Smt J.J.Kundalia Commerce College
Computer Science Department
Mobile Computing using Android and iPhone

UNIT-1

Introduction to Kotlin Programming

Basics of Kotlin

What is Kotlin

Kotlin is a general-purpose, statically typed, and open-source programming language. It runs on JVM and can be used anywhere Java is used today. It can be used to develop Android apps, server-side apps and much more.

History of Kotlin

Kotlin was developed by JetBrains team. A project was started in 2010 to develop the language and officially, first released in February 2016. Kotlin was developed under the Apache 2.0 license.

Features of Kotlin

- **Concise:** Kotlin reduces writing the extra codes. This makes Kotlin more concise.
- **Null safety:** Kotlin is null safety language. Kotlin aimed to eliminate the NullPointerException (null reference) from the code.
- **Interoperable:** Kotlin easily calls the Java code in a natural way as well as Kotlin code can be used by Java.
- **Smart cast:** It explicitly typecasts the immutable values and inserts the value in its safe cast automatically.
- **Compilation Time:** It has better performance and fast compilation time.
- **Tool-friendly:** Kotlin programs are build using the command line as well as any of Java IDE.
- **Extension function:** Kotlin supports extension functions and extension properties which means it helps to extend the functionality of classes without touching their code.

Kotlin Variable

Variable refers to a memory location. It is used to store data. The data of variable can be changed and reused depending on condition or on information passed to the program.

Variable Declaration

Kotlin variable is declared using keyword **var** and **val**.

```
1.  var language ="Java"
2.  val salary = 30000
```

The difference between var and val is specified later on this page.

Here, variable language is String type and variable salary is Int type. We don't require specifying the type of variable explicitly. Kotlin complier knows this by initilizer expression ("Java" is a String and 30000 is an Int value). This is called type inference in programming.

We can also explicitly specify the type of variable while declaring it.

```
var language: String ="Java"
val salary: Int = 30000
```

It is not necessary to initialize variable at the time of its declaration. Variable can be initialized later on when the program is executed.

```
var language: String
... ... ...
language = "Java"
```

```
val salary: Int
... ... ...
salary = 30000
```

Difference between var and val

- **var** (Mutable variable): We can change the value of variable declared using **var** keyword later in the program.
- **val** (Immutable variable): We cannot change the value of variable which is declared using **val** keyword.

Example

```
1. var salary = 30000
2. salary = 40000 //execute
```

Here, the value of variable salary can be changed (from 30000 to 40000) because variable salary is declared using **var** keyword.

```
1. val language = "Java"
2. language = "Kotlin" //Error
```

Here, we cannot re-assign the variable language from "Java" to "Kotlin" because the variable is declared using **val** keyword.

Data type (basic type) refers to type and size of data associated with variables and functions. Data type is used for declaration of memory location of variable which determines the features of data.

In Kotlin, everything is an object, which means we can call member function and properties on any variable.

Kotlin built in data type are categorized as following different categories:

- Number
- Character
- Boolean
- Array
- String

Number Types

Number types of data are those which hold only number type data variables. It is further categorized into different Integer and Floating point.

Data Type	Bit Width (Size)	Data Range
Byte	8 bit	-128 to 127
Short	16 bit	-32768 to 32767
Int	32 bit	-2,147,483,648 to 2,147,483,647
Long	64 bit	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Float	32 bit	1.40129846432481707e-45 to 3.40282346638528860e+38
Double	64 bit	4.94065645841246544e-324 to 1.79769313486231570e+308

Character (Char) Data Type

Characters are represented using the keyword **Char**. Char types are declared using single quotes ('').

Data Type	Bit Width (Size)	Data Range
Char	4 bit	-128 to 127

Example

```
1.  val value1 = 'A'
2.  //or
3.  val  value2: Char
4.  value2= 'A'
```

Boolean Data Types

Boolean data is represented using the type Boolean. It contains values either true or false.

Data Type	Bit Width (Size)	Data Value
Boolean	1 bit	true or false

Example

```
val flag = true
```

Array

Arrays in Kotlin are represented by the Array class. Arrays are created using library function arrayOf() and Array() constructor. Array has get (), set() function, size property as well as some other useful member functions.

Creating Array using library function arrayOf()

The arrayOf() function creates array of wrapper types. The item value are passed inside arrayOf() function like arrayOf(1,2,3) which creates an array[1,2,3].

The elements of array are accessed through their index values (array[index]). Array index are start from zero.

```
val id = arrayOf(1,2,3,4,5)

val firstId = id[0]

val lasted = id[id.size-1]
```

Creating Array using Array() constructor

Creating array using Array() constructor takes two arguments in Array() constructor:

1. First argument as a size of array, and
2. Second argument as the function, which is used to initialize and return the value of array element given its index.

```
val asc = Array(5, { i -> i * 2 }) //asc[0,2,4,6,8]
```

String

String in Kotlin is represented by String class. String is immutable, which means we cannot change the elements in String.

String declaration:

```
val text ="Hello, j.j.kundalia"
```

Types of String

String are categorize into two types. These are:

1. **Escaped String:** Escape String is declared within double quote (" ") and may contain escape characters like '\n', '\t', '\b' etc.

```
val text1 ="Hello, J.J.Kundalia"
//or
val text2 ="Hello, j.j.kundalia\n"
//or
val text3 ="Hello, \nj.j.kundalia"
```

2. Raw String: Raw String is declared within triple quote (""" """). It provides facility to declare String in new lines and contain multiple lines. Raw String cannot contain any escape character.

```
val text1 ="""
        Welcome
           To
       j.j.kundalia
  """
```

Kotlin Operator

Operators are special characters which perform operation on operands (values or variable).There are various kind of operators available in Kotlin.

- Arithmetic operator
- Relation operator
- Assignment operator
- Unary operator
- Bitwise operation
- Logical operator

Arithmetic Operator

Arithmetic operators are used to perform basic mathematical operations such as addition (+), subtraction (-), multiplication (*), division (/) etc.

Operator	Description	Expression	Translate to
+	Addition	a+b	a.plus(b)
-	Subtraction	a-b	a.minus(b)
*	Multiply	a*b	a.times(b)
/	Division	a/b	a.div(b)
%	Modulus	a%b	a.rem(b)

Example of Arithmetic Operator

```
fun main(args : Array<String>) {
var a=10;
var b=5;
println(a+b);
println(a-b);
println(a*b);
```

```
println(a/b);
println(a%b);
}
```

Output:

```
15
5
50
2
0
```

Relation Operator

Relation operator shows the relation and compares between operands. Following are the different relational operators:

Operator	Description	Expression	Translate to
>	greater than	a>b	a.compateTo(b)>0
<	Less than	a<b	a.compateTo(b)<0
>=	greater than or equal to	a>=b	a.compateTo(b)>=0
<=	less than or equal to	a<=b	a?.equals(b)?:(b===null)
==	is equal to	a==b	a?.equals(b)?:(b===null)
!=	not equal to	a!=b	!(a?.equals(b)?:(b===null))

Example of Relation Operator

```
fun main(args : Array<String>) {
   val a = 5
   val b = 10
   val max = if (a > b) {
   println("a is greater than b.")
      a
   } else{
    println("b is greater than a.")
      b
   }
   println("max = $max")
}
```

Output:

```
b is greater than a.
max = 10
```

Assignment operator

Assignment operator **"="** is used to assign a value to another variable. The assignment of value takes from right to left.

Operator	Description	Expression	Convert to

+=	add and assign	a+=b	a.plusAssign(b)
-=	subtract and assign	a-=b	a.minusAssign(b)
*=	multiply and assign	a*=b	a.timesAssign(b)
/=	divide and assign	a/=b	a.divAssign(b)
%=	mod and assign	a%=b	a.remAssign(b)

Example of Assignment operator

```
fun main(args : Array<String>) {
   var a =20;var b=5
   a+=b
   println("a+=b :"+ a)
   a-=b
   println("a-=b :"+ a)
   a*=b
   println("a*=b :"+ a)
   a/=b
  println("a/=b :"+ a)
   a%=b
 println("a%=b :"+ a)
 }
```

Output:

```
a+=b :25
a-=b :20
a*=b :100
a/=b :20
a%=b :0
```

Unary Operator

Unary operator is used with only single operand. Following are some unary operator given below.

Operator	Description	Expression	Convert to
+	unary plus	+a	a.unaryPlus()
-	unary minus	-a	a.unaryMinus()
++	increment by 1	++a	a.inc()
--	decrement by 1	--a	a.dec()
!	not	!a	a.not()

Example of Unary Operator

```
fun main(args: Array<String>){
 var a=10
var b=5
 var flag = true
 println("+a :"+ +a)
println("-b :"+ -b)
 println("++a :"+ ++a)
```

```
println("--b :"+ --b)
 println("!flag :"+ !flag)
}
```

Output:

```
+a :10
-b :-5
++a :11
--b :4
!flag :false
```

Logical Operator

Logical operators are used to check conditions between operands. List of logical operators are given below.

Operator	Description	Expression	Convert to
&&	return true if all expression are true	(a>b) && (a>c)	(a>b) and (a>c)
||	return true if any expression are true	(a>b) || (a>c)	(a>b) or(a>c)
!	return complement of expression	!a	a.not()

Example of Logical Operator

```
fun main(args: Array<String>){
var a=10
var b=5
var c=15
var flag = false
var result: Boolean
result = (a>b) && (a>c)
println("(a>b) && (a>c) :"+ result)
result = (a>b) || (a>c)
println("(a>b) || (a>c) :"+ result)
result = !flag
println("!flag :"+ result)
}
```

Output:

```
(a>b) && (a>c) :false
(a>b) || (a>c) :true
!flag :true
```

Bitwise Operation

In Kotlin, there is not any special bitwise operator. Bitwise operation is done using named function.

Named Function	Description	Expression
shl (bits)	signed shift left	a.shl(b)
shr (bits)	signed shift right	a.shr(b)
ushr (bits)	unsigned shift right	a.ushr(b)
and (bits)	bitwise and	a.and(b)
or (bits)	bitwise or	a.or(b)

xor (bits)	bitwise xor	a.xor(b)
inv()	bitwise inverse	a.inv()

Example of Bitwise Operation

```
fun main(args: Array<String>){
var a=10
var b=2
println("a.shl(b): "+a.shl(b))
println("a.shr(b): "+a.shr(b))
println("a.ushr(b:) "+a.ushr(b))
println("a.and(b): "+a.and(b))
println("a.or(b): "+a.or(b))
 println("a.xor(b): "+a.xor(b))
println("a.inv(): "+a.inv())
}
```

Output:

```
a.shl(b): 40
a.shr(b): 2
a.ushr(b:) 2
a.and(b): 2
a.or(b): 10
a.xor(b): 8
a.inv(): -11
```

Kotlin Standard Input/Output

Kotlin standard input output operations are performed to flow byte stream from input device (keyboard) to main memory and from main memory to output device (screen).

Kotlin Output

Kotlin output operation is performed using the standard methods **print()** and **println()**. Let's see an example:

```
fun main(args: Array<String>) {
   println("Hello World!")
   print("Welcome to  JJKundalia")
}
```

Output

```
Hello World!
Welcome to  JJKundalia
```

The methods print() and println() are internally call System.out.print() and System.out.println() respectively.

Difference between print() and println() methods:

print(): print() method is used to print values provided inside the method "()".

println(): println() method is used to print values provided inside the method "()" and moves cursor to the beginning of next line.

Example

```
fun main(args: Array<String>){
   println(10)
   println("Welcome to  JJKundalia")
   print(20)
   print("Hello")
}
```

Output:

```
10
Welcome to  JJKundalia
20Hello
```

Kotlin Input

Kotlin has standard library function **readLine()** which is used for reads line of string input from standard input stream. It returns the line read or null. Let's see an example:

```
fun main(args: Array<String>) {
   println("Enter your name")
   val name = readLine()
   println("Enter your age")
   var age: Int =Integer.valueOf(readLine())
   println("Your name is $name and your age is $age")
}
```

Output:

Enter your name
Rahul
Enter your age
25
Your name is Rahul and your age is 25

While using the readLine() function, input lines other than String are explicitly converted into their corresponding types.

To input other data type rather than String, we need to use Scanner object of java.util.Scanner class from Java standard library.

Example Getting Integer Input

```
import java.util.Scanner
fun main(args: Array<String>) {
   val read = Scanner(System.`in`)
   println("Enter your age")
   var age = read.nextInt()
   println("Your input age is "+age)
}
```

Output:

```
Enter your age
25
Your input age is 25
```

Here nextInt() is a method which takes integer input and stores in integer variable. The other data types Boolean, Float, Long and Double uses nextBoolean(), nextFloat(), nextLong() and nextDouble() to get input from user.

Kotlin Comment

Comments are the statements that are used for documentation purpose. Comments are ignored by compiler so that don't execute. We can also used it for providing information about the line of code. There are two types of comments in Kotlin.

Single line comment.

Multi line comment.

Single line comment

Single line comment is used for commenting single line of statement. It is done by using '//' (double slash). For example:

```
fun main(args: Array<String>) {
// this statement used for print
   println("Hello World!")
}
```

Output

```
Hello World!
```

Multi line comment

Multi line comment is used for commenting multiple line of statement. It is done by using /* */ (start with slash strict and end with star slash). For example:

```
fun main(args: Array<String>) {
/* this statement
  is used
  for print */
   println("Hello World!")
}
```

Output:

```
Hello World!
```

Kotlin if Expression

In Kotlin, **if** is an expression is which returns a value. It is used for control the flow of program structure. There is various type of if expression in Kotlin.

- if-else expression
- if-else if-else ladder expression
- nested if expression

Traditional if Statement

Syntax of traditional if statement

```
if(condation){
//code statement
}
```

Syntax of traditional **if else** statement

```
if(condation){
//code statement
}
else{
```

```
//code statement
}
```

Kotlin if-else Expression

As if is an expression it is not used as standalone, it is used with if-else expression and the result of an if-else expression is assign into a variable.

Syntax of if-else expression

```
val returnValue = if (condation) {
 //code statement
} else {
// code statement
}
   println(returnValue)
```

Kotlin if-else Expression Example

Let's see an example of if-else expression.

```
fun main(args: Array<String>) {
val num1 = 10
val num2 =20
val result = if (num1 > num2) {
"$num1 is greater than $num2"
} else {
"$num1 is smaller than $num2"
}
println(result)
}
```

Output:

```
10 is smaller than 20
```

We can remove the curly braces of if-else body by writing if expression in only one statement.

For example:

```
fun main(args: Array<String>) {
val num1 = 10
val num2 =20
val result = if (num1 > num2) "$num1 is greater than $num2" else "$num1 is smaller than $num2"
println(result)
}
```

Using if-else expression in one single line statement is like ternary operator in Java. Kotlin does not support any ternary operator.

Kotlin if-else if-else Ladder Expression

Let's see an example of if-else if-else ladder expression.

```
fun main(args: Array<String>) {
val num = 10
```

```
val result = if (num > 0){
"$num is positive"
}else if(num < 0){
"$num is negative"
}else{
"$num is zero"
}
println(result)
}
```

Output:

```
10 is positive
```

Kotlin Nested if Expression

Let's see an example of nested if expression.

```
fun main(args: Array<String>) {
val num1 = 25
 val num2 = 20
val num3 = 30
val result = if (num1 > num2){
val max = if(num1 > num3){
 num1
}else{
num3
}
"body of if "+max
}else if(num2 > num3){
"body of else if"+num2
}else{
"body of else "+num3
 }
println("$result")
}
```

Output:

```
body of if 30
```

Kotlin when Expression

Kotlin, when expression is a conditional expression which returns the value. Kotlin, when expression is replacement of switch statement. Kotlin, when expression works as a switch statement of other language (Java, C++, C).

Using when as an Expression

Let's see a simple example of when expression.

```
fun main(args: Array<String>){
   var number = 4
   var numberProvided = when(number) {
      1 -> "One"
      2 -> "Two"
      3 -> "Three"
```

```
    4 -> "Four"
    5 -> "Five"
    else -> "invalid number"
  }
  println("You provide $numberProvided")
}
```

Output:
You provide Four

Using when Without Expression

It is not mandatory to use when as an expression, it can be used as normally as it used in other language.

For Example

```
fun main(args: Array<String>){
   var number = 4
  when(number) {
    1 -> println("One")
    2 -> println("Two")
    3 -> println("Three")
    4 -> println("Four")
    5 -> println("Five")
    else -> println("invalid number")
  }
 }
```

Output:
Four

Multiple Statement of when Using Braces

We can use multiple statement enclosed within block of condition.

For Example

```
fun main(args: Array<String>){
  var number = 1
  when(number) {
    1 -> {
      println("Monday")
      println("First day of the week")
    }
    7 -> println("Sunday")
    else -> println("Other days")
  }
}
```

Output:
Monday
First day of the week

Multiple branches of when

We can use multiple branches of condition separated with a comma. It is used, when we need to run a same logic for multiple choices.

```
fun main(args: Array<String>){
   var number = 8
   when(number) {
      3, 4, 5, 6 ->
         println("It is summer season")
      7, 8, 9 ->
         println("It is rainy season")
      10, 11 ->
         println("It is autumn season")
      12, 1, 2 ->
         println("It is winter season")
      else -> println("invalid input")
   }
}
```

Output:

It is rainy season

Using when in the range

The when expression also check the ranges of input provided in when condition. A range is created using .. (double dot) operator. The in operator is used to check if a value belongs to a range.

For Example:

```
fun main(args: Array<String>){
   var number = 7
   when(number) {
      in 1..5 -> println("Input is provided in the range 1 to 5")
      in 6..10 -> println("Input is provided in the range 6 to 10")
      else -> println("none of the above")
   }
}
```

Output:

Input is provided in the range 6 to 10

Kotlin for Loop

Kotlin **for** loop is used to *iterate* a part of program several times. It iterates through arrays, ranges, collections, or anything that provides for iterate. Kotlin for loop is equivalent to the **foreach** loop in languages like C#.

Syntax of for loop in Kotlin:

```
for (item in collection){
//body of loop
}
```

Iterate through array

Let's see a simple example of iterating the elements of array.

```
fun main(args : Array<String>) {
   val marks = arrayOf(80,85,60,90,70)
   for(item in marks){
      println(item)
   }
```

```
}
```
Output:
```
80
85
60
90
70
```
If the body of for loop contains only one single line of statement, it is not necessary to enclose within curly braces **{}.**
```
fun main(args : Array<String>) {
   val marks = arrayOf(80,85,60,90,70)
   for(item in marks)
      println(item)
}
```
The elements of an array are iterated on the basis of *indices* (index) of array. For example:
```
fun main(args : Array<String>) {
  val marks = arrayOf(80,85,60,90,70)
   for(item in marks.indices)
     println("marks[$item]: "+ marks[item])
}
```
Output:
```
marks[0]: 80
marks[1]: 85
marks[2]: 60
marks[3]: 90
marks[4]: 70
```
Iterate through range

Let's see an example of iterating the elements of range.
```
fun main(args : Array<String>) {
 print("for (i in 1..5) print(i) = ")
   for (i in 1..5) print(i)
   println()
   print("for (i in 5..1) print(i) = ")
   for (i in 5..1) print(i)          // prints nothing
   println()
   print("for (i in 5 downTo 1) print(i) = ")
   for (i in 5 downTo 1) print(i)
   println()
   print("for (i in 5 downTo 2) print(i) = ")
   for (i in 5 downTo 2) print(i)
   println()
   print("for (i in 1..5 step 2) print(i) = ")
   for (i in 1..5 step 2) print(i)
   println()
   print("for (i in 5 downTo 1 step 2) print(i) = ")
```

```
   for (i in 5 downTo 1 step 2) print(i)
}
```

Output:

```
for (i in 1..5) print(i) = 12345
for (i in 5..1) print(i) =
for (i in 5 downTo 1) print(i) = 54321
for (i in 5 downTo 2) print(i) = 5432
for (i in 1..5 step 2) print(i) = 135
for (i in 5 downTo 1 step 2) print(i) = 531
```

Kotlin while Loop

The **while loop** is used to iterate a part of program several time. Loop executed the block of code until the condition has true. Kotlin while loop is similar to Java while loop.

Syntax

```
while(condition){
//body of loop
}
```

Example of while Loop

Let's see a simple example of while loop printing value from 1 to 5.

```
fun main(args: Array<String>){
   var i = 1
   while (i<=5){
      println(i)
      i++
   }
}
```

Output:

```
1
2
3
4
5
```

Kotlin Infinite while Loop

The while loop executes a block of code to infinite times, if while condition remain true.

For example:

```
fun main(args: Array<String>){
      while (true){
      println("infinite loop")
      }
}
```

Output:

```
infinite loop
infinite loop
infinite loop
.
.
```

.
.
infinite loop
infinite loop

Kotlin do-while Loop

The **do-while** loop is similar to **while** loop except one key difference. A *do-while* loop first execute the body of **do** block after that it check the condition of while.

As a do block of *do-while* loop executed first before checking the condition, *do-while* loop execute at least once even the condition within *while* is false. The *while* statement of do-while loop end with ";" (semicolon).

Syntax

```
do{
//body of do block
}
while(condition);
```

Example of do -while loop

Let's see a simple example of do-while loop printing value 1 to 5.

```
fun main(args: Array<String>){
   var i = 1
   do {
      println(i)
      i++
   }
   while (i<=5);
}
```

Output:

```
1
2
3
4
5
```

Example of do -while loop even condition of while if false

In this example do-while loop execute at once time even the condition of *while* is false.

```
fun main(args: Array<String>){
   var i = 6
   do {
      println(i)
      i++
   }
   while (i<=5);
}
```

Output:

```
6
```

Kotlin Return and Jump

There are three jump expressions in Kotlin. These jump expressions are used for control the flow of program execution. These jump structures are:

break
continue
return

Break Expression

A **break** expression is used for terminate the nearest enclosing loop. It is almost used with if-else condition.

For example:

```
for(..){
    //body of for
    if(checkCondition){
       break;
    }
}
```

In the above example, **for** loop terminates its loop when **if** condition execute break expression.

Kotlin break example:

```
fun main(args: Array<String>) {
  for (i in 1..5) {
     if (i == 3) {
        break
     }
     println(i)
  }
}
```

Output:

```
1
2
```

In the above example, when the value of **i** became equal to 3 and satisfy the **if** condition(i==3) than the break expression execute and terminate **for** loop.

Kotlin Labeled break Expression

Labeled is the form of identifier followed by the **@** sign, for example **abc@**, **test@**. To make an expression as label, we just put a label in front of expression.

Kotlin labeled break expression is used to terminate the specific loop. This is done by using break expression with **@** sign followed by label name (break@loop).

Kotlin labeled break example

```
fun main(args: Array<String>) {
  loop@ for (i in 1..3) {
     for (j in 1..3) {
        println("i = $i and j = $j")
        if (i == 2)
           break@loop
     }
  }
}
```

Output:

i = 1 and j = 1
i = 1 and j = 2
i = 1 and j = 3
i = 2 and j = 1

In the above example, when the value of i became 2 and satisfy the **if** condition which execute break expression followed by labeled name. The *break* expression followed by labeled name terminates the body of label identifier

Kotlin continue Jump Structure

Kotlin, **continue** statement is used to repeat the loop. It continues the current flow of the program and skips the remaining code at specified condition.

The *continue* statement within a nested loop only affects the inner loop.

For example

```
for(..){
    //body of for above if
    if(checkCondition){
       continue
    }
//body of for below if
}
```

In the above example, **for** loop repeat its loop when **if** condition execute continue. The *continue* statement makes repetition of loop without executing the below code of if condition.

Kotlin continue example

```
fun main(args: Array<String>) {
     for (i in 1..3) {
        println("i = $i")
        if (j == 2) {
           continue
        }
        println("this is below if")
     }
}
```

Output:

```
i = 1
this is below if
i = 2
i = 3
this is below if
```

Kotlin Labeled continue Expression

Labeled is the form of identifier followed by the **@** sign, for example **abc@, test@.** To make an expression as label, we just put a label in front of expression.

Kotlin, labeled *continue* expression is used for repetition of specific loop (labeled loop). This is done by using continue expression with **@** sign followed by label name (continue@labelname).

Kotlin labeled continue example

```
fun main(args: Array<String>) {
   labelname@ for (i in 1..3) {
   for (j in 1..3) {
      println("i = $i and j = $j")
      if (i == 2) {
         continue@labelname
      }
      println("this is below if")
   }
 }
}
```

Output:

```
i = 1 and j = 1
this is below if
i = 1 and j = 2
this is below if
i = 1 and j = 3
this is below if
i = 2 and j = 1
i = 3 and j = 1
this is below if
i = 3 and j = 2
this is below if
i = 3 and j = 3
this is below if
```

Kotlin Array

Array is a collection of similar data either of types Int, String etc. Array in Kotlin has mutable in nature with fixed size. Which means we can perform both read and writes operations on elements of array.

Syntax of array decleration:

It initializes the element of array of int type with size 5 with all elements as 0 (zero).

```
var myArray = Array<Int>(5){0}
```

Kotlin array declaration - using arrayOf function

```
var myArray1 = arrayOf(1,10,4,6,15)
var myArray2 = arrayOf<Int>(1,10,4,6,15)
val myArray3 = arrayOf<String>("Ajay","Prakesh","Michel","John","Sumit")
```

```
var myArray4= arrayOf(1,10,4, "Ajay","Prakesh")
```

Kotlin array declaration - using arrayOf function

```
var myArray5: IntArray = intArrayOf(5,10,20,12,15)
```

Let's see an example of array in Kotlin. In this example we will see how to initialize and traverse its elements.

Kotlin Array Example 1:

In this example, we are simply initialize an array of size 5 with default value as 0. The index value of array starts with 0. First element of array is placed at index 0 and last element at one less than the size of array.

```
fun main(args: Array<String>){
   var myArray = Array<Int>(5){0}
    for(element in myArray){
      println(element)
   }
}
```

Output:

```
0
0
0
0
0
```

Kotlin Array Example 2:

We can also able to rewrite the value of array using its index value. Since we can able to modify the value of array, so it is called as **mutable** in nature. For example:

```
fun main(args: Array<String>){
   var myArray = Array<Int>(5){0}
    myArray[1]= 10
   myArray[3]= 15
    for(element in myArray){
      println(element)
   }
}
```

Output:

```
0
10
0
15
0
```

Kotlin Array Example 3 - using arrayOf() and intArrayOf() function:

Array in Kotlin also declare using different functions such as arrayOf(), intArrayOf(), etc. Let's see the example arrayOf() and intArrayOf() function.

```
fun main(args: Array<String>){
   val name = arrayOf<String>("Ajay","Prakesh","Michel","John","Sumit")
   var myArray2 = arrayOf<Int>(1,10,4,6,15)
   var myArray3 = arrayOf(5,10,20,12,15)
```

```
  var myArray4= arrayOf(1,10,4, "Ajay","Prakesh")
  var myArray5: IntArray = intArrayOf(5,10,20,12,15)
   for(element in name){
     println(element)
  }
    println()
  for(element in myArray2){
     println(element)
  }
  println()
  for(element in myArray3){
     println(element)
  }
  println()
  for(element in myArray4){
     println(element)
  }
  println()
  for(element in myArray5){
     println(element)
  }
 }
```

Output:

```
Ajay
Prakesh
Michel
John
Sumit

1
10
4
6
15

5
10
20
12
15

1
10
4
Ajay
```

Prakesh

5
10
15
20
25

Kotlin Array Example 4

Suppose that when we try to insert an element at index position greater than array size than what happen? It will throw an *ArrayIndexOutOfBoundException*. This is because the index value is not present where we want to insert the element. Due to this array is called **fixed size** length. Let's see the example:

```
fun main(args: Array<String>){
   var myArray5: IntArray = intArrayOf(5,10,20,12,15)
     myArray5[6]=18 // ArrayIndexOutOfBoundsException
   for(element in myArray5){
      println(element)
   }
}
```

Output:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
        at ArrayListKt.main(Array.kt:4)
```

Kotlin Array Example 5 - traversing using range:

The Kotlin's array elements are also traverse using index range (minValue..maxValue) or (maxValue..minvalue). Let's see an example of array traversing using range.

```
fun main(args: Array<String>){
   var myArray5: IntArray = intArrayOf(5,10,20,12,15)
    for (index in 0..4){
      println(myArray5[index])
   }
   println()
   for (index in 0..myArray5.size-1){
      println(myArray5[index])
   }
}
```

Output:

```
5
10
20
12
15

5
10
20
```

12
15

Kotlin Collections

Collections in Kotlin are used to store group of related objects in a single unit. By using collection, we can store, retrieve manipulate and aggregate data.

Types of Kotlin Collections

Kotlin collections are broadly categories into two different forms. These are:

Immutable Collection (or **Collection**)

Mutable Collection

Immutable Collection:

Immutable collection also called Collection supports read only functionalities. Methods of immutable collection that supports read functionalities are:

Collection Types	Methods of Immutable Collection
List	listOf() listOf<T>()
Map	mapOf()
Set	setOf()

Mutable Collection:

Mutable collections supports both read and write functionalities. Methods of mutable collections that supports read and write functionalities are:

Collection Types	Methods of Mutable Collection
List	ArrayList<T>() arrayListOf() mutableListOf()
Map	HashMap hashMapOf() mutableMapOf()
Set	hashSetOf() mutableSetOf()

Kotlin List Interface

Kotlin **List** is an interface and generic collection of elements. The List interface inherits form Collection<T> class. It is immutable and its methods supports only read functionalities.

To use the List interface we need to use its function called **listOf(), listOf<E>().**

The elements of list follow the sequence of insertion order and contains index number same as array.

List Interface Declaration

public interface List<out E> : Collection<E> (source)

Function of Kotlin List Interface

There are several functions are available in the List interface. Some functions of List interface are mention below.

Functions	Descriptions
abstract fun contains(element: E): Boolean	It checks specified element is contained in this collection.
abstract fun containsAll(elements: Collection<E>): Boolean	It checks all elements specified are contained in this collection.

abstract operator fun get(index: Int): E	It returns the element at given index from the list.
abstract fun indexOf(element: E): Int	Returns the index of first occurrence of specified element in the list, or -1 if specified element is not present in list.
abstract fun isEmpty(): Boolean	It returns the true if list is empty, otherwise false.
abstract fun iterator(): Iterator<E>	It returns an iterator over the elements of this list.
abstract fun lastIndexOf(element: E): Int	It returns the index of last occurrence of specified element in the list, or return -1 if specified element is not present in list.
abstract fun listIterator(): ListIterator<E>	It returns a list iterator over the elements in proper sequence in current list.
abstract fun listIterator(index: Int): ListIterator<E>	It returns a list iterator over the elements in proper sequence in current list, starting at specified index.
abstract fun subList(fromIndex: Int, toIndex: Int): List	It returns a part of list between fromIndex (inclusive) to toIndex (exclusive).

Kotlin List Example 1

Let's see an example of list using *listOf()* function.

```
fun main(args: Array<String>){
   var list = listOf("Ajay","Vijay","Prakash")//read only, fix-size
   for(element in list){
      println(element)
   }
```

Output:

```
Ajay
Vijay
Prakash
```

Kotlin List Example 2

In the *listOf()* function we can pass the different types of data at the same time. List can also traverse the list using index range.

```
fun main(args: Array<String>){
   var list = listOf(1,2,3,"Ajay","Vijay","Prakash")//read only, fix-size
   for(element in list){
      println(element)
   }
   println()
   for(index in 0..list.size-1){
      println(list[index])
   }
}
```

Output:

```
1
2
3
Ajay
Vijay
Prakash
```

1
2
3
Ajay
Vijay
Prakash

Kotlin List Example 3

For more specific we can provide the *generic* types of list such as *listOf<Int>(), listOf<String>(), listOf<Any>()* Let's see the example.

```
fun main(args: Array<String>){
   var intList: List<Int> = listOf<Int>(1,2,3)
   var stringList: List<String> = listOf<String>("Ajay","Vijay","Prakash")
   var anyList: List<Any> = listOf<Any>(1,2,3,"Ajay","Vijay","Prakash")
   println("print int list")
   for(element in intList){
      println(element)
   }
   println()
   println("print string list")
   for(element in stringList){
      println(element)
   }
   println()
   println("print any list")
   for(element in anyList){
      println(element)
   }
}
```

Output:

```
print int list
1
2
3

print string list
Ajay
Vijay
Prakash

print any list
1
2
3
Ajay
Vijay
```

Prakash

Kotlin List Example 4

Let's see the use of different function of Kotlin list interface using *listOf<T>()* function.

```
fun main(args: Array<String>){
   var stringList: List<String> = listOf<String>("Ajay","Vijay","Prakash","Vijay","Rohan")
   var list: List<String> = listOf<String>("Ajay","Vijay","Prakash")
   for(element in stringList){
      print(element+" ")
   }
   println()
   println(stringList.get(0))
   println(stringList.indexOf("Vijay"))
   println(stringList.lastIndexOf("Vijay"))
   println(stringList.size)
   println(stringList.contains("Prakash"))
   println(stringList.containsAll(list))
   println(stringList.subList(2,4))
   println(stringList.isEmpty())
   println(stringList.drop(1))
   println(stringList.dropLast(2))
}
```

Output:

```
Ajay Vijay Prakash Vijay Rohan
Ajay
1
3
5
true
true
[Prakash, Vijay]
false
[Vijay, Prakash, Vijay, Rohan]
[Ajay, Vijay, Prakash]
```

The limitation of List interface is that it is immutable. It cannot add more elements in list after its declaration. To solve this limitation Collection framework provide mutable list.

Kotlin MutableList (mutableListOf())

Kotlin **MutableList** is an interface and generic collection of elements. MutableList interface is mutable in nature. It inherits form Collection<T> class. The methods of MutableList interface supports both read and write functionalities. Once the elements in MutableList have declared, it can be added more elements in it or removed, so it has no fixed size length.

To use the MutableList interface we use its function called *mutableListOf()* or *mutableListOf<E>()*.

The elements of MutableList follow the sequence of insertion order and contains index number same as array.

MutableList Interface Declaration

1. interface MutableList<E> : List<E>, MutableCollection<E> (source)

Function of Kotlin MutableList

There are several methods are available in MutableList interface. Some methods of MutableList interface are mention below.

Function	Descriptions
abstract fun add(element: E): Boolean	It adds the given element to the collection.
abstract fun add(index: Int, element: E)	It adds the element at specified index.
abstract fun addAll(elements: Collection<E>): Boolean	It adds all the elements of given collection to current collection.
abstract fun clear()	It removes all the elements from this collection.
abstract fun listIterator(): MutableListIterator<E>	It returns a list iterator over the elements in proper sequence in current list.
abstract fun listIterator(index: Int): MutableListIterator<E>	It returns a list iterator starting from specified index over the elements in list in proper sequence.
abstract fun remove(element: E): Boolean	It removes the specified element if it present in current collection.
abstract fun removeAll(elements: Collection<E>): Boolean	It removes all the elements from the current list which are also present in the specified collection.
abstract fun removeAt(index: Int): E	It removes element at given index from the list.
abstract fun retainAll(elements: Collection<E>): Boolean	It retains all the elements within the current collection which are present in given collection.
abstract operator fun set(index: Int, element: E): E	It replaces the element and add new at given index with specified element.
abstract fun subList(fromIndex: Int, toIndex: Int): MutableList<E>	It returns part of list from specified fromIndex (inclusive) to toIndex (exclusive) from current list. The returned list is backed by current list, so non-structural changes in the returned list are reflected in current list, and vice-versa.

Kotlin MutableList Example 1

Let's see an example of MutableList using *mutableListOf()* function and traverse its elements.

```
fun main(args: Array<String>){
  var mutableList = mutableListOf("Ajay","Vijay","Prakash","Vijay")
   for(element in mutableList){
     println(element)
  }
  println()
  for(index in 0..mutableList.size-1){
```

```
    println(mutableList[index])
  }
}
```

Output:

```
Ajay
Vijay
Prakash
Vijay

Ajay
Vijay
Prakash
Vijay
```

Kotlin MutableList Example 2

The function *mutableListOf()* of MutableList interface provides facilities to add elements after its declaration. MutableList can also be declared as empty and added elements later but in this situation we need to define its generic type. For example:

```
fun main(args: Array<String>){
  var mutableList1 = mutableListOf("Ajay","Vijay")
  mutableList1.add("Prakash")
  mutableList1.add("Vijay")
  var mutableList2 = mutableListOf<String>()
  mutableList2.add("Ajeet")
  mutableList2.add("Amit")
  mutableList2.add("Akash")
   for(element in mutableList1){
     println(element)
  }
  println()
  for(element in mutableList2){
     println(element)
  }
}
```

Output:

```
Ajay
Vijay
Prakash
Vijay

Ajeet
```

```
Amit
Akash
```

Kotlin MutableList Example 3

For more specific we can provide the generic types of MutableList interface such as *mutableListOf<Int>(), mutableListOf<String>(), mutableListOf<Any>()*. The *mutableListOf<Int>()* takes only integer value, *mutableListOf<String>()* takes only String value and *mutableListOf<Any>()* takes different data types value at the same time. Let's see the example.

```
fun main(args: Array<String>){
   var mutableListInt: MutableList<Int> = mutableListOf<Int>()
   var mutableListString: MutableList<String> = mutableListOf<String>()
   var mutableListAny: MutableList<Any> = mutableListOf<Any>()
   mutableListInt.add(5)
   mutableListInt.add(7)
   mutableListInt.add(10)
   mutableListInt.add(2,15) //add element 15 at index 2
   mutableListString.add("Ajeet")
   mutableListString.add("Ashu")
   mutableListString.add("Mohan")
   mutableListAny.add("Sunil")
   mutableListAny.add(2)
   mutableListAny.add(5)
   mutableListAny.add("Raj")
    println(".....print Int type.....")
   for(element in mutableListInt){
      println(element)
   }
   println()
   println(".....print String type.....")
   for(element in mutableListString){
      println(element)
   }
   println()
   println(".....print Any type.....")
   for(element in mutableListAny){
      println(element)
   }
}
```

Output:

```
.....print Int type.....
5
7
15
10

.....print String type.....
Ajeet
Ashu
Mohan

.....print Any type.....
Sunil
2
5
Raj
```

Kotlin MutableList Example 4

Let's see the use of different function of MutableList interface using *mutableListOf<T>()* function.

```
fun main(args: Array<String>){
   var mutableList = mutableListOf<String>()
   mutableList.add("Ajay") // index 0
   mutableList.add("Vijay") // index 1
   mutableList.add("Prakash") // index 2
   var mutableList2 = mutableListOf<String>("Rohan","Raj")
   var mutableList3 = mutableListOf<String>("Dharmesh","Umesh")
   var mutableList4 = mutableListOf<String>("Ajay","Dharmesh","Ashu")
   println(".....mutableList.....")
   for(element in mutableList){
      println(element)
   }
   println(".....mutableList[2].....")
   println(mutableList[2])
   mutableList.add(2,"Rohan")
   println("......mutableList.add(2,\"Rohan\").....")
   for(element in mutableList){
      println(element)
   }
```

```
mutableList.add("Ashu")
println(".....mutableList.add(\"Ashu\").....")
for(element in mutableList){
   println(element)
}
mutableList.addAll(1,mutableList3)
println("... mutableList.addAll(1,mutableList3)....")
for(element in mutableList){
   println(element)
}
mutableList.addAll(mutableList2)
println("...mutableList.addAll(mutableList2)....")
for(element in mutableList){
   println(element)
}
mutableList.remove("Vijay")
println("...mutableList.remove(\"Vijay\")....")
for(element in mutableList){
   println(element)
}
mutableList.removeAt(2)
println("....mutableList.removeAt(2)....")
for(element in mutableList){
   println(element)
}
mutableList.removeAll(mutableList2)
println("....  mutableList.removeAll(mutableList2)....")
for(element in mutableList){
   println(element)
}
println("....mutableList.set(2,\"Ashok\")....")
mutableList.set(2,"Ashok")
for(element in mutableList){
   println(element)
}
```

```
println(".... mutableList.retainAll(mutableList4)....")
mutableList.retainAll(mutableList4)
for(element in mutableList){
   println(element)
}
println(".... mutableList2.clear()....")
mutableList2.clear()
for(element in mutableList2){
   println(element)
}
println(".... mutableList2 after mutableList2.clear()....")
println(mutableList2)
println("....mutableList.subList(1,2)....")
println(mutableList.subList(1,2))
}
```

Output:

```
.....mutableList.....
Ajay
Vijay
Prakash
.....mutableList[2].....
Prakash
......mutableList.add(2,"Rohan").....
Ajay
Vijay
Rohan
Prakash
.....mutableList.add("Ashu").....
Ajay
Vijay
Rohan
Prakash
Ashu
... mutableList.addAll(1,mutableList3)....
Ajay
Dharmesh
Umesh
Vijay
Rohan
Prakash
```

```
Ashu
...mutableList.addAll(mutableList2)....
Ajay
Dharmesh
Umesh
Vijay
Rohan
Prakash
Ashu
Rohan
Raj
...mutableList.remove("Vijay")....
Ajay
Dharmesh
Umesh
Rohan
Prakash
Ashu
Rohan
Raj
....mutableList.removeAt(2)....
Ajay
Dharmesh
Rohan
Prakash
Ashu
Rohan
Raj
....  mutableList.removeAll(mutableList2)....
Ajay
Dharmesh
Prakash
Ashu
....mutableList.set(2,"Ashok")....
Ajay
Dharmesh
Ashok
Ashu
.... mutableList.retainAll(mutableList4)....
Ajay
Dharmesh
Ashu
.... mutableList2.clear()....
.... mutableList2 after mutableList2.clear()....
[]
....mutableList.subList(1,2)....
[Dharmesh]
```

Kotlin ArrayList class

Kotlin **ArrayList** class is used to create a dynamic array. Which means the size of ArrayList class can be increased or decreased according to requirement. ArrayList class provides both read and write functionalities.

Kotlin ArrayList class follows the sequence of insertion order. ArrayList class is non synchronized and it may contains duplicate elements. The elements of ArrayList class are accessed randomly as it works on index basis.

Constructor of Kotlin ArrayList

Constructor	Description
ArrayList<E>()	It is used to create an empty ArrayList
ArrayList(capacity: Int)	It is used to create an ArrayList of specified capacity.
ArrayList(elements: Collection<E>)	It is used to create an ArrayList filled from the elements of collection.

Functions of Kotlin ArrayList

Function	Description
open fun add(element: E): Boolean	It is used to add the specific element into the collection.
open fun add(index: Int, element: E)	It is used to insert an element at specific index.
open fun addAll(elements: Collection<E>): Boolean	It is used to add all the elements in the specified collection to current collection.
open fun addAll(index: Int, elements: Collection<E>): Boolean	It is used to add all the elements of specified collection into the current list at the specified index.
open fun clear()	It is used to removes all elements from the collection.
open fun get(index: Int): E	It is used to return the element at specified index in the list.
open fun indexOf(element: E): Int	It is used to return the index of first occurrence of specified element in the list or return -1 if the specified element in not present in the list.
open fun lastIndexOf(element: E): Int	It is used to return the last occurrence of given element from the list or it returns -1 if the given element is not present in the list.
open fun remove(element: E): Boolean	It is used to remove a single instance of the specific element from current collection, if it is available.
open fun removeAt(index: Int): E	It is used to remove the specific index element from the list.

open fun removeRange(startIndex: Int, endIndex: Int)	Its remove the range of elements starting from startIndex to endIndex in which endIndex is not includes.
open fun set(index: Int, element: E): E	It is used to replaces the element from the specified position from current list with the specified element.
open fun toArray(): Array<Any?>	It is used to return new array of type Array<Any?> with the elements of this collection.
open fun toString(): String	It is used to returns a string representation of the object.
fun trimToSize()	It does nothing in this ArrayList implementation.

Kotlin ArrayList Example 1- empty ArrayList

Let's create a simple example of ArrayList class define with empty ArrayList of String and add elements later.

```
fun main(args: Array<String>){
 val arrayList = ArrayList<String>()//Creating an empty arraylist
   arrayList.add("Ajay")//Adding object in arraylist
   arrayList.add("Vijay")
   arrayList.add("Prakash")
   arrayList.add("Rohan")
   arrayList.add("Vijay")
   println(".....print ArrayList.....")
   for (i in arrayList) {
      println(i)
   }
}
```

Output:

```
.....print ArrayList.....
Ajay
Vijay
Prakash
Rohan
Vijay
```

Kotlin ArrayList Example 2- initialize ArrayList capacity

Let's create an ArrayList class with initialize its initial capacity. The capacity of ArrayList class is not fixed and it can be change later in program according to requirement.

```
fun main(args: Array<String>){
 val arrayList1 = ArrayList<String>(5)
   arrayList1.add("Ajay")//Adding object in arraylist
   arrayList1.add("Vijay")
   arrayList1.add("Prakash")
   arrayList1.add("Rohan")
   arrayList1.add("Vijay")
   println(".....print ArrayList1.....")
   for (i in arrayList1) {
```

```
    println(i)
  }
  println("size of arrayList1 = "+arrayList1.size)
  val arrayList2 = ArrayList<Int>(5)
  arrayList2.add(14)
  arrayList2.add(20)
  arrayList2.add(80)
  println("......print ArrayList2......")
  for (i in arrayList2) {
    println(i)
  }
  println("size of arrayList2 = "+arrayList2.size)
}
```

Output:

```
.......print ArrayList1......
Ajay
Vijay
Prakash
Rohan
Vijay
size of arrayList1 = 5
......print ArrayList2......
14
20
80
size of arrayList2 = 3
```

Kotlin ArrayList Example 3- filled elements in ArrayList using collection

The elements in Kotlin ArratList class can also be added using other collection. For more specific in ArrayList class it is declared by its generic types. Elements of ArrayList class also be traverse using iterator() function. For example:

```
fun main(args: Array<String>){
 val arrayList: ArrayList<String> = ArrayList<String>(5)
 var list: MutableList<String> = mutableListOf<String>()
  list.add("Ajay")
  list.add("Vijay")
  list.add("Prakash")
   arrayList.addAll(list)
   println("......print ArrayList.....")
   val itr = arrayList.iterator()
   while(itr.hasNext()) {
     println(itr.next())
   }
   println("size of arrayList = "+arrayList.size)
}
```

Output:

```
.......print ArrayList.......
Ajay
```

```
Vijay
Prakash
size of arrayList = 3
```

Kotlin ArrayList Example 4 - get()

The get() function of ArrayList class is used to retrieve the element present at given specified index. For **example:**

```
fun main(args: Array<String>){
 val arrayList: ArrayList<String> = ArrayList<String>(5)
  arrayList.add("Ajay")
   arrayList.add("Vijay")
   arrayList.add("Prakash")
    arrayList.add("Rohan")
    arrayList.add("Vijay")
    println(".......print ArrayList.......")
    for (i in arrayList) {
        println(i)
    }
    println(".......arrayList.get(2).......")
    println( arrayList.get(2))
}
```

Output:

```
.......print ArrayList.......
Ajay
Vijay
Prakash
Rohan
Vijay
.......arrayList.get(2).......
Prakash
```

Kotlin ArrayList Example 5 - set()

The set() function of ArrayList class is used to set the given element at specified index and replace if any element present at specified index. For example:

```
fun main(args: Array<String>){
   val arrayList: ArrayList<String> = ArrayList<String>(5)
    arrayList.add("Ajay")
    arrayList.add("Vijay")
    arrayList.add("Prakash")
    arrayList.add("Rohan")
    arrayList.add("Vijay")
    println(".......print ArrayList.......")
    for (i in arrayList) {
        println(i)
    }
    println(".......arrayList.set(2,\"Ashu\").......")
    arrayList.set(2,"Ashu")
```

```
   println(".......print ArrayList.......")
   for (i in arrayList) {
      println(i)
   }
}
```

Output:

```
.......print ArrayList.......
Ajay
Vijay
Prakash
Rohan
Vijay
.......arrayList.set(2,"Ashu").......
.......print ArrayList.......
Ajay
Vijay
Ashu
Rohan
Vijay
```

Kotlin ArrayList: arrayListOf()

An **arrayListOf()** is a function of **ArrayList** class. ArrayList is mutable which means it provides both read am write functionalities. The arrayListOf() function returns an ArrayList type.

Syntax of arrayListOf() function

```
inline fun <T> arrayListOf(): ArrayList<T> (source)
fun <T> arrayListOf(vararg elements: T): ArrayList<T> (source)
```

Function of Kotlin ArrayList

Function	**Description**
open fun add(element: E): Boolean	It is used to add the specific element into the collection.
open fun add(index: Int, element: E)	It is used to insert an element at specific index.
open fun addAll(elements: Collection<E>): Boolean	It is used to add all the elements in the specified collection to current collection.
open fun addAll(index: Int, elements: Collection<E>): Boolean	It is used to add all the elements of specified collection into the current list at the specified index.
open fun clear()	It is used to removes all elements from the collection.
open fun get(index: Int): E	It is used to return the element at specified index in the list.
open fun indexOf(element: E): Int	It is used to return the index of first occurrence of specified element in the list or return -1 if the specified element in not present in the list.
open fun lastIndexOf(element: E): Int	It is used to return the last occurrence of given element from

	the list or it returns -1 if the given element is not present in the list.
open fun remove(element: E): Boolean	It is used to remove a single instance of the specific element from current collection, if it is available.
open fun removeAt(index: Int): E	It is used to remove the specific index element from the list.
open fun removeRange(startIndex: Int, endIndex: Int)	Its remove the range of elements starting from startIndex to endIndex in which endIndex is not includes.
open fun set(index: Int, element: E): E	It is used to replaces the element from the specified position from current list with the specified element.
open fun toArray(): Array<Any?>	It is used to return new array of type Array<Any?> with the elements of this collection.
open fun toString(): String	It is used to returns a string representation of the object.
fun trimToSize()	It does nothing in this ArrayList implementation.

Kotlin arrayListOf() Example 1

Let's create a simple example of arrayListOf() function.

```
fun main(args: Array<String>){
    var arrayList = arrayListOf<Int>(4,7,12)
     for(element in arrayList){
       println(element)
   }
}
```

Output:

```
4
7
12
```

Kotlin arrayListOf() Example 2

For more specific we can define the generic types of arrayListOf() function such as arrayListOf<Int>(), arrqayListOf<String>(),arrayListOf<Any>(). Let's see the example.

```
fun main(args: Array<String>){
   var intArrayList: ArrayList<Int> = arrayListOf<Int>(1,2,3)
   var stringArrayList: ArrayList<String> = arrayListOf<String>("Ajay","Vijay","Prakash")
   var anyArrayList: ArrayList<Any> = arrayListOf<Any>(1,2,3,"Ajay","Vijay","Prakash")
   println("print int ArrayList")
   for(element in intArrayList){
      println(element)
   }
   println()
   println("print string ArrayList")
   for(element in stringArrayList){
```

```
      println(element)
   }
      println()
   println("print any ArrayList")
   for(element in anyArrayList){
      println(element)
   }
}
```

Output:

```
print int ArrayList
1
2
3

print string ArrayList
Ajay
Vijay
Prakash

print any ArrayList
1
2
3
Ajay
Vijay
Prakash
```

Kotlin arrayListOf() Example 3- iterator() function

The elements of ArrayList class is also be traverse using inbuilt *iterator()* function. For example:

```
fun main(args: Array<String>){
   val list: ArrayList<String> = arrayListOf<String>()

   list.add("Ajay")
   list.add("Vijay")
   list.add("Prakash")
   println(".......print ArrayList.......")
   val itr = list.iterator()
   while(itr.hasNext()) {
      println(itr.next())
   }
}
```

Output:

```
.......print ArrayList.......
Ajay
Vijay
Prakash
```

Kotlin arrayListOf() Example 4 - get()

The *get()* function of arrayListOf() is used to retrieve the element present at specified index. For example:

```
fun main(args: Array<String>){
  val list: ArrayList<String> = arrayListOf<String>()
  list.add("Ajay")
   list.add("Vijay")
   list.add("Prakash")
   list.add("Rohan")
   list.add("Vijay")
   println(".......print list.......")
   for (i in list) {
      println(i)
   }
   println(".......list.get(2).......")
   println( list.get(2))
}
```

Output:

```
.......print list.......
Ajay
Vijay
Prakash
Rohan
Vijay
.......list.get(2).......
Prakash
```

Kotlin arrayListOf() Example 5 - set()

The *set()* function of arrayListOf() is used to set the given element at specified index and replace if any element already present at that index. For example:

```
fun main(args: Array<String>){
 val list: ArrayList<String> = arrayListOf<String>()
    list.add("Ajay")
   list.add("Vijay")
   list.add("Prakash")

   println(".......print list.......")
   for (i in list) {
      println(i)
   }
   println(".......arrayList.set(2,\"Rohan\").......")
   list.set(2,"Rohan")
   println(".......print ArrayList.......")
   for (i in list) {
      println(i)
   }
```

```
}
```

Output:

```
.......print list.......
Ajay
Vijay
Prakash
.......list.set(2,"Rohan").......
.......print list.......
Ajay
Vijay
Rohan
```

Kotlin arrayListOf() Example - add and print Employee data

Let's create an another example of arrayListOf() function of ArrayList class. In this example we are adding and traversing Employee class data. Here Employee class is bean class which defines the property of Employee.

```
class Employee(var id: Int, var name: String, var phone: Int, var city: String)
fun main(args: Array<String>){
   val arrayList: ArrayList<Employee> = arrayListOf<Employee>()
   val e1 = Employee(101, "Ajay", 55555, "Delhi")
   val e2 = Employee(102, "Rahul", 44443, "Mumbai")
   val e3 = Employee(103, "Sanjay", 45422, "Noida")
   arrayList.add(e1)
   arrayList.add(e2)
   arrayList.add(e3)
   for (e in arrayList) {
      println("${e.id} ${e.name} ${e.phone} ${e.city}")
   }
}
```

Output:

```
101 Ajay 55555 Delhi
102 Rahul 44443 Mumbai
103 Sanjay 45422 Noida
```

Kotlin Map Interface

Kotlin **Map** is an interface and generic collection of elements. Map interface holds data in the form of key and value pair. Map key are unique and holds only one value for each key. The key and value may be of different pairs such as <Int, Int>,<Int, String>, <Char, String>etc. This interface is immutable, fixed size and its methods support read only access.

To use the Map interface we need to use its function called **mapOf() or mapOf<k,v>().**

Map Interface Declaration

```
interface Map<K, out V> (source)
```

Properties of Map Interface

Properties	Description
abstract val entries:	It returns only read all key and value pair of Set Interface in current

Set<Entry<K, V>>	map.
abstract val keys: Set<K>	It returns only read all key of Set Interface in current map.
abstract val keys: Set<K>	It returns the number of key and value pair in current map.
abstract val values: Collection<V>	It returns only read Collection of all valued in current map. This collection may contain duplicate values.

Functions of Map Interface

There are several functions are available in Map interface. Some functions of Map interface are mention below.

Functions	**Description**
fun <K, V> Map<key, value>.getValue(key: K): V	It returns a value of given key or throws an exception if no such key is available in the map.
operator fun <V, V1 : V> Map<in String, V>.getValue(thisRef: Any?, property: KProperty<*>): V1	It returns the value of the property for the given object from current read- only map.
operator fun <K, V> Map<out K, V>.contains(key: K): Boolean	It checks is the given key contains in map.
fun <K> Map<out K, *>.containsKey(key: K): Boolean	If map contains the specified key it returns true.
fun <K, V> Map<K, V>.containsValue(value: V): Boolean	If map maps one or more keys to specified value it returns true.
fun <K, V> Map<out K, V>.getOrDefault(key: K, defaultValue: V): V	It returns the value which is given by key in mapped, or returns default value if map dose not contains mapping for the given key.
fun <K, V> Map<out K, V>.asIterable(): Iterable<Entry<K, V>>	It creates an instance of Iterable interface which wraps the original map returning its entries when being iterated.
fun <K, V> Map<out K, V>.asIterable(): Iterable<Entry<K, V>>	It creates an instance of Iterable interface which wraps the original map returning its entries when being iterated.
fun <K, V> Map<out K, V>.asSequence(): Sequence<Entry<K, V>>	It creates a Sequence interface instance which wraps the current map and returning its entries when it has iterated.
operator fun <K, V> Map<out K, V>.iterator(): Iterator<Entry<K, V>>	It returns an Iterator over the entries in the Map.
operator fun Map.minus(key: K): Map	It returns a map which contains all the entries of original map except the entry of mention key.
operator fun <K, V> Map<out K, V>.minus(keys: Iterable<K>): Map<K, V>	It returns a map which contains all the entries of original map except those entries key which are contained in the mention key collection.
operator fun <K, V> Map<out K,	It returns a map which contains all the entries of original

V>.minus(keys: Sequence<K>): Map<K, V>	map except those entries key which are contained in the given key sequence.
operator fun <K, V> Map<out K, V>.plus(pair: Pair<K, V>): Map<K, V>	It creates a new read only map by adding or replacing an entry to current map from a given key-value pair.
operator fun <K, V> Map<out K, V>.plus(pairs: Iterable<Pair<K, V>>): Map<K, V>	It creates a new read only map by adding or replacing entries to current map from a given collection of key-value pairs.
operator fun <K, V> Map<out K, V>.plus(pairs: Sequence<Pair<K, V>>): Map<K, V>	It creates a new read only map by adding or replacing entries to current map from a given sequence of key-value pairs.

Kotlin Map Interface Example

Let's create an example of declaring and traversing the value of map using mapOf<k,v>() function. In this example, we create key of Int and value of String types.

```
fun main(args: Array<String>){
 val myMap = mapOf<Int,String>(1 to "Ajay", 4 to "Vijay", 3 to "Prakash")
   for(key in myMap.keys){
      println(myMap[key])
   }
}
```

Output:

```
Ajay
Vijay
Prakash
```

Kotlin HashMap class

Kotlin HashMap is class of collection based on MutableMap interface. Kotlin HashMap class implements the MutableMap interface using Hash table. It store the data in the form of key and value pair. It is represented as HashMap<key, value> or HashMap<K, V>.

The implementation of HashMap class does not make guarantees about the order of data of key, value and entries of collections.

Constructor of Kotlin HashMap class

Constructor	Description
HashMap()	It constructs an empty HashMap instance
HashMap(initialCapacity: Int, loadFactor: Float = 0f)	It is used to constructs a HashMap of specified capacity.
HashMap(original: Map<out K, V>)	It constructs a HashMap instance filled with contents of specified original map.

Functions of Kotlin HashMap class

Functions	Description
open fun put(key: K, value: V): V?	It puts the specified key and value in the map

open operator fun get(key: K): V?	It returns the value of specified key, or null if no such specified key is available in map.
open fun containsKey(key: K): Boolean	It returns true if map contains specifies key.
open fun containsValue(value: V): Boolean	It returns true if map maps one of more keys to specified value.
open fun clear()	It removes all elements from map.
open fun remove(key: K): V?	It removes the specified key and its corresponding value from map

Kotlin HashMap Example - empty HashMap

Let's create a simple example of HashMap class define with empty HashMap of <Int, String> and add elements later. To print the value of HashMap we will either use *HashMap[key]* or *HashMap.get(key)*.

```
fun main(args: Array<String>){
  val hashMap:HashMap<Int,String> = HashMap<Int,String>() //define empty hashmap
   hashMap.put(1,"Ajay")
   hashMap.put(3,"Vijay")
   hashMap.put(4,"Praveen")
   hashMap.put(2,"Ajay")
   println(".....traversing hashmap.......")
   for(key in hashMap.keys){
      println("Element at key $key = ${hashMap[key]}")
   }}
```

Output:

```
.....traversing hashmap.......
Element at key 1 = Ajay
Element at key 2 = Ajay
Element at key 3 = Vijay
Element at key 4 = Praveen
```

Kotlin HashMap: hashMapOf()

A **hashMapOf()** is a function of **HashMap** class. It returns a new HashMap with the specified contents. It contains pairs of data in the form of key and value. HashMap is mutable collection which provides both read am write functionalities.

Syntax of hashMapOf() function

inline fun <K, V> hashMapOf(): HashMap<K, V> (source)

fun <K, V> hashMapOf(vararg pairs: Pair<K, V>): HashMap<K, V> (source)

Functions of Kotlin HashMap class

Function	Description
open fun put(key: K, value: V): V?	It puts the specified key and value in the map
open operator fun get(key: K): V?	It returns the value of specified key, or null if no such specified key is available in map.
open fun containsKey(key: K): Boolean	It returns true if map contains specifies key.
open fun containsValue(value: V): Boolean	It returns true if map maps one of more keys to specified value.
open fun clear()	It removes all elements from map.
open fun remove(key: K): V?	It removes the specified key and its corresponding value from map

Kotlin hashMapOf() Example

The hashMapOf() function of HashMap can be declared as different generic types such as hashMapOf<Int, String>(), hashMapOf<String, String>(), hashMapOf<Any, Any>() etc.

```
fun main(args: Array<String>){
 val intMap: HashMap<Int, String> = hashMapOf<Int,String>(1 to "Ashu",4 to "Rohan", 2 to "Ajeet", 3 to "Vi
jay")
   val stringMap: HashMap<String,String> = hashMapOf<String,String>("name" to "Ashu")
   stringMap.put("city", "Delhi")
   stringMap.put("department", "Development")
   stringMap.put("hobby", "Playing")
   val anyMap: HashMap<Any, Any> = hashMapOf<Any, Any>(1 to "Ashu", "name" to "Rohsan", 2 to 200)
   println(".....traverse intMap........")
   for(key in intMap.keys){
      println(intMap[key])
   }
   println("......traverse stringMap.......")
   for(key in stringMap.keys){
      println(stringMap[key])
   }
   println("......traverse anyMap.......")
   for(key in anyMap.keys){
      println(anyMap[key])
   }
}
```

Output:

```
.....traverse intMap........
Ashu
Ajeet
Vijay
Rohan
......traverse stringMap.......
Ashu
Development
Delhi
Playing
......traverse anyMap.......
Rohsan
Ashu
200
```

Kotlin MutableMap Interface

Kotlin **MutableMap** is an interface of collection framework that holds the object in the form of key and value pair. The values of MutableMap interface are retrieved by using their corresponding keys. The key and value may be of different pairs such as <Int, Int>,<Int, String>, <Char, String> etc. Each key of MutableMap holds only one value.

To use the MutableMap interface we need to use its function called **mutableMapOf()** or **mutableMapOf <k,v>()**.

Kotlin MutableMap Interface Declaration

interface MutableMap<K, V> : Map<K, V> (source)

Properties of MutableMap

Properties	Description
abstract val entries: MutableSet<MutableEntry<K, V>>	This returns a MutableSet of all its key and value pairs in the map.
abstract val keys: MutableSet<K>	This returns all the keys of MutableSet in this map.
abstract val values: MutableCollection<V>	This returns all the values of MutableCollection in the current map. This collection may contain duplicate values.

Function of Kotlin MutableMap

Function	Description
abstract fun put(key: K, value: V): V?	**It adds the given value with the specified key in the map.**
abstract fun putAll(from: Map<out K, V>)	**This updates the current map with key/value pairs from the mentioned map.**
abstract fun remove(key: K): V?	**It removes the specified key with its corresponding value from the map.**
open fun remove(key: K, value: V): Boolean	**It removes the key and value entities from the map only if it exist in the map.**
abstract fun clear()	**This function is used to removes all the elements from the map.**
operator fun <K, V> Map<out K, V>.contains(key: K): Boolean	**It checks the given key in the map.**
abstract fun containsKey(key: K): Boolean	**It returns the true if map contains the specified key.**
fun <K> Map<out K, *>.containsKey(key: K): Boolean	**It returns the true if map contains the specified key.**
abstract fun containsValue(value: V): Boolean	**It returns true if the map maps one or more keys for the given value.**
fun <K, V> Map<K, V>.containsValue(value: V): Boolean	**It returns true if the map maps one or more keys for the given value.**
fun <K, V> Map<out K, V>.count(): Int	**It returns the total number of entities of the map**
operator fun <K, V> Map<out K, V>.get(key: K): V?	**It returns the value corresponding to mention key, or null if no such key found in the map.**
fun <K, V> Map<out K, V>.getOrDefault(key: K, defaultValue: V): V	**It returns the value with corresponding mention key, or it returns default value if no such mapping for the key in the map.**
fun <K, V> Map<K, V>.getOrElse(key: K,	**It returns the value for the mention key in the map, or it returns the default value function if no such entry found for the given key.**

defaultValue: () -> V): V	
fun <K, V> Map<K, V>.getValue(key: K): V	**It returns the value corresponding to given key, or it throws an exception if no key found in the map.**

Kotlin MutableMap Example - traversing MutableMap

Let's create an example to create a MutableMap using mutablemapOf() function and traverse it. In this example we create three different types (MutableMap<Int, String>, MutableMap<String, String> and MutableMap<Any, Any>) of MutableMap with different ways.

```
fun main(args: Array<String>) {
  val mutableMap1: MutableMap<Int, String> = mutableMapOf<Int, String>(1 to "Ashu", 4 to "Rohan", 2 to "Ajeet",
3 to "Vijay")
   val mutableMap2: MutableMap<String, String> = mutableMapOf<String, String>()
   mutableMap2.put("name", "Ashu")
   mutableMap2.put("city", "Delhi")
   mutableMap2.put("department", "Development")
   mutableMap2.put("hobby", "Playing")
 val mutableMap3: MutableMap<Any, Any> = mutableMapOf<Any, Any>(1 to "Ashu", "name" to "Rohsan", 2 to 200)
     println(".....traverse mutableMap1.....")
   for (key in mutableMap1.keys) {
      println("Key = ${key}, Value = ${mutableMap1[key]}")
   }
   println("......traverse mutableMap2.......")
   for (key in mutableMap2.keys) {
      println("Key = "+key +", "+"Value = "+mutableMap2[key])
   }
   println("......traverse mutableMap3......")
   for (key in mutableMap3.keys) {
      println("Key = ${key}, Value = ${mutableMap3[key]}")
   }
}
```

Output:

```
.....traverse mutableMap1.......
Key = 1, Value = Ashu
Key = 4, Value = Rohan
Key = 2, Value = Ajeet
Key = 3, Value = Vijay
......traverse mutableMap2.......
Key = name, Value = Ashu
Key = city, Value = Delhi
Key = department, Value = Development
Key = hobby, Value = Playing
......traverse mutableMap3......
Key = 1, Value = Ashu
Key = name, Value = Rohsan
Key = 2, Value = 200
```

Kotlin Set Interface

Kotlin **Set** *interface* is a generic unordered collection of elements. Set interface does not support duplicate elements. This interface is immutable in nature its methods supports read-only functionality of the set.

Set interface uses *setOf()* function to create the list of object of set interface which contains list of elements.

Set Interface declaration
interface Set<out E> : Collection<E> (source)

Properties of Set Interface

Properties	Description
abstract val size: Int	**It returns the size of collection.**

Functions of Set Interface

Kotlin Set **interface has several functions. Some of its functions are mention below.**

Functions	Description
abstract fun contains(element: E): Boolean	**It checks the mention element is present in this collection. If it contains element, it returns true else returns false.**
abstract fun containsAll(elements: Collection<E>): Boolean	**It checks all the mention elements of specified collection are present in this collection. If it contains element, it returns true else returns false.**
abstract fun isEmpty(): Boolean	**It returns true if the collection is empty (contains no elements) otherwise it returns false.**
abstract fun iterator(): Iterator<E>	**It returns an iterator over the elements of set object.**
fun <T> Iterable<T>.all(predicate: (T) -> Boolean): Boolean	**It returns true if all the elements matches with given predicate.**
fun <T> Iterable<T>.any(): Boolean	**It returns true if the collection contains at least one element.**
fun <T> Iterable<T>.count(predicate: (T) -> Boolean): Int	**It returns the total number of elements matching with given predicate.**
fun <T> Iterable<T>.distinct(): List<T>	**It returns a list which contains only distinct elements from the given collection.**
fun <T> Iterable<T>.drop(n: Int): List<T>	**It returns a list which contains all elements except first n elements.**
fun <T> Iterable<T>.elementAtOrElse(index: Int, defaultValue: (Int) -> T): T	**It returns an element at given index or result of calling the defaultValue function if the index is out bounds in current collection.**
fun <T> Iterable<T>.filter(predicate: (T) -> Boolean): List<T>	**It returns a list which contains only those elements matches with given predicate.**
fun <T> Iterable<T>.filterIndexed(predicate: (index: Int, T) -> Boolean): List<T>	**It returns a list which contains only those elements matches with given predicate.**

fun <T> Iterable<T>.filterNot(predicate: (T) -> Boolean): List<T>	It returns a list which contains only those elements which does not matches with given predicate.
fun <T> Iterable<T>.find(predicate: (T) -> Boolean): T?	It returns the first element which matches with given predicate, or *null* if no such element was found.
fun <T> Iterable<T>.findLast(predicate: (T) -> Boolean): T?	It returns the last element which matches with given predicate, or *null* if no such element was found.
fun <T> Iterable<T>.first(): T	It returns the first element.
fun <T> Iterable<T>.first(predicate: (T) -> Boolean): T	It returns the first element which matches the given predicate.
fun <T> Iterable<T>.firstOrnull(): T?	It returns the first element or *null* if collection is empty.
fun <T> Iterable<T>.indexOf(element: T): Int	It returns the first index of given element, or -1 if element does not contains in collection.
fun <T> Iterable<T>.indexOfFirst(predicate: (T) -> Boolean): Int	It returns the index of first element which matches the given predicate, or -1 if the element does not contains in collection.
fun <T> Iterable<T>.indexOfLast(predicate: (T) -> Boolean): Int	It returns the index of last element which matches the given predicate, or -1 if the element does not contains in collection.
infix fun <T> Iterable<T>.intersect(other: Iterable<T>): Set<T>	It returns a set which contains all elements present in both this set and given collection.
fun <T> Collection<T>.isNotEmpty(): Boolean	It returns true if is not empty.
fun <T> Iterable<T>.last(): T	It returns the last element.
fun <T> Iterable<T>.last(predicate: (T) -> Boolean): T	It returns the last element which matches with given predicate.
fun <T> Iterable<T>.lastIndexOf(element: T): Int	It returns the last index of given element, or -1 if element does not exist in collection.
fun <T> Iterable<T>.lastOrnull(): T?	It returns the last element of collection, or *null* if collection is empty.
fun <T> Iterable<T>.lastOrnull(predicate: (T) -> Boolean): T?	It returns the last element after matching the given predicate, or returns *null* if no such element found in collection.

fun <T : Comparable<T>> Iterable<T>.max(): T?	**It returns the largest element or *null* if no elements in collection.**
fun <T, R : Comparable<R>> Iterable<T>.maxBy(selector: (T) -> R): T?	**It returns the first element yielding the largest value of the given function, or it returns *null* if there are no elements in collection.**
fun <T : Comparable<T>> Iterable<T>.min(): T?	**It returns the smallest element or *null* if there is no element in the collection.**
fun <T, R : Comparable<R>> Iterable<T>.minBy(selector: (T) -> R): T?	**It returns the first element which gives the smallest value of the given function or *null* if there are no elements.**
operator fun <T> Set<T>.minus(element: T): Set<T>	**It returns a set which contains all the elements of original set except those given element.**
operator fun <T> Set<T>.minus(elements: Iterable<T>): Set<T>	**It returns a set which contains all the elements of original set except those given elements collection.**
operator fun <T> Iterable<T>.minus(element: T): List<T>	**It returns a list which contains all the elements of original collection except those contained in the given elements array.**
fun <T> Set<T>.minusElement(element: T): Set<T>	**It returns a set which contains all the elements of original set except those given element.**
fun <T> Iterable<T>.minusElement(element: T): List<T>	**It returns a list which contains all the elements of original collection except the first occurrence of the given element.**
operator fun <T> Set<T>.plus(element: T): Set<T>	**It returns a set of all elements of original set as well as the given element if it is not already present in the set.**
operator fun <T> Set<T>.plus(elements: Iterable<T>): Set<T>	**It returns a set which contains all the elements of original set as well as the given elements collection which are not already present in the set. The returned set preserves the iteration of element in the same order of the original set.**
operator fun <T> Iterable<T>.plus(element: T): List<T>	**It returns a list which contains all the elements of the original collection as well as the given element.**
fun <T> Set<T>.plusElement(element: T): Set<T>	**It returns a set which contains all the elements of the original set as well as the given element.**
fun <T> Iterable<T>.plusElement(element: T): List<T>	**It returns a list which contains all the elements of the original collection as well as the given element.**

fun <T> Iterable<T>.reversed(): List<T>	**It returns a list with elements in the reverse order.**
fun <T> Iterable<T>.single(): T	**It returns the single element, or it throws an exception if the collection has more than one elements or empty.**
fun <T> Iterable<T>.singleOrnull(): T?	**It returns a single element, or *null* if the collection has more than one element or it is empty.**

Kotlin Set Interface Example

Let create an example of declaring and traversing set element using *setOf()* function. In this example we create a set of Int type non generic and another generic set of *Any* type.

```
fun main(args: Array<String>){
   val intSet = setOf(2,6,4,29,4,5)
   val mySet: Set<Any> = setOf(2,6,4,29,4,5,"Ashu","Ajay")
   println(".......print Int set.........")
   for(element in intSet){
      println(element)
   }
   println(".......print Any set.........")
   for(element in mySet){
      println(element)
   }

}
```

Output:

```
.......print Int set.........
2
6
4
29
5
.......print Any set.........
2
6
4
29
5
Ashu
Ajay
```

In the above example we declare element 4 twice in both *intSet* and *mySet* but while traversing them they print the element 4 only once. This is because the set interface does not support duplicate elements.

Kotlin MutableSet Interface

Kotlin **MutableSet** interface is a generic unordered collection of elements. MutableSet interface does not support duplicate elements. This interface is mutable so its methods support read-write functionality supports adding and removing elements.

Set interface uses **mutableSetOf()** function to create the list of object of set interface which contains list of elements.

MutableSet Interface declaration

1. interface MutableSet<E> : Set<E>, MutableCollection<E> (source)

Inherited Properties of MutableSet Interface

Properties	Description
abstract val size: Int	**It returns the size of collection.**

Functions of MutableSet Interface

Kotlin MutableSet **interface has several functions. Some of its functions are mention below.**

Functions	Description
abstract fun add(element: E): Boolean	**It adds the given element to the collection.**
abstract fun addAll(elements: Collection<E>): Boolean	**It adds all the elements given collection to the current collection.**
abstract fun clear()	**It removes all the elements from this collection.**
abstract fun iterator(): MutableIterator<E>	**It returns an iterator over the elements of this object.**
abstract fun remove(element: E): Boolean	**It removes a single specified element from this collection, if it is present in collection.**
abstract fun removeAll(elements: Collection<E>): Boolean	**It removes all the elements from current collection which are given in collection.**
abstract fun retainAll(elements: Collection<E>): Boolean	**It retains only those elements in current collection which are present in specified collection.**
abstract fun contains(element: E): Boolean	**It checks the specified element is contained in current collection.**
abstract fun containsAll(elements: Collection<E>): Boolean	**It checks all the elements of specified collection are present in current collection.**
abstract fun isEmpty(): Boolean	**If collection is empty (not containing any element) it returns true, otherwise it returns false.**
fun <T> Iterable<T>.any(): Boolean	**It returns true if collection contains at least one element.**
fun <T> Iterable<T>.any(predicate: (T) -> Boolean): Boolean	**It returns true if at least element matches the given the given predicate.**
fun <T> Iterable<T>.distinct(): List<T>	**It returns a list which contains only distinct elements from the given collection.**
fun <T> Iterable<T>.drop(n: Int): List<T>	**It returns a list which contains all elements except first n elements.**
fun <T> Iterable<T>.elementAt(index: Int): T	**It returns an element at given index or throw an IndexOutOfBoundException if given index is not present in collection.**

fun <T> Iterable<T>.elementAtOrElse(index: Int, defaultValue: (Int) -> T): T	**It returns an element at given index or result of calling the defaultValue function if the index is out bounds in current collection.**
fun <T : Comparable<T>> Iterable<T>.max(): T?	**It returns the largest element or null if there is no element in the collection.**
fun <T : Comparable<T>> Iterable<T>.min(): T?	**It returns the smallest element or null if there is no element in the collection.**
fun <T> MutableCollection<out T>.remove(element: T): Boolean	**It removes the single specified element if it in present in current collection.**
fun <T> MutableCollection<out T>.removeAll(elements: Collection<T>): Boolean	**It removes all the elements of current collection which are contained in specified collection.**
fun <T> MutableCollection<out T>.retainAll(elements: Collection<T>): Boolean	**It retains all the elements in current collection which are contained in specified collection.**
fun <T> Iterable<T>.reversed(): List<T>	**It returns the elements in reversed order.**

Kotlin MutableSet Interface Example

Let's create an example of MutableSet declaring and traversing its elements.

```
fun main(args: Array<String>) {
   val intmutableSet = mutableSetOf<Int>(2, 6, 4, 29, 4, 5)
        val anymutableSet: Set<Any> = setOf(2, 6, 4, 29, 4, 5, "Ajay", "Ashu", "Ajay")
   println("....intmutableSet....")
   for(element in intmutableSet){
      println(element)
   }
   println("....anymutableSet......")
   for(element in anymutableSet){
      println(element)
   }
}
```

Output:

```
....intmutableSet....
2
6
4
29
5
....anymutableSet......
2
6
```

```
4
29
5
Ajay
Ashu
```

In the above example, elements "4" and "Ajay" are declared twice. But while traversing these MutableSet they are printed only once, this is because MutableSet interface does not support duplicate elements.

Kotlin HashSet class

Kotlin **HashSet** is class of collection which extends AbstractMutableSet class and implements Set interface. The HashSet class store elements using hashing mechanism. It support both read and write functionality. It does not support duplicate value and does not make guarantees about the order sequence of element.

HashSet class declaration

1. open class HashSet<E> : AbstractMutableSet<E> (source)

Constructor of Kotlin HashSet class

Constructor	Description
HashSet()	**It constructs an empty HashSet instance**
HashSet(initialCapacity: Int, loadFactor: Float = 0f)	**It is used to constructs a HashSet of specified capacity.**
HashSet(elements: Collection<E>)	**It constructs a HashSet instance using elements of specified collection.**

Functions of Kotlin HashSet class

Functions	Description
open fun add(element: E): Boolean	**It adds the given element to the collection.**
open operator fun contains(element: E): Boolean	**It checks the specified element is present in current collection.**
open fun isEmpty(): Boolean	**It checks the current collection is empty (not contain any element). If found collection is empty returns *true* otherwise *false*.**
open fun iterator(): MutableIterator<E>	**It returns an iterator over the elements of current object.**
open fun remove(element: E): Boolean	**It removes the mention element if present in current collection. It returns true if it removes otherwise false.**
open fun clear()	**It deletes all the elements from this collection.**

Property of Kotlin HashSet class

Property	Description
open val size: Int	**This property is used to return the size of HashSet collection.**

Kotlin HashSet Example - capacity

Let's create an example of HashSet defining it capacity. Capacity defines the total number of element to be added in the HashSet. It can be increase of decrease later according to need.

```
fun main(args: Array<String>){
   var hashSet = HashSet<Int>(6)
   hashSet.add(2)
   hashSet.add(13)
   hashSet.add(6)
   hashSet.add(5)
   hashSet.add(2)
   hashSet.add(8)
   println("......traversing hashSet......")
   for (element in hashSet){
      println(element)
   }
}
```

Output:

```
......traversing hashSet......
8
2
13
5
6
```

Kotlin Function

Function is a group of inter related block of code which performs a specific task. Function is used to break a program into different sub module. It makes reusability of code and makes program more manageable.

In Kotlin, functions are declared using **fun** keyword. There are two types of functions depending on whether it is available in standard library or defined by user.

Standard library function

User defined function

Standard Library Function

Kotlin Standard library function is built-in library functions which are implicitly present in library and available for use.

For example

```
fun main(args: Array<String>){
   var number = 25
   var result = Math.sqrt(number.toDouble())
   print("Square root of $number is $result")
}
```

Output:

Square root of 25 is 5.0

Here, **sqrt()** is a library function which returns square root of a number (Double value).

print() library function which prints a message to standard output stream.

User defined Function

User defined function is a function which is created by user. User defined function takes the parameter(s), perform an action and return the result of that action as a value.

Kotlin functions are declared using the fun keyword. For example:

```
fun functionName(){
// body of function
}
```

We have to call the function to run codes inside the body of the function.

```
functionName()
```

Kotlin simple function example

```
fun main(args: Array<String>){
   sum()
   print("code after sum")
}
fun sum(){
   var num1 =5
   var num2 = 6
   println("sum = "+(num1+num2))
}
```

Output:

```
sum = 11
code after sum
```

Kotlin Parameterize Function and Return Value

Functions are also takes parameter as arguments and return value. Kotlin functions are defined using Pascal notation, i.e. *name:type* (name of parameter and its type). Parameters in function are separated using commas.

If a function does not returns any value than its return type is *Unit*. It is optional to specify the return type of function definition which does not returns any value.

```
fun functionName(number1: Int, number2: Int){
.. .. ..
}
.. .. ..
functionName(value1, value2)
.. .. ..
```

Kotlin parameterize function example

```
fun main(args: Array<String>){
  val result = sum(5, 6)
   print(result)
}
fun sum(number1: Int, number2:Int): Int{
   val add = number1+number2
   return add
}
```

Output:

```
11
```

Kotlin Recursion Function

Recursion function is a function which calls itself continuously. This technique is called recursion.

Syntax

```
fun functionName(){
.. .. ..
functionName() //calling same function
}
```

Kotlin recursion function example 1: Finite times

Let's see an example of recursion function printing count.

```
var count = 0
fun rec(){
   count++;
```

```
   if(count<=5){
      println("hello "+count);
      rec();
   }
}
fun main(args: Array<String>) {
   rec();
}
```

Output:

```
hello 1
hello 2
hello 3
hello 4
hello 5
```

Kotlin recursion function example 2: Factorial Number

Let's see an example of recursion function calculating factorial of number.

```
fun main(args: Array<String>) {
   val number = 5
   val result: Long
   result = factorial(number)
   println("Factorial of $number = $result")
}
fun factorial(n: Int): Long {
   return if(n == 1){
        n.toLong()
   }
   else{
      n*factorial(n-1)
   }
}
```

Output:

```
Factorial of 5 = 120
```

Working process of above factorial example

```
factorial(5)
  factorial(4)
    factorial(3)
      factorial(2)
        factorial(1)
          return 1
        return 2*1 = 2
      return 3*2 = 6
    return 4*6 = 24
  return 5*24 = 120
```

Kotlin Tail Recursion

Before we will discuss about the tail recursion, let's try to make an example which calculate sum of nth (100000 larger number) using general (normal) recursion.

General Recursion

Let's see an example of calculating sum of nth (100000 larger number) using general (normal) recursion.

```
fun main(args: Array<String>) {
```

```
   var result = recursiveSum(100000)
   println(result)
}
fun recursiveSum(n: Long) : Long {
   return if (n <= 1) {
      n
   } else {
      n + recursiveSum(n - 1)
   }
}
```

Output:

Exception in thread "main" java.lang.StackOverflowError

The above example throws an exception of *"java.lang.StackOverflowError"*. This is because the compiler is unable to call large number of recursive function call.

Tail Recursion

Tail recursion is a recursion which performs the calculation first, then makes the recursive call. The result of current step is passed into the next recursive call.

Tail recursion follows one rule for implementation. This rule is as follow:

The recursive call must be the last call of the method. To declare a recursion as tail recursion we need to use **tailrec** modifier before the recursive function.

Kotlin Tail Recursion Example 1: calculating sun of nth(100000) number

Let's see an example of calculating sum of nth (100000 larger number) using tail recursion.

```
fun main(args: Array<String>) {
   var number = 100000.toLong()
   var result = recursiveSum(number)
   println("sun of upto $number number = $result")
}
tailrec fun recursiveSum(n: Long, semiresult: Long = 0) : Long {
   return if (n <= 0) {
      semiresult
   } else {
      recursiveSum( (n - 1), n+semiresult)
   }
}
```

Output:

sun of upto 100000 number = 5000050000

Kotlin Tail Recursion Example 2: calculating factorial of number

Let's see an example of calculating factorial of number using tail recursion.

```
fun main(args: Array<String>) {
   val number = 4
   val result: Long
   result = factorial(number)
   println("Factorial of $number = $result")
}

tailrec fun factorial(n: Int, run: Int = 1): Long {
   return if (n == 1){
      run.toLong()
   } else {
```

```
    factorial(n-1, run*n)
  }
}
```

Output:

Factorial of 4 = 24

Kotlin Default and Named Argument

Kotlin Default Argument

Kotlin provides a facility to assign *default argument* (parameter) in a function definition.

If a function is called without passing any argument than default argument are used as parameter of function definition. And when a function is called using argument, than the passing argument is used as parameter in function definition.

Default argument example 1: passing no argument in function call

```
fun main(args: Array<String>) {
  run()
}
fun run(num:Int= 5, latter: Char ='x'){
  print("parameter in function definition $num and $latter")
}
```

Output:

parameter in function definition 5 and x

In the above program, run() function calls with no argument, the default parameter are used in function definition.

Default argument example 2: passing some argument in function call

```
fun main(args: Array<String>) {
  run(3)
}
fun run(num:Int= 5, latter: Char ='x'){
  print("parameter in function definition $num and $latter")
}
```

Output:

parameter in function definition 3 and x

In the above program, run() function calls with one (first) argument, the first parameter of the function definition is uses the value passed to the function. And the second parameter is uses as a default argument.

Default argument example 3: passing all argument in function call

```
fun main(args: Array<String>) {
  run(3,'a')
}
fun run(num:Int= 5, latter: Char ='x'){
  print("parameter in function definition $num and $latter")
}
```

Output:

parameter in function definition 3 and a

As all the arguments are passed in run() function call, the parameters of function definition uses the argument passed in function call.

Kotlin Named Argument

Before we will discuss about the named parameter, let's do some modify in the above program.

For example:

```
fun main(args: Array<String>) {
  run('a')
}
```

```
fun run(num:Int= 5, latter: Char ='x'){
   print("parameter in function definition $num and $latter")
}
```

Output:

Error: Kotlin: The character literal does not conform to the expected type Int

Here, we are try to pass parameter 'a' from function call to function definition in the second argument. But compiler assumes that the parameter 'a' (Char type) passed for first argument (Int type) this causes error in program.

Named Argument

To solve the above problem a named argument is used.

A named argument is an argument in which we define the name of argument in the function call. The name defined to argument of function call checks the name in the function definition and assign to it.

Kotlin Named Argument Example

```
fun main(args: Array<String>) {
   run(latter='a')
}
fun run(num:Int= 5, latter: Char ='x'){
   print("parameter in function definition $num and $latter")
}
```

Output:

parameter in function definition 5 and a

Kotlin Lambda Function

Lambda is a function which has no name. Lambda is defined with a curly braces **{}** which takes *variable as a parameter* (if any) and body of function. The *body of function* is written after variable (if any) followed by **->** operator.

Syntax of lambda

```
{ variable -> body_of_function}
```

Before we talk about lambda, let's see a simple example of addition of two numbers using normal function.

Normal function: addition of two numbers

In this example, we create a function *addNumber()* passing two arguments *(a,b)* calling from the main function.

```
fun main(args: Array<String>){
   addNumber(5,10)
}
fun addNumber(a: Int, b: Int){
   val add = a + b
   println(add)
}
```

Output:

15

Lambda function: addition of two numbers

The above program will be rewritten using lambda function as follow:

```
fun main(args: Array<String>){
   val myLambda: (Int) -> Unit= {s: Int -> println(s) } //lambda function
   addNumber(5,10,myLambda)
}
fun addNumber(a: Int, b: Int, mylambda: (Int) -> Unit ){   //high level function lambda as parameter
   val add = a + b
   mylambda(add) // println(add)
}
```

Output:

15

In the above program we create a lambda expression *{s: Int -> println(s) }* with its return type Unit. The lambda function is padded as an parameter in high level function *addNumber(5,10,myLambda).* The variable *mylambda* in function definition is actually a lambda function. The functionality (body) of *mylambda* is already given in lambda function.

Higher order function

High order function (Higher level function) is a function which accepts function as a parameter or returns a function or can do both. Means, instead of passing Int, String, or other types as a parameter in a function we can pass a function as a parameter in other function.

Let's see the following example:

```
fun myFun(org: String,portal: String, fn: (String,String) -> String): Unit {
   val result = fn(org,portal)
   println(result)
}
```

In this above example, we defined a function myFun() with three parameters. The first and second parameter take String and the third parameter as a type of function from String to String. The parameter String to String type means function takes string as an input and returns output as string types.

To call this above function, we can pass function literal or lambda. For example:

```
fun myFun(org: String,portal: String, fn: (String,String) -> String): Unit {
   val result = fn(org,portal)
   println(result)
}

fun main(args: Array<String>){
   val fn:(String,String)->String={org,portal->"$org develop $portal"}
   myFun("sssit.org","jjkundalia.com",fn)
}
```

Output:

```
sssit.org develop jjkundalia.com
```

The above higher order function can also be called in another ways as below mention code in main() function:

```
myFun("sssit.org","jjkundalia.com",{org,portal->"$org develop $portal"})
```

Inline Function

An **inline function** is declare with a keyword inline. The use of inline function enhances the performance of higher order function. The inline function tells the compiler to copy parameters and functions to the call site.

The *virtual* function or *local* function cannot be declared as **inline**. Following are some expressions and declarations which are not supported anywhere inside the inline functions:

- Declaration of local classes
- Declaration of inner nested classes
- Function expressions
- Declarations of local function
- Default value for optional parameters

Let's see the basic example of inline function:

```
fun main(args: Array<String>) {
inlineFunction({ println("calling inline functions")})
}
inline fun inlineFunction(myFun: () -> Unit ) {
myFun()
```

```
   print("code inside inline function")
}
```

Output:

```
calling inline functions
code inside inline function
```

Non local control flow

From inline function, we can return from lambda expression itself. This will also lead to exit from the function in which inline function was called. The function literal is allowed to have non local return statements in such case.

```
fun main(args: Array<String>) {
inlineFunction({ println("calling inline functions")
      return},{ println("next parameter in inline functions")})
}
 inline fun inlineFunction(myFun: () -> Unit, nxtFun: () -> Unit) {
myFun()
nxtFun()
   print("code inside inline function")
}
```

Output:

```
calling inline functions
```

crossinline annotation

To prevent return from lambda expression and inline function itself, we can mark the lambda expression as **crossinline**. This will throw a compiler error if it found a return statement inside that lambda expression.

```
fun main(args: Array<String>) {
inlineFunction({ println("calling inline functions")
      return // compile time error
},{ println("next parameter in inline functions")})
}
 inline fun inlineFunction(crossline myFun: () -> Unit, nxtFun: () -> Unit) {
myFun()
nxtFun()
   print("code inside inline function")
}
```

noinline modifier

In inline function, when we want some of lambdas passed in inline function to be an inlined, mark other function parameter with noinline modifier. This is used to set expressions not to be inlined in the call.

```
fun main(args: Array<String>) {
inlineFunctionExample({  println("calling inline functions")},
        {  println("next parameter in inline functions")} )

println("this is main function closing")
}
 inline fun inlineFunctionExample(myFun: () -> Unit, noinline nxtFun: () -> Unit  ) {
```

```
myFun()
    nxtFun()
    println("code inside inline function")
    }
```

Output:

```
calling inline functions
next parameter in inline functions
code inside inline function
this is main function closing
```

If an inline function does not contain any noinline function parameter and no reified type parameters then compiler will generate a warning.

Kotlin Class and Object

Kotlin supports both object oriented programming (OOP) as well as functional programming. Object oriented programming is based on real time *objects* and *classes*. Kotlin also support pillars of OOP language such as encapsulation, inheritance and polymorphism.

Kotlin Class

Kotlin **class** is similar to Java class, a class is a blueprint for the objects which have common properties. Kotlin classes are declared using keyword **class**. Kotlin class has a class header which specifies its type parameters, constructor etc. and the class body which is surrounded by curly braces.

Syntax of Kotlin class declaration

```
class className{   // class header
    // property
    // member function
}
```

In above example, class className is an empty constructor. It is generated by compiler automatically but if we want to provide a constructor, we need to write a constructor keyword followed by class name as:

```
class className constructor(){   // class header
    // property
    // member function
}
```

Example of Kotlin class

```
class account {
var acc_no: Int = 0
var name: String? = null
var amount: Float = 0f

  fun deposit() {
     //deposite code
  }

  fun withdraw() {
    // withdraw code
  }
   fun checkBalance() {
     //balance check code
```

```
    }
  }
```

The account class has three properties acc_no, name, amount and three member functions deposit(), withdraw(),checkBalance().

In Kotlin, property must be initialize or declare as abstract. In above class, properties acc_no initialize as 0, name as null and amount as 0f.

Kotlin Object

Object is real time entity or may be a logical entity which has state and behavior. It has the characteristics:

state: it represents value of an object.

behavior: it represent the functionality of an object.

Object is used to access the properties and member function of a class. Kotlin allows to create multiple object of a class.

Create an object

Kotlin object is created in two steps, the first is to create reference and then create an object.

```
var obj1 = className()
```

Creating multiple object

```
var obj1 = className()
var obj2 = className()
```

Here obj1 and obj2 are reference and className() is an object.

Access class property and member function

Properties and member function of class are accessed by . operator using object. For example:

```
obj.deopsit()
obj.name = Ajay
```

Let's create an example, which access the class property and member function using . operator.

```
class Account {
   var acc_no: Int = 0
   var name: String =  ""
   var amount: Float = 0.toFloat()
   fun insert(ac: Int,n: String, am: Float ) {
      acc_no=ac
      name=n
      amount=am
      println("Account no: ${acc_no} holder :${name} amount :${amount}")
   }
    fun deposit() {
      //deposite code
   }
    fun withdraw() {
     // withdraw code
   }
    fun checkBalance() {
      //balance check code
    }
 }
fun main(args: Array<String>){
```

```
    Account()
    var acc= Account()
    acc.insert(832345,"Ankit",1000f) //accessing member function
    println("${acc.name}") //accessing class property
}
```

Output:

```
Account no: 832345 holder :Ankit amount :1000.0
Ankit
```

Kotlin Inheritance

Inheritance is an important feature of object oriented programming language. Inheritance allows to inherit the feature of existing class (or base or parent class) to new class (or derived class or child class).

The main class is called super class (or parent class) and the class which inherits the superclass is called subclass (or child class). The subclass contains features of superclass as well as its own.

The concept of inheritance is allowed when two or more classes have same properties. It allows code reusability. A derived class has only one base class but may have multiple interfaces whereas a base class may have one or more derived classes.

In Kotlin, the derived class inherits a base class using: operator in the class header (after the derive class name or constructor)

```
open class Base(p: Int){
}
class Derived(p: Int) : Base(p){
 }
```

Suppose that,we have two different classes "Programmer" and "Salesman" having the common properties 'name','age', and 'salary' as well as their own separate functionalitiesdoProgram() and fieldWork(). The feature of inheritance allows that we can inherit (Employee) containing the common features.

```
open class Employee(name: String, age: Int, salary: Float) {
    // code of employee
}
 class Programmer(name: String, age: Int, salary: Float): Employee(name,age,salary) {
    // code of programmer
}
class Salesman(name: String, age: Int, salary: Float): Employee(name,age,salary) {
    // code of salesman
}
```

All Kotlin classes have a common superclass "Any". It is a default superclass for a class with no supertypes explicitly specified.

For example, a class Example is implicitly inherited from Any.

```
class Example
```

Kotlin open keyword

As Kotlin classes are **final** by default, they cannot be inherited simply. We use the **open** keyword before the class to inherit a class and make it to non-final,

For example:

```
open class Example{
// I can now be extended!
}
```

Kotlin Inheriting fields from a class

When we inherit a class to derive class, all the fields and functionalities are inherited. We can use these fields and functionalities in derived class.

For example:

```
open class Base{
val x = 10
}
class Derived: Base() {
   fun foo() {
println("x is equal to " + x)
   }
}
fun main(args: Array<String>) {
val derived = Derived()
   derived.foo()
}
```

Output:

```
x is equal to 10
```

Kotlin Inheriting methods from a class

```
open class Bird {
   fun fly() {
println("flying...")
   }
}
class Duck: Bird() {
   fun swim() {
println("swimming...")
   }
}
fun main(args: Array<String>) {
val duck = Duck()
   duck.fly()
duck.swim()
}
```

Output:

```
flying...
swimming...
```

Kotlin Inheritance Example

Here, we declare a class Employee is superclass and Programmer and Salesman are their subclasses. The subclasses inherit properties name, age and salary as well as subclasses containtheir own functionalitieslike doProgram() and fieldWork().

```
open class Employee(name: String, age: Int, salary: Float) {
init {
println("Name is $name.")
println("Age is $age")
println("Salary is $salary")
   }
```

```
}
class Programmer(name: String, age: Int, salary: Float):Employee(name,age,salary){
   fun doProgram() {
println("programming is my passion.")
   }
}
class Salesman(name: String, age: Int, salary: Float):Employee(name,age,salary){
   fun fieldWork() {
println("travelling is my hobby.")
   }
}
fun main(args: Array<String>){
val obj1 = Programmer("Ashu", 25, 40000f)
   obj1.doProgram()
val obj2 = Salesman("Ajay", 24, 30000f)
   obj2.fieldWork()
}
```

Output:

```
Name is Ashu.
Age is 25
Salary is 40000.0
programming is my passion.
Name is Ajay.
Age is 24
Salary is 30000.0
travelling is my hobby.
```

Kotlin Inheritance and primary constructor

If the base and derived class both having primary constructor in that case the parameters are initialized in the primary constructor of base class. In above example of inheritance, all classes contain three parameters "name", "age" and "salary" and all these parameters are initialized in primary constructor of base class.

When a base and derived class both contains different numbers of parameters in their primary constructor then base class parameters are initialized form derived class object.

For example:

```
open class Employee(name: String,salary: Float) {
init {
println("Name is $name.")
println("Salary is $salary")
   }
}
class Programmer(name: String, dept: String, salary: Float):Employee(name,salary){
init {
println("Name $name of department $dept with salary $salary.")
   }
   fun doProgram() {
println("Programming is my passion.")
    }
```

```
}
class Salesman(name: String, dept: String, salary: Float):Employee(name,salary){
init {
println("Name $name of department $dept with salary $salary.")
   }
   fun fieldWork() {
println("Travelling is my hobby.")
    }
}
fun main(args: Array<String>){
val obj1 = Programmer("Ashu", "Development", 40000f)
   obj1.doProgram()
println()
val obj2 = Salesman("Ajay", "Marketing", 30000f)
   obj2.fieldWork()
}
```

Output:

```
Name is Ashu.
Salary is 40000.0
Name Ashu of department Development with salary 40000.0.
Programming is my passion.

Name is Ajay.
Salary is 30000.0
Name Ajay of department Marketing with salary 30000.0.
Travelling is my hobby.
```

When an object of derived class is created, it calls its superclass first and executes init block of base class followed by its own.

Kotlin Inheritance and secondary constructor

If derived class does not contain any primary constructor then it is required to call the base class secondary constructor from derived class using **super** keyword.

For example,

```
open class Patent {
  constructor(name: String, id: Int) {
println("execute super constructor $name: $id")
   }
}
 class Child: Patent {
   constructor(name: String, id: Int, dept: String): super(name, id) {
      print("execute child class constructor with property $name, $id, $dept")
   }
}
fun main(args: Array<String>) {
val child = Child("Ashu",101, "Developer")
}
```

Output:

```
execute super constructor Ashu: 101
execute child class constructor with property Ashu, 101, Developer
```

In above example, when object of Child class is created, it calls its constructor and initializes its parameters with values "Ashu", "101" and "Developer". At the same time Child class constructor calling its supper class constructor using super keyword with values of name and id. Due to the presence of super keyword thebody of superclass constructor executes first and returns to Child class constructor.

Kotlin Method Overriding

Method overriding means providing the specific implementation of method of super (parent) class into its subclass (child) class.

In other words, when subclass redefines or modifies the method of its superclass into subclass, it is known as method overriding. Method overriding is only possible in inheritance.

KotlinRules of method overriding

Parent class and its method or property which is to be overridden must be open (non-final).

Method name of base class and derived class must have same.

Method must have same parameter as in base class.

Example of inheritance without overriding

```
open class Bird {
   open fun fly() {
println("Bird is flying...")
   }
}
class Parrot: Bird() {
}
class Duck: Bird() {
}
fun main(args: Array<String>) {
val p = Parrot()
   p.fly()
val d = Duck()
   d.fly()
}
```

Output:

```
Bird is flying...
Bird is flying...
```

In above example, a program without overriding the method of base class we found that both derived classes Parrot and Duck perform the same common operation. To overcome with this problem we use the concept of method overriding.

Example of Kotlin method overriding

In this example, the method *fly()* of parent class Bird is overridden in its subclass *Parrot* and *Duck*. To override the method of parent class, the parent class and its method which is going to override must be declare as *open*. At the same time method which is overridden in child class must be prefaced with keyword *override*.

```
open class Bird {
   open fun fly() {
println("Bird is flying...")
   }
}
class Parrot: Bird() {
```

```
   override fun fly() {
println("Parrot is flying...")
   }
}
class Duck: Bird() {
   override fun fly() {
println("Duck is flying...")
   }
}
fun main(args: Array<String>) {
val p = Parrot()
   p.fly()
val d = Duck()
   d.fly()
}
```

Output:

```
Parrot is flying...
Duck is flying...
```

Example of Kotlin property overriding

Property of superclass can also be overridden in its subclass as similar to method. A color property of *Bird* class is overridden in its subclass *Parrot* and *Duck* and modified.

```
open class Bird {
   open var color = "Black"
   open fun fly() {
println("Bird is flying...")
   }
}
class Parrot: Bird() {
   override var color = "Green"
   override fun fly() {
println("Parrot is flying...")
   }
}
class Duck: Bird() {
   override var color = "White"
   override fun fly() {
println("Duck is flying...")
   }
}
fun main(args: Array<String>) {
val p = Parrot()
   p.fly()
println(p.color)
val d = Duck()
   d.fly()
println(d.color)
```

```
}
```

Output:

```
Parrot is flying...
Green
Duck is flying...
White
```

We can override the val property with var property in inheritance but vice-versa is not true.

Kotlin superclass implementation

Derived class can also call its superclass methods and property using **super** keyword.

For example:

```
open class Bird {
   open var color = "Black"
   open fun fly() {
println("Bird is flying...")
   }
}
class Parrot: Bird() {
   override var color = "Green"
   override fun fly() {
      super.fly()
println("Parrot is flying...")
    }
}

fun main(args: Array<String>) {
val p = Parrot()
   p.fly()
println(p.color)
}
```

Output:

```
Bird is flying...
Parrot is flying...
Green
```

Kotlin multiple class implementation

In Kotlin, derived class uses a supertype name in angle brackets, e.gsuper<Base> when it implements same function name provided in multiple classes.

For example, a derived class Parrotextends its superclass Bird and implement *Duck* interface containing same function *fly()*. To call particular method of each class and interface we must be mention supertype name in angle brackets as super<Bird>.fly() and super<Duck>.fly() for each method.

```
open class Bird {
   open var color = "Black"
   open fun fly() {
println("Bird is flying...")
   }
}
interface Duck {
    fun fly() {
```

```
println("Duck is flying...")
   }
}
class Parrot: Bird(),Duck {
   override var color = "Green"
   override fun fly() {
      super<Bird>.fly()
      super<Duck>.fly()
println("Parrot is flying...")
  }
}
fun main(args: Array<String>) {
val p = Parrot()
   p.fly()
println(p.color)
}
```

Output:

```
Bird is flying...
Duck is flying...
Parrot is flying...
```

Kotlin Abstract class

A class which is declared with **abstract** keyword is known as *abstract class*. An abstract class cannot be instantiated. Means, we cannot create object of abstract class. The method and properties of abstract class are non-abstract unless they are explicitly declared as abstract.

Declaration of abstract class

```
abstract class A {
var x = 0
   abstract fun doSomething()
}
```

Abstract classes are partially defined classes, methods and properties which are no implementation but must be implemented into derived class. If the derived class does not implement the properties of base class then is also meant to be an abstract class.

Abstract class or abstract function does not need to annotate with open keyword as they are *open by default*. Abstract member function does not contain its body. The member function cannot be declared as abstract if it contains in body in abstract class.

Example of abstract class that has abstract method

In this example, there is an abstract class **Car** that contains an abstract function run(). The implementation of run() function is provided by its subclass Honda.

```
abstract class Car{
   abstract fun run()
}
class Honda: Car(){
  override fun run(){
println("Honda is running safely..")
  }
}
```

```
fun main(args: Array<String>){
val obj = Honda()
obj.run();
}
```

Output:

```
Honda is running safely..
```

A non-abstract open member function can be over ridden in an abstract class.

```
open class Car {
   open fun run() {
println("Car is running..")
   }
}
abstract class Honda : Car() {
   override abstract fun run()
}
class City: Honda(){
   override fun run() {
   //  TODO("not implemented") //To change body of created functions use File | Settings | File Templates.

println("Honda City is running..")
   }
}
fun main(args: Array<String>){
val car = Car()
car.run()
val city = City()
city.run()
}
```

Output:

```
Car is running..
Honda City is running..
```

In above example, An abstract class Honda extends the class *Car* and its function *run()*. Honda class override the *run()* function of Car class. The *Honda* class did not give the implementation of *run()* function as it is also declared as abstract. The implementation of abstract function *run()* of Honda class is provided by City class.

Example of real scenario of abstract class

In this example, an abstract class *Bank* that contains an abstract function *simpleInterest()* accepts three parameters p,r,and t. The class *SBI* and *PNB* provides the implementation of simpleInterest() function and returns the result.

```
abstract class Bank {
   abstract fun simpleInterest(p: Int, r: Double, t: Int) :Double
}
class SBI : Bank() {
   override fun simpleInterest(p: Int, r: Double, t: Int): Double{
      return (p*r*t)/100
   }
```

```
}
class PNB : Bank() {
   override fun simpleInterest(p: Int, r: Double, t: Int): Double{
      return (p*r*t)/100
   }
}
fun main(args: Array<String>) {
var sbi: Bank = SBI()
val sbiint = sbi.simpleInterest(1000,5.0,3)
println("SBI interest is $sbiint")
var pnb: Bank = PNB()
val pnbint = pnb.simpleInterest(1000,4.5,3)
println("PNB interest is $pnbint")
}
```

Output:

```
SBI interest is 150.0
PNB interest is 135.0
```

Kotlin Interface

An interface is a blueprint of class.Kotlin interface is similar to Java 8. It contains abstract method declarations as well as implementation of method.

Defining Interface

An interface is defined using the keyword **interface**. For example:

```
interface MyInterface {
val id: Int // abstract property
   fun absMethod()// abstract method
   fun doSomthing() {
    // optional body
   }
}
```

The methods which are only declared without their method body are **abstract** by default.

Why use Kotlin interface?

Following are the reasons to use interface:

Using interface supports functionality of multiple inheritance.

It can be used achieve to loose coupling.

It is used to achieve abstraction.

Subclass extends only one super class but implements multiple interfaces. Extension of parent class or interface implementation are done using (:) operator in their subclass.

Implementing Interfaces

In this example, we are implementing the interface *MyInterface* in *InterfaceImp* class. InterfaceImp class provides the implementation of property *id* and *abstract* method *absMethod()* declared in MyInterface interface.

```
interface MyInterface  {
var id: Int          // abstract property
   fun absMethod():String    // abstract method
   fun doSomthing() {
println("MyInterface doing some work")
```

```
    }
}
class InterfaceImp : MyInterface {
    override var id: Int = 101
    override fun absMethod(): String{
return "Implementing abstract method.."
    }
}
fun main(args: Array<String>) {
val obj = InterfaceImp()
println("Calling overriding id value = ${obj.id}")
obj.doSomthing()
println(obj.absMethod())
}
```

Output:

```
Calling overriding id value = 101
MyInterface doing some work
Implementing abstract method..
```

Implementing multiple interface

We can implement multiple abstract methods of different interfaces in same class. All the abstract methods must be implemented in subclass. The other non-abstract methods of interface can be called from derived class.

For example, creating two interface *MyInterface1* and *MyInterface2* with abstract methods *doSomthing()* and *absMethod()* respectively. These abstract methods are overridden in derive class *MyClass*.

```
interface MyInterface1 {
    fun doSomthing()
}
interface MyInterface2 {
    fun absMethod()
}
class MyClass : MyInterface1, MyInterface2 {
    override fun doSomthing() {
println("overriding doSomthing() of MyInterface1")
    }
  override fun absMethod() {
println("overriding absMethod() of MyInterface2")
    }
}
fun main(args: Array<String>) {
val myClass = MyClass()
myClass.doSomthing()
myClass.absMethod()
}
```

Output:

```
overriding doSomthing() of MyInterface1
```

```
overriding absMethod() of MyInterface2
```

Resolving different Interfaces having same method overriding conflicts

Let's see an example in which interface MyInterface1 and interface MyInterface2 both contains same non-abstract method. A class MyClass provides the implementation of these interfaces. Calling the method of interface using object of MyClass generates an error.

```
interface MyInterface1 {
   fun doSomthing(){
println("overriding doSomthing() of MyInterface1")
   }
}
interface MyInterface2 {
   fun doSomthing(){
println("overriding doSomthing() of MyInterface2")
   }
}
class MyClass : MyInterface1, MyInterface2 {

}
fun main(args: Array<String>) {
val myClass = MyClass()
myClass.doSomthing()
}
```

Output:

```
Kotlin: Class 'MyClass' must override public open fun doSomthing(): Unit defined in
MyInterface1 because it
inherits multiple interface methods of it
```

To solve the above problem we need to specify particular method of interface which we are calling. Let's see an example below.

In below example, two interfaces *MyInterface1* and *MyInterface2* contain two abstract *methods adsMethod()* and *absMethod(name: String)* and non-abstract method *doSomthing()* in both respectively. A class *MyClass implements* both interface and override abstract method *absMethod()* and *absMethod(name: String)* . To override the non-abstract method *doSomthing()* we need to specify interface name with method using **super keyword** as *super<interface_name>.methodName()*.

```
interface MyInterface1 {
   fun doSomthing() {
println("MyInterface 1 doing some work")
   }
      fun absMethod()
}
interface MyInterface2 {
   fun doSomthing(){
println("MyInterface 2 doing some work")
   }
   fun absMethod(name: String)
}
class MyClass : MyInterface1, MyInterface2 {
```

```
   override fun doSomthing() {
      super<MyInterface2>.doSomthing()
   }
   override fun absMethod() {
println("Implements absMethod() of MyInterface1")
   }
   override fun absMethod(n: String) {
println("Implements absMethod(name) of MyInterface2 name is  $n")
   }
}
fun main(args: Array<String>) {
val myClass = MyClass()
myClass.doSomthing()
myClass.absMethod()
myClass.absMethod("Ashu")
}
```

Output:

```
MyInterface 2 doing some work
Implements absMethod() of MyInterface1
Implements absMethod(name) of MyInterface2 name is  Ashu
```

```
interface MyInterface1 {
   fun doSomthing() {
println("MyInterface 1 doing some work")
   }
   fun absMethod()
}
 interface MyInterface2 {
   fun doSomthing() {
println("MyInterface 2 doing some work")
   }
  fun absMethod() {
println("MyInterface 2 absMethod")
   }
 }
 class C : MyInterface1 {
   override fun absMethod() {
println("MyInterface1 absMethod implementation")
   }
}
 class D : MyInterface1, MyInterface2 {
   override fun doSomthing() {
      super<MyInterface1>.doSomthing()
      super<MyInterface2>.doSomthing()
   }
     override fun absMethod() {
      super<MyInterface2>.absMethod()
```

```
    }
}
 fun main(args: Array<String>) {
val d = D()
val c = C()
d.doSomthing()
d.absMethod()
c.doSomthing()
c.absMethod()
}
```

Output:

```
MyInterface 1 doing some work
MyInterface 2 doing some work
MyInterface 2 absMethod
MyInterface 1 doing some work
MyInterface1 absMethod implementation
```

Kotlin Visibility Modifier

Visibility modifiers are the keywords which are used to restrict the use of class, interface, methods, and property of Kotlin in the application. These modifiers are used at multiple places such as class header or method body.

In Kotlin, visibility modifiers are categorized into four different types:

public
protected
internal
private

public modifier

A **public** modifier is accessible from everywhere in the project. It is a default modifier in Kotlin. If any class, interface etc. are not specified with any access modifier then that class, interface etc. are used in public scope.

```
public class Example{
}
class Demo{
}
public fun hello()
fun demo()
public val x = 5
val y = 10
```

All public declaration can be placed at top of the file. If a member of class is not specified then it is by default public.

protected modifier

A **protected** modifier with class or interface allows visibility to its class or subclass only. A protected declaration (when overridden) in its subclass is also protected modifier unless it is explicitly changed.

```
open class Base{
   protected val i = 0
}
class Derived : Base(){
```

```
 fun getValue() : Int
   {
      return i
   }
}
```

In Kotlin, protected modifier cannot be declared at top level.

Overriding of protected types

```
open class Base{
  open protected val i = 5
}
class Another : Base(){
   fun getValue() : Int
   {
      return i
   }
   override val i =10
}
```

internal modifier

The **internal** modifiers are newly added in Kotlin, it is not available in Java. Declaring anything makes that field marked as internal field. The internal modifier makes the field visible only inside the module in which it is implemented.

```
internal class Example{
   internal val x = 5
   internal fun getValue(){
   }
}
internal val y = 10
```

In above, all the fields are declared as internal which are accessible only inside the module in which they are implemented.

private modifier

A **private** modifier allows the declaration to be accessible only within the block in which properties, fields, etc. are declare. The private modifier declaration does not allow to access the outside the scope. A private package can be accessible within that specific file.

```
private class Example {
   private val x = 1
    private valdoSomething() {
   }
}
```

In above class Example,val x and function doSomthing() are declared as private. The class "Example" is accessible from the same source file, "val x" and "fun doSomthing()" are accessible within Example class.

Example of Visibility Modifier

```
open class Base() {
var a = 1 // public by default
   private var b = 2 // private to Base class
   protected open val c = 3  // visible to the Base and the Derived class
```

```
    internal val d = 4 // visible inside the same module
    protected fun e() { } // visible to the Base and the Derived class
}
 class Derived: Base() {
    // a, c, d, and e() of the Base class are visible
    // b is not visible
    override val c = 9 // c is protected
}
fun main(args: Array<String>) {
val base = Base()
    // base.a and base.d are visible
    // base.b, base.c and base.e() are not visible
val derived = Derived()
    // derived.c is not visible
}
```

UNIT-2

Introduction to Android and Android Application Design

The Open Handset Alliance

Android is a software package and linux based operating system for mobile devices such as tablet computers and smartphones.

It is developed by Google and later the OHA (Open Handset Alliance). Java language is mainly used to write the android code even though other languages can be used.

The goal of android project is to create a successful real-world product that improves the mobile experience for end users.

There are many code names of android such as Lollipop, Kitkat, Jelly Bean, Ice cream Sandwich, Froyo, Ecliar, Donut etc which is covered in next page.

What is Open Handset Alliance (OHA)

It's a consortium of 84 companies such as google, samsung, AKM, synaptics, KDDI, Garmin, Teleca, Ebay, Intel etc.

It was established on 5th November, 2007, led by Google. It is committed to advance open standards, provide services and deploy handsets using the Android Plateform.

WHAT IS ANDROID?

Android is an operating system based on the Linux kernel, and designed primarily for touch screen mobile devices such as smart phones and tablet computers. Initially developed by Android, Inc., which Google backed financially and later bought in 2005.

Android is a Linux-based operating system for smart phones and tablets. It includes a touch screen user interface, widgets, camera, network data monitoring and all the other features that enable a cell phone to be called a Smartphone. Android is a platform that supports various applications, available through the Android Play Store. The Android platform also allows end users to develop, install and use their own applications on top of the Android Framework. The Android framework is licensed under the Apache License, with Android application developers holding the right to distribute their applications under their customized license.

Features of Android

After learning what is android, let's see the features of android. The important features of android are given below:

1) It is open-source.

2) Anyone can customize the Android Platform.

3) There are a lot of mobile applications that can be chosen by the consumer.

4) It provides many interesting features like weather details, opening screen, live RSS (Really Simple Syndication) feeds etc.

It provides support for messaging services(SMS and MMS), web browser, storage (SQLite), connectivity (GSM, CDMA, Blue Tooth, Wi-Fi etc.), media, handset layout etc.

Categories of Android applications

There are many android applications in the market. The top categories are:

- Entertainment
- Tools
- Communication
- Productivity
- Personalization
- Music and Audio

- Social
- Media and Video
- Travel and Local etc.

History of Android

The history and versions of android are interesting to know. The code names of android ranges from A to J currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat and Lollipop. Let's understand the android history in a sequence.
1) Initially, Andy Rubin founded Android Incorporation in Palo Alto, California, United States in October, 2003.
2) In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.
3) The key employees of Android Incorporation are Andy Rubin, Rich Miner, Chris White and Nick Sears.
4) Originally intended for camera but shifted to smart phones later because of low market for camera only.
5) Android is the nick name of Andy Rubin given by coworkers because of his love to robots.
6) In 2007, Google announces the development of android OS.
7) In 2008, HTC launched the first android mobile.

Android Versions, Codename and API

Let's see the android versions, codenames and API Level provided by Google.

Version	Code name	API Level
1.5	Cupcake	3
1.6	Donut	4
2.1	Eclair	7
2.2	Froyo	8
2.3	Gingerbread	9 and 10
3.1 and 3.3	Honeycomb	12 and 13
4.0	Ice Cream Sandwitch	15
4.1, 4.2 and 4.3	Jelly Bean	16, 17 and 18
4.4	KitKat	19
5.0	Lollipop	21

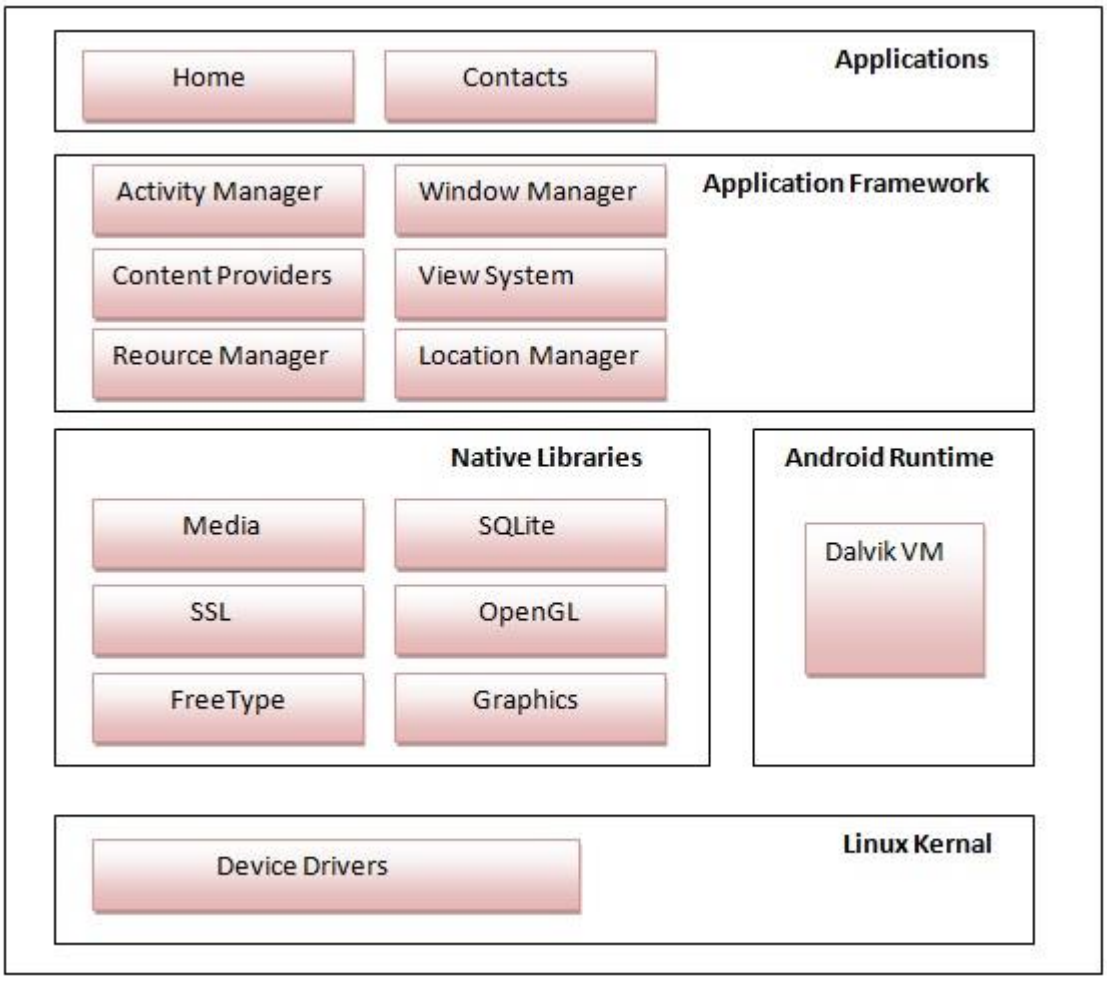
The Android Platform

Android is an operating system and a software platform upon which applications are developed. A core set of applications for everyday tasks, such as web browsing and email, are included on Android handsets. As a product of the OHA's vision for a robust and open source development environment for wireless, Android is an emerging mobile development platform. The platform was designed for the sole purpose of encouraging a free and open market that all mobile applications phone users might want to have and software developers might want to develop.

Android Architecture

Android architecture or **Android software stack** is categorized into five parts:
linux kernel
native libraries (middleware),

Android Runtime
Application Framework
Applications
Let's see the android architecture first.



1) Linux kernel
It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access.

---

2) Native Libraries
On the top of linux kernel, their are Native libraries such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc.
The WebKit library is responsible for browser support, SQLite is for database, FreeType for font support, Media for playing and recording audio and video formats.

---

3) Android Runtime
In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.

4) Android Framework

On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

5) Applications

On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using linux kernel.

Android SDK

The Android SDK provides an extensive set of application programming interfaces (APIs) that is both modern and robust. Android handset core system services are exposed and accessible to all applications. When granted the appropriate permissions, Android applications can share data among one another and access shared resources on the system securely.

Building a sample Android application

Set Up Your Environment

Before you start this class, be sure you have your development environment set up. You need to:

1. Download Android Studio.
2. Download the latest SDK tools and platforms using the SDK Manager.

Installing android studio

Before you set up Android Studio, be sure you have installed JDK 6 or higher (the JRE alone is not sufficient)—JDK 7 is required when developing for Android 5.0 and higher.

To set up Android Studio on Windows:

1. Launch the .exe file you just downloaded.
2. Follow the setup wizard to install Android Studio and any necessary SDK tools.
   On some Windows systems, the launcher script does not find where Java is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.
   Select Start menu > Computer > System Properties > Advanced System Properties. Then open **Advanced tab > Environment Variables** and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\ProgramFiles\Java\jdk1.7.0_21.
   The individual tools and other SDK packages are saved outside the Android Studio application directory. If you need to access the tools directly, use a terminal to navigate to the location where they are installed. For example:\Users\<user>\sdk\

Android Studio is now ready and loaded with the Android developer tools, but there are still a couple packages you should add to make your Android SDK complete

Adding SDK Packages

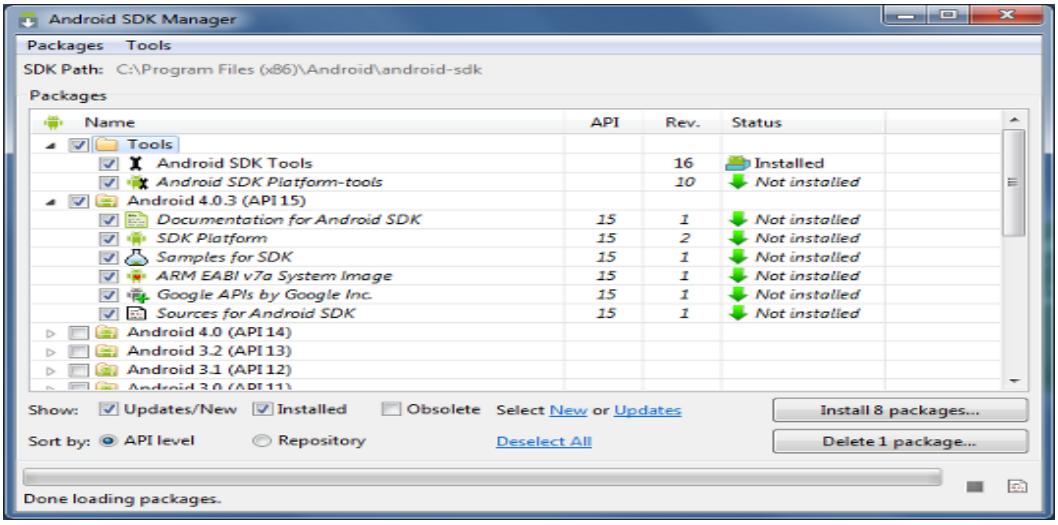
By default, the Android SDK does not include everything you need to start developing. The SDK separates tools, platforms, and other components into packages you can download as needed using the Android SDK Manager. So before you can start, there are a few packages you should add to your Android SDK.

To start adding packages, launch the Android SDK Manager in one of the following ways:

- In Android Studio, click **SDK Manager**  in the toolbar.
- If you're not using Android Studio:
    - Windows: Double-click the SDK Manager.exe file at the root of the Android SDK directory.
    - Mac/Linux: Open a terminal and navigate to the tools/ directory in the Android SDK, then execute android sdk.

When you open the SDK Manager for the first time, several packages are selected by default. Leave these selected, but be sure you have everything you need to get started by following these steps:

1. Get the latest SDK tools



   As a minimum when setting up the Android SDK, you should download the latest tools and Android platform:

    1. Open the Tools directory and select:
        - **Android SDK Tools**
        - **Android SDK Platform-tools**
        - **Android SDK Build-tools** (highest version)
    2. Open the first Android X.X folder (the latest version) and select:
        - **SDK Platform**
        - A system image for the emulator, such as **ARM EABI v7a System Image**

2. Get the support library for additional APIs

   The support library is required for:
    - Android Wear
    - Android TV
    - Google Cast

   It also provides these popular APIs:
    - Navigation drawer
    - Swipe views
    - Backward-compatible action bar

   The Android Support Library provides an extended set of APIs that are compatible with most versions of Android.

   Open the **Extras** directory and select:

- **Android Support Repository**
- **Android Support Library**

3. Get Google Play services for even more APIs
   The Google Play services APIs provide a variety of features and services for your Android apps, such as:
   - User authentication
   - Google Maps
   - Google Cast
   - Games achievements and leader boards
   - And much more

   To develop with Google APIs, you need the Google Play services package:
   Open the **Extras** directory and select:
   - **Google Repository**
   - **Google Play services**

   Note: Google Play services APIs are not available on all Android-powered devices, but are available on all devices with Google Play Store. To use these APIs in the Android emulator, you must also install the the **Google APIs** system image from the latest Android X.X directory in the SDK Manager.
4. Install the packages
   Once you've selected all the desired packages, continue to install:
   0. Click **Install X packages**.
   1. In the next window, double-click each package name on the left to accept the license agreement for each.
   2. Click **Install**.

   The download progress is shown at the bottom of the SDK Manager window. **Do not exit the SDK Manager** or it will cancel the download.
5. Build something!
   With the above packages now in your Android SDK, you're ready to build apps for Android. As new tools and other APIs become available, simply launch the SDK Manager to download the new packages for your SDK.
   Here are a few options for how you should proceed:
   Create a Project with Android Studio
   1.In Android Studio, create a new project:
   If you don't have a project opened, in the **Welcome** screen, click **New Project**.
   If you have a project opened, from the **File** menu, select **New Project**.

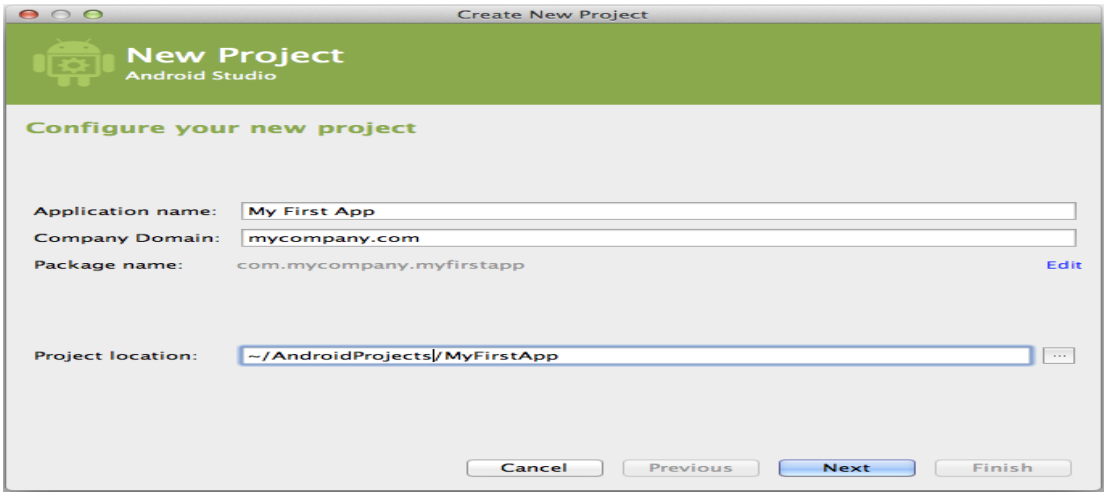


Figure 1. Configuring a new project in Android Studio.

2. Under **Configure your new project**, fill in the fields as shown in figure 1 and click **Next**.

It will probably be easier to follow these lessons if you use the same values as shown.

- **Application Name** is the app name that appears to users. For this project, use "My First App."
- **Company domain** provides a qualifier that will be appended to the package name; Android Studio will remember this qualifier for each new project you create.
- **Package name** is the fully qualified name for the project (following the same rules as those for naming packages in the Java programming language). Your package name must be unique across all packages installed on the Android system. You can **Edit** this value independently from the application name or the company domain.
- **Project location** is the directory on your system that holds the project files.

3. Under **Select the form factors your app will run on**, check the box for **Phone and Tablet**.

4. For **Minimum SDK**, select **API 8: Android 2.2 (Froyo)**.

The Minimum Required SDK is the earliest version of Android that your app supports, indicated using the API level. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the app's core feature set, you can enable the feature only when running on the versions that support it (as discussed in Supporting Different Platform Versions).

5. Leave all of the other options (TV, Wear, and Glass) unchecked and click **Next.**

Activities

6. Under **Add an activity to your project**, select **Blank Activity** and click **Next**.

7. Under **Describe the new activity for your project**, leave the fields as they are and click **Finish**.

Your Android project is now a basic "Hello World" app that contains some default files. Take a moment to review the most important of these:

app/src/main/res/layout/activity_my.xml

This is the XML layout file for the activity you added when you created the project with Android Studio. Following the New Project workflow, Android Studio presents this file with both a text view and a preview of the screen UI. The file includes some default settings and a TextView element that displays the message, "Hello world!"

app/src/main/java/com.mycompany.myfirstapp/MyActivity.java

A tab for this file appears in Android Studio when the New Project workflow finishes. When you select the file you see the class definition for the activity you created. When you build and run the app, the Activity class starts the activity and loads the layout file that says "Hello World!"

app/src/res/AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll revisit this file as you follow these lessons and add more components to your app.

app/build.gradle

Android Studio uses Gradle to compile and build your app. There is a build.gradle file for each module of your project, as well as a build.gradle file for the entire project. Usually, you're only interested in the build.gradle file for the module, in this case the app or application module. This is where your app's build dependencies are set, including the defaultConfig settings:

- compiledSdkVersion is the platform version against which you will compile your app. By default, this is set to the latest version of Android available in your SDK. (It should be Android 4.1 or greater; if you don't have such a version available, you must install one using the SDK Manager.) You can still build your app to support older versions, but setting this to the latest version allows you to enable new features and optimize your app for a great user experience on the latest devices.
- applicationId is the fully qualified package name for your application that you specified during the New Project workflow.
- minSdkVersion is the Minimum SDK version you specified during the New Project workflow. This is the earliest version of the Android SDK that your app supports.
- targetSdkVersion indicates the highest version of Android with which you have tested your application. As new versions of Android become available, you should test your app on the new version and update this value to match the latest API level and thereby take advantage of new platform features. For more information, read Supporting Different Platform Versions.

See Building Your Project with Gradle for more information about Gradle.

Note also the /res subdirectories that contain the resources for your application:

drawable-hdpi/

Directory for drawable objects (such as bitmaps) that are designed for high-density (hdpi) screens. Other drawable directories contain assets designed for other screen densities. Here you'll find the ic_launcher.png that appears when you run the default app.

layout/

Directory for files that define your app's user interface like activity_my.xml, discussed above, which describes a basic layout for the MyActivity class.

values/

Directory for other XML files that contain a collection of resources, such as string and color definitions. The strings.xml file defines the "Hello world!" string that displays when you run the default app.

Running your app

How you run your app depends on two things: whether you have a real device running Android and whether you're using Android Studio. This lesson shows you how to install and run your app on a real device and on the Android emulator, and in both cases with either Android Studio

Run on a Real Device

If you have a device running Android, here's how to install and run your app.
Set up your device

1. Plug in your device to your development machine with a USB cable. If you're developing on Windows, you might need to install the appropriate USB driver for your device. For help installing drivers, see the OEM USB Drivers document.
2. Enable **USB debugging** on your device.
    - On most devices running Android 3.2 or older, you can find the option under **Settings > Applications > Development**.
    - On Android 4.0 and newer, it's in **Settings > Developer options**.
      Note: On Android 4.2 and newer, **Developer options** is hidden by default. To make it available, go to **Settings > About phone** and tap **Build number** seven times. Return to the previous screen to find **Developer options**.

Run the app from Android Studio

1. Select one of your project's files and click **Run**  from the toolbar.
2. In the **Choose Device** window that appears, select the **Choose a running device** radio button, select your device, and click **OK** .

Android Studio installs the app on your connected device and starts it.
Run the app from a command line

1. Change directories to the root of your Android project and execute: ant debug
2. Make sure the Android SDK platform-tools/ directory is included in your PATH environment variable, then execute:
   adb install bin/MyFirstApp-debug.apk
3. On your device, locate *MyFirstApp* and open it.

That's how you build and run your Android app on a device! To start developing, continue to the next lesson.

Run on the Emulator

Whether you're using Android Studio or the command line, to run your app on the emulator you need to first create an Android Virtual Device (AVD). An AVD is a device configuration for the Android emulator that allows you to model a specific device.
Create an AVD

1. Launch the Android Virtual Device Manager:
    - In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager icon  in the toolbar.
    - Or, from the command line, change directories to <sdk>/tools/ and execute:

      android avd
      Note: The AVD Manager that appears when launched from the command line is different from the version in Android Studio, so the following instructions may not apply.

Figure 1. The AVD Manager main screen shows your current virtual devices.

2. On the AVD Manager main screen (figure 1), click **Create Virtual Device**.
3. In the Select Hardware window, select a device configuration, such as Nexus 6, then click **Next**.
4. Select the desired system version for the AVD and click **Next**.
5. Verify the configuration settings, then click **Finish**.

For more information about using AVDs, see Managing AVDs with AVD Manager.
Run the app from Android Studio

1. In **Android Studio**, select your project and click **Run**  from the toolbar.
2. In the **Choose Device** window, click the **Launch emulator** radio button.
3. From the **Android virtual device** pull-down menu, select the emulator you created, and click **OK**.

It can take a few minutes for the emulator to load itself. You may have to unlock the screen. When you do, My First App appears on the emulator screen.

Anatomy (Structure) of an Android Application

Classical computer science classes often define a program in terms of functionality and data, and Android applications are no different. They perform tasks, display information to the screen, and act upon data from a variety of sources.

Developing Android applications for mobile devices with limited resources requires a thorough understanding of the application lifecycle. Android also uses its own terminology for these application building blocks—terms such as Context, Activity, and Intent.

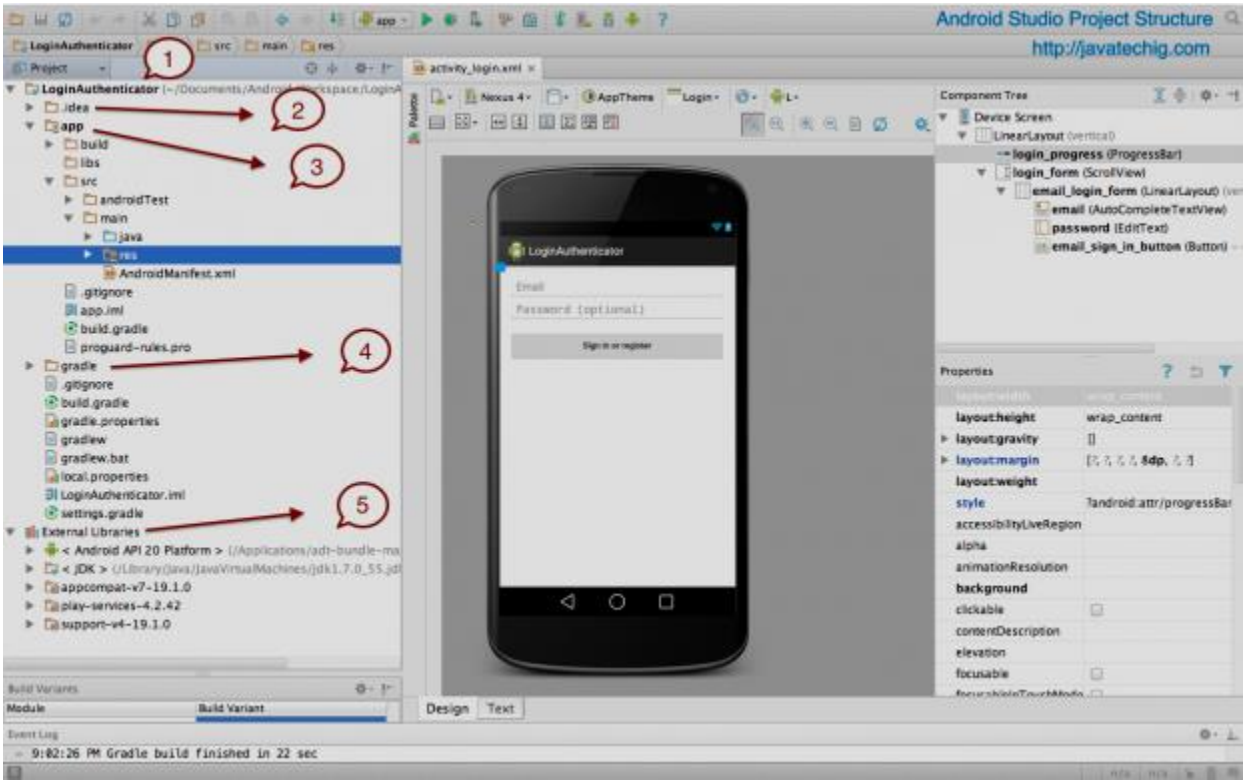
Android Studio Project Structure

1. Main Project

This would be entire project context. Whatever you do in IntelliJ IDEA, you do that in the context of a project. A project is an organizational unit that represents a complete software solution. A project in Android Studio is like a workspace in Eclipse. In android Studio a project, can contain multiple modules. A module in Android Studio is like a project in Eclipse. In the above screenshot “LoginAuthenticator” is the name of my project

This means that, in theory it is possible to build multiple apps within the same project. From my personal experience, creating multiple apps within the same project doesn’t works well. So, I

recommend not to make your hands dirty trying the same thing. Instead, It is a better idea to create single app per single project.



2. .idea

Eclipse uses project.properties file for project specific metadata. Here in android studio, this .idea does the same thing. This means the project specific metadata is stored by Android Studio.

3. Project Module (app)

This is the actual project folder where your application code resides. The application folder has following sub directories

a. **build:** This has all the complete output of the make process i.e. classes.dex, compiled classes and resources, etc.

In the Android Studio GUI, only a few folders are shown. The important part is that your R.java is found here under build/source/r/<build variant>/<package>/R.java

b. **libs :** This is a commonly seen folder in eclipse land too, which optionally can hold the libraries or .jar files.

c. **src:** The src folder can have both application code and android unit test script. You will find two folders named “androidTest” and “main” correspond to src folder. The main folder contains two subfolders java and res. The java folder contains all the java codes and res contains drawables, layouts, etc.

4. gradle

This is where the gradle build system’s jar wrapper i.e. this jar is how AS communicates with gradle installed in Windows (the OS in my case).

5. External Libraries

This is not actually a folder but a place where Referenced Libraries and information on targeted platform SDK is shown.

Android terminologies

Context: The context is the central command center for an Android application. All application-specific functionality can be accessed through the context.
Activity: An Android application is a collection of tasks, each of which is called an Activity. Each Activity within an application has a unique task or purpose.
Intent: The Android operating system uses an asynchronous messaging mechanism to match task requests with the appropriate Activity. Each request is packaged as an Intent. You can think of each such request as a message stating an intent to do something.
Service: Tasks that do not require user interaction can be encapsulated in a service. A service is most useful when the operations are lengthy (offloading time-consuming processing) or need to be done regularly (such as checking a server for new mail).
.apk file: Android application package file.
Each Android application is compiled and packaged in as single file that includes all of the application's code (.dex files), resources, assets and manifest file. The application package file can have any name but must use the .apk extension. For example: myfirstproj.apk. For convenience, an application package file is often referred to as an ".apk".
.dex file : Compiled Android application code and file.
Android programs are compiled into .dex (Dalvik Executable) files, which are in turn zipped into a single .apk file on the device. .dex files can be created by automatically translating compiled applications written in the java programming language.
adb : Android Debug Bridge A command-line debugging application included with the SDK. It provides tools to browse the device, copy tools on the device, and forward ports for debugging. If you are developing in Eclipse using the ADT Plugin, adb is integrated into your development environment.
Content Provider : A content provider is built on the ContentProvider class, which handles content query strings of a specific format to return data in a s specific format.
Dalvik : The Android platform's virtual machine. The Dalvik VM is an interpreter only virtual machine that executes files in the files in the Dalvik Executable (.dex) format, a virtual machine is register-based, and it can run classes compiled by a Java language compiler that have been transformed into its native format using the included "dx" tool. The Vm runs on top of Posix-compliant operating systems, which it relies on for underlying functionality. The Dalvik core class library is intended to provide a familiar development base for those used to programming with Java Standard Edition, but it is geared specifically to the needs of a small mobile device.
DDMS : Dalvik Debug Moniter Service, a GUI debugging application included with the SDK. It provides screen capture, log dump, and processs examination capabilities. If you are developing in Eclipse using the ADT Plugin, DDMS is integrated into your development environment.
Widget : One of a set of fully implemented View subclasses that render form elements and other UI components, such as a text box or popup menu. Because a widget is fully implemented, it handles measuring and drawing itself and responding to screen events. Widgets are all in the android.widget.package.

Application Context, Activities, Services, Intents

Using the Application Context

The application Context is the central location for all top-level application functionality. The Context class can be used to manage application-specific configuration details as well as application-wide operations and data. Use the application Context to access settings
and resources shared across multiple Activity instances.

Retrieving the Application Context

You can retrieve the Context for the current process using the getApplicationContext() method, like this: Context context getApplicationContext();

Accessing Other Application Functionality Using Context

The application Context provides access to a number of other top-level application features.
Here are a few more things you can do with the application Context:

- Launch Activity instances
- Retrieve assets packaged with the application
- Request a system service (for example, location service)
- Manage private application files, directories, and databases
- Inspect and enforce application permissions

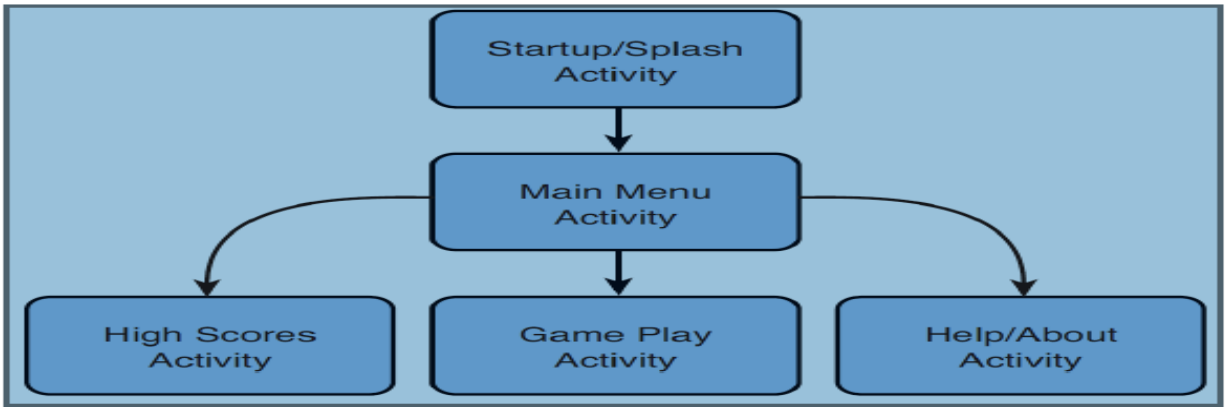
Activities

- Each Activity is a "screen"
- An Activity usually presents a single visual user interface from which number of actions can be performed.
- Allthough activities work together to form a cohesive user interface,each activity is independent of others.
- Typically ,one of the activities is marked as the first one that should be presented to the user when the application is launched
- There can be only one active application visible to the user at a time.

A simple game application might have the following five Activities,as shown in Figure

- **A Startup or Splash screen**: This activity serves as the primary entry point to the application. It displays the application name and version information and transitions to the Main menu after a short interval.
- **A Main Menu screen:** This activity acts as a switch to drive the user to the core Activities of the application. Here the users must choose what they want to do within the application.
- **A Game Play screen:** This activity is where the core game play occurs.
- **A High Scores screen:** This activity might display game scores or settings.
- **A Help/About screen:** This activity might display the information the user might need to play the game.

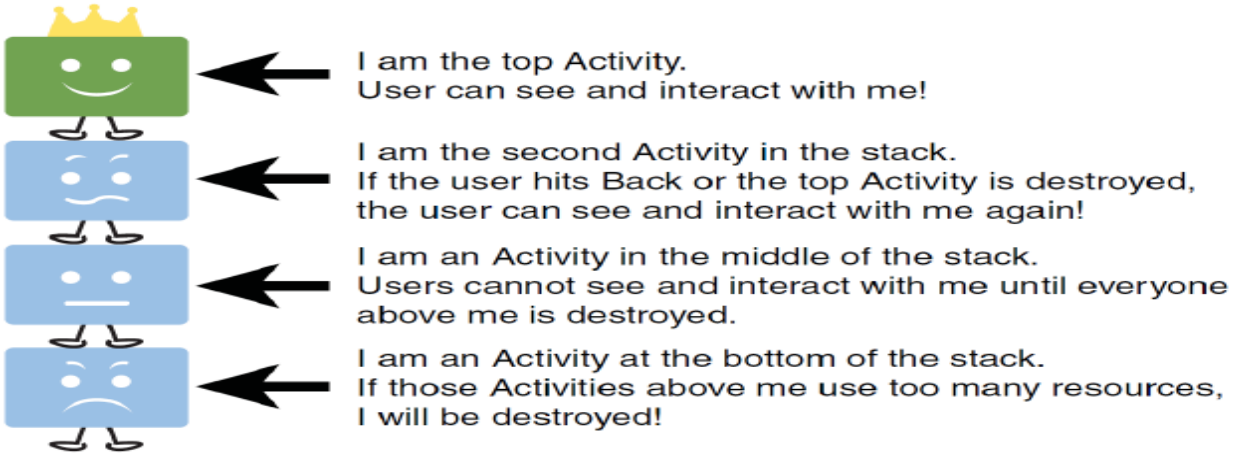
A simple game with **five activities**



The Lifecycle of an Android Activity

Android applications can be multi-process, and the Android operating system allows multiple applications to run concurrently, provided memory and processing power is available.
Applications can have background processes, and applications can be interrupted and paused when events such as phone calls occur.There can be only one active application visible to the user at a time—specifically, a single application Activity is in the foreground at any given time.
The Android operating system keeps track of all Activity objects running by placing them on an Activity stack (see Figure).When a new Activity starts, the Activity on the top of the stack (the current foreground Activity) pauses, and the new Activity pushes onto the top of the stack.When that Activity finishes, that Activity is removed from the activity stack, and the previous Activity in the stack resumes.

The Activity stack.



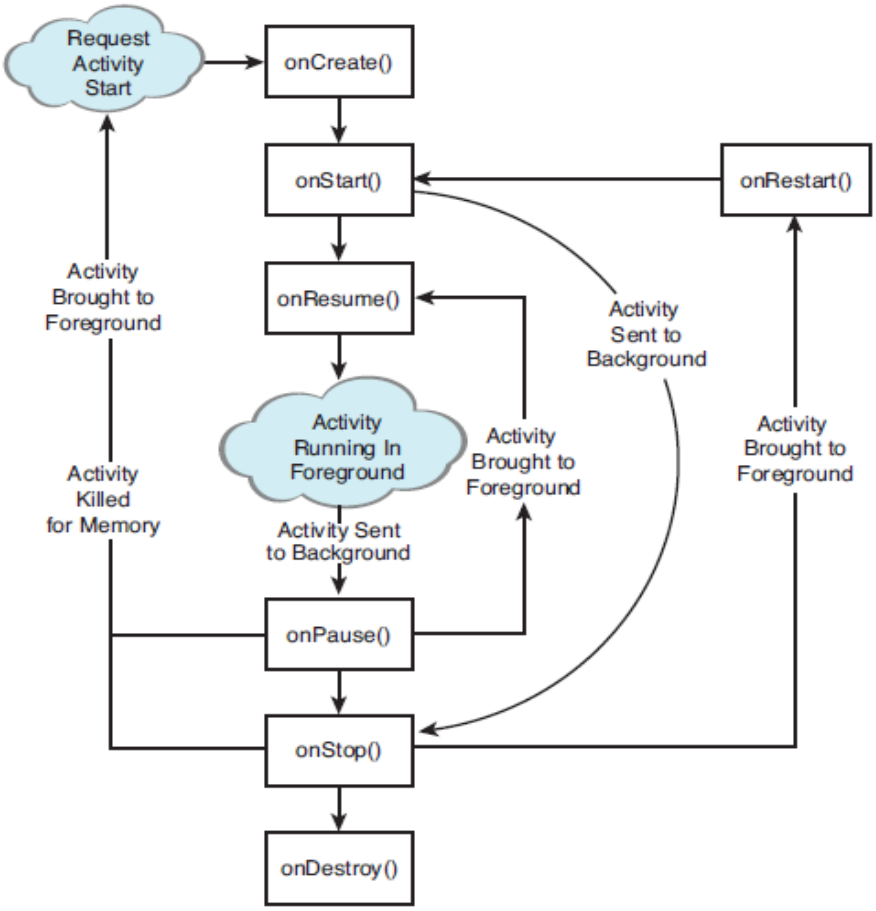
Android applications are responsible for managing their state and their memory, resources, and data.They must pause and resume seamlessly. Understanding the different states within the Activity lifecycle is the first step to designing and developing robust Android applications.

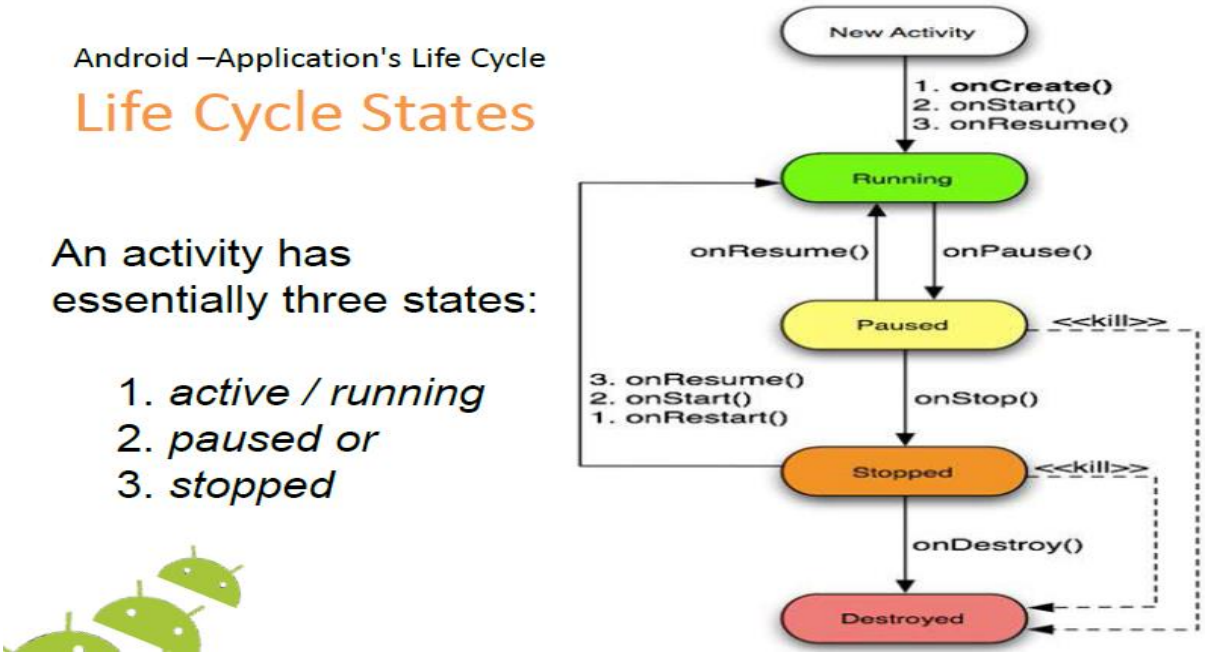
Using Activity Callbacks to Manage Application State and Resources

Different important state changes within the Activity lifecycle are punctuated by a series of important method callbacks.These callbacks are shown in Figure Here are the method stubs for the most important callbacks of the Activity class:

```
public class MyActivity extends Activity {
protected void onCreate(Bundle savedInstanceState);
protected void onStart();
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
}
```

Android-Application's Life Cycle





Life Cycle states ⇨ **Running**

- It is active or running when it is in the foreground of the screen(at the top of the activity stack for the current task.)
- This is the activity that is the focus for the user actions.

Life Cycle states ⇨ **Paused**

- It is paused if it has lost focus but is still visible to the user.
- That is, another activity lies on top of it and that new activity either is transparent or doesn't cover the full screen.
- A paused activity is completely alive(it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.

Life Cycle states ⇨ **Stopped**

- It is stopped if it is completely obscured by another activity.
- It still retains all state and member information. However, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere

Application's Lifetime

The seven transition methods define the entire lifecycle of an activity.

- The entire lifetime of an activity happens between the first call to onCreate() through to a single final call to onDestroy().
- An activity does all its initial setup of "global" state in onCreate(), and releases all remaining resources in onDestroy().

Visible Lifetime

The visible lifetime of an activity happens between a call to onStart() until a corresponding call to onStop().

During this time, the user can see the activity on-screen, though it may not be in the foreground and interacting with the user

- The onStart() and onStop() methods can be called multiple times, as the activity alternates between being visible and hidden to the user.
- Between these two methods, you can maintain resources that are needed to show the activity to the user.

Foreground Lifetime

The foreground lifetime of an activity happens between a call to onResume() until a corresponding call to onPause().

During this time, the activity is in front of all other activities on screen and is interacting with the user

- An activity can frequently transition between the resumed and paused states-for examples,

-onPause() is called when the device goes to sleep or when a new activity is started,

-onResume() is called when an activity result or a new intent is delivered.

Life Cycle Methods

onCreate()

- Called when the activity is first created.
- This is where you should do all of your normal static set up-create views, bind data to lists, and so on.
- This method is passed a Bundleobject containing the activity's previsous state, if that state was captured.
- Always followed by onStart()

onRestart()

- Called after the activity has been stopped, just prior to it being started again.
- Always followed by onStart()

onStart()

- Called just before the activity becomes visible to the user.
- Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.

onResume()

- Called just before the activity starts interacting with the user.
- At this point the activity is at the top of the activity stack, with user input going to it.
- Always followed by onPause()

onPause()

- Called when the system is about to start resuming another activity.
- This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on.
- It should do whatever it does very quickly,because the next activity will not be resumed until it returns.
- Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.
- The activity in this state is killable by the system.

onStop()

- Called when the activity is no longer visible to the user.
- This may happen because it is being destroyed, or because another activity(either an existing one or a new one) has been resumed and is covering it.
- Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.
- The activity in this state is killable by the system.

onDestroy()

- Called before the activity is destroyed.
- This is the final call that the activity will receive.
- It could be called either because the activity is finishing, or because the system is temporarily destroying this instance of the activity to save space.
- You can distinguish between these two scenarios with the isFinishing() method
- The activity in this state is killable by the system.

Service

One application component is the service.An Android Service is basically an Activity without a user interface. It can run as a background process or act much like a web service does, processing requests from third parties.You can use Intents and Activities to launch services using the startService() and bindService() methods.Any Services exposed by an Android application must be registered in the Android Manifest file.

You can use services for different purposes. Generally, you use a service when no input is required from the user. Here are some circumstances in which you might want to implement or use an Android service:

- A weather, email, or social network app might implement a service to routinely check for updates. (Note:There are other implementations for polling, but this is a common use of services.)
- A photo or media app that keeps its data in sync online might implement a service to package and upload new content in the background when the device is idle.
- A video-editing app might offload heavy processing to a queue on its service in order to avoid affecting overall system performance for non-essential tasks.
- A news application might implement a service to “pre-load” content by downloading news stories in advance of when the user launches the application, to improve performance.

A good rule of thumb is that if the task requires the use of a worker thread and might affect application responsiveness and performance, consider implementing a service to handle the task outside the main application lifecycle

Receiving and Broadcasting Intents

Intents serve yet another purpose.You can broadcast an Intent object (via a call to broadcastIntent()) to the Android system, and any application interested can receive that broadcast (called a BroadcastReceiver).Your application might do both sending of and listening for Intent objects.These types of Intent objects are generally used to inform the greater system that something interesting has happened and use special Intent Action types. For example, the Intent action ACTION_BATTERY_LOW broadcasts a warning when the battery is low. If your application is a battery-hogging Service of some kind, you might want to listen for this Broadcast and shut down your Service until the battery power is sufficient.You can register to listen for battery/charge level changes by listening for the broadcast Intent object with the Intent action ACTION_BATTERY_CHANGED.There are also broadcast Intent objects for other interesting system events, such as SD card state changes, applications being installed or removed, and the wallpaper being changed. Your application can also share information using the broadcast mechanism. For example, an email application might broadcast an Intent whenever a new email arrives so that other applications (such as spam or anti-virus apps) that might be interested in this type of event can react to it.

Android Manifest File and its common settings

Android projects use a special configuration file called the Android manifest file to determine application settings—settings such as the application name and version, as well as what permissions the application requires to run and what application components it is comprised of.

Configuring the Android Manifest File

The Android application manifest file is a specially formatted XML file that must accompany each Android application.This file contains important information about the application's identity. Here you define the application's name and version information and what application components the application relies upon, what permissions the application requires to run, and other application configuration information. The Android manifest file is named AndroidManifest.xml and must be included at the top level of any Android project.The information in this file is used by the Android system to

- Install and upgrade the application package.
- Display the application details such as the application name, description, and icon

to users.

- Specify application system requirements, including which Android SDKs are supported,

what hardware configurations are required (for example, d-pad navigation), and which platform features the application relies upon (for example, uses multitouch capabilities).

- Launch application activities.
- Manage application permissions
- Configure other advanced application configuration details, including acting as a service, broadcast receiver, or content provider.
- Enable application settings such as debugging and configuring instrumentation for application testing.

Editing the Android Manifest File

The manifest resides at the top level of your Android project.You can edit the Android manifest file using the Eclipse Manifest File resource editor (a feature of the Android ADT plug-in for Eclipse) or by manually editing the XML

Editing the Manifest File Manually

The Android manifest file is a specially formatted XML file.You can edit the XML manually by clicking on the AndroidManifest.xml tab.

Android manifest files generally include a single <manifest> tag with a single <application> tag.The following is a sample AndroidManifest.xml file for an application called Multimedia

```
<?xml version "1.0" encoding "utf-8"?>
<manifest xmlns:android "http://schemas.android.com/apk/res/android"
package "com.androidbook.multimedia"
android:versionCode "1"
android:versionName "1.0">
<application android:icon "@drawable/icon"
android:label "@string/app_name"
android:debuggable "true">
<activity android:name ".MultimediaMenuActivity"
android:label "@string/app_name">
<intent-filter>
```

```
<action
android:name "android.intent.action.MAIN" />
<category
android:name "android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name "AudioActivity"></activity>
<activity android:name "StillImageActivity"></activity>
<activity android:name "VideoPlayActivity"></activity>
<activity android:name "VideoRecordActivity"></activity>
</application>
<uses-permission
android:name "android.permission.WRITE_SETTINGS" />
<uses-permission
android:name "android.permission.RECORD_AUDIO" />
<uses-permission
android:name "android.permission.SET_WALLPAPER" />
<uses-permission
android:name "android.permission.CAMERA"></uses-permission>
<uses-sdk
android:minSdkVersion "3"
android:targetSdkVersion "8">
</uses-sdk>
<uses-feature
android:name "android.hardware.camera" />
</manifest>
```

Managing Your Application's Identity

Your application's Android manifest file defines the application properties.The package name must be defined within the Android manifest file within the <manifest> tag using the package attribute:

```
<manifest
xmlns:android "http://schemas.android.com/apk/res/android"
package "com.androidbook.multimedia"
android:versionCode "1"
android:versionName "1.0">
```

Versioning Your Application

Versioning your application appropriately is vital to maintaining your application in the field. Intelligent versioning can help reduce confusion and make product support and upgrades simpler.There are two different version attributes defined within the <manifest> tag: the version name and the version code. The version name (android:versionName) is a u ser-friendly, developer-defined version attribute.This information is displayed to users when they manage applications on their devices and when they download the application from marketplaces. Developers use this version information to keep track of their application versions in the field.

Setting the Application Name and Icon

Overall application settings are configured with the <application> tag of the Android manifest file. Here you set information such as the application icon (android:icon) and friendly name (android:label).These settings are attributes of the <application> tag. For example, here we set the application icon to a drawable resource provided with the application package and the application label to a string resource: <application android:icon "@drawable/icon" android:label "@string/app_name"> You can also set optional application settings as attributes in the <application> tag, such as the application description (android:description) and the setting to enable the application for debugging on the device (android:debuggable "true").

Enforcing Application System Requirements

In addition to configuring your application's identity, the Android manifest file is also used to specify any system requirements necessary for the application to run properly. For example, an augmented reality application might require that the handset have GPS, a compass, and a camera. Similarly, an application that relies upon the Bluetooth APIs available within the Android SDK requires a handset with an SDK version of API Level 5 or higher (Android 2.0).These types of system requirements can be defined and enforced in the Android manifest file.Then, when an application is listed on the Android Market, applications can be filtered by these types of information; the Android platform also checks these requirements when installing the application package on the system and errors out if necessary. Some of the application system requirements that developers can configure through the Android manifest file include

- The Android SDK versions supported by the application
- The Android platform features used by the application
- The Android hardware configurations required by the application
- The screen sizes and pixel densities supported by the application
- Any external libraries that the application links to

Targeting Specific SDK Versions

Android devices run different versions of the Android platform. Often, you see old, less powerful, or even less expensive devices running older versions of the Android platform, whereas newer, more powerful devices that show up on the market often run the latest Android software.
There are now dozens of different Android devices in users' hands. Developers must decide who their target audience is for a given application.Are they trying to support the largest population of users and therefore want to support as many different versions of the platform as possible? Or are they developing a bleeding-edge game that requires the latest device hardware?
Developers can specify which versions of the Android platform an application supports within its Android manifest file using the <uses-sdk> tag.This tag has three important attributes:

- The minSdkVersion attribute:This attribute specifies the lowest API level that the application supports.
- The targetSdkVersion attribute:This attribute specifies the optimum API level that the application supports.
- The maxSdkVersion attribute:This attribute specifies the highest API level that the application supports

<uses-sdk android:minSdkVersion=”4” android:targetSdkVersion=”8” />

Android SDK Versions & API Levels

Android SDK Version	API Level
Android 1.0 SDK	1
Android 1.1 SDK	2
Android 1.5 SDK (Cupcake)	3
Android 1.6 SDK (Donut)	4
Android 2.0 SDK (Éclair)	5
Android 2.0.1 SDK (Éclair)	6
Android 2.1 SDK (Éclair)	7
Android 2.2 SDK (FroYo)	8
Android SDK (Gingerbread)	9

Enforcing Application Platform Requirements

Android devices have different hardware and software configurations. Some devices have built-in keyboards and others rely upon the software keyboard. Similarly, certain Android devices support the latest 3-D graphics libraries and others provide little or no graphics support.The Android manifest file has several informational tags for flagging the system features and hardware configurations supported or required by an Android application.

Specifying Supported Input Methods

The <uses-configuration> tag can be used to specify which input methods the application supports.There are different configuration attributes for five-way navigation, the hardware keyboard and keyboard types; navigation devices such as the directional pad, trackball, and wheel; and touch screen settings. There is no “OR” support within a given attribute. If an application supports multiple input configurations, there must be multiple <uses-configuration> tags—one for each complete configuration supported. For example, if your application requires a physical keyboard and touch screen input using a finger or a stylus, you need to define two separate <uses-configuration> tags in your manifest file, as follows:

<uses-configuration android:reqHardKeyboard "true"
android:reqTouchScreen "finger" />
<uses-configuration android:reqHardKeyboard "true"
android:reqTouchScreen "stylus" />

Specifying Required Device Features

Not all Android devices support every Android feature. Put another way:There are a number of APIs (and related hardware) that Android devices may optionally include. For example, not all Android devices have multi-touch ability or a camera flash. The <uses-feature> tag can be used to specify which Android features the application requires to run properly.These settings are for informational purposes only—the Android operating system does not enforce these settings, but publication channels such as the Android Market use this information to filter the applications available to a given user. If your application requires multiple features, you must create a <uses-feature> tag for each. For example, an application that requires both a light and proximity sensor requires two tags:

<uses-feature android:name "android.hardware.sensor.light" />

<uses-feature android:name "android.hardware.sensor.proximity" />

Specifying Supported Screen Sizes

Android devices come in many shapes and sizes. Screen sizes and pixel densities vary widely across the range of Android devices available on the market today. The Android platform categorizes screen types in terms of sizes (small, normal, and large) and pixel density (low, medium, and high).These characteristics effectively cover the variety of screen types available within the Android platform.
An application can provide custom resources for specific screen sizes and pixel densities. The <supportsscreen> tag can be used to specify which Android types of screens the application supports. For example, if the application supports QVGA screens (small) and HVGA screens (normal) regardless of pixel density, the <supports-screen> tag is configured as follows:

<supports-screens android:smallScreens "true"
android:normalScreens "true"
android:largeScreens"false"
android:anyDensity "true"/>

Working with External Libraries

You can register any shared libraries your application links to within the Android manifest file. By default, every application is linked to the standard Android packages (such as android.app) and is aware of its own package. However, if your application links to additional packages, they must be registered within the <application> tag of the Android manifest file using the <uses-library> tag. For example <uses-library android:name "com.sharedlibrary.sharedStuff" />

Registering Activities and Other Application Components

Each Activity within the application must be defined within the Android manifest file with an <activity> tag. For example, the following XML excerpt defines an Activity class called AudioActivity:

<activity android:name "AudioActivity" />

This Activity must be defined as a class within the com.androidbook.multimedia package.That is, the package specified in the <manifest> element of the Android manifest file.You can also enforce scope of the activity class by using the dot as a prefix in the Activity name:

<activity android:name ".AudioActivity" />

Or you can specify the complete class name:

<activity android:name "com.androidbook.multimedia.AudioActivity" />

Designating a Primary Entry Point Activity for Your Application Using an Intent Filter

An Activity class can be designated as the primary entry point by configuring an intent filter using the Android manifest tag <intent-filter> in the application's AndroidManifest.xml file with the MAIN action type and the LAUNCHER category. The following tag of XML configures the Activity class called MultimediaMenuActivity as the primary launching point of the application:

<activity android:name ".MultimediaMenuActivity"
android:label "@string/app_name">
<intent-filter>
<action android:name "android.intent.action.MAIN" />
<category android:name "android.intent.category.LAUNCHER" />
</intent-filter>
</activity>

Working with Permissions

The Android operating system has been locked down so that applications have limited capability to adversely affect operations outside their process space. Instead,Android applications run within the bubble of their own virtual machine, with their own Linux user account (and related permissions).

Registering Permissions Your Application Requires

Android applications have no permissions by default. Instead, any permissions for shared resources or privileged access—whether it's shared data, such as the Contacts database, or access to underlying hardware, such as the built-in camera—must be explicitly registered within the Android manifest file.These permissions are granted when the application is installed. When users install the application, they are informed what permissions the application requires to run and must approve these permissions. Request only the permissions your application requires.

The following XML excerpt for the preceding Android manifest file defines a permission using the <uses-permission> tag to gain access to the built-in camera:

<uses-permission android:name "android.permission.CAMERA" />

A complete list of the permissions can be found in the android.Manifest.permission class.Your application manifest should include only the permissions required to run.The user is informed what permissions each Android application requires at install time.

Registering Permissions Your Application Grants to Other Applications

Applications can also define their own permissions by using the <permission> tag. Permissions must be described and then applied to specific application components, such as Activities, using the android:permission attribute. Permissions can be enforced at several points:

- When starting an Activity or Service
- When accessing data provided by a content provider
- At the function call level
- When sending or receiving broadcasts by an Intent

Permissions can have three primary protection levels: normal, dangerous, and signature.The normal protection level is a good default for fine-grained permission enforcement within the application.The dangerous protection level is used for higherrisk Activities, which might adversely affect the device. Finally, the signature protection level permits any application signed with the same certificate to use that component for controlled application interoperability

Working with different types of resources

What Are Resources?

All Android applications are composed of two things: functionality (code instructions) and data (resources).The functionality is the code that determines how your application behaves. This includes any algorithms that make the application run. Resources include text strings, images and icons, audio files, videos, and other data used by the application.

Storing Application Resources

Android resource files are stored separately from the java class files in the Android project. Most common resource types are stored in XML. You can also store raw data files and graphics as resources

Understanding the Resource Directory Hierarchy

Resources are organized in a strict directory hierarchy within the Android project. All resources must be stored under the /res project directory in specially named subdirectories that must be lowercase.

Different resources types are stored in different directories. The resource sub-directories generated when you create an Android project using the Eclipse plug-in are shown in Table below:

Table 6.1 **Default Android Resource Directories**

Resource Subdirectory	Purpose
/res/drawable-*/	Graphics Resources
/res/layout/	User Interface Resources
/res/values/	Simple Data such as Strings and Color Values, and so on

Each resource type corresponds to a specific resource subdirectory name. For example, all graphics are stored under the /res/drawable directory structure. Resources can be further organized in a variety of ways using even more specially named directory qualifiers. For example, the /res/drawable-hdpi directory stores graphics for high-density screens, the /res/drawable-ldpi directory stores graphics for low-density screens, and the /res/drawable-mdpi directory stores graphics for medium-density screens. If you had a graphic resource that was shared by all screens, you would simply store that resource in the /res/drawable directory.

Using the Android Asset Packaging Tool

If you use the Eclipse with the Android Development Tools Plug-In, you will find that adding resources to your project is simple.The plug-in detects new resources when you add them to the appropriate project resource directory under /res automatically.These resources are compiled, resulting in the generation of the R.java file, which enables you to access your resources programmatically. If you use a different development environment, you need to use the aapt tool command- line interface to compile your resources and package your application binaries to deploy to the phone or emulator.You can find the aapt tool in the /tools subdirectory of each specific Android SDK version.

Storing Different Resource Value Types

The aapt traverses all properly formatted files in the /res directory hierarchy and generates the class file R.java in your source code directory /src to access all variables.

res/subdirectory

- res/anim/

-XML files that are compiled into frame by frame animation objects

- **res/drawable/**

-.png, .9.png, .jpg files. To get a resource of this type
- use Context.getResources().getDrawable(R.drawable.imageId)

- **res/layout/**

-XML files that can be compiled into many kinds of resource.

- **res/values/**

-arrays.xml to define arrays.
- colors.xml to define color drawables and color string values.

- use Resources.getDrawable() & Resources.getColor()

-dimens.xml to define dimension value.

- Resources.getDimension()

-strings.xml to define string values

- Resources.getString()
- Resources.getText() (retain any rich text styling which is usually desirable for UI strings)

-styles.xml to define style objects.

- **res/xml/**

-Arbitary XML files that are compiled and cen be read at run time by calling Resources.getXML().

- **res/raw/**

-Arbitrary files to copy directly to the device.
-To use these resources in your application, call Resources.openRawResource() with the resource ID, which is R.raw.somefilename

UNIT-3
Android User Interface Design

User Interface Screen elements

Introducing the Android View

The Android SDK has a Java packaged named android.view.This package contains a number of interfaces and classes related to drawing on the screen. However, when we refer to the View object,we actually refer to only one of the classes within this package: the android.view.View class. The **View class** is the basic user interface building block within Android. It represents a rectangular portion of the screen.The View class serves as the base class for nearly all the user interface controls and layouts within the Android SDK. View is the base class for widgets, which are used to create interactive UI components (buttons, textfields, etc.).

Using Views

View can be created by

-java code

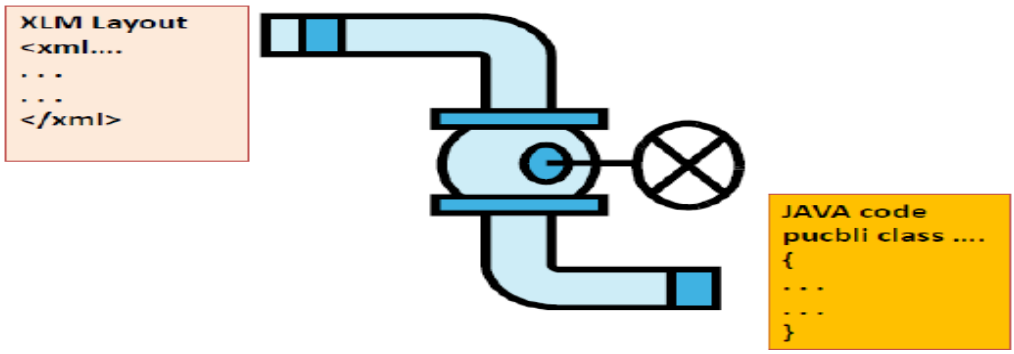
- Specifying a tree of views in one or more XML layout files

Common operations of View

1. **Set properties:** for example setting the text of TextView. Properties that are known at build time can be set in the XML layout files.
2. **Set focus:** The framework will handled moving focus in responsible to user input. To force focus to a specific view, call requestFocus().
3. **Set up listeners:** Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, a Button exposes a listener to notify clients when the button is clicked.
4. **Set visibility**: You can hide or show views using setVisibility(int).

Attaching Layouts to Java Code

PLUMBING: You must 'connect' the XML elements with equivalent objects in your java activity. This allows you to manipulate the UI with code.



This layout could be called by an application using the statement **setContentView(R.layout.main);** Individual widgets, such as myButton could be accessed by the application using the statement findViewByID(…) as in **Button btn=(Button) findViewByID(R.id.myButton);**

Where R is a class automatically generated to keep track of resources available to the application. In particular R.id… is the collection of widgets defined in the XML layout

Attaching Listeners to the Widgets

The button of our example could now be used, for instance a listener for the click event could be written as:

btn.setOnClickListener (new OnClickListener()

```
{
        @Override
        public void onClick (View v)
        {       updateTime(); }
});
private  void updateTime()
{
        Btn.setText (newDate() . toString() );
}
```

Basic Widgets

TextView

One of the most basic user interface elements, or controls, in the Android SDK is the TextView control. You use it, quite simply, to draw text on the screen. You primarily use it to display fixed text strings or labels. Frequently, the TextView control is a child control within other screen elements and controls. As with most of the user interface elements, it is derived from View and is within the android. Widget package. Because it is a View, all the standard attributes such as width, height, padding, and visibility can be applied to the object. However, as a text-displaying control, you can apply many other TextView-specific attributes to control behavior and how the text is viewed in a variety of situations

Example

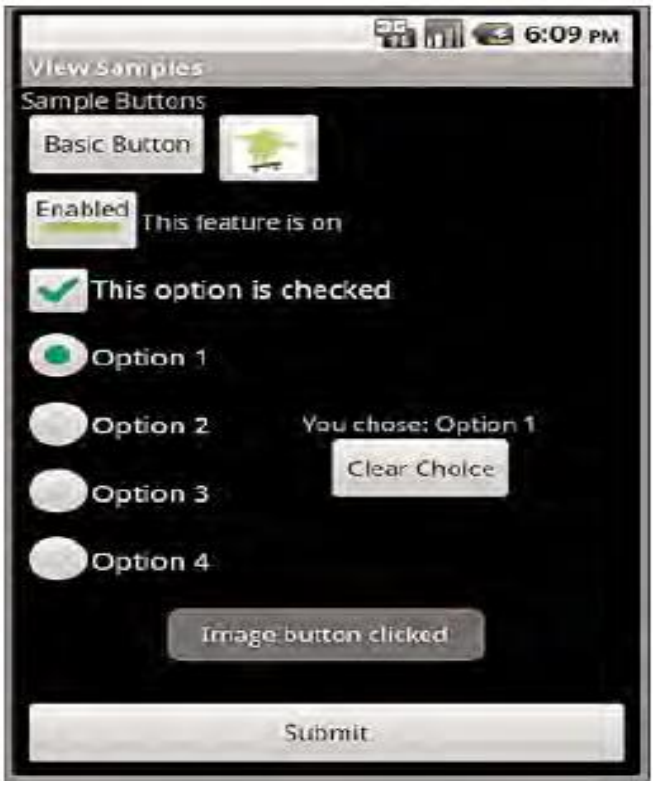

```
<TextView
android:id="@+id/myTextView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="#ff0000ff"
android:padding="3px"
android:text="Enter User Name"
android:textSize="16sp"
android:textStyle="bold"
android:gravity="center"
</TextView>
```

Using Buttons, Check Boxes, and Radio Groups

Another common user interface element is the button. In this section, you learn about different kinds of buttons provided by the Android SDK. These include the basic Button, ToggleButton, CheckBox, and RadioButton. You can find examples of each button type in Figure below. A basic Button is often used to perform some sort of action, such as submitting a form or confirming a selection. A basic Button control can contain a text or image label. A CheckBox is a button with two states—checked or unchecked. You often use CheckBox controls to turn a feature on or off or to pick multiple items from a list. A ToggleButton is similar to a CheckBox, but you use it to visually show the state. The default behavior of a toggle is like that of a power on/off button. A RadioButton provides selection of an item. Grouping RadioButton controls together in a container called a RadioGroup enables the developer to enforce that only one RadioButton is selected at a time.

Using Basic Buttons

The android.widget.Button class provides a basic button implementation in the Android SDK.Within the XML layout resources, buttons are specified using the Button element. The primary attribute for a basic button is the text field.This is the label that appears on the middle of the button's face.You often use basic Button controls for buttons with text such as "Ok,""Cancel," or "Submit.



The following XML layout resource file shows a typical Button control definition:

```
<Button
android:id "@+id/basic_button"
android:layout_width "wrap_content"
android:layout_height "wrap_content"
android:text "Basic Button" />
```

A button won't do anything, other than animate, without some code to handle the click event. Here is an example of some code that handles a click for a basic button and displays a Toast message on the screen:

```
setContentView(R.layout.buttons);
final Button basic_button (Button) findViewById(R.id.basic_button);
basic_button.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
Toast.makeText(ButtonsActivity.this,
"Button clicked", Toast.LENGTH_SHORT).show();
}
});
```

To handle the click event for when a button is pressed,we first get a reference to the Button by its resource identifier. Next, the setOnClickListener() method is called. It requires a valid instance of the

class View.OnClickListener.A simple way to provide this is to define the instance right in the method call.This requires implementing the onClick() method.Within the onClick() method, you are free to carry out whatever actions you need. Here,we simply display a message to the users telling them that the button was, in fact, clicked. A button with its primary label as an image is an ImageButton.An ImageButton is, for most purposes, almost exactly like a basic button. Click actions are handled in the same way.The primary difference is that you can set its src attribute to be an image. Here is an example of an ImageButton definition in an XML layout resource file:

```
<ImageButton
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:id ”@+id/image_button”
android:src ”@drawable/droid” />
```

Using Check Boxes and Toggle Buttons

The check box button is often used in lists of items where the user can select multiple items.The Android check box contains a text attribute that appears to the side of the check box.This is used in a similar way to the label of a basic button. In fact, it’s basically a TextView next to the button.
Here’s an XML layout resource definition for a CheckBox control:

```
<CheckBox
android:id ”@+id/checkbox”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Check me?” />
```

The following example shows how to check for the state of the button programmatically and change the text label to reflect the change:

```
final CheckBox check_button (CheckBox) findViewById(R.id.checkbox);
check_button.setOnClickListener(new View.OnClickListener() {
public void onClick (View v) {
TextView tv (TextView)findViewById(R.id.checkbox);
tv.setText(check_button.isChecked() ?
“This option is checked” :
“This option is not checked”);
}
});
```

This is similar to the basic button.A check box automatically shows the check as enabled or disabled.This enables us to deal with behavior in our application rather than worrying about how the button should behave.The layout shows that the text starts out one way but, after the user presses the button, the text changes to one of two different things depending on the checked state.As the code shows, the CheckBox is also a TextView. A Toggle Button is similar to a check box in behavior but is usually used to show or alter the on or off state of something. Like the CheckBox, it has a state (checked or not). Also like the check box, the act of changing what displays on the button is handled for us. Unlike the CheckBox, it does not show text next to it. Instead, it has two text fields.The first attribute is textOn, which is the text that displays on the button when its checked state is on.The second attribute is textOff, which is the text that displays on the button when its checked state is off.The default text for these is “ON” and “OFF,” respectively. The following layout code shows a definition for a toggle button that shows “Enabled” or “Disabled” based on the state of the button:

<ToggleButton
android:id ”@+id/toggle_button”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Toggle”
android:textOff ”Disabled”
android:textOn ”Enabled” />

This type of button does not actually display the value for the text attribute, even though it’s a valid attribute to set. Here, the only purpose it serves is to demonstrate that it doesn’t display.You can see what this ToggleButton looks like in Figure above.

Using RadioGroups and RadioButtons

You often use radio buttons when a user should be allowed to only select one item from a small group of items. For instance, a question asking for gender can give three options: male, female, and unspecified. Only one of these options should be checked at a time.The RadioButton objects are similar to CheckBox objects.They have a text label next to them, set via the text attribute, and they have a state (checked or unchecked). However, you can group RadioButton objects inside a RadioGroup that handles enforcing their combined states so that only one RadioButton can be checked at a time. If the user selects a RadioButton that is already checked, it does not become unchecked. However, you can provide the user with an action to clear the state of the entire RadioGroup so that none of the buttons are checked. Here we have an XML layout resource with a RadioGroup containing four RadioButton objects (shown in Figure , toward the bottom of the screen).The RadioButton objects have text labels,“Option 1,” and so on.The XML layout resource definition is shown here:

<RadioGroup
android:id ”@+id/RadioGroup01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”>
<RadioButton
android:id ”@+id/RadioButton01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Option 1”></RadioButton>
<RadioButton
android:id ”@+id/RadioButton02”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Option 2”></RadioButton>
<RadioButton
android:id ”@+id/RadioButton03”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:text ”Option 3”></RadioButton>
<RadioButton
android:id ”@+id/RadioButton04”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”

```
android:text ”Option 4”></RadioButton>
</RadioGroup>
```

You handle actions on these RadioButton objects through the RadioGroup object.The following example shows registering for clicks on the RadioButton objects within the RadioGroup:

```
final RadioGroup group (RadioGroup)findViewById(R.id.RadioGroup01);
final TextView tv (TextView)
findViewById(R.id.TextView01);
group.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
public void onCheckedChanged(RadioGroup group, int checkedId) {
if (checkedId ! -1) {
RadioButton rb (RadioButton)
findViewById(checkedId);
if (rb ! null) {
tv.setText(“You chose: “ + rb.getText());
}
} else {
tv.setText(“Choose 1”);
}
}
});
```

As this layout example demonstrates, there is nothing special that you need to do to make the RadioGroup and internal RadioButton objects work properly.The preceding code illustrates how to register to receive a notification whenever the RadioButton selection changes.

The code demonstrates that the notification contains the resource identifier for the specific RadioButton that the user chose, as assigned in the layout file.To do something interesting with this, you need to provide a mapping between this resource identifier (or the text label) to the corresponding functionality in your code. In the example,we query for the button that was selected, get its text, and assign its text to another TextView control that we have on the screen. As mentioned, the entire RadioGroup can be cleared so that none of the RadioButton objects are selected.The following example demonstrates how to do this in response to a button click outside of the **RadioGroup:**

```
final Button clear_choice (Button) findViewById(R.id.Button01);
clear_choice.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
RadioGroup group (RadioGroup)
findViewById(R.id.RadioGroup01);
if (group ! null) {
group.clearCheck();
}
}
}
```

The action of calling the clearCheck() method triggers a call to the onCheckedChangedListener() callback method.This is why we have to make sure that the resource identifier we received is valid.

Right after a call to the clearCheck() method, it is not a valid identifier but instead is set to the value -1 to indicate that no RadioButton is currently checked.

Getting Dates and Times from Users

The Android SDK provides a couple controls for getting date and time input from the user.The first is the DatePicker control (Figure 7.8, top). It can be used to get a month, day, and year from the user.

Date and time controls.

The basic XML layout resource definition for a DatePicker follows:

```
<DatePicker
android:id ”@+id/DatePicker01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content” />
```

As you can see from this example, there aren’t any attributes specific to the DatePicker control.As with many of the other controls, your code can register to receive a method call when the date changes.You do this by implementing the onDateChanged() method. However, this isn’t done the usual way.

```
final DatePicker date (DatePicker)findViewById(R.id.DatePicker01);
date.init(date.getYear(), date.getMonth(), date.getDayOfMonth(),
new DatePicker.OnDateChangedListener() {
public void onDateChanged(DatePicker view, int year,
int monthOfYear, int dayOfMonth) {
Date dt new Date(year-1900,

monthOfYear, dayOfMonth, time.getCurrentHour(),
time.getCurrentMinute());
text.setText(dt.toString());
}
```

```
});
```

The preceding code sets the DatePicker.OnDateChangedListener by a call to the DatePicker.init() method.A DatePicker control is initialized with the current date.A TextView is set with the date value that the user entered into the DatePicker control. The value of 1900 is subtracted from the year parameter to make it compatible with the java.util.Date class**. A TimePicker control** (also shown in Figure above, bottom) is similar to the DatePicker control. It also doesn't have any unique attributes. However, to register for a method call when the values change, you call the more traditional method of TimePicker.setOnTimeChangedListener().

```
time.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener()
{
public void onTimeChanged(TimePicker view,
int hourOfDay, int minute) {
Date dt new Date(date.getYear()-1900, date.getMonth(),
date.getDayOfMonth(), hourOfDay, minute);
text.setText(dt.toString());
}
});
```

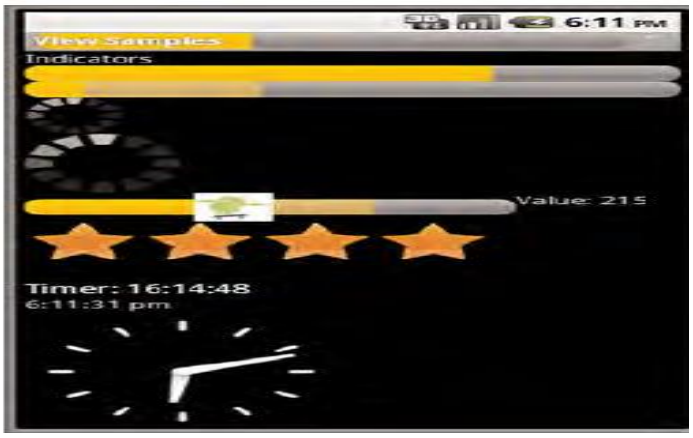
As in the previous example, this code also sets a TextView to a string displaying the time value that the user entered.When you use the DatePicker control and the TimePicker control together, the user can set a full date and time.

Using Indicators to Display Data to Users

The Android SDK provides a number of controls that can be used to visually show some form of information to the user.These indicator controls include **progress bars, clocks**, and other similar controls.

Indicating Progress with ProgressBar

Applications commonly perform actions that can take a while.A good practice during this time is to show the user some sort of progress indicator that informs the user that the application is off "doing something."Applications can also show how far a user is through some operation, such as a playing a song or watching a video.The Android SDK provides several types of progress bars. The standard progress bar is a circular indicator that only animates. It does not show how complete an action is. It can, however, show that something is taking place.This is useful when an action is indeterminate in length.There are three sizes of this type of progress indicator see Figure below.



Various types of progress and rating indicators

The second type is a horizontal progress bar that shows the completeness of an action. (For eg, you can see how much of a file is downloading.) This horizontal progress bar can also have a secondary progress indicator on it.This can be used, for instance, to show the completion of a downloading media file while that file plays. This is an XML layout resource definition for a basic indeterminate progress bar:

```
<ProgressBar
android:id ”@+id/progress_bar”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content” />
```

The default style is for a medium-size circular progress indicator; not a “bar” at all. The other two styles for indeterminate progress bar are progressBarStyleLarge and progressBarStyleSmall.This style animates automatically.The next sample shows the layout definition for a horizontal progress indicator

```
<ProgressBar
android:id ”@+id/progress_bar”
style ”?android:attr/progressBarStyleHorizontal”
android:layout_width ”fill_parent”
android:layout_height ”wrap_content”
android:max ”100” />
```

We have also set the attribute for max in this sample to 100.This can help mimic a percentage progress bar.That is, setting the progress to 75 shows the indicator at 75 percent complete. We can set the indicator progress status programmatically as follows:

```
mProgress (ProgressBar) findViewById(R.id.progress_bar);
mProgress.setProgress(75);
```

You can also put these progress bars in your application’s title bar (as shown in Figure above) This can save screen real estate.This can also make it easy to turn on and off an indeterminate progress indicator without changing the look of the screen. Indeterminate progress indicators are commonly used to display progress on pages where items need to be loaded before the page can finish drawing.This is often employed on web browser screens.The following code demonstrates how to place this type of indeterminate progress indicator on your Activity screen:

```
requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
requestWindowFeature(Window.FEATURE_PROGRESS);
setContentView(R.layout.indicators);
setProgressBarIndeterminateVisibility(true);
setProgressBarVisibility(true);
setProgress(5000);
```

Showing Time Passage with the Chronometer

Sometimes you want to show time passing instead of incremental progress. In this case, you can use the Chronometer control as a timer .This might be useful if it’s the user who is taking time doing some task or in a game where some action needs to be timed.The Chronometer control can be formatted with text, as shown in this XML layout resource definition:

```
<Chronometer
android:id ”@+id/Chronometer01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:format ”Timer: %s” />
```

You can use the Chronometer object's format attribute to put text around the time that displays.A Chronometer won't show the passage of time until its start() method is called.To stop it, simply call its stop() method. Finally, you can change the time from which the timer is counting.That is, you can set it to count from a particular time in the past instead of from the time it's started.You call the setBase() method to do this In this next example code, the timer is retrieved from the View by its resource identifier. We then check its base value and set it to 0. Finally,we start the timer counting up from there.

```
final Chronometer timer
(Chronometer)findViewById(R.id.Chronometer01);
long base timer.getBase();
Log.d(ViewsMenu.debugTag, "base "+ base);
timer.setBase(0);
timer.start();
```

Displaying the Time

Displaying the time in an application is often not necessary because Android devices have a status bar to display the current time. However, there are two clock controls available to display this information: the DigitalClock and AnalogClock controls.

Using the DigitalClock

The DigitalClock control is a compact text display of the current time in standard numeric format based on the users' settings. It is a TextView, so anything you can do with a TextView you can do with this control, except change its text.You can change the color and style of the text, for example.

By default, the DigitalClock control shows the seconds and automatically updates as each second ticks by. Here is an example of an XML layout resource definition for a DigitalClock control:

```
<DigitalClock
android:id "@+id/DigitalClock01"
android:layout_width "wrap_content"
android:layout_height "wrap_content" />
```

Using the AnalogClock

The AnalogClock control is a dial-based clock with a basic clock face with two hands, one for the minute and one for the hour. It updates automatically as each minute passes.The image of the clock scales appropriately with the size of its View. Here is an example of an XML layout resource definition for an AnalogClock control:

```
<AnalogClock
android:id "@+id/AnalogClock01"
android:layout_width "wrap_content"
android:layout_height "wrap_content" />
```

The AnalogClock control's clock face is simple. However you can set its minute and hour hands.You can also set the clock face to specific drawable resources, if you want to jazz it up. Neither of these clock controls accepts a different time or a static time to display.They can show only the current time in the current time zone of the device, so they are not particularly useful

Working with Dialogs

An Activity can use dialogs to organize information and react to user-driven events. For example, an activity might display a dialog informing the user of a problem or ask the user to confirm an action such as deleting a data record. Using dialogs for simple tasks helps keep the number of application activities manageable

Exploring the Different Types of Dialogs

There are a number of different dialog types available within the Android SDK. Each has a special function that most users should be somewhat familiar with.The dialog types available include

- Dialog: The basic class for all Dialog types. A basic Dialog is shown in the top left of Figure below.
- AlertDialog: A Dialog with one, two, or three Button controls. An AlertDialog is shown in the top center of Figure below.
- CharacterPickerDialog: A Dialog for choosing an accented character associated with a base character.A CharacterPickerDialog is shown in the top right of Figure below.
- DatePickerDialog: A Dialog with a DatePicker control. A DatePickerDialog is shown in the bottom left of Figure below.
- ProgressDialog: A Dialog with a determinate or indeterminate ProgressBar control. An indeterminate ProgressDialog is shown in the bottom center of Figure below.
- TimePickerDialog: A Dialog with a TimePicker control. A TimePickerDialog is shown in the bottom right of Figure below.

The different dialog types available in Android

Tracing the Lifecycle of a Dialog

Each Dialog must be defined within the Activity in which it is used.A Dialog may be launched once, or used repeatedly. Understanding how an Activity manages the Dialog lifecycle is important to implementing a Dialog correctly. Let's look at the key methods that an Activity must use to manage a Dialog:

- The showDialog() method is used to display a Dialog.
- The dismissDialog() method is used to stop showing a Dialog.The Dialog is kept around in the Activity's Dialog pool. If the Dialog is shown again using showDialog(), the cached version is displayed once more.
- The removeDialog() method is used to remove a Dialog from the Activity objects Dialog pool.The Dialog is no longer kept around for future use. If you call showDialog() again, the Dialog must be re-created.

Adding the Dialog to an Activity involves several steps:

1. Define a unique identifier for the Dialog within the Activity.
2. Implement the onCreateDialog() method of the Activity to return a Dialog of the appropriate type, when supplied the unique identifier
3. Implement the onPrepareDialog() method of the Activity to initialize the Dialog as appropriate.
4. Launch the Dialog using the showDialog() method with the unique identifier.

Defining a Dialog

A Dialog used by an Activity must be defined in advance. Each Dialog has a special identifier (an integer).When the showDialog() method is called, you pass in this identifier. At this point, the onCreateDialog() method is called and must return a Dialog of the appropriate type. It is up to the developer to override the onCreateDialog() method of the Activity and return the appropriate Dialog for a given identifier. If an Activity has multiple Dialog windows, the onCreateDialog() method generally contains a switch statement to return the appropriate Dialog based on the incoming parameter—the Dialog identifier.

Initializing a Dialog

Because a Dialog is often kept around by the Activity in its Dialog pool, it might be important to re-initialize a Dialog each time it is shown, instead of just when it is created the first time. For this purpose, you can override the onPrepareDialog() method of the Activity. Although the onCreateDialog() method may only be called once for initial Dialog creation, the onPrepareDialog() method is called each time the showDialog() method is called, giving the Activity a chance to modify the Dialog before it is shown to the user.

Launching a Dialog

You can display any Dialog defined within an Activity by calling its showDialog() method and passing it a valid Dialog identifier—one that will be recognized by the onCreateDialog() method.

Dismissing a Dialog

Most types of dialogs have automatic dismissal circumstances. However, if you want to force a Dialog to be dismissed, simply call the dismissDialog() method and pass in the Dialog identifier.

Removing a Dialog from Use

Dismissing a Dialog does not destroy it. If the Dialog is shown again, its cached contents are redisplayed. If you want to force an Activity to remove a Dialog from its pool and not use it again, you can call the removeDialog() method, passing in the valid Dialog identifier.

Providing Users with Options and Context Menus

You need to be aware of two special application menus for use within your Android applications: **the options menu** and the **context menu**.

Enabling the Options Menu

The Android SDK provides a method for users to bring up a menu by pressing the menu key from within the application (see Figure below).You can use options menus within your application to bring up help, to navigate, to provide additional controls, or to configure options.The OptionsMenu control can contain icons, submenus, and keyboard shortcuts.

An option menu

For an options menu to show when a user presses the Menu button on their device, you need to override the implementation of onCreateOptionsMenu() in your Activity. Here is a sample implementation that gives the user three menu items to choose from:

```
public boolean onCreateOptionsMenu( android.view.Menu menu) {
super.onCreateOptionsMenu(menu);
menu.add("Forms")
.setIcon(android.R.drawable.ic_menu_edit)
.setIntent(new Intent(this, FormsActivity.class));
menu.add("Indicators")
.setIntent(new Intent(this, IndicatorsActivity.class))
.setIcon(android.R.drawable.ic_menu_info_details);
menu.add("Containers")
.setIcon(android.R.drawable.ic_menu_view)
.setIntent(new Intent(this, ContainersActivity.class));
return true; }
```

Enabling the ContextMenu

The ContextMenu is a subtype of Menu that you can configure to display when a long press is performed on a View.As the name implies, the ContextMenu provides for contextual menus to display to the user for performing additional actions on selected items. ContextMenu objects are slightly more complex than OptionsMenu objects.You need to implement the onCreateContextMenu() method of your Activity for one to display. However, before that is called, you must call the registerForContextMenu() method and pass in the View for which you want to have a context menu.This means each View on your screen can have a different context menu, which is appropriate as the menus are designed to be highly contextual Here we have an example of a Chronometer timer, which responds to a long click with a context menu:

```
registerForContextMenu(timer);
```

After the call to the registerForContextMenu() method has been executed, the user can then long click on the View to open the context menu. Each time this happens, your Activity gets a call to the onCreateContextMenu() method, and your code creates the menu each time the user performs the long click. The following is an example of a context menu for the Chronometer control, as previously used:

```
public void onCreateContextMenu(
ContextMenu menu, View v, ContextMenuInfo menuInfo) {
super.onCreateContextMenu(menu, v, menuInfo);
if (v.getId() R.id.Chronometer01) {
getMenuInflater().inflate(R.menu.timer_context, menu);
menu.setHeaderIcon(android.R.drawable.ic_media_play)
.setHeaderTitle("Timer controls");
} }
```

Recall that any View control can register to trigger a call to the onCreateContextMenu() method when the user performs a long press.That means we have to check which View control it was for which the user tried to get a context menu. Next,we inflate the appropriate menu from a menu resource that we defined with XML. Because we can't define header information in the menu resource file,we set a stock Android SDK resource to it and add a title. Here is the menu resource that is inflated:

```
<menu
```

```
xmlns:android ”http://schemas.android.com/apk/res/android”>
<item
android:id ”@+id/start_timer”
android:title ”Start” />
<item
android:id ”@+id/stop_timer”
android:title ”Stop” />
<item
android:id ”@+id/reset_timer”
android:title ”Reset” />
</menu>
```

This defines three menu items. If this weren't a context menu,we could have assigned icons. However, context menus do not support icons, submenus, or shortcuts

Designing User Interfaces with Layouts

Application user interfaces can be simple or complex, involving many different screens or only a few. Layouts and user interface controls can be defined as application resources or created programmatically at runtime.

Creating Layouts Using XML Resources

Android provides a simple way to create layout files in XML as resources provided in the /res/layout project directory.This is the most common and convenient way to build Android user interfaces and is especially useful for defining static screen elements and control properties that you know in advance, and to set default attributes that you can modify programmatically at runtime.

An XML based layout is a specification of the various UI components(widgets) and the relationships to each other –and to their containers-all written in XML format-".

XML Layouts/Containers

1. LinerLayout (the box model),
2. RelativeLayout (a rule-based model), and
3. TableLayout (a grid model), along with
4. ScrollView, a container designed to assist with implementing scrolling containers
5. Other (ListView, GridView, WebView, MapView,…….)

The LinerLayout

LinerLayout is a box model-widgets or child containers are lined up in a column or row, one after the next.

Configure a LinerLayout

- Container's contents 5 areas to be configure

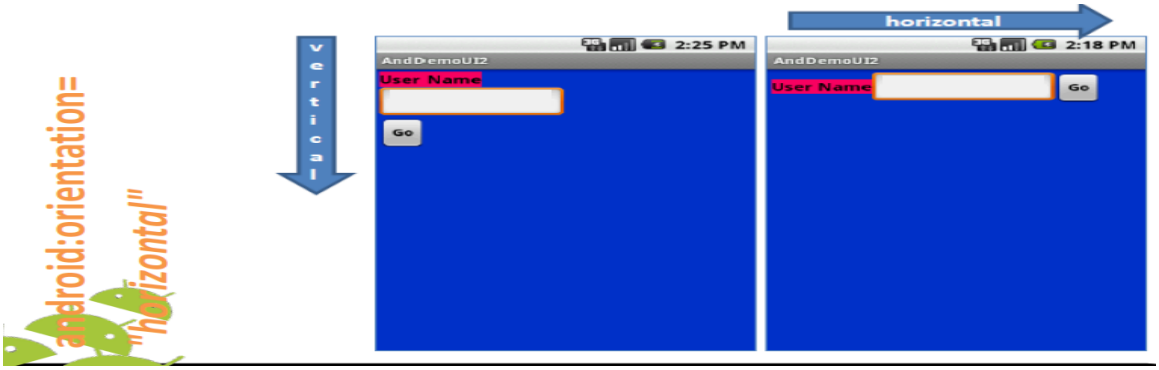
1. Orientation,
2. Fill model,
3. Weight,
4. Gravity,
5. Padding,
6. margin

Orientation

- Indicates whether the LinerLayout represents a row or a column.
- Add the android:orientation property to your LinearLayout element in your XML layout, setting the value to be horizontal for a row or vertical for a column.

- The orientation can be modified at runtime by invoking setOrientation()

Orientation indicates whether the LinerLayout represents a row(HORIZONTAL) or a column (VERTICAL).



Fill Model

- Widgets hava a natural size based on their accompanying text.
- When their combined sizes does not exactly match the width of the Android device's screen, we may have the issue of what to do with the remaining space.

All widgets inside a LinearLayout must supply dimensional attributes
android:layout_width and android:layout_height
to help address the issue of empty space. Values used in defining height and width are:

1. Specific a particular dimension, such as 125dip(device independent pixels)
2. Provide wrap_constant, which means the widget should fill up its natural space, unless that is too big, in which case Android can use Word_wrap as needed to make it fit.
3. Provide fill_parent, which means the widget should fill up all available space in its enclosing container, after all other widgets are taken care of.

Weight

It is used to proportionally assign space to widgets in a view.
You set android:layout_weight to a value(1, 2, 3,……) to indicates what proportion of the free space should go to that widget.

Example

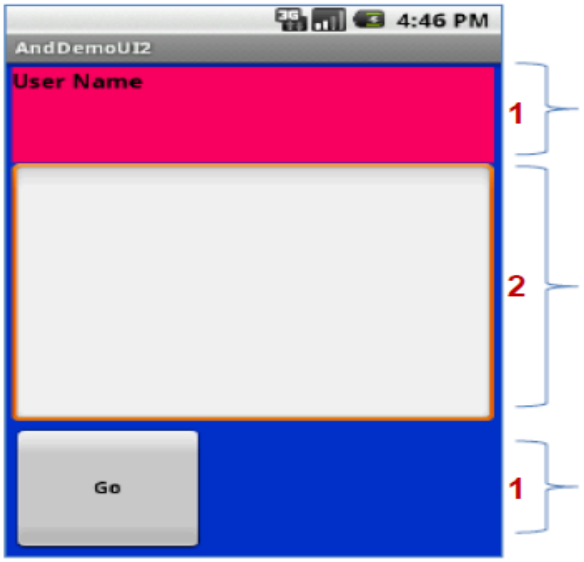
Both the textview and the Button widgets have been set as in the previous example. Both have the additional property

android:layout_weight=”1”

whereas the EditText control has

android:layout_weight=”2”

Default value is 0



Gravity

It is used to indicate how a control will align on the screen. By default, widgets are left and top-aligned.
You may use the XML property android:layout_gravity=”…”
To set other possible arrangements left, center, right, top, bottom, etc.

gravity vs.layout_gravity

android:gravity

specifies how to place the content of an object, both on the x-and y-axis within the object itself.

android:gravity=”center”

android:layout_gravity

positions the view with respect to its parent(i.e. what the view is contained in).

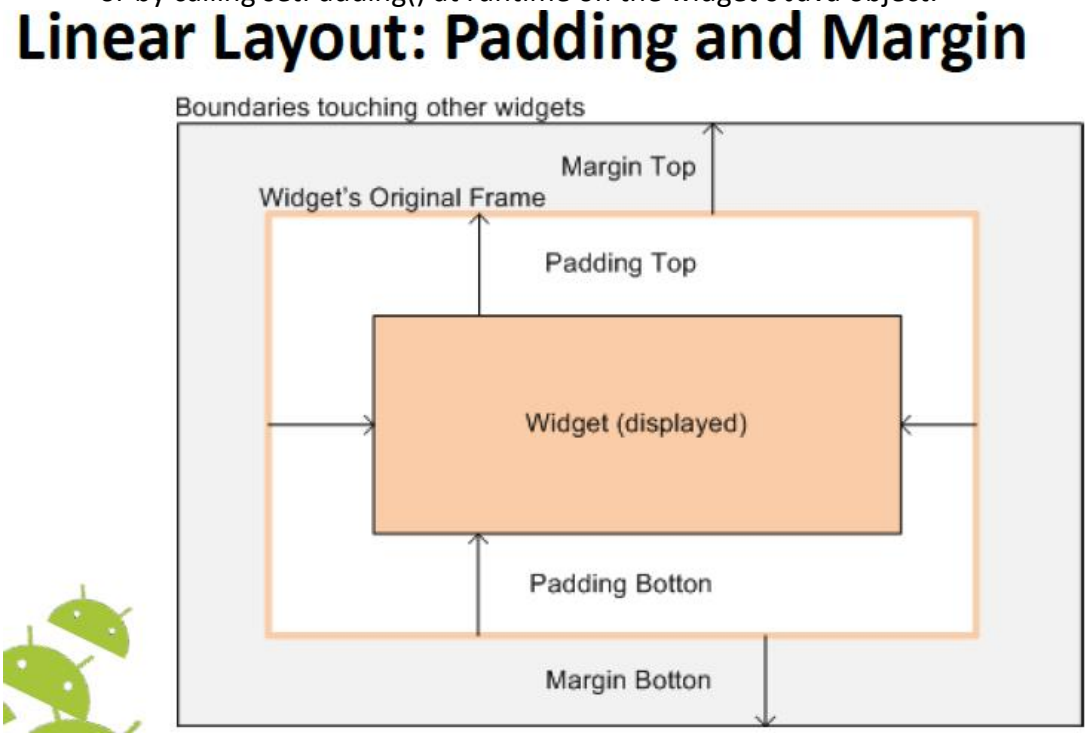
android:layout_gravity=”center”

Padding

The padding specifies how much space there is between the boundaries of the widgets’s “cell” and the actual widget contents.

If you want to increase the internal whitspace between the edges of the and its contents, you will want to use the:

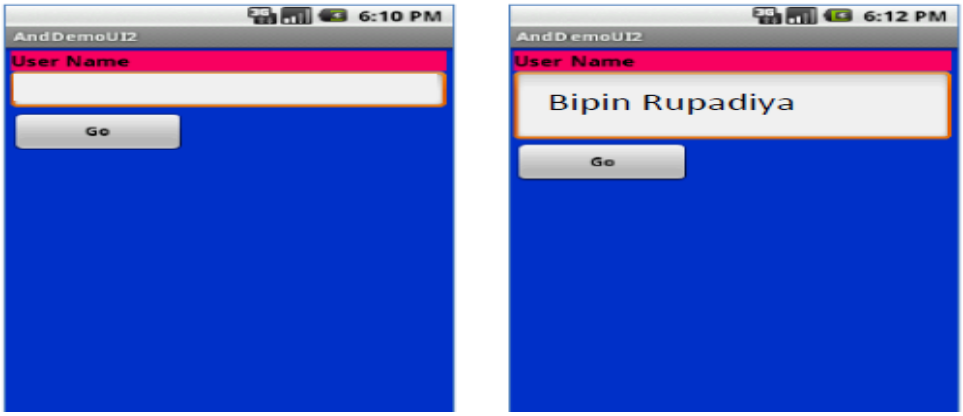
- **android:padding** property
- or by calling setPadding() at runtime on the widget’s Java object.



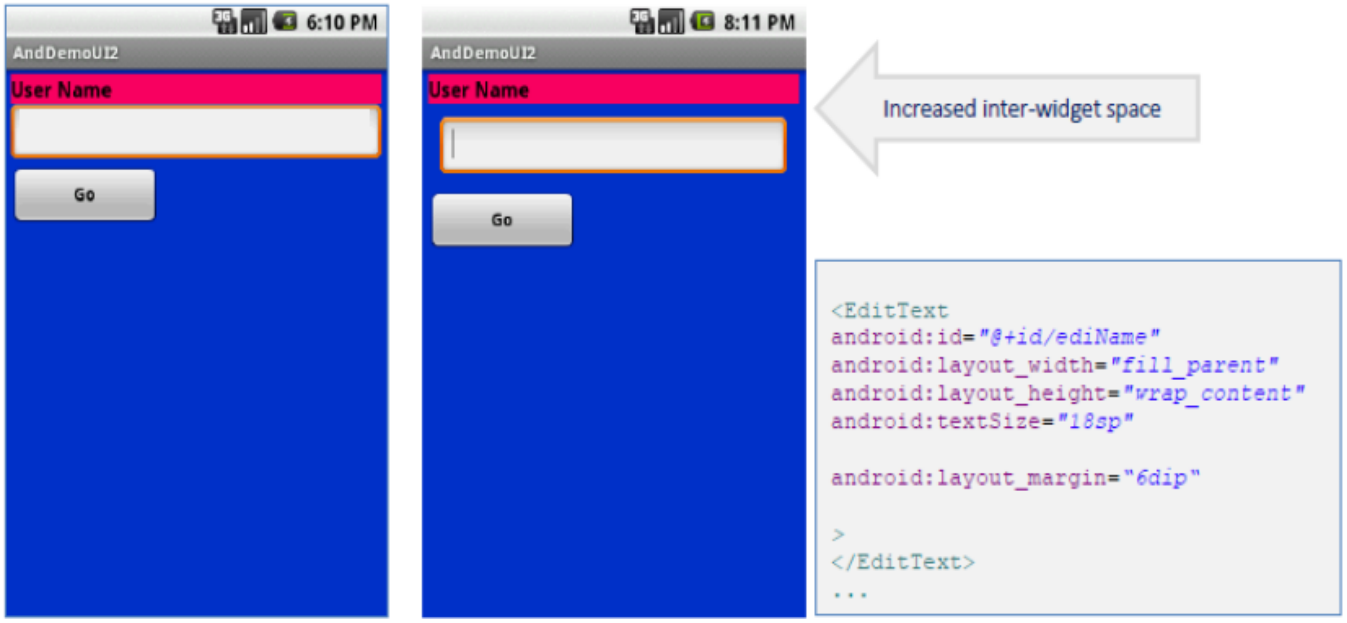
Linear Layout:Internal Margins Using Padding

Example:
The EditText box has been changed to display 30dip of padding all around

```
<EditText
android:id=”@+id/ediName”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:textSize=”18sp”
android:padding=”30dip” >
</EditText>
…
```



Liner Layout:(External) Marging
By default, widgets are tightly packed next to each other.
To increase space between them use the **android:layout_margin** attribute



Example:

```
Public void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
```

```
        TextView text1=new TextView(this);
        text1.setText("hi there!");
        TextView text2=new TextView(this);
        text2.setText("I'm second. I need to wrap.");
        text2.setTextSize((float) 60);
        LinearLayout ll=new LinearLayout(this);
        ll.setOrientation(LinearLayout.VERTICAL);
        ll.addView(text1);
        ll.addView(text2);
        setContentView(ll);
}
```

Relative Layout

The RelativeLayout view enables you to specify where the child view controls are in relation to each other. For instance, you can set a child View to be positioned "above" or "below" or "to the left of " or "to the right of " another View, referred to by its unique identifier.You can also align child View objects relative to one another or the parent layout edges. Combining RelativeLayout attributes can simplify creating interesting user interfaces without resorting to multiple layout groups to achieve a desired effect. Figure 8.6 shows how each of the button controls is relative to each other. You can find the layout attributes available for RelativeLayout child View objects in android.control.RelativeLayout.LayoutParams.Table 8.4 describes some of the important attributes specific to RelativeLayout views

Figure 8.6 An example of **RelativeLayout** usage.

Table 8.4 Important RelativeLayout View Attributes

Attribute Name	Applies To	Description	Value
android:gravity	Parent view	Gravity of child views within layout	One or more constants separated by "|". The constants available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill.
android: layout_ centerInParent	Child view	Centers child view horizontally and vertically within parent view	True or false
android: layout_ centerHorizontal	Child view	Centers child view horizontally within parent view.	True or false
android: layout_ centerVertical	Child view	Centers child view vertically within parent view	True or false
android: layout_ alignParentTop	Child view	Aligns child view with top edge of parent view.	True or false
android: layout_ alignParentBottom	Child view	Aligns child view with bottom edge of parent view.	True or false
android: layout_ alignParentLeft	Child view	Aligns child view with left edge of parent view.	True or false
android: layout_ alignParentRight	Child view	Aligns child view with right edge of parent view.	True or false
android: layout_ alignRight	Child view	Aligns child view with right edge of another child view, specified by ID.	A view ID; for example, @id/Button1
android: layout_ alignLeft	Child view	Aligns child view with left edge of another child view, specified by ID.	A view ID; for example, @id/Button1
android: layout_ alignTop	Child view	Aligns child view with top edge of another child view, specified by ID.	A view ID; for example, @id/Button1
android:layout_ alignBottom	Child view	Aligns child view with bottom edge of another child view, specified by ID.	A view ID; for example, @id/Button1

android: layout_ above	Child view	Positions bottom edge of child view above another child view, specified by ID.	A view ID; for example, @id/Button1
android: layout_ below	Child view	Positions top edge of child view below another child view, specified by ID.	A view ID; for example, @id/Button1
android:layout_ toLeftOf	Child view	Positions right edge of child view to the left of another child view, specified by ID.	A view ID; for example, @id/Button1
android:layout_ toRightOf	Child view	Positions left edge of child view to the right of another child view, specified by ID.	A view ID; for example, @id/Button1

Here's an example of an XML layout resource with a RelativeLayout and two child View objects, a Button object aligned relative to its parent, and an ImageView aligned and positioned relative to the Button (and the parent):

```
<?xml version ”1.0” encoding ”utf-8”?>
<RelativeLayout xmlns:android
“http://schemas.android.com/apk/res/android”
android:id ”@+id/RelativeLayout01”
android:layout_height ”fill_parent”
android:layout_width ”fill_parent”>
<Button
android:id ”@+id/ButtonCenter”
android:text ”Center”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:layout_centerInParent ”true” />
<ImageView
android:id ”@+id/ImageView01”
android:layout_width ”wrap_content”
android:layout_height ”wrap_content”
android:layout_above ”@id/ButtonCenter”
android:layout_centerHorizontal ”true”
android:src ”@drawable/arrow” />
</RelativeLayout>
```

Table Layout

A TableLayout view organizes children into rows, as shown in Figure 8.7.You add individual View objects within each row of the table using a TableRow layout View (which is basically a horizontally oriented LinearLayout) for each row of the table. Each column of the TableRow can contain one View (or layout with child View objects).You place View items added to a TableRow in columns in the order they are added.You can specify the column number (zero-based) to skip columns as necessary (the bottom row shown in Figure 8.7 demonstrates this); otherwise, the View object is put in the next column to the right. Columns scale to the size of the largest View of that column.You can also include normal View objects instead of TableRow elements, if you want the View to take up an entire row.

Figure 8.7 An example of `TableLayout` usage.

You can find the layout attributes available for TableLayout child View objects in android.control.TableLayout.LayoutParams.You can find the layout attributes available for TableRow child View objects in android.control.TableRow.LayoutParams. Table 8.5 describes some of the important attributes specific to TableLayout View objects.

Table 8.5 Important TableLayout and TableRow View Attributes

Attribute Name	Applies To	Description	Value
android: collapseColumns	TableLayout	A comma-delimited list of column indices to collapse (0-based)	String or string resource. For example, “0,1,3,5”
android: shrinkColumns	TableLayout	A comma-delimited list of column indices to shrink (0-based)	String or string resource. Use “*” for all columns. For example, “0,1,3,5”
andriod: stretchColumns	TableLayout	A comma-delimited list of column indices to stretch (0-based)	String or string resource. Use “*” for all columns. For example, “0,1,3,5”
android: layout_column	TableRow child view	Index of column this child view should be displayed in (0-based)	Integer or integer resource. For example, 1
android: layout_span	TableRow child view	Number of columns this child view should span across	Integer or integer resource greater than or equal to 1. For example, 3

Here’s an example of an XML layout resource with a TableLayout with two rows (two TableRow child objects).The TableLayout is set to stretch the columns to the size of the screen width.The first

TableRow has three columns; each cell has a Button object.The second TableRow puts only one Button view into the second column explicitly:

```
<TableLayout xmlns:android
"http://schemas.android.com/apk/res/android"
android:id "@+id/TableLayout01"
android:layout_width "fill_parent"
android:layout_height "fill_parent"
android:stretchColumns "*">
<TableRow
android:id "@+id/TableRow01">
<Button
android:id "@+id/ButtonLeft"
android:text "Left Door" />
<Button
android:id "@+id/ButtonMiddle"
android:text "Middle Door" />
<Button
android:id "@+id/ButtonRight"
android:text "Right Door" />
</TableRow>
<TableRow
android:id "@+id/TableRow02">
<Button
android:id "@+id/ButtonBack"
android:text "Go Back"
android:layout_column "1" />
</TableRow>
</TableLayout>
```

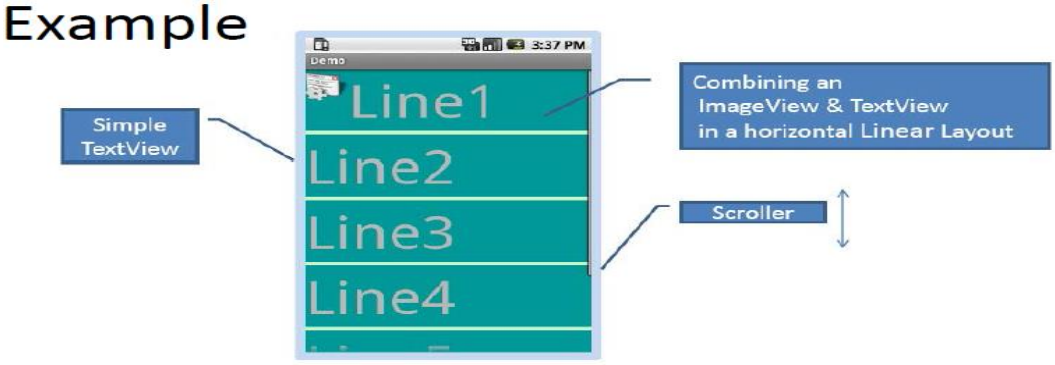
ScrollView Layout

When we have more data than what can be shown on a single screen you may use the ScrollView control.

It provides a scrolling access to the data. This way the user can only see part of your layout at one time, but the rest is available via scrolling.

This is similar to browsing a large web page that forces the user to scroll up the page to see the bottom part of the form.

Example



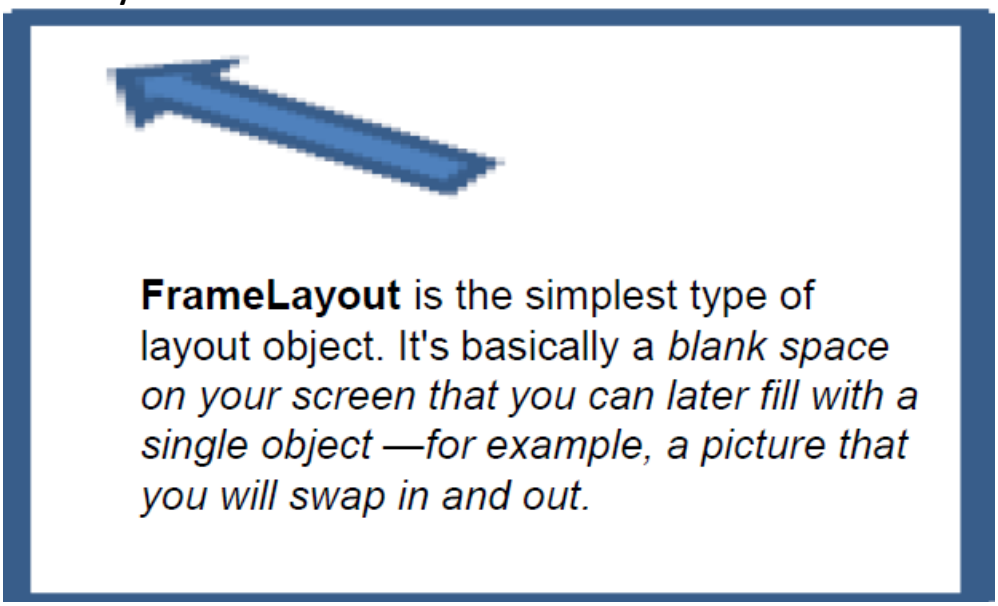
Absolute Layout

A layout that lest you specify exact locations (x/y coordinates) of its children.

Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.

```
<?xml version=”1.0” encoding=”utf-8”?>
<AbsoluteLayout
android:id=”@+id/myAbsoluteLayout”
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
xmlns:android=”http://schemas.android.
Com/apk/res/android”>

<Button android:id=”@+id/myButton”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android:text=”Button”
android:layout_x=”120px”
android:layout_y=”32px”>
</Button>
</AbsoluteLayout>
```

FrameLayout



1. Android's simplest layout manager is called:Frame Layout.
2. All child elements of the FrameLayout are pinned to the top left corner of the screen; you cannot specify a different location for a child view.
3. Adding multiple views to a frame layout just stacks one on top of the other (overlapping the views)
4. Subsequent child views will simply be drawn over previous ones, partially or totally obscuring them (unless the newer object is transparent).

Selection Widgets

- RadioButtons and CheckButtons are suitable for selecting from a small set of options.
- When the pool of choices is larger other widgets are more appropriate, those include classic UI controls such as: listboxes,comboboxes, drop-down lists, picture galleries, etc.
- Android offers a framework of data adapters that provide a common interface to selection lists ranging from static arrays to database contents.
- Selection views-widgets for presenting lists of choices –are handed an adapter to supply the actual choices.

ArrayAdapter

The easiet Adapter to use is ArrayAdapter-all you need to do is wrap one of these around a Java array or java.util.List instance, and you have a fully functioning adapter:

String[] items={"this", "is", "a", "really", "silly", "list"};

new ArrayAdapter<String> (this,android.R.layout.simple_list_item_1, items);

ArrayAdapter constructor takes three parameters:

1. The context to use (this will be your activity instance)
2. The resource ID of a view to use(such as the built-in system resource android:R.layout.simple_list_item_1 as shown above)
3. The actual (source) array or list of items to show

Instead of *Activity* we will use a *ListActivity* which is an Android class specializing in the use of ListViews.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView
        android:id="@+id/selection"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000cc"
        android:textStyle="bold" />
    <!-- Here is the list. Since we are using a ListActivity, we have to call it "@android:id/list" so ListActivity will find it -->
    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:drawSelectorOnTop="false" />

    <TextView android:id="@android:id/empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Empty set"  />
</LinearLayout>
```

Android's built-in list layout

Used on empty lists

```
import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class ArrayAdapterDemo extends ListActivity {
TextView selection;
String[] items={“this”, “is”, “a”, “really”, “really2”, “really3”, “really4”, “really5”, “silly”, “list”};
//next time try an empty list such as:
//String[] items={};
```

Note:

The ListActivity class is implicitly bound to an object identified by @android:id/list

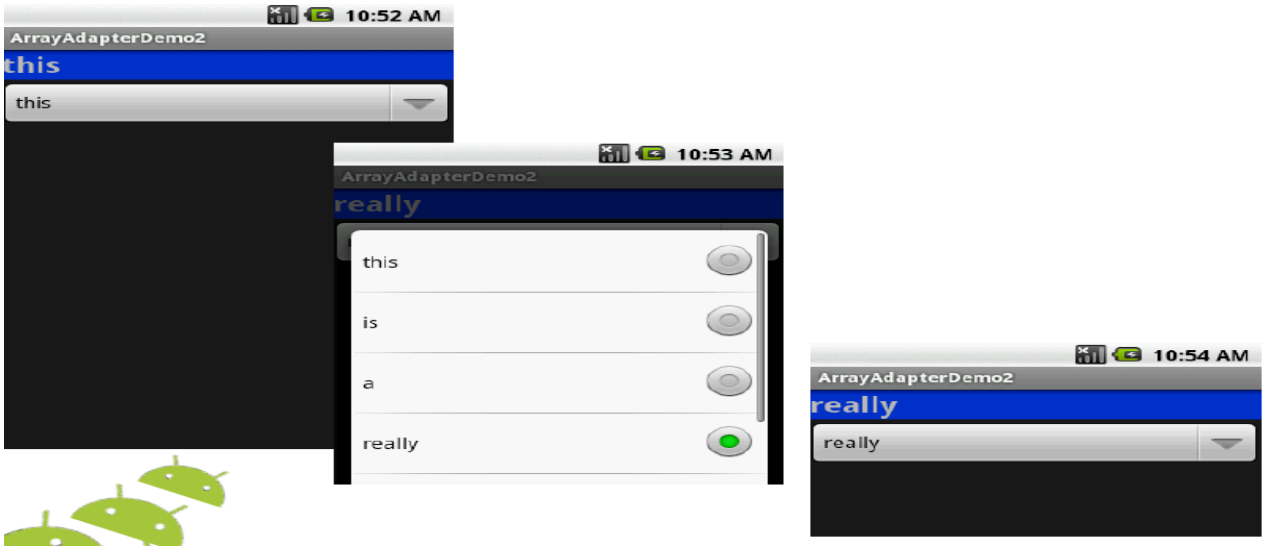
```
@override
public void onCreate ( Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        setListAdapter(new ArrayAdapter <String> (this, android.R.layout.simple_list_item_1,items));
        selection=(TextView) findviewById(R.id.selection);
}
@override
protected void onListItemClick(Listview 1, View v, int position, long id) {
super.onListItemClick(1, v, position, id);
String text=” position :”+ position +”   “+items [position];
Selection.setText(text);
}
}
```

List adapter

Selection listener

Spin Control

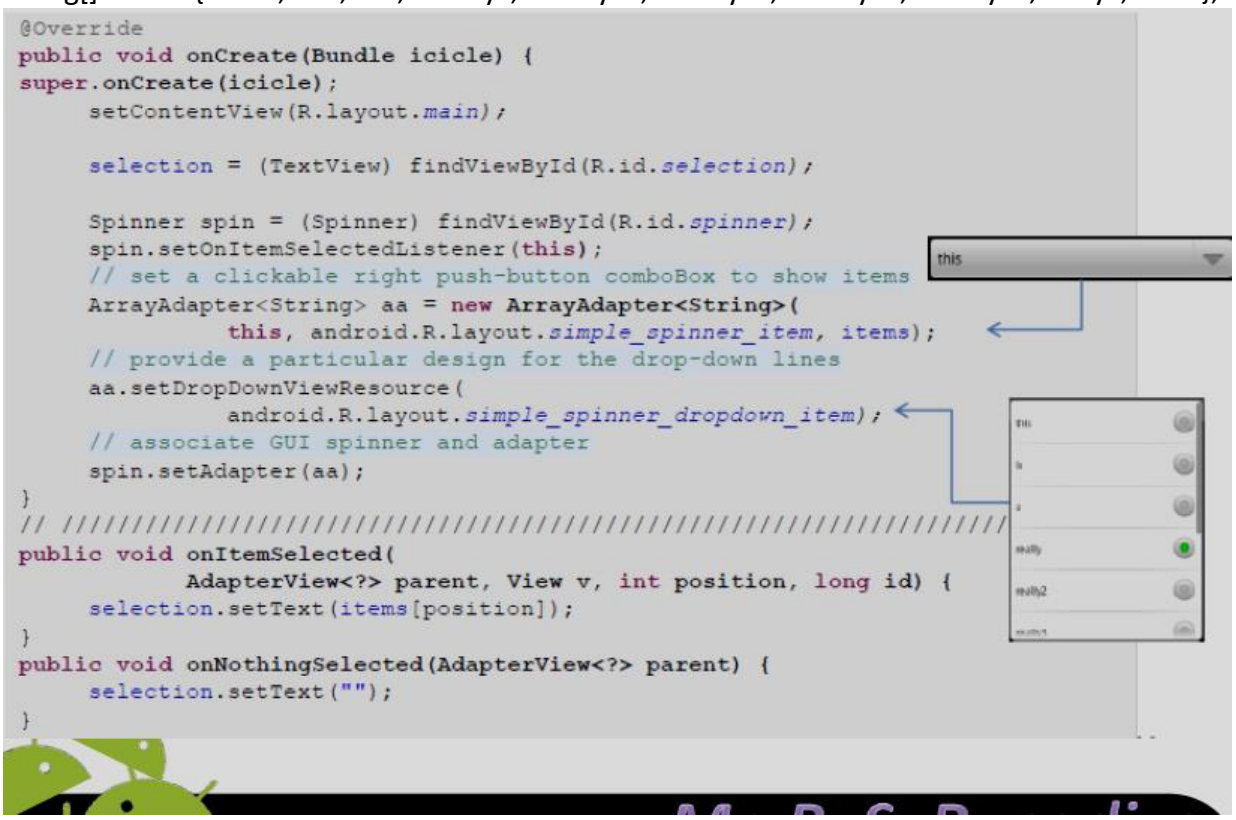
- In Android, the Spinner is the equivalent of the drop-down selector.
- Spinners have the same functionality of a ListView but take less space.
- As with ListView, you provide the adapter for linking data to child views using setAdapter()
- Add a listener object to capture selections made from the list with setOnItemSelectedListener().
- Use the setDropDownViewResource() method to supply the resource ID of the multi-line selection list view to use.




```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayout
android:id=”@+id/myLinearLayout”
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
android:orientation=”vertical”
xmlns:android=”http://schemas.android.com/apk/res/android>
<TextView
android:id=”@+id/selection”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:background=”#ff0033cc”
android:textSize=”14pt”
android:textStyle=”bold”>
</TextView>
<Spinner
android:id ”@+id/Spinner01”
android:layout_width= ”fill_parent”
android:layout_height ”wrap_content”>
</Spinner>
</LinearLayout>

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.AdapterView;
import android.widget.Spinner;
import android.widget.TextView;
public class ArrayAdapterDemo2 extends Activity
```

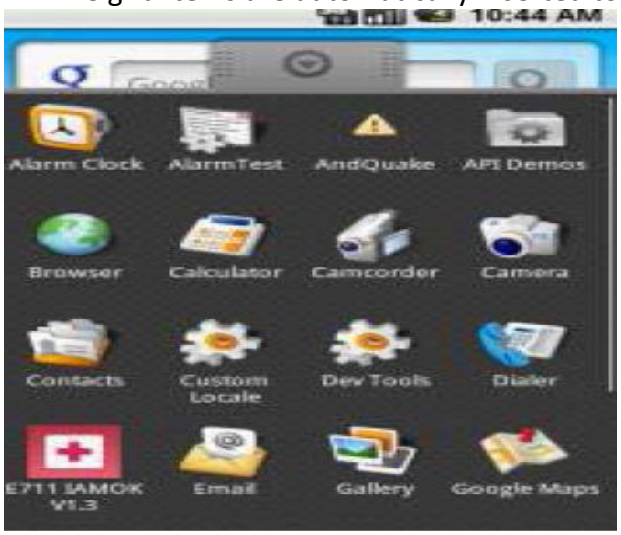
implements AdapterView.onItemSelectionListener{
TextView selection;
String[] items={“this”, “is”, “a”, “really”, “really2”, “really3”, “really4”, “really5”, “silly”, “list”};

```
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
     setContentView(R.layout.main);

     selection = (TextView) findViewById(R.id.selection);

     Spinner spin = (Spinner) findViewById(R.id.spinner);
     spin.setOnItemSelectedListener(this);
     // set a clickable right push-button comboBox to show items
     ArrayAdapter<String> aa = new ArrayAdapter<String>(
               this, android.R.layout.simple_spinner_item, items);
     // provide a particular design for the drop-down lines
     aa.setDropDownViewResource(
               android.R.layout.simple_spinner_dropdown_item);
     // associate GUI spinner and adapter
     spin.setAdapter(aa);
}
// ////////////////////////////////////////////////////////////////////
public void onItemSelected(
          AdapterView<?> parent, View v, int position, long id) {
     selection.setText(items[position]);
}
public void onNothingSelected(AdapterView<?> parent) {
     selection.setText("");
}
```



GridView

- GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.
- The grid items are automatically inserted to the layout using a ListAdapter.



GridView Properties

android:numColumns
how many columns there are or, if you supply a value of auto_fit, Android will compute the number of columns based on available space and the properties listed below.

android:verticalSpacing / android:horizontalSpacing
indicate how much whitespace there should be between items in the grid.

android:columnWidth
indicates how many pixels wide each column should be.

android:stretchMode
indicates, for grids with *auto_fit for android:numColumns, what should happen for any available space not taken up by columns or spacing .*

Drawing and working with animation

Working with Frame-by-Frame Animation

You can think of frame-by-frame animation as a digital flipbook in which a series of similar images display on the screen in a sequence, each subtly different from the last.When you display these images quickly, they give the illusion of movement.This technique is called frame-by-frame animation and is often used on the Web in the form of animated GIF images. Frame-by-frame animation is best used for complicated graphics transformations that are not easily implemented programmatically.
For example,we can create the illusion of a genie juggling gifts using a sequence of three images, as shown in Figure 9.13.

Figure 9.13 Three frames for an animation of a genie juggling.

In each frame, the genie remains fixed, but the gifts are repositioned slightly.The smoothness of the animation is controlled by providing an adequate number of frames and choosing the appropriate speed on which to swap them.

Working with Tweened Animations

With tweened animation, you can provide a single Drawable resource—it is a Bitmap graphic (see Figure 9.15, left), a ShapeDrawable, a TextView (see Figure 9.15, right), or any other type of View object—and the intermediate frames of the animation are rendered by the system.Android provides tweening support for several common image transformations, including alpha, rotate, scale, and translate animations.You can apply tweened animation transformations to any View, whether it is an ImageView with a Bitmap or shape Drawable, or a layout such as a TableLayout

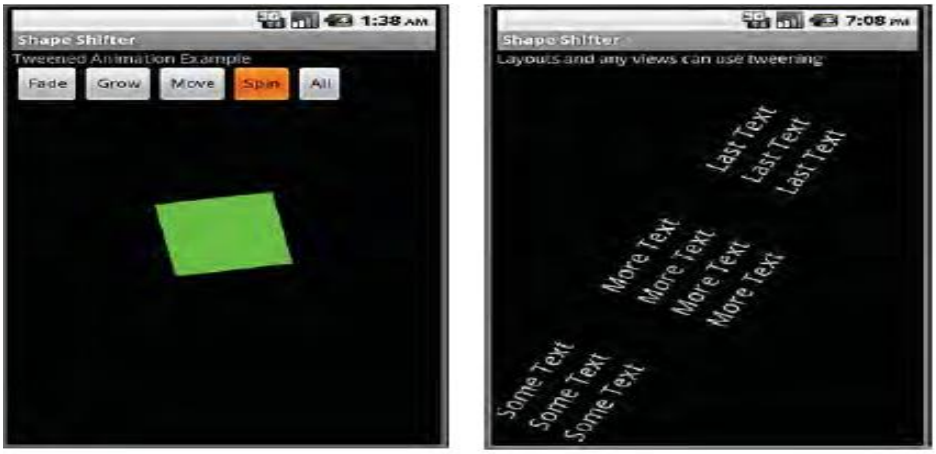


Figure 9.15 Rotating a green rectangle shape **drawable** (left) and a **TableLayout** (right).

Defining Tweened Animations as XML Resources

In Chapter 6,we showed you how to store animation sequences as specially formatted XML files within the /res/anim/ resource directory. For example, the following resource file called /res/anim/spin.xml describes a simple five-second rotation:

```
<?xml version ”1.0” encoding ”utf-8” ?>
<set xmlns:android
“http://schemas.android.com/apk/res/android”
android:shareInterpolator ”false”>
<rotate
android:fromDegrees ”0”
android:toDegrees ”360”
android:pivotX ”50%”
android:pivotY ”50%”
android:duration ”5000” />
</set>
```

Defining Tweened Animations Programmatically

You can programmatically define these animations.The different types of transformations are available as classes within the android.view.animation package. For example, you can define the aforementioned rotation animation as follows:

```
import android.view.animation.RotateAnimation;
...
RotateAnimation rotate new RotateAnimation(
0, 360, RotateAnimation.RELATIVE_TO_SELF, 0.5f,
RotateAnimation.RELATIVE_TO_SELF, 0.5f);
rotate.setDuration(5000);
```

UNIT-4
Database Connectivity Using SQLite and Content Provider

Data Storage

Android provides the following four mechanisms for storing and retrieving data:

1. Preferences, 2. Files, 3. Databases, and 4. Network

A typical desktop operating system provides a common file system that any application can use to store files that can be read by other applications Android uses a different system: On Android, all application data (including files) are private to that application.

Android also provides a standard way for an application to expose its private data to other applications — through

content providers.

Android supplies a number of content providers for standard data types, such as image, audio, video files and personal contact information.

Preferences

Preferences is an Android lightweight mechanism to store and retrieve key-value pairs of primitive data types also called Maps, and Associative Arrays.

Typically used to keep state information and shared data among several activities of an application. On each entry <key value> the key is a string and the value key-must be a primitive data type. Preferences are similar to Bundles however they are persistent while Bundles are not.

Using Preferences API calls

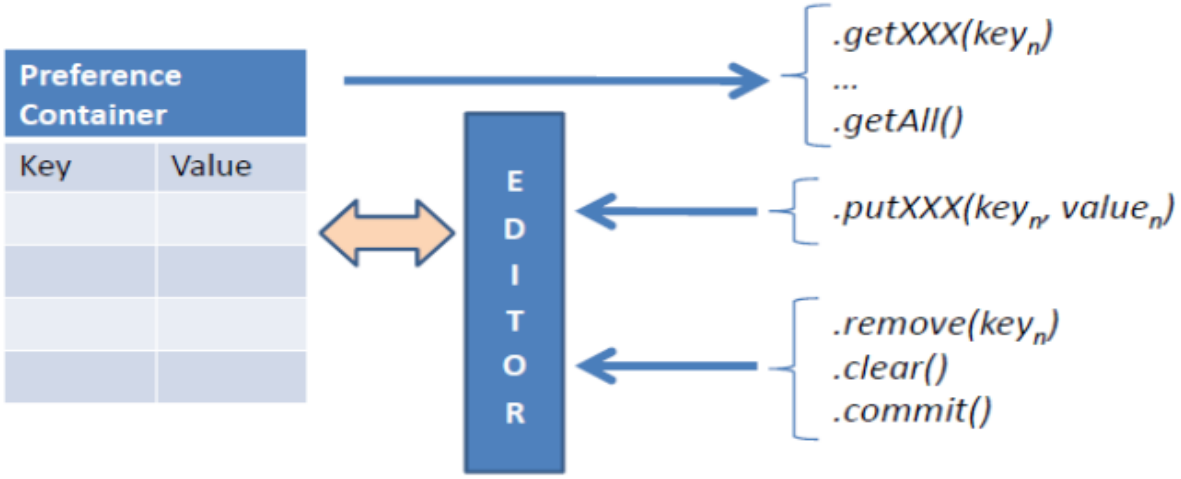
You have three API choices to pick a Preference:

1. **getPreferences()** from within your Activity, to access activity specific preferences
2. **getSharedPreferences()** from within your Activity to access application-level preferences
3. **getDefaultSharedPreferences(),** on PreferencesManager, to get the shared preferences that work in concert with Android's overall preference framework.

Using Preferences API calls

All of the get… Preference methods return a Preference object whose contents can be manipulated by an editor that allows putXXX… and getXXX… commands to place data in and out of the Preference container.

XXX = { Long, Int, Double, Boolean, String}



Searching and Reading Preferences

Reading preferences is straightforward. Simply retrieve the SharedPreferences instance you want to read.You can check for a preference by name, retrieve strongly typed preferences, and register to listen for changes to the preferences.Table below describes some helpful methods in the SharedPreferences interface.

Important android.content.SharedPreferences Methods

Method	Purpose
SharedPreferences.contains()	Sees whether a specific preference exists by name
SharedPreferences.edit()	Retrieves the editor to change these preferences
SharedPreferences.getAll()	Retrieves a map of all preference key/value pairs
SharedPreferences.getBoolean()	Retrieves a specific Boolean-type preference by name
SharedPreferences.getFloat()	Retrieves a specific Float-type preference by name
SharedPreferences.getInt()	Retrieves a specific Integer-type preference by name
SharedPreferences.getLong()	Retrieves a specific Long-type preference by name
SharedPreferences.getString()	Retrieves a specific String-type preference by name

Adding, Updating, and Deleting Preferences

To change preferences, you need to open the preference Editor, make your changes, and commit them.Table 10.2 describes some helpful methods in the SharedPreferences.Editor interface.

Table 10.2 **Important `android.content.SharedPreferences`.Editor Methods**

Method	Purpose
`SharedPreferences.Editor.clear()`	Removes all preferences. This operation happens first, regardless of when it is called within an editing session; then all other changes are made and committed.
`SharedPreferences.Editor.remove()`	Removes a specific preference by name. This operation happens first, regardless of when it is called within an editing session; then all other changes are made and committed.
`SharedPreferences.Editor.putBoolean()`	Sets a specific Boolean-type preference by name.
`SharedPreferences.Editor.putFloat()`	Sets a specific Float-type preference by name.
`SharedPreferences.Editor.putInt()`	Sets a specific Integer-type preference by name.

Working with Files and Directories

Remember from Chapter 1,"Introducing Android," that each Android application is its own user on the underlying Linux operating system. It has its own private application directory and files.Within the Android SDK, you can also find a variety of standard Java file utility classes (such as java.io) for handling different types of files, such as text files, binary files, and XML files.

In Managing Application Resources topic," you also learned that Android applications can also include static raw and XML files as resources.Although retrieving the file is handled slightly differently when accessing resources, the file can be read like any other file. Android application files are stored in a standard directory hierarchy on the Android file system.You can browse an application's directory structure using the DDMS File Explorer.

Exploring with the Android Application Directories

Android application data is stored on the Android file system in the following top-level directory:

/data/data/<package name>/

Several default subdirectories are created for storing databases, preferences, and files as necessary.You can also create other custom directories as needed. File operators all begin by interacting with the application Context object.Table 10.3 lists some important methods available for application file management.You can use all the standard java.io package utilities to work with FileStream objects and such.

Important android.content.Context File and Directory Management Methods

Method	Purpose
Context.openFileInput()	Opens an application file for reading. These files are located in the /files subdirectory
Context.openFileOutput()	Creates or opens an application file for writing. These files are located in the /files subdirectory.
Context.deleteFile()	Deletes an application file by name. These files must be located in the /files subdirectory
Context.fileList()	Gets a list of all files in the /files subdirectory
Context.getFilesDir()	Retrieves the application /files subdirectory object.

Context.getCacheDir()	Retrieves the application /cache subdirectory object.
Context.getDir()	Creates or retrieves an application subdirectory by name.

Creating and Writing to Files to the Default Application Directory

Android applications that require only the occasional file rely upon the helpful method called openFileOutput(). Use this method to create files in the default location under the application data directory: /data/data/<package name>/files/

For example, the following code snippet creates and opens a file called Filename.txt.We write a single line of text to the file and then close the file:

```
import java.io.FileOutputStream;
...
FileOutputStream fos;
String strFileContents “Some text to write to the file.”;
fos openFileOutput(“Filename.txt”, MODE_PRIVATE);
fos.write(strFileContents.getBytes());
fos.close();
```

We can append data to the file by opening it with the mode set to MODE_APPEND:

```
import java.io.FileOutputStream;
...
FileOutputStream fos;
String strFileContents “More text to write to the file.”;
fos openFileOutput(“Filename.txt”, MODE_APPEND);
fos.write(strFileContents.getBytes());
fos.close();
```

The file we created has the following path on the Android file system:
/data/data/<package name>/files/Filename.txt

Reading from Files in the Default Application Directory

Again we have a shortcut for reading files stored in the default /files subdirectory.The following code snippet opens a file called Filename.txt for read operations:

```
import java.io.FileInputStream;
...
String strFileName “Filename.txt”;
FileInputStream fis openFileInput(strFileName);
```

Reading Raw Files Byte-by-Byte

You handle file-reading and -writing operations using standard Java methods. Check out the subclasses of java.io.InputStream for reading bytes from different types of primitive file types. For example, DataInputStream is useful for reading one line at a time. Here’s a simple example of how to read a text file, line by line, and store it in a StringBuffer:

```
FileInputStream fis openFileInput(filename);
StringBuffer sBuffer new StringBuffer();
DataInputStream dataIO new DataInputStream(fis);
String strLine null;
while ((strLine dataIO.readLine()) ! null) {
sBuffer.append(strLine + “\n”);
```

}
dataIO.close();
fis.close();

Working with Other Directories and Files on the Android File System

Using Context.openFileOutput() and Context.openFileInput() are great if you have a few files and you want them stored in the /files subdirectory, but if you have more sophisticated file-management needs, you need to set up your own directory structure.To do this, you must interact with the Android file system using the standard java.io.File class methods.

The following code gets a File object for the /files application subdirectory and retrieves a list of all filenames in that directory:

```
import java.io.File;
...
File pathForAppFiles getFilesDir();
String[] fileList pathForAppFiles.list();
```

Here is a more generic method to create a file on the file system.This method works anywhere on the Android file system you have permission to access, not the /files directory:

```
import java.io.File;
import java.io.FileOutputStream;
...
File fileDir getFilesDir();
String strNewFileName “myFile.dat”;
String strFileContents “Some data for our file”;
File newFile new File(fileDir, strNewFileName);
newFile.createNewFile();
FileOutputStream fo
new FileOutputStream(newFile.getAbsolutePath());
fo.write(strFileContents.getBytes());
fo.close();
```

You can use File objects to manage files within a desired directory and create subdirectories. For example, you might want to store “track” files within “album” directories. Or perhaps you want to create a file in a directory other than the default.

Storing Structured Data Using SQLite Databases

For occasions when your application requires a more robust data storage mechanism, the Android file system includes support for application-specific relational databases using SQLite. SQLite databases are lightweight and file-based, making them ideally suited for embedded devices. These databases and the data within them are private to the application.To share application data with other applications, you must expose the data you want to share by making your application a content provider .

The Android SDK includes a number of useful SQLite database management classes. Many of these classes are found in the android.database.sqlite package. Here you can find utility classes for managing database creation and versioning, database management, and query builder helper classes to help you format proper SQL statements and queries. The package also includes specialized Cursor objects for iterating query results.You can also find all the specialized exceptions associated with SQLite. Here we focus on creating databases within our Android applications. For that,we use the built-in SQLite

support to programmatically create and use a SQLite database to store application information. However, if your application works with a different sort of database, you can also find more generic database classes (within the android.database package) to help you work with data from other providers. In addition to programmatically creating and using SQLite databases, developers can also interact directly with their application's database using the sqlite3 command-line tool that's accessible through the ADB shell interface.This can be an extremely helpful debugging tool for developers and quality assurance personnel, who might want to manage the database state (and content) for testing purposes.

Creating a SQLite Database

You can create a SQLite database for your Android application in several ways.To illustrate how to create and use a simple SQLite database, let's create an Android project called SimpleDatabase

Creating a SQLite Database Instance Using the Application Context

The simplest way to create a new SQLiteDatabase instance for your application is to use the openOrCreateDatabase() method of your application Context, like this:

```
import android.database.sqlite.SQLiteDatabase;
...
SQLiteDatabase mDatabase;
mDatabase openOrCreateDatabase("my_sqlite_database.db", SQLiteDatabase.CREATE_IF_NECESSARY, null);
```

Finding the Application's Database File on the Device File System

Android applications store their databases (SQLite or otherwise) in a special application directory:

/data/data/<application package name>/databases/<databasename>

So, in this case, the path to the database would be

/data/data/com.androidbook.SimpleDatabase/databases/my_sqlite_database.db

You can access your database using the sqlite3 command-line interface using this path

Configuring the SQLite Database Properties

Now that you have a valid SQLiteDatabase instance, it's time to configure it. Some important database configuration options include version, locale, and the thread-safe locking feature.

```
import java.util.Locale;
...
mDatabase.setLocale(Locale.getDefault());
mDatabase.setLockingEnabled(true);
mDatabase.setVersion(1);
```

Creating Tables and Other SQLite Schema Objects

Creating tables and other SQLite schema objects is as simple as forming proper SQLite statements and executing them.The following is a valid CREATE TABLE SQL statement. This statement creates a table called tbl_authors.The table has three fields: a unique id number, which auto-increments with each record and acts as our primary key, and firstname and lastname text fields:

```
CREATE TABLE tbl_authors (
id INTEGER PRIMARY KEY AUTOINCREMENT,firstname TEXT,lastname TEXT);
```

You can encapsulate this CREATE TABLE SQL statement in a static final String variable (called CREATE_AUTHOR_TABLE) and then execute it on your database using the execSQL() method:

```
mDatabase.execSQL(CREATE_AUTHOR_TABLE);
```

The execSQL() method works for nonqueries.You can use it to execute any valid SQLite SQL statement. For example, you can use it to create, update, and delete tables, views, triggers, and other common

SQL objects. In our application,we add another table called tbl_books.The schema for tbl_books looks like this:

```
CREATE TABLE tbl_books (
id INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT, dateadded DATE,authorid INTEGER NOT NULL
CONSTRAINT authorid REFERENCES tbl_authors(id) ON DELETECASCADE);
```

Unfortunately, SQLite does not enforce foreign key constraints. Instead,we must enforce them ourselves using custom SQL triggers. So we create triggers, such as this one that enforces that books have valid authors:

```
private static final String CREATE_TRIGGER_ADD
"CREATE TRIGGER fk_insert_book BEFORE INSERT ON tbl_books
FOR EACH ROW
BEGIN
SELECT RAISE(ROLLBACK, 'insert on table \"tbl_books\" violates foreign key constraint
\"fk_authorid\"') WHERE (SELECT id FROM tbl_authors WHERE id NEW.authorid) IS NULL;
END;";
```

We can then create the trigger simply by executing the CREATE TRIGGER SQL statement:

```
mDatabase.execSQL(CREATE_TRIGGER_ADD);
```

We need to add several more triggers to help enforce our link between the author and book tables, one for updating tbl_books and one for deleting records from tbl_authors.

Creating, Updating, and Deleting Database Records

Now that we have a database set up,we need to create some data.The SQLiteDatabase class includes three convenience methods to do that.They are, as you might expect, insert(), update(), and delete().

Inserting Records

We use the insert() method to add new data to our tables.We use the ContentValues object to pair the column names to the column values for the record we want to insert. For example, here we insert a record into tbl_authors for J.K. Rowling:

```
import android.content.ContentValues;
...
ContentValues values new ContentValues();
values.put("firstname", "J.K.");
values.put("lastname", "Rowling");
long newAuthorID mDatabase.insert("tbl_authors", null, values);
```

The insert() method returns the id of the newly created record.We use this author id to create book records for this author. You might want to create simple classes (that is, class Author and class Book) to encapsulate your application record data when it is used programmatically.

Updating Records

You can modify records in the database using the update() method.The update() method takes four arguments:

- The table to update records
- A ContentValues object with the modified fields to update
- An optional WHERE clause, in which ? identifies a WHERE clause argument
- An array of WHERE clause arguments, each of which is substituted in place of the ?'s from the second parameter.

Passing null to the WHERE clause modifies all records within the table, which can be useful for making sweeping changes to your database. Most of the time, we want to modify individual records by their unique identifier. The following function takes two parameters: an updated book title and a bookId.We find the record in the table called tbl_books that corresponds with the id and update that book's title. Again, we use the ContentValues object to bind our column names to our data values:

```
public void updateBookTitle(Integer bookId, String newtitle) {
ContentValues values new ContentValues();
values.put("title", newtitle);
mDatabase.update("tbl_books",values, "id ?", new String[] { bookId.toString() });
}
```

Because we are not updating the other fields,we do not need to include them in the ContentValues object.We include only the title field because it is the only field we change.

Deleting Records

You can remove records from the database using the remove() method.The remove() method takes three arguments:

- The table to delete the record from
- An optional WHERE clause, in which ? identifies a WHERE clause argument
- An array of WHERE clause arguments, each of which is substituted in place of the ?'s from the second parameter.

Passing null to the WHERE clause deletes all records within the table. For example, this function call deletes all records within the table called tbl_authors:

```
mDatabase.delete("tbl_authors", null, null);
```

Most of the time, though, we want to delete individual records by their unique identifiers. The following function takes a parameter bookId and deletes the record corresponding to that unique id (primary key) within the table called tbl_books:

```
public void deleteBook(Integer bookId) {
mDatabase.delete("tbl_books", "id ?",new String[] { bookId.toString() });
}
```

You need not use the primary key (id) to delete records; the WHERE clause is entirely up to you. For instance, the following function deletes all book records in the table tbl_books for a given author by the author's unique id:

```
public void deleteBooksByAuthor(Integer authorID) {
int numBooksDeleted mDatabase.delete("tbl_books", "authorid ?",new String[] { authorID.toString() });
}
```

Working with Transactions

Often you have multiple database operations you want to happen all together or not at all. You can use SQL Transactions to group operations together; if any of the operations fails, you can handle the error and either recover or roll back all operations. If the operations all succeed, you can then commit them. Here we have the basic structure for a transaction:

```
mDatabase.beginTransaction();
try {
// Insert some records, updated others, delete a few
// Do whatever you need to do as a unit, then commit it
mDatabase.setTransactionSuccessful();
} catch (Exception e) {
```

```
// Transaction failed. Failed! Do something here.
// It's up to you.
} finally {
mDatabase.endTransaction();
}
```

Now let's look at the transaction in a bit more detail.A transaction always begins with a call to beginTransaction() method and a try/catch block. If your operations are successful, you can commit your changes with a call to the setTransactionSuccessful() method. If you do not call this method, all your operations are rolled back and not committed. Finally, you end your transaction by calling endTransaction(). It's as simple as that. In some cases, you might recover from an exception and continue with the transaction. For example, if you have an exception for a read-only database, you can open the database and retry your operations. Finally, note that transactions can be nested, with the outer transaction either committing or rolling back all inner transactions.

Working with Cursors

When results are returned from a SQL query, you often access them using a Cursor found in the android.database.Cursor class. Cursor objects are rather like file pointers; they allow random access to query results. You can think of query results as a table, in which each row corresponds to a returned record.The Cursor object includes helpful methods for determining how many results were returned by the query the Cursor represents and methods for determining the column names (fields) for each returned record.The columns in the query results are defined by the query, not necessarily by the database columns.These might include calculated columns, column aliases, and composite columns. Cursor objects are generally kept around for a time. If you do something simple (such as get a count of records or in cases when you know you retrieved only a single simple record), you can execute your query and quickly extract what you need; don't forget to close the Cursor when you're done, as shown here:

```
// SIMPLE QUERY: select * from tbl_books
Cursor c mDatabase.query("tbl_books",null,null,null,null,null,null);
// Do something quick with the Cursor here...
c.close();
```

Executing Simple Queries

Your first stop for database queries should be the query() methods available in the SQLiteDatabase class.This method queries the database and returns any results as in a Cursor object The query() method we mainly use takes the following parameters:

- [String]:The name of the table to compile the query against
- [String Array]: List of specific column names to return (use null for all)
- [String] The WHERE clause: Use null for all; might include selection args as ?'s
- [String Array]:Any selection argument values to substitute in for the ?'s in the earlier parameter
- [String] GROUP BY clause: null for no grouping
- [String] HAVING clause: null unless GROUP BY clause requires one
- [String] ORDER BY clause: If null, default ordering used
- [String] LIMIT clause: If null, no limit.

Previously in the chapter,we called the query() method with only one parameter set to the table name.

```
Cursor c mDatabase.query("tbl_books",null,null,null,null,null,null);
```

This is equivalent to the SQL query

```
SELECT * FROM tbl_books;
```

Add a WHERE clause to your query, so you can retrieve one record at a time:
Cursor c mDatabase.query("tbl_books", null, "id ?", new String[]{"9"}, null, null, null);
This is equivalent to the SQL query
SELECT * tbl_books WHERE id 9;
Selecting all results might be fine for tiny databases, but it is not terribly efficient.You should always tailor your SQL queries to return only the results you require with no extraneous information included. Use the powerful language of SQL to do the heavy lifting for you whenever possible, instead of programmatically processing results yourself. For example, if you need only the titles of each book in the book table, you might use the following call to the query() method:
String asColumnsToReturn[] { "title", "id" };
String strSortOrder "title ASC";
Cursor c mDatabase.query("tbl_books", asColumnsToReturn, null, null, null, null, strSortOrder);
This is equivalent to the SQL query
SELECT title, id FROM tbl_books ORDER BY title ASC;

Closing and Deleting a SQLite Database

Although you should always close a database when you are not using it, you might on occasion also want to modify and delete tables and delete your database.

Deleting Tables and Other SQLite Objects

You delete tables and other SQLite objects in exactly the same way you create them. Format the appropriate SQLite statements and execute them. For example, to drop our tables and triggers,we can execute three SQL statements:
mDatabase.execSQL("DROP TABLE tbl_books;");
mDatabase.execSQL("DROP TABLE tbl_authors;");
mDatabase.execSQL("DROP TRIGGER IF EXISTS fk_insert_book;");

Closing a SQLite Database

You should close your database when you are not using it.You can close the database using the close() method of your SQLiteDatabase instance, like this: **mDatabase.close();**

Deleting a SQLite Database Instance Using the Application Context

The simplest way to delete a SQLiteDatabase is to use the deleteDatabase() method of your application Context.You delete databases by name and the deletion is permanent. You lose all data and schema information. **deleteDatabase("my_sqlite_database.db");**

Designing Persistent Databases

Generally speaking, an application creates a database and uses it for the rest of the application's lifetime—by which we mean until the application is uninstalled from the phone. So far,we've talked about the basics of creating a database, using it, and then deleting it. In reality, most mobile applications do not create a database on-the-fly, use them, and then delete them. Instead, they create a database the first time they need it and then use it. The Android SDK provides a helper class called SQLiteOpenHelper to help you manage your application's database. To create a SQLite database for your Android application using the SQLiteOpenHelper, you need to extend that class and then instantiate an instance of it as a member variable for use within your application.To illustrate how to do this, let's create a new Android project called PetTracker

Keeping Track of Database Field Names

You've probably realized by now that it is time to start organizing your database fields programmatically to avoid typos and such in your SQL queries. One easy way you do this is to make a class to encapsulate your database schema in a class, such as PetDatabase, shown here:

```
import android.provider.BaseColumns;
public final class PetDatabase {
private PetDatabase() {}
public static final class Pets implements BaseColumns {
private Pets() {}
public static final String PETS_TABLE_NAME ”table_pets”;
public static final String PET_NAME ”pet_name”;
public static final String PET_TYPE_ID ”pet_type_id”;
public static final String DEFAULT_SORT_ORDER ”pet_name ASC”;
}
public static final class PetType implements BaseColumns {
private PetType() {}
public static final String PETTYPE_TABLE_NAME ”table_pettypes”;
public static final String PET_TYPE_NAME ”pet_type”;
public static final String DEFAULT_SORT_ORDER ”pet_type ASC”;
}
}
```

By implementing the BaseColumns interface,we begin to set up the underpinnings for using database-friendly user interface controls in the future, which often require a specially named column called _id to function properly.We rely on this column as our primary key.

Extending the SQLiteOpenHelper Class

To extend the SQLiteOpenHelper class,we must implement several important methods, which help manage the database versioning.The methods to override are onCreate(), onUpgrade(), and onOpen().We use our newly defined PetDatabase class to generate appropriate SQL statements, as shown here:

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import com.androidbook.PetTracker.PetDatabase.PetType;
import com.androidbook.PetTracker.PetDatabase.Pets;
class PetTrackerDatabaseHelper extends SQLiteOpenHelper {
private static final String DATABASE_NAME “pet_tracker.db”;
private static final int DATABASE_VERSION 1;
PetTrackerDatabaseHelper(Context context) {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
@Override
public void onCreate(SQLiteDatabase db) {
db.execSQL(“CREATE TABLE “ +PetType.PETTYPE_TABLE_NAME+” (“
+ PetType._ID + “ INTEGER PRIMARY KEY AUTOINCREMENT ,”
+ PetType.PET_TYPE_NAME + “ TEXT”+ “);”);
db.execSQL(“CREATE TABLE “ + Pets.PETS_TABLE_NAME + “ (“
```

```
+ Pets._ID + " INTEGER PRIMARY KEY AUTOINCREMENT ,"+ Pets.PET_NAME + " TEXT,"
+ Pets.PET_TYPE_ID + " INTEGER" // FK to pet type table + ");");
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion){
// Housekeeping here.
// Implement how "move" your application data
// during an upgrade of schema versions
// Move or delete data as required. Your call.
}
@Override
public void onOpen(SQLiteDatabase db) {
super.onOpen(db);
}
}
```

Now we can create a member variable for our database like this:

```
PetTrackerDatabaseHelper mDatabase new
PetTrackerDatabaseHelper(this.getApplicationContext());
```

Now, whenever our application needs to interact with its database,we request a valid database object.We can request a read-only database or a database that we can also write to.We can also close the database. For example, here we get a database we can write data to:

```
SQLiteDatabase db mDatabase.getWritableDatabase();
```

Sharing Data Between Applications with Content Providers

Applications can access data within other applications on the Android system through content provider interfaces and expose internal application data to other applications by becoming a content provider. First,we take a look at some of the other content providers available on the Android platform and what you can do with them. Next, you see some examples of how to use content providers to improve the sample applications used in previous chapters. Finally, you learn how applications can become content providers to share information—for example, with LiveFolders.

Exploring Android's Content Providers

Android devices ship with a number of built-in applications, many of which expose their data as content providers.Your application can access content provider data from a variety of sources.You can find the content providers included with Android in the package android.provider.Table below lists some useful content providers in this package

Useful Built-In Content Providers

Provider	**Purpose**
MediaStore	Audio-visual data on the phone and external storage
CallLog	Sent and received calls
Browser	Browser history and bookmarks
ContactsContract	Phone contact database or phonebook
Settings	System-wide device settings and preferences
UserDictionary	A dictionary of user-defined words for use with predictive text input

Using the MediaStore Content Provider

You can use the MediaStore content provider to access media on the phone and on external storage devices.The primary types of media that you can access are audio, images, and video.You can access these different types of media through their respective content provider classes under android.provider.MediaStore. Most of the MediaStore classes allow full interaction with the data.You can retrieve, add, and delete media files from the device.There are also a handful of helper classes that define the most common data columns that can be requested. Table below lists some commonly used classes that you can find under android.provider.MediaStore.

Common MediaStore Classes

Class	Purpose
Video.Media	Manages video files on the device
Images.Media	Manages image files on the device
Images.ThumbNails	Retrieves thumbnails for the images
Audio.Media	Manages audio files on the device
Audio.Albums	Manages audio files organized by the album
Audio.Artists	Manages audio files by the artist who created them
Audio.Genres	Manages audio files belonging to a particular genre
Audio.Playlists	Manages audio files that are part of a particular playlist

The following code demonstrates how to request data from a content provider.A query is made to the MediaStore to retrieve the titles of all the audio files on the SD card of the handset and their respective durations.This code requires that you load some audio files onto the virtual SD card in the emulator.

```
String[] requestedColumns {
MediaStore.Audio.Media.TITLE,
MediaStore.Audio.Media.DURATION  };
Cursor cur managedQuery(
MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
requestedColumns, null, null, null);
Log.d(DEBUG_TAG, “Audio files: “ + cur.getCount());
Log.d(DEBUG_TAG, “Columns: “ + cur.getColumnCount());
String[] columns cur.getColumnNames();
int name cur.getColumnIndex(MediaStore.Audio.Media.TITLE);
int size cur.getColumnIndex(MediaStore.Audio.Media.DURATION);
cur.moveToFirst();
while (!cur.isAfterLast()) {
Log.d(DEBUG_TAG, “Title” + cur.getString(name));
Log.d(DEBUG_TAG, “Length: “ +
cur.getInt(size) / 1000 + “ seconds”);
cur.moveToNext();  }
```

The MediaStore.Audio.Media class has predefined strings for every data field (or column) exposed by the content provider.You can limit the audio file data fields requested as part of the query by defining a string array with the column names required. In this case, we limit the results to only the track title and the duration of each

audio file. We then use a managedQuery() method call.The first parameter is the predefined URI of the content provider you want to query (in most cases, the primary external storage is the SD card).The second parameter is the list of columns to return (audio file titles and
durations).The third and fourth parameters control any selection filtering arguments, and the fifth parameter provides a sort method for the results.We leave these null, as we want all audio files at this location. By using the managedQuery() method,we get a managed Cursor as a result.We then examine our Cursor for the results.

Using the CallLog Content Provider

Android provides a content provider to access the call log on the handset via the class android.provider.CallLog. At first glance, the CallLog might not seem to be a useful provider for developers, but it has some nifty features.You can use the CallLog to filter recently dialed calls, received, and missed calls.The date and duration of each call is logged and tied back to the Contact application for caller identification purposes. The CallLog is a useful content provider for customer relationship management (CRM) applications.The user can also tag specific phone numbers with custom labels within the Contact application. To demonstrate how the CallLog content provider works, let's look at a hypothetical situation where we want to generate a report of all calls to a number with the custom labeled HourlyClient123.Android allows for custom labels on these numbers, which we leverage for this example:

```
String[] requestedColumns {
CallLog.Calls.CACHED_NUMBER_LABEL,
CallLog.Calls.DURATION };
Cursor calls managedQuery(
CallLog.Calls.CONTENT_URI, requestedColumns,
CallLog.Calls.CACHED_NUMBER_LABEL
+ " ?", new String[] { "HourlyClient123" } , null);
Log.d(DEBUG_TAG, "Call count: " + calls.getCount());
int durIdx calls.getColumnIndex(CallLog.Calls.DURATION);
int totalDuration 0;
calls.moveToFirst();
while (!calls.isAfterLast()) {
Log.d(DEBUG_TAG, "Duration: " + calls.getInt(durIdx));
totalDuration + calls.getInt(durIdx);
calls.moveToNext(); }
```

Log.d(DEBUG_TAG, "HourlyClient123 Total Call Duration: " + totalDuration);

This code is similar to the code shown for the MediaStore audio files.Again,we start with listing our requested columns: the call label and the duration of the call.This time, however,we don't want to get every call in the log, only those with a label of HourlyClient123.To filter the results of the query to this specific label, it is necessary to specify the third and fourth parameters of the managedQuery() call.Together, these two parameters are equivalent to a database WHERE clause.The third parameter specifies the format of the WHERE clause with the column name with selection parameters (shown as ?s) for each selection argument value.The fourth parameter, the String array, provides the values to substitute for each of the selection arguments (?s) in order as you would do for a simple SQLite database query.

As before, the Activity manages the Cursor object lifecycle.We use the same method to iterate the records of the Cursor and add up all the call durations

Using the Browser Content Provider

Another useful, built-in content provider is the Browser.The Browser content provider exposes the user's browser site history and their bookmarked websites.You access this content provider via the android.provider.Browser class.As with the CallLog class, you can use the information provided by the Browser

content provider to generate statistics and to provide cross-application functionality.You might use the Browser content provider to add a bookmark for your application support website.
In this example,we query the Browser content provider to find the top five most frequently visited bookmarked sites.

```
String[] requestedColumns {
Browser.BookmarkColumns.TITLE,
Browser.BookmarkColumns.VISITS,
Browser.BookmarkColumns.BOOKMARK
};
Cursor faves managedQuery(Browser.BOOKMARKS_URI, requestedColumns,
Browser.BookmarkColumns.BOOKMARK + " 1", null,
Browser.BookmarkColumns.VISITS + " DESC limit 5");
Log.d(DEBUG_TAG, "Bookmarks count: " + faves.getCount());
int titleIdx faves.getColumnIndex(Browser.BookmarkColumns.TITLE);
int visitsIdx faves.getColumnIndex(Browser.BookmarkColumns.VISITS);
int bmIdx faves.getColumnIndex(Browser.BookmarkColumns.BOOKMARK);
faves.moveToFirst();
while (!faves.isAfterLast()) {
Log.d("SimpleBookmarks", faves.getString(titleIdx) + " visited "
+ faves.getInt(visitsIdx) + " times : "
+ (faves.getInt(bmIdx) ! 0 ? "true" : "false"));
faves.moveToNext(); }
```

Again, the requested columns are defined, the query is made, and the cursor iterates through the results. Note that the managedQuery() call has become substantially more complex. Let's take
a look at the parameters to this method in more detail.The first parameter, Browser.BOOKMARKS_URI, is a URI for all browser history, not only the Bookmarked items.The second parameter defines the requested columns for the query results.The third parameter specifies that the bookmark property must be true.This parameter is needed in order to filter within the query. Now the results are only browser history entries that have been bookmarked.The fourth parameter, selection arguments, is used only when replacement values are used, which is not used in this case, so the value is set to
null. Lastly, the fifth parameter specifies an order to the results (most visited in descending order). Retrieving browser history information requires setting the READ_HISTORY_BOOKMARKS permission.

Using the Contacts Content Provider

The Contacts database is one of the most commonly used applications on the mobile phone. People always want phone numbers handy for calling friends, family, coworkers, and clients.Additionally, most phones show the identity of the caller based on the contacts application, including nicknames, photos, or icons. Android provides a built-in Contact application, and the contact data is exposed to other Android applications using the content provider interface.As an application developer, this means you can leverage the user's contact data within your application for a more robust user experience.

Using the UserDictionary Content Provider

Another useful content provider is the UserDictionary provider.You can use this content provider for predictive text input on text fields and other user input mechanisms. Individual words stored in the dictionary are weighted by frequency and organized by locale.You can use the addWord() method within the UserDictionary.Words class to add words to the custom user dictionary.

Using the Settings Content Provider

Another useful content provider is the Settings provider.You can use this content provider to access the device settings and user preferences. Settings are organized much as they are in the Settings application—by category.You can find information about the Settings content provider in the **android.provider.Settings class**.

UNIT-5
Location Based Services (LBS),Common Android API,Notifications,Services,Deployment of applications

Introduction

A GPS receiver calculates its position by precisely timing the signals sent by GPS satellites high above the Earth. Each satellite continually transmits message that include.

- The time the message was transmitted and,
- Satellite position at time of message transmission.

How GPS Works?

The receiver uses the messages it receives to determine the transit time of each message and computes the distance to each satellite using the speed of light. Each of these distances and satellites locations defines a area. The receiver is on the surface of each of these areas when the distances and the satellites locations are correct. These distances and satellites locations are used to compute the location of the receiver using the navigation equations. This location is then displayed, perhaps with a moving map display or latitude and longitude.

Basic GPS measurements surrender only a position, and neither speed nor direction. However, most GPS units can automatically derive speed and direction of movement from two or more position measurements. The disadvantage of this principle is that changes in speed or direction can only be computed with a delay, and that derived direction becomes inaccurate when the distance travelled between two position measurements drops below or near the random error of position measurement. More advanced navigation systems use additional sensors like a compass or an inertial navigation system to complement GPS.

In typical GPS operation, four or more satellites must be visible to obtain an accurate result. The solution of the navigation equations gives the positions of the receiver along with difference between the time kept by the receiver's on-board clock and the true time-of-day, thereby eliminating the need for a more precise and possibly impractical receiver based clock. Applications for GPS such as time transfer, traffic signal timing, and synchronization of cell phone base station, make use of this cheap and highly accurate timing. Some GPS applications use this time for display, or, other than for the basic position calculations, do not use it at all.

Using Global Positioning Services (GPS)

The Android SDK provides means for accessing location via a built-in GPS hardware, when it's available. Generally speaking, just about every Android phone has some LBS capabilities. For example, in the United States, mobile phone location information is used by emergency services.That said, not all Android devices are phones, nor do all phones enable consumer-usage of LBS services. If GPS features are disabled, or an Android device does not have LBS hardware, the Android SDK provides additional APIs for determining alternate location providers.These other providers might have advantages and disadvantages in terms of power use, speed, and accuracy.

Working with GPS

LBS services and hardware such as a built-in precision GPS are optional features for Android devices. In addition to requiring the appropriate permissions, you can specify which optional features your application requires within the Android Manifest file.You can declare that your application uses or requires specific LBS services using the <usesfeature> tag of the Android Manifest file. Although this tag is not enforced by the Android operating system, it enables popular publication mechanisms such as the Android Market to filter your app and provide it only to users with appropriate devices. If your

application will only function well on devices with some sort of method for determining the current location, you could use the following <uses-feature> tag in your application's manifest file:

<uses-feature android:name "android.hardware.location" />

If your application requires a precise location fix (that is, the device has functional GPS hardware, not just cell tower triangulation or other such mechanisms), you could use the following <uses-feature> tag instead:

<uses-feature android:name "android.hardware.location.gps" />

GeoCoding Locations :

Geocoding is a process of assigning locations to addresses to that they can be placed as points on a map, similar to putting pins on a paper map, and analyzed with other spatial data. The process assigns geographic coordinates to the original data, hence the name geocoding.

The android platform API provides a feature that returns an estimated street addresses for latitude and longitude values.

Note: Address lookup requires a backend service that is not included in the core android framework. If this backend service is not available, Geocoder.getFromLocation() returns an empty list. The helper method isPresent(), available in API level 9 and later, checks to see if the backend service is available.

Define the Address Lookup Task:

The Geocoder object can be used without any special permissions.The following block of code demonstrates using the Geocoder object to get the location names of a Location object passed in to the onLocationChanged() method of a LocationListener:

```
Geocoder coder new Geocoder(this);
try {
Iterator<Address> addresses coder
.getFromLocation(location.getLatitude(),
location.getLongitude(), 3).iterator();
if (addresses ! null) {
while (addresses.hasNext()) {
Address namedLoc addresses.next();
String placeName namedLoc.getLocality();
String featureName namedLoc.getFeatureName();
String country namedLoc.getCountryName();
String road namedLoc.getThoroughfare();
locInfo + String.format("\n[%s][%s][%s][%s]",
placeName, featureName, road, country);
int addIdx namedLoc.getMaxAddressLineIndex();
while (addIdx > 0 ) {
String addLine namedLoc.getAddressLine(addIdx);
locInfo + String.
format("\nLine %d: %s", addIdx, addLine);
addIdx—;
}
}
}
} catch (IOException e) {
Log.e("GPS", "Failed to get address", e);
```

}

You can extract information from the results of the call to the getFromLocation() method in two ways, both of which are demonstrated. Note that a particular location might have multiple Address results in the form of a List<Address> object.Typically, the first Address is the most detailed, and the subsequent Address objects have less detail and describe a broader region.

The first method is to query for specific information, such as by using the getFeatureName() method or the getLocality() method.These methods are not guaranteed to return useful information for all locations.They are useful, though, when you know you need only a specific piece of general information, such as the country. The second method for querying information is by "address lines."This is generally used for displaying the "address" of a location to the user. It might also be useful to use the location in directions and in other cases where a street address is desired.That said, the addresses returned might not be complete. Simply use the getMaxAddressLineIndex() and getAddressLine() methods to iterate through the addresses. Figure below shows a sample location with three resulting addresses.

The Geocoder object also supports using named locations or address lines to generate latitude and longitude information.The input is forgiving and returns reasonable results in most cases. For instance, all the following returns valid and correct results:"Eiffel Tower," "London, UK,""Iceland,""BOS,""Yellowstone," and "1600 Pennsylvania Ave,DC."

The following code demonstrates a button handler for computing location data based on user input of this kind:

```
public void onClick(View v) {
String placeName name.getText().toString();
try {
List<Address> geocodeResults
coder.getFromLocationName(placeName, 3);
Iterator<Address> locations geocodeResults.iterator();
String locInfo "Results:\n";
while (locations.hasNext()) {
Address loc locations.next();
```

```
locInfo + String.format("Location: %f, %f\n",
loc.getLatitude(), loc.getLongitude());
}
results.setText(locInfo);
} catch (IOException e) {
Log.e("GeoAddress", "Failed to get location info", e);
}
}
```

The result of the call to the getFromLocationName() method is a List of Address objects,much like the previous example. Figure below shows the results for entering Eiffel Tower.

Always assume that you will get more than one result. It is good form to provide a picker for the user to select from the results and choose the most appropriate location. Another good way to confirm with the user that they entered the correct location is to map it.We now discuss a couple of different methods for mapping locations using Google Maps.

LOCATION AND MAPS

Location and maps-based apps offer a compelling experience on mobile devices. You can build these capabilities into your app using the classes of the android.location package and the Google Maps Android API. The sections below provide an introduction to how you can add the features.

- **Location Services:**

Android gives your applications access to the location services supported by the device through classes in the android.location package. The central component of the location framework is the

LocationManager system service, which provides APIs to determine location and behavior of the basic device (if available).
As with other system services, you do not instantiate a LocationManager directly , Rather you request an instance from the system by calling getSystemService (Context.LOCATION_SERVICE). The method returns a handle to a new LocationManager instance.
Once your application has a LocationManager, your application is able to do the three things:

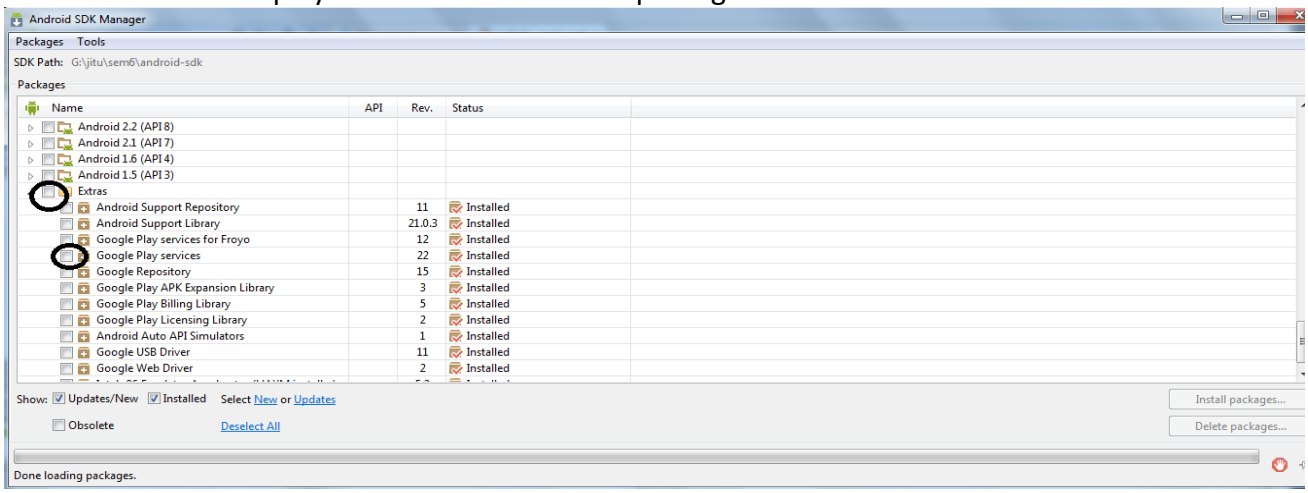
- → Query for the list of all LocationProviders for the last known user location.
- → Register/unregister for periodic updates of the user's current location from a location provider (specified either by criteria or name).
- → Register/unregister for a given Intent to be fired if the device comes withing a given closeness (specified by radius in metres) of a given latitude/longitude.

➢ **Google Maps Android API :**

With the Google Maps Android API, you can add maps to your app that are based on Google Maps data. The API automatically handles access to Google Maps servers, data downloading, map display, and touch gestures on the map. You can also use API calls to add markers, polygons and overlays, and to change the user's view of a particular map area.
The key class in the Google Maps Android API is MapView. A MapView displays a map with data obtained from the Google Maps service. When the MapView has focus, it will capture key presses and touch gestures to pan and zoom the map automatically, including handling network requests for additional maps tiles. It also provides all of the UI elements necessary for users to control the map. Your application can also use MapView class methods to control the map programmatically and draw a number of overlays on top of the map.
The Google Maps Android APIs are not included in the Android platform, but are available on any device with the Google play Store running Android 2.2 or higher, through Google Play Services.
To integrate Google Maps into your app, you need to install the Google Play services libraries for your Android SDK.

Download Google play service in your application.

Try to find out "**google-play-services_lib**" folder in your android-sdk-windows\extras\google.
If not be able to find it then download it. By opening Android Studio and clicking on Sdk Manager icon and check whether you have already downloaded Google Play Services or not under Extras section. If not select play services and install the package.



To make the Google Play services APIs available to your app:

1. Open the build.gradle file inside your application module directory.
   Note: Android Studio projects contain a top-level build.gradle file and a build.gradle file for each module. Be sure to edit the file for your application module. See Building Your Project with Gradle for more information about Gradle.
2. Add a new build rule under dependencies for the latest version of play-services. For example:

```
apply plugin: 'com.android.application'
...
dependencies {
  compile 'com.android.support:appcompat-v7:21.0.3'
  compile 'com.google.android.gms:play-services:6.5.87'
}
```

Be sure you update this version number each time Google Play services is updated.

Note: If the number of method references in your app exceeds the 65K limit, your app may fail to compile. You may be able to mitigate this problem when compiling your app by specifying only the specific Google Play services APIs your app uses, instead of all of them. For information on how to do this, see Selectively compiling APIs into your executable.

3. Save the changes and click Sync Project with Gradle Files in the toolbar.
4. Open your app's manifest file and add the following tag as a child of the <application> element:

```
<meta-data android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

You can now begin developing features with the Google Play services APIs

Many more with location based services :

Here we have seen number of different tools provided on android, using them we can find location. You can also find different tools or net.

The LocationManager supports proximity Alerts, which are alerts that trigger a PendingIntent when the handset comes within some distance of a location. This can be useful for warning the user of an upcoming turn in directions, for scanvenger hunts, or help in geocaching.

Common Android API

Android Networking API :

This class explains the basic tasks involved in connecting to the network, monitoring the network connection (including connection changes), and giving users control over an app's network usage. It also describes how to parse and consume XML data.

CONNECTING TO THE NETWORK

This lesson shows you how to implement a simple application that connects to the network. It explains some of the best practices you should follow in creating even the simplest network-connected app. Note that to perform the network operations described in this lesson, your application manifest must include the following permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- **Choose an HTTP Client :**

Most network-connected Android apps use HTTP to send and receive data. Android includes two HTTP clients: HttpURLConnection and Apache HttpClient. Both support HTTPS, streaming uploads and downloads, configurable timeouts, IPv6, and connection pooling. We recommend using HttpURLConnection for applications targeted at Gingerbread and higher.

- **Check the Network Connection :**

Before your app attempts to connect to the network, it should check to see whether a network connection is available using getActiveNetworkInfo() and isConnected(). Remember, the device may be out of range of a network, or the user may have disabled both Wi-Fi and mobile data access.

Let's consider a basic example to check if network is available or not.

[1] **Manifeast.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.networkusage"
   ...>
   <uses-sdk android:minSdkVersion="4"
        android:targetSdkVersion="14" />
    <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

   <application
      ...>
      ...
      <activity android:label="SettingsActivity" android:name=".SettingsActivity">
          <intent-filter>
            <action android:name="android.intent.action.MANAGE_NETWORK_USAGE" />
            <category android:name="android.intent.category.DEFAULT" />
       </intent-filter>
      </activity>
   </application>
</manifest>
```

[2] Activity_main.xml

```
<RelativeLayout
xmls : android=”http://schemas.android.com/apk/res/android”
xmlns : tools=”http://schemas.android.com/tools”
android : layout_width=”match_parent”
android : layout_height=”match_parent”
android : paddingBottom=”@dimen/activity_vertical_margin”
android : paddingLeft=”@dimen/activity_horizontal_margin”
android : paddingRight=”@dimen/activity_horizontal_margin”
android : paddingTop=”@dimen/activity_vertical_margin”
tools : context=”.MainActivity”>
<TextView
android : id=”@id/textView1”
android : layout_width=”wrap_content”
android : layout_height=”wrap_content”
android : layout_alignParentLeft=”true”
android : layout_alignParentTop=”true”
android : layout_marginLeft=”48dp”
android : layout_marginTop=”73dp”
android : text=”Large Text”
android : textApperance=”?android : attr/textApperanceLarge” />
</RelativeLayout>
```

[3] MainActivity.java

```
package="com.example.android.networkusage"
```

```
import com.example.android.networkusage.R.id;
import android.net.ConnnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.view.Menu;
import android.widget.TextView;
public class MainActivity extends Activity {
TextView tv;
@override
Protected void onCreate ( Bundle savedInstanceState) {
super.onCreate (savedInstanceState);
tv= (TextView) findViewById(id.textView1);
ConnectivityManager cm= (ConnectivityManager) getSystemService (Context.CONNECTIVITY_SERVICE);
NetworkInfo ni=cm.getActiveNetworkInfo();
If ( ni !=null && ni.isConnected())
{
        tv.setText("connected");
}

else
{
        tv.setText("Not connected");
}
}
}
```

Perform Network Operations on a Separate Thread :

Network operations can involve unpredictable delays. To prevent this from causing a poor user experience, always perform network operations on a separate thread from the UI. The AsyncTask class provides one of the simplest ways to fire off a new task from the UI thread.

In this section, the myClickHandler() method invokes new DownloadWebpageTask().execute(stringUrl). The DownloadWebpageTask class is a subclass of AsyncTask. DownloadWebpageTask implements the following AsyncTask methods:

- doInBackground() executes the method downloadUrl(). It passes the web page URL as a parameter. The method downloadUrl() fetches and processes the web page content. When it finishes, it passes back a result string.
- onPostExecute() takes the returned string and displays it in the UI.

Now lets consider an example that will display data from your webpage.

(1) Provide following uses permission to your application :

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

(2) Now make the activity_main.xml file like this.

```
<RelativeLayout
xmls : android="http://schemas.android.com/apk/res/android"
xmlns : tools="http://schemas.android.com/tools"
android : layout_width="match_parent"
android : layout_height="match_parent"
```

```
android : paddingBottom=”@dimen/activity_vertical_margin”
android : paddingLeft=”@dimen/activity_horizontal_margin”
android : paddingRight=”@dimen/activity_horizontal_margin”
android : paddingTop=”@dimen/activity_vertical_margin”
tools : context=”.MainActivity”>
<TextView
android : id=”@id/textView1”
android : layout_width=”wrap_content”
android : layout_height=”wrap_content”
android : layout_alignParentLeft=”true”
android : layout_alignParentTop=”true”
android : layout_marginLeft=”22dp”
android : layout_marginTop=”136dp”
android : text=”Medium Text”
android : textApperance=”?android : attr/textApperanceMedium” />
<Button
android : id=”@id/button1”
android : layout_width=”wrap_content”
android : layout_height=”wrap_content”
android : layout_alignLeft=”@+id/textView1”
android : layout_alignParentTop=”true”
android : layout_marginTop=”23dp”
android : onClick=”mybtnclick”
android : text=”Button”
</RelativeLayout>
```

(3) And finally apply following code to it. Well this code is in sections so that read carefully.

```
package "com.example.android.networkusage"
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.io.UnsupportedEncodingException;
import java.net.URL;
import "com.example.android.networkusage"
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Textview;
public class MainActivity extends Activity {
@Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
```

```
    urlText = (EditText) findViewById(R.id.myUrl);
    textView = (TextView) findViewById(R.id.myText);
  }
  // When user clicks button, calls AsyncTask.
  // Before attempting to fetch the URL, makes sure that there is a network connection.
  public void myClickHandler(View view) {
    // Gets the URL from the UI's text field.
    String stringUrl = urlText.getText().toString();
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
        new DownloadWebpageTask().execute(stringUrl);
    } else {
        textView.setText("No network connection available.");
    }
  }
   // Uses AsyncTask to create a task away from the main UI thread. This task takes a
   // URL string and uses it to create an HttpUrlConnection. Once the connection
   // has been established, the AsyncTask downloads the contents of the webpage as
   // an InputStream. Finally, the InputStream is converted into a string, which is
   // displayed in the UI by the AsyncTask's onPostExecute method.
   private class DownloadWebpageTask extends AsyncTask<String, Void, String> {
     @Override
     protected String doInBackground(String... urls) {
             // params comes from the execute() call: params[0] is the url.
        try {
            return downloadUrl(urls[0]);
        } catch (IOException e) {
            return "Unable to retrieve web page. URL may be invalid.";
        }
     }
     // onPostExecute displays the results of the AsyncTask.
     @Override
     protected void onPostExecute(String result) {
        textView.setText(result);
      }
    }
    ...
}
```

The sequence of events in this snippet is as follows:

1. When users click the button that invokes myClickHandler(), the app passes the specified URL to the AsyncTask subclass DownloadWebpageTask.
2. The AsyncTask method doInBackground() calls the downloadUrl() method.
3. The downloadUrl() method takes a URL string as a parameter and uses it to create a URL object.
4. The URL object is used to establish an HttpURLConnection.

5. Once the connection has been established, the HttpURLConnection object fetches the web page content as an InputStream.
6. The InputStream is passed to the readIt() method, which converts the stream to a string.
7. Finally, the AsyncTask's onPostExecute() method displays the string in the main activity's UI.

Connect and Download Data :

In your thread that performs your network transactions, you can use HttpURLConnection to perform a GET and download your data. After you call connect(), you can get an InputStream of the data by calling getInputStream().

In the following snippet, the doInBackground() method calls the method downloadUrl(). The downloadUrl() method takes the given URL and uses it to connect to the network via HttpURLConnection. Once a connection has been established, the app uses the method getInputStream() to retrieve the data as an InputStream.

```
// Given a URL, establishes an HttpUrlConnection and retrieves
// the web page content as a InputStream, which it returns as
// a string.
private String downloadUrl(String myurl) throws IOException {
   InputStream is = null;
   // Only display the first 500 characters of the retrieved
   // web page content.
   int len = 500;
    try {
      URL url = new URL(myurl);
      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
      conn.setReadTimeout(10000 /* milliseconds */);
      conn.setConnectTimeout(15000 /* milliseconds */);
      conn.setRequestMethod("GET");
      conn.setDoInput(true);
      // Starts the query
      conn.connect();
      int response = conn.getResponseCode();
      Log.d(DEBUG_TAG, "The response is: " + response);
      is = conn.getInputStream();
      // Convert the InputStream into a string
      String contentAsString = readIt(is, len);
      return contentAsString;
   // Makes sure that the InputStream is closed after the app is
   // finished using it.
   } finally {
      if (is != null) {
         is.close();
      }
   }
}
```

Note that the method getResponseCode() returns the connection's status code. This is a useful way of getting additional information about the connection. A status code of 200 indicates success.

Convert the InputStream to a string :

An InputStream is a readable source of bytes. Once you get an InputStream, it's common to decode or convert it into a target data type. For example, if you were downloading image data, you might decode and display it like this:

```
InputStream is = null;
...
Bitmap bitmap = BitmapFactory.decodeStream(is);
ImageView imageView = (ImageView) findViewById(R.id.image_view);
imageView.setImageBitmap(bitmap);
```

In the example shown above, the InputStream represents the text of a web page. This is how the example converts the InputStream to a string so that the activity can display it in the UI:

```
/ Reads an InputStream and converts it to a String.
public String readIt(InputStream stream, int len) throws IOException, UnsupportedEncodingException {
   Reader reader = null;
   reader = new InputStreamReader(stream, "UTF-8");
   char[] buffer = new char[len];
   reader.read(buffer);
   return new String(buffer);
}
```

MANAGING NETWORK USAGE

If your application performs a lot of network operations, you should provide user settings that allow users to control your app's data habits, such as how often your app syncs data, whether to perform uploads/downloads only when on Wi-Fi, whether to use data while roaming, and so on. With these controls available to them, users are much less likely to disable your app's access to background data when they approach their limits, because they can instead precisely control how much data your app uses.

- **Check a Device's Network Connection :**

A device can have various types of network connections. This lesson focuses on using either a Wi-Fi or a mobile network connection. Wi-Fi is typically faster. Also, mobile data is often metered, which can get expensive. A common strategy for apps is to only fetch large data if a Wi-Fi network is available; in case of mobile data even 3g is based on coverage and traffic of devices.

Before you perform network operations, it's good practice to check the state of network connectivity. Among other things, this could prevent your app from inadvertently using the wrong radio. If a network connection is unavailable, your application should respond gracefully. To check the network connection, you typically use the following classes:

- ConnectivityManager: Answers queries about the state of network connectivity. It also notifies applications when network connectivity changes.
- NetworkInfo: Describes the status of a network interface of a given type (currently either Mobile or Wi-Fi).

This code snippet tests network connectivity for Wi-Fi and mobile. It determines whether these network interfaces are available (that is, whether network connectivity is possible) and/or connected (that is, whether network connectivity exists and if it is possible to establish sockets and pass data):

```
private static final String DEBUG_TAG = "NetworkStatusExample";
...
ConnectivityManager connMgr = (ConnectivityManager)
      getSystemService(Context.CONNECTIVITY_SERVICE);
```

```
NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
boolean isWifiConn = networkInfo.isConnected();
networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
boolean isMobileConn = networkInfo.isConnected();
Log.d(DEBUG_TAG, "Wifi connected: " + isWifiConn);
Log.d(DEBUG_TAG, "Mobile connected: " + isMobileConn);
```

Note that you should not base decisions on whether a network is "available." You should always check isConnected() before performing network operations, since isConnected() handles cases like flaky mobile networks, airplane mode, and restricted background data.

A more concise way of checking whether a network interface is available is as follows. The method getActiveNetworkInfo() returns a NetworkInfo instance representing the first connected network interface it can find, or null if none of the interfaces is connected (meaning that an internet connection is not available): we already checked it in first example.

Manage Network Usage :

You can implement a preferences activity that gives users explicit control over your app's usage of network resources. For example:

- You might allow users to upload videos only when the device is connected to a Wi-Fi network.
- You might sync (or not) depending on specific criteria such as network availability, time interval, and so on.

To write an app that supports network access and managing network usage, your manifest must have the right permissions and intent filters.

- The manifest excerpted below includes the following permissions:
  - android.permission.INTERNET—Allows applications to open network sockets.
  - android.permission.ACCESS_NETWORK_STATE—Allows applications to access information about networks.

Now let’s consider an example in that we are going to provide settings facility in our application so that user can customize the things as they like and then try to store it up to next change. For that change your android manifests file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.android.networkusage"
  ...>

  <uses-sdk android:minSdkVersion="4"
      android:targetSdkVersion="14" />
      <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <application
    ...>
    ...
    <activity android:label="SettingsActivity" android:name=".SettingsActivity">
       <intent-filter>
         <action android:name="android.intent.action.MANAGE_NETWORK_USAGE" />
```

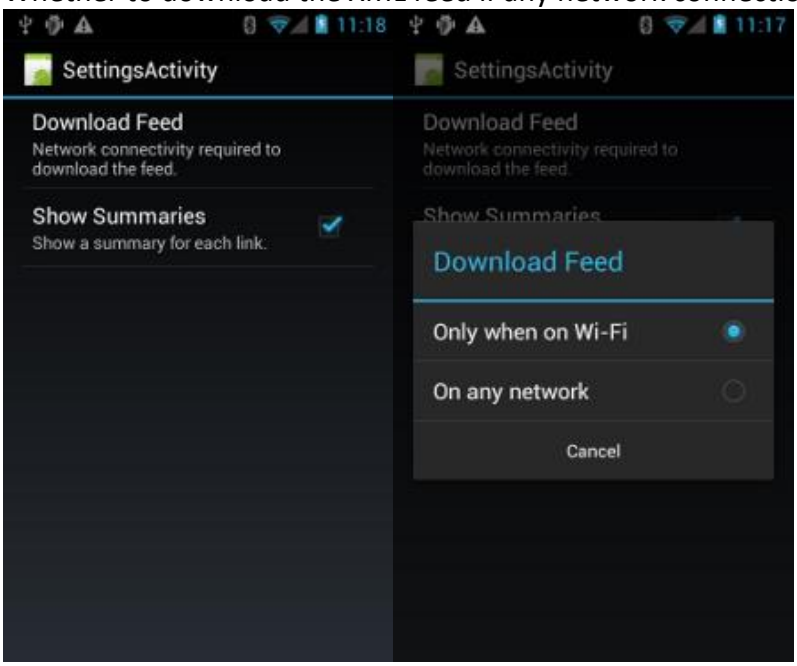
```
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
      </activity>
   </application>
</manifest>
```

Implement a Preferences Activity :

As you can see in the manifest excerpt above, the sample app's activity SettingsActivity has an intent filter for the ACTION_MANAGE_NETWORK_USAGE action. SettingsActivity is a subclass of PreferenceActivity. It displays a preferences screen (shown in figure 1) that lets users specify the following:

Whether to display summaries for each XML feed entry, or just a link for each entry.

Whether to download the XML feed if any network connection is available, or only if Wi-Fi is available.



Here is SettingsActivity. Note that it implements OnSharedPreferenceChangeListener. When a user changes a preference, it fires onSharedPreferenceChanged(), which sets refreshDisplay to true. This causes the display to refresh when the user returns to the main activity:

```
public class SettingsActivity extends PreferenceActivity implements
OnSharedPreferenceChangeListener {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
           // Loads the XML preferences file
      addPreferencesFromResource(R.xml.preferences);
   }
    @Override
   protected void onResume() {
      super.onResume();
      // Registers a listener whenever a key changes
      getPreferenceScreen().getSharedPreferences().registerOnSharedPreferenceChangeListener(this);
```

```
}
  @Override
protected void onPause() {
    super.onPause();
   // Unregisters the listener set in onResume().
   // It's best practice to unregister listeners when your app isn't using them to cut down on
   // unnecessary system overhead. You do this in onPause().
  getPreferenceScreen().getSharedPreferences().unregisterOnSharedPreferenceChangeListener(this);       }
  // When the user changes the preferences selection,
// onSharedPreferenceChanged() restarts the main activity as a new
// task. Sets the refreshDisplay flag to "true" to indicate that
// the main activity should update its display.
// The main activity queries the PreferenceManager to get the latest settings.
    @Override
public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {
    // Sets refreshDisplay to true so that when the user returns to the main
    // activity, the display refreshes to reflect the new settings.
    NetworkActivity.refreshDisplay = true;
} }
```

Android Web API

- **Web Apps:**

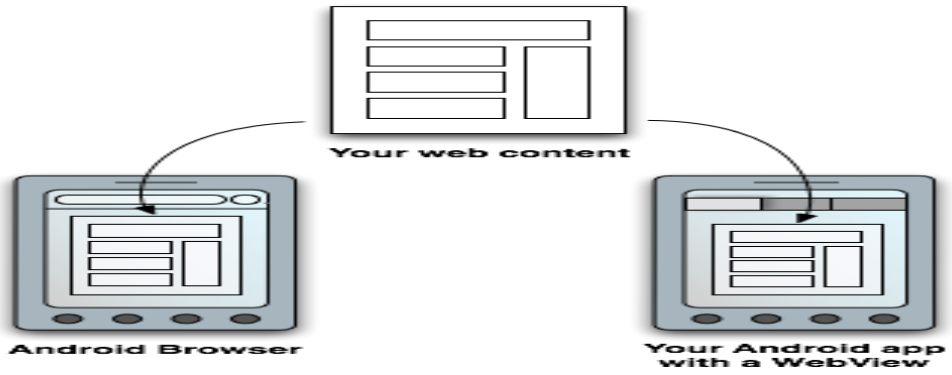


Figure: YOU CAN MAKE YOUR WEB CONTENT AVAILABLE TO USERS IN TWO WAYS: IN A TRADITIONAL WEB BROWSER AND IN AN ANDROID APPLICATION, BY INCLUDING A WEBVIEW IN THE LAYOUT.

There are essentially two ways to deliver an application on Android: as a client-side application (developed using the Android SDK and installed on user devices in an APK) or as a web application (developed using web standards and accessed through a web browser—there's nothing to install on user devices).

If you chose to provide a web-based app for Android-powered devices, you can rest assured that major web browsers for Android (and the WebView framework) allow you to specify viewport and style properties that make your web pages appear at the proper size and scale on all screen configurations.

Above figure illustrates how you can provide access to your web pages from either a web browser or your your own Android app. However, you shouldn't develop an Android app simply as a means to view your web site. Rather, the web pages you embed in your Android app should be designed especially for that environment. You can even define an interface between your Android application

and your web pages that allows JavaScript in the web pages to call upon APIs in your Android application—providing Android APIs to your web-based application.
To start developing web pages for Android-powered devices, see the following documents:

- **Building Web Apps in WebView**
  - How to embed web pages into your Android application using WebView and bind JavaScript to Android APIs.
- **Debugging Web Apps**
  - How to debug web apps using JavaScript Console APIs.
- **Best Practices for Web Apps**
  A list of practices you should follow, in order to provide an effective web application on Android-powered devices.

Building Web Apps In Webview

If you want to deliver a web application (or just a web page) as a part of a client application, you can do it using WebView. The WebView class is an extension of Android's View class that allows you to display web pages as a part of your activity layout. It does *not* include any features of a fully developed web browser, such as navigation controls or an address bar. All that WebView does, by default, is show a web page.
A common scenario in which using WebView is helpful is when you want to provide information in your application that you might need to update, such as an end-user agreement or a user guide. Within your Android application, you can create an Activity that contains a WebView, then use that to display your document that's hosted online.
Another scenario in which WebView can help is if your application provides data to the user that always requires an Internet connection to retrieve data, such as email. In this case, you might find that it's easier to build a WebView in your Android application that shows a web page with all the user data, rather than performing a network request, then parsing the data and rendering it in an Android layout. Instead, you can design a web page that's tailored for Android devices and then implement a WebView in your Android application that loads the web page.

- **Adding a WebView to Your Application :**

To add a WebView to your Application, simply include the <WebView> element in your activity layout. For example, here's a layout file in which the WebView fills the screen:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/webview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"/>
```

To load a web page in the WebView, use loadUrl(). For example:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

Before this will work, however, your application must have access to the Internet. To get Internet access, request the INTERNET permission in your manifest file. For example:

```
<manifest ... >
   <uses-permission android:name="android.permission.INTERNET" />
```

```
    ...
</manifest>
```

- **Using JavaScript in WebView:**

If the web page you plan to load in your WebView use JavaScript, you must enable JavaScript for your WebView. Once JavaScript is enabled, you can also create interfaces between your application code and your JavaScript code.

Enabling JavaScript:

JavaScript is disabled in a WebView by default. You can enable it through the WebSettings attached to your WebView. You can retrieve WebSettings with getSettings(), then enable JavaScript with setJavaScriptEnabled().

For example:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

- **Handling Page Navigation:**

When the user clicks a link from a web page in your WebView, the default behavior is for Android to launch an application that handles URLs. Usually, the default web browser opens and loads the destination URL. However, you can override this behavior for your WebView, so links open within your WebView. You can then allow the user to navigate backward and forward through their web page history that's maintained by your WebView.

To open links clicked by the user, simply provide a WebViewClient for your WebView, using setWebViewClient(). For example:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new WebViewClient());
```

That's it. Now all links the user clicks load in your WebView.

- → **Navigating web page History:**

When your WebView overrides URL loading, it automatically accumulates a history of visited web pages. You can navigate backward and forward through the history with goBack() and goForward(). For example, here's how your Activity can use the device *Back* button to navigate backward:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    // Check if the key event was the Back button and if there's history
    if ((keyCode == KeyEvent.KEYCODE_BACK) && myWebView.canGoBack()) {
        myWebView.goBack();
        return true;
    }
    // If it wasn't the Back key or there's no web page history, bubble up to the default
    // system behavior (probably exit the activity)
    return super.onKeyDown(keyCode, event);
}
```

The canGoBack() method returns true if there is actually web page history for the user to visit. Likewise, you can use canGoForward() to check whether there is a forward history. If you don't perform this check, then once the user reaches the end of the history, goBack() or goForward() does nothing.

- **Android telephony API**

As a software developer for mobile platforms, you may be interested in incorporating telephony features in your app. For instance, your interactive app might need to notify the user when it starts accessing the Internet while roaming or your high-bandwidth video chat application might need to alter its stream quality, based on the type of connection and its latency. These are situations where mobile apps need an interface to the basic phone features and its state, which is when the Telephony APIs come in. Here, we explore Android API support for GSM networks, and the important steps you need to take when embedding telephony support.
Telephony Manager provides access of information about the telephony services on the device. Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information. Applications can also register a listener to receive notification of telephony state changes.
You do not instantiate this class directly, instead , you retrieve a reference to an instance through Context.getSystemService(Context.TELEPHONY_SERVICE).
Permission Required: To work with Telephony Manager and to read the phone details we need to add this permission statement in AndroidMainfest file.
<uses-permission
android : name="android.permission.READ_PHONE_STATE"/>
permission. So add this permission in your manifest file.

- **Get Phone status:**

Using TelephonyManager we can retrieve the current status of mobile phone and also get some basic information about device.
Following statement will give you current telephony object using that your can interact.
TelephonyManager tm= (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
With a valid TelephonyManager instance, an application can now make several queries. One important method is getCallState().This method can determine the voice call status of the handset.The following block of code shows how to query for the call state and all the possible return values:

```
int callStatus telManager.getCallState();
String callState null;
switch (callStatus) {
case TelephonyManager.CALL_STATE_IDLE:
callState "Phone is idle.";
break;
case TelephonyManager.CALL_STATE_OFFHOOK:
callState "Phone is in use.";
break;
case TelephonyManager.CALL_STATE_RINGING:
callState "Phone is ringing!";
break;
}
Log.i("telephony", callState);
```

Querying for the call state can be useful in certain circumstances. However, listening for changes in the call state can enable an application to react appropriately to something the user might be doing. For instance, a game might automatically pause and save state information when the phone rings so that

the user can safely answer the call.An application can register to listen for changes in the call state by making a call to the listen() method of TelephonyManager.

```
telManager.listen(new PhoneStateListener() {
public void onCallStateChanged(
int state, String incomingNumber) {
String newState getCallStateString(state);
if (state TelephonyManager.CALL_STATE_RINGING) {
Log.i("telephony", newState +
" number " + incomingNumber);
} else {
Log.i("telephony", newState);
}
}
}, PhoneStateListener.LISTEN_CALL_STATE);
```

The listener is called, in this case, whenever the phone starts ringing, the user makes a call, the user answers a call, or a call is disconnected.The listener is also called right after it is assigned so an application can get the initial state.

- **Requesting Service Information:**

In addition to call and service state information, your application can retrieve other information about the device.This information is less useful for the typical application but can diagnose problems or provide specialized services available only from certain provider networks. The following code retrieves several pieces of service information:

```
String opName telManager.getNetworkOperatorName();
Log.i("telephony", "operator name " + opName);
String phoneNumber telManager.getLine1Number();
Log.i("telephony", "phone number " + phoneNumber);
String providerName telManager.getSimOperatorName();
Log.i("telephony", "provider name " + providerName);
```

The network operator name is the descriptive name of the current provider that the handset connects to.This is typically the current tower operator.The SIM operator name is typically the name of the provider that the user is subscribed to for service.The phone number for this application programming interface (API) is defined as the MSISDN, typically the directory number of a GSM handset (that is, the number someone would dial to reach that particular phone).

Now , consider the following different types of functions and services with which you can get different values.

- **Get the phone type CDMA/GSM/NONE :**

```
//get the type of network you are connected with
int phoneType=tm.getPhoneType();
switch (phoneType)
{
        case (TelephonyManager.PHONE_TYPE_CDMA);
              //your code
      break;
        case (TelephonyManager.PHONE_TYPE_GSM);
                  //your code
```

```
        break;
          case (TelephonyManager.PHONE_TYPE_NONE);
                    //your code
        break;
}
```

Find whether the Phone is in Roaming, returns true if in roaming:

```
Boolean isRoaming=tm.isNetworkRoaming();
if( isRoaming)
        phoneDetails+="\n Is in Roaming : "+"YES";
else
        phoneDetails+="\n Is in Roaming : "+"NO";
```

- **Get the SIM state/Details:**

Using the object of Telephony Manager class we can get the details like SIM serial number, Country Code, Network Provider code and other Details.

```
int SIMState=tm.getSimsState();
switch (SIMState)
{
        Case TelephonyManager.SIM_STATE_ABSENT :
                //YOUR CODE
                break;
        Case TelephonyManager.SIM_STATE_NETWORK_LOCKED :
                //YOUR CODE
                break;
        Case TelephonyManager.SIM_STATE_PIN_REQUIRED:
                //YOUR CODE
                break;
        Case TelephonyManager.SIM_STATE_PUK_REQUIRED :
                //YOUR CODE
                break;
        Case TelephonyManager.SIM_STATE_READY :
                //YOUR CODE
                break;
        Case TelephonyManager.SIM_STATE_UNKNOWN :
                //YOUR CODE
                break;          }
```

Notifications

Introduction

Applications often need to communicate with the user even when the application isn't actively running.Applications can alert users with text notifications, vibration, blinking lights, and even audio.

Notifying the User:

A notification is a user interface element that you display outside your app's normal UI to indicate that an event has occurred. Users can choose to view the notification while using other apps and respond to it when it's convenient for them.

Notifying with the Status Bar:

Notifying with the Status Bar

The standard location for displaying notifications and indicators on an Android device is the status bar that runs along the top of the screen.Typically, the status bar shows information such as the current date and time. It also displays notifications (like incoming SMS messages) as they arrive—in short form along the bar and in full if the user pulls down the status bar to see the notification list.The user can clear the notifications by pulling down the status bar and hitting the Clear button. Developers can enhance their applications by using notifications from their applications to inform the user of important events. For example, an application might want to send a simple notification to the user whenever new content has been downloaded.A simple notification has a number of important components:

- → An icon (appears on status bar and full notification)
- → Ticker text (appears on status bar)
- → Notification title text (appears in full notification)
- → Notification body text (appears in full notification)
- → An intent (launches if the user clicks on the full notification)

➢ **Building A Notification**

The example in this class are based on the NotifcationCompact.Builder class.
NotificationCompact.Builder is in the Support Library. You should use NotificationCompact and its subclasses, particularly NotificationCompact.Builder, to provide the best notification support for a wide range of platforms.
Create a Notification Builder:
When creating a notification, specify the UI content and actions with a NotificationCompact.Builder object. At bare minimum, a Builder object must include the following:

- A small icon, set by setSmallIcon()
- A title, set by setContentTitle()
- Detail text, set by setContentText()

For example:

```
NotificationCompact.Builder ncb=new NotificationCompact.Builder(this)
.setSmallIcon(R.drawable.ic_launcher)
.setContentTitle("from Jitu")
.setContentText("wel come and hi");
```

➢ **Define the Notification's Action:**

Although they're optional, you should add at least one action to your notification. An action allows users to go directly from the notification to an Activity in your application, where they can look at one or more events or do further work.
A notification can provide multiple actions. You should always define the action that's triggered when the user clicks the notification; usually this action opens an Activity in your application. You can also add buttons to the notification that perform additional actions such as snoozing an alarm or responding immediately to a text message; this feature is available as of Android 4.1. If you use additional action buttons, you must also make their functionality available in an Activity in your app;.
Inside a Notification, the action itself is defined by a PendingIntent containing an Intent that starts an Activity in your application.
How you construct the PendingIntent depends on what type of Activity you're starting. When you start an Activity from a notification, you must preserve the user's expected navigation experience. In the

snippet below, clicking the notification opens a new activity that effectively extends the behavior of the notification.

```
Intent inte=new Intent(this,secActivity.class);
//Because clicking the notification opens a new ("special") activity ,there's
//no need to create an artificial back stack.
pendingIntent pi= PendingIntent.getActivity(this, 0, inte, PendingIntent.FLAG_UPDATE_CURRENT);
```

- **Set the Notification's Click Behavior:**

To associate the PendingIntent created in the previous step with a gesture, call the appropriate method of NotificationCompact.Builder. For example, to start an activity when the user clicks the notification text in the notification drawer, and the PendingIntent by calling setContentIntent(). For example:

```
ncb.setContentIntent(pi);
```

Issue the Notification:

To issue the notification:

- → Get an instance of NotificationManager.
- → Use the notify() method to issue the notification. When you call notify(), specify a notification ID. You can use this ID to update the notification later on. This is described in more detail in Managing Notifications.
- → Call build(), which returns a Notification object containing your specifications.

For example:

```
NotificationCompat.Builder mBuilder;
...
// Sets an ID for the notification
int mNotificationId = 001;
// Gets an instance of the NotificationManager service
NotificationManager mNotifyMgr =
        (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
// Builds the notification and issues it.
mNotifyMgr.notify(mNotificationId, mBuilder.build());
```

- **Preserving Navigation when Starting an Activity:**

When you start an Activity from a notification, you must preserve the user's expected navigation experience. Clicking Back should take the user back through the application's normal work flow to the Home screen, and clicking Recents should show the Activity as a separate task. To preserve the navigation experience, you should start the Activity in a fresh task. How you set up the PendingIntent to give you a fresh task depends on the nature of the Activity you're starting. There are two general situations:

- → **Regular activity**
  You're starting an Activity that's part of the application's normal workflow. In this situation, set up the PendingIntent to start a fresh task, and provide the PendingIntent with a back stack that reproduces the application's normal Back behavior.
  Notifications from the Gmail app demonstrate this. When you click a notification for a single email message, you see the message itself. Touching Back takes you backwards through Gmail to the Home screen, just as if you had entered Gmail from the Home screen rather than entering it from a notification.
- → **Special activity**
  The user only sees this Activity if it's started from a notification. In a sense, the Activity extends the notification by providing information that would be hard to display in the notification itself. For this situation, set up the PendingIntent to start in a fresh task. There's no need to create a back stack, though, because the started Activity isn't part of the application's activity flow. Clicking Back will still take the user to the Home screen.

Let's check a simple example to raise notification as you program starts and by clicking on notification you will see another part of your program.
Create new project and name it "Stbar".
[1] keep your manifest file as it is. It does not require any changes.
[2] create two layout files one is by default as you create new project and add another layout file. To do so, right click on layout folder and select "new" and in that select "android XML file" give your xml file name, select layout option and it's done. So now we have two files.

activity_main.xml

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">
   <TextView
      android:text="@string/hello_world"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
</RelativeLayout>
```

sec_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
    android:orientation="vertical" >
   <TextView
      android:id="@+id/textView1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_weight="0.05"
      android:text="This page is loaded on activity click"
      android:textAppearance="?android:attr/textAppearanceLarge" />
   </LinearLayout>
```

Now let's code for its source files.

MainActivity.java

```
package com.example.admin.stbar;
import android.app.Notification;
import android.support.v4.app.NotificationCompat;
import  android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
```

```
import android.view.MenuItem;
import android.app.Activity;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      NotificationCompat.Builder ncb=new NotificationCompat.Builder(this)
      .setSmallIcon(R.drawable.ic_launcher)
      .setContentTitle("from Jitu")
       .setContentText("welcome  and Hello");
      Intent inte=new Intent(this, secActivity.class);
      PendingIntent pi=PendingIntent.getActivity(this,0,inte,PendingIntent.FLAG_UPDATE_CURRENT);
      ncb.setContent(pi);
      NotificationManager nm=(NotificationManager)getSystemService(NOTIFICATION_SERVICE);
      nm.notify(001,ncb.build());
   }
}
```

secActivity.java

```
package com.example.admin.stbar;
import android.app.Activity;
import android.os.Bundle;
public class secActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.sec_activity);
   }
}
```

It's done just execute your code.

Updating Notifications

When you need to issue a notification multiple times for the same type of event, you should avoid making a completely new notification. Instead, you should consider updating a previous notification, either by changing some of its values or by adding to it, or both.

Modify a Notification:

To set up a notification so it can be updated, issue it with a notification ID by calling NotificationManager.notify(ID, notification). To update this notification once you've issued it, update or create a NotificationCompat.Builder object, build a Notification object from it, and issue the Notification with the same ID you used previously.

The following snippet demonstrates a notification that is updated to reflect the number of events that have occurred. It stacks the notification, showing a summary:

```
mNotificationManager =
      (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
```

```
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
    // Because the ID remains unchanged, the existing notification is
    // updated.
    mNotificationManager.notify(
            notifyID,
            mNotifyBuilder.build());
```

Remove Notifications

Notifications remain visible until one of the following happens:

- The user dismisses the notification either individually or by using "Clear All" (if the notification can be cleared).
- The user touches the notification, and you called setAutoCancel() when you created the notification.
- You call cancel() for a specific notification ID. This method also deletes ongoing notifications.
- You call cancelAll(), which removes all of the notifications you previously issued.

- **Set Up the Notification to Launch a New Activity**

The sample application uses an IntentService subclass (PingService) to construct and issue the notification.

In this snippet, the IntentService method onHandleIntent() specifies the new activity that will be launched if the user clicks the notification itself. The method setContentIntent() defines a pending intent that should be fired when the user clicks the notification, thereby launching the activity.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
resultIntent.putExtra(CommonConstants.EXTRA_MESSAGE, msg);
resultIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
      Intent.FLAG_ACTIVITY_CLEAR_TASK);
// Because clicking the notification launches a new ("special") activity,
// there's no need to create an artificial back stack.
PendingIntent resultPendingIntent =  PendingIntent.getActivity( this, 0, resultIntent,
       PendingIntent.FLAG_UPDATE_CURRENT);
// This sets the pending intent that should be fired when the user clicks the
// notification. Clicking the notification launches a new activity.
builder.setContentIntent(resultPendingIntent);
```

- **Vibrating the phone:**

Vibration is a great way to enable notifications to catch the attention of a user in noisy environments or alert the user when visible and audible alerts are not appropriate (though a vibrating phone is often noisy on a hard desktop surface).Android notifications give a fine level of control over how vibration is performed. However, before the application can use vibration with a notification, an explicit permission is needed.The following XML within your application's AndroidManifest.xml file is required to use vibration:

<uses-permission android:name="android.permission.VIBRATE" />

Warning

The vibrate feature must be tested on the handset. The emulator does not indicate vibration in any way. Also, some Android devices do not support vibration.

Without this permission, the vibrate functionality will not work nor will there be any error.With this permission enabled, the application is free to vibrate the phone however it wants.This is accomplished by describing the vibrate member variable, which determines the vibration pattern.An array of long values describes the vibration duration. Thus, the following line of code enabled a simple vibration pattern that occurs whenever the notification is triggered:

notify.vibrate = new long[] {0, 200, 200, 600, 600};

This vibration pattern vibrates for 200 milliseconds and then stops vibrating for 200 milliseconds. After that, it vibrates for 600 milliseconds and then stops for that long.To repeat the Notification alert, a notification flag can be set so it doesn't stop until the user clears the notification.

notify.flags |= Notification.FLAG_INSISTENT;

An application can use different patterns of vibrations to alert the user to different types of events or even present counts. For instance, think about a grandfather clock with which you can deduce the time based on the tones that are played.

Blinking the lights:

Blinking lights are a great way to pass information silently to the user when other forms of alert are not appropriate.The Android SDK provides reasonable control over a multicolored indicator light, when such a light is available on the device. Users might recognize this light as a service indicator or battery level warning.An application can take advantage of this light as well, by changing the blinking rate or color of the light.

Warning:

Indicator lights are not available on all Android devices. Also, the emulator does not display the light's state. This mandates testing on actual hardware.

You must unplug the phone from the USB port for the colors to change. An application can use different colors and different blinking rates to indicate different information to the user. The blinking light continues until the Notification is cleared by the user. See the following piece of code to implement this.

```
NotificationCompat.Builder ncb=new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.ic_launcher)
        .setContentTitle("from Jitu")
        .setContentText("wel come and hello students")
        .setLights(Color.GREEN,1,1)
        .setNumber(++nummsg)
        .setAutoCancel(true);
```

Here you must set the 2nd and 3rd argument in order to blink the led. Otherwise it will not start blinking.

Customizing the Notification:

Although the default notification behavior in the expanded status bar tray is sufficient for most purposes, developers can customize how notifications are displayed if they so choose. To do so, developers can use the RemoteViews object to customize the look and feel of a notification.

The following code demonstrates how to create a RemoteViews object and assign custom text to it:

```
RemoteViews remote =new RemoteViews(getPackageName(), R.layout.remote);
remote.setTextViewText(R.id.text1, "Big text here!");
remote.setTextViewText(R.id.text2, "Red text down here!");
```

```
notify.contentView = remote;
```

To better understand this, here is the layout file remote.xml referenced by the preceding code:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:id="@+id/text1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="31dp"
android:textColor="#000" />
<TextView
android:id="@+id/text2"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18dp"
android:textColor="#f00" />
</LinearLayout>
```

This particular example is similar to the default notification but does not contain an icon.

Services

A Service is an application component that can perform long-running operations in the background and does not provide a user interface. Another application component can start a service and it will continue to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

A service can essentially take two forms:

- **Started:**
  A service is "started" when an application component (such as an activity) starts it by calling startService(). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.
- **Bound:**
  A service is "bound" when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Although this documentation generally discusses these two types of services separately, your service can work both ways—it can be started (to run indefinitely) and also allow binding. It's simply a matter

of whether you implement a couple callback methods: onStartCommand() to allow components to start it and onBind() to allow binding.
Regardless of whether your application is started, bound, or both, any application component can use the service (even from a separate application), in the same way that any component can use an activity—by starting it with an Intent. However, you can declare the service as private, in the manifest file, and block access from other applications.
Note: A service runs in the main thread of its hosting process—the service does **not** create its own thread and does **not** run in a separate process (unless you specify otherwise). This means that, if your service is going to do any CPU intensive work or blocking operations (such as MP3 playback or networking), you should create a new thread within the service to do that work. By using a separate thread, you will reduce the risk of Application Not Responding (ANR) errors and the application's main thread can remain dedicated to user interaction with your activities.

THE BASICS

- **Should you use a service or a thread?**

A service is simply a component that can run in the background even when the user is not interacting with your application. Thus, you should create a service only if that is what you need.
If you need to perform work outside your main thread, but only while the user is interacting with your application, then you should probably instead create a new thread and not a service. For example, if you want to play some music, but only while your activity is running, you might create a thread in onCreate(), start running it in onStart(), then stop it in onStop().
To create a service, you must create a subclass of Service (or one of its existing subclasses). In your implementation, you need to override some callback methods that handle key aspects of the service lifecycle and provide a mechanism for components to bind to the service, if appropriate. The most important callback methods you should override are:

- → **onStartCommand()**
  The system calls this method when another component, such as an activity, requests that the service be started, by calling startService(). Once this method executes, the service is started and can run in the background indefinitely. If you implement this, it is your responsibility to stop the service when its work is done, by calling stopSelf() or stopService(). (If you only want to provide binding, you don't need to implement this method.)
- → **onBind()**
  The system calls this method when another component wants to bind with the service (such as to perform RPC), by calling bindService(). In your implementation of this method, you must provide an interface that clients use to communicate with the service, by returning an IBinder. You must always implement this method, but if you don't want to allow binding, then you should return null.
- → **onCreate()**
  - **The** system calls this method when the service is first created, to perform one-time setup procedures (before it calls either onStartCommand() or onBind()). If the service is already running, this method is not called.
- → **onDestroy()**
  The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc. This is the last call the service receives.

If a component starts the service by calling startService() (which results in a call to onStartCommand()), then the service remains running until it stops itself with stopSelf() or another component stops it by calling stopService().
If a component calls bindService() to create the service (and onStartCommand() is *not* called), then the service runs only as long as the component is bound to it. Once the service is unbound from all clients, the system destroys it.
The Android system will force-stop a service only when memory is low and it must recover system resources for the activity that has user focus. If the service is bound to an activity that has user focus, then it's less likely to be killed, and if the service is declared to run in the foreground (discussed later), then it will almost never be killed. Otherwise, if the service was started and is long-running, then the system will lower its position in the list of background tasks over time and the service will become highly susceptible to killing—if your service is started, then you must design it to gracefully handle restarts by the system. If the system kills your service, it restarts it as soon as resources become available again.

- **Declaring a service in the manifest**

Like activities (and other components), you must declare all services in your application's manifest file.
To declare your service, add a <service> element as a child of the <application> element. For example:

```
<manifest ... >
  ...
  <application ... >
    <service android:name=".ExampleService" />
    ...
  </application>
</manifest>
```

There are other attributes you can include in the <service> element to define properties such as permissions required to start the service and the process in which the service should run. The android:name attribute is the only required attribute—it specifies the class name of the service. Once you publish your application, you should not change this name, because if you do, you risk breaking code due to dependence on explicit intents to start or bind the service.
Additionally, you can ensure that your service is available to only your app by including the android:exported attribute and setting it to "false". This effectively stops other apps from starting your service, even when using an explicit intent.

Creating a Started Service

A started service is one that another component starts by calling startService(), resulting in a call to the service's onStartCommand() method.
When a service is started, it has a lifecycle that's independent of the component that started it and the service can run in the background indefinitely, even if the component that started it is destroyed. As such, the service should stop itself when its job is done by calling stopSelf(), or another component can stop it by calling stopService().
An application component such as an activity can start the service by calling startService() and passing an Intent that specifies the service and includes any data for the service to use. The service receives this Intent in the onStartCommand() method.
For instance, suppose an activity needs to save some data to an online database. The activity can start a companion service and deliver it the data to save by passing an intent to startService(). The service

receives the intent in onStartCommand(), connects to the Internet and performs the database transaction. When the transaction is done, the service stops itself and it is destroyed.
Note: A services runs in the same process as the application in which it is declared and in the main thread of that application, by default. So, if your service performs intensive or blocking operations while the user interacts with an activity from the same application, the service will slow down activity performance. To avoid impacting application performance, you should start a new thread inside the service.

Traditionally, there are two classes you can extend to create a started service:

- **Service**
  This is the base class for all services. When you extend this class, it's important that you create a new thread in which to do all the service's work, because the service uses your application's main thread, by default, which could slow the performance of any activity your application is running.
- **IntentService**
  This is a subclass of Service that uses a worker thread to handle all start requests, one at a time. This is the best option if you don't require that your service handle multiple requests simultaneously. All you need to do is implement onHandleIntent(), which receives the intent for each start request so you can do the background work.
  Notice that onStartCommand() method must return an integer. The integer is a value that describes how the system should continue the service in the event that the system kills it. The return value from onStartCommand() must be one of the following constants:

→ **START_NOT_STICKY**
If the system kills the service after onStartCommand() returns, do not recreate the service, unless there are pending intents to deliver. This is the safest option to avoid running your service when not necessary and when your application can simply restart any unfinished jobs.

→ **START_STICKY:**
If the system kilss the service after onStartCommand() returns, recreate the service and call onStartCommand(), but do not redeliver the last intent. Instead the system calls onStartCommand() with a null intent, unless there were pending intents to start the service, in which case, those intents are delivered. This is suitable for media players (or similar services) that are not executing commands, but running indefinitely and waiting for a job.

→ **START_REDELIVER_INTENT:**
If the system kills the service after onStartCommand() returns, recreate the service and call onStartCommand() with the last intent that was delivered to the service. Any pending intents are delivered in turn. This is suitable for services that are avtively performing a job that should be immediately resumed, such as downloading a file.

- **Starting a service:**

You can start a service from an activity or other application component by passing an Intent (specifying the service to start) to startService(). The Android system calls the service's onStartCommand() method and passes it the Intent. (You should never call onStartCommand() directly.)
For example, an activity can start the example service in the previous section (HelloSevice) using an explicit intent with startService():

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

The startService() method returns immediately and the Android system calls the service's onStartCommand() method. If the service is not already running, the system first calls onCreate(), then calls onStartCommand().
If the service does not also provide binding, the intent delivered with startService() is the only mode of communication between the application component and the service. However, if you want the service to send a result back, then the client that starts the service can create a PendingIntent for a broadcast (with getBroadcast()) and deliver it to the service in the Intent that starts the service. The service can then use the broadcast to deliver a result.
Multiple requests to start the service result in multiple corresponding calls to the service's onStartCommand(). However, only one request to stop the service (with stopSelf() or stopService()) is required to stop it.

- **Stopping a service**

A started service must manage its own lifecycle. That is, the system does not stop or destroy the service unless it must recover system memory and the service continues to run after onStartCommand() returns. So, the service must stop itself by calling stopSelf() or another component can stop it by calling stopService().
Once requested to stop with stopSelf() or stopService(), the system destroys the service as soon as possible.
However, if your service handles multiple requests to onStartCommand() concurrently, then you shouldn't stop the service when you're done processing a start request, because you might have since received a new start request (stopping at the end of the first request would terminate the second one). To avoid this problem, you can use stopSelf(int) to ensure that your request to stop the service is always based on the most recent start request. That is, when you call stopSelf(int), you pass the ID of the start request (the startId delivered to onStartCommand()) to which your stop request corresponds. Then if the service received a new start request before you were able to call stopSelf(int), then the ID will not match and the service will not stop.
Note: It's important that your application stops its services when it's done working, to avoid wasting system resources and consuming battery power.
Now let’s check a simple example for service.
Create a new project and name it Simpserv.

[1] AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.admin.simperv" >
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.admin.simperv.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
```

```
    <service android:name="MyServices"></service>
  </application>
</manifest>
```

[2] activity_main.xml

```
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin"
  tools:context=".MainActivity">
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Start Service"
    android:id="@+id/button1"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:onClick="startNewService" />
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Stop Service"
    android:id="@+id/button2"
    android:layout_below="@+id/button1"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="51dp"
    android:onClick="stopNewService" />
</RelativeLayout>
```

[3] MainActivity.java

```
package com.example.admin.simperv;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.content.Intent;
import android.app.Activity;
public class MainActivity extends Activity {
  @Override
```

```
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

   }
//start the service
   public void startNewService(View v) {
      startService(new Intent(this,MyServices.class));
   }
//stop the service
   public void stopNewService(View v) {
      stopService(new Intent(this,MyServices.class));
   }
   @Override
     public boolean onCreateOptionsMenu(Menu menu) {
      //inflate the menu; this adds items to the action bar if it is present.
      getMenuInflater().inflate(R.menu.menu_main,menu);
           return true;
     }
}
```

Myservices.java // this file is used to implement services.

```
package com.example.admin.simperv;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;
public class MyServices extends Service{
   public MyServices() {

   }
    @Override
   public IBinder onBind(Intent intent) {
       //todo: return the communication channel to the servicee.
       throw new UnsupportedOperationException("Not yet  implemented");
    }
   @Override
   public void onCreate() {
      Toast.makeText(this,"the new service was created",Toast.LENGTH_LONG).show();
   }
   @Override
   public void onStart(Intent intent, int startId) {
      // for time consuming an long tasks you can launch a new thread here....
      Toast.makeText(this,"Service started",Toast.LENGTH_LONG).show();
   }
   @Override
```

```
    public void onDestroy() {
        Toast.makeText(this,"service destroyed",Toast.LENGTH_LONG).show();
    }
}
```

- **Creating a Bound Service**

A bound service is one that allows application components to bind to it by calling bindService() in order to create a long-standing connection (and generally does not allow components to start it by calling startService()).

You should create a bound service when you want to interact with the service from activities and other components in your application or to expose some of your application's functionality to other applications, through interprocess communication (IPC).

To create a bound service, you must implement the onBind() callback method to return an IBinder that defines the interface for communication with the service. Other application components can then call bindService() to retrieve the interface and begin calling methods on the service. The service lives only to serve the application component that is bound to it, so when there are no components bound to the service, the system destroys it (you do not need to stop a bound service in the way you must when the service is started through onStartCommand()).

To create a bound service, the first thing you must do is define the interface that specifies how a client can communicate with the service. This interface between the service and a client must be an implementation of IBinder and is what your service must return from the onBind() callback method. Once the client receives the IBinder, it can begin interacting with the service through that interface. Multiple clients can bind to the service at once. When a client is done interacting with the service, it calls unbindService() to unbind. Once there are no clients bound to the service, the system destroys the service.

- **Managing the Lifecycle of a Service**

The lifecycle of a service is much simpler than that of an activity. However, it's even more important that you pay close attention to how your service is created and destroyed, because a service can run in the background without the user being aware.

The service lifecycle—from when it's created to when it's destroyed—can follow two different paths:

- A started service
  The service is created when another component calls startService(). The service then runs indefinitely and must stop itself by calling stopSelf(). Another component can also stop the service by calling stopService(). When the service is stopped, the system destroys it..
- A bound service
  The service is created when another component (a client) calls bindService(). The client then communicates with the service through an IBinder interface. The client can close the connection by calling unbindService(). Multiple clients can bind to the same service and when all of them unbind, the system destroys the service. (The service does not need to stop itself.)

These two paths are not entirely separate. That is, you can bind to a service that was already started with startService(). For example, a background music service could be started by calling startService() with an Intent that identifies the music to play. Later, possibly when the user wants to exercise some control over the player or get information about the current song, an activity can bind to the service by calling bindService().

- **Implementing the lifecycle callbacks**

Like an activity, a service has lifecycle callback methods that you can implement to monitor changes in the service's state and perform work at the appropriate times. The following skeleton service demonstrates each of the lifecycle methods:

```
public class ExampleService extends Service {
   int mStartMode;       // indicates how to behave if the service is killed
   IBinder mBinder;      // interface for clients that bind
   boolean mAllowRebind; // indicates whether onRebind should be used
   @Override
   public void onCreate() {
      // The service is being created
   }
   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
      // The service is starting, due to a call to startService()
      return mStartMode;
   }
   @Override
   public IBinder onBind(Intent intent) {
      // A client is binding to the service with bindService()
      return mBinder;
   }
   @Override
   public boolean onUnbind(Intent intent) {
      // All clients have unbound with unbindService()
      return mAllowRebind;
   }
   @Override
   public void onRebind(Intent intent) {
      // A client is binding to the service with bindService(),
      // after onUnbind() has already been called
   }
   @Override
   public void onDestroy() {
      // The service is no longer used and is being destroyed
   }
}
```

Note: Unlike the activity lifecycle callback methods, you are *not* required to call the superclass implementation of these callback methods.

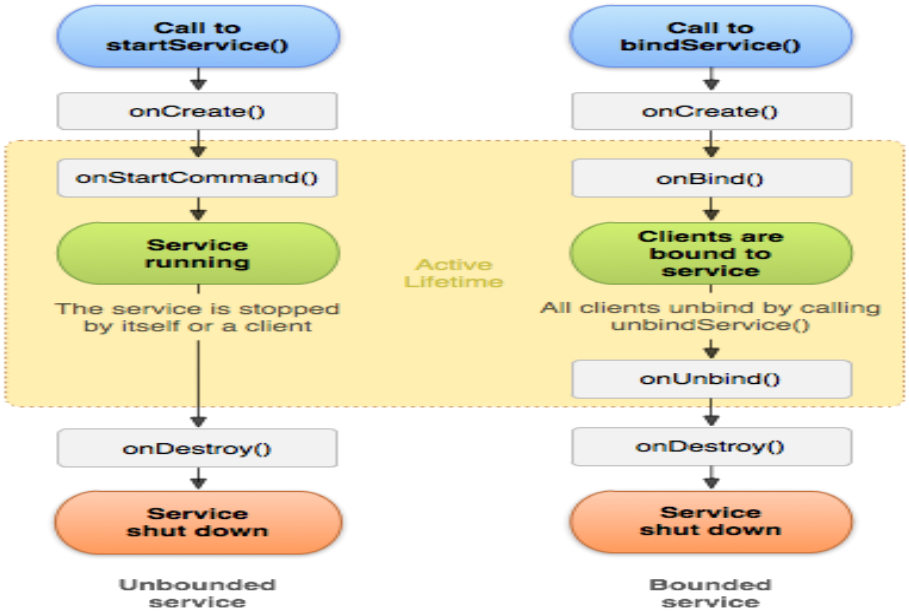


Figure : The service lifecycle. The diagram on the left shows the lifecycle when the service is created with startService() and the diagram on the right shows the lifecycle when the service is created with bindService().

By implementing these methods, you can monitor two nested loops of the service's lifecycle:

- The entire lifetime of a service happens between the time onCreate() is called and the time onDestroy() returns. Like an activity, a service does its initial setup in onCreate() and releases all remaining resources in onDestroy(). For example, a music playback service could create the thread where the music will be played in onCreate(), then stop the thread in onDestroy(). The onCreate() and onDestroy() methods are called for all services, whether they're created by startService() or bindService().
- The active lifetime of a service begins with a call to either onStartCommand() or onBind(). Each method is handed the Intent that was passed to either startService() or bindService(), respectively.
  If the service is started, the active lifetime ends the same time that the entire lifetime ends (the service is still active even after onStartCommand() returns). If the service is bound, the active lifetime ends when onUnbind() returns.

Note: Although a started service is stopped by a call to either stopSelf() or stopService(), there is not a respective callback for the service (there's no onStop() callback). So, unless the service is bound to a client, the system destroys it when the service is stopped—onDestroy() is the only callback received.

Figure: illustrates the typical callback methods for a service. Although the figure separates services that are created by startService() from those created by bindService(), keep in mind that any service, no matter how it's started, can potentially allow clients to bind to it. So, a service that was initially started with onStartCommand() (by a client calling startService()) can still receive a call to onBind() (when a client calls bindService()).

Bound Services

A bound service is the server in a client-server interface. A bound service allows components (such as activities) to bind to the service, send requests, receive responses, and even perform interprocess communication (IPC). A bound service typically lives only while it serves another application component and does not run in the background indefinitely.

- **The Basics:**

A bound service is an implementation of the Service class that allows other applications to bind to it and interact with it. To provide binding for a service, you must implement the onBind() callback method. This method returns an IBinder object that defines the programming interface that clients can use to interact with the service.

- **Binding to a started Services:**

you can create a service that is both started and bound. That is, the service can be started by calling startService(), which allows the service to run indefinitely, and also allow a client to bind to the service by calling bindService().

If you do allow your service to be started and bound, then when the service has been started, the system does *not* destroy the service when all clients unbind. Instead, you must explicitly stop the service, by calling stopSelf() or stopService().

Although you should usually implement either onBind() *or* onStartCommand(), it's sometimes necessary to implement both. For example, a music player might find it useful to allow its service to run indefinitely and also provide binding. This way, an activity can start the service to play some music and the music continues to play even if the user leaves the application. Then, when the user returns to the application, the activity can bind to the service to regain control of playback.

When you implement your bound service, the most important part is defining the interface that your onBind() callback method returns. There are a few different ways you can define your service's IBinder interface and the following section discusses each technique.

- **Creating a Bound Service:**

When creating a service that provides binding, you must provide an IBinder that provides the programming interface that clients can use to interact with the service. There are three ways you can define the interface:

→ **Extending the Binder class**

If your service is private to your own application and runs in the same process as the client (which is common), you should create your interface by extending the Binder class and returning an instance of it from onBind(). The client receives the Binder and can use it to directly access public methods available in either the Binder implementation or even the Service.

This is the preferred technique when your service is merely a background worker for your own application. The only reason you would not create your interface this way is because your service is used by other applications or across separate processes.

→ **Using a Messenger**

If you need your interface to work across different processes, you can create an interface for the service with a Messenger. In this manner, the service defines a Handler that responds to different types of Message objects. This Handler is the basis for a Messenger that can then share an IBinder with the client, allowing the client to send commands to the service using Message objects. Additionally, the client can define a Messenger of its own so the service can send messages back.

This is the simplest way to perform interprocess communication (IPC), because the Messenger queues all requests into a single thread so that you don't have to design your service to be thread-safe.

→ **Using AIDL**

AIDL (Android Interface Definition Language) performs all the work to decompose objects into primitives that the operating system can understand and marshall them across processes to perform IPC. The previous technique, using a Messenger, is actually based on AIDL as its underlying structure. As mentioned above, the Messenger creates a queue of all the client

requests in a single thread, so the service receives requests one at a time. If, however, you want your service to handle multiple requests simultaneously, then you can use AIDL directly. In this case, your service must be capable of multi-threading and be built thread-safe.

To use AIDL directly, you must create an .aidl file that defines the programming interface. The Android SDK tools use this file to generate an abstract class that implements the interface and handles IPC, which you can then extend within your service.

- **Managing the Lifecycle of a Bound Service:**

When a service is unbound from all clients, the Android system destroys it (unless it was also started with onStartCommand()). As such, you don't have to manage the lifecycle of your service if it's purely a bound service—the Android system manages it for you based on whether it is bound to any clients. However, if you choose to implement the onStartCommand() callback method, then you must explicitly stop the service, because the service is now considered to be started. In this case, the service runs until the service stops itself with stopSelf() or another component calls stopService(), regardless of whether it is bound to any clients.

Additionally, if your service is started and accepts binding, then when the system calls your onUnbind() method, you can optionally return true if you would like to receive a call to onRebind() the next time a client binds to the service (instead of receiving a call to onBind()). onRebind() returns void, but the client still receives the IBinder in its onServiceConnected() callback. Below, figure illustrates the logic for this kind of lifecycle.

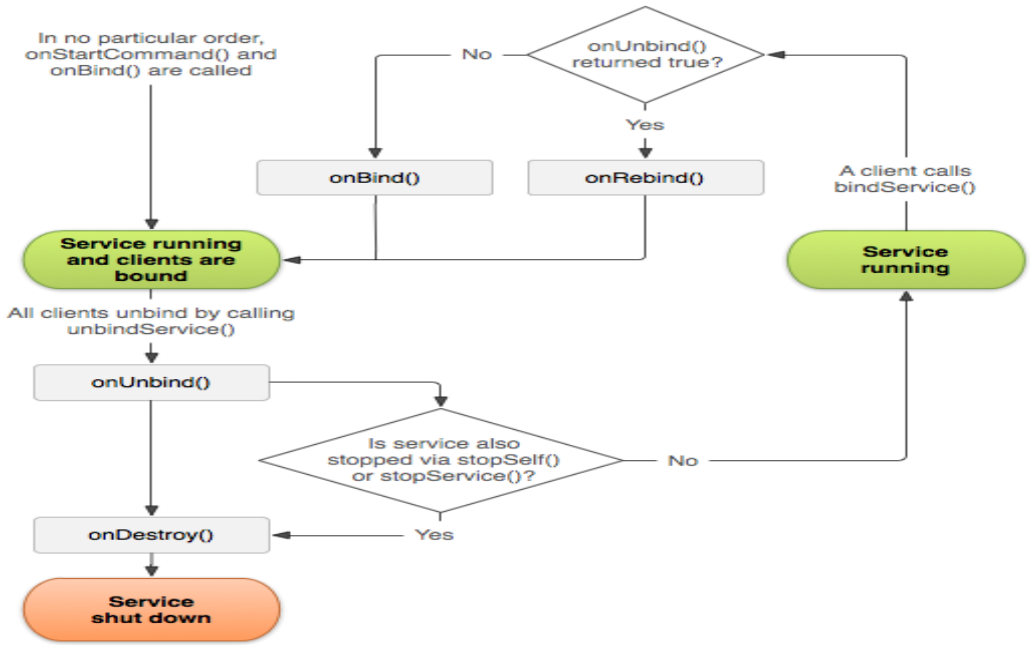


Figure : The lifecycle for a service that is started and also allows binding.

How to connect Android with PHP, MySQL

We are going see how to make a very simple Android app (in our case, a product inventory app) that will call a PHP script to perform basic CRUD(Create, Read, Update, Delete) operations. To brief you on the architecture, this is how it works. First your android app calls a PHP script in order to perform a data operation, lets say "create". The PHP script then connects to your MySQL database to perform the operation.

So the data flows from your Android app to PHP script then finally is stored in your MySQL database.

[1] WAMP SERVER:

WAMP is short form for Windows, Apache, MySQL and PHP, Perl, Python. WAMP software is one click installer which creates an environment for developing PHP, MySQL web application. By installing this software you will be installing **Apache**, **MySQL** and **PHP**. Alternatively you can use XAMP Server also.

[2] Installing and Running WAMP SERVER:

Download & Install WAMP server from www.wampserver.com/en/. Once you have installed wamp server, launch the program from Start -> All Programs -> WampServer -> StartWampServer. You can test your server by opening the address http://localhost/ in your browser. Also you can check phpmyadmin by opening http://localhost/phpmyadmin.

[3] Creating and Running PHP Project:

Now you have the environment ready to develop a PHP & MySQL project. Go to the location where you installed WAMP server (In my case i installed in C:\wamp\) and go to www folder and create a new folder for your project. You have to place all your project files inside this folder.

Create a folder called android_connect and create a new php file called **test.php** and try out simple php code. After placing following code try to open http://localhost/android_connect/test.php and you should see a message called "Welcome, I am connecting Android to PHP, MySQL".

test.php

```
<?php
   echo "Welcome, I am connecting Android to PHP, MySQL";
?>
```

[4] Creating MYSQL Database and Tables:

Create a simple database with one table. Through out this use same table to perform example operations. Now open **phpmyadmin** by opening the address http://localhost/phpmyadmin/ in your browser. You can use the PhpMyAdmin tool to create a database and a table.

I am creating a database named jitudb and a table called products.

```
CREATE DATABASE jitudb;
CREATE TABLE products(
pid int(11) primary key auto_increment,
name varchar(100) not null,
price decimal(10,2) not null,
description text);
```

[5] Connectiong to MYSQL Database using PHP:

Now the actual server side coding starts. Create a PHP class to connect to MySQL database. The main purpose of this class is to open a connection to database and close the connection whenever its not needed. So create two files called **db_config.php** and **db_connect.php**

db_config.php – will have database connection variables

db_connect.php – a class file to connect to database

Following is code for two php files

db_config.php

```
<?php
 /*
 * All database connection variables
 */
 define('DB_USER', "root"); // db user
define('DB_PASSWORD', ""); // db password (mention your db password here)
```

```
define('DB_DATABASE', "jitudb"); // database name
define('DB_SERVER', "localhost"); // db server
?>
```

db_connect.php

```
<?php

/**
 * A class file to connect to database
 */
class DB_CONNECT {
    // constructor
   function __construct() {
      // connecting to database
      $this->connect();
   }
    // destructor
   function __destruct() {
      // closing db connection
      $this->close();
   }
    /**
    * Function to connect with database
    */
   function connect() {
      // import database connection variables
      require_once __DIR__ . '/db_config.php';
       // Connecting to mysql database
      $con = mysql_connect(DB_SERVER, DB_USER, DB_PASSWORD) or die(mysql_error());
       // Selecing database
      $db = mysql_select_db(DB_DATABASE) or die(mysql_error()) or die(mysql_error());
       // returing connection cursor
      return $con;
   }
    /**
    * Function to close db connection
    */
   function close() {
      // closing db connection
      mysql_close();
   }
 }
 ?>
```

[6] Basic MYSQL CRUD Operations using PHP:

(A) Creating a row in MYSQL (Creating a new Product row)

In your PHP project create a new php file called **create_product.php** and place the following code. This file is mainly for creating a new product in products table.

```
<?php
 /*
 * Following code will create a new product row
 * All product details are read from HTTP Post Request
 */
 // array for JSON response
$response = array();
 // check for required fields
if (isset($_POST['name']) && isset($_POST['price']) && isset($_POST['description'])) {
    $name = $_POST['name'];
   $price = $_POST['price'];
   $description = $_POST['description'];
    // include db connect class
   require_once __DIR__ . '/db_connect.php';
    // connecting to db
   $db = new DB_CONNECT();
    // mysql inserting a new row
   $result = mysql_query("INSERT INTO products(name, price, description) VALUES('$name', '$price',
'$description')");
    // check if row inserted or not
   if ($result) {
      // successfully inserted into database
      $response["success"] = 1;
      $response["message"] = "Product successfully created.";

      // echoing JSON response
      echo json_encode($response);
   } else {
      // failed to insert row
      $response["success"] = 0;
      $response["message"] = "Oops! An error occurred.";

      // echoing JSON response
      echo json_encode($response);
   }
} else {
   // required field is missing
   $response["success"] = 0;
   $response["message"] = "Required field(s) is missing";

   // echoing JSON response
   echo json_encode($response);
}
```

```
?>
```

For the above code JSON response will be like

When POST param(s) is missing

```
{
   "success": 0,
   "message": "Required field(s) is missing"
}
```

When product is successfully created

```
{
   "success": 1,
   "message": "Product successfully created."
}
```

When error occurred while inserting data

```
{
   "success": 0,
   "message": "Oops! An error occurred."
}
```

(B) Reading a Row from MYSQL (Reading product Details):

Create a new php file called **get_product_details.php** and write the following code. This file will get single product details by taking product id (pid) as post parameter.

```
<?php
 /*
 * Following code will get single product details
 * A product is identified by product id (pid)

 */
 // array for JSON response
$response = array();
 // include db connect class
require_once __DIR__ . '/db_connect.php';
 // connecting to db
$db = new DB_CONNECT();
 // check for post data
if (isset($_GET["pid"])) {
   $pid = $_GET['pid'];
    // get a product from products table
   $result = mysql_query("SELECT *FROM products WHERE pid = $pid");
    if (!empty($result)) {
      // check for empty result
      if (mysql_num_rows($result) > 0) {
          $result = mysql_fetch_array($result);
          $product = array();
         $product["pid"] = $result["pid"];
         $product["name"] = $result["name"];
         $product["price"] = $result["price"];
```

```
        $product["description"] = $result["description"];
        $product["created_at"] = $result["created_at"];
        $product["updated_at"] = $result["updated_at"];
        // success
        $response["success"] = 1;
         // user node
        $response["product"] = array();
         array_push($response["product"], $product);
         // echoing JSON response
        echo json_encode($response);
      } else {
        // no product found
        $response["success"] = 0;
        $response["message"] = "No product found";

        // echo no users JSON
        echo json_encode($response);
      }
   } else {
      // no product found
      $response["success"] = 0;
      $response["message"] = "No product found";
       // echo no users JSON
      echo json_encode($response);
   }
} else {
   // required field is missing
   $response["success"] = 0;
   $response["message"] = "Required field(s) is missing";
    // echoing JSON response
   echo json_encode($response);
}
?>
```

(C)Reading All Rows from MYSQL (Reading all products)

We need a json to list all the products on android device. So create a new php file named **get_all_products.php** and write following code.

```
<?php
 /*
 * Following code will list all the products
 */
 // array for JSON response
$response = array();
 // include db connect class
require_once __DIR__ . '/db_connect.php';
 // connecting to db
```

```
$db = new DB_CONNECT();
 // get all products from products table
$result = mysql_query("SELECT *FROM products") or die(mysql_error());
 // check for empty result
if (mysql_num_rows($result) > 0) {
   // looping through all results
   // products node
   $response["products"] = array();
    while ($row = mysql_fetch_array($result)) {
      // temp user array
      $product = array();
      $product["pid"] = $row["pid"];
      $product["name"] = $row["name"];
      $product["price"] = $row["price"];
      $product["created_at"] = $row["created_at"];
      $product["updated_at"] = $row["updated_at"];
       // push single product into final response array
      array_push($response["products"], $product);
   }
   // success
   $response["success"] = 1;
    // echoing JSON response
   echo json_encode($response);
} else {
   // no products found
   $response["success"] = 0;
   $response["message"] = "No products found";
    // echo no users JSON
   echo json_encode($response);
}
?>
```

(D)Updating a Row in MYSQL (Updating product details)

Create a php file named **update_product.php** to update product details. Each product is identified by pid.

```
<?php
 /*
 * Following code will update a product information
 * A product is identified by product id (pid)
 */
 // array for JSON response
$response = array();
 // check for required fields
if (isset($_POST['pid']) && isset($_POST['name']) && isset($_POST['price']) &&
isset($_POST['description'])) {

   $pid = $_POST['pid'];
```

```
   $name = $_POST['name'];
   $price = $_POST['price'];
   $description = $_POST['description'];
    // include db connect class
   require_once __DIR__ . '/db_connect.php';
    // connecting to db
   $db = new DB_CONNECT();
    // mysql update row with matched pid
   $result = mysql_query("UPDATE products SET name = '$name', price = '$price', description =
'$description' WHERE pid = $pid");
    // check if row inserted or not
   if ($result) {
      // successfully updated
      $response["success"] = 1;
      $response["message"] = "Product successfully updated.";
       // echoing JSON response
      echo json_encode($response);
   } else {

   }
} else {
   // required field is missing
   $response["success"] = 0;
   $response["message"] = "Required field(s) is missing";
    // echoing JSON response
   echo json_encode($response);
}
?>
```

(E) Deleting a Row in MYSQL (Deleting a product)

The last operation is deletion on database. Create a new php file called **delete_product.php** and paste the following code. The main functionality of this file is to delete a product from database.

```
<?php
 /*
 * Following code will delete a product from table
 * A product is identified by product id (pid)
 */

// array for JSON response
$response = array();
 // check for required fields
if (isset($_POST['pid'])) {
   $pid = $_POST['pid'];
    // include db connect class
   require_once __DIR__ . '/db_connect.php';
    // connecting to db
```

```
    $db = new DB_CONNECT();
     // mysql update row with matched pid
    $result = mysql_query("DELETE FROM products WHERE pid = $pid");
     // check if row deleted or not
    if (mysql_affected_rows() > 0) {
        // successfully updated
        $response["success"] = 1;
        $response["message"] = "Product successfully deleted";
         // echoing JSON response
        echo json_encode($response);
    } else {
        // no product found
        $response["success"] = 0;
        $response["message"] = "No product found";
         // echo no users JSON
        echo json_encode($response);
    }
} else {
    // required field is missing
    $response["success"] = 0;
    $response["message"] = "Required field(s) is missing";
     // echoing JSON response
    echo json_encode($response);
}
?>
```

[7] Creating Android Application

Create a new project in Android studio by filling the required details.

1. Create new project in Android studio by going to **File ⇒New ⇒Android Project** and name the Activity class name as **MainActivity**.

2. Open your **AndroidManifest.xml** file and add following code. First i am adding all the classes i am creating to manifest file. Also i am adding INTERNET Connect permission.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.admin.testphpoperations" >
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.admin.testphpoperations.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

```
            <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
        </activity>
        <activity
          android:name="com.example.admin.testphpoperations.AllProductActivity"
          android:label="@string/title_activity_all_product" >
        </activity>
        <activity
          android:name="com.example.admin.testphpoperations.NewProductActivity"
          android:label="@string/title_activity_new_product" >
        </activity>
        <activity
          android:name="com.example.admin.testphpoperations.EditProductActivity"
          android:label="@string/title_activity_edit_product" >
        </activity>
    </application>
</manifest>
```

3. Now edit the xml file under **res ⇒layout** folder named as **activity_main.xml** this layout file contains two simple buttons to view all products and add a new product.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:gravity="center_horizontal">

    <!--  Sample Dashboard screen with Two buttons -->
    <!--  Button to view all products screen -->
    <Button android:id="@+id/btnViewProducts"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="View Products"
        android:layout_marginTop="25dip"/>
    <!--  Button to create a new product screen -->
    <Button android:id="@+id/btnCreateProduct"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Add New Products"
        android:layout_marginTop="25dip"/>
</LinearLayout>
```

4.Open you main activity class which is **MainActivity.java** and write click events for two button which are mentioned in **main_activity.xml** layout.

```
package com.example.admin.testphpoperations;
import android.app.Activity;
import android.content.Intent;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class MainActivity extends Activity{
   Button btnViewProducts;
   Button btnNewProduct;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      // Buttons
      btnViewProducts = (Button) findViewById(R.id.btnViewProducts);
      btnNewProduct = (Button) findViewById(R.id.btnCreateProduct);
      // view products click event
      btnViewProducts.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            // Launching All products Activity
            Intent i = new Intent(getApplicationContext(), AllProductActivity.class);
            startActivity(i);
         }
      });
      // view products click event
      btnNewProduct.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            // Launching create new product activity
            Intent i = new Intent(getApplicationContext(), NewProductActivity.class);
            startActivity(i);

         }
      });
   }
}
```

Displaying all Products in ListView (Read)

5. Now we need an Activity display all the products in list view format.so create a new activity and name it **AllProductActivity.** As we know list view needs two xml files, one for listview and other is for single list row. Create new xml files under **res ⇒layout** folder and name it as **list_item.xml and** edit the xml file named **activity_all_product.xml**,which is created by android studio when a new activity created.

activity_all_product.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
```

```
    android:orientation="vertical">
    <!-- Main ListView
        Always give id value as list(@android:id/list)
    -->
    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

list_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

    <!-- Product id (pid) - will be HIDDEN - used to pass to other activity -->
    <TextView
        android:id="@+id/pid"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:visibility="gone" />
    <!-- Name Label -->
    <TextView
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingTop="6dip"
        android:paddingLeft="6dip"
        android:textSize="17dip"
        android:textStyle="bold" />
</LinearLayout>
```

6. The Code of the **AllProductActivity.java** is as follows

```
package com.example.admin.testphpoperations;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import org.apache.http.NameValuePair;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import android.app.ListActivity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
```

```
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;
public class AllProductActivity extends ListActivity {
   // Progress Dialog
   private ProgressDialog pDialog;
   // Creating JSON Parser object
   JSONParser jParser = new JSONParser();
   ArrayList<HashMap<String, String>> productsList;
   // url to get all products list
   private static String url_all_products =
"http://10.0.2.2:80/android_connect/get_all_products.php";
   // JSON Node names
   private static final String TAG_SUCCESS = "success";
   private static final String TAG_PRODUCTS = "products";
   private static final String TAG_PID = "pid";
   private static final String TAG_NAME = "name";
   // products JSONArray
   JSONArray products = null;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_all_product);
      // Hashmap for ListView
      productsList = new ArrayList<HashMap<String, String>>();
      // Loading products in Background Thread
      new LoadAllProducts().execute();
      // Get listview
      ListView lv = getListView();
      // on seleting single product
      // launching Edit Product Screen
      lv.setOnItemClickListener(new OnItemClickListener() {
         @Override
         public void onItemClick(AdapterView<?> parent, View view,
                        int position, long id) {
            // getting values from selected ListItem
            String pid = ((TextView) view.findViewById(R.id.pid)).getText()
                  .toString();
```

```
            // Starting new intent
            Intent in = new Intent(getApplicationContext(),
                  EditProductActivity.class);
            // sending pid to next activity
            in.putExtra(TAG_PID, pid);

            // starting new activity and expecting some response back
            startActivityForResult(in, 100);
        }
    });
}
// Response from Edit Product Activity
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // if result code 100
    if (resultCode == 100) {
        // if result code 100 is received
        // means user edited/deleted product
        // reload this screen again
        Intent intent = getIntent();
        finish();
        startActivity(intent);
    }

}
/**
 * Background Async Task to Load all product by making HTTP Request
 * */
class LoadAllProducts extends AsyncTask<String, String, String> {
    /**
     * Before starting background thread Show Progress Dialog
     * */
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        pDialog = new ProgressDialog(AllProductActivity.this);
        pDialog.setMessage("Loading products. Please wait...");
        pDialog.setIndeterminate(false);
        pDialog.setCancelable(false);
        pDialog.show();
    }
    /**
     * getting All products from url
     * */
```

```
protected String doInBackground(String... args) {
   // Building Parameters
   List<NameValuePair> params = new ArrayList<NameValuePair>();
   // getting JSON string from URL
   JSONObject json = jParser.makeHttpRequest(url_all_products, "GET", params);
   // Check your log cat for JSON reponse
   Log.d("All Products: ", json.toString());
   try {
      // Checking for SUCCESS TAG
      int success = json.getInt(TAG_SUCCESS);

      if (success == 1) {
         // products found
         // Getting Array of Products
         products = json.getJSONArray(TAG_PRODUCTS);
         // looping through All Products
         for (int i = 0; i < products.length(); i++) {
            JSONObject c = products.getJSONObject(i);
            // Storing each json item in variable
            String id = c.getString(TAG_PID);
            String name = c.getString(TAG_NAME);
            // creating new HashMap
            HashMap<String, String> map = new HashMap<String, String>();
            // adding each child node to HashMap key => value
            map.put(TAG_PID, id);
            map.put(TAG_NAME, name);
            // adding HashList to ArrayList
            productsList.add(map);
         }
      } /*else {
         // no products found
         // Launch Add New product Activity
         Intent i = new Intent(getApplicationContext(),
               NewProductActivity.class);
         // Closing all previous activities
         i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
         startActivity(i);
      }*/
   } catch (JSONException e) {
      e.printStackTrace();
   }
   return null;
}

/**
```

```
     * After completing background task Dismiss the progress dialog
     * **/
    protected void onPostExecute(String file_url) {
      // dismiss the dialog after getting all products
      if (pDialog!= null) {
        pDialog.dismiss();
        pDialog= null;
      }
      // updating UI from Background Thread
      runOnUiThread(new Runnable() {
        public void run() {
          /**
           * Updating parsed JSON data into ListView
           * */
          ListAdapter adapter = new SimpleAdapter(
              AllProductActivity.this, productsList,
              R.layout.list_item, new String[] { TAG_PID,
              TAG_NAME},
              new int[] { R.id.pid, R.id.name });
          // updating listview
          setListAdapter(adapter);
        }
      });
    }
  }
}
```

Adding a new Product (Write)

7. Create a new view and acivity to add a new product into mysql database. Create a simple form which contains EditText for product name, price and description. Name your activity as **NewProductActivity** and xml file named **activity_new_product.xml** will be also created with it by android studio.

Add the following code to your **activity_new_product.xml** file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <!-- Name Label -->
  <TextView android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Product Name"
    android:paddingLeft="10dip"
    android:paddingRight="10dip"
    android:paddingTop="10dip"
    android:textSize="17dip"/>
```

```
    <!-- Input Name -->
    <EditText android:id="@+id/inputName"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dip"
        android:layout_marginBottom="15dip"
        android:singleLine="true"/>
    <!-- Price Label -->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Price"
        android:paddingLeft="10dip"
        android:paddingRight="10dip"
        android:paddingTop="10dip"
        android:textSize="17dip"/>
    <!-- Input Price -->
    <EditText android:id="@+id/inputPrice"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dip"
        android:layout_marginBottom="15dip"
        android:singleLine="true"
        android:inputType="numberDecimal"/>
    <!-- Description Label -->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Description"
        android:paddingLeft="10dip"
        android:paddingRight="10dip"
        android:paddingTop="10dip"
        android:textSize="17dip"/>
    <!-- Input description -->
    <EditText android:id="@+id/inputDesc"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dip"
        android:layout_marginBottom="15dip"
        android:lines="4"
        android:gravity="top"/>
    <!-- Button Create Product -->
    <Button android:id="@+id/btnCreateProduct"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Create Product"/>
</LinearLayout>
```

The code of the **NewProductActivity.java** is as follows:

```
package com.example.admin.testphpoperations;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONException;
import org.json.JSONObject;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
public class NewProductActivity extends Activity {
   // Progress Dialog
   private ProgressDialog pDialog;
   JSONParser jsonParser = new JSONParser();
   EditText inputName;
   EditText inputPrice;
   EditText inputDesc;
   // url to create new product
   private static String url_create_product =
"http://10.0.2.2:80/android_connect/create_product.php";
   // JSON Node names
   private static final String TAG_SUCCESS = "success";
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_new_product);
      // Edit Text
      inputName = (EditText) findViewById(R.id.inputName);
      inputPrice = (EditText) findViewById(R.id.inputPrice);
      inputDesc = (EditText) findViewById(R.id.inputDesc);

      // Create button
      Button btnCreateProduct = (Button) findViewById(R.id.btnCreateProduct);

      // button click event
      btnCreateProduct.setOnClickListener(new View.OnClickListener() {

         @Override
```

```
        public void onClick(View view) {
            // creating new product in background thread
            new CreateNewProduct().execute();
        }
    });
}
/**
 * Background Async Task to Create new product
 * */
class CreateNewProduct extends AsyncTask<String, String, String> {

    /**
     * Before starting background thread Show Progress Dialog
     * */
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        pDialog = new ProgressDialog(NewProductActivity.this);
        pDialog.setMessage("Creating Product..");
        pDialog.setIndeterminate(false);
        pDialog.setCancelable(true);
        pDialog.show();
    }

    /**
     * Creating product
     * */
    protected String doInBackground(String... args) {
        String name = inputName.getText().toString();
        String price = inputPrice.getText().toString();
        String description = inputDesc.getText().toString();

        // Building Parameters
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("name", name));
        params.add(new BasicNameValuePair("price", price));
        params.add(new BasicNameValuePair("description", description));
        // getting JSON Object
        // Note that create product url accepts POST method
        JSONObject json = jsonParser.makeHttpRequest(url_create_product,
                "POST", params);

        // check log cat fro response
        Log.d("Create Response", json.toString());
```

```
        // check for success tag
        try {
            int success = json.getInt(TAG_SUCCESS);

            if (success == 1) {
                // successfully created product
                Intent i = new Intent(getApplicationContext(), AllProductActivity.class);
                startActivity(i);

                // closing this screen
                finish();
            } else {
                // failed to create product
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * After completing background task Dismiss the progress dialog
     * **/
    protected void onPostExecute(String file_url) {
        // dismiss the dialog once done
            if (pDialog!= null) {
            pDialog.dismiss();
            pDialog= null;
        }
    }
     }
}
```

8.Now add the new activity again named **EditProductActivity** for making chances to data and its xml file named **activity_edit_product.xml** will be created by Android studio.

The code of the **activity_edit_product.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <!-- Name Label -->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Product Name"
        android:paddingLeft="10dip"
```

```
   android:paddingRight="10dip"
   android:paddingTop="10dip"
   android:textSize="17dip"/>
<!-- Input Name -->
<EditText android:id="@+id/inputName"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_margin="5dip"
   android:layout_marginBottom="15dip"
   android:singleLine="true"/>
<!-- Price Label -->
<TextView android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Price"
   android:paddingLeft="10dip"
   android:paddingRight="10dip"
   android:paddingTop="10dip"
   android:textSize="17dip"/>
<!-- Input Price -->
<EditText android:id="@+id/inputPrice"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_margin="5dip"
   android:layout_marginBottom="15dip"
   android:singleLine="true"
   android:inputType="numberDecimal"/>
<!-- Description Label -->
<TextView android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Description"
   android:paddingLeft="10dip"
   android:paddingRight="10dip"
   android:paddingTop="10dip"
   android:textSize="17dip"/>
<!-- Input description -->
<EditText android:id="@+id/inputDesc"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_margin="5dip"
   android:layout_marginBottom="15dip"
   android:lines="4"
   android:gravity="top"/>
<LinearLayout android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:orientation="horizontal">
```

```
    <!-- Button Create Product -->
    <Button android:id="@+id/btnSave"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Save Changes"
      android:layout_weight="1"/>
    <!-- Button Create Product -->
    <Button android:id="@+id/btnDelete"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Delete"
      android:layout_weight="1"/>
  </LinearLayout>
</LinearLayout>
```

The code of the **EditProductActivity.java**

```
package com.example.admin.testphpoperations;
import java.util.ArrayList;
import java.util.List;
import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class EditProductActivity extends Activity {
  EditText txtName;
  EditText txtPrice;
  EditText txtDesc;
  EditText txtCreatedAt;
  Button btnSave;
  Button btnDelete;
  String pid;
  // Progress Dialog
  private ProgressDialog pDialog;
  // JSON parser class
  JSONParser jsonParser = new JSONParser();
```

```
    // single product url
    private static final String url_product_detials =
"http://10.0.2.2:80/android_connect/get_product_details.php";
    // url to update product
    private static final String url_update_product =
"http://10.0.2.2:80/android_connect/update_product.php";
    // url to delete product
    private static final String url_delete_product =
"http://10.0.2.2:80/android_connect/delete_product.php";
    // JSON Node names
    private static final String TAG_SUCCESS = "success";
    private static final String TAG_PRODUCT = "product";
    private static final String TAG_PID = "pid";
    private static final String TAG_NAME = "name";
    private static final String TAG_PRICE = "price";
    private static final String TAG_DESCRIPTION = "description";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_edit_product);

        // save button
        btnSave = (Button) findViewById(R.id.btnSave);
        btnDelete = (Button) findViewById(R.id.btnDelete);

        // getting product details from intent
        Intent i = getIntent();

        // getting product id (pid) from intent
        pid = i.getStringExtra(TAG_PID);

        // Getting complete product details in background thread
        new GetProductDetails().execute();

        // save button click event
        btnSave.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                // starting background task to update product
                new SaveProductDetails().execute();
            }
        });

        // Delete button click event
```

```
    btnDelete.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View arg0) {
            // deleting product in background thread
            new DeleteProduct().execute();
        }
    });
}
/**
 * Background Async Task to Get complete product details
 * */
class GetProductDetails extends AsyncTask<String, String, String> {

    /**
     * Before starting background thread Show Progress Dialog
     * */
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        pDialog = new ProgressDialog(EditProductActivity.this);
        pDialog.setMessage("Loading product details. Please wait...");
        pDialog.setIndeterminate(false);
        pDialog.setCancelable(true);
        pDialog.show();
    }

    /**
     * Getting product details in background thread
     * */
    protected String doInBackground(String... params) {

        // updating UI from Background Thread
        runOnUiThread(new Runnable() {
            public void run() {
                // Check for success tag
                int success;
                try {
                    // Building Parameters
                    List<NameValuePair> params = new ArrayList<NameValuePair>();
                    params.add(new BasicNameValuePair("pid", pid));

                    // getting product details by making HTTP request
                    // Note that product details url will use GET request
                    JSONObject json = jsonParser.makeHttpRequest(
```

```
                url_product_detials, "GET", params);

            // check your log for json response
            Log.d("Single Product Details", json.toString());

            // json success tag
            success = json.getInt(TAG_SUCCESS);
            if (success == 1) {
               // successfully received product details
               JSONArray productObj = json
                     .getJSONArray(TAG_PRODUCT); // JSON Array

               // get first product object from JSON Array
               JSONObject product = productObj.getJSONObject(0);

               // product with this pid found
               // Edit Text
               txtName = (EditText) findViewById(R.id.inputName);
               txtPrice = (EditText) findViewById(R.id.inputPrice);
               txtDesc = (EditText) findViewById(R.id.inputDesc);

               // display product data in EditText
               txtName.setText(product.getString(TAG_NAME));
               txtPrice.setText(product.getString(TAG_PRICE));
               txtDesc.setText(product.getString(TAG_DESCRIPTION));

            }else{
               // product with pid not found
            }
         } catch (JSONException e) {
            e.printStackTrace();
         }
      }
   });

   return null;
}

/**
 * After completing background task Dismiss the progress dialog
 * **/
protected void onPostExecute(String file_url) {
   // dismiss the dialog once got all details
   pDialog.dismiss();
}
```

```
}

/**
 * Background Async Task to  Save product Details
 * */
class SaveProductDetails extends AsyncTask<String, String, String> {

   /**
    * Before starting background thread Show Progress Dialog
    * */
   @Override
   protected void onPreExecute() {
      super.onPreExecute();
      pDialog = new ProgressDialog(EditProductActivity.this);
      pDialog.setMessage("Saving product ...");
      pDialog.setIndeterminate(false);
      pDialog.setCancelable(true);
      pDialog.show();
   }

   /**
    * Saving product
    * */
   protected String doInBackground(String... args) {

      // getting updated data from EditTexts
      String name = txtName.getText().toString();
      String price = txtPrice.getText().toString();
      String description = txtDesc.getText().toString();

      // Building Parameters
      List<NameValuePair> params = new ArrayList<NameValuePair>();
      params.add(new BasicNameValuePair(TAG_PID, pid));
      params.add(new BasicNameValuePair(TAG_NAME, name));
      params.add(new BasicNameValuePair(TAG_PRICE, price));
      params.add(new BasicNameValuePair(TAG_DESCRIPTION, description));

      // sending modified data through http request
      // Notice that update product url accepts POST method
      JSONObject json = jsonParser.makeHttpRequest(url_update_product,
            "POST", params);

      // check json success tag
      try {
         int success = json.getInt(TAG_SUCCESS);
```

```
            if (success == 1) {
                // successfully updated
                Intent i = getIntent();
                // send result code 100 to notify about product update
                setResult(100, i);
                finish();
            } else {
                // failed to update product
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }

        return null;
    }

    /**
     * After completing background task Dismiss the progress dialog
     * **/
    protected void onPostExecute(String file_url) {
        // dismiss the dialog once product uupdated
        pDialog.dismiss();
    }
}
/****************************************************************
 * Background Async Task to Delete Product
 * */
class DeleteProduct extends AsyncTask<String, String, String> {

    /**
     * Before starting background thread Show Progress Dialog
     * */
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        pDialog = new ProgressDialog(EditProductActivity.this);
        pDialog.setMessage("Deleting Product...");
        pDialog.setIndeterminate(false);
        pDialog.setCancelable(true);
        pDialog.show();
    }

    /**
     * Deleting product
```

```
     * */
    protected String doInBackground(String... args) {

      // Check for success tag
      int success;
      try {
        // Building Parameters
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("pid", pid));

        // getting product details by making HTTP request
        JSONObject json = jsonParser.makeHttpRequest(
            url_delete_product, "POST", params);

        // check your log for json response
        Log.d("Delete Product", json.toString());

        // json success tag
        success = json.getInt(TAG_SUCCESS);
        if (success == 1) {
          // product successfully deleted
          // notify previous activity by sending code 100
          Intent i = getIntent();
          // send result code 100 to notify about product deletion
          setResult(100, i);
          finish();
        }
      } catch (JSONException e) {
        e.printStackTrace();
      }

      return null;
    }
    /**
     * After completing background task Dismiss the progress dialog
     * **/
    protected void onPostExecute(String file_url) {
      // dismiss the dialog once product deleted
      pDialog.dismiss();
    }
  }
}
```

9. JSON Parser Class

I used a JSON Parser class to get JSON from URL. This class supports two http request methods GET and POST to get json from url. So add a new java file in your project name **JSONParser** and its code is as follows:

```
package com.example.admin.testphpoperations;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.util.List;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.utils.URLEncodedUtils;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;

import android.util.Log;
public class JSONParser {
   static InputStream is = null;
   static JSONObject jObj = null;
   static String json = "";

   // constructor
   public JSONParser() {
   }

   // function get json from url
   // by making HTTP POST or GET mehtod
   public JSONObject makeHttpRequest(String url, String method,
                           List<NameValuePair> params) {

      // Making HTTP request
      try {

         // check for request method
         if(method == "POST"){
            // request method is POST
            // defaultHttpClient
```

```
        DefaultHttpClient httpClient = new DefaultHttpClient();
        HttpPost httpPost = new HttpPost(url);
        httpPost.setEntity(new UrlEncodedFormEntity(params));
        HttpResponse httpResponse = httpClient.execute(httpPost);
        HttpEntity httpEntity = httpResponse.getEntity();
        is = httpEntity.getContent();
    }else if(method == "GET"){
        // request method is GET
        DefaultHttpClient httpClient = new DefaultHttpClient();
        String paramString = URLEncodedUtils.format(params, "utf-8");
        url += "?" + paramString;
        HttpGet httpGet = new HttpGet(url);
        HttpResponse httpResponse = httpClient.execute(httpGet);
        HttpEntity httpEntity = httpResponse.getEntity();
        is = httpEntity.getContent();
    }
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
} catch (ClientProtocolException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
try {
    BufferedReader reader = new BufferedReader(new InputStreamReader(
            is, "iso-8859-1"), 8);
    StringBuilder sb = new StringBuilder();
    String line = null;
    while ((line = reader.readLine()) != null) {
        sb.append(line + "\n");
    }
    is.close();
    json = sb.toString();
} catch (Exception e) {
    Log.e("Buffer Error", "Error converting result " + e.toString());
}

// try parse the string to a JSON object
try {
    jObj = new JSONObject(json);
} catch (JSONException e) {
    Log.e("JSON Parser", "Error parsing data " + e.toString());
}
// return JSON String
return jObj;
```

```
    }
}
```

Deployment of Applications

As you are reading this chapter, you have completed your prior exercises and developed ne project that you would like to realease for real world as your own work.

Wait, before we are going to release the application for real world there are few things keep in mind. And "Ask Yourself following Questions".

→ **Is Application Tested ?**

If the answer to this is "no" then you need to test it again and do some precise testing. In addition to basic tests on the emulator, you should also test your application on a real, physical device. In an ideal world, you world test on multiple hardware devices running different versions of the Android OS. Faulty code makes for poor sales and a worse reputation.

→ **Application Performs Well**

Performance is really important, especially if you're programming a game. If your application is not responsive in certain cases, try to see if you can optimize those parts.

→ **Checked SDK compatibility.**

According to data collected around August 12, 2014 Android 2.2 is still active on 0.7% of devices that have accessed the Market.

Version	Code name	Release date	API level	Distribution
4.4	KitKat	October 31,2013	19	20.9%
4.3	Jelly Bean	July 24,2013	18	7.9%
4.2.x		November 13,2012	17	19.8%
4.1.x		July 9,2012	16	26.5%
4.0.3-4.0.4	Ice Cream Sandwich	December 16,2011	15	10.6%
2.3.3-2.3.7	Gingerbread	February 9,2011	10	13.6%
2.2	Froyo	May 20,2010	8	0.7%

Upto now we had get basic information about how we can target our application devices. And make them more reliable, and beneficial for us.

Now it's time to recheck our configuration for final launch.

Let's start by checking AndroidManifest.xml file.

(1) Request necessary Android permissions

Make sure that you're requesting all necessary permissions; otherwise your application won't work. Examples for permission declarations include:

<uses-permission android:name="android.permission.VIBRATE" />

<uses-permission android:name="android.permission.INTERNET" />

(2) Specify a name and icon

Name your application and give it an icon using the android:label and android:icon attribute in the application tag. For example:

<application android:label="@string/app_name" android:icon="@drawable/myIcon">

(3) Configure version manifest data

Pick a version your application using android:versionCode and android:versionName in the manifest XML tag. versionCode should have an integer value that you must increment for every

application update, as this will be used by the system. It's a common practice to start at 1. versionName on the other hand represents a user-friendly value. Feel free to set this to whatever you think is appropriate -0.1 or 1.0b or 2.4.1 would all technically work.

<manifest xmlns:android=http://schemas.android.com/apk/res/android package="com.example"
andriod:versionName="1.0.0">

(4) Set compatibility options

If you're utilizing newer Android features that aren't available in older versions, then you'll have to specify a set of version requirements. To do this, create bodiless XML tag name uses-sdk. The following attributes are at your disposal:

- → **android:minSdkVersion** The minimum Android platform API level on which your application will be able to run.
- → **android:targetSdkVersion** The API level that your application was designed to run on.
- → **android:maxSdkVersion** An upper limit for compatibility . Don't set this unless you have a very good reason to.

Note that the API level refers to a special identifier. Here's a handy table that you can refer to:

Platform Version	**API Level**	**VERSION_CODE**
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2,4.2.2	17	JELLY_BEAN_MR1
Android 4.1,4.1.1	16	JELLY_BEAN
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1,4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOM_MR1
Android 3.0.X	11	HONEYCOMB
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD
Android 2.2.X	8	FROYO
Android 2.1.X	7	ÉCLAIR_MR1
Android2.0.1	6	ÉCLAIR_0_1
Android 2.0	5	ÉCLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

(5) Cleanup files and remove logging

Go through your project and remove any logging calls, old log files, private data and unwanted resources files.

(6) Sign and ZIP- align your application

One of very important distribution requirements is that android applications must be digitally signed with a certificate that the developer holds. This is used to ensure the authenticity of an application so it's important that you pick a strong password for private key and ensure that you keep it safe.

To sign your app in release mode in Android Studio, follow these steps:

1. On the menu bar, click Build > Generate Signed APK.
2. On the Generate Signed APK Wizard window, click Create new to create a new keystore. If you already have a keystore, go to step 4.
3. On the New Key Store window, provide the required information as shown in figure 1. Your key should be valid for at least 25 years, so you can sign app updates with the same key through the lifespan of your app.

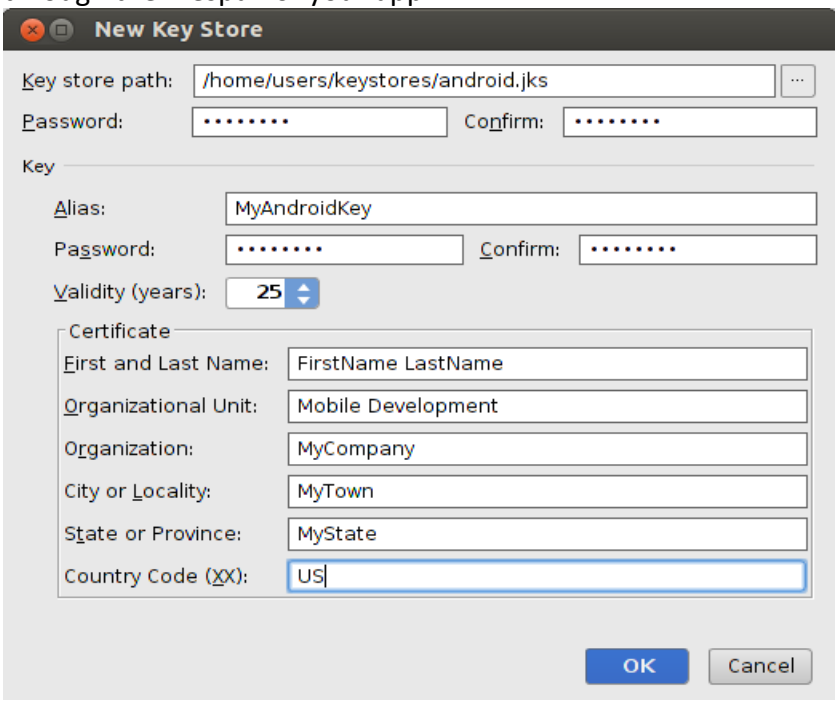


Figure 1. Create a new keystore in Android Studio.

4. On the Generate Signed APK Wizard window, select a keystore, a private key, and enter the passwords for both. Then click Next.

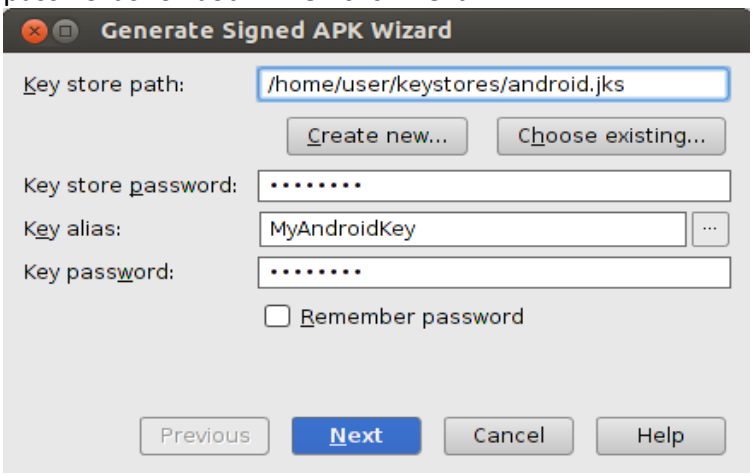


Figure 2. Select a private key in Android Studio.

5. On the next window, select a destination for the signed APK and click Finish.

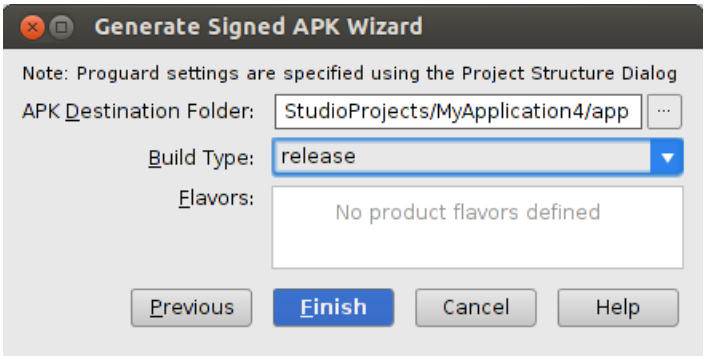
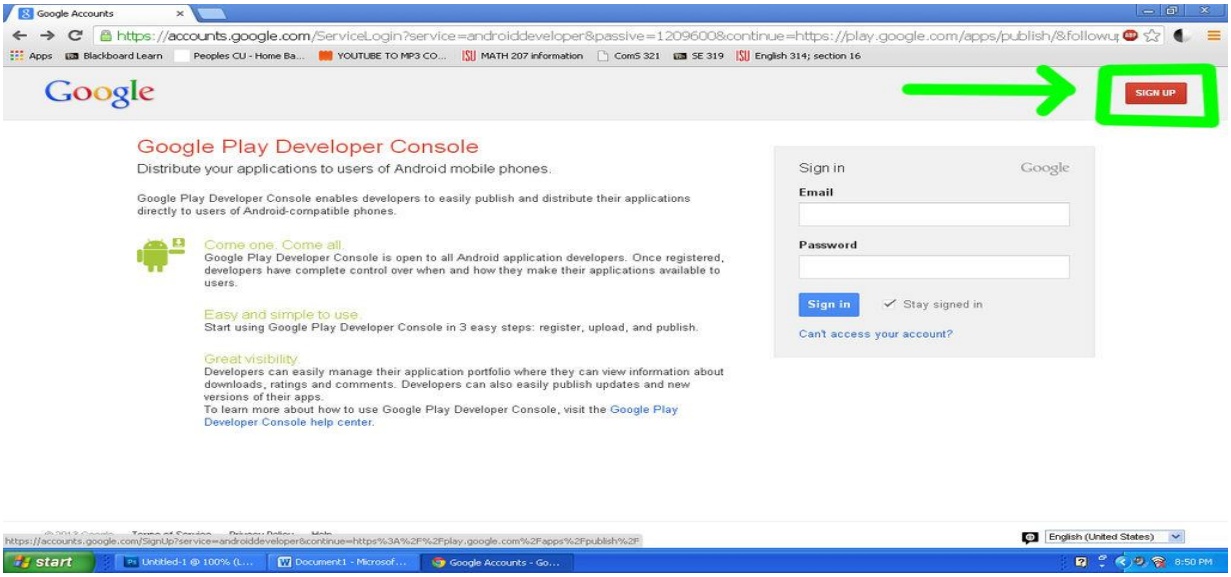


Figure 3. Generate a signed APK in Android Studio.

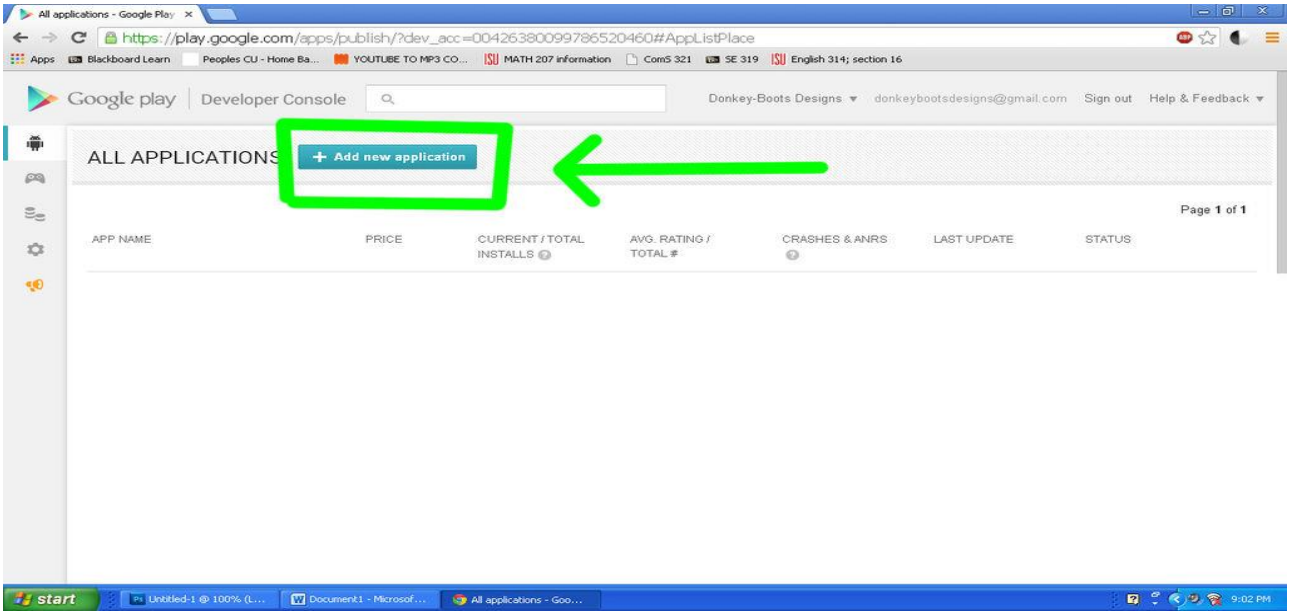
Publishing your App

[1] Logging into Google Developer Console



Get logged into google's developer console.

[2] Adding a New Application

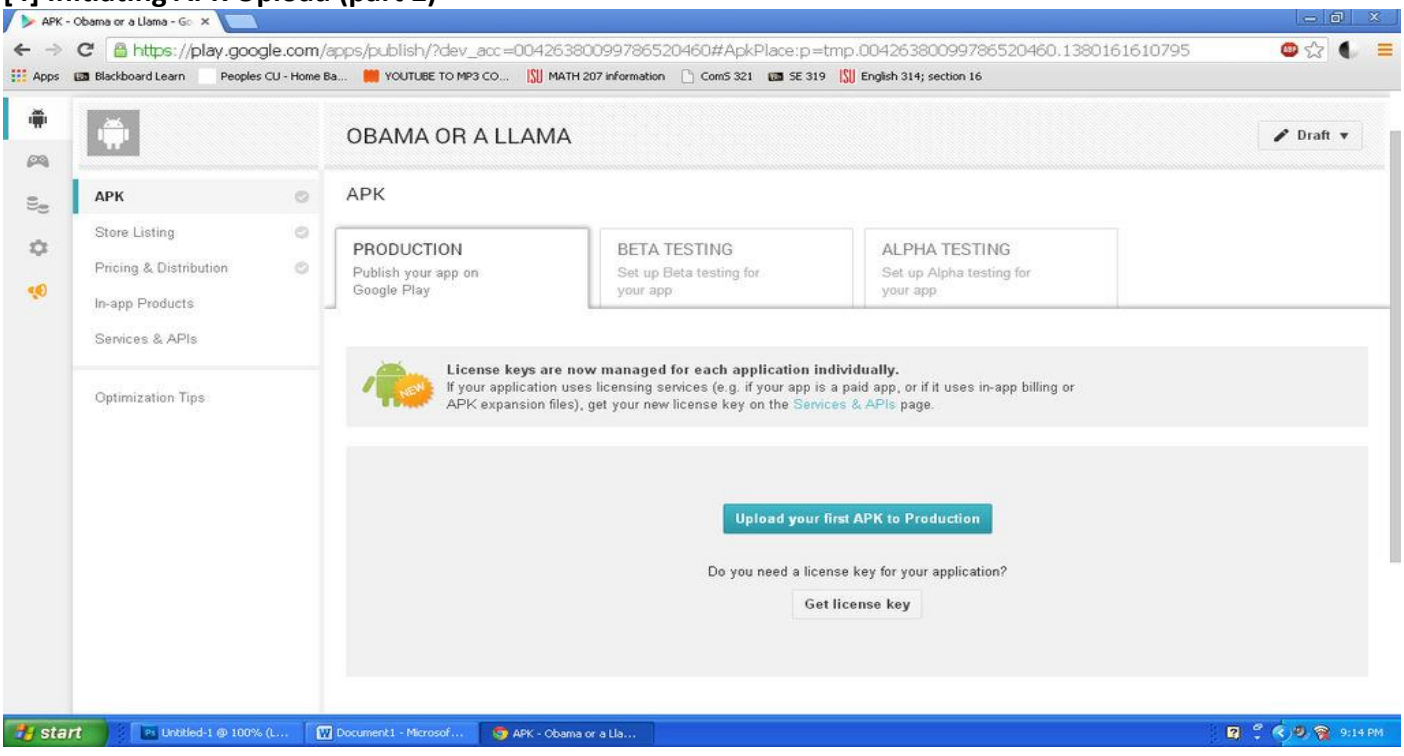


Once inside the console, click the blue button at the top labeled, "+Add New Application."

[3] Initiating APK Upload

In the "Add New Application" dialog, ensure that the correct language is selected from the drop-down menu and then type the name of the app as you wish for it to appear in the Google Play store.
Then, select the "Upload APK" button at the bottom.
The console will then take you to the new homepage for your app.

[4] Initiating APK Upload (part 2)

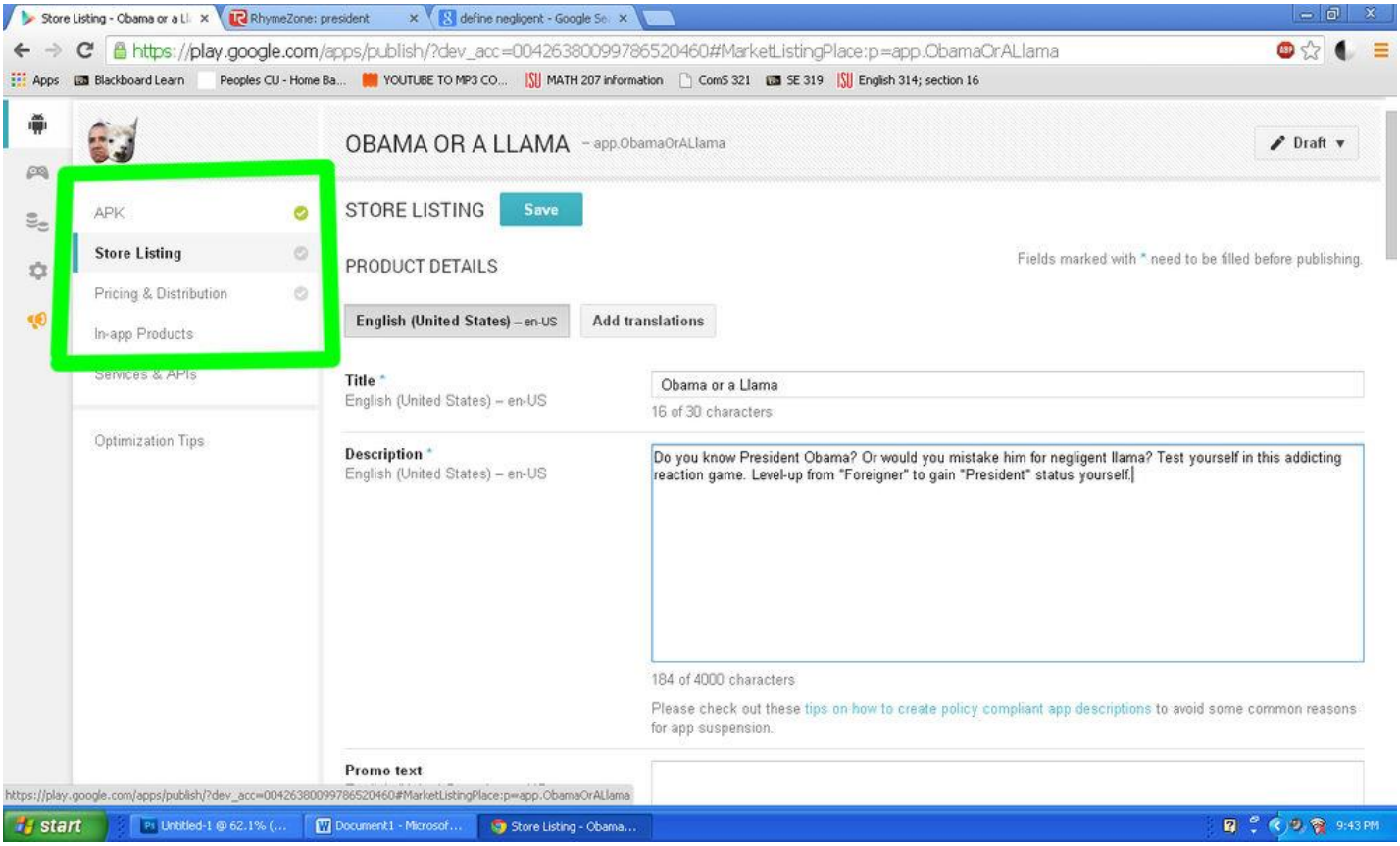


At the new homepage for your app, select the blue button labeled "Upload your first APK to Production," centered on your screen.

[5] Selecting and Uploading APK File

A dialog will appear, where you can “browse” and “open” your “.apk” file you exported. Upload progress is shown, and if the upload is successful, you will be taken back to the console.

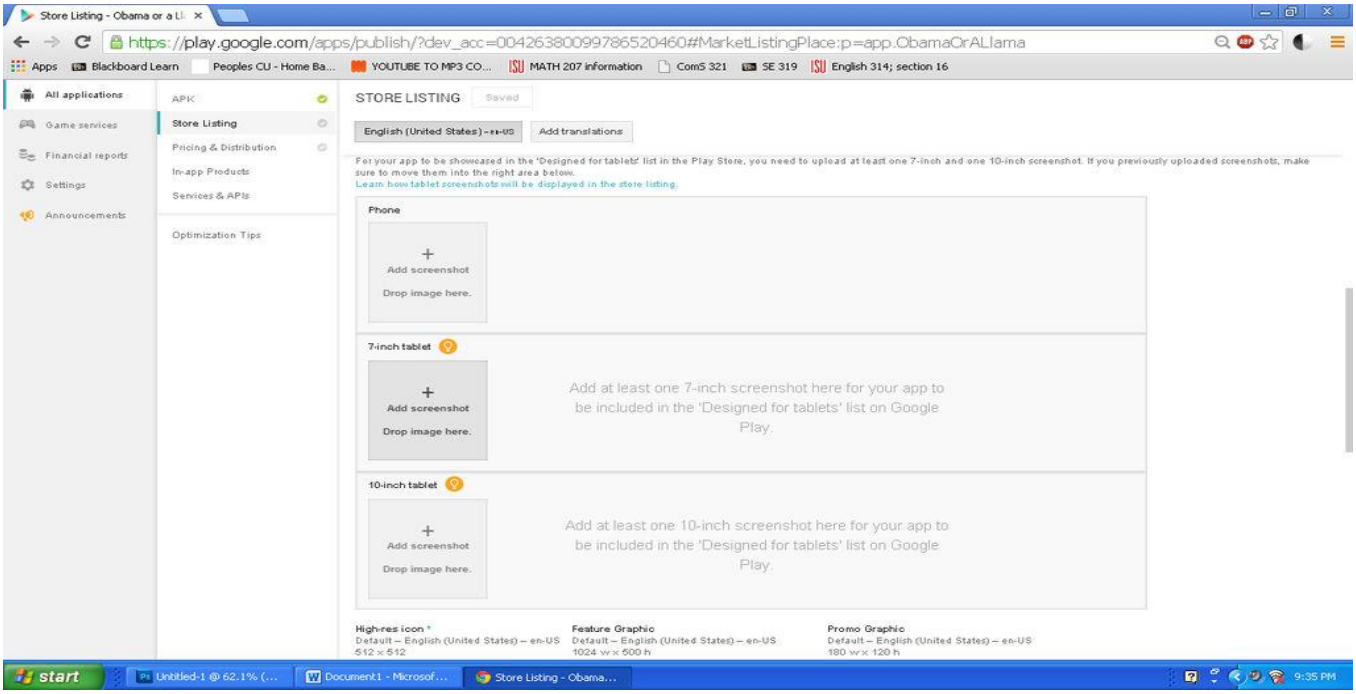
[6] Adding a Description



After successful uploading of the APK, a description needs to be added to the “store listing” page found by navigating the tabs at the left (tabs are shown in green box in attached figure).

Type an application description in the “description” text box. The description will appear in the Google Play store on the page for your app.

[7] Adding a Screenshots

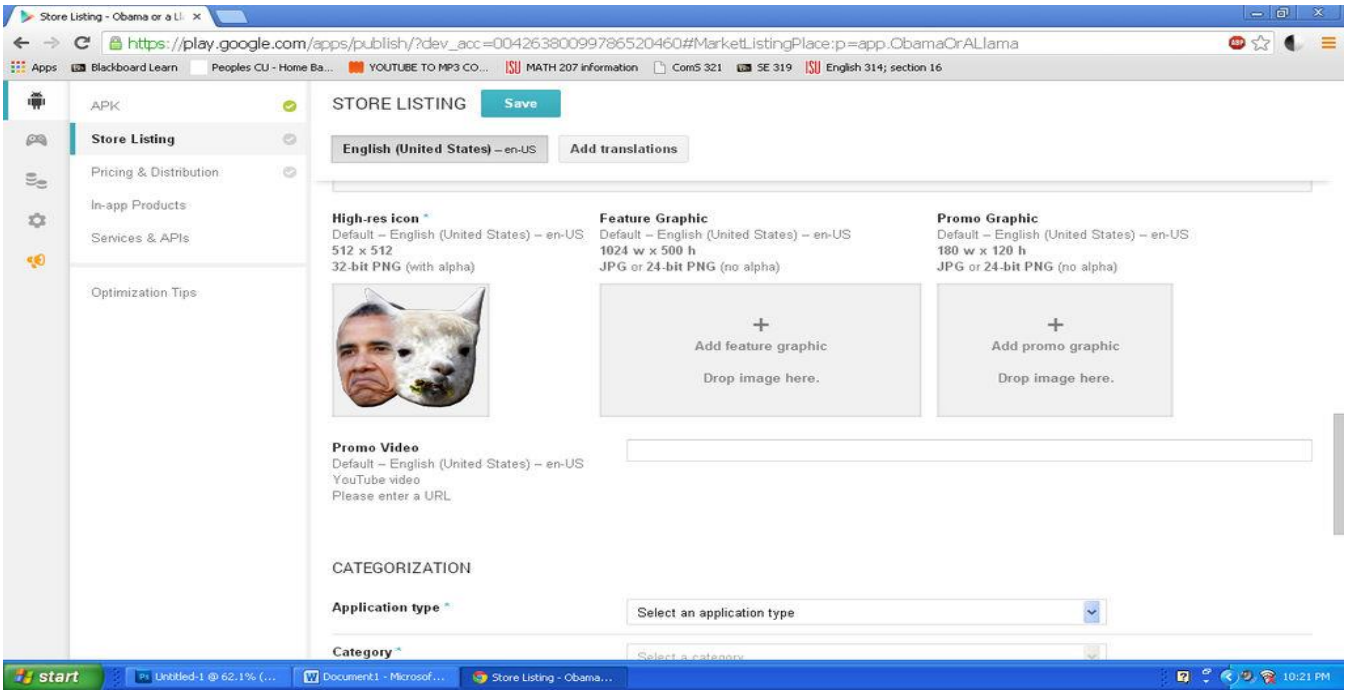


The next step in the “Store Listing” tab is adding sample screenshots.

To upload screen-shots, scroll down and click one of the three “+ Add Screenshot” buttons corresponding to the type of device they were taken on.

A browse dialog box will open allowing you to select the screenshots from a directory on your computer.

[8] Adding a store listing Icon



Next, add a store listing icon.

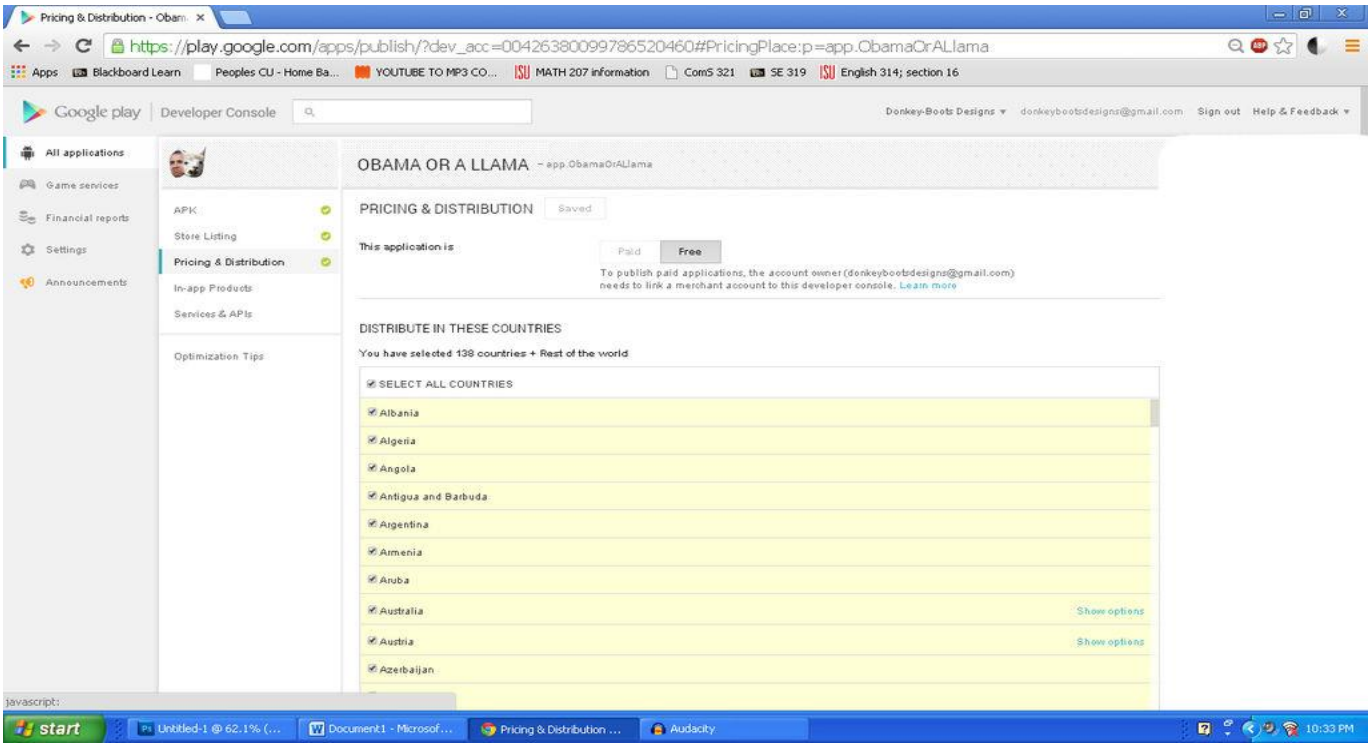
The listing icon is the main picture that pops up at the top of the screen in the Google Play Store.A browse dialog box will open allowing you to select the icon the same way Step 7 allowed you to select a screenshot.

[9] Filling in Details

The final step on the “Store Listing” page is filling out the categorization, contact details, and privacy policy, located below the icon selection area.

Fill out the required fields as you wish and hit save. The “store listing” tab should now have a green check-mark next to it.

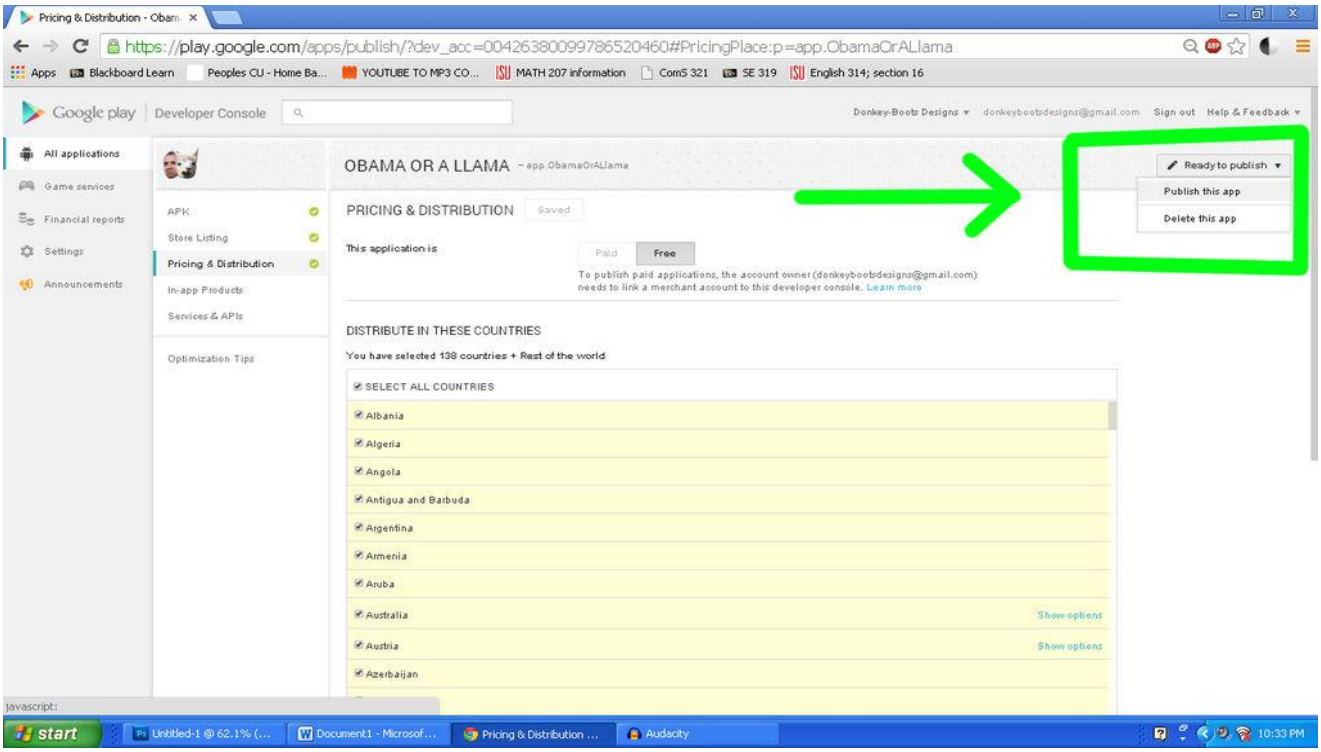
[10]



Finally, click the “Pricing & Distribution” tab, where you will select paid or free, distribution countries, and check the boxes saying that your app complies with content guidelines and US export laws.

Select "Save," at the bottom.

[11] Publishing the Application



Once all three tabs at the left have a green check-mark, you are now able to select “Publish this app” from the “Ready to Publish” drop-down menu in the top right corner of the developer console.
A confirmation bar should appear at the top, stating that your app will appear in the Google Play store in just a few hours.