Developing Android Applications with Kotlin: The Big Picture

Introducing Android App Development with Kotlin

Markus Neuhoff
Android Developer

Overview

- Why develop native Android apps?
- What is Kotlin?
- Benefits and disadvantages of Kotlin
- Android Studio
- Available Android APIs

Course Prerequisites

Don’t need

Previous android development experience

Need

Object oriented language

Web/Mobile fundamentals

User-first mindset

Course Goals

Decide if native Android app development is the right choice

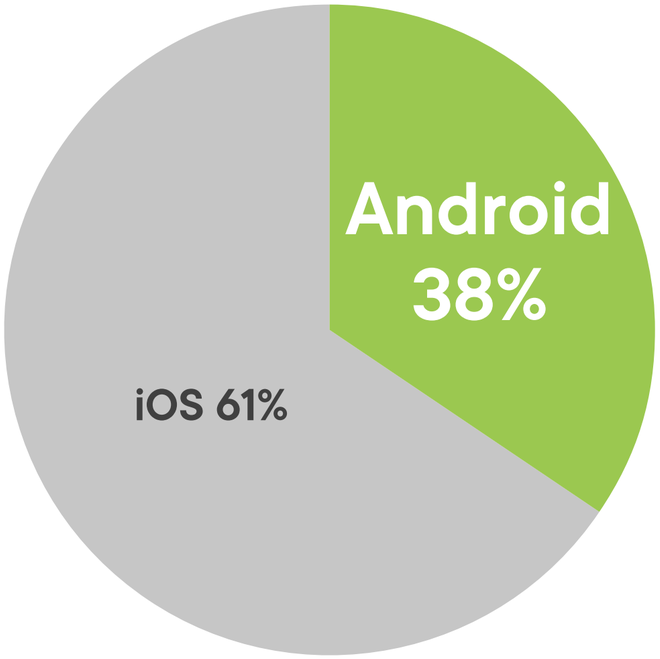
Understand the benefits and drawbacks of Kotlin

Become familiar with Android development with Kotlin

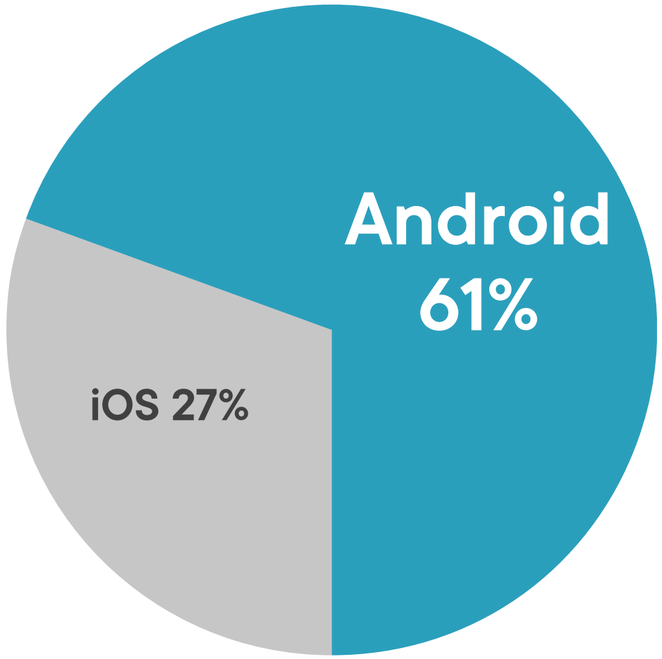
Why Native Android Development?

Mobile OS Market Share

USA

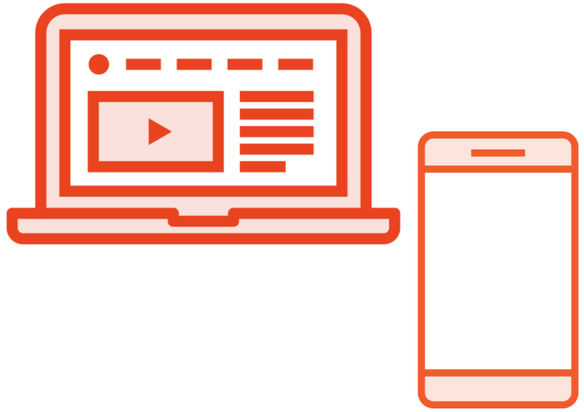


Global

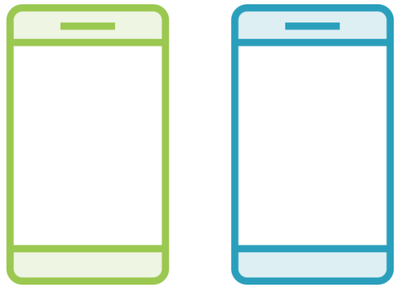


Data source: Statcounter

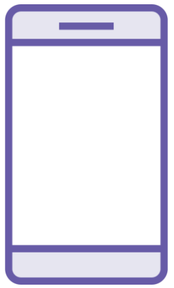
Android Development Approaches



Responsive Web



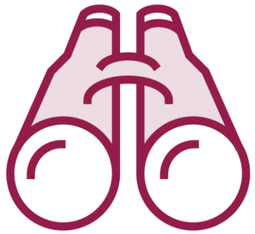
Hybrid



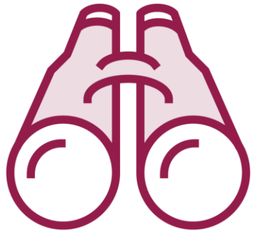
Native

What is Kotlin?

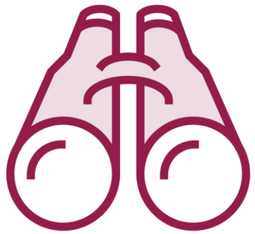
Kotlin Overview



Its own language



Fully interoperable with Java



Object Oriented and functional

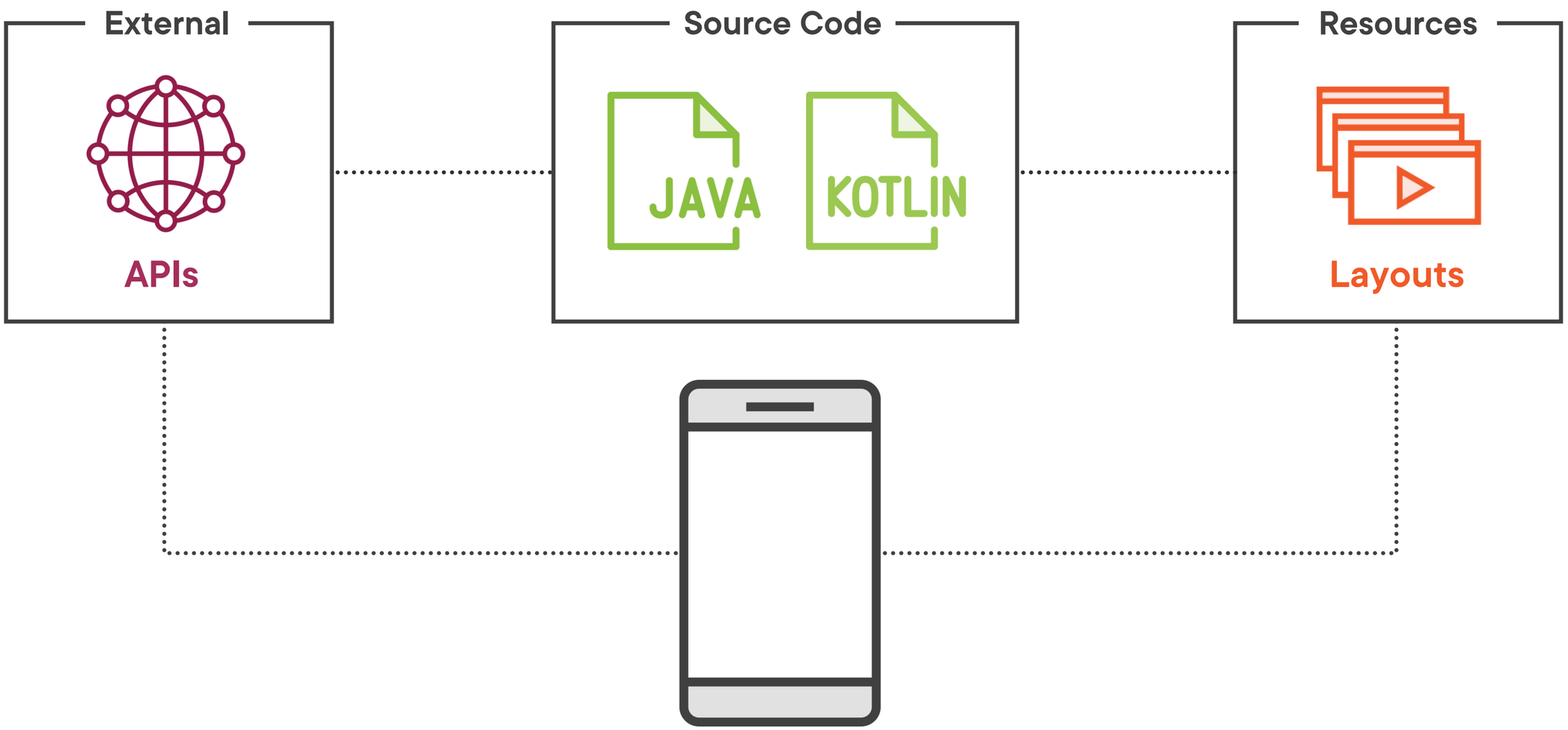
Benefits of Kotlin

3

- Conciseness
- Improved Null Safety
- Functional Programming

Benefit 1: Conciseness

Android Architecture



Changing Text Views

MainActivity.java

```
public void
onCreate(savedInstanceState: Bundle?) {

  super.onCreate(savedInstanceState)

  setContentView(R.layout.main)

  TextView welcome = (TextView)
    findViewById(R.id.welcome);

  welcome.setText(“Hello ” +
    user.first);
```

MainActivity.kt

```
private lateinit var binding: MainBinding

override fun onCreate(savedInstanceState:
Bundle?) {

  super.onCreate(savedInstanceState)

  binding =
    MainBinding.inflate(layoutInflater)

  val view = binding.root

  setContentView(view)

  binding.welcome.text = “Hello
    ${user.first}”
```

Kotlin can reduce lines
of code by 40%.

Benefit 2: Improved Null Safety

Null Pointer Exception Example

Java

JavaUser.java

```
private String First;

private String Last;

public JavaUser(String first, String
last) {

 this.First = first;

 this.Last = last;

}

public String getName(){

 return First.concat(" ").concat(Last);
```

NullUserTest.kt

```
@Test

fun testNullJava() {

 val user = JavaUser("Alice", null)

 Assert.assertNotNull(user.name)

}
```

```
Output:

java.lang.NullPointerException
   at java.lang.String.concat
   at com.psdemo.demoapp.JavaUser.getName
   ...
```

Null Pointer Exception Example

Kotlin

KotlinUser.kt

```
data class KotlinUser(var first:
String, var last: String) {

 fun getName(): String = "$first $last"

}
```

NullUserTest.kt

```
@Test

fun testNullJava() {

 val user = JavaUser("Alice", null)

 Assert.assertNotNull(user.name)

}
```

⚠ Null can not be a value of a non-null type String

```
data class User(var first: String?, var
last: String?)

val l = if (Last != null) last else “”
```

- Declare variables as nullable with ?
- Check for nulls explicitly

```
val l = Last ?: “”
```

- Elvis Operator

```
data class User(var first: String?, var
last: String?, var friend: User?)

val friend = user?.friend

val friendName = user?.friend?.getName()
```

- Safe calls
- Returns null if user is null
- Works on chained objects too!

```
val friend = user!!.friend
```

- Null pointer exceptions with !!

Benefit 3: Functional Programming

Object Oriented Programming

```
class User {
    private String First;
    private String Last;
    public void setFirst(String first) {
        First = first;
    }
    public String getName(){
        return First.concat(" ").concat(Last);
    }
}
```

Functional Programming

```
fun getName(first: String, last: String): String {
    val name = first.plus(" ").plus(last)
    return name
}
```

Extension Functions

```
val distance = 2.3

public double getMiles(double distance){
    return distance / 1609f;
}

fun Double.toMiles() = this / 1609f
```

Lambda Functions

```
val users = arrayOf(User(“Bob”, ”red”), User(“Jill”, “red”), User(“Amir”, ”yellow”),
                    User(“Noel”, “blue”), User(“Quinten”, “yellow”))
users.forEach{
    if(it.color=="red") println(it.name)
}
```

Kotlin Tradeoffs and Considerations

Native Android Development

Multiple Languages/Platforms

Kotlin Learning Curve

Missing Java Features

Checked Exceptions

```
File() throws FileNotFoundException
```

Decreases code readability

Increases likelihood of empty catches

@Throws is available if needed

Ternary Operator

```
a == 4 ? b = 3 : b = 5;
```

Valuable functionality, hard to read

```
b = if (a == 4) 3 else 5
```

Summary

Importance of Android native development

Kotlin Benefits

- **Conciseness**
- **Null safety**
- **Functional programming**

Kotlin downsides

Up Next:
Essential Tools and Functionality