HSBC

Android App Development using Kotlin

Infosys®
Navigate your next

Android App Development Using Kotlin

HSBC | Infosys®

What is Mobility ?

Mobility

- “Mobility” – refers to the ability of the application to be rendered on a mobile device such as Smart Phone, Tablet, Smart Watches , Smart TVs and so on.
- Mobile apps can be downloaded from a play store or can be accessed via the mobile browsers
- Mobile apps are used across various areas such as Finance, Education ,Travel ,Healthcare, Sports and so on
- Mobile apps are categorized into Consumer Mobile Apps and Enterprise Mobile Apps

Mobile App Development Approaches

Native Approach

Cross/Hybrid Approach

Mobile Web Approaches

The right approach?

- Choosing an approach largely depends on an app's requirements.
- Some sample scenarios are illustrated in the table below

Native Platforms	Cross Platform apps	Mobile Web apps
Rich user experience	Device agnostic	Device Agnostic
High Native feature access	High code portability	Always online
Advanced graphics	Lower cost and rapid development	Cost effective
Offline support	Single code base	Use only web technologies

- The choice between Native apps, Mobile Web apps and Hybrid apps is always a tradeoff based on the app requirements

Starting Android

What is Android?

- Product of OHA (Open Handset Alliance) led by Google.
- Software Platform that provides a framework to develop applications for mobile devices
- It includes an OS, middleware and key applications.
- Layered platform built on Linux kernel.

Android Platform Architecture

App Development Procedure

1. Android API
2. Programming Language
3. Data Handling
4. Test, Sign and Packaging
5. Publishing

Unboxing Android

Home Screen

- Customizable space
- Holds folders, widgets and shortcuts
- Contains a favorite tray that houses all the important shortcuts

All Apps screen

- Displays the whole set of apps and widgets that are installed in the phone

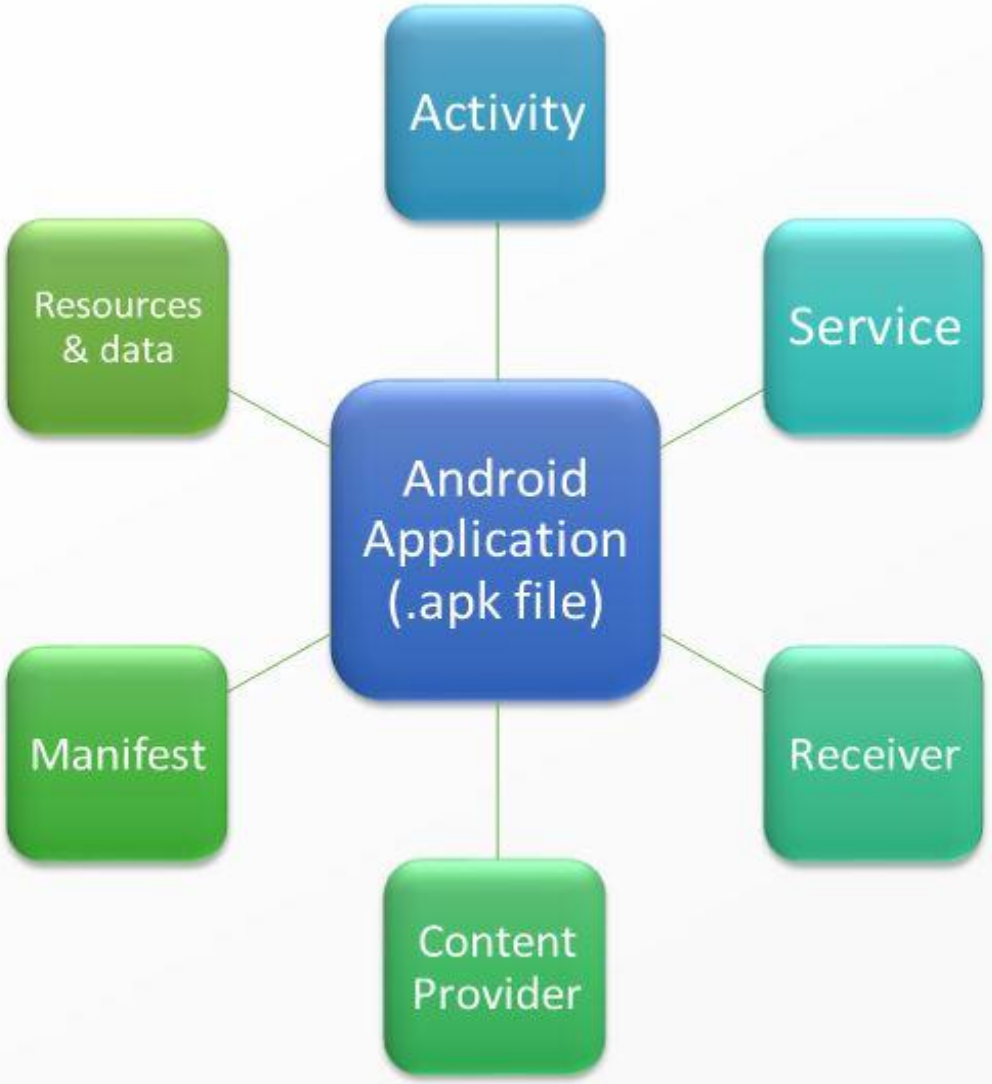
Settings Screen

- Used for changing the behavior of the apps in the device
- Gives the user some control over the installed applications as well as on the working of the device

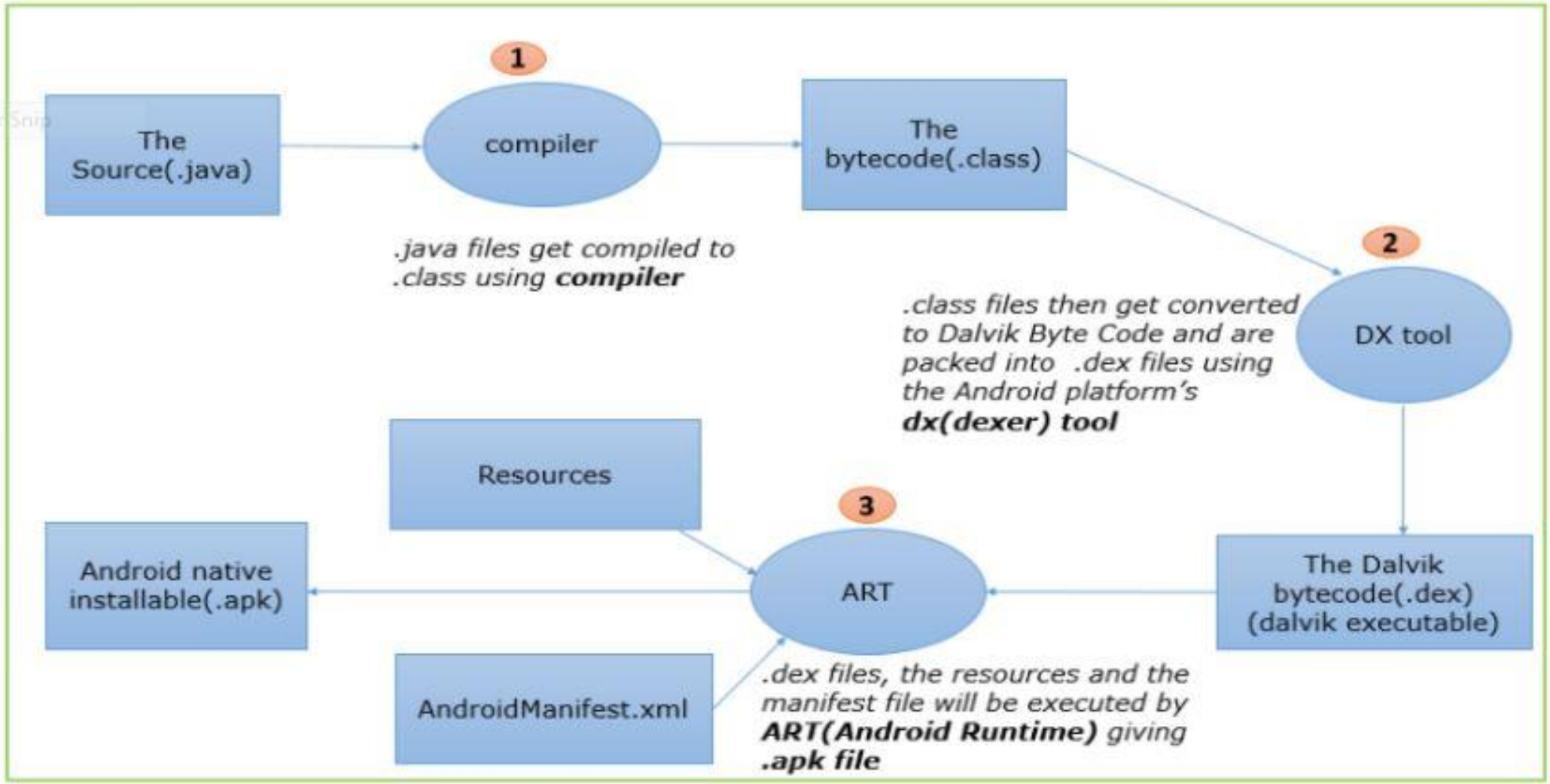
Building Blocks of an Android App

Building Blocks

- Activity
- Services
- Content Provider
- Broadcast Receiver

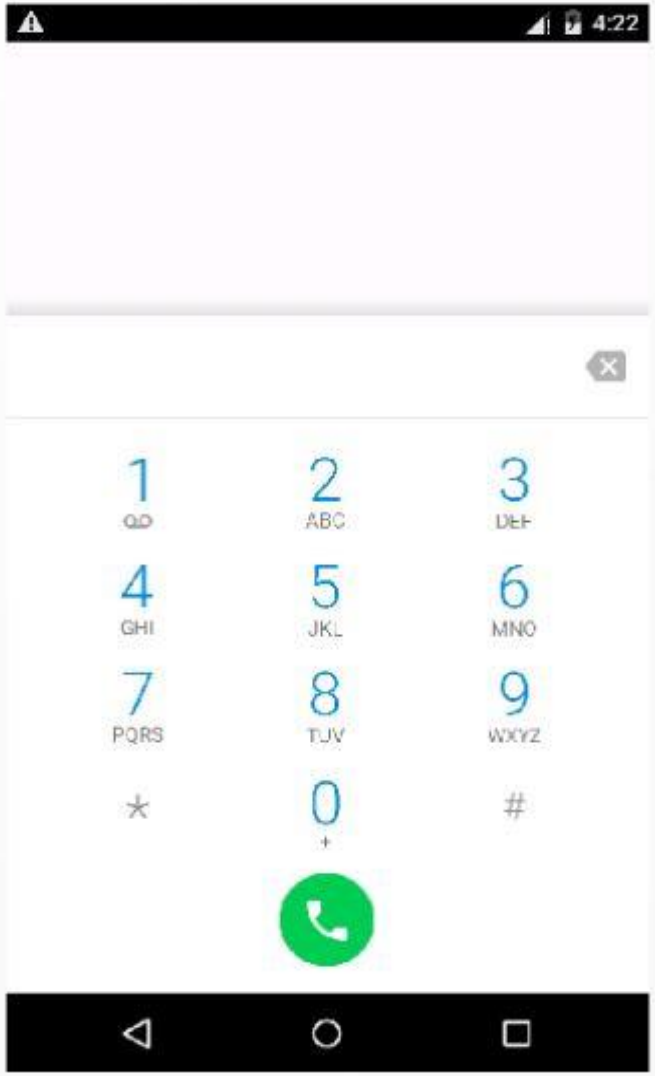


Behind the Scenes



Activity

- Provides user interface
- Each activity has a window whose content is provided by hierarchy of views (e.g. button, scroll bar, text field etc.)
- Activities can be invoked from other apps.

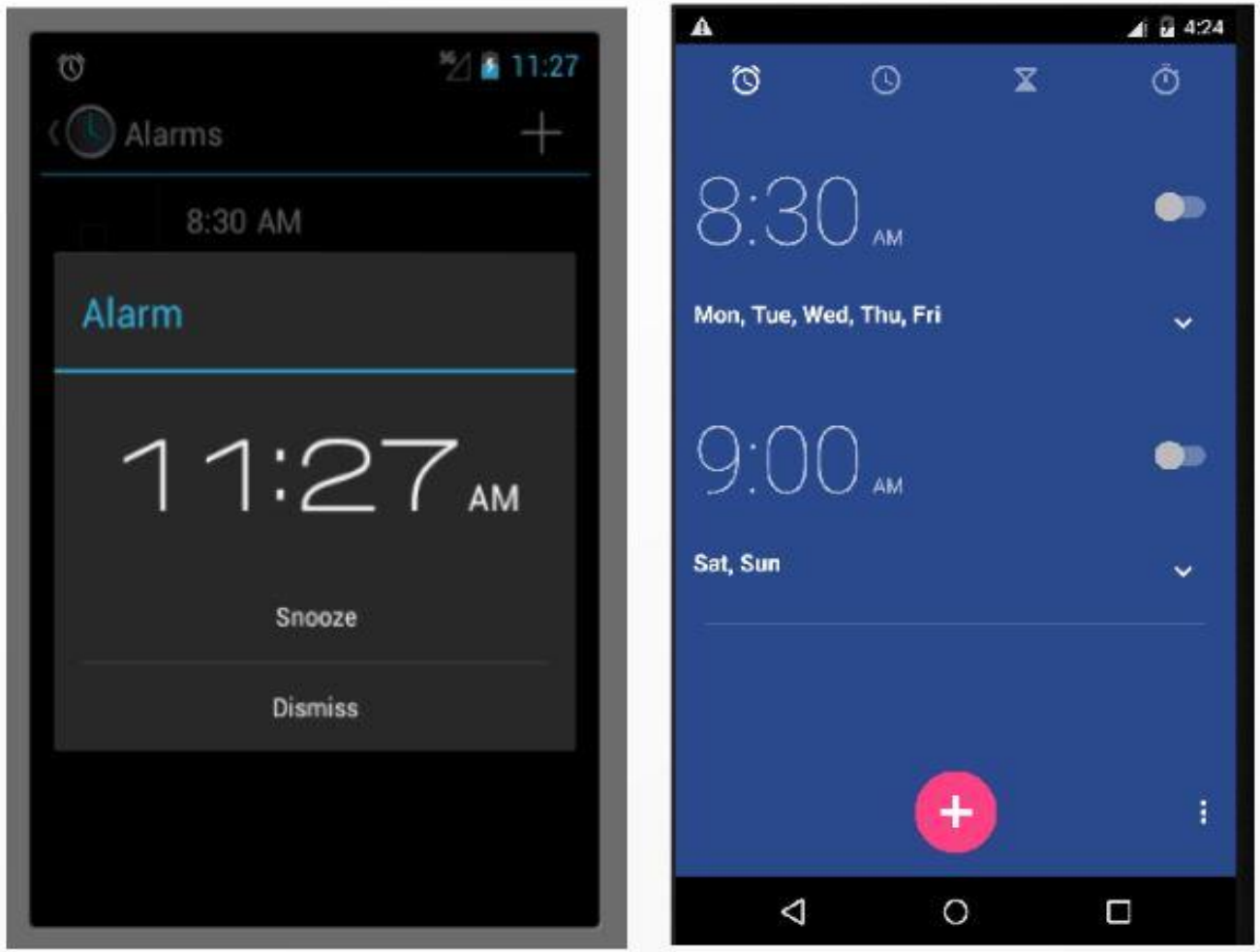


Service

- Does not have a visual interface
- Runs in the background, usually for long durations.

- Can run indefinitely or be bounded by the lifetime of the calling activity

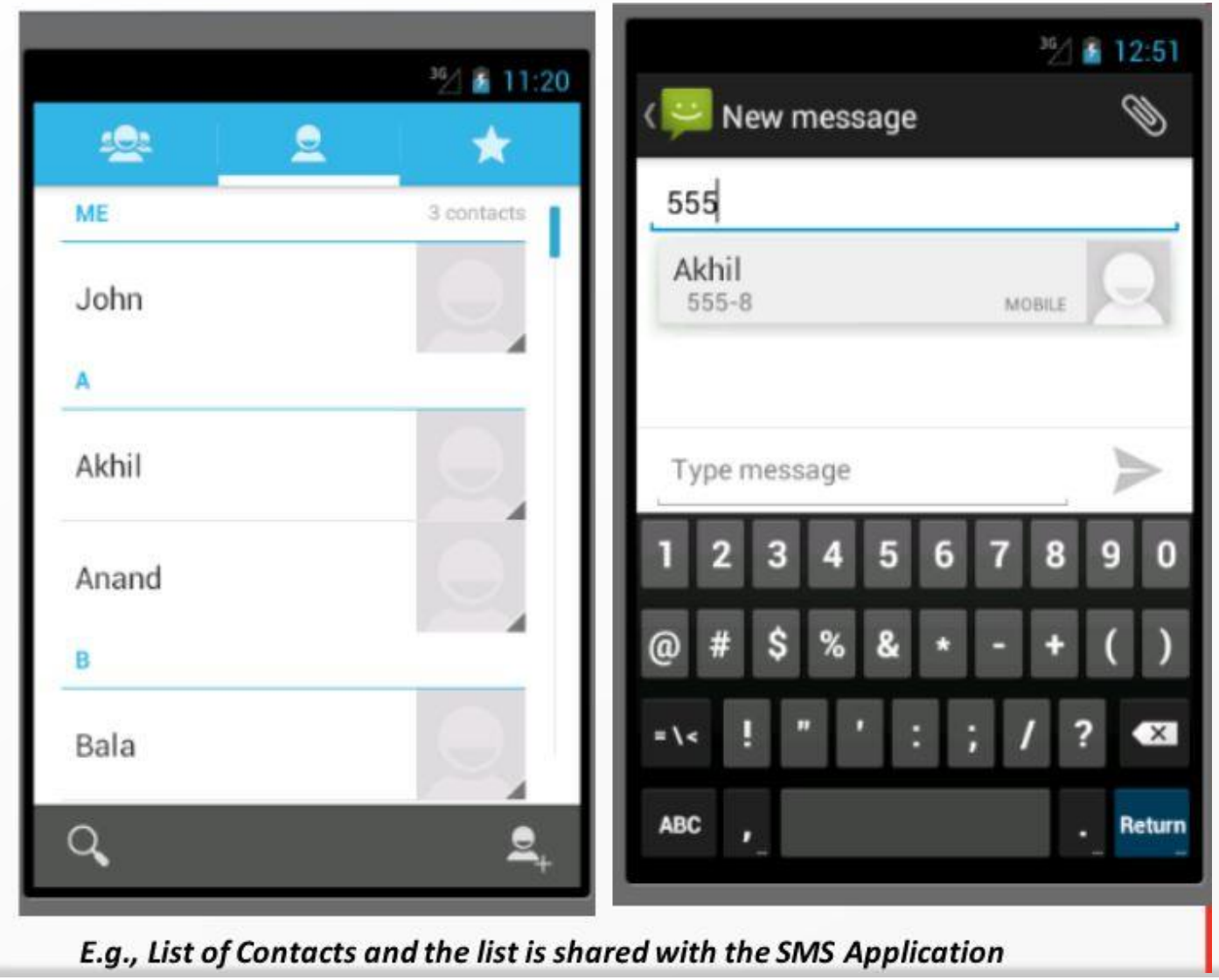
- Allows IPC between apps



Service provides an interface for communication via notifications. E.g., to stop / pause the music player or to snooze or turn off alarm.

Content Provider

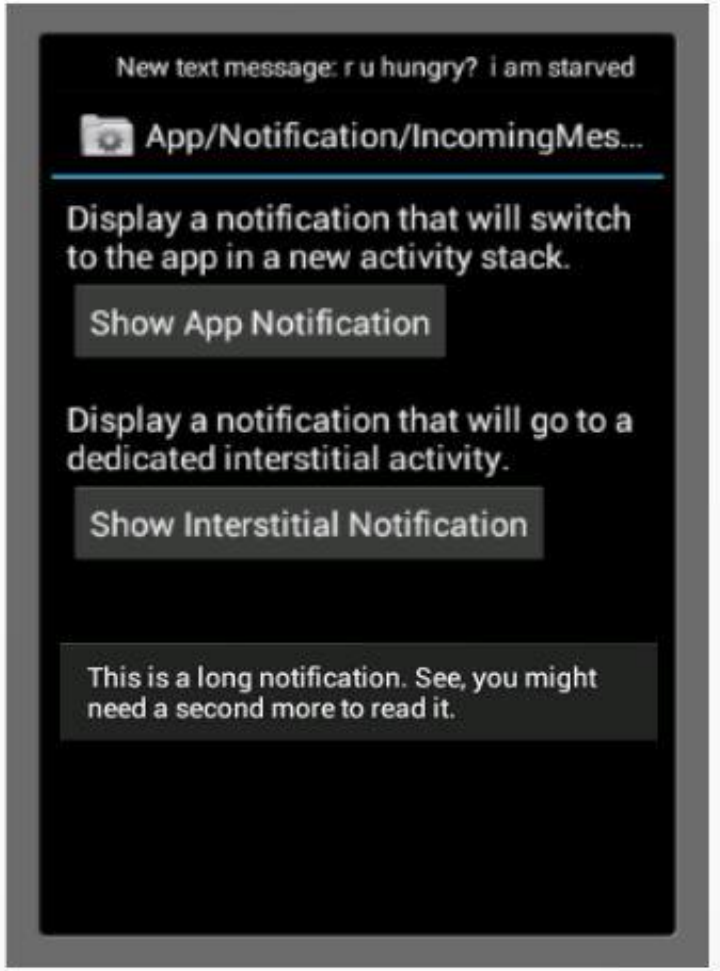
- One application's data is made available to another application by content provider.
- Application's data can be stored in files/SQLite database etc.



E.g., List of Contacts and the list is shared with the SMS Application

Broadcast Receiver

- Receive and respond to system-wide broadcast announcements
- It will continuously listen for intents that are broadcasted by some other activity or app
- Has no UI



E.g. change in time zone, battery low message, notification when USB storage is turned off etc.

Development Environment and Tools

Development Environment

- **Android Studio** is an intelligent code editor capable of advanced code completion, refactoring and code analysis.

- Android Studio comes with Android SDK tools, an IDE, Android 9.0 platform(Pie) and Android 9.0 emulator system image with Google APIs

Tools

The various tools that comes with Android Studio IDE are:

- SDK Manager : A tool that enables the developer to build applications for Android platform.
- AVD Manager : A tool used to create and manage the virtual device or emulator.
- Logcat: A command-line tool for logging of system messages, stack traces and error messages.
- ADB: A tool used to manage either an emulator instance or an actual Android device
- Gradle: A build tool used to build android packages (.apk files) by managing dependencies and providing custom build logic.

Why and What is Kotlin?

History

- Unveiled by JetBrains in 2011
- Named after Island – Kotlin
- Feb, 2016 - Stable version (v1.0) was released
- May, 2017 - Official Android Development Language

What is Kotlin?

- Static typed programming language
- Can be used for –
  - Android Development
  - Server-side apps
  - Web Development
  - Desktop Applications
- Can be transpiled to JavaScript
- Use JVM to execute files – JVM Language

Why Kotlin?

- More Powerful and Flexible than Java
- Procedural/Functional + Object Oriented Language
- Avoid Null Pointer Exception
- Support all the features needed in Java
- Interoperable with JAVA
- Less Code – Less Error – More Readable

Popular Apps with Kotlin

Source - Wikipedia

Basics of Syntax

- Semi colon is not mandatory
- No concept of Static
- No implicit type conversion
- Comments are same as Java
- Output
  - print() and println()
- Input
  - readLine()

Variables

- Two Types
  - Var (Variable)
    - Mutable Reference
  - Val (Value)
    - Immutable Reference
- Variable must have a type annotation OR be initialized

```
var str = "My World"

val myString = "Testing"
```

```
var myString1 = "HSBC"
var myString2 : String
myString2 = "Infosys"
```

Datatypes

- No primitive datatypes in Kotlin
- Built-in types in Kotlin
  - Numbers
    - Byte, Short, Int, Long, Float, Double
  - Characters
    - Char
  - Booleans
    - true, false

Arrays

- Container that holds Data of Single Type
- Fixed Length
- Mutable in Nature
- var myArray = Array<Int>(3) { 0 }

  myArray[2] = 22

  myArray[1] = 11

Array
Properties – size
Methods – get() set()

Conditional Statement

- If Block

```
if(str == ""){
    print("Empty String")
}
```

- If-Else Block

```
if(str == ""){
    print("Empty String")
}else{
    print(str)
}
```

- If-Else If-Else Block (ladder)
- Nested If-Else

```
if(str == ""){
    print("Empty String")
}else if(str == " "){
    print("String with one Space")
}else{
    print(str)
}
```

Conditional Statement (Cont.)

- “if” is an expression, and not a keyword.

```
var maxValue = if (a > b) a else b
```

- For Returning Value, by Default, Last Line will be returned.

```
var maxValue = if(a > b){
    print("a is Greater")
    a
}else{
    print("b is Greater")
    b
}
```

Conditional Statement (Cont.)

- When – Replacement for Switch-Case
- No Need of Break Statement
- When is also an Expression

```
when (num){
    1-> print("value of num is 1")
    2-> print("value of num is 2")
    else -> print("value of num is not determined")
}
```

```
var myString = when (num){
    1-> "value of num is 1"
    2-> "value of num is 2"
    else -> {
        print("unknown")
        "value of num is not determined"
    }
}
```

Range Operator

- Generates the Range of Values
- Operator – Double Dot
- Method – rangeTo()

```
var myRange = 1..6
for(num in myRange){
    print(num)
}
```

```
var myRange = 1.rangeTo( other: 6)
for(num in myRange){
    print(num)
}
```

Loops

- While Loop
- Do-While Loop
- For Loop (For-each loop)
  - Iterates through Ranges, Arrays, etc.

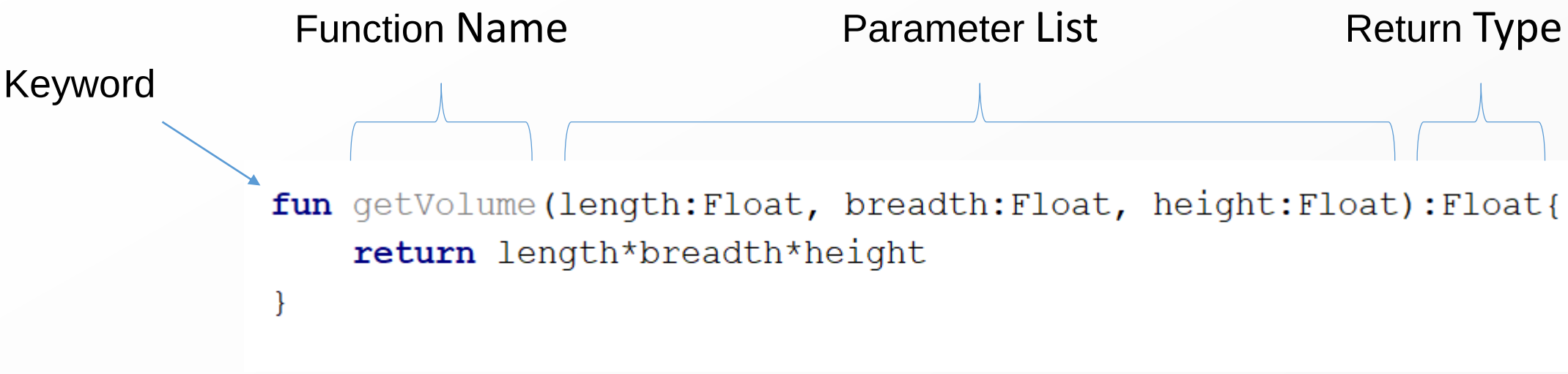
```
for(i in 0..4){
    print(i)
}
```

```
var i = 0
while(i<5){
    print(i)
    i++
}
```

```
var i = 0
do{
    print(i)
    i++
}while(i<0)
```

Functions

Syntax:



As Expression:

```
fun max(x: Int, y: Int): Int = if (x>y) x else y
```

String Interpolation

```
var arr = arrayOf(1,2,3,4)
var number = 10
```

Traditional Method:

```
print("Number is "+number+" and array length is "+arr.size)
```

Using String Interpolation:

```
print("Number is $number and array length is ${arr.size}")
```

Object Oriented Programming (OOP) using Kotlin

Class

- Syntax:
  - Class ClassName
  - Rest everything is Optional
- Properties “must” be Initialized.

```
class MyString{
    var size:Int = 0

    fun get(): Int{
        return size
    }
    fun set(size:Int){
        this.size = size
    }
}
```

Constructor

- Class can have
  - 1 Primary Constructor
  - 1 or More Secondary Constructor
- Primary Constructor can Declare Properties
- Secondary Constructor can only modify the Properties

Primary Constructor:

```
class Employee(var name: String){

}
```

Secondary Constructor:

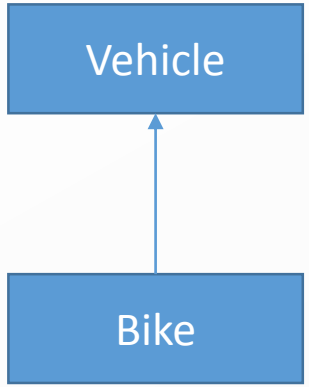
```
class Employee{
    var name = "Amit"
    constructor(name:String){
        this.name = name
    }

}
```

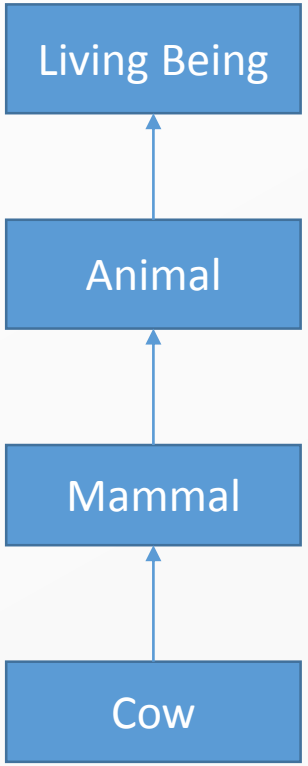
Access Modifier

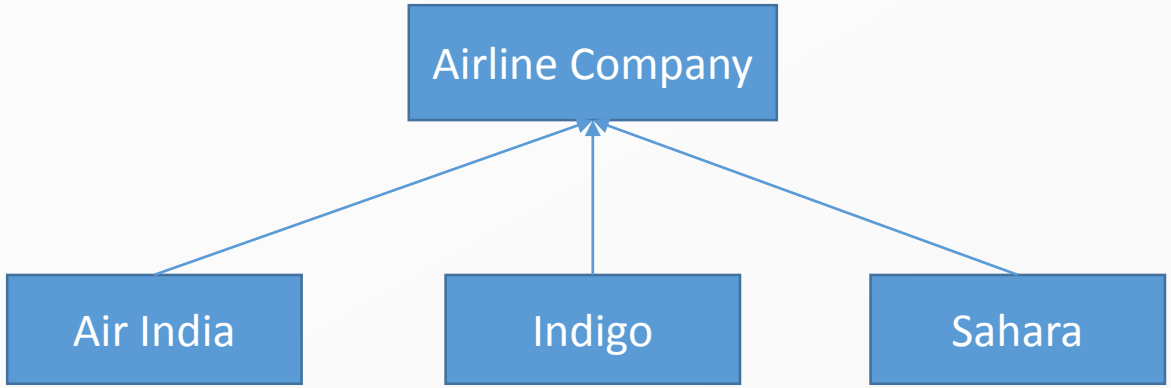
- 4 Types of Access Modifier:
  - Public – Visible to all
  - Internal – Visible to Module (Project)
  - Protected – Visible to Subclass
  - Private – Visible to File/Class
- Protected is not for top-level declaration

Inheritance

- “is-a” relationship between Classes
- Types of Inheritance:
  - Simple
  - Multi-Level
  - Hierarchical



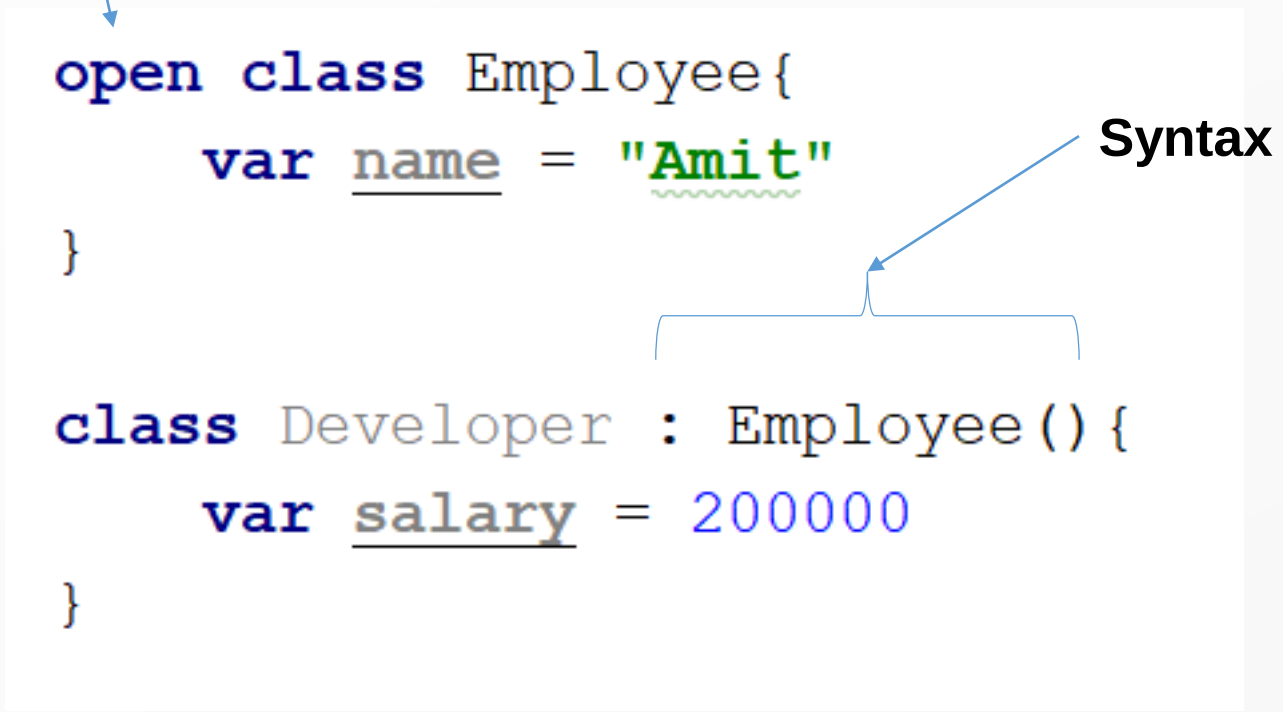




Inheritance (Cont.)

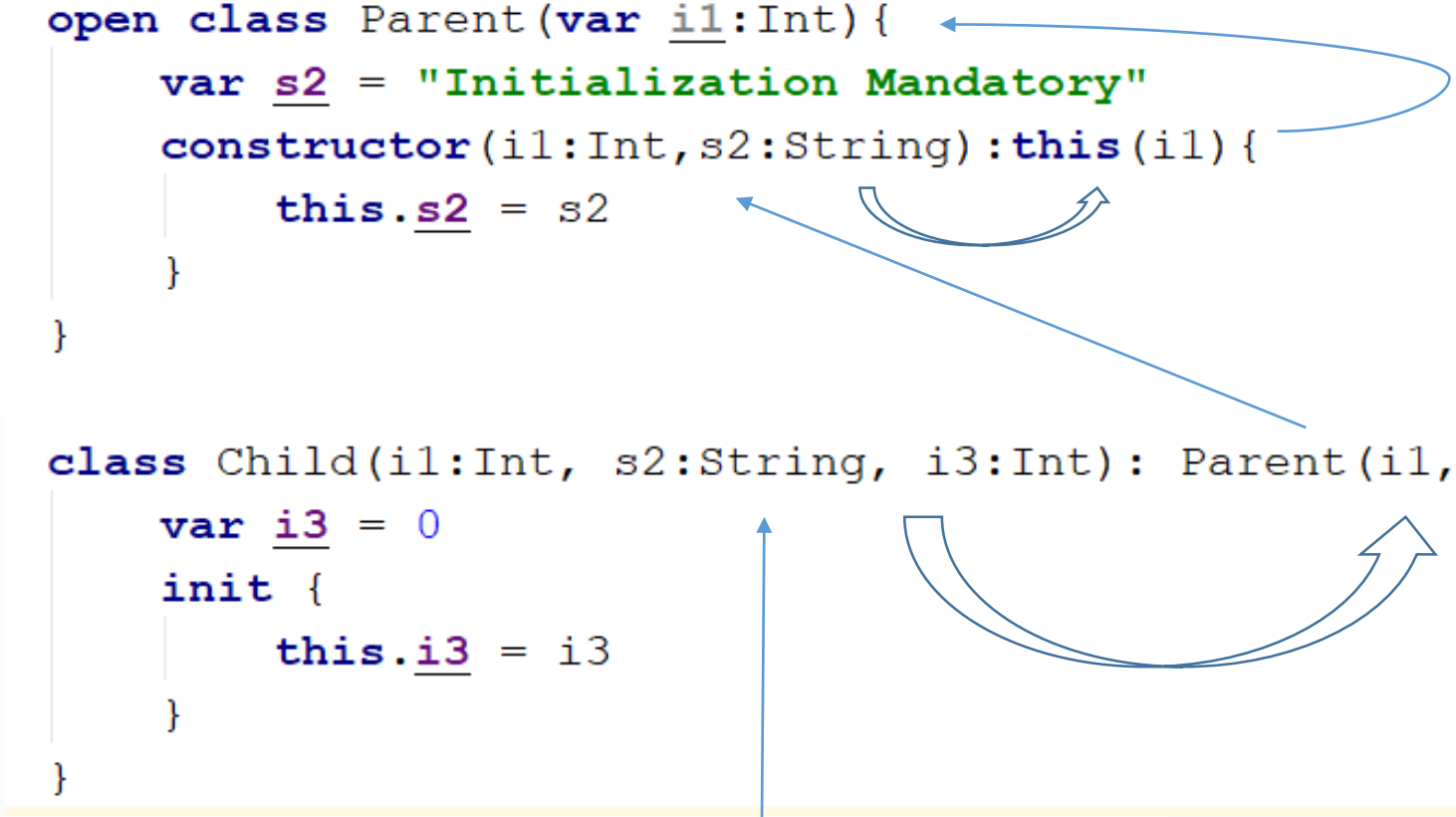
- By Default, Every Class in Kotlin is Final – not Inheritable
- ‘open’ keyword is used

Keyword



Constructor with Inheritance

```
open class Parent(var i1:Int){
    var s2 = "Initialization Mandatory"
    constructor(i1:Int,s2:String):this(i1){
        this.s2 = s2
    }
}

class Child(i1:Int, s2:String, i3:Int): Parent(i1,s2){
    var i3 = 0
    init {
        this.i3 = i3
    }
}
```



```
var child = Child(12,"data",22)
```

Override

- Re-Implementing Parent class method(s) in the Child Class – Overriding
- By Default, All the methods are final – not overridable
- Use ‘open’ keyword before method to be overridden
- Use ‘override’ keyword before method in the subclass

```
open class Parent{
    open fun method1(){
        print("method1 implemented")
    }
}
```

```
class Child :Parent(){
    override fun method1() {
        print("method1 is modified")
    }
}
```

Data Class

- Data is important, not Objects and Memory Allocations
- Declared using ‘data’ keyword
- Must have a Primary Constructor
- Printing the Object – will give data, not Memory location

Keyword

```
data class Employee(var name:String, var sal: Float)
```

Abstract Class

- Abstract Class can not be Instantiated.
- Use 'abstract' before declaration
- every abstract Class/Methods are 'public' and 'open' in nature
- Abstract methods does not have Bodies.
- Abstract class can have normal Methods

Abstract Class (Cont.)

Abstract Class

```
abstract class Phone{
    abstract var model:String
    abstract fun call()
    fun ring(){
        print("tring tring")
    }
}
```

Implementation of Abstract Class

```
class SmartPhone :Phone(){
    override var model: String = "S7"
    override fun call() {
        print("make call")
    }
}
```

Interface

- Use ‘interface’ keyword instead of Class
- Every Attribute is Abstract in Nature
- Can Have – Normal Method and Abstract Method
- Normal Method – public and open
- Abstract Method – Function without Body
- Any number of Interfaces can be implemented by a Class

Interface

Interface →

```
interface MyButtonListener{
    fun onDoubleClick(){
        print("Button Clicked Twice")
    }
    fun onClick()
}
```

Class Implementing Interface →

```
class MyApplication :MyButtonListener{
    override fun onClick() {
        //Write your logic
    }
}
```

Higher Order Functions in Kotlin

What is Higher Order?

- A Function that takes Functions as Parameters, or Returns a Function – Higher Order

```
fun compare(x:Int, y:Int): Int{
    var max = if(x>y) x else y
    return max
}
fun sum(x:Int, y:Int): Int{
    return x + y
}
```

```
fun action(x:Int, y:Int, act: (Int,Int)-> Int):Int{
    var result = act(x,y)
    return result
}
```

Lambda Function

- A kind of Optimized Function (Function Body only)
- Syntax:

```
var myLambda = { argumentList -> codeBody }
```

```
var addition = { a:Int, b:Int -> a+b }
```

```
var compare = { x:Int, y:Int -> if(x>y) x else y }
```

```
var modifiedPrint = { s:String -> print("Thankyou Mr. $s. See you Again!!")}
```

- In Kotlin, Lambda is an Expression

Lambda Function (Cont.)

- ‘it’ keyword – for Single Parameter Lambda Expression

```
{ a:String -> a.subSequence(1..(a.length-2)) }
```

Can be Changed to

```
{ it.subSequence(2..(it.length-2)) }
```

Closures

- This is Amazing Feature of Lambda Expressions

- Lambda Expression can modify the values of a variable present outside that expression

```
var result = 0
var addition = { a:Int, b:Int -> result = a+b }
addition(4,9)
print(result)          //Output = 13
```

Extension Function

- Method defined outside the Class.
- Extend any Class with new functionality
- New Functionality is not inserted inside Class
- Just for Current File
  - OR other files by Importing Current File

```
class Test{
    fun meth1(){
        print("in Meth1")
    }
}
```

```
fun Test.meth2(){
    print("in Meth2")
}
```

```
var test = Test()
test.meth2()
```

Infix Function

- Every Infix Functions are Extension Function
- They have single parameter.
- Use prefix Keyword ‘infix’

```
infix fun String.longer(that:String): String{
    if (this.length > that.length)
        return this
    else
        return that
}
```

```
var a = "Infosys"
var b = "HSBC"
print("The longer String is ${a longer b}")
```

Collections in Kotlin

Collections

- In Kotlin, Collection –
  - List
  - Map
  - Set
- Each Collection is of two Types
  - Immutable
  - Mutable
- Immutable type is preferred to ensure Less Errors.

Immutable Collections

List:

```
var myList = listOf<String>("str1","str2","str3")
```

Map:

```
var myMap = mapOf<Int,String>(101 to "Amit",102 to "Raj", 103 to "Mathew")
```

Set:

```
var mySet = setOf<Int>(3,4,5,6,3,5)
```

Mutable Collections

List:

```
var list1 = ArrayList<String>()
var list2 = arrayListOf<Boolean>(true, false)
var list3 = mutableListOf<Char>('d','n','a')
```

Map:

```
var map1 = HashMap<Float, String>()
var map2 = hashMapOf<Int, String>(101 to "Amit", 102 to "Raj")
var map3 = mutableMapOf<Int, String>(101 to "Mathew", 102 to "Mac")
```

Set:

```
var set1 = HashSet<Int>()
var set3 = hashSetOf<Int>(3,4,5,6)
var set2 = mutableSetOf<Int>(2,3,4)
```

Filter Function

- Returns a Collection with Elements matching the provided Predicate.

- Predicate – any True/False Condition

```
var list = listOf<String>("ab","bcd","dbdac","a","badc")
```

```
var filteredList = list.filter { it.length < 4 }
```

All, Any, Count, Find

- all : Do all elements satisfy the predicate?
- any : Do any elements satisfy the predicate?
- count : Total elements that satisfy the predicate.
- find : Returns the FIRST element that satisfy the predicate.

```
var myNumbers = listOf( 3,5,19,43,1,11,32 )
var check1 = myNumbers.all { it > 10 }
var check2 = myNumbers.any { it > 10 }
var totalCount = myNumbers.count { it > 10 }
```

Android App Development

Building your first Android App using Android Studio

Android UI

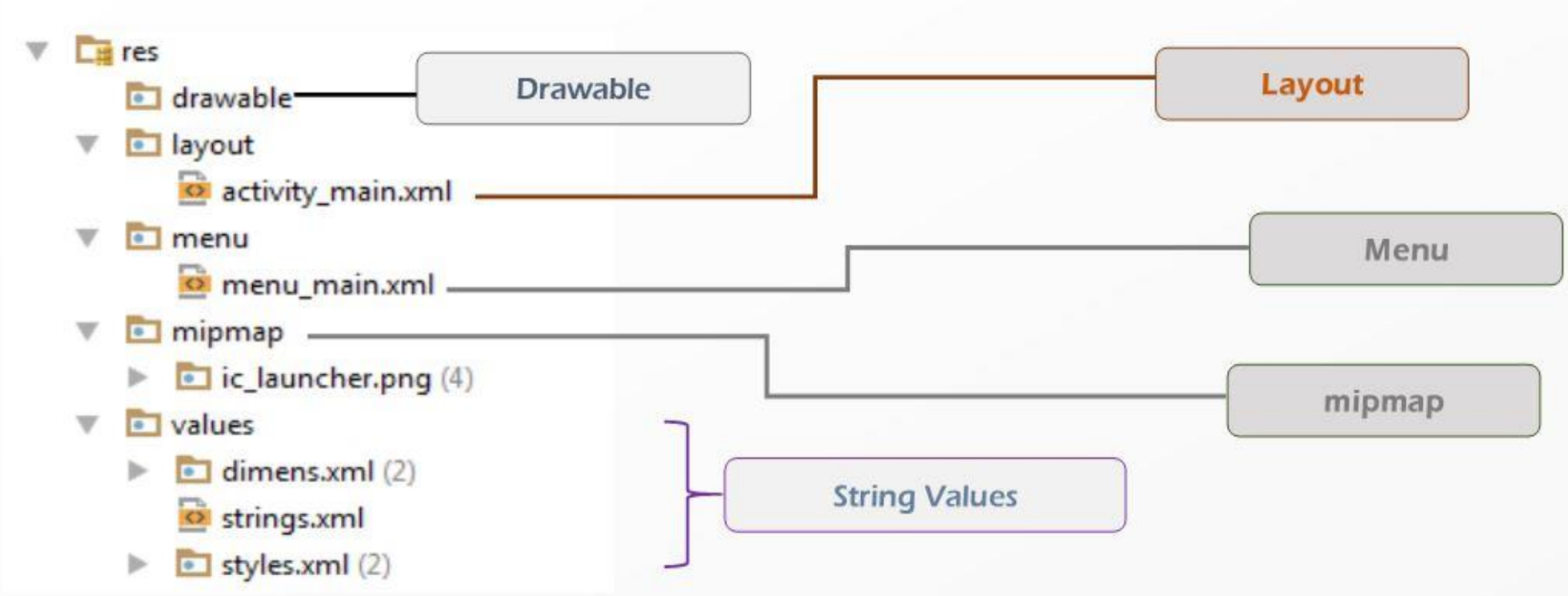
Android User Interface(Widgets)

- An app in Android usually consists of multiple screens
- Each screen in Android represents an **Activity.**
  - ***E.g:*** *An app with* ***3 screens*** *will have* ***3 Activities***.
- Each screen in turn comprises of multiple UI Components or **Views**.
- These views are declared in a **layout**.

Resources

UI Resources

- Most of the Android apps require different types of UI resources like drawables, values, menu etc.



UI Resources

Drawables:

- Drawables are Android Components which can be drawn on screen. E.g. images etc.
- Android categorizes drawables based on device screen density into 5 categories: ldpi, mdpi, hdpi, xhdpi, xxhdpi in increasing order of density.
- For best results, we need to place different resolution images in these folders so that Android may pick appropriate image for a particular device

UI Resources (cont.)

String Values

- A string resource provides text strings with optional text styling or formatting.
- It is best practice to create string resources and refer to them when using text in layouts. E.g. text of Button or hint of EditText.
- String resources go into res/value (-xxx) folder.
- This makes implementing localization easy.

Layouts

- The UI of an Activity is declared in **a layout file**.
- A layout is an xml file which describes order and arrangement of widgets.
- Layout files go in *res/layout* folder of the app.
- Each screen in Android will have a layout (.xml) file and an Activity(.java) file

UI Elements

Views

- Android provide many UI Elements called **views**.
- Some common views used in development are:

  - Button
  - TextView
  - EditText
  - ListView
  - Checkbox
  - RadioButton
  - Spinner
  - ImageView

- A view can be customized by setting its properties at the time of declaration on the layout xml file.

- We can provide id to a view to be able to manipulate it using kotlin code

Layouts and UI Elements

Type of Layout

```
<RelativeLayout ...>
    <TextView
        1        ... />
    <EditText
        2        .../>
    <Button
        3        ... />
</RelativeLayout>
```

UI Widgets

activity_main.xml

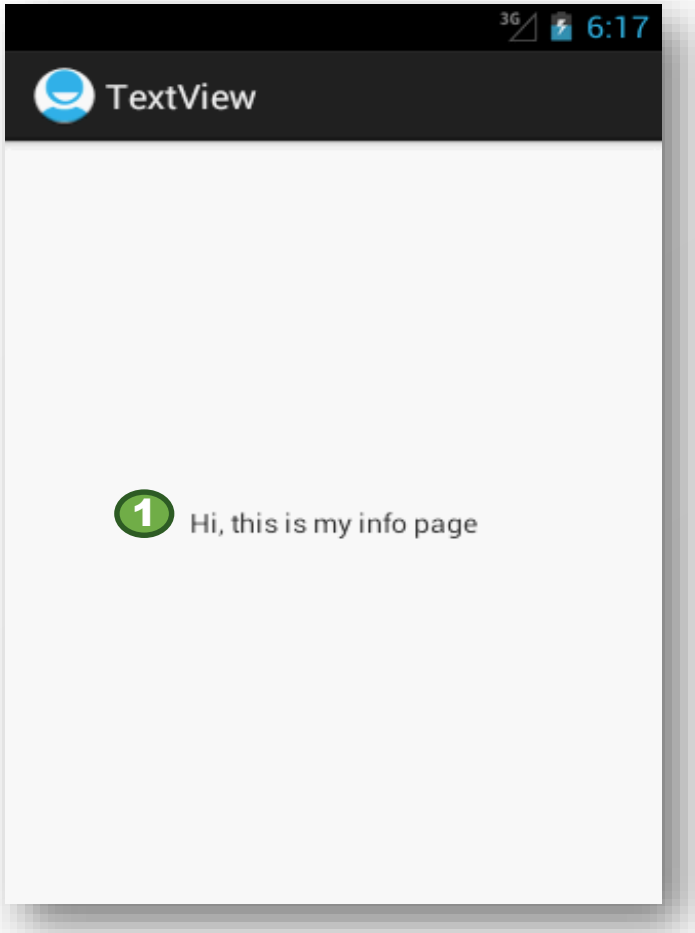
Layouts

- Layouts act as containers in which all UI components are captured.
- Since different applications want to position the UI components differently, Android provides various layouts:
  - **Linear Layout**: Used to display the contents in a linear fashion in horizontal rows or vertical columns.
  - **Relative Layout**: Used to position the contents of aa UI relative to other components of the UI. It enables complete control of the display of the activity.
  - **Table Layout**: Organizes different components in rows and columns. It provides a more structured UI.
  - **Frame Layout**: The simplest amongst Android layouts, which is used to organize view controls. Generally, it is used to display only one view at a time or for overlapping views.
  - **Constraint Layout**: To create layouts with flat view hierarchy (no nested view groups). Making Large layouts simple.

TextView -

1

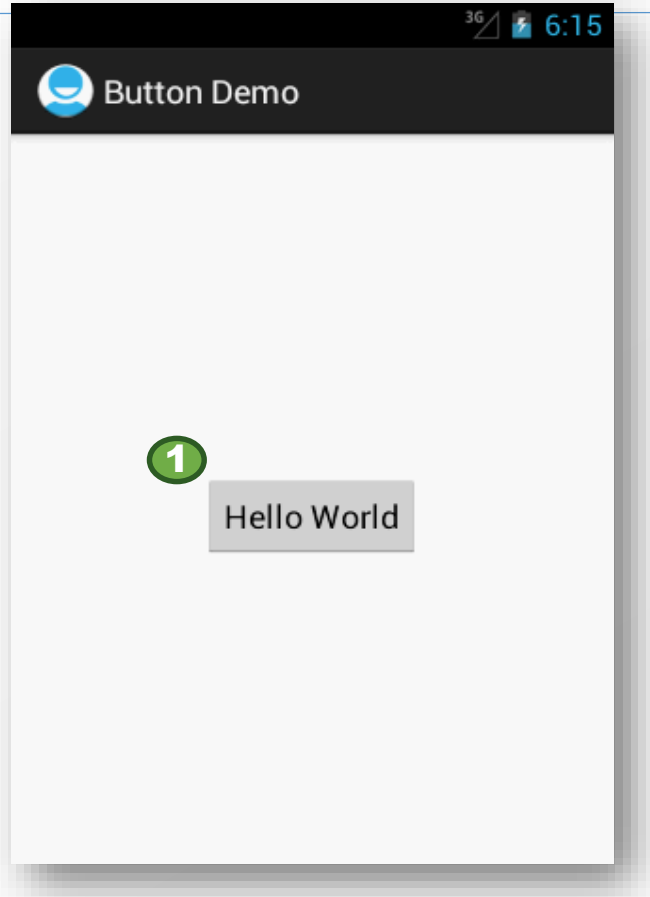
```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerHorizontal="true"
  android:layout_centerVertical="true"
  android:text="Hi, this is my info page"
  android:id="@+id/myMessage"
/>
```



Button

1

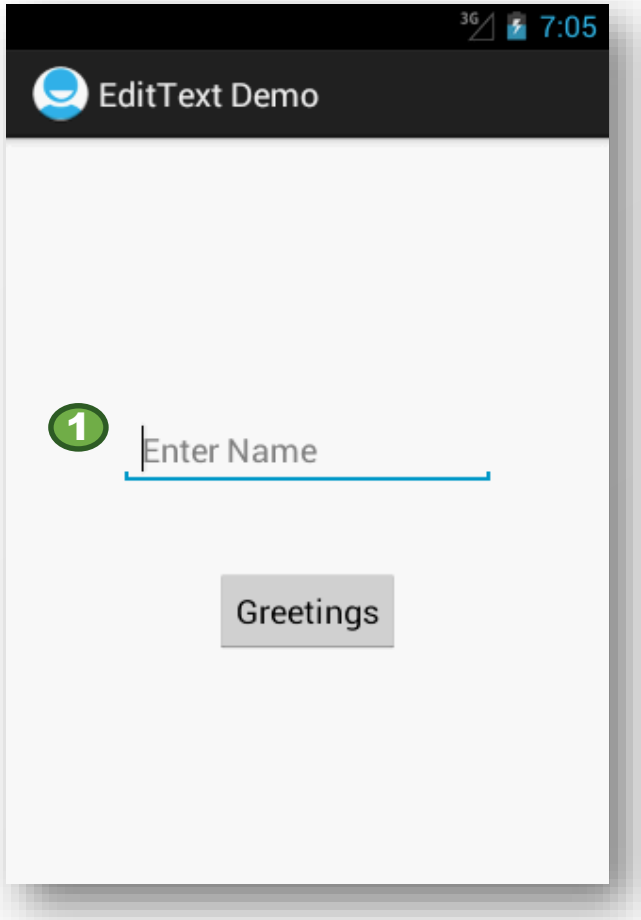
```
<Button
  android:id="@+id/button1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentTop="true"
  android:layout_centerHorizontal="true"
  android:layout_marginTop="189dp"
  android:text="Hello World" />
```



EditText

1

```
<EditText
    android:id="@+id/EditText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="169dp"
    android:hint="@string/enter_name"
    android:inputType="textNoSuggestions"
    android:minWidth="200dp" >
</EditText>
```



Basic UI Elements

Demo on Basic UI Elements- TextView, EditText and Button

Thank You

HSBC | Infosys®