Describing Types with Kotlin



Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT
@hedgehogjim blog.jwhh.com



What to Expect from This Module



Using Kotlin with Android

Basic types and variable declarations

Defining types

Properties

Primary constructor

Functions and passing parameters

Type Initialization



Kotlin

Coding Efficiency

Concise syntax

Avoids boilerplate code

Error Reduction

Null safety

Ability to indicate intentions

Compatibility

Java

Android



Kotlin Compatibility



Java

Works with existing Java-based Android libraries



Android Studio

Can include both Java and Kotlin code in a single project



Android Deployment

Apps work across
Android versions
No special on-device
software required



Basic Types

Signed Integers

Type	Size
Byte	8 bit
Short	16 bit
Int	32 bit
Long	64 bit

Floating Point

Type	Size
Float	32 bit
Double	64 bit

Other Basic Types

Type	Description
Boolean	true or false
Char	Single character
String	Sequence of characters



Declaring Variables

Var

Val

Mutable Variable

Value set when assigned Value can later be changed

Assign-once (read-only) Variable

Value set when assigned Value cannot be changed once set



```
var student: String
student = "Jenny Student1"
// Do some work ...
student = "Amit Student2"
// Do some more work ...
val company: String
company = "Pluralsight"
company = "Amother Company"
```

- **◄** Declare mutable variable
- Assign initial value

■ Assign new value

- **◄** Declare assign-once variable
- Assign initial value
- **■** ERROR!

```
var student = "Jenny Student1"
// Do some work ...
student = "Amit Student2"
// Do some more work ...
val company = "Pluralsight"
company = "Ansther Company"
```

- Mutable variable type inferred
- Assign new value

- Assign-once variable type inferred
- ERROR!

Defining Types

Define types using the class keyword

Type name follows the class keyword

Class body enclosed in brackets



What Commonly Makes Up a Class?

Properties

Primary Constructor

Functions

Initialization Blocks **Secondary Constructors**



Properties

Represent a value within a class

Must specify mutability

- Declare with var when mutable
- Declare with val when assign-once

Can simply store and return value

Can optionally associate code

- Can provide getter code
- Can provide setter code



Class with Properties

```
class Person {
  val name: String = "Jim"
  var weightLbs: Double = 0.0
  var weightKilos: Double
    get() = weightLbs / 2.2
    set(value) {
        weightLbs = value * 2.2
```

val p = Person()
val name = p.name

p.weightLbs = 220.0
val kilos = p.weightKilos

p.weightKilos = 50.0
val lbs = p.weightLbs

- **◄** Creates new instance of Person
- Returns "Jim"

- **◄** Stores 220.0 in weightLbs
- Runs weightKilos getter Returns 100.0

- Runs weightKilos setter
- **◄** Returns 110.0

Primary Constructor

Accepts list of construction parameters

- Appears after the class name
- Optionally use the constructor keyword
- Parameters used to initialize class
- Contains no code



Primary Constructor

```
class Person {
  val name: String = "Jim"
  var weightLbs: Double = 0.0
  var weightKilos: Double
    get() = weightLbs / 2.2
    set(value) {
        weightLbs = value * 2.2
```

Primary Constructor

```
class Person(name: String, weightLbs: Double) {
  val name: String = name
  var weightLbs: Double = weightLbs
  var weightKilos: Double
    get() = weightLbs / 2.2
    set(value) {
        weightLbs = value * 2.2
```

Primary Constructor

```
class Person(name: String, weightLbs: Double) {
 val name = name
 var weightLbs
 var weightKilos: Double
   get() = weightLbs / 2.2
   set(value) {
       weightLbs = value * 2.2
```

val p = Person("Bob", 176.0)

val name = p.name

val lbs = p.weightLbs

val kilos = p.weightKilos

■ Creates new instance of Person

- Returns "Bob"
- **◄** Returns 176.0
- Runs weightKilos getter Returns 80.0

Declare Properties in the Primary Constructor

```
class Person(name: String, weightLbs: Double) {
 val name = name
 var weightLbs
 var weightKilos: Double
   get() = weightLbs / 2.2
   set(value) {
       weightLbs = value * 2.2
```

Declare Properties in the Primary Constructor

```
class Person(name: String, weightLbs: Double) {
 val name = name
 var weightLbs
 var weightKilos: Double
   get() = weightLbs / 2.2
   set(value) {
       weightLbs = value * 2.2
```

Declare Properties in the Primary Constructor

```
class Person(val name: String, var weightLbs: Double) {
```

```
var weightKilos: Double

get() = weightLbs / 2.2

set(value) {
    weightLbs = value * 2.2
}
```

val p = Person("Bob", 176.0)

val name = p.name

val lbs = p.weightLbs

val kilos = p.weightKilos

■ Creates new instance of Person

- Returns "Bob"
- **◄** Returns 176.0
- Runs weightKilos getter Returns 80.0

Declaring functions

- Use fun keyword
- Optionally has list of parameters
- Parameters can have default values



Specifying function return type

- Return type specified after parameters
- Technically all functions return a value
- If no useful value, return type is Unit
- Unit return type can be omitted



```
class Person(val name: String, var weightLbs: Double) {
   // weightKilos declaration elided for clarity
   fun eatDessert(addedIceCream: Boolean): Unit {
     weightLbs += if (addedIceCream) 4.0 else 2.0
   }
```



```
class Person(val name: String, var weightLbs: Double) {
   // weightKilos declaration elided for clarity
   fun eatDessert(addedIceCream: Boolean) {
     weightLbs += if (addedIceCream) 4.0 else 2.0
   }
```



```
class Person(val name: String, var weightLbs: Double) {
   // weightKilos declaration elided for clarity
   fun eatDessert(addedIceCream: Boolean = true) {
     weightLbs += if (addedIceCream) 4.0 else 2.0
   }
```



```
class Person(val name: String, var weightLbs: Double) {
  // weightKilos declaration elided for clarity
  fun eatDessert(addedIceCream: Boolean = true) {
    weightLbs += if (addedIceCream) 4.0 else 2.0
  fun calcGoalWeightLbs(lbsToLose: Double = 10.0): Double {
    return weightLbs - lbsToLose
```

Passing Parameters

All parameters must have a value specified

- Can be passed explicitly
- Can use default value



```
val p = Person("Bob", 176.0)
```

p.eatDessert(false)

val lbs = p.weightLbs

p.eatDessert()

lbs = p.weightLbs

val gw = p.calcGoalWeightLbs()

■ addedIceCream passed as false

◄ Returns 178.0

■ addedIceCream passed as default (true)

◄ Returns 182.0

■ IbsToLose passed as default (10)
 Returns 172



Passing Parameters

Parameters can be passed positionally

- 1st value passed to 1st parameter
- 2nd value passed to 2nd parameter
- So forth...



Passing Parameters

Parameters can be passed by name

- Include names when calling function
- Can be passed in any order

Using named parameters

- Can use when calling functions
- Can use when calling constructors



Passing Parameters

```
val p1 = Person("Jim", 185.0)
```

val p2 = Person(weightLbs = 185.0, name = "Jim")



Executing Code When Instance Created

Initializer Blocks

Code is run automatically as part of all instance creations

Secondary Constructor

Code run only when instance created with specific constructor



Executing Code When Instance Created

Initializer block

Always runs during construction

Can have multiple if desired

All will run every time

Secondary constructor

Runs only when used

Can have multiple if desired

Runs when used to construct instance

Code runs after all initializer blocks

Must delegate to primary constructor if type includes one



Summary



Kotlin compatibility with Java & Android

- Can use Java libraries
- Projects can have both Java & Kotlin
- No special deployment requirements

Declaring variables

- Mutable variables declared with var
- Assign-once variables declare with val

Variable typing can be inferred

- When assigned as part of declaration
- Compiler will use assignment's type



Summary



Types defined as classes

Properties

- Mutable properties declared with var
- Assign-once properties declared with val
- Can optionally provide custom get/set

Primary constructor

- Accepts construction parameters
- Parameters can have default values
- Can specify parameters as properties



Summary



Functions

- Declared using fun keyword
- Parameters can have default values

Passing parameters

- Can be passed positionally
- Can be passed using name

Initializer blocks

- Run as part of instance creation
- Can interact with class properties
- Can access primary constructor parameters

