: String, val sport: Spo
```

## Speaker notes

- ▶ Statically typed:
  - ▶ Know type of variable at compile time
    - ▶ Java
    - ▶ C, C++, C#
  - ▶ Kotlin uses type inference
    - ▶ Type system can figure the type out
    - ▶ Don't always have to specify
    - ▶ Will warn if not clear
- ▶ Modern:
  - ▶ Concise code
  - ▶ Lots of features

# MULTI-PLATFORM

Build Applications For



## Speaker notes

- ▶ First-class language on Android
- ▶ Interoperable with JVM
- ▶ Transpile Kotlin into JavaScript
- ▶ Compile native code (without VM)
  - ▶ Windows
  - ▶ Linux
  - ▶ MacOS
  - ▶ iOS
  - ▶ Android
  - ▶ WebAssembly

# 100% JAVA INTEROPERABILITY

```
import io.reactivex.Flowable
import io.reactivex.schedulers.Schedulers

Flowable
    .fromCallable {
        Thread.sleep(1000) // imitate expensive computation
        "Done"
    }
    .subscribeOn(Schedulers.io())
    .observeOn(Schedulers.single())
    .subscribe(::println, Throwable::printStackTrace)
```

## Speaker notes

- ▶ Call existing Java libraries
- ▶ Use Java classes
- ▶ Can call Kotlin classes from Java
- ▶ May need a bit of tweaking

# FIRST-CLASS LANGUAGE ON ANDROID



```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
    setSupportActionBar(toolbar)  
    ...
```

## Speaker notes

- ▶ Official language on Android
- ▶ Announced at Google I/O 2017 (May 2017)
- ▶ Option to build apps in Kotlin right away
- ▶ Only adds about 100K to the APK size

# Why Kotlin?



## Concise

Drastically reduce the amount of boilerplate code.

[See example](#)



## Safe

Avoid entire classes of errors such as null pointer exceptions.

[See example](#)



## Interoperable

Leverage existing libraries for the JVM, Android, and the browser.

[See example](#)



## Tool-friendly

Choose any Java IDE or build from the command line.

[See example](#)

Speaker notes

+

# VAL AND VAR

```
// Java  
final String name = "Joe Smith"
```

```
// Kotlin  
val name: String = "Joe Smith"
```

```
// Java  
int age = 23
```

```
// Kotlin  
var age: Int = 23
```

Speaker notes

+

# DATA CLASSES

```
// Java
public class Person {
    private final String first;
    private final String last;
    private int age;

    public Person(String first, String last, int age) {
        this.first = first;
        this.last = last;
        this.age = age;
    }

    public String getFirst() {
        return first;
    }
}
```

## Speaker notes

- ▶ Compiler automatically creates:
  - ▶ equals() and hashCode()
  - ▶ toString() -> Person(first=Michael, last=Fazio, age=33)
  - ▶ componentN() functions in order of declaration
    - ▶ e.g. val first = person.component1(), val age = person.component3()
- ▶ Cannot be abstract, open, sealed, or inner

# DATA CLASSES

## COMPONENT FUNCTIONS, DESTRUCTURING DECLARATIONS, COPY

```
data class Person(val first: String, val last: String, var age: Int)
```

```
val hazel = Person(first = "Hazel", last = "Fazio", age = 4)
println(hazel) // Person(first=Hazel, last=Fazio, age=4)

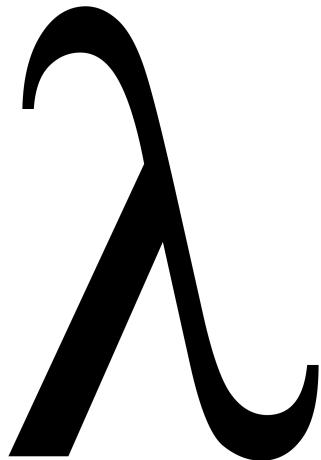
val (first, last, age) = hazel
println("$first $last, $age years old.") // Hazel Fazio, 4 years old.

val hazelNY = hazel.copy(age = 5) // Next year
```

### Speaker notes

- ▶ componentN() functions in order of declaration (created automatically)
  - ▶ e.g. val first = person.component1(), val age = person.component3()
- ▶ Destructuring Declarations
  - ▶ Helped by component functions
- ▶ copy() can be used to create a new instance of the type
  - ▶ person.copy(age = 34)
- ▶ String interpolation

# LAMBDA FUNCTIONS



```
val sum = { x: Int, y: Int -> x + y }
```

```
val baseballTeams: List<Team> =
    teams.filter { team -> team.sport == Sport.Baseball}
```

```
strings
    .filter { it.length == 5 }
    .sortedBy { it }
```

Speaker notes

+

# SMART CASTS

```
// Java
private int calculateLength(Object o) {
    if(o instanceof String) {
        final String s = (String) o;
        return s.length();
    }
    return 0;
}
```

```
// C#
private int CalculateLength(object o) {
    if(o is string s) return s.Length;
    return 0;
}
```

```
// Kotlin
fun calculateLength(v: Any) : Int {
    if(v is String) return v.length //v is automatically cast to a Str
```

Speaker notes

+

# RANGES

```
// Java
private boolean isInRange(int i) {
    return i >= 1 && i<= 10;
}
```

```
// Kotlin
fun isInRange(i: Int) : Boolean {
    return i in 1..10
}
```

```
// Kotlin
fun isInRange(i: Int) = i in 1..10
```

## Speaker notes

- ▶ rangeTo() +
- ▶ downTo()
- ▶ reversed()
  - ▶ Last to first with negative step
- ▶ step()
  - ▶ (1..12 step 3).last == 10
- ▶ Single-expression function in last example

# WHEN EXPRESSIONS

```
when (x) {  
    1 -> print("x == 1")  
    2 -> print("x == 2")  
    else -> { // Note the block  
        print("x is neither 1 nor 2")  
    }  
}
```

```
val result: String = when (x) {  
    0, 1 -> "x == 0 or x == 1"  
    else -> "otherwise"  
}
```

```
when (x) {  
    in 1..10 -> print("x is in the range")  
    in validNumbers -> print("x is valid")  
    !in 10..20 -> print("x is outside the range")  
    else -> print("none of the above")  
}
```

Speaker notes

- ▶ Replaces switch

# TRaversing MAPS

```
// Java
public void printTeams(Map<String, Team> teamMap) {
    for(Map.Entry<String, Team> entry : teamMap.entrySet()) {
        System.out.println(entry.getKey() + " - " + entry.getValue());
    }
}
```

```
// Kotlin
fun printTeams(teamMap: Map<String, Team>) {
    for ((key, value) in teamMap) {
        println("$key - $value")
    }
}
```

Speaker notes

- ▶ Deconstructed declaration of the entry

# NULL

## THE BILLION-DOLLAR MISTAKE

```
var notNullString: String = "This can't be null"  
  
var nullableString: String? = "This could be null"
```

### Speaker notes

- ▶ Tony Hoare
- ▶ Created null reference in 1965, causes lots of issues

# NULL SAFETY

```
var a: String = "abc"  
a = null // compilation error
```

```
var b: String? = "abc"  
b = null // ok
```

```
val l = a.length
```

```
val l = b.length // error: variable 'b' can be null
```

Speaker notes

- ▶ Nulls are checked at runtime

# NULL SCENARIOS

- ▶ `throw NullPointerException();`
- ▶ Usage of the `!!` operator
- ▶ Data inconsistency related to initialization
  - ▶ An uninitialized `this` available in a constructor and is used somewhere
  - ▶ A superclass constructor calls an open member whose implementation uses uninitialized state
- ▶ Java interop

## Speaker notes

- ▶ `!!` == Not-null assertion operator
  - ▶ Converts any value to a non-null type or throws an exception if null
  - ▶ "[!!]" is for NPE-lovers"
- ▶ Incorrect nullability
  - ▶ Add null to MutableList instead of MutableList

# NULL SAFETY

## SAFE CALLS

```
val brewers: Team? = Team("Milwaukee", "Brewers", Sport.Baseball)
val rockies: Team? = Team("Colorado", "Rockies", Sport.Baseball)
val devilRays: Team? = null

val baseballTeams = listOf(brewers, rockies, devilRays)
```

```
val nicknames: List<String> = baseballTeams.map {
    if (it != null) it.name else ""
}
```

```
val nicknames: List<String> = baseballTeams.map {
    it?.name ?: ""
}
```

Speaker notes

+

# ELVIS OPERATOR

```
// Java
public Sport getTeamSport(Team team) {
    if(team != null) {
        return team.getSport();
    }

    return Sport.Unknown;
}
```

```
// Kotlin
fun getTeamSport(team: Team?): Sport
    = if(team != null) team.sport else Sport.Unknown

fun getTeamSport(team: Team?): Sport = team?.sport ?: Sport.Unknown
```

## Speaker notes

- ▶ Same as ?? in C#
- ▶ "Null coalescing operator"

# ELVIS OPERATOR



Why ?: is called the  
**Elvis operator**

Speaker notes

+

# NULL SAFE SCOPING

```
fun findTeam(city: String = "", nickname: String = ""): Team? {  
    return this.teams.firstOrNull {  
        team -> team?.city == city || team?.name == nickname  
    }  
}  
  
findTeam(nickname = "Brewers")?.let {  
    //This will only run if findTeam(...) returns a non-null value  
}
```

Speaker notes

+

# NULL SAFE SCOPING

```
fun listTeams() {  
    val brewers: Team? = Team("Milwaukee", "Brewers", Sport.Baseball)  
    val rockies: Team? = Team("Colorado", "Rockies", Sport.Baseball)  
    val devilRays: Team? = null  
  
    println(this.getSummary(brewers))  
    println(this.getSummary(rockies))  
    println(this.getSummary(devilRays))  
}
```

```
private fun getSummary(team: Team?): String {  
    var fullName: String = "N/A"  
  
    team?.let({t -> fullName = "${t.city} ${t.name} (${t.sport})"})  
  
    return fullName  
}
```

Speaker notes

+

# NULL SAFE SCOPING

```
private fun getSummary(team: Team?): String {  
    return team?.let({ t -> "${t.city} ${t.name} (${t.sport})" }) ?: "  
}
```

```
private fun getSummary(team: Team?): String {  
    return team?.let { t -> "${t.city} ${t.name} (${t.sport})" } ?: "N  
A"  
}
```

```
private fun getSummary(team: Team?): String {  
    return team?.let { "${it.city} ${it.name} (${it.sport})" } ?: "N/A"  
}
```

```
private fun getSummary(team: Team?): String =  
    team?.let { "${it.city} ${it.name} (${it.sport})" } ?: "N/A"
```

Speaker notes

+

# JAVA HAS THESE, KOTLIN DOES NOT

- ▶ Ternary-operator (a ? b : c)
- ▶ Checked exceptions
- ▶ Primitive types that are not classes
- ▶ Static members
- ▶ Non-private fields

## Speaker notes

- ▶ Checked exceptions lead to useless try...catch blocks
- ▶ Instead of static members, use package-level functions
  - ▶ Also can use object declarations or companion objects
- ▶ Use properties instead of fields
- ▶ if works just fine
  - ▶ `val max = if (a < b) b else a`

# INSTEAD OF STATIC

```
package org.faziodev.kotlin

// Package-level function
fun nullSafeToUpper(value: String?) : String {
    return value?.toUpperCase() ?: ""
}

// Object declaration
object TeamSource {
    fun myFunction() { ... }
}
```

```
// Import the package and call the method
val upper: String = nullSafeToUpper(brewers?.name)

// Reference the function similarly to a static method
TeamSource.myFunction()
```

## Speaker notes

- ▶ Package-level functions
- ▶ Object declarations
  - ▶ Work as singletons
- ▶ Companion objects

# COMPANION OBJECTS

```
data class Team(val city: String, val name: String, var sport: Sport)
    companion object {
        fun createBaseballTeam(city: String, name: String)
            = Team(city, name, Sport.Baseball)
    }
}
```

```
val brewers = Team.createBaseballTeam("Milwaukee", "Brewers")
```

Speaker notes

+

# ALL CLASSES ARE CLOSED

```
class Base(p: Int)  
  
//This will not work  
class Derived(p: Int) : Base(p)
```

```
open class Base(p: Int)  
  
class Derived(p: Int) : Base(p)
```

## Speaker notes

- ▶ open in Kotlin is opposite of final in Java
- ▶ Design and document for inheritance or else prohibit it.
  - ▶ Effective Java
- ▶ Things are explicit in Kotlin
- ▶ Exception: Abstract classes are open by default
- ▶ Classes are also public by default

# ALL FUNCTIONS ARE CLOSED

```
open class Base {  
    open fun openFun() {}  
    open fun alsoOpenFun() {}  
    fun notOpenFun() {}  
}
```

```
class Derived() : Base() {  
    //Overridden methods are open by default  
    override fun openFun() {}  
  
    // This can no longer be overridden  
    final override fun alsoOpenFun() {}  
}
```

Speaker notes

+

# COLLECTIONS ARE IMMUTABLE

```
val teams: List<Team> = this.getBaseballTeams(this.allTeams)

val brewers: Team = Team("Milwaukee", "Brewers", Sport.Baseball)

teams.add(brewers) //add doesn't exist on List
```

```
val teams: MutableList<Team> = this.getBaseballTeams(this.allTeams)

val brewers: Team = Team("Milwaukee", "Brewers", Sport.Baseball)

teams.add(brewers) //Brewers added
```

```
val teams: List<Team> = listOf(brewers, reds, cubs)

val mutableTeams: MutableList<Team> = mutableListOf(brewers, reds, cub
```

Speaker notes

+

# KOTLIN ON ANDROID



## Speaker notes

- ▶ Official language for Android
- ▶ Available in Android Studio by default +

# KOTLIN ON ANDROID

## ONCLICKHANDLERS

```
// Java  
loginButton.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        logInUser();  
    }  
});
```

```
// Kotlin  
loginButton.setOnClickListener {v -> logInUser()}  
  
loginButton.setOnClickListener { logInUser() }
```

Speaker notes

+

# KOTLIN ON ANDROID

## LATEINIT

```
private lateinit var logo: ImageView
```

```
this.logo = this.findViewById<ImageView>(R.id.logo)
```

Speaker notes

+

# KOTLIN ON ANDROID

## ANKO

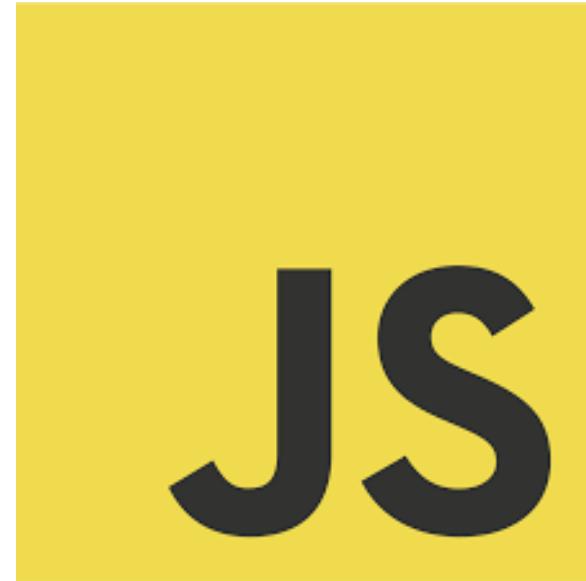
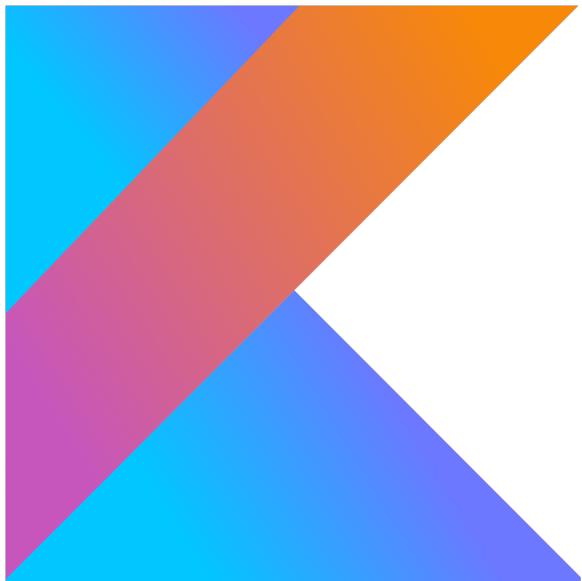
- ▶ Anko Commons
- ▶ Anko Layouts
- ▶ Anko SQLite



### Speaker notes

- ▶ Speed up Android dev
- ▶ Anko Commons
  - ▶ Intents
  - ▶ Dialogs/toasts
  - ▶ Logging
  - ▶ Resources/Dimensions
- ▶ Anko Layouts
  - ▶ DSL (Domain-specific language) for Android layouts
- ▶ Anko SQLite
  - ▶ Simplifies working with SQLite DBs
- ▶ Anko Coroutines
  - ▶ Helps with Kotlin coroutines (async calls)

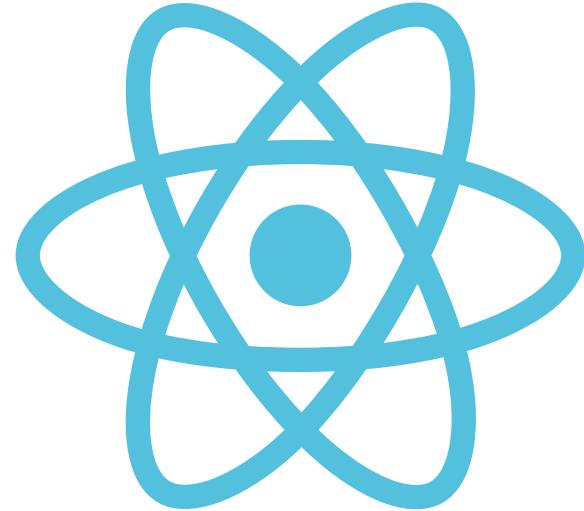
# KOTLIN AND JAVASCRIPT



Speaker notes

- ▶ Converts into ES

# KOTLIN AND REACT



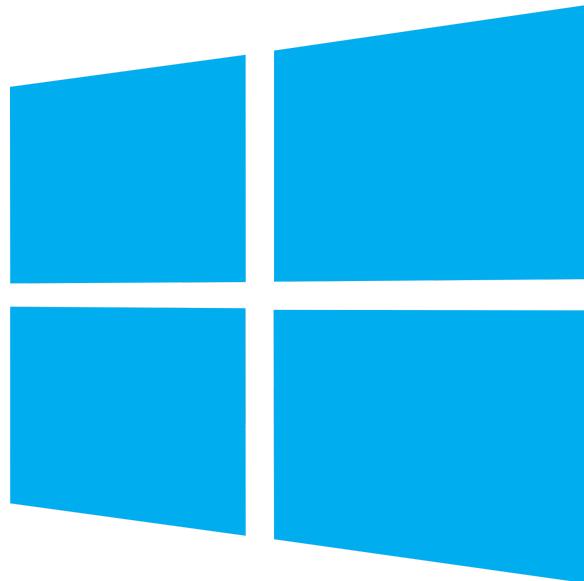
<https://github.com/JetBrains/create-react-kotlin-app>

## Speaker notes

- ▶ Transpiles to ECMAScript 5.1
  - ▶ ES2015 in the future
- ▶ Works with 3rd party libraries
- ▶ Strong typing available from TypeScript

# KOTLIN NATIVE

## DESKTOP



### Speaker notes

- ▶ Compiles to Native code
- ▶ Currently in development

# KOTLIN NATIVE

## IOS



Speaker notes

+

# KOTLIN NATIVE

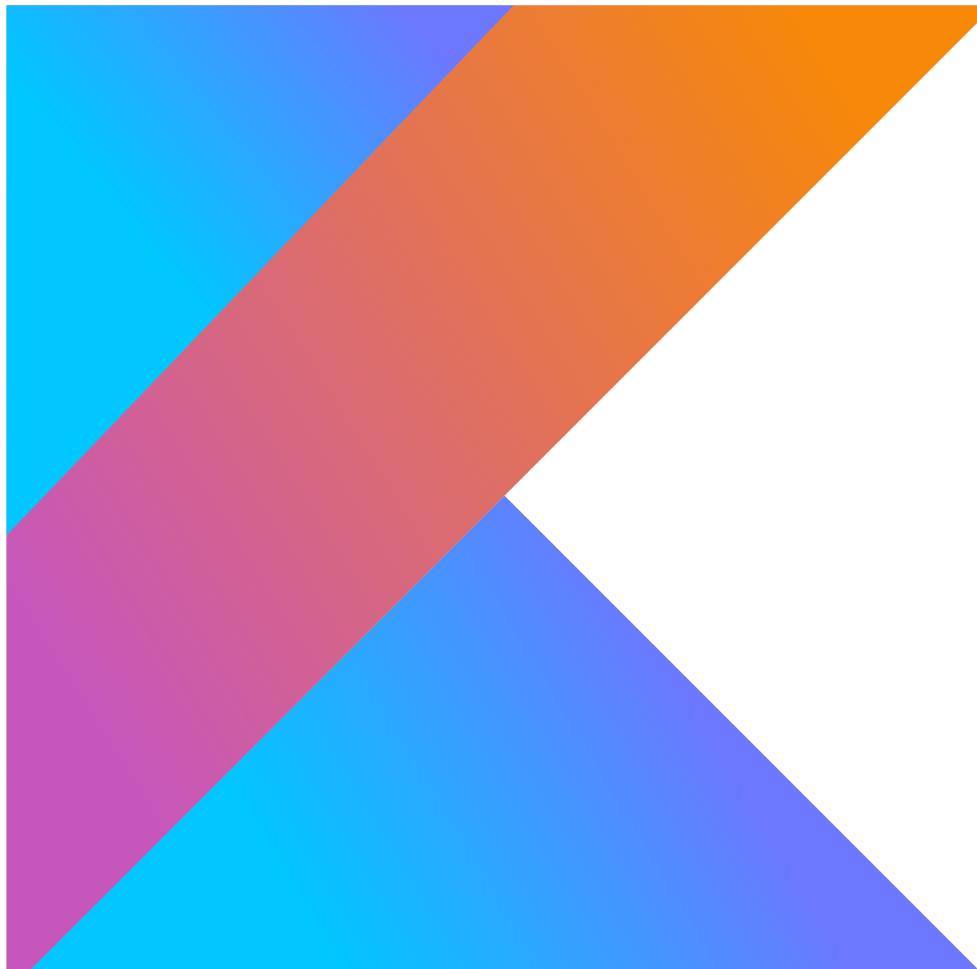
## ANDROID



Speaker notes

+

# QUESTIONS?



Speaker notes

+

# THANK YOU!



# Kotlin

@faziodev

Speaker notes

+