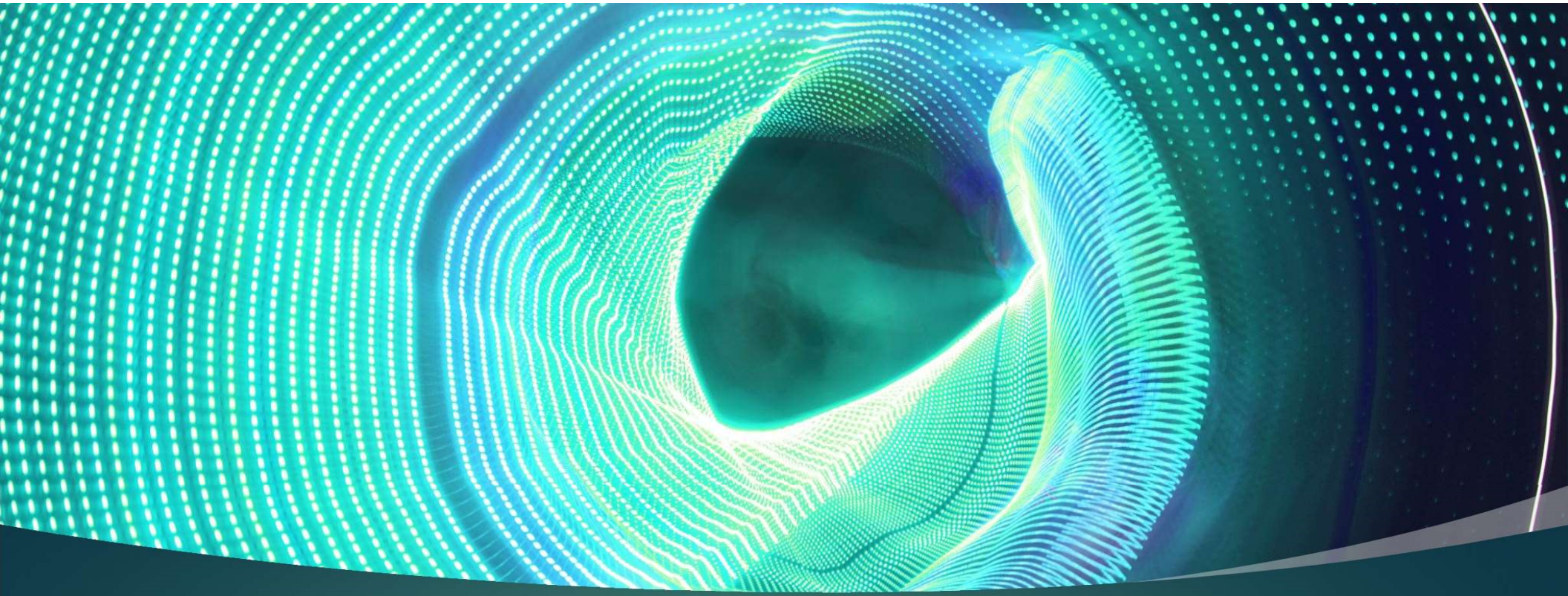
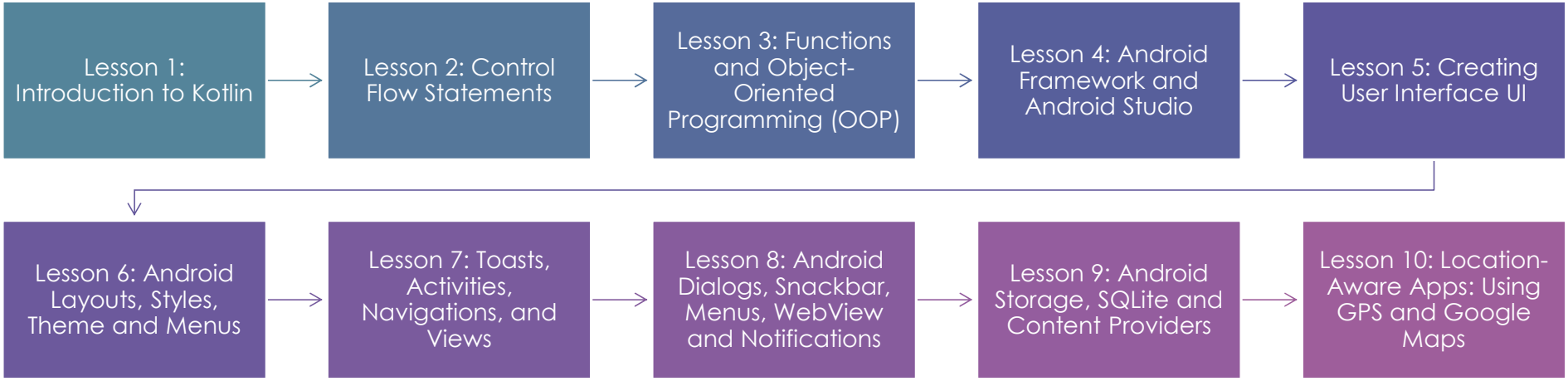


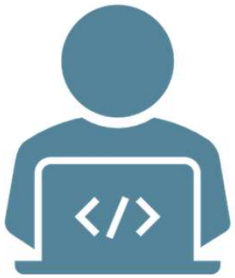
Android Application Development

AND-801

Course Contents



Your Trainer



Asmaliza Ahzan @ Emma

asmaliza@iverson.com.my

Participants Introduction

- Name
- Job Title/Description
- Programming Experience(s)
- Expectation(s)
- Thank you! ☺

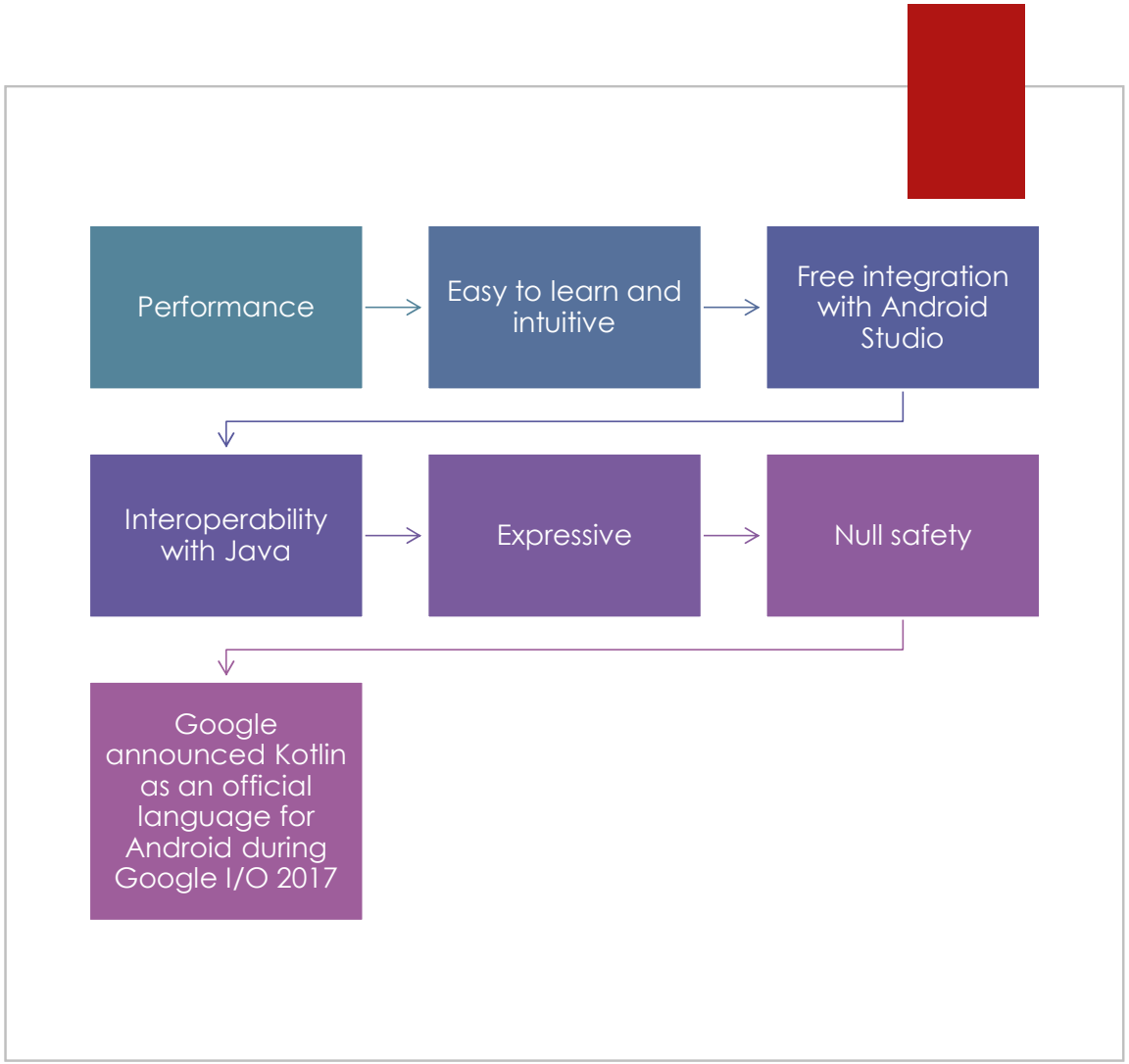
Introduction to Kotlin

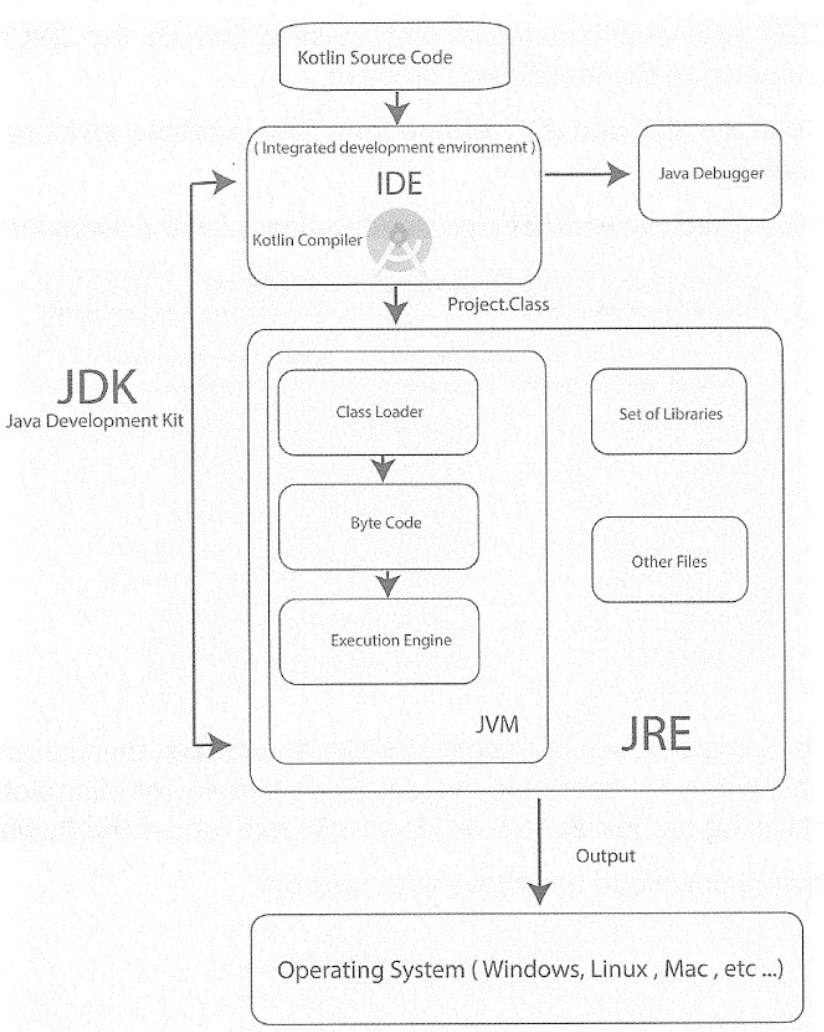
LESSON 1

Kotlin History

- July 2011, JetBrains unveiled Project Kotlin – a new language for the JVM.
- It has been under development for a year.
- One of the stated goals of Kotlin is to compiled as quickly as Java.
- February 2012, JetBrains open-sourced the project under the Apache 2 license.
- Kotlin is 100% interoperable with Java and Android.

Kotlin Advantages





How Kotlin Programs Work?

Kotlin Software Prerequisites

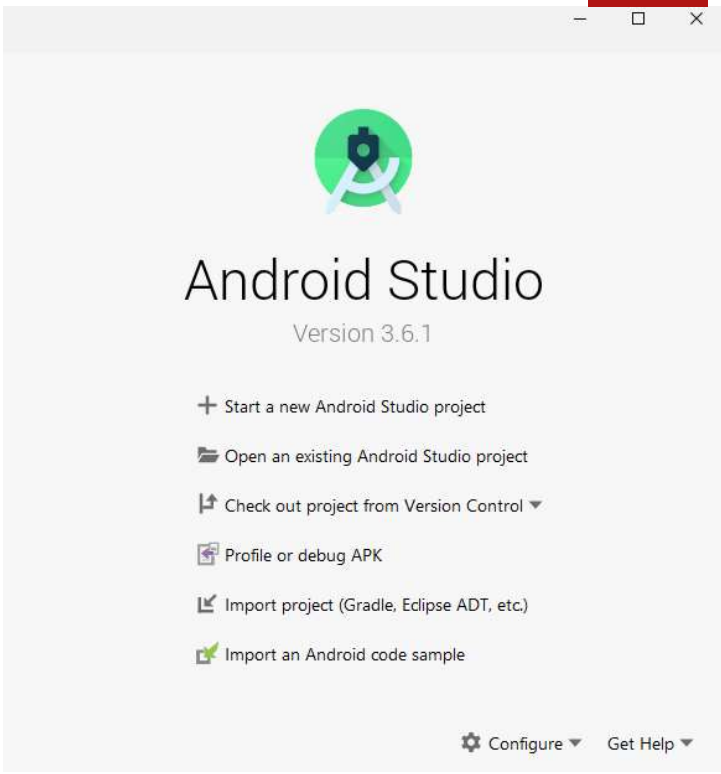
Install Java JDK and JRE

Installing IntelliJ IDEA (optional)

Installing Android Studio

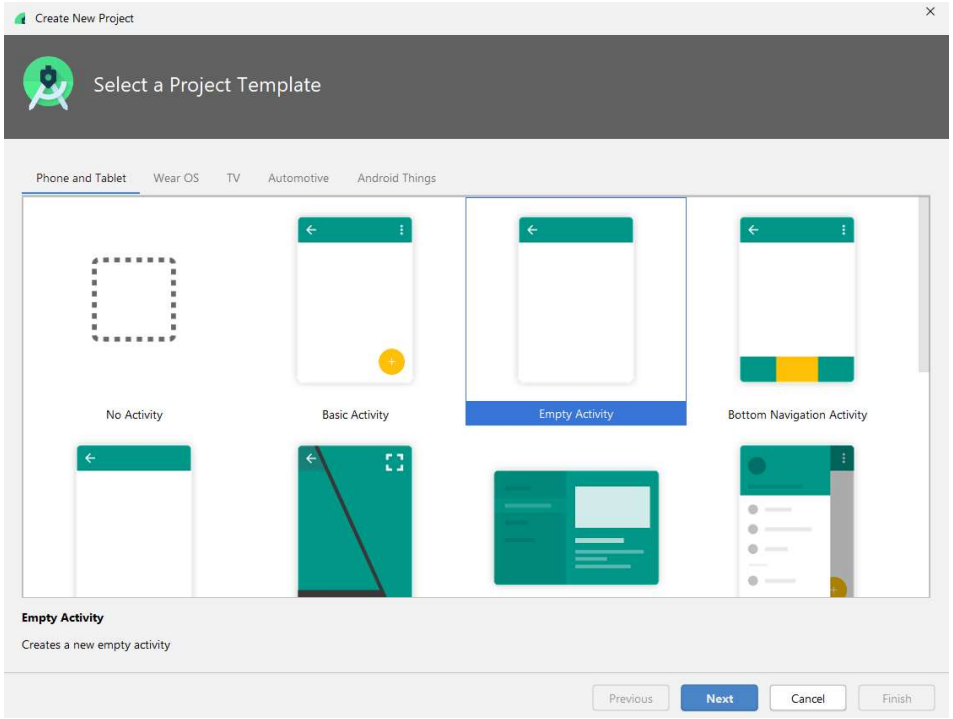
Creating Your First Kotlin Project using Android Studio

- Launch Android Studio
- Start a new Android Studio Project



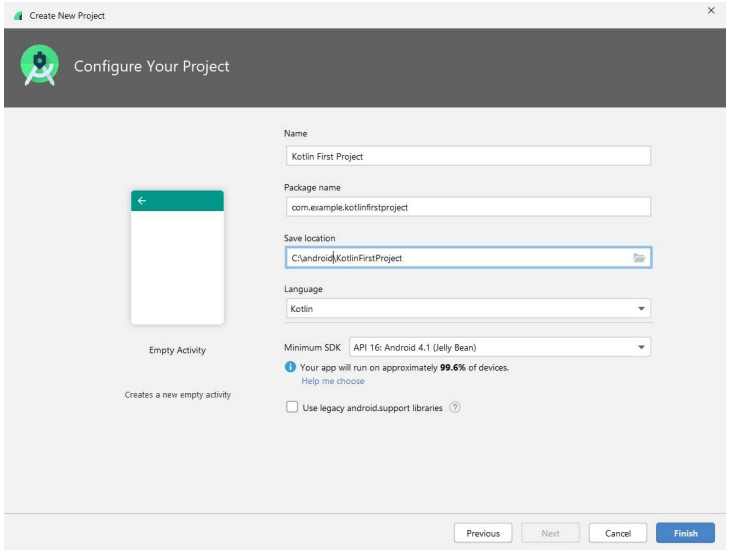
Select a Project Template

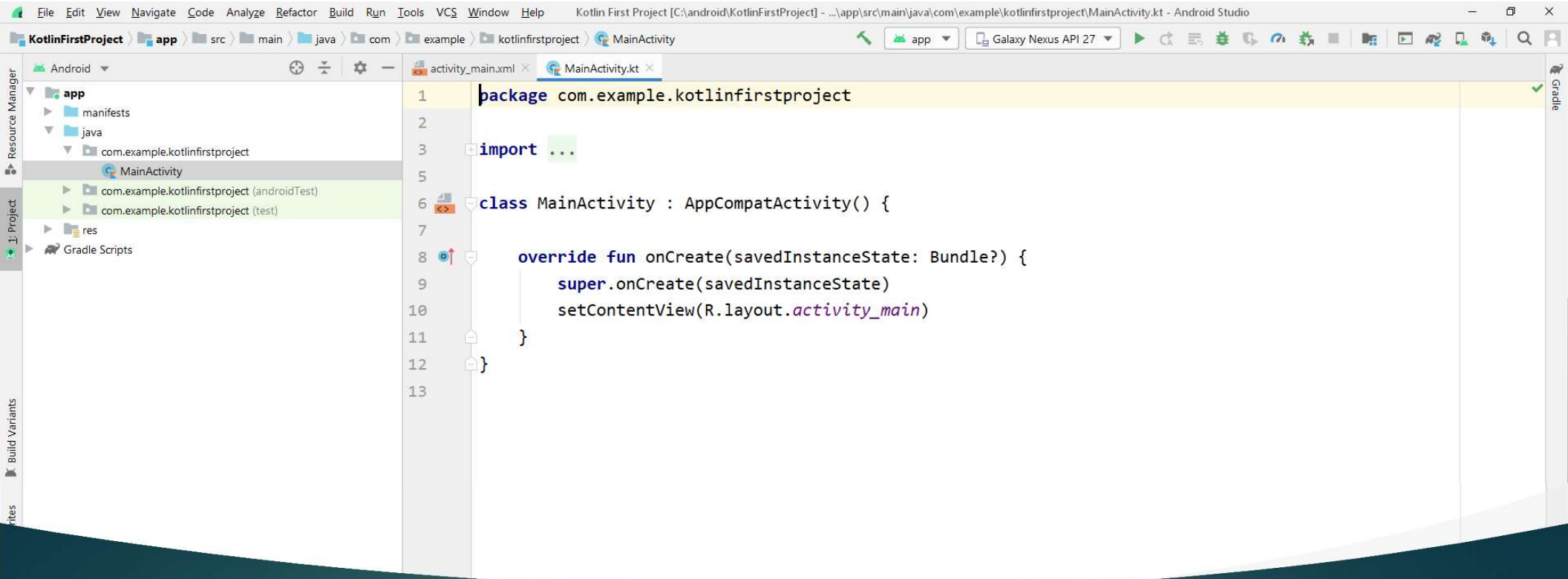
- Select ‘Empty Activity’



Configure Your Project

- Name: Kotlin First Project
- Package name: com.example.kotlinfirstproject
- Save location: c:\android\KotlinFirstProject
- Language: Kotlin
- Minimum SDK: API 16



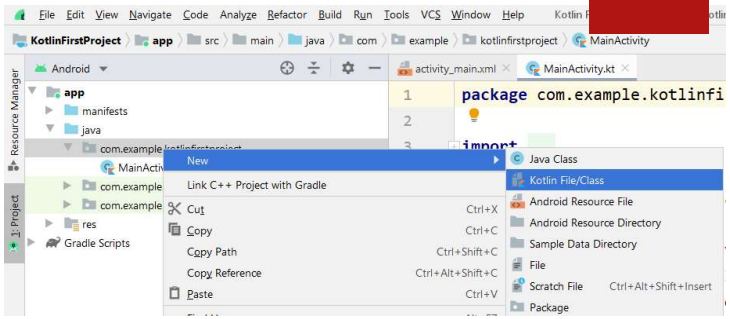


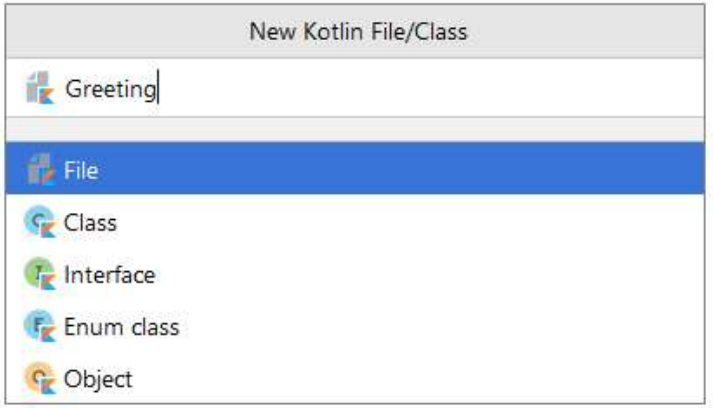
New Project Created

YOU ARE NOW READY TO CODE!

Creating a Kotlin Program

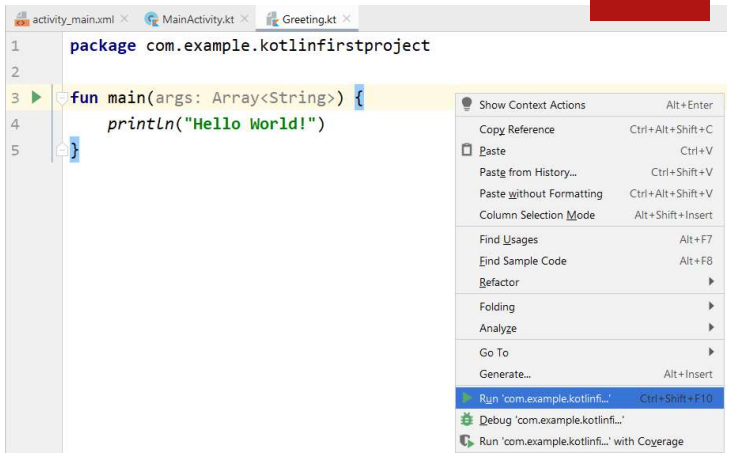
- A Kotlin program consists of a group of classes; each file performs a part of the Kotlin program.
- To create a Kotlin file
  - Right-click package > New > Kotlin File/Class
  - Name: Greeting
  - Type: File

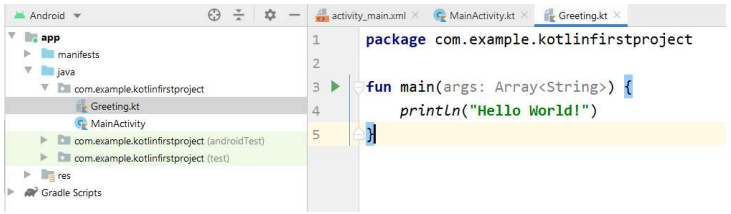




Running a Kotlin Program

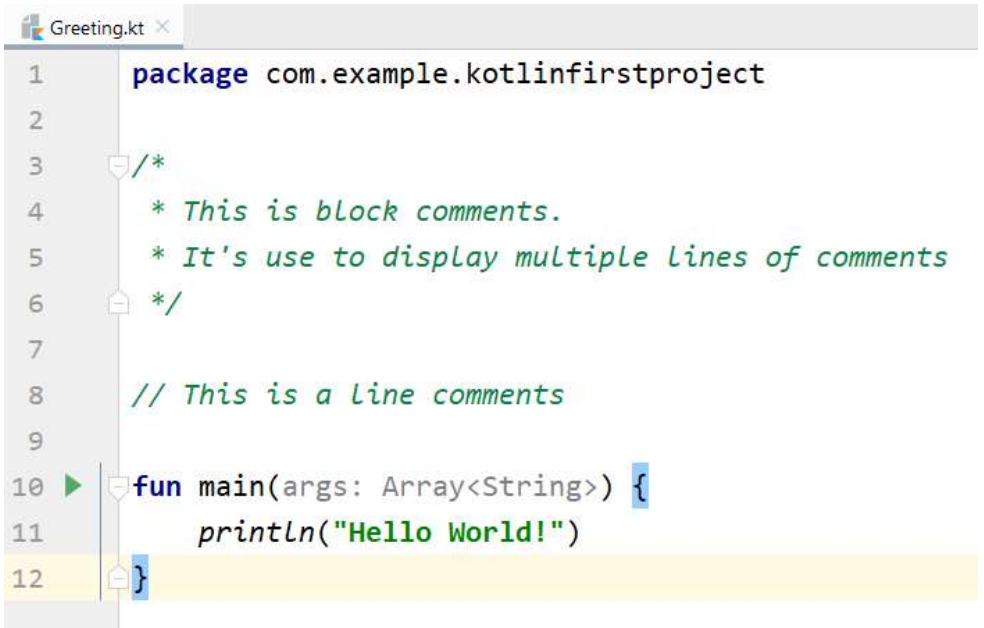
- Type the code to display “Hello World!”
- Right-click file > Run…
- “Hello World!” message should be displayed in the output console





Writing Comments

- Line Comments
- Block Comments



Kotlin Variables

- Mutable Variables
  - Value can be changed/updated anytime
  - Use the **var** keyword
- Immutable Variables
  - Value cannot be changed, like a constant
  - Use the **val** keyword

```
10  fun main(args: Array<String>) {
11      println("Hello World!")
12
13      // mutable variable
14      var name = "John"
15
16      // immutable variable
17      val age = 19
18
19      println(name + " is " + age + " years old")
20  }
```

Kotlin Data Types

- String
  - Use double-quotes
  - Used to store words or sentences
- Character
  - Use single-quote
  - Used to store a single character
- Boolean
  - True or False
- Numbers – next slides

```
10  fun main(args: Array<String>) {
11      println("Hello World!")
12
13      // string
14      val name = "John Smith"
15
16      // character
17      val gender = 'M'
18
19      // boolean
20      val overtime = true
21  }
```

Kotlin Numbers

Data Type	Description	Default Value
Byte	8-bits signed integer	0
Short	16-bits signed integer	0
Int	32-bits signed integer	0
Long	64-bits signed integer	0L
Float	32-bits floating point number	0.0F
Double	64-bits floating point number	0.0

Kotlin Array

- An array is used to store a group of values, all of which have the same data type.
- Its length/size is fixed.
- Cannot be resized.
- To create an array, use the function arrayOf()
- Array has index.
- To access elements in an array, use the index value.

```kotlin
fun main(args: Array<String>) {
    // create an array of 5 numbers
    val numbers = arrayOf(1, 2, 3, 4, 5)

    // access the first element in array
    println("First element: " + numbers[0])

    // access the last element in array
    println("Last element: " + numbers[numbers-1])
}
```

Data Type Conversions

- In some cases you may need to convert a data type for a variable to another data type such as changing an integer to a short.
- List of functions available;
  - toByte()
  - toShort()
  - toInt()
  - toLong()
  - toFloat()
  - toDouble()
  - toChar()
  - toString()

```
3  fun main(args: Array<String>) {
4
5      val num1 : Int = 99
6      val num2 : Short = num1.toShort()
7      val num3 : Byte = num1.toByte()
8      val strNum : String = num1.toString()
9
10 }
```

Input of Information to Kotlin Program

- The readLine() function allows the program user to enter a string values or intercept keyboard input from the console.

```
3  fun main(args: Array<String>) {
4      println("====================================")
5      println("Welcome to Android ATC")
6      println("====================================")
7      println("Enter Your Name : ")
8      var name = readLine()
9      println("Enter Your Age : ")
10     var age = readLine()
11     println("Thank you, your name is $name and age is $age")
12
13     println("====================================")
14     println("Welcome to Android ATC Calculator")
15     println("====================================")
16     println("Enter First Number : ")
17     var number1 : Int = Integer.valueOf(readLine().toString())
18     println("Enter Second Number : ")
19     var number2 : Int = Integer.valueOf(readLine().toString())
20     println("Add : " + (number1 + number2))
21     println("Sub : " + (number1 - number2))
22     println("Mul : " + (number1 * number2))
23     println("Div : " + (number1 / number2))
24 }
```

Mathematical Operations

- Alternatively, use the mathematical functions instead of operators.

```
3  fun main(args: Array<String>) {
4      println("====================================")
5      println("Welcome to Android ATC Calculator")
6      println("====================================")
7      println("Enter First Number : ")
8      var number1 : Int = Integer.valueOf(readLine().toString())
9      println("Enter Second Number : ")
10     var number2 : Int = Integer.valueOf(readLine().toString())
11
12     // use function instead of operator
13     println("Add : " + number1.plus(number2))
14     println("Sub : " + number1.minus(number2))
15     println("Mul : " + number1.times(number2))
16     println("Div : " + number1.div(number2))
17
18     // alternatively
19     println("Add : ${number1.plus(number2)}")
20     println("Sub : ${number1 - number2}")
21
22     // another alternative
23     println("Enter another two numbers : ")
24     var first = readLine()!!.toInt()
25     var second = readLine()
26     println("User inputs : $first $second")
27 }
```

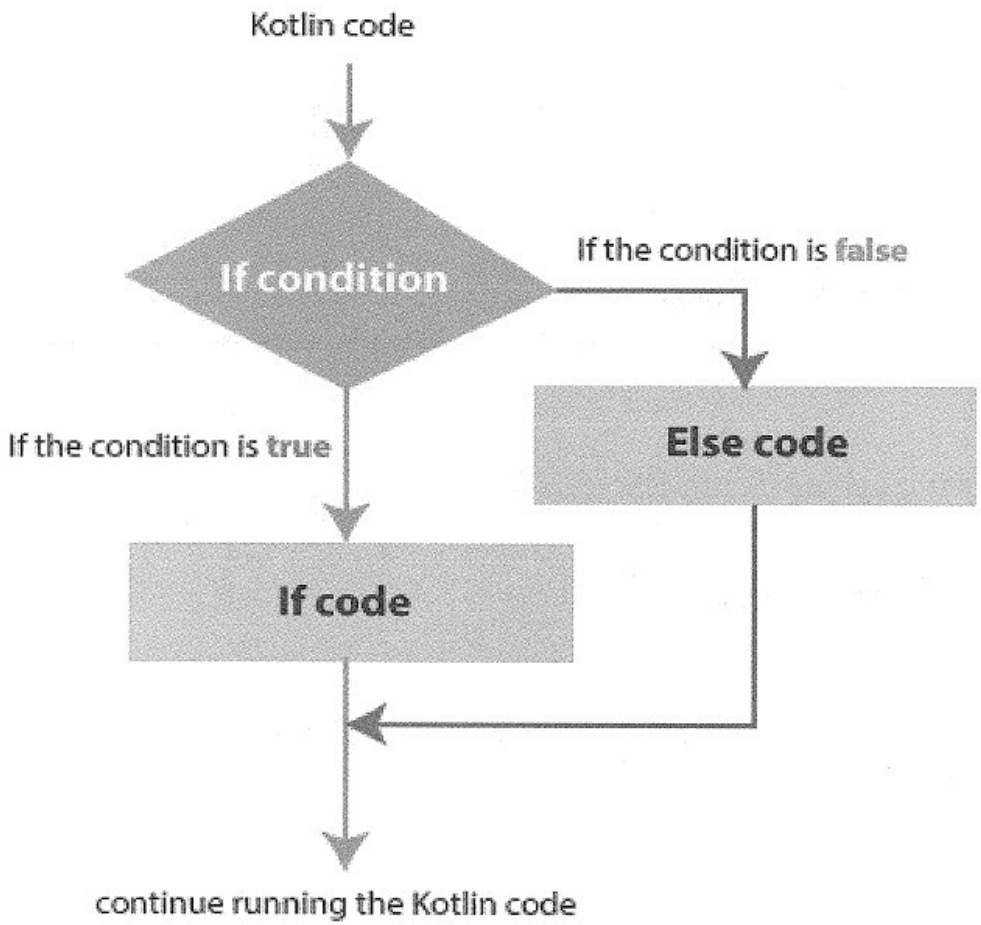
Control Flow Statements

LESSON 2

Introduction

- The statements inside the program generally executed from top to bottom.
- Control Flow Statements allows the execution to be broken by applying decision making, iteration, branching or conditionally execute a partial part of the program.

Operator	Name
==	Equal to
!=	Not Equal to
<	Less Than
<=	Less Than or Equal to
>	Greater Than
>=	Greater Than or Equal to



If-Else Statement

```
3  fun main(args: Array<String>) {
4      var x = 10
5      if (x > 30) {
6          println("$x is greater than 30")
7      } else {
8          println("$x is not greater than 30")
9      }
10     println("the end")
11
12     var score: Int = 75
13     var grade: String? = null  // nullable variable
14
15     if (score >= 90) grade = "A"
16     else if (score >= 80) grade = "B"
17     else if (score >= 70) grade = "C"
18     else if (score >= 50) grade = "D"
19     else grade = "F"
20
21     println("Score : $score, Grade : $grade")
22 }
```

If-Else-If Statement

If-Else and Logical Operator

A	B	A && B	A || B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

If-Else and Logical Operator

```
3  fun main(args: Array<String>) {
4      // logical operator
5      var age = 16
6      var year = 1998
7
8      // && and operator
9      if (age >= 18 && year >= 1998) println("Authorized")
10     else println("Not authorized")
11
12     // || or operator
13     if (age >= 18 || year >= 1998) println("Authorized")
14     else println("Not authorized")
15 }
```

```
3  fun main(args: Array<String>) {
4      // when statement
5      println("===== Pizza Order =====")
6      println("Enter the Pizza size: 1=> Small, 2=>Medium, 3=>Large")
7      var size = readLine()!!.toInt()
8      var price: Int? = null
9
10     when (size) {       // check for equality only
11         1 -> {
12             price = 5
13             println("Size: Small")
14         }
15         2 -> price = 7
16         3 -> price = 9
17         else -> println("Invalid pizza size")
18     }
19     if (price != null) {
20         println("Total price: $price")
21     }
22 }
```

When Statements

For Loops

```
3  fun main(args: Array<String>) {
4      // for loop
5      for (x in 0..5) {   // x is a number range 0-5
6          println(x)
7      }
8
9      println("Enter upper : ")
10     var upper = readLine()!!.toInt()
11     for (x in 1..upper) {
12         println(x)
13     }
14
15     var numbers: IntArray = intArrayOf(10, 20, 30, 40, 50, 60)
16     for (index in 0..numbers.size - 1) {
17         println(numbers[index])
18     }
19 }
```

```
3  fun main(args: Array<String>) {
4      var numbers: IntArray = intArrayOf(10, 20, 30, 40, 50, 60)
5
6      // while loop
7      var y = 0   // 1. initialization
8      while (y <= 5) {     // 2. expression (checking)
9          println(y)
10         y++          // 3. very important - update y
11     }
12
13     var index = 0
14     while (index < numbers.size) {
15         println(numbers[index])
16         index++
17     }
18 }
```

While Loops

```
3  fun main(args: Array<String>) {
4      var numbers: IntArray = intArrayOf(10, 20, 30, 40, 50, 60)
5
6      // do-while loop
7      var y = 0       // 1. initialization
8      do {
9          println(y)
10         y++     // 2. update
11     } while (y <= 5)     // 3. expression
12
13     var index = 0
14     do {
15         println(numbers[index])
16         index++
17     } while (index < numbers.size)
18 }
```

Do-While Loops

Branching Statements

- Break
  - Exit from the current block
- Continue
  - Continue with the next iteration
- Return – next lesson

```
3  fun main(args: Array<String>) {
4      var numbers: IntArray = intArrayOf(10, 20, 30, 40, 50, 60)
5
6      // break - exit from current block
7      for (index in 0..numbers.size - 1) {
8          if (numbers[index] == 30) {
9              println("Found 30")
10             break
11         }
12         println(numbers[index])
13     }
14
15     // continue - skip current but continue with next iteration
16     for (index in 0..numbers.size - 1) {
17         if (numbers[index] == 30) {
18             println("Found 30")
19             continue
20         }
21         println(numbers[index])
22     }
23 }
```

Functions and OOP

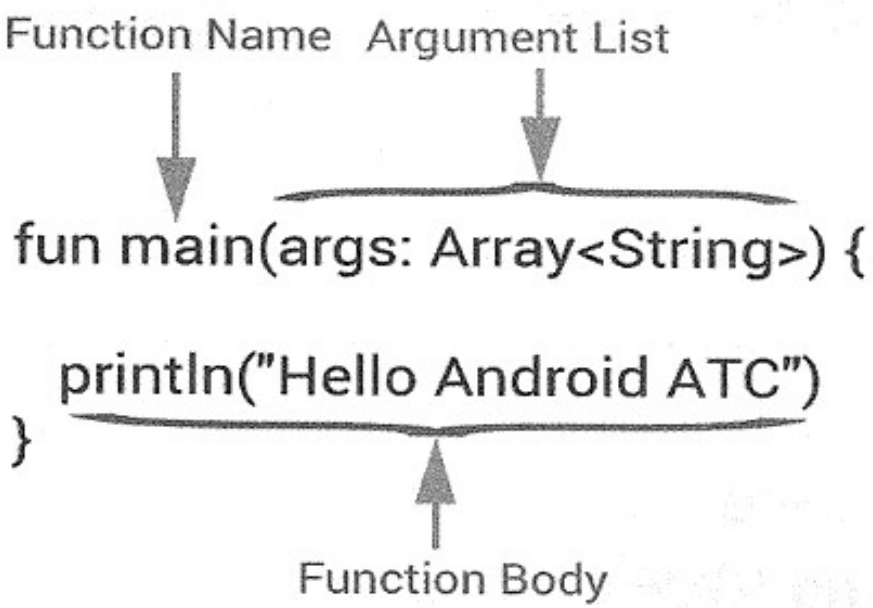
LESSON 3

Functions

A function is a block of codes or collection of statements grouped together to perform an operation.

Each function has a unique name which is used to call this function inside the Kotlin program.

Function removes the need to duplicate codes.



Function Structure

Creating a Function

- If you have a number of lines of codes that need to be used more than once within your program, you can gather them inside a function which has a specific name and then call it, as many times as needed.
- The function greet() is the most basic form of function, since it does not take any parameter (input) and does not return anything (output).

```
3  fun main(args: Array<String>) {
4      val country = "Malaysia"    // local scope
5      println("Country (main): $country")
6
7      // call the function greet
8      greet()
9  }
10
11 // define a function to display greeting
12 fun greet() {
13     println("Good Afternoon")
14 }
```

```
3  fun main(args: Array<String>) {
4      // call function and pass a parameter
5      greetUser( name: "John")
6
7      // call function addition and pass 2 number
8      addition( num1: 23,  num2: 76)
9  }
10
11 // define a function that takes a parameter
12 fun greetUser(name: String) {   // parameter also local scope
13     println("Hello $name")
14 }
15
16 // define a function to do addition, takes 2 numbers as input
17 fun addition(num1: Int, num2: Int) {
18     var total: Int = num1 + num2
19     println("Total : $total")
20 }
```

Function that takes parameter(s)

Function that returns

```
3  fun main(args: Array<String>) {
4      // call function multiple and get return value
5      var result = multiply( num1: 4,  num2: 5)
6      println("Result : $result")
7
8      result = division( num1: 40,  num2: 5)
9      println("Result : $result")
10 }
11
12 // define a function that takes 2 parameters and return a value
13 fun multiply(num1: Int, num2: Int): Int {
14     var total: Int = num1 * num2
15     return total
16 }
17
18 fun division(num1: Int, num2: Int): Int {
19     return num1 / num2
20 }
```

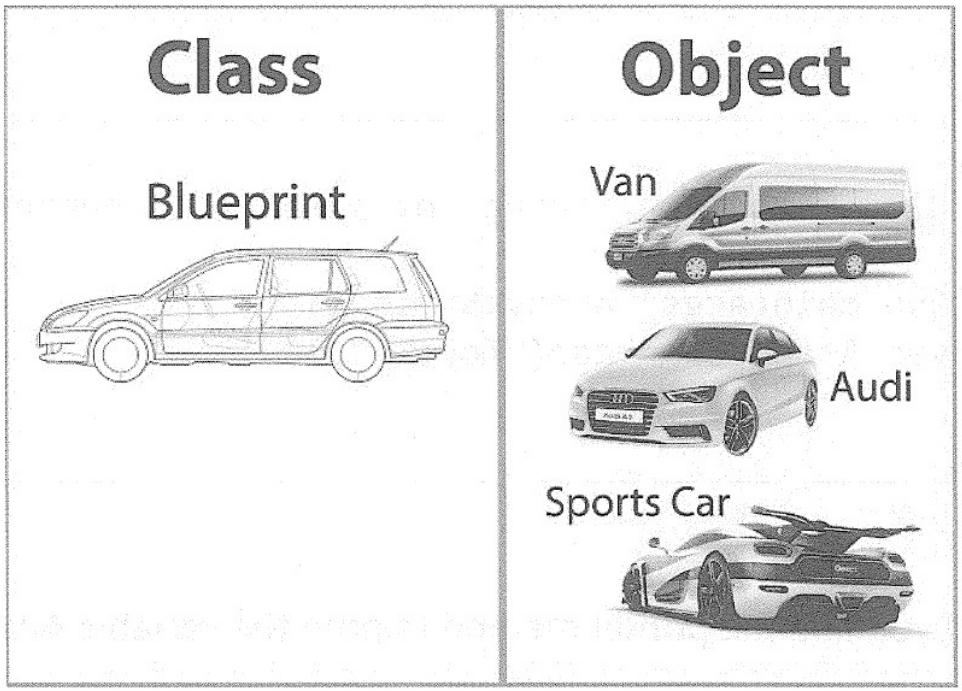
Function and Variable Scopes

- Global Variable
  - Defined outside any function
  - Accessible within the program
- Local Variable
  - Defined inside function
  - Function parameter also local
  - Accessible within function only

```
3   val my_company = "Nichicon"     // global scope
4
5   fun main(args: Array<String>) {
6       val country = "Malaysia"    // local scope
7       println("Country (main): $country")
8
9       println("Company (main): $my_company")  // global variable
10  }
11
12  fun greet() {
13      println("Good Afternoon")
14      println("Company (greet): $my_company")
15  //    println("Country (greet): $country")     // error
16  }
```

Object-Oriented Programming (OOP)

- Class
  - A blueprint or template that defines the structure of the object
- Object
  - An instance of a class
  - Object have properties and behaviours



```
3   // class definition with a default constructor that takes 3 parameters
4   class Car(type: String, maxspeed: Int, number_of_seats: Int) {
5
6   }
```

```
27  fun main(args: Array<String>) {
28      // create object Car
29      var mycar = Car( type: "Van",  maxspeed: 180,  number_of_seats: 10)
30      println(mycar)
31  }
```

Creating a Class

A VERY BASIC CLASS DEFINITION

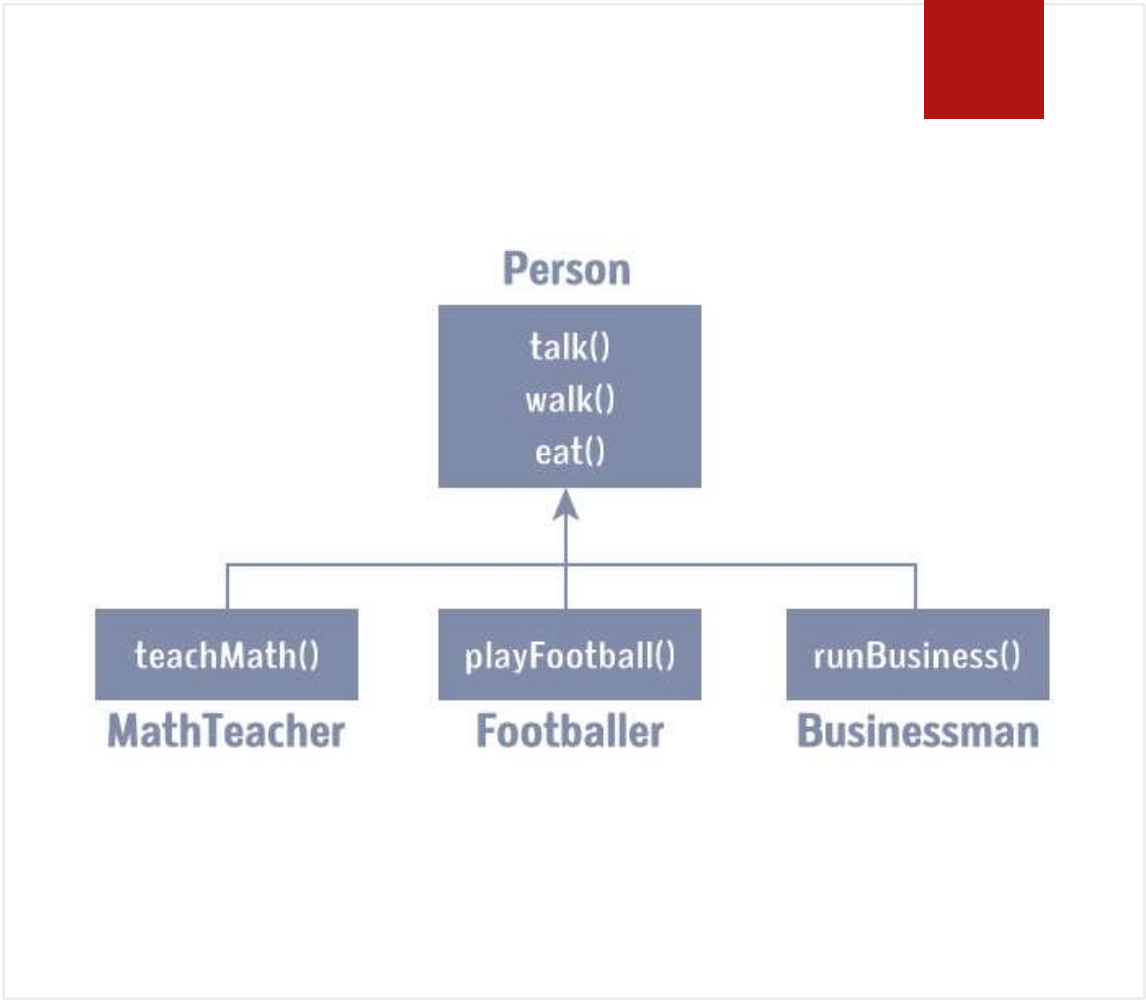
Better Class Definition

- A proper class definition should consists of
  - List of fields/properties
  - Constructor(s)
  - Functions
- To create new object from the class, use the constructor.

```
8   class Car {
9       var type: String? = null
10      var maxspeed: Int? = null
11      var number_of_seats: Int? = null
12
13      constructor(type: String, maxspeed: Int, number_of_seats: Int) {
14          this.type = type
15          this.maxspeed = maxspeed
16          this.number_of_seats = number_of_seats
17          display()
18      }
19
20      fun display(){
21          println("Type: ${this.type}")
22          println("Max speed: ${this.maxspeed}")
23          println("No of seats: ${this.number_of_seats}")
24      }
25  }

27  fun main(args: Array<String>) {
28      // create object Car
29      var mycar = Car( type: "Van",  maxspeed: 180,  number_of_seats: 10)
30      var yourcar = Car( type: "Sedan",  maxspeed: 200,  number_of_seats: 5)
31      yourcar.display()
32  }
```

Inheritance

- An “is-a” relationship between 2 classes
  - Superclass/Parent
  - Subclasses/Children
- Superclass usually is a more generic type.
- Subclass is more of a specific type.



Parent Class

- In this example, Person class contains
  - 3 fields
  - 2 constructors

```
3   // parent class (superclass)
4   // 1. open class of extension (inheritance)
5   // 3. update class to contain primary constructor ()
6   open class Person() {
7       var Name: String? = null
8       var Email: String? = null
9       var Age: Int? = null
10
11      // 4. call primary constructor from this secondary constructor
12      //      add this() at the end of constructor definition
13      constructor(Name: String, Email: String, Age: Int) : this() {
14          this.Name = Name
15          this.Email = Email
16          this.Age = Age
17          println("Name: " + this.Name)
18          println("Email: " + this.Email)
19          println("Age: " + this.Age)
20      }
21
22      // overloading 1 - call primary constructor
23      constructor(Name: String, Email: String) : this() {
24          this.Name = Name
25          this.Email = Email
26          println("Name: " + this.Name)
27          println("College: " + this.Email)
28      }
```

```
35  // child class (subclass)
36  // subclass will have the same set of properties, functions
37  // 2. Teacher class extends (:) Person class
38  //      Person() - calling the primary constructor
39  class Teacher() : Person()
```

Child Class

```
41  fun main(args: Array<String>) {
42      var person = Person()
43      person.Name = "John Smith"
44      person.Email = "john@example.com"
45      person.Age = 20
46      println("Name: " + person.Name)
47      println("Email: " + person.Email)
48      println("Age: " + person.Age)
49
50      var otherPerson = Person( Name: "Bob Taylor",  Email: "bob@example.com",  Age: 19)
51
52      var theTeacher = Teacher()
53      theTeacher.Name = "Sarah White"
54      theTeacher.Email = "Third"
55      theTeacher.Age = 40
56      println("Name: " + theTeacher.Name)
57      println("Email: " + theTeacher.Email)
58      println("Age: " + theTeacher.Age)
59
60      // create object using overloaded 1
61      var stud1 = Person( Name: "Lily Thomas",  Email: "lily@example.com")
62
63      // create object using overloaded 2
64      var stud2 = Person( Name: "James Bond")
65      stud2.Age = 37
66      println(stud2.Name + " age is " + stud2.Age)
67  }
```

Main Function

Overloading Constructors

- Functions that have the same name but different parameter list.
- Defined within the same class.

```
3   // parent class (superclass)
4   // 1. open class of extension (inheritance)
5   // 3. update class to contain primary constructor ()
6   open class Person() {
7       var Name: String? = null
8       var Email: String? = null
9       var Age: Int? = null
10
11      // 4. call primary constructor from this secondary constructor
12      //      add this() at the end of constructor definition
13      constructor(Name: String, Email: String, Age: Int) : this() {
14          this.Name = Name
15          this.Email = Email
16          this.Age = Age
17          println("Name: " + this.Name)
18          println("Email: " + this.Email)
19          println("Age: " + this.Age)
20      }
21
22      // overloading 1 - call primary constructor
23      constructor(Name: String, Email: String) : this() {
24          this.Name = Name
25          this.Email = Email
26          println("Name: " + this.Name)
27          println("College: " + this.Email)
28      }
```

Overriding Properties and Functions

- Subclass(es) override Superclass's properties and functions.
- Overriding functions requires that the methods have the SAME name and parameter list.

```
3   // open class of extension
4   open class Computer {
5       // open properties for overriding
6       open var x: Int = 5
7       open var y: Int = 11
8
9       open fun display(){
10          println("Computer x : " + this.x + ", y : " + this.y)
11      }
12  }
13
14  class Tablet : Computer() {
15      // override superclass's properties
16      override var x : Int = 2
17      override var y : Int = 4
18      var z : Int = 7
19
20      override fun display() {
21          println("Tablet x : " + this.x + ", y : " + this.y + ", z : " + this.z)
22      }
23  }
24
25  fun main(args: Array<String>) {
26      var comp1 = Computer()
27      comp1.display()     // display() in Computer
28
29      var tab1 = Tablet()
30      tab1.display()      // display() in Tablet
31  }
```

Abstract Class

- A class that contains at least one abstract function.
- An abstract function is a function without implementation i.e. no function body.
- An abstract class cannot be instantiated.

```
3   // abstract method is a method without body (implementation)
4   // if class contains at least 1 abstract method -> class is abstract
5   // cannot create object from abstract class
6   // abstract class can contain non-abstract method
7
8   abstract class Course {
9       var title: String? = null
10      var code: String? = null
11
12      abstract fun courseprice()
13      abstract fun courseprerequisite()
14
15      fun display() {
16          println("Title: " + title)
17          println("Code: " + code)
18      }
19  }
```

Inherit from Abstract Class

- Extends the abstract class in order to provide the implementation of the abstract function.
- If the subclass is not abstract, then it can be instantiated.

```
21  class ComputerCourse() : Course() {
22      override fun courseprice() {
23          println("Computer course price")
24      }
25
26      override fun courseprerequisite() {
27          println("Computer course pre-requisite")
28      }
29
30  }
31
32  class EngineeringCourse() : Course() {
33      override fun courseprice() {
34          println("Engineering course price")
35      }
36
37      override fun courseprerequisite() {
38          println("Engineering course pre-requisite")
39      }
40
```

```
43  fun main(args: Array<String>) {
44  //    var course1 = Course()  // error
45      var compCourse = ComputerCourse()
46      compCourse.title = "Kotlin Programming"
47      compCourse.code = "KOTLIN"
48      compCourse.display()
49      compCourse.courseprice()
50      compCourse.courseprerequisite()
51
52      var engCourse = EngineeringCourse()
53      engCourse.title = "Electronics"
54      engCourse.code = "ELEC"
55      engCourse.display()
56      engCourse.courseprice()
57      engCourse.courseprerequisite()
58  }
```

Test Abstract Class

Interface

- An interface is a type of class/structure that contains abstract functions only.
- Cannot instantiate from an interface.
- Allow multiple-inheritance.

```
6  interface Calc {
7      fun sum(x: Int, y: Int)
8
9      fun display() {
10         println("x : $x, y : $y")
11     }
12 }
13
14 interface Printable {
15     fun print()
16 }
```

Interface Implementations

- Create class(es) that implements the interface.
- If class is not abstract, can be instantiated.

```
18  // define a class that implements interface
19  // Math1 implements 2 interfaces (multiple inheritance)
20  class Math1 : Calc, Printable {
21      override fun sum(x: Int, y: Int) {
22          println("Sum is " + (x + y))
23      }
24
25      override fun print() {
26          println("Math1 print function")
27      }
28  }
29
30  class Math2 : Calc {
31      override fun sum(x: Int, y: Int) {
32          println("Sum is " + x.plus(y))
33      }
34  }
35
36  fun main(args: Array<String>) {
37      var m1 = Math1()
38      m1.sum( x: 23,  y: 45)
39      m1.display()
40
41      var m2 = Math2()
42      m2.sum( x: 23,  y: 45)
43  }
```

Generic Class

- A generic class allows a single function to handle many different types of data.
- Help reduces code duplications.

```
3   // non-generic class
4   class Permission {
5       var username: String? = null
6       var password: String? = null
7   }
8
9   // generic class
10  class GenPermission<T> {
11      var username: T? = null
12      var password: T? = null
13  }
```

```
15  fun main(args: Array<String>) {
16      var perm1 = Permission()
17      perm1.username = "William"
18      perm1.password = "abc123"
19
20      var genperm1 = GenPermission<Int>()
21      genperm1.username = 123
22      genperm1.password = 456
23
24      var genperm2 = GenPermission<String>()
25      genperm2.username = "Bob"
26      genperm2.password = "password"
27
28      var genperm3 = GenPermission<Boolean>()
29      genperm3.username = true
30      genperm3.password = false
31
32  }
```

Generic Class Usage

Enum Class

- An enum class is a special data type that enables a variable to be a set of predefined constants.

```
3   // enum class is a type that contain predefined set of values
4
5   enum class Colleges(desc: String) {
6       ITCollege( desc: "IT College"),
7       BusinessCollege( desc: "Business College"),
8       ArtsCollege( desc: "Arts College"),
9       EngineeringCollege( desc: "Engineering College")
10  }
11
12  enum class Days {
13      Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
14  }
15
16  fun main(args: Array<String>) {
17      // define a variable of type enum
18      var major = Colleges.ArtsCollege
19      println("Major: $major")
20
21      var today = Days.Monday
22      println("Today is $today")
23  }
```

Class Variables

- Also known as companion object.
- The variables belongs to the class instead of object.
- To access class variables, prefix with the class name.

```
3   // class variable - variable belongs to class its
4   // use companion object block to define class variable
5
6   class Game {
7       companion object {
8           val gamesPlayed = 10
9       }
10      var id: Int? = null
11  }
12
13  fun main(args: Array<String>) {
14      var game1 = Game()
15      game1.id = 1
16
17      var game2 = Game()
18      game2.id = 2
19
20      // access class variable via class name
21      println("Total games played: " + Game.gamesPlayed)
22
23  }
```

Access Modifiers

- Public
  - Default, accessible from all classes.
- Private
  - Accessible within the same class only.
- Protected
  - Private but accessible by subclass(es).

```
3  open class Employee {
4      protected var id: Int? = null
5      var name: String? = null
6      var email: String? = null
7      var salary: Double? = null
8
9      open fun display() {
10         println("$id $name $email $salary")
11     }
12 }
13
14 class Programmer : Employee() {
15     private var project : String? = null
16
17     override fun display() {
18         println("$id $name $email $salary $project")
19     }
20 }
```

Kotlin Collections - HashMap

- An associative array, use key-value paring.
- Key must be unique and point to one value only.
- Basic functions;
  - put()
  - get()

```
3  fun main(args: Array<String>) {
4      // create a hashmap
5      var myHashMap = HashMap<String, String>()
6
7      myHashMap.put("first", "One")
8      myHashMap.put("second", "Two")
9  //    myHashMap.put(3, "Three")   // error
10     myHashMap.put("second", "TWO")  // overwrite previous value
11
12     println(myHashMap.get("first"))
13     println(myHashMap.get("second"))
14
15     for (key in myHashMap.keys){
16         println(myHashMap.get(key))
17     }
18 }
```

Kotlin Collections - ArrayList

- A dynamic array.
- Automatically extends and shrinks.
- Can be resized.

```
3  fun main(args: Array<String>) {
4      // create a list to store int numbers
5      var numList = ArrayList<Int>()
6      numList.add(12)
7      numList.add(34)
8      numList.add(12)
9  //    numList.add("test") // error
10
11     println("Size: " + numList.size)
12     println("First item: " + numList.get(0))
13
14     for (num in numList) {
15         println(num)
16     }
17
18     numList.remove( element: 12)
19     println("Size: " + numList.size)
20     println("First item: " + numList.get(0))
21
22     // immutable list
23     var iList = listOf(1, "Android", 500)
24 //    iList[0] = 2    // error
25     println(iList[0])   // ok
26
27     // mutable list
28     var mList = mutableListOf(1, "Android", 500)
29     mList[0] = 2    // ok
30     println(mList[0])
31 }
```

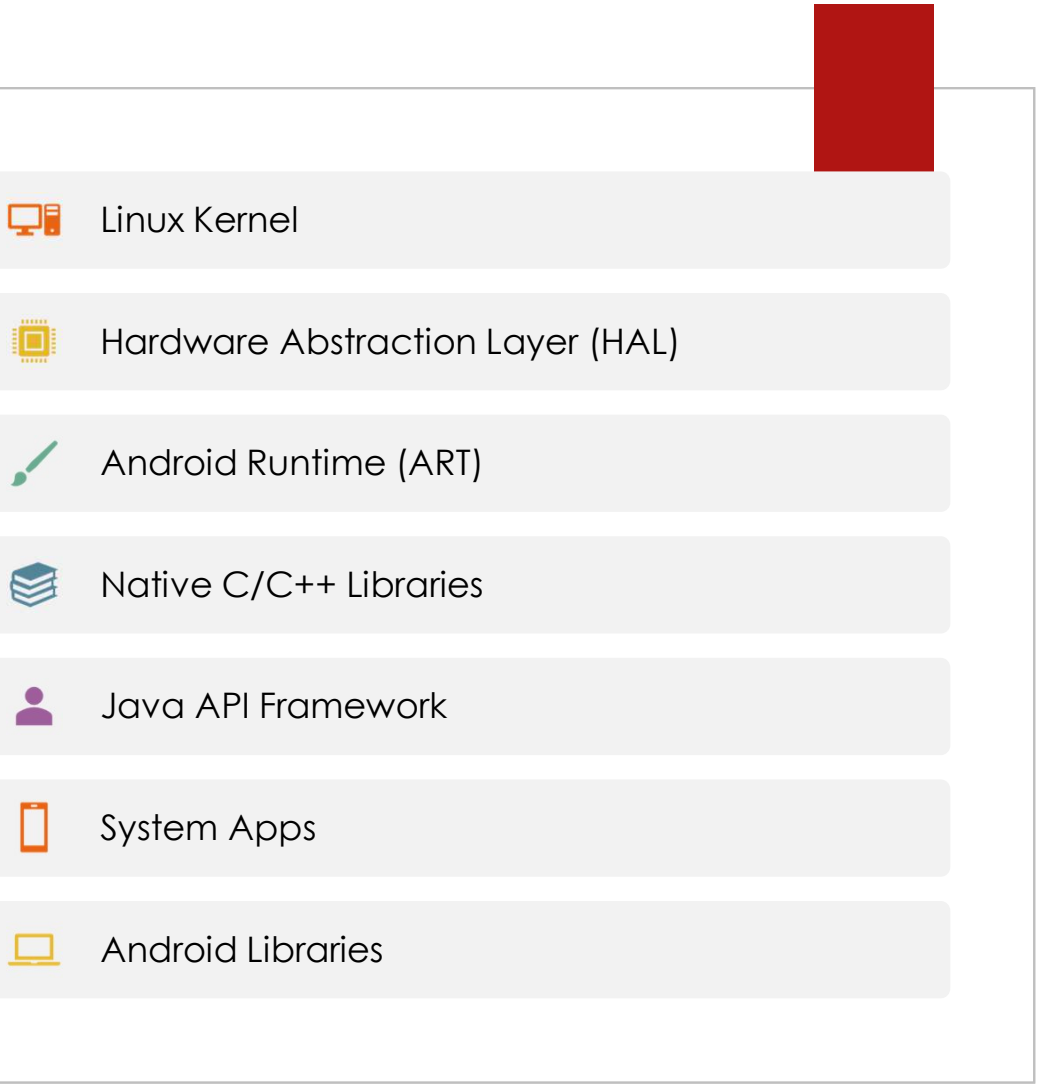
```kotlin
fun main(args: Array<String>) {
    // create a list to store int numbers
    var numList = ArrayList<Int>()
    numList.add(12)
    numList.add(34)
    numList.add(12)
//    numList.add("test") // error

    println("Size: " + numList.size)
    println("First item: " + numList.get(0))

    for (num in numList) {
        println(num)
    }

    numList.remove( element: 12)
    println("Size: " + numList.size)
    println("First item: " + numList.get(0))

    // immutable list
    var iList = listOf(1, "Android", 500)
//    iList[0] = 2    // error
    println(iList[0])   // ok

    // mutable list
    var mList = mutableListOf(1, "Android", 500)
    mList[0] = 2    // ok
    println(mList[0])
}
```

Method mutableListOf

Android Framework and Android Studio

LESSON 4

Android Platform Architecture



Components of Android Applications

ACTIVITIES

VIEWS

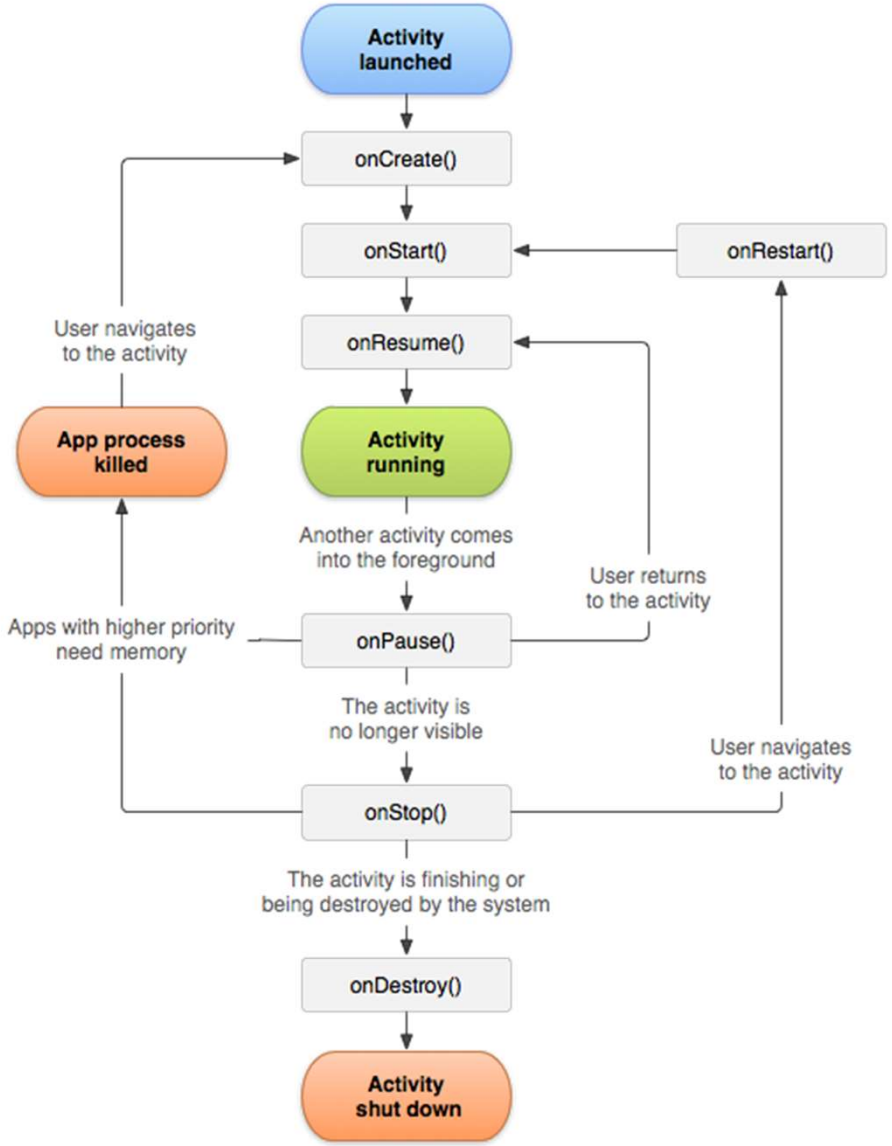
SERVICES

CONTENT PROVIDERS

INTENT

BROADCAST RECEIVERS

NOTIFICATIONS



Application Life Cycle

Types of Android Processes and Their Priorities

Type	Description	Priority
Foreground	Applications with components currently interaction with users. Android will keep this process responsive.	High
Visible	Visible, but inactive and responding to user events.	Medium
Service	Processes hosting services that have been started, ongoing services without a visible interface.	Low
Background	Aren't visible and has services that hasn't started.	Low
Empty	Does not hold any active application component.	Low

Android Application Development

Android SDK

Integrated Environment Development (IDE)

Android Studio

Gradle

Lab 4: Creating Your First Application

Create Your First Android Application

Create an Android Device

Build a Simple Calculator Application

Creating User Interface

LESSON 5

Android Project Structure

- Project name
- Package name
- Project location
- Language
- Minimum SDK
- Gradle script
- AndroidManifest.xml

View

- All user interface elements in an Android application are built using View and ViewGroup objects.
- A View is a public class that draws something on the screen that the user can interact with.
- View is the based class for widgets, which are used to create interactive user interface components (buttons, text fields, etc)
- All of the views in a window are arranged in a single tree map.
- Can be added programmatically or by using the designer.

Creating a User Interface

Add a text box.

Add an image.

Add a check box.

Add a radio button.

Lab 5: Creating a Pizza Order Application

Create Your Application User Interface

Configure the Android Application Code

Run Your Application

Android Layouts, Styles, Themes and Menus

LESSON 6

Layouts

- Constraint Layout
- Linear Layout
- Relative Layout
- Table Layout
- Frame Layout
- ScrollView Layout

Android Styles and Themes

Android SDK allows developers to create the user interface style of their applications using XML resource files.

Defining the style of user interfaces involves specifying values for colors, fonts, dimensions, buttons, etc.

When a resource is used to define the style of a certain view it is called a **style**.

However, when the same style is applied on the whole activity or application it is called a **theme**.

Adaptive Icons

Android 8.0 (API 26) introduces adaptive launcher icons, which can display a variety of shapes across different device models.

For example, an adaptive launcher icon can display a circular shape on one OEM device, and display a square on another device.

Each device provides a mask, which the system uses to render all the adaptive icons of the same shape.

An adaptive launcher icon is also used in shortcuts, settings app, sharing dialogs, and the overview screen.

Lab 6: Android Application Layouts, Styles and Themes

Create Your Application Layout

Configure Your Styles and Themes

Configure Your App Icon

Toasts, Activities, Navigations and Views

LESSON 7

Context Class

Context is an abstract class (model) that is provided by Android SDK.

Context class contains information about activity or application.

Functions to get context;

getApplicationContext()
getBaseContext()
this keyword

Toast Class

Toast is a public class that is displayed to show a quick brief message to the user.

A toast provide a simple feedback about an operation in a small group.

It only fills the amount of space required for the message while the current activity remains visible and interactive.

Toasts automatically disappear after a timeout.

Toast duration constants;

LENGTH_LONG: 3.5 seconds
LENGTH_SHORT: 2 seconds

What is an Activity?

An activity is an entry point for interacting with the user.

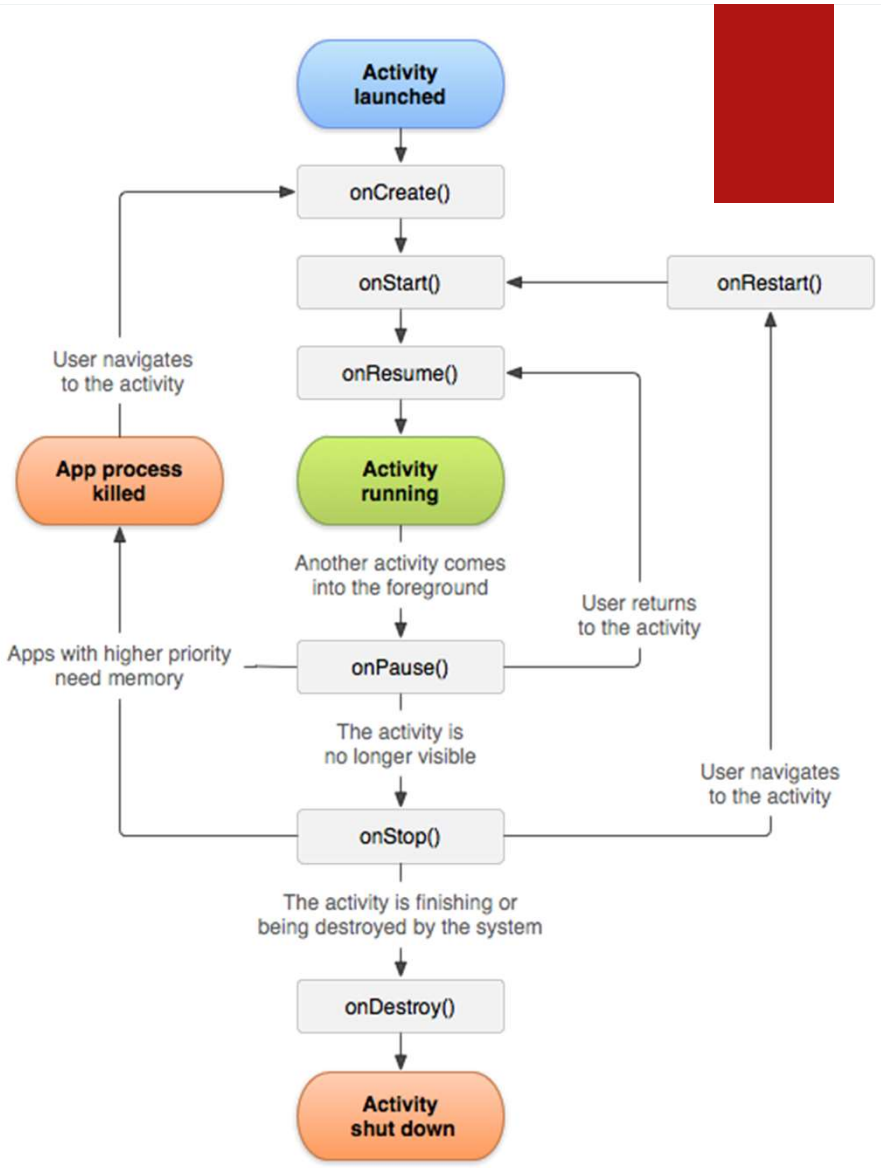
It represents a single screen with a user interface.

The apps consists of one or more activities, and the user can navigate between them through buttons, menus, images, back button and other navigation tools.

An activity in Android apps can be configured to be the main activity by configuring the AndroidManifest.xml file.

Activity Lifecycle

- To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks:
  - onCreate(),
  - onStart(),
  - onResume(),
  - onPause(),
  - onStop(),
  - and onDestroy().
- The system invokes each of these callbacks as an activity enters a new state.



Lifecycle Callbacks – onCreate()

You must implement this callback, which fires when the system first creates the activity.

In the onCreate() method, you perform basic application startup logic that should happen only once for the entire life of the activity.

For example, your implementation of onCreate() might bind data to lists, associate the activity with a ViewModel, and instantiate some class-scope variables.

This method receives the parameter savedInstanceState, which is a Bundle object containing the activity's previously saved state. If the activity has never existed before, the value of the Bundle object is null.

Lifecycle Callbacks – onStart()

When the activity enters the Started state, the system invokes this callback.

The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.

For example, this method is where the app initializes the code that maintains the UI.

Lifecycle Callbacks – onResume()

When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the onResume() callback.

This is the state in which the app interacts with the user.

The app stays in this state until something happens to take focus away from the app.

Such an event might be, for instance, receiving a phone call, the user's navigating to another activity, or the device screen's turning off.

Lifecycle Callbacks – onPause()

The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed).

It indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode).

Use the onPause() method to pause or adjust operations that should not continue (or should continue in moderation) while the Activity is in the Paused state, and that you expect to resume shortly.

There are several reasons why an activity may enter this state. For example: Some event interrupts app execution, as described in the onResume() section. This is the most common case.

Lifecycle Callbacks – onStop()

When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the onStop() callback.

This may occur, for example, when a newly launched activity covers the entire screen.

The system may also call onStop() when the activity has finished running and is about to be terminated.

Lifecycle Callbacks – onDestroy()

onDestroy() is called before the activity is destroyed.

The system invokes this callback either because:

the activity is finishing (due to the user completely dismissing the activity or due to finish() being called on the activity), or	the system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)

Android Intent

An Intent is an object that provides runtime binding between separate components, such as two activities.

The Intent represents an app's intent to do something.

You can use intents for a wide variety of tasks, but in this lesson, your intent starts another activity.

Intent Structures

Action

The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.

Data

The data to operate on, such as a person record in the contacts database, expressed as a Uri.

Intent Structures

Category

Gives additional information about the action to execute. For example, CATEGORY_LAUNCHER means it should appear in the Launcher as a top-level application.

Type

Specifies an explicit type (a MIME type) of the intent data.

Intent Structures

Component

Specifies an explicit name of a component class to use for the intent.

Extras

This is a Bundle of any additional information. This can be used to provide extended information to the component

Intent Resolution

Explicit Intents

have specified a component (via setComponent(ComponentName) or setClass(Context, Class)), which provides the exact class to be run.

Implicit Intents

have not specified a component; instead, they must include enough information for the system to determine which of the available components is best to run for that intent.

Navigating Between Activities

- Normally, when you run your Android app, the main activity will appear as Main Activity.
- By default, the main activity's name is MainActivity and its name is configured within AndroidManifest.xml file to startup first.
- The main activity includes all the navigation tools such as buttons, images, and others to browse or navigate your app which is usually consists or more than one activity.
- Android Activity is a class which inherits all parameters from the AppCompatActivity class.

Passing Data Between Activity

It is very likely that you want to transfer some data to the activity you want to start through Intent.

Android SDK provides this option by using extras methods.

To add data to extras use the putExtra() function.

To get/read data from extras use the getExtra() function.

Android Views

This class represents the basic building block for user interface components.

View is the based class for widgets to create UI such as buttons, text fields, layout, etc.

All of the views in a window are arranged in a single tree.

List View

ListView is a view group that displays a list of scrollable items.

The list are automatically inserted using an Adapter that pulls content from a source such as an array or database query.

Adapter will convert each item result into a view that is placed into the list.

Lab 7: Configuring Android RecyclerView

Adding a RecyclerView to an Activity

Creating a CardView

Creating Your RecyclerView Adapter

Adding Data to Your RecyclerView

Running and Testing Your RecyclerView

Adding Event Listeners to Each RecyclerView Row

Android Dialogs, Snackbar, Menus, WebView and Notifications

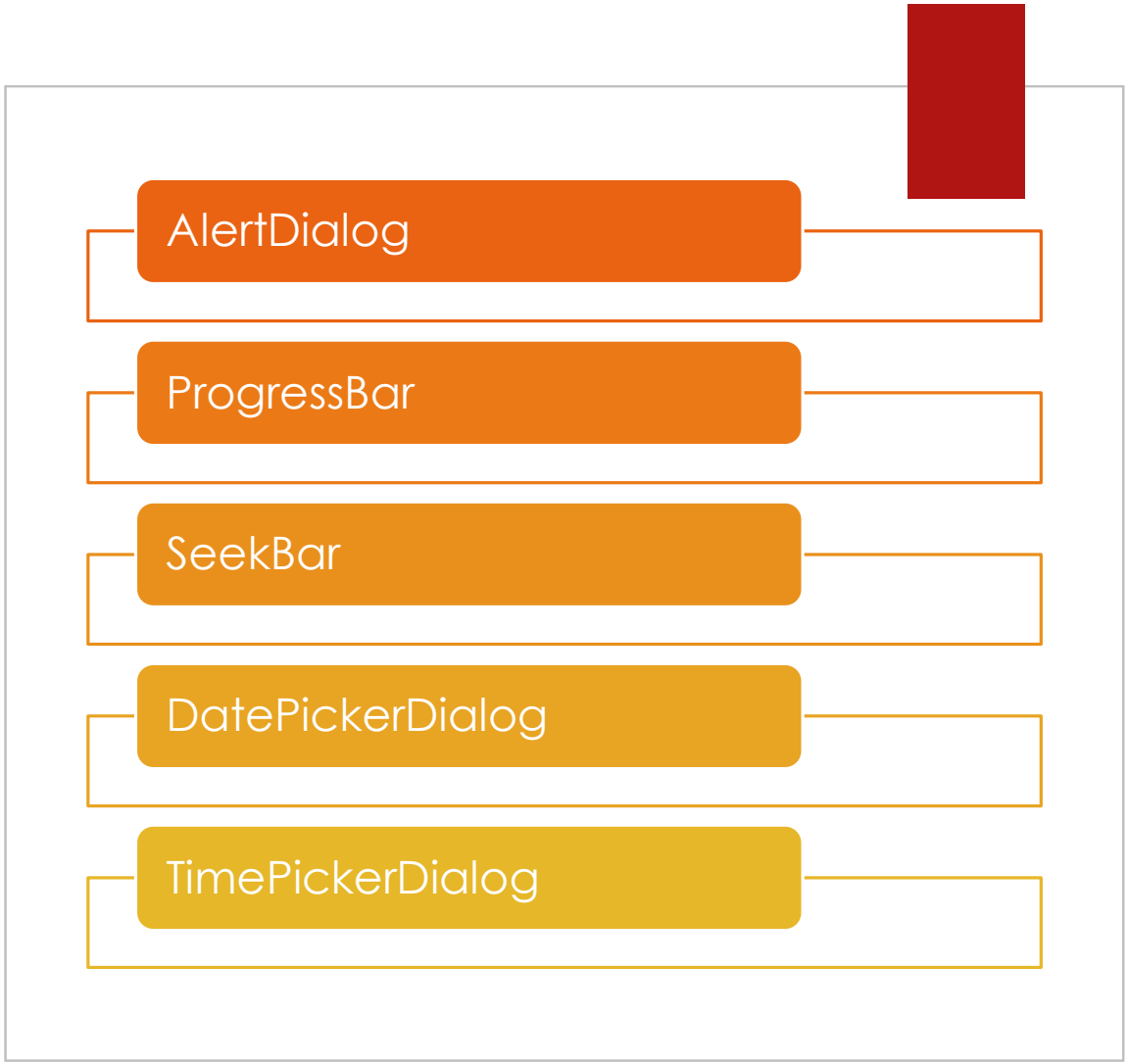
LESSON 8

Android Dialogs

A dialog is a small window that prompts the user to make a decision, enter additional information or give a feedback.

It does not take the full screen but part of it and asks users to take a specific action before they can proceed.

Dialog Sub-Classes



Menus

Menus are common UI components used to provide user actions and other options in your activities.

As on Android 3.0, menus are provided as a part of an action bar of the activity.

Each activity in an application can get its own options-menu.

The user can access the options menu from the action bar by pressing on the three dots icon.

WebView

You can embed a web browser inside your application using web views.

A WebView class is a crucial component for many applications that need to display a website without worrying about using Android's native views.

Android Notifications

- A notification is a message you display to the user outside of you're app's regular user interface.
- When your app sends a notification to the Android OS, the notification manager service (Android service) receives your app notification and issues it.
- The notification appears first as an icon in the **notification area**.
- To the details of the notification, the user should open the **notification drawer**.
- Both notification area and drawer are system-controlled areas that the user can view at any time.

Android Notifications

- In supported launchers and on devices running Android 8.0 (API level 26) and higher;
  - Apps can display notification badges on app icons.
  - Users can also long-press on an app icon to glance at the notifications associated with a notification badge.

Creating an Android Notification

- A small icon, set by setSmallIcon()
- A title, set by setContentTitle()
- Detail text, set by setContentText()
- Notification Channel
  - On Android 8.0 (API level 26) and later versions, a valid notification channel ID is set by setChannelId() or is provided in the NotificationCompat.Builder constructor when creating a channel.
  - You should configure a notification channel code before you configure notification message, because the configuration of the notification message needs to use the notification channel id which have configured first.

Notification Channel

- Starting in Android 8.0 (API level 26), all notifications must be assigned to a channel.
- For each channel, you can set the visual and auditory behavior that is applied to all notifications in that channel.
- Then, users can change these settings and decide which notification channels from your app should be intrusive or visible at all.
- When you target Android 8.0 (API level 26), you must implement one or more notification channels.
- If your targetSdkVersion is set to 25 or lower, when your app runs on Android 8.0 (API level 26) or higher, it behaves the same as it would on devices running Android 7.1 (API level 25) or lower.

Lab 8: Configuring Android Web Browser, Menu and Notification

Configuring Android Web Browser

Adding Android Menu

Creating a notification channel and a notification message

Android Storage, SQLite and Content Providers

LESSON 9

Android Storage Options

Shared Preferences

- This technique stores application-specific primitive data in key-values pairs.

Internal Storage

- Store private data on the device memory using file I/O technique. This data will not be accessible by other applications.

External Storage

- Store public data on the shared external storage (such as SD card).

Network Connection

- Store data on the web using your own network server.

SQLite Databases

- Store structured data in a private database on the phone.

SQLite Database in Your Application

- Default directory /data/data/APP_NAME/databases/DATABASE_FILE
- SQLiteOpenHelper
  - This is a helper class to manage database creation and version management.
- SQLiteDatabase
  - SQLiteDatabase exposes methods to manage an SQLite database.
  - It has methods to create, execute SQL commands and perform other common database management tasks.
- Cursors
  - The Cursor interface provides random access-write access to the result set returned by a database query.

Content Providers

Content Providers are one of the building blocks of Android applications.

They manage access to central repository of data.

Usually Android applications keep the data private and hide it from other applications.

However, sometimes you might need to share the data with other applications.

For this purpose, content providers are mainly used.

Should each Android app with SQLite database have a content provider?

You need to create a content provider only if you want to share your application data with other applications.

You do not need a content provider to use SQLite database if the use occurs completely within your application.

Creating a Content Provider

File Data

- Any file such as video, an image or an audio.
- App stores the files in a private memory.

Structured Data

- Any databases, arrays, etc.
- Data stored compatible with tables of rows and columns.
- SQLite database is the common way for storing this type of data.

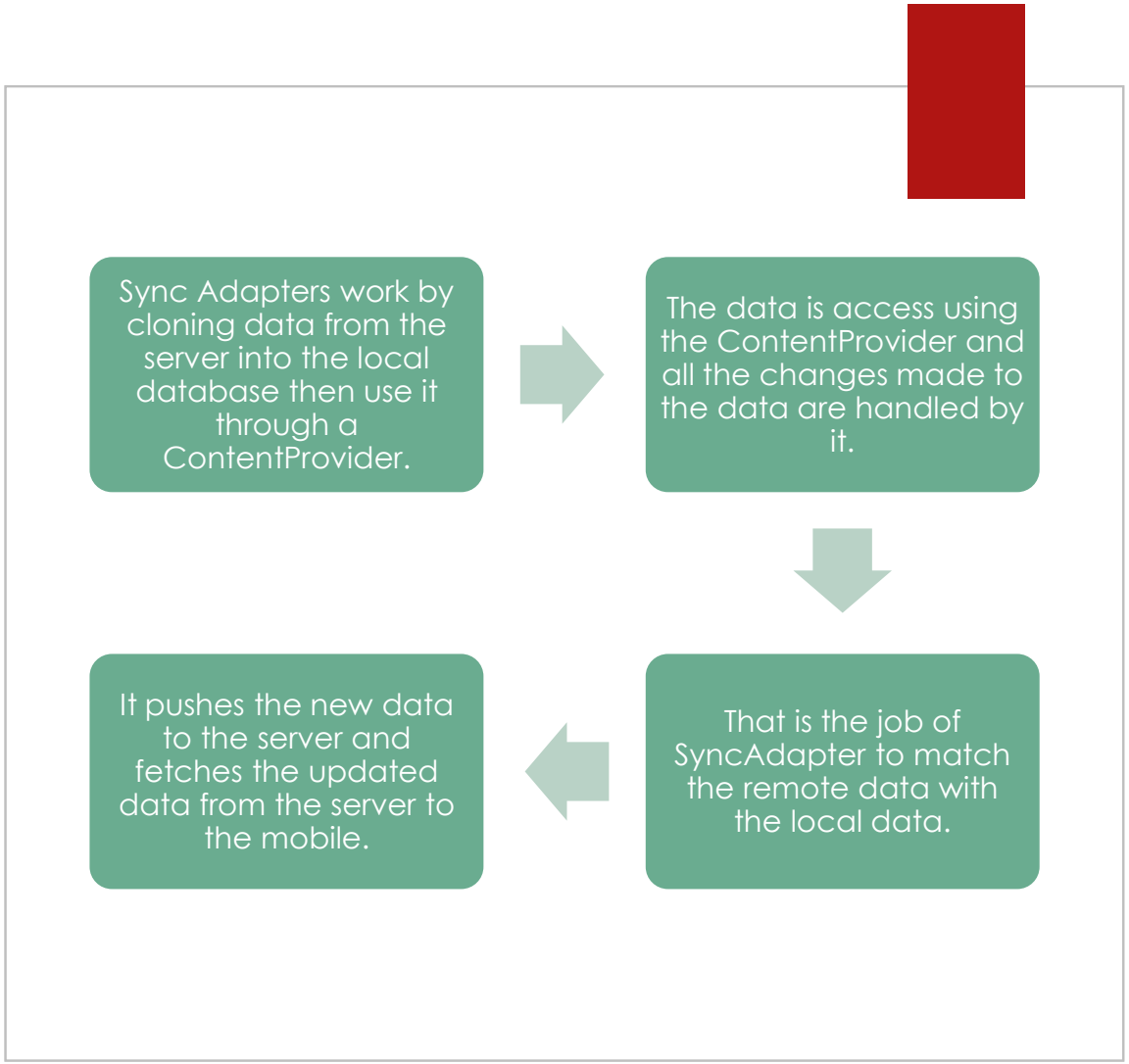
Sync Adapters

There is often a need to send data to and from a mobile device to a server.

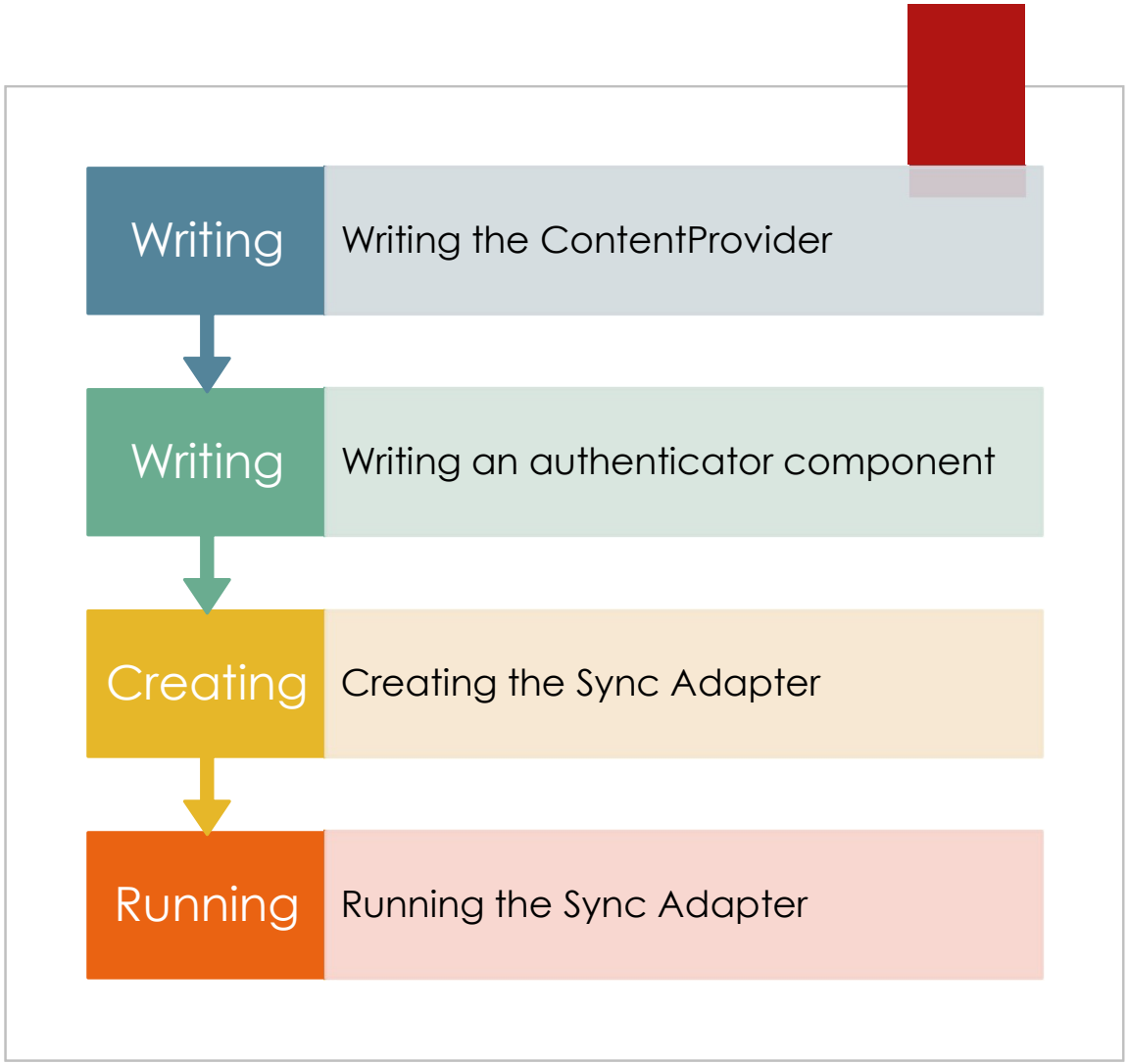
Use Android's sync adapter framework which is very useful in data fetching, synchronizing and scheduling updates between the mobile and the server.

Sync adapter run asynchronously.

How Sync Adapters Work



Steps to create SyncAdapter framework



ORMLite

Object-Relational Mapping

A third-party library that provides a simple and lightweight functionality of persisting objects to SQL databases.

Advantages of ORMLite

Simple setup of classes by adding annotations.

Easy construction of simple and complex queries.

Support a wide variety of database technologies.

Automatic generation of SQL to create and drop tables.

Making native calls to Android SQLite database APIs.

How-to ORMLite

- @DatabaseTable
- @DatabaseField

```
7   @DatabaseTable(tableName = "person")
8   data class Person(
9       @DatabaseField(generatedId = true)
10      var accountId: Int? = null,
11      @DatabaseField
12      var name: String? = null
13  )
```

Lab 9: SQLite Databases and Content Providers

Creating an SQLite database

Querying a SQLite DB table

Creating Content Provider

Utilizing ORMLite Library

Location-Aware Apps: Using GPS and Google Maps

LESSON 10

Introduction

Most of the mobile application nowadays rely on user's geolocation and web mapping services.

The GPS is considered one of the most accurate geolocation providers.

In order to increase users' experience of location awareness, geo-coordinates should be represented graphically which can be achieved using web-mapping services such as Google Maps.

What is GPS and how does it work?

- The GPS is a navigation system based on satellite.
- The GPS coordinates depend on the latitude and longitude values.
- The connection between a GPS satellite and the receivers (i.e. smart devices) is a one-way connection.
- The connection works through the use of a procedure called Trilateration.
- NAVSTAR is one of the mostly used satellite in all mobile devices.

Other Location Service Providers

- Both Cell-ID and WI-FI can be used as location service providers.
- Android SDK options;
  - LocationManager.GPS_PROVIDER
  - LocationManager.NETWORK_PROVIDER
- Determining which provider to use relies on three criteris;
  - Accuracy
  - Speed
  - Battery consumption

Steps…

- Configuring Google Maps
- Map Fragment
- Getting a Google API key
- Adding a Google Map Marker
- Methods to Capture a User's Location
  - Location Manager
  - Location Listener
- Reverse Geolocation and Geocoder Class
- JSON and APi

Lab 10: Location-Aware Apps using the GPS and Google Maps


Creating a Search App Interface


Getting a Google API key


Configuring Your App to Use Your API Key


Configuring User App's Permission


Capturing Users' Location


Finding the Nearest Restaurants to a User's Current Location

THE END

Iverson Online Evaluation

https://feedback.iverson.com.my

Class ID:

THANK YOU! ☺