# Android Development in Kotlin

## Lab Guide

# Lab 01: Kotlin Overview

1. Run the HelloKotlinWorld app in the Kotlin Playground
2. Run the HelloKotlinWorld app using command line tools

**END OF LAB**

# Lab 02: Data Types and Variables

1. Write a program that creates a constant pi (3.1415926). Use it to print the area of a circle with a radius of 3 cm.

2. Write a program that converts the numbers: 75, 111, 116, 108, 105, 110 to characters and prints them.

3. Write a program that adds a Int (35000) to a Short (30000). Use underscores in your literals.

**END OF LAB**

# Lab 03: Keywords and Operators

1. Execute the code sample from the *Short Circuiting Operations* slide. Make sure you can explain short circuit behavior.

2. Write a program that compares two values (42.0 and 42.0f) and prints true if they are equal. (hint: they should be equal!)

**END OF LAB**

# LAB 04: Arrays and Strings

1. Write program that uses a single Escaped String to print the following

   ***The customer said, "I think it's pretty expensive!"***
   ***The clerk replied, "No, it's just $10."***

2. Modify the program from Exercise 1 using a Raw String.
3. Write a program that creates an array of Ints and uses String Templates to print the array and the average (converted to an Int).

   ***The average of [1,2,3,4,5] is 3.***

4. Write a program that creates an array of values from 1 to 10. Use a filter to extract and print the values that are factors of 42. (hint: 2,3,6,7 – exclude 1)

5. Write program that takes this string, "She sells seashells by the seashore" and prints the number of distinct (non-space) characters (treat upper and lower case characters as the same).

   Hints:

   - you can write this program in one line by chaining together methods.
   - use code completion or the Kotlin docs to look at the various array conversion (toXXX) methods available
   - consider how to get rid of the spaces for counting purposes

**END OF LAB**

# Lab 05: Clustering Columns

1. Write a program to print the sum of all the even numbers between 1 and 100 (inclusive)
2. Write a program to print all the prime numbers from 1 to 100
3. Write a program to print the letters (lowercase) of the alphabet *not contained* in "kotlin"
4. Use a when statement to print the odd numbers and multiples of 4 between 1 and 100
5. Write a program that uses a loop to test whether a string is a palindrome.

   Hint: a palindrome is a word/phrase that is the same (ignoring case/spaces) forwards and backwards (e.g. "kayak", "taco cat")

**END OF LAB**

# Lab 06: Null Safety

1. Write a Kotlin app that throws a NullPointerException
2. Create a nullable String. Use the *safe call* and *Elvis* operators to print the String length. If the String is null print "null String"
3. Implement the three ways to call a method on a nullable reference. Demonstrate them with both a null and non-null reference. Do not duplicate any code.

**END OF LAB**

# Lab 07: Object Orientation

1. Create *Point* class
   - Points have two Int fields (x,y)

   Create a singleton *PointFactory* class with a function to create Point objects

   The PointFactory should keep track of the number of Points created

   Implement a main() method that uses the PointFactory, creates a few Points

   then prints the number of Points created

2. Create a hierarchy of Employee types. Include at least:
   - Managers who get a fixed salary
   - Wait Staff who get paid hourly plus tips
   - Hosts who get paid hourly
   - Salespeople who get a base + % of sales

   Create at least a couple of each and put them in an array. Print out how much each person is making this week.

   Print out the total amount paid in salary (manager salary + sales base), hourly (wait staff hourly + host hourly), tips, commissions

**END OF LAB**

# Lab 08: Functions and Lambdas

1. Create a point class similar to the one from the previous module. Add an extension function to find the distance of the point from the origin (0,0)
   (The formula is square_root(x^2 + y^2))

2. Create a *higher order function* that returns a *lambda* with this signature:
   () -> Unit
   Store the lambda in a variable. Call the lambda multiple times. Each time it is called, the lambda should print a higher number (1,2,3..). Use a closure instead of a companion object.

3. Create your own IntArray.myFilter() extension function. It should behave exactly like IntArray.filter() hint: use **var** list = *arrayListOf*<Int>()

**END OF LAB**

# Lab 9: More Functions

1. Create a Point class that stores the (X,Y) coordinates of a Point. Overload the + (plus) and == operators.

   Hint: base your comparisons on distance from (0,0)

2. Create a Rectangle class that constructs a rectangle from two points.

   Create a new infix function so that this expression is true if the rectangles are similar: *rect1 similar rect2*

   Hint: rectangles are similar if their aspect ratios are the same

3. Create a recursive function to calculate the sum of numbers between 1 and *n.* Is your function a candidate for *tail recursion?*

4. Create an array of Points, use the filter() function select all Points on the x or y axis. Use let() to print the results

**END OF LAB**

# Lab 10: Advanced Object Orientation

1. Modify the Employee hierarchy from an earlier lab so that each specialized Employee type keeps track of how many instances of that type are created. Also have the superclass (Employee) track the total number of Employees created of all types. Hint: use companion objects

2. Create a Price interface that defines a *cost* field and a *total() function.* Create an Item class with a description field that implements the Price interface through a delegate. Create two implementations (delegates) of *Price.* On that returns the cost from total() and one that returns the cost plus 7% tax from the total.

   Create some taxable and non-taxable items and print their prices and descriptions.

3. Create a singleton FishFactory that creates Fish objects. Fish objects should have name(species) and length(inches). Make Fish an inner or nested class of FishFactory. Create an ArrayList of fish objects. Iterate across the ArrayList and print the name(species) of each Fish. Explain why you chose to use a nested or inner class

**END OF LAB**

# Lab 11: Collections

1. Create a map using months (pick 3 or 4) as keys (e.g. "JAN", "FEB","MAR"). And IntArrays as values. Populate the IntArrays with random integers representing individual sales in a given month as shown below

```
var salesByMonth = mapOf(
  "JAN" to intArrayOf(100,110,10,600,2000,20,110,10),
  "FEB" to intArrayOf(50,4,200),
  "MAR" to intArrayOf(1000,1123,1234)
)
```

   Use map and reduce functions to reduce the intArrays to a monthly sales total and create a new map. Print the new map. The output for the map above looks like:

```
{JAN=2960, FEB=254, MAR=3357}
```

2. Create a list of Collections of Ints as shown

```
var a = listOf(
  listOf(100,200,300),
  setOf(10,20,30,40,50),
  listOf(1000,2000)
 )
```

   Select the appropriate functions (map(), reduce(), filter(), flatten()) to

   a)create a list of the sums of each nested list [600,150,3000]

   b) find the sum of all the elements of all the collections

**END OF LAB**

# Lab 12: Concurrency

1. Write a program that initiates two coroutines. Each coroutine should call a function that prints a character 10 times pausing for a few milliseconds between prints. Pass the character and the delay time into the function. The application should not terminate until all the characters have been printed.

**END OF LAB**

# Lab 13: Android Tools

1. Log into the Linux VM (user: wasadmin, pass: wasadmin)
2. Open a terminal and start Android Studio with this script:

   ```
   ~/android-studio/studio.sh
   ```

3. Create a new Android application called *Android Tools* from the *Empty Compose Activity* template
4. Build and run the application in the Android Emulator

   Note: There is already an AVD in the Device Manager. This AVD may take 5 or minutes to start up because it is running in a virtual machine and cannot use hardware acceleration. Normally the emulator is fast.

**END OF LAB**

# Lab 14: Anatomy of an Application

1. Examine the application you created in the previous lab
2. What are the 4 categories of dependencies defined in the *build.gradle* file?

   _____

   _____

   _____

   _____

3. What is the default font size defined in the theme?
4. How many Android tests were generated?
5. What is the HEX value for the application-defined color *purple_500*?

**END OF LAB**

# Lab 15: Composable Functions

1. Create a new Android application
2. Create a composable function that displays your name as below:

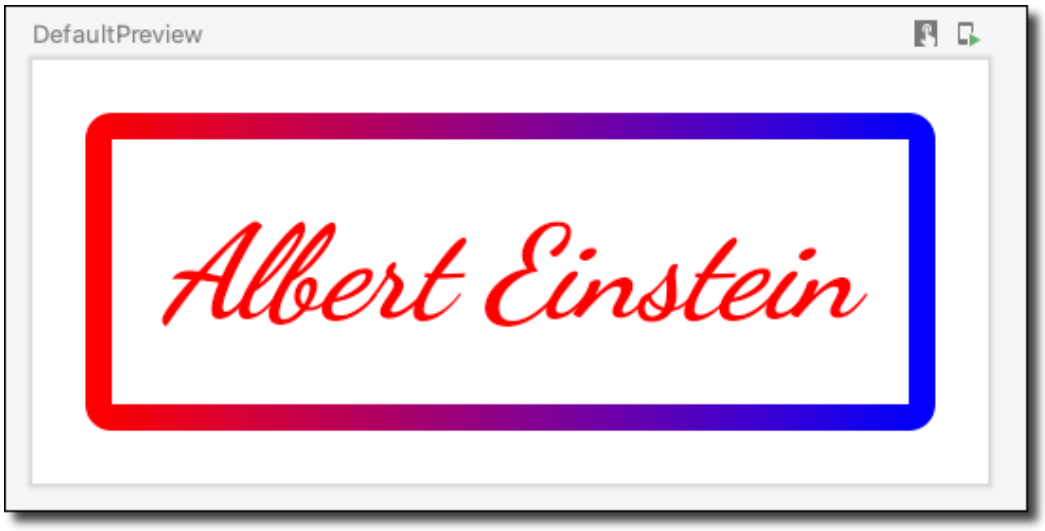

**END OF LAB**

# Lab 16: Modifiers

1. Add modifiers to the application from the *Composables* lab to produce this output:

   You will need to use a border and specify a shape and you will need to define a *gradient brush*.

   Also, be careful about the order in which you apply your modifiers!



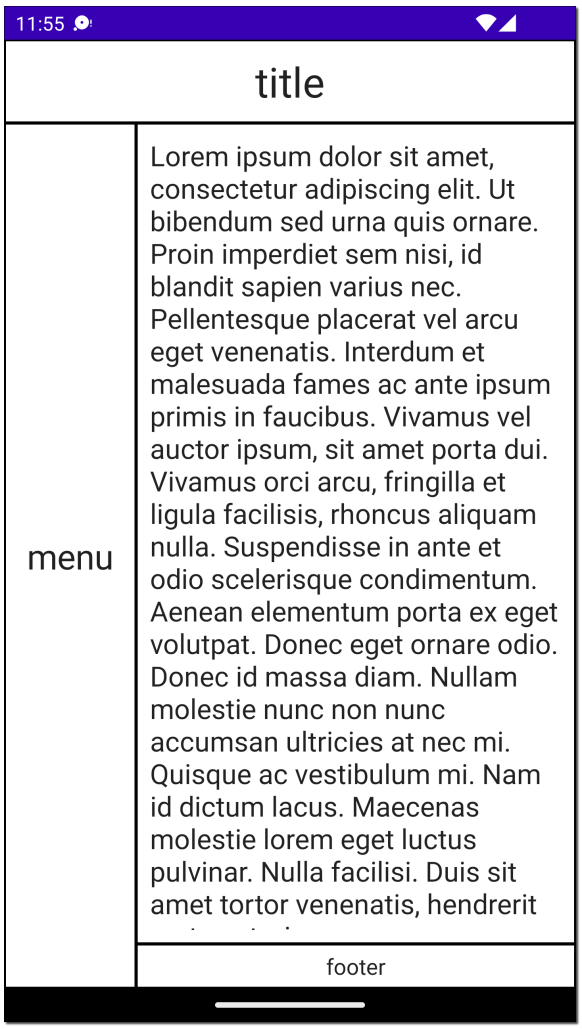

d

**END OF LAB**

# Lab 17: Layouts

1. Use nested layouts to reproduce the complex layout on the right

2. The sample used these modifiers:
   - border()
   - fillMaxWidth()
   - fillMaxHeight()
   - fillMaxSize()
   - padding()
   - weight()




**END OF LAB**