

Kotlin e Android Studio



Why Android **W**Kotlin







Safety



Interoperability

Kotlin on Android in 2020

50% More likely to be very satisfied 60% Pro Android developers use Kotlin 70%+ Top 1k apps contain Kotlin code

Expressiveness Kotlin

```
class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    updateTextButton.setOnClickListener {
     val a = Artist(1, "Vasco Rossi", "vasco.com", "vscrss")
     welcomeTextView.text = a.toString()
    }
}
```

```
data class Artist (
var id: Long,
var name: String,
var url: String,
var mbid: String)
```

compiler

Automatically generated functions for data class in Kotlin

- 1. equals()
- 2. hashCode()
- 3. toString()
- 4. copy()
- 5. componentN()

Expressiveness

Java

```
public class MainActivity extends AppCompatActivity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
  final Button button = findViewById(R.id.updateTextButton);
   final TextView welcomeTextView = findViewById(R.id.welcomeTextView);
   button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
      Artist a = new Artist(1, "Vasco Rossi", "vasco.com", "vscrss");
      welcomeTextView.setText(a.toString());
  });
```

```
public class Artist {
 private longid;
 private String name;
 private String url;
 private String mbid;
 public Artist(long id, String name, String url, String mbid){
  this.id = id;
  this.name = name;
  this.url = url;
  this.mbid = mbid;
 public longgetId() {
  return id;
 public void setId(long id) {
  this.id = id;
 public String getName(){
 return name;
 public void s etName(String name) {
 this.name = name;
 public String getUrl() {
 return url;
 public void setUrl(String url) {
 this.url = url;
 public String getMbid(){
 return mbid;
 public void setMbid(String mbid) {
 this.mbid = mbid:
 @Override public String to String() {
  return "Artist{" +
      "id=" + id +
       ". name='" + name + '\'' +
      ", url="" + url + '\" +
      ", mbid="" + mbid + '\" +
```

Null Safety in Kotlin



```
// This won't compile. Artist can't be null
var notNullArtist: Artist = null
// Artist can be null
var artist: Artist? = null
// Won't compile, artist could be null and we need to deal with that
artist.name
// Will access name only if artist != null
artist?.name
// Smart cast. We don't need to use safe call operator if we previously
// checked nullity
if (artist != null) {
  artist.name
// Only use it when we are sure it's not null. Will throw an exception otherwise.
artist!!.name
// Use Elvis operator to give an alternative in case the object is null.
val name = artist?.name ?: "empty"
// equivalents to
val name = if (artist != null) artist.name else "empty"
```

Null in Java

```
// This will compile. Artist can be null
Artist artist = null;
// java.lang.NullPointerException: Attempt to invoke a virtual method on a null object reference
artist.toString();
// Smart cast. We previously checked nullity
if (artist != null) {
 artist.toString();
// Use Elvis operator to give an alternative in case the object is null.
String name = artist != null ? artist.getName() : "empty";
```

Extension functions

```
java.lang.Object

Jandroid.content.Context

Jandroid.content.ContextWrapper

Jandroid.view.ContextThemeWrapper

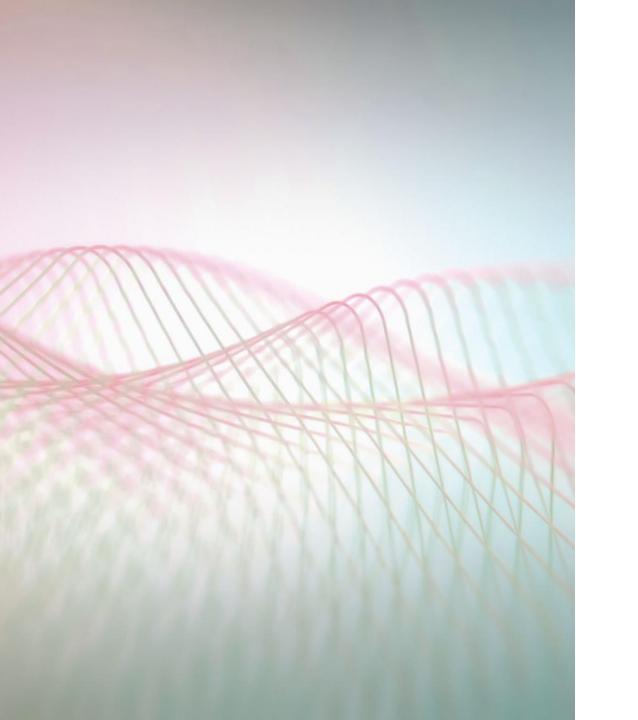
Jandroid.app.Activity

Jandroidx.activity.ComponentActivity
```

```
    androidx.fragment.app.FragmentActivity
    androidx.appcompat.app.AppCompatActivity
```

```
class MainActivity : AppCompatActivity() {
 fun Context.toast(message: CharSequence, duration: Int = Toast.LENGTH_SHORT) {
   Toast.makeText(applicationContext, message, duration).show()
 override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)
   updateTextButton.setOnClickListener{
     val a = Artist(1, "Vasco Rossi", "vasco.com", "vscrss")
      welcomeTextView.text = a.toString()
      toast("This is a Toast!")
```





Kotlin
Interfaces &
Classes



Interfaces

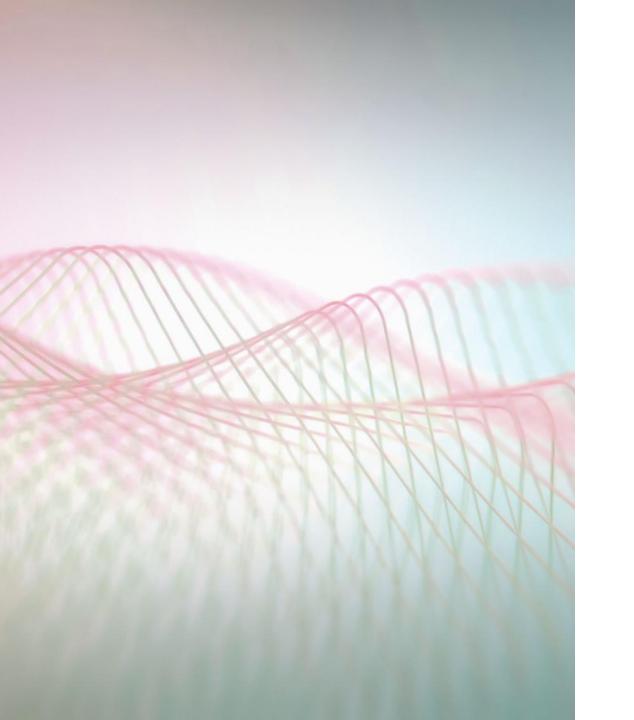
- can contain
 - declarations of abstract methods
 - method implementations
 - property that provide accessor implementation.
- Interfaces cannot
 - store state (unlike abstract classes)

```
interface Animal {
 val age: Int // abstract property
 // property with implementation
 val name: String
   get() = "foo"
 fun bar() // abstract method
 // method with implementation
 fun foo() {
   print(age)
class Dog : Animal {
 override val age: Int = 4
```

Classes

```
class UniClass constructor(var name: String /*prop*/, age: Int /*param*/) // class header
{ // class body
 // property (data member)
 private var age = age
 private var grade = 0.0
 constructor(name: String /*prop*/, age: Int /*param*/, grade: Double) : this(name, age){
   this.grade = grade
 init { //The init block will execute immediately after the primary constructor
   println("First initializer block that prints ${name}")
   name = name.toUpperCase()
 // member function
 fun printMe() {
   print("You are at the University of Insubria, course-$name - $grade - $age")
fun main() {
 val obj = UniClass("pdm-21", 25) // create obj object
 obj.printMe()
```

- In Kotlin, class declaration consists of
 - a class header and
 - a class body surrounded by curly braces,
 - similar to Java.
- A class can have a primary constructor and one or more secondary constructors.
- The primary constructor is part of the class header



Kotlin functional programming



Higher order functions

```
fun calculate(x: Int, y: Int, operation: (Int, Int) -> Int): Int { // higher-order function
    return operation(x, y) // operation invocation
}

fun sum(x: Int, y: Int) = x + y // a function that matches the operation signature

fun main() {
    val sumResult = calculate(4, 5, ::sum) // Invokes the higher-order function
    val mulResult = calculate(4, 5) { a, b -> a * b } // Invokes the higher-order fun
    println("sumResult $sumResult, mulResult $mulResult")
}
```

- Invokes the higher-order function passing in two integer values and the function argument :: sum.
- Invokes the higher-order function passing in a lambda as a function argument.

Lambda

val lambdaName : Type = { argumentList -> codeBody }

- Some examples of lambda functions that transform a string to uppercase.
- So they are all examples of String to String function

```
val upperCase1: (String) -> String = { str: String -> str.toUpperCase() }
val upperCase2: (String) -> String = { str -> str.toUpperCase() }
val upperCase3 = { str: String -> str.toUpperCase() }
// val upperCase4 = { str -> str.toUpperCase() } // compile error
val upperCase5: (String) -> String = { it.toUpperCase() }
val upperCase6: (String) -> String = String::toUpperCase
println(upperCase1("hello"))
println(upperCase2("hello"))
println(upperCase3("hello"))
println(upperCase5("hello"))
println(upperCase6("hello"))
```

Returning from a Lambda

 The final expression is the value that will be returned after a lambda is executed:

```
val calculateGrade = { grade : Int ->
  when(grade) {
    in 0..40 -> "Fail"
    in 41..70 -> "Pass"
    in 71..100 -> "Distinction"
    else -> false
  }
}
val g = calculateGrade(50)
println(g)
```

Operator overloading

• When you use operator in Kotlin, it's corresponding member function is called.

operator fun plus(other: Int): Int

```
val a = 5
val b = 10

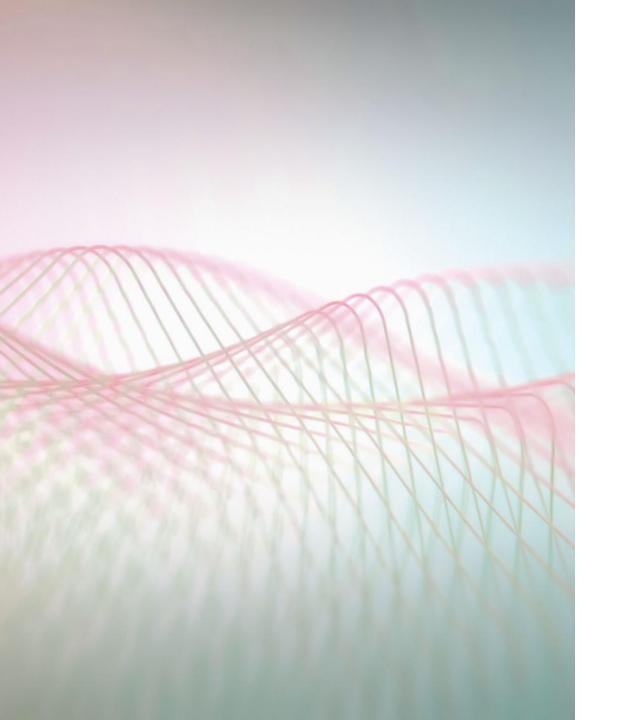
println(a.plus(b))
println(a+b)
```

```
fun main(args: Array<String>) {
  val p1 = Point(3, -8)
  val p2 = Point(2, 9)

  var sum: Point = p1 + p2

  println("sum = (${sum.x}, ${sum.y})")
}
```

```
class Point(val x: Int = 0, val y: Int = 10) {
    // overloading plus function
    operator fun plus(p: Point) : Point {
      return Point(x + p.x, y + p.y)
    }
}
```



Kotlin
Collections &
Generics

