الجامعة الافتراضية السورية
SYRIAN VIRTUAL UNIVERSITY

Mobile Applications

Doctor Bassel Alkhatib

ISSN: 2617-989X

Books & Refrences

التطبيقات النقالة

الدكتور باسل الخطيب

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2020

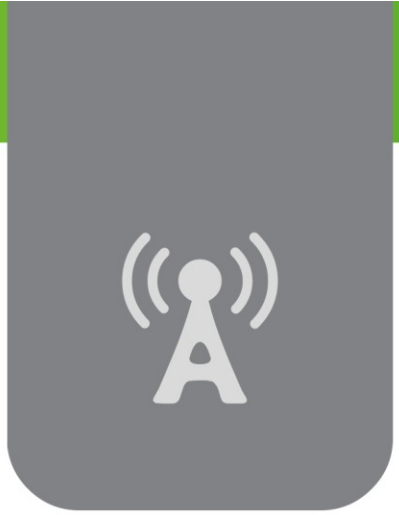
Mobile Applications

Dr. Bassel Alkhatib

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2020

Index



Chapter 1:
Java Basics

Title:

Java Basics

Key Words:

Variables, Data Types, Parameters, Conditional Statement, Loops, Arrays.

Summary:

This unit provides an overview of Java programming language basics.

Outcomes:

Student will learn in this unit:

- Using different data types.
- Passing parameters and returning values.
- Using conditional statements.
- Using loops.
- Dealing with arrays.

Plan:

2 Learning Objects:

1. Java Basics: Data Types
2. Java Basics: Statements

1. Data Types

Learning outcomes:

Using variables and data types.

Java's Variable

- **Variable:** A piece of the computer's memory that is given a name and type, and can store a value.
- A variable can be declared / initialized in one statement.
- Syntax:
  ```
  type name = value;
  ```
- Example:

```
double myGPA = 3.95;
int x = (11 % 3) + 12; // x will be 14
```

Java's primitive types:

- **primitive types:** 8 simple types for numbers, text, etc.
  - Java also has **object types**, which we'll talk about later

Name	Description	Examples
`int`	integers	`42, -3, 0, 926394`
`double`	real numbers	`3.1, -0.25, 9.4e3`
`char`	single text characters	`'a', 'X', '?', '\n'`
`boolean`	logical values	`true, false`

Type Casting

- **Type cast**: A conversion from one type to another.
  - To promote an int `int` a `double` to get exact division from /
  - To truncate a `double` from a real number to an integer
- Syntax:
  `(type) expression`
- Examples:

```
double result = (double) 19 /5;  // 3.8
int result2 = (int) result;      //   3
int x = (int) Math.pow(10, 3);   //   1000
```

Increment and decrement:

shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
variable++;	**variable** = **variable** + 1;
variable--;	**variable** = **variable** - 1;

- Examples:

```
int x = 2;
x++;                        // x = x + 1;
                            // x now stores 3

double gpa = 2.5;
gpa--;                      // gpa = gpa - 1;
                            //gpa now stores 1.5
```

Precedence:

- **Precedence**: Order in which operators are evaluated.
- Generally, operators evaluate left-to-right

```
1-2-3 is (1-2) -3 which is -4
```

- But * / % have a higher level of precedence than + -

```
1 + 3 * 4         is 13
6 + 8 / 2* 3
6 + 4 * 3
6 + 12       is 18
```

- Parentheses can force a certain order of evaluation

```
(1 + 3)*4         is 16
```

- Spacing does not affect order of evaluation

```
1 + 3 * 4 - 2     is 11
```

String concatenation:

- string concatenation: Using + between a string and another value to make a longer string.

```
"hello" + 42 is "hello42"
1 + "abc" + 2 is "1abc2"
"abc" + 1 + 2 is "abc12"
1 + 2 + "abc" is "3abc"
"abc" + 9 * 3 is "abc27"
"1" + 1 is "11"
4 - 1 + "abc" is "3abc"
```

- Use + to print a string and an expression's value together.
- Example:

```
System.out.println ("Grade:" + (95.1 + 71.9) / 2)
```

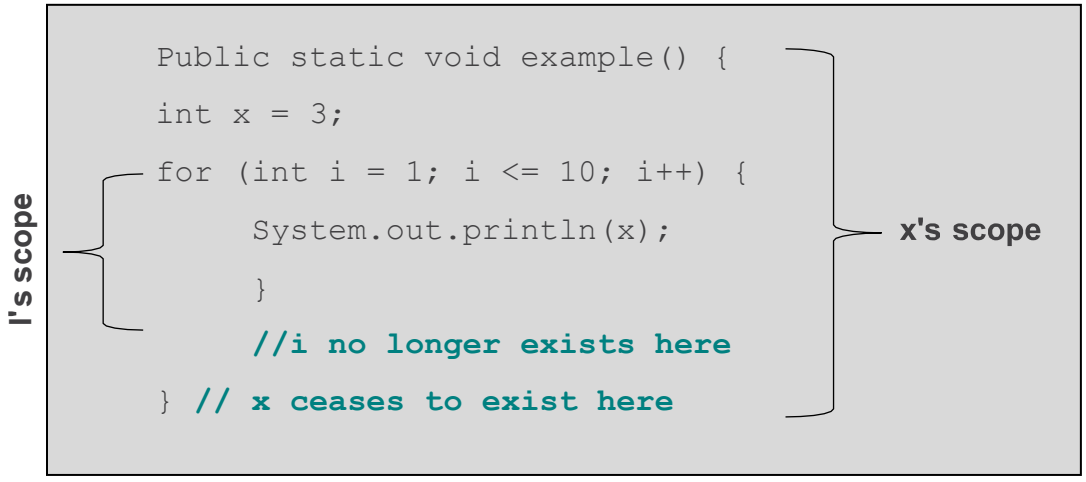
  - Output: `Grade: 83.5`

Variable scope:

- **Scope**: The part of a program where a variable exists.

  From its declaration to the end of the { } braces

  - A variable declared in a for loop exists only in that loop.
  - A variable declared in a method exists only in that method.



Class constants:

- **class constant**: A value visible to the whole program.
  - value can only be set at declaration
  - value can't be changed while the program is running
- Syntax:

  `public static final` **`type name`** = **`value;`**
  - name is usually in ALL_UPPER_CASE
  - Examples:

```
public static final int DAYS_IN_WEEK = 7;
public static final double INTEREST_RATE = 3.5
public static final int SSN = 658234569;
```

Passing parameters:

- Declaration:

```
public void name (type  name, ..., type  name) {
        statement(s);
}
```

- Call:

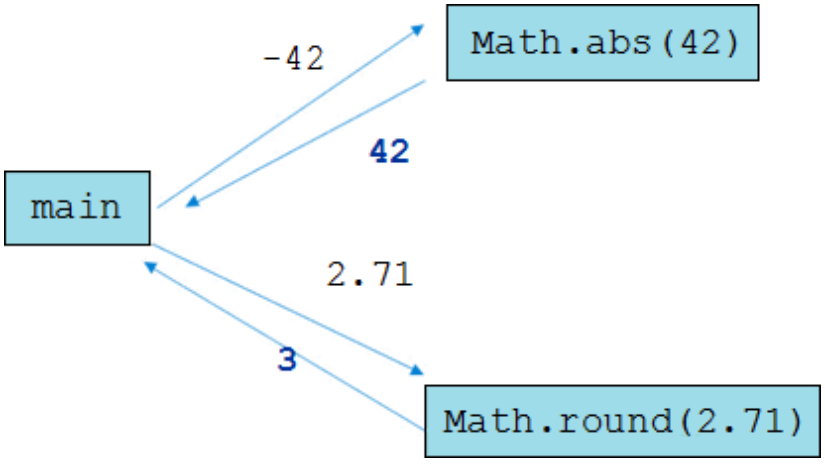
```
methodName (value, value, ..., value);
```

- Example:

```
public static void main(String[] args) {
     sayPassword(42);    //The password is: 42
     sayPassword(12345); //The password is: 12345
}


public static void sayPassword(int code) {
     System.out.println("The password is: " +
     code);
}
```

Return:

- **return**: To send out a value as the result of a method.
- The opposite of a parameter:
  - Parameters send information in from the caller to the method.
  - Return values send information out from a method to its caller.



Java's Math class:

Method name	Description
`Math.abs(value)`	absolute value
`Math.round(value)`	nearest whole number
`Math.ceil(value)`	rounds up
`Math.floor(value)`	rounds down
`Math.log10(value)`	logarithm, base 10
`Math.max(value1, value2)`	larger of two values
`Math.min(value1, value2)`	smaller of two values
`Math.pow(base, exp)`	base to the exp power
`Math.sqrt(value)`	square root
`Math.sin(value)`	sine of an angle in radians
`Math.cos(value)`	cosine of an angle in radians
`Math.tan(value)`	tangent of an angle in radians
`Math.toDegrees(value)`	convert degrees to radians and back
`Math.toRadians(value)`	random double between 0 and 1
`Math.random()`	sine of an angle in radians

Returning a value:

- Syntax:

```
public type name(parameters) { statements;
...
return expression;
}
```

- Example:

```
// Returns the slope of the line between the given points.
public double slope(int x1, int y1, int x2, int y2)
{
    double dy = y2 - y1;
    double dx = x2 - x1;
    return dy/dx;
}
```

Strings:

- **string**: An object storing a sequence of text characters.

```
String name = "text";
String name = expression;
```

  - Characters of a string are numbered with 0-based indexes:

```
String name = "P. Diddy";
```

Index	0	1	2	3	4	5	6	7
Char	P	.		D	i	d	d	y

- The first character's index is always 0
- The last character's index is 1 less than the string's length
- The individual characters are values of type `char`

`String` methods:

Method name	Description
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (−1 if it is not there)
`length()`	number of characters in this string

`substring(`**`index1`**`, `**`index2`**`) or`
 	
`substring(`**`index1`**`)`
	number of characters in this string
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* omitted, grabs till end of string

`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (−1 if it is not there)
`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- **These methods are called using the dot notation**

```
String gangsta = "Dr. Dre";
System.out.println (gangsta.length()); // 7
```

String test methods:

Method	Description
`equals(str)`	whether two strings contain the same characters
`equalsIgnoreCase(str)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(str)`	whether one contains other's characters at start
`endsWith(str)`	whether one contains other's characters at end
`contains(str)`	whether the given string is found within this one

```
String name = console.next();
if (name.startsWith("Dr.")) {
   System.out.println("Are you single?");
} else if (name.equalsIgnoreCase("LUMBERG")) {
   System.out.println("I need your TPS
   reports.");
}
```

The equals method:

- **Objects are compared using a method named** `equals`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
     System.out.println("I love you, you love me,");
     System.out.println("We're a happy family!");
}
```

Technically this is a method that returns a value of type `boolean`, the type used in logical tests.

Type char:

- **`char`**: A primitive type representing single characters.
  - Each character inside a `String` is stored as a `char` value.
  - Literal char values are surrounded with apostrophe (single-quote) marks, such as 'a' or '4' or '\n' or '\"'
  - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
System.out.println(letter); // S
```

- **`char`** values can be concatenated with strings.

```
char initial = 'P';
System.out.println(initial + " Diddy");// P Diddy
```

char vs. String:

- `"h"` is a String
- `'h'` is a `char` (the two behave differently)
- `String` is an object; it contains methods

```
String s = "h";                    // 'H'
s = s.toUpperCase();               // 1
int len = s.length();        // 1
char first = s.charAt(0);          // 'H'
```

- **Char**: is primitive; you can't call methods on it

```
char c = 'h';
c = c.toUpperCase();// ERROR: "cannot be dereferenced"
```

2. Java Basics: Statements

Learning outcomes:

conditional statements, loops, and array

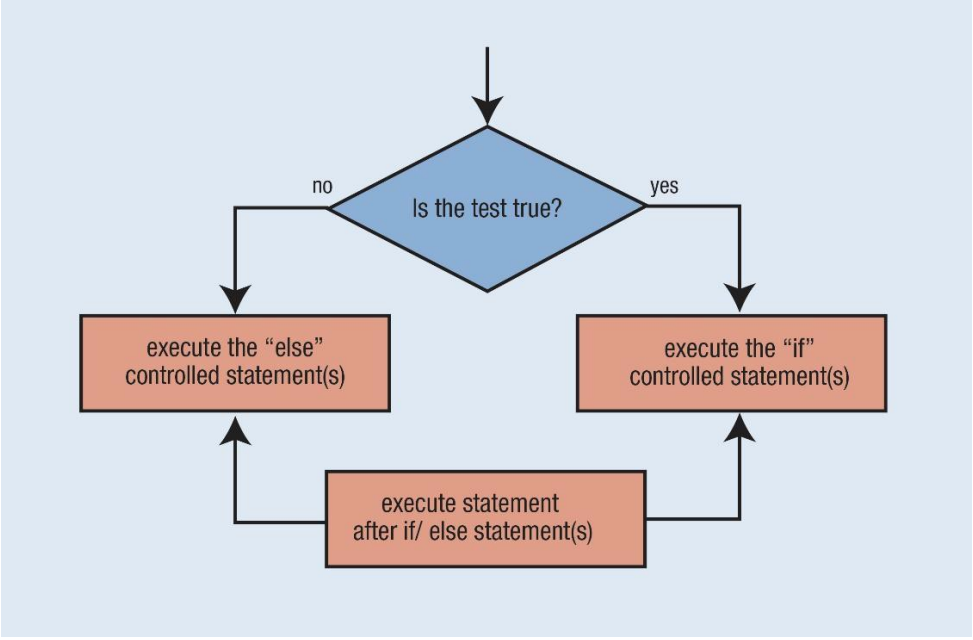
Relational expressions:

if/else

Executes one block if a test is true, another if false.

- Syntax:

```
if (test) {
        statement(s);
} else {
        statement(s);
}
```



- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println ("Welcome to Mars
    University!");
} else {
      System.out.println ("Application
      denied.");
}
```

- **A test** in an if is the same as in a for loop.

```
for (int i = 1; i <= 10; i++) { ...
if (i <= 10) { ...
```

  - **These are** `boolean` expressions.

- Tests use *relational operators:*

Operator	Meaning	Example	Value
==	equals	1 + 1 == 2	true
!=	does not equal	3.2 != 2.5	true
<	less than	10 < 5	false
>	greater than	10 > 5	true
<=	less than or equal to	126 <= 100	false
>=	greater than or equal to	5.0 >= 5.0	true

Logical operators: &&, ||, !

Conditions can be combined using *logical operators*:

Operator	Description	Example	Result
`&&`	and	`(2 == 3) && (-1 < 5)`	`false`
`||`	or	`(2 == 3) || (-1 < 5)`	`true`
`!`	not	`!(2 == 3)`	`true`

Truth table:

p	q	p && q	p || q	! p
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Type boolean:

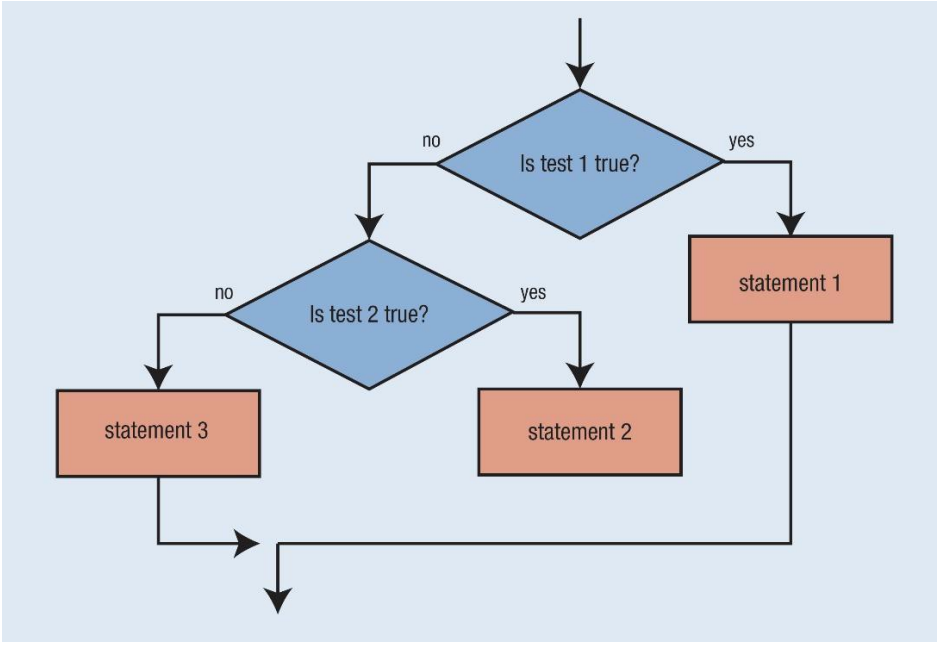
- **boolean**: A logical type whose values are true and false.
- **A test** in an `if`, `for`, or while is a `boolean` expression.
  - You can create `boolean` variables, pass `boolean` parameters, return boolean values from methods, ...
- Example

```
boolean minor = (age < 21);
boolean expensive = iPhonePrice > 200.00;
boolean iLoveCS = true;
if (minor) {
   System.out.println("Can't purchase alcohol!");
}
if (iLoveCS || !expensive) {
System.out.println("Buying an iPhone");
 }
```

If / else Structures:

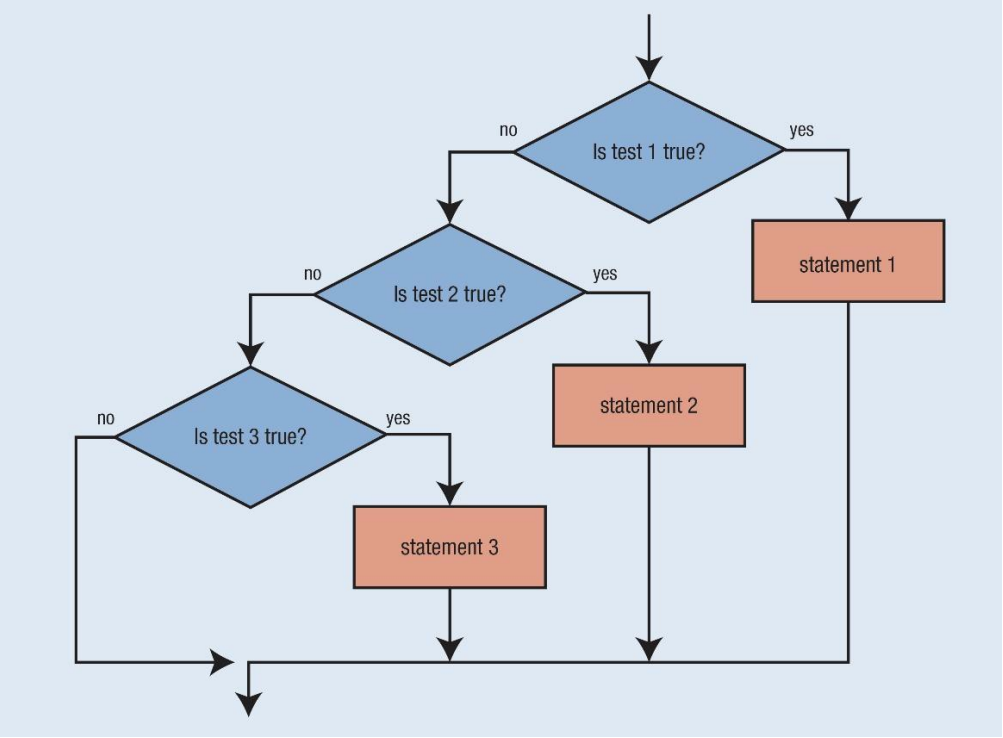
- Exactly 1 path: (mutually exclusive)
  - Syntax:

```
if (test) {
      statement(s); statement(s);
} else if (test) {
      statement(s);
} else {
      statement(s);
      }
```



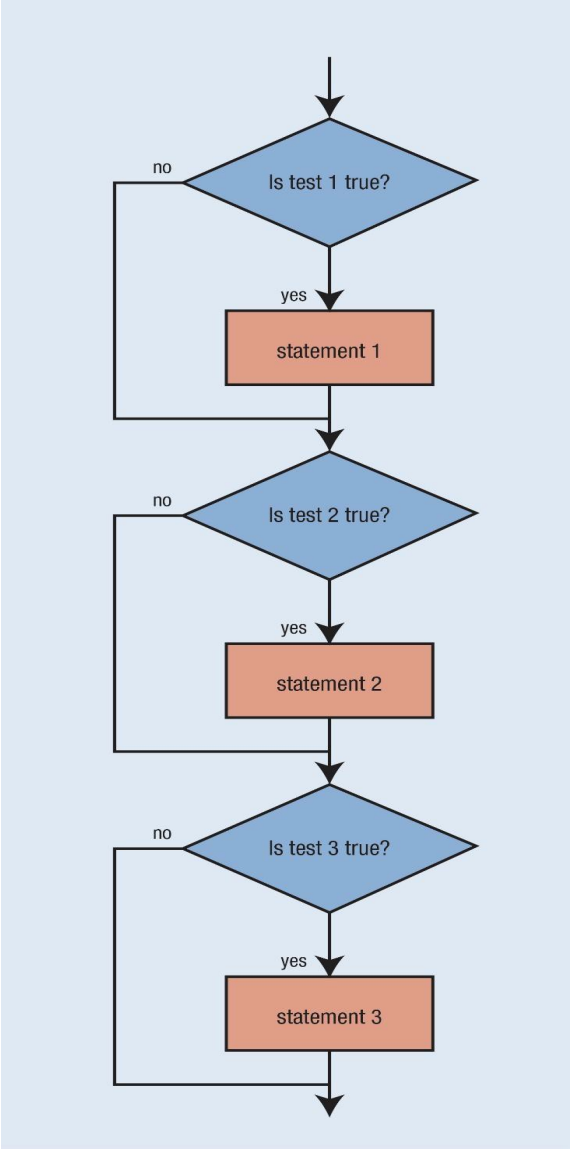
- 0 or 1 path:
  - Syntax:

```
if (test) {
statement(s);
} else if (test) {
statement(s);
} else if (test) {
statement(s);
}
```



- 0, 1, or many paths: (independent tests, not exclusive)
  - Syntax:

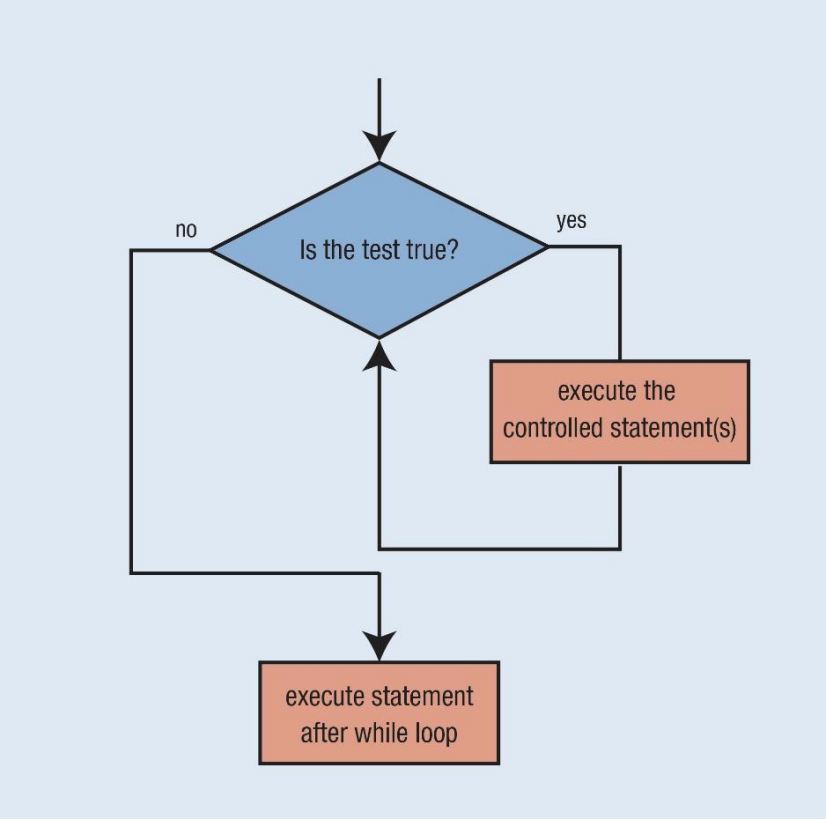
```
if (test) {
      statement(s);
}
if (test) {
      statement(s);
}
if (test) {
      statement(s);
}
```



while loops:

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.
  - Syntax:

```
while (test) {
        statement(s);
}
```



- Example:

```
int num=1;                      // initialization
while (num <=200) {             // test
   System.out.print(num + " ");
   num = num  * 2;              // update
}
```

  - Output: 1 2 4 8 16 32 64 128

The Random class:

- A **`Random`** object generates pseudo-random* numbers.
  - **Class `Random` is found in the `java.util` package.**
    ```
    import java.util.*;
    ```

Method name	Description
`nextInt()`	returns a random integer
`nextInt(max)`	returns a random integer in the range [0, max) in other words, 0 to max-1 inclusive
`nextDouble()`	returns a random real number in the range [0.0, 1.0)

- Example:

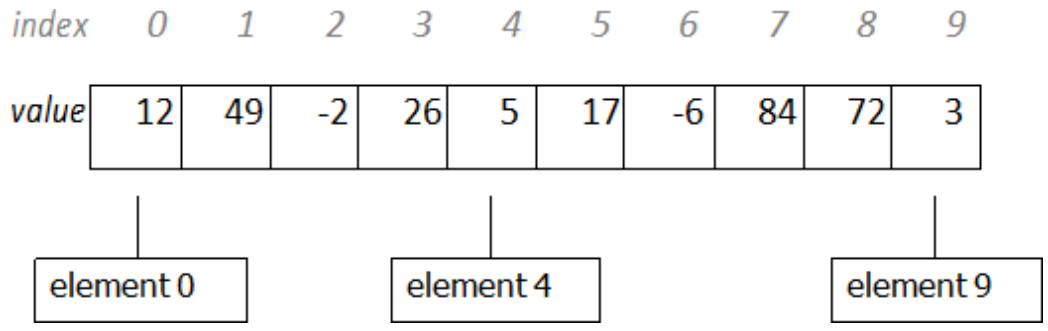
```
Random rand = new Random();
int randomNumber = rand.nextInt(10);      // 0-9
```

break:

- **`break`** statement: Immediately exits a loop.
  - Can be used to write a loop whose test is in the middle.
  - Such loops are often called *"forever" loops* because their header's boolean test is often changed to a trivial `true`.
  - Syntax:
    ```
    while (true) {
      statement(s);
      if (test) {
        break;
      }
      statement(s);
    }
    ```
  - Some programmers consider `break` to be bad style.

Arrays:

- **array**: object that stores many values of the same type.
  - **element**: One value in an array.
  - **index**: A 0-based integer to access an element from an array.



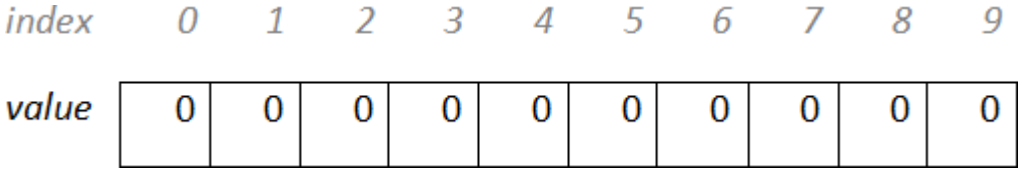
Array declaration:

- Syntax:

```
type[] name = new type[length];
```

- Example:

```
int[] numbers = new int[10];
```



Accessing elements:

- Syntax:

```
name[index]                    // access
name[index] = value;           // modify
```

- Example:

```
numbers[0] = 27;
numbers[3] = -6;
System.out.println (numbers[0]);
if (numbers[3] < 0) {
     System.out.println("Element 3 is negative.");
     }
```

Index	1	2	3	4	5	6	7	8	9
value	27	0	0	-6	0	0	0	0	0

Out-of-bounds:

- Legal indexes: between **0** and the **array's length** − **1**.
  - Reading or writing any index outside this range will throw an `ArrayIndexOutOfBoundsException`.
- Example:

```
int[] data = new int[10];
System.out.println(data[0]);     // okay
System.out.println(data[9]);     // okay
System.out.println(data[-1]);         // exception
System.out.println(data[10]);         // exception
```

The length field:

- An array's `length` field stores its number of elements.
  - Syntax:
    `name``.length`
- Example:

```
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
// output: 0 2 4 6 8 10 12 14
```

- It does not use parentheses like a String's `.length()`.

Quick array initialization:

`type``[]` **`name`** `= {`**`value, value, … value`**`};`

- Example:

```
int[] numbers = {12, 49, -2, 26, 5, 17, -6};
```

Index	0	1	2	3	4	5	6
value	12	49	−2	26	5	17	−6

- Useful when you know what the array's elements will be.
- The compiler figures out the size by counting the values.

The Arrays class:

- Class `Arrays` in package `java.util` has useful static methods for manipulating arrays:

Method name	Description
`binarySearch(`**`array, key`**`)`	returns the index of the given value in a
`equals(`**`array1, array2`**`)`	returns `true` if the two arrays contain the same elements in the same order
`fill(`**`array, value`**`)`	sets every element in the array to have the
`sort(`**`array`**`)`	arranges the elements in the array into
`String toString (`**`array`**`)`	returns a string representing the array, such

Arrays as parameters:

- Declaration:
  `public type methodName(type[] name) {`
- Example:

```
public double average(int[] numbers) {
}
```

- Call:
  `methodName(arrayName);`
- Example:

```
int[] scores = {13, 17, 12, 15, 11};
double avg = average(scores);
```

Arrays as return:

- Declaring:

```
public type[] methodName(parameters) {
```

- Example:

```
public int[] countDigits(int n) {
    int[] counts = new int[10];
    ...
    return counts;
}
```

- Calling:

```
type[] name = methodName(parameters);
```

- Example:

```
public static void main(String[] args) {
    int[] tally = countDigits(229231007);
    System.out.println(Arrays.toString(tally));
}
```

Value semantics (primitives):

- **value semantics**: Behaviour where values are copied when assigned to each other or passed as parameters.
  - When one primitive variable is assigned to another, its value is copied.
  - Modifying the value of one variable does not affect others.

```
int x = 5;
int y = x;    // x = 5, y =5
y = 17;       // x = 5, y = 17
x = 8;
```

Reference semantics (objects):

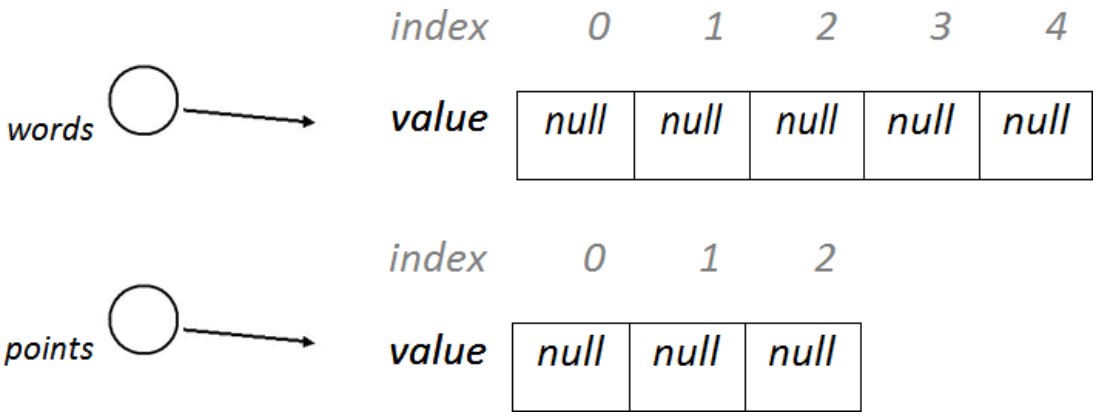
- **reference semantics**: Behaviour where variables actually store the address of an object in memory.
  - When one reference variable is assigned to another, the object is not copied; both variables refer to the same object.
  - Modifying the value of one variable *will* affect others.
- Example:

```
int[] a1 = {4, 5, 2, 12, 14, 14, 9};
int[] a2 = a1; // refer to same array as a1
a2[0] = 7;
System.out.println (a1[0]);       // 7
```

Null:

- **null**: A reference that does not refer to any object.
  - Fields of an object that refer to objects are initialized to `null`.
  - The elements of an array of objects are initialized to `null`.
- Example:

```
String[] words = new String[5];
Point[] points = new Point[3];
```



Null pointer exception:

- **dereference**: To access data or methods of an object with the dot notation, such as `s.length()`.
  - It is illegal to dereference null (causes an exception).
  - `null` is not any object, so it has no methods or data.

```
String[] words = new String[5];
System.out.println("word is: " + words[0]);
words[0] = words[0].toUpperCase();
```

  - Output:

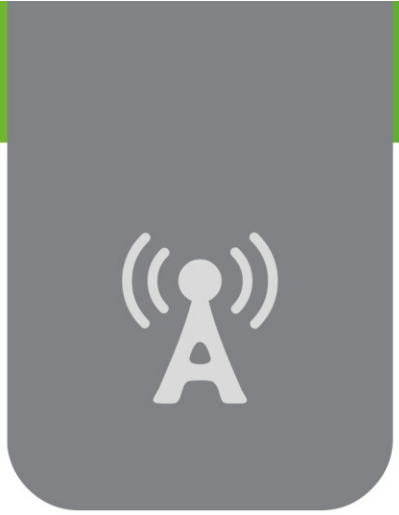
```
word is: null

Exception in thread "main"
java.lang.NullPointerExceptin at Example.main
(Example.java:8)
```

Throwing exceptions:

- Syntax

```
throw new ExceptionType();
throw new ExceptionType("message");
```

- Generates an exception that will crash the program, unless it has code to handle (catch) the exception.
- Common exception types:
  ArithmeticException, ArrayIndexOutOfBoundsException, FileNotFoundException, IllegalArgumentException, IllegalStateException, IOException, NoSuchElementException, NullPointerException, RuntimeException, UnsupportedOperationException



Chapter 2:
Object Oriented Programming in Java

Key Words:

Classes and Objects, Inheritance, Polymorphism.

Summary:

This unit provides an overview of the object oriented programming in Java.

Outcomes:

Student will learn in this unit:

- Declaring classes.
- Using polymorphism and inheritance.

Plan:

3: Learning Objects

1. Object Oriented Programming in Java
2. Inheritance
3. Advanced Topics in OOP

1. Object Oriented Programming in Java

Learning outcomes:

Learning main object oriented programming using Java.

Classes and objects:

- **class**: A program entity that represents either:
  - A program / module, or
  - A template for a new type of objects.
  - The Point class is a template for creating Point objects
- object: An entity that combines state and behavior.
  - object-oriented programming (OOP): Programs that perform their behavior as interactions between objects.

Fields:

- **field**: A variable inside an object that is part of its state.
  - Each object has *its own copy* of each field.
  - **encapsulation**: Declaring fields `private` to hide their data.
- Declaration syntax:

  `private` **`type name`**
- Example:

```
public class Student {
   private String name; // each object now has
   private double gpa;  // a name and gpa field
}
```

Instance methods:

- **instance method**: One that exists inside each object of a class and defines behavior of that object.

```
public type name(parameters) { statements;
}
```

- Example:

```
  public void shout() {
     System.out.println("HELLO THERE!");
  }
```

A Point class:

```
public class Point
{
   private int x;
   private int y;
   // Changes the location of this Point object.
    public void draw(Graphics g) {
        g.fillOval(x, y, 3, 3);
        g.drawString("(" + x + ", " + y + ")", x, y);
    }
}
```

- Each `Point` object contains data fields named `x` and `y`.
- Each `Point` object contains a method named `draw` that draws that point at its current `x/y` position.

The implicit parameter:

- implicit parameter:

  The object on which an instance method is called.

  - During the call `p1.draw(g);`

    the object referred to by p1 is the implicit parameter.

  - During the call `p2.draw(g);`

    the object referred to by `p2` is the implicit parameter.

  - The instance method can refer to that object's fields.

    We say that it executes in the *context* of a particular object.

    `draw` can refer to the `x` and `y` of the object it was called on

Kinds of methods:

- Instance methods take advantage of an object's state.
  - Some methods allow clients to access / modify its state.
- **accessor**: A method that lets clients examine object state.
  - Example: A distanceFromOrigin method that tells how far a Point is away from $(0, 0)$.
  - Accessors often have a non-void return type.
- **mutato**r: A method that modifies an object's state.
  - Example: A `translate` method that shifts the position of a `Point` by a given amount.

Constructors:

- **`constructor`**: Initializes the state of new objects.

```
public type(parameters) {
        statements;
}
```

- Example:

```
public Point(int initialX, int initialY)
        { x = initialX;
y = initialY;
```

  - runs when the client uses the new keyword
  - does not specify a return type; implicitly returns a new object
  - If a class has no constructor, Java gives it a *default constructor* with no parameters that sets all fields to 0.

toString method:

- Tells Java how to convert an object into a `String`

```
public String toString() {
        code that returns a suitable String;
}
```

- Example:

```
public String toString() {
    return "(" + x + ", " + y + ")";
}
```

- Called when an object is printed / concatenated to a `String`:

```
    Point p1 = new Point(7, 2);
    System.out.println("p1: " + p1);
```

- Every class has a `toString`, even if it isn't in your code.
  - Default is class's name and a hex number: `Point@9e8c34`

this keyword:

- **`this`**: A reference to the implicit parameter.
  - *implicit parameter*: object on which a method is called
- Syntax for using `this`:
  - To refer to a field:
    `this.field`
  - To call a method:
    `this.method(parameters);`
  - To call a constructor from another constructor:
    `this(parameters);`

Static methods:

- **static method**: Part of a class, not part of an object.
  - shared by all objects of that class
  - good for code related to a class but not to each object's state
  - does not understand the *implicit parameter*, this; therefore, cannot access an object's fields directly
  - if public, can be called from inside or outside the class
- Declaration syntax:

```
public static type name(parameters) {
statements;
 }
```

2. Inheritance

Learning outcomes: using inheritance:

- **inheritance**: A way to form new classes based on existing classes, taking on their attributes / behavior.
  - a way to group related classes
  - a way to share code between two or more classes
- One class can *extend* another, absorbing its data / behavior.
  - **superclass**: The parent class that is being extended.
  - **subclass**: The child class that extends the superclass and inherits its behavior.
  - Subclass gets a copy of every field and method from superclass

Inheritance syntax:

```
public class name extends superclass {
```

- Example:

```
public class Secretary extends Employee {
...
}
```

Overriding methods

- **override**: To write a new version of a method in a subclass that replaces the superclass's version.
- No special syntax required to override a superclass method. Just write a new version of it in the subclass.

```
public class Secretary extends Employee {
        // overrides getVacationForm in
        Employee public String
        getVacationForm() {
                return "pink";
        }
        ...
}
```

super keyword:

- Subclasses can call overridden methods with super

  `super.`**method(parameters)**

- Example:

```
public class LegalSecretary extends Secretary {
    public double getSalary() {
          double baseSalary =
          super.getSalary();
          return baseSalary + 5000.0;
    }
    ...
```

Polymorphism:

- **polymorphism**: Ability for the same code to be used with different types of objects and behave differently with each.
- Example: `System.out.println` can print any type of object.
  - Each one displays in its own way on the console.
- A variable of type *T* can hold an object of any subclass of T.

  `Employee ed` `= new LegalSecretary();`
  - You can call any methods from `Employee` on ed.
  - You can *not* call any methods specific to `LegalSecretary`.
- When a method is called, it behaves as a `LegalSecretary`.

```
System.out.println(ed.getSalary()); //55000.0
System.out.println(ed.getVacationForm()) //pink
```

3. Advanced Topics in OOP

Learning outcomes: Learning some advanced topics in OOP:

- **inheritance**: Forming new classes based on existing ones.
  - **superclass**: Parent class being extended.
  - **subclass**: Child class that inherits behavior from superclass.
  - gets a copy of every field and method from superclass
- **override**: To replace a superclass's method by writing a new version of that method in a subclass.

```
public class Lawyer extends Employee {
     // overrides getSalary in Employee; a raise!
     public double getSalary() {
            return 55000.00;
     }
}
```

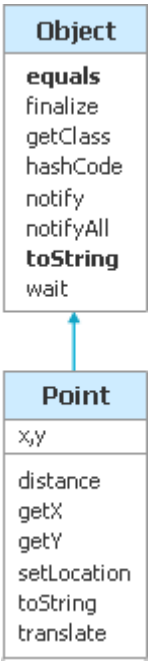
The super keyword:

- Syntax:
  `super.method(parameters)`
  `super (parameters);`

- Subclasses can call overridden methods/constructors with `super`

```
public class Lawyer extends Employee
   { private boolean passedBarExam;
   public Lawyer(int vacationDays, boolean bar)
        { super(vacationDays * 2);
          this.passedBarExam = bar;
   }
   public double getSalary() {
          double baseSalary = super.getSalary();
          return baseSalary + 5000.00;    // $5K raise
   }
   ...
}
```

The class Object:



- The class Object forms the root of the overall inheritance tree of all Java classes.
  - Every class is implicitly a subclass of `Object`
- The `Object` class defines several methods that become part of every class you write. For example:
  - public String `toString()`

  Returns a text representation of the object, usually so that it can be printed.

Object methods:

method	description
`protected Object clone()`	creates a copy of the object
`public boolean equals(Object o)`	returns whether two objects have the same state
`protected void finalize()`	used for garbage collection
`public Class<?> getClass()`	info about the object's type
`public int hashCode()`	a code suitable for putting this object into a hash collection
`public String toString()`	text representation of object

`public void notify()`
`public void notifyAll()`
`public void wait()`
`public void wait()`
	methods related to concurrency and locking

Using the Object class:

- You can store any object in a variable of type Object.

```
Object o1 = new Point(5, -3);
Object o2 = "hello there";
```

- You can write methods that accept an Object parameter.

```
public void
      checkNotNull(Object
      o) { if (o != null) {
            throw new IllegalArgumentException();
   }
```

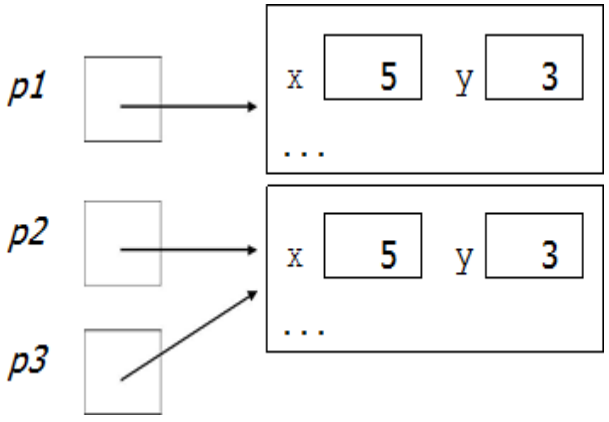
- You can make arrays or collections of `Objects`.

```
Object[] a = new
Object[5]; a[0] = "hello";
a[1] = new Random();
List<Object> list = new ArrayList<Object>();
```

Recall: comparing objects:

- The == operator does not work well with objects.
  - It compares references, not objects' state.
  - It produces true only when you compare an object to itself.

```
Point p1 = new Point(5, 3);
Point p2 = new Point(5, 3);
Point p3 = p2;


// p1 == p2 is false;
// p1 == p3 is false;
// p2 == p3 is true
// p1.equals(p2)?
// p2.equals(p3)?
```



Default equals method:

- The `Object` class's `equals` implementation is very simple:

```
public class Object {
        ...
        public booleanequals(Object
                o) { return this ==
                o;
        }
}
```

- However:
  - When we have used equals with various objects, it didn't behave like == .Why not? `if (str1.equals(str2)) { ...`
  - The Java API documentation for equals is elaborate. Why?

Implementing equals:

- Syntax:

```
public boolean equals(Object name) {
        statement(s) that return a boolean value ;
}
```

- The parameter to `equals` must be of type `Object`.
- Having an `Object` parameter means *any* object can be passed.
  - If we don't know what type it is, how can we compare it?

Casting references:

```
Object o1 = new Point(5, -3);
Object o2 = "hello there";

((Point) o1).translate(6,  2);        // ok
int len = ((String) o2).length();     // ok
Point p = (Point) o1;
int x = p.getX();                     // ok
```

- Casting references is different than casting primitives.
  - Really casting an `Object` reference into a `Point` reference.
  - Doesn't actually change the object that is referred to.
  - Tells the compiler to assume that `o1` refers to a `Point` object.

The instanceof keyword:

- Syntax:

```
if (variable instanceof type) {
        statement(s);
        }
```

  - Asks if a variable refers
  - to an object of a given type.
    - Used as a `boolean` test.

```
String s = "hello";
Point p = new Point();
```

expression	result
`s instanceof Point`	false
`s instanceof String`	true
`p instanceof Point`	true
`p instanceof String`	false
`p instanceof Object`	true
`s instanceof Object`	true
`null instanceof String`	false
`null instanceof Object`	false

equals method for Points:

```
//   Returns whether o refers to a Point object with
//   the same (x, y) coordinates as this Point.
boolean equals(Object o) { if (o instanceof Point) {
            //    o is a Point; cast and compare it Point
            other = (Point) o;
            return x == other.x && y == other.y; } else {
            //    o is not a Point; cannot be equal
            return false;
      }
}
```

More about equals:

- Equality is expected to be reflexive, symmetric, and transitive:

  `a.equals(a)` is true for every object
  `a a.equals(b)` ↔ `b.equals(a)`
  `(a.equals(b) && b.equals(c))` ↔ `a.equals(c)`

- No non-null object is equal to null:

  `a.equals(null)` is false for every object a

- Two sets are equal if they contain the same elements:

```
Set<String> set1 = new HashSet<String>();
Set<String> set2 = new TreeSet<String>();
for(String s: "hi how are you".split(" ")) {
   set1.add(s); set2.add(s);
}
System.out.println(set1.equals(set2));   // true
```

The hashCode method:

`public int hashCode()`

Returns an integer hash code for this object, indicating its preferred to place it in a hash table / hash set.

- Allows us to store non-`int` values in a hash set / map:

```
public static int hashFunction(Object o) {
   return Math.abs(o.hashCode()) % elements.length;
}
```

- How is `hashCode` implemented?
  - Depends on the type of object and its state.
    Example: a String's `hashCode` adds the ASCII values of its letters.
  - You can write your own `hashCode` methods in classes you write.
    All classes come with a default version based on memory address.

Polymorphism:

- **polymorphism**: Ability for the same code to be used with different types of objects and behave differently with each.
- A variable or parameter of type T can refer to any subclass of T.

```
Employee ed = new Lawyer();
Object otto = new Secretary();
```

- When a method is called on `ed`, it behaves as a `Lawyer`.
- You can call any `Employee` methods on `ed`. You can call any `Object` methods on `otto`.
  - You can *not* call any `Lawyer`- only methods on `ed` (e.g. `sue`).
  - You can *not* call any `Employee` methods on `otto` (e.g. `getHours`).

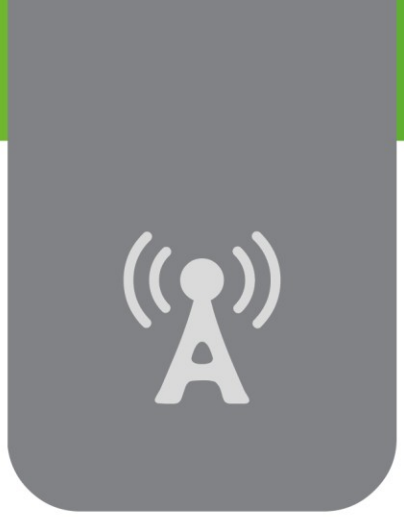
Polymorphism examples:

- You can use the object's extra functionality by casting.

```
Employee ed = new Lawyer();
ed.getVacationDays();                    // ok
otto.getVacationDays();                  // compiler error
((Employee) otto).getVacationDays();     // ok
((Lawyer) otto).sue();                   // runtime error
```

- You can't cast an object into something that it is not.

```
Object otto= new Secretary();
System.out.println(otto.toString());    // ok
otto.getVacationDays();                  // compiler error
((Employee) otto).getVacationDays();     // ok
((Lawyer) otto).sue();                   // runtime error
```



Chapter 3:
Data Structures in Java

Key Words:

ArrayList, HashSet, TreeSet, HashMap, TreeMap

Summary:

This unit provides an overview of different collections data structures used in Java.

Outcomes:

Student will learn in this unit:

- Choosing the appropriate data structure.
- Using different available data structures in Java.

Plan:

1 Learning Object

1. Data Structures in Java

1. Data Structures in Java

Learning outcomes:

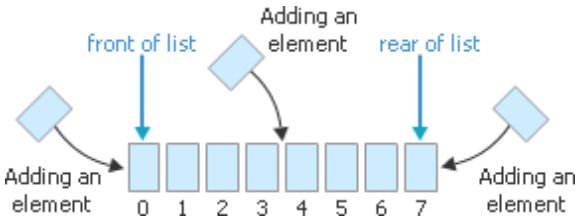
Learning main data structures in Java.

Collections and lists:

- **collection**: an object that stores data ("**elements**")

```
import java.util.*;  // to use Java's collections
```

- **list**: a collection of elements with 0-based **indexes**
  - elements can be added to the front, back, or elsewhere
  - a list has a **size** (number of elements that have been added)
  - in Java, a list can be represented as an **`ArrayList`** object



Idea of a list:

- An ArrayList is like an array that resizes to fit its contents.
- When a list is created, it is initially empty.

  `[]`
- You can add items to the list. (By default, adds at end of list)

`[hello, ABC, goodbye, okay]`

The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
You can add, remove, get, set, ... any index at any time.

Type parameters (generics):

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an `ArrayList`, you must specify the type of its elements in < >
  - This is called a *type parameter* ; `ArrayList` is a *generic* class.
  - Allows the `ArrayList` class to store lists of different types.
- Example:

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty Stepp");
names.add("Stuart Reges");
```

ArrayList methods:

`add(value)`	appends value at end of list
`add(index, value)`	inserts given value just before the given index, shifting subsequent values to the right
`clear()`	removes all elements of the list
`indexOf(value)`	returns first index where given value is found in list (-1 if not found)
`get(index)`	returns the value at given index
`remove(index)`	removes/returns value at given index, shifting subsequent values to the left
`set(index, value)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as "[3, 42, -7, 15]"

ArrayList vs. array:

```
String[] names = new String[5];     // construct
names[0] = "Jessica";               // store
String s = names[0];                // retrieve
for (int i = 0; i < names.length; i++) {
     if (names[i].startsWith("B")) { ... }
}                                   // iterate
```

```
ArrayList<String> list = new ArrayList<String>();
list.add("Jessica");                // store
String s = list.get(0);             // retrieve
for (int i = 0; i < list.size(); i++) {
     if (list.get(i).startsWith("B")) { ... }
}                                     // iterate
```

ArrayList as param / return:

- Syntax:

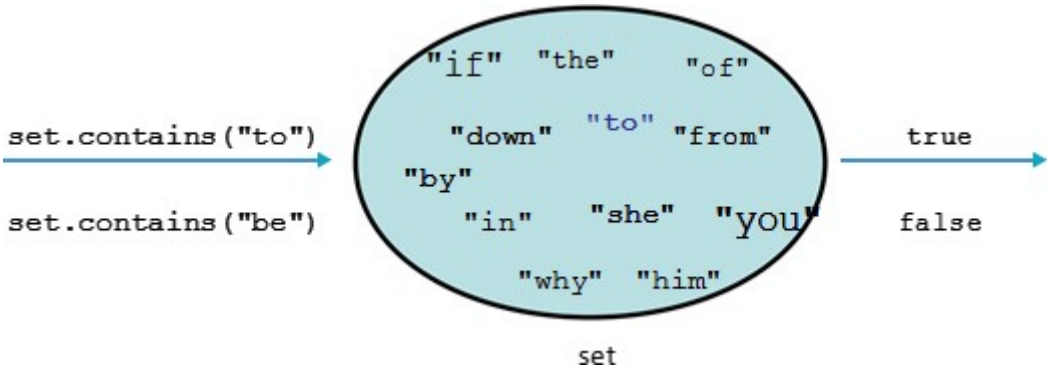
```
Public void name(ArrayList<Type> name) {          // param
Public ArrayList<Type> name(params)               // return
```

- Example:

```
// Returns count of plural words in the given list.
public int countPlural(ArrayList<String> list)
   { int count = 0;
   for    (int i = 0; i < list.size(); i++){
          String str = list.get(i);
              if (str.endsWith("s"))
                  { count++;
              }
        }
        return count;
}
```

Sets:

- **set**: A collection of unique values (no duplicates allowed) that can perform the following operations efficiently:
  - add, remove, search (contains)
  - We don't think of a set as having indexes; we just add things to the set in general and don't worry about order



Set implementation:

- in Java, sets are represented by `Set` type in `java.util`
- `Set` is implemented by `HashSet` and `TreeSet` classes
  - `HashSet`: implemented using a "hash table" array;
  - very fast: **O(1)** for all operations
  - elements are stored in unpredictable order
  - `TreeSet`: implemented using a "binary search tree";
  - pretty fast: **O(log N)** for all operations
  - elements are stored in sorted order

Set methods:

- Example:

```
List<String> list = new ArrayList<String>();
…
Set<Integer> set = new TreeSet<Integer>();      // empty
Set<String> set2 = new HashSet<String>(list);
```

Can construct an empty set, or one based on a given collection

`add(value)`	adds the given value to the set
`contains(value)`	returns `true` if the given value is found in this set
`remove(value)`	removes the given value from the set
`clear()`	removes all elements of the set
`size()`	returns the number of elements in list
`isEmpty()`	returns `true` if the set's size is 0
`toString()`	returns a string such as `"[3, 42, -7, 15]"`

The "for each" loop:

- Syntax:

```
for (type name:
        collection) {
        statements;
}
```

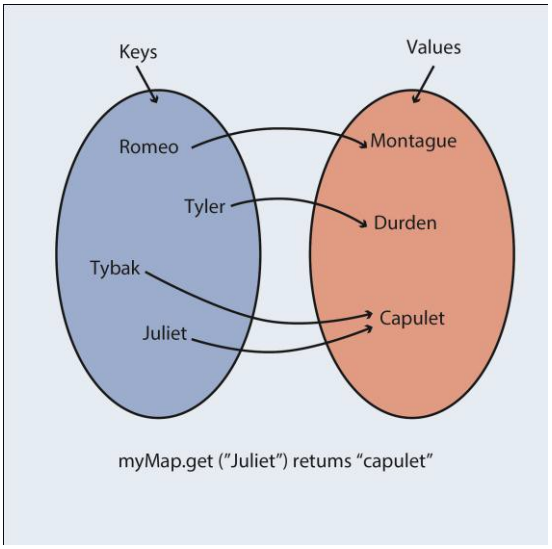
- Provides a clean syntax for looping over the elements of a `Set`, `List`, array, or other collection
- Example:

```
Set<Double> grades = new HashSet<Double>();
...
for (double grade: grades) {
  System.out.println("Student's grade: " + grade);
}
```

  - needed because sets have no indexes; can't `get` element `i`

Maps:

- **map**: Holds a set of unique keys and a collection of *values*, where each key is associated with one value.
  - a.k.a. "dictionary", "associative array", "hash"
- basic map operations:
  - **put** (*key, value*): Adds a mapping from a key to a value.
  - **get** (*key*) : Retrieves the value mapped to the key.
  - **Remove** *(key)*: Removes the given key and its mapped value.



Map implementation:

- in Java, maps are represented by `Map` type in `java.util`
- `Map` is implemented by the `HashMap` and `TreeMap` classes
  - `HashMap`: implemented using an array called a "hash table"; extremely fast: **O(1)**; keys are stored in unpredictable order
  - `TreeMap`: implemented as a linked "binary tree" structure; very fast: **O(log N)** ; keys are stored in sorted order
- A map requires 2 type params: one for keys, one for values.

- Example:

```
// maps from String keys to Integer values
Map<String, Integer> votes = new HashMap<String, Integer>();
```

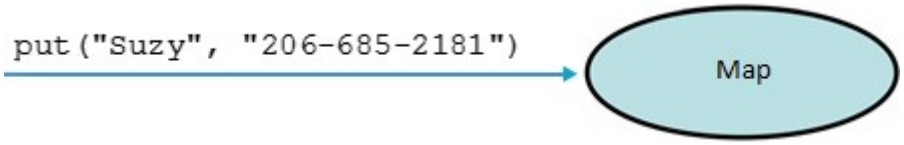
Map methods:

`put(`**`key, value`**`)`	adds a mapping from the given key to the given value; if the key already exists, replaces its value with the given one
`get(`**`key`**`)`	returns the value mapped to the given key (`null` if not found)
`containsKey(`**`key`**`)`	returns `true` if the map contains a mapping for the given key
`remove(`**`key`**`)`	removes any existing mapping for the given key
`clear()`	removes all key/value pairs from the map
`size()`	returns the number of key/value pairs in the map
`isEmpty()`	returns `true` if the map's size is 0
`toString()`	returns a string such as "`{a=90, d=60, c=70}`"
`keySet()`	returns a set of all keys in the map
`values()`	returns a collection of all values in the map
`putAll(`**`map`**`)`	adds all key/value pairs from the given map to this map
`equals(`**`map`**`)`	returns `true` if given map has the same mappings as this one

Using maps:

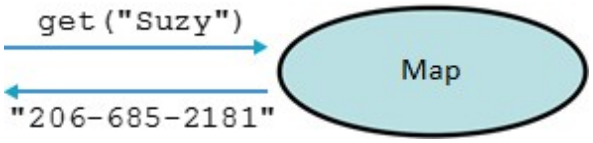
- A map allows you to get from one half of a pair to the other.
  - Remembers one piece of information about every index (key).

```
//      Keyvalue
```



Later, we can supply only the key and get back the related value:

Allows us to ask: `What is Suzy's phone number?`



keySet and values:

- `keySet` method returns a `Set` of all keys in the map
  - can loop over the keys in a foreach loop
  - can get each key's associated value by calling `get` on the map

- Example:

```
Map<String, Integer> ages = new TreeMap<String, Integer>();
ages.put("Marty", 19);
ages.put("Geneva", 2);    // ages.keySet() returns Set<String>
ages.put("Vicki", 57);
for (String name: ages.keySet()) {       // Geneva -> 2
     int age = ages.get(name);                 // Marty -> 19
     System.out.println(name + " -> " + age);  // Vicki -> 57
    }
```

- `values` method returns a collection of all values in the map
  - can loop over the values in a foreach loop
  - no easy way to get from a value to its associated key (s)

The compareTo method:

- The standard way for a Java class to define a `comparison` function for its objects is to define a compareTo method.
  - Example: in the `String` class, there is a method:
    `public int compareTo(String other)`
- A call of **A**`.compareTo(`**B**`)` will return:
  `a value < 0 if` **A** `comes "before"` **B** `in the ordering,`
  `a value > 0 if` **A** `comes "after"` **B** `in the ordering,`
  `or 0 if` **A** `and` **B** `are considered "equal" in the ordering`

compareTo:

- `compareTo` can be used as a test in an `if` statement.

Primitives	Objects
`if (a < b) { ...`	`if (a.compareTo(b) < 0) { ...`
`if (a <= b) { ...`	`if (a.compareTo(b) <= 0) { ...`
`if (a == b) { ...`	`if (a.compareTo(b) == 0) { ...`
`if (a != b) { ...`	`if (a.compareTo(b) != 0) { ...`
`if (a >= b) { ...`	`if (a.compareTo(b) >= 0) { ...`
`if (a > b) { ...`	`if (a.compareTo(b) > 0) { ...`

- Example:

```
String a = "alice";
String b = "bob";
if (a.compareTo(b) < 0) {      // true
        ...
}
```

compareTo and collections:

- You can use an array or list of strings with Java's included binary search method because it calls `compareTo` internally.

  Example:

```
String[] a = {"al", "bob", "cari", "dan", "mike"};
int index = Arrays.binarySearch(a, "dan"); // 3
```

- Java's `TreeSet` / `Map` use `compareTo` internally for ordering.

  Example:

```
Set<String> set = new TreeSet<String>();
for (String s: a) {
     set.add(s);
 }
 System.out.println(s);
 // [al, bob, cari, dan, mike]
```

Ordering our own types:

- We cannot binary search or make a `TreeSet / Map` of arbitrary types, because Java doesn't know how to order the elements.
  - The program compiles but crashes when we run it.

  Example:

```
Set<HtmlTag> tags = new TreeSet<HtmlTag>();
tags.add(new HtmlTag("body", true));
tags.add(new HtmlTag("b", false));
...
```

```
Exception in thread "main" java.lang.ClassCastException
at java.util.TreeSet.add(TreeSet.java:238
```

Comparable:

```
public interface Comparable<E> {
    public int compareTo(E other);
}
```

- A class can implement the `Comparable` interface to define a natural ordering function for its objects.
- A call to your `compareTo` method should return:

```
a value < 0 if this object comes "before" the other object,
a value > 0 if this object comes "before" the other object,
Or 0 if this object is considered "equal" to the other
```

Comparable example:

```
public class Point implements Comparable<Point> {
     private int x;
      private int y;
...
     // sort by x and break ties by y
     public int compareTo(Point other) {
          if (x < other.x) {
               return -1;
          } else if (x > other.x) { return 1;
          } else if (y < other.y) {
          return -1;      // same x, smaller y
          } else if (y > other.y) {
          return 1; // same x, larger y
          } else {
          return 0; // same x and same y
          }
     }
```

Collections class:

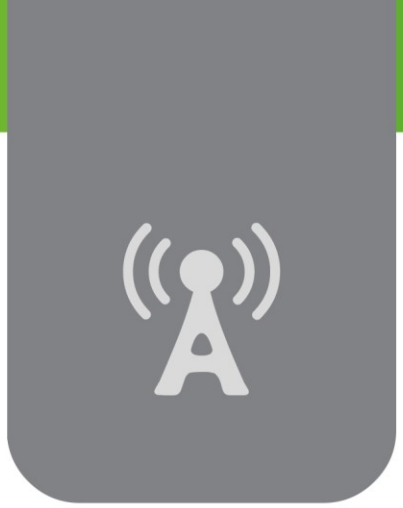
Method name	Description
`binarySearch(list, value)`	returns the index of the given value in a sorted list (< 0 if not found)
`copy(listTo, listFrom)`	copies **listFrom**'s elements to **listTo**
`emptyList(), emptyMap(), emptySet()`	returns a read-only collection of the given type that has no elements
`fill(list, value)`	sets every element in the list to have the given value
`max(collection), min(collection)`	returns largest/smallest element
`replaceAll(list, old, new)`	replaces an element value with another
`reverse(list)`	reverses the order of a list's elements
`shuffle(list)`	arranges elements into a random order
`sort(list)`	arranges elements into ascending order

Sorting methods in Java:

- The `Arrays` and `Collections` classes in `java.util` have a static method `sort` that sorts the elements of an array / list

Example

```
// sorting array elements
String[] words = {"foo", "bar", "baz", "ball"};
Arrays.sort(words);
System.out.println(Arrays.toString(words));
// [ball, bar, baz, foo]

List<String> words2 = new
ArrayList<String>();
for (String word: words) {
        words2.add(word);
}
Collections.sort(words2);
System.out.println(words2);
// [ball, bar, baz, foo]
```



Chapter 4:
Developing Android applications using Android Studio

Key Words:

Android Studio, Activity, View, Layout, Widgets, Events.

Summary:

This unit shows the basics of developing Android applications using Android Studio. We examine the structure of the application and its various components. We also show how to use Virtual Devices.

Outcomes:

Student will learn in this unit:

- Working with Android Studio.
- Building simple application: Hello World.
- Building simple application: How old am I.

Plan:

1. Learning objects
    a. Android Studio
    b. Example: How old am i?

1. Android Studio

Learning outcomes:

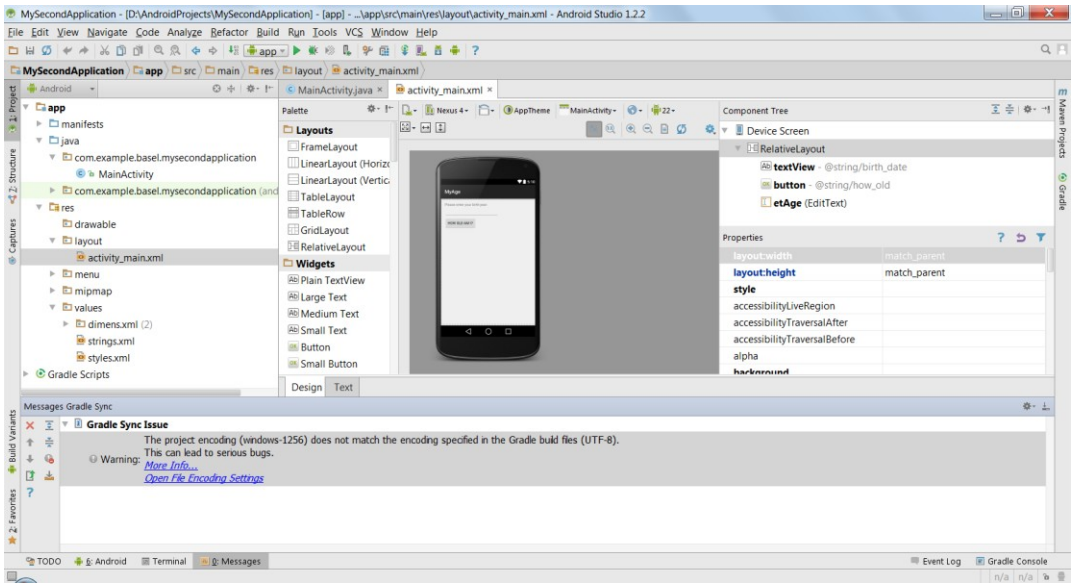
Creating mobile applications using Android Studio

What is Android?

- Mobile operating system maintained by **Google**
- Originally purchased from Android, Inc. in 2005
- Runs on phones, tablets, watches, TVs, …
- Based on **Java** (development language) and **Linux** (kernel)
- The #1 mobile OS worldwide
- Now #1 overall OS worldwide!
- Has over 1 million apps published in Play Store
- Code is released as open source (periodically)
- Easier to customize, license, pirate, etc. than iOS

What is Android Studio?

- Google's official Android IDE, in v1.0 as of November 2014
  - Replaces previous Eclipse–based environment
  - ased on IntelliJ IDEA editor; free to download and use

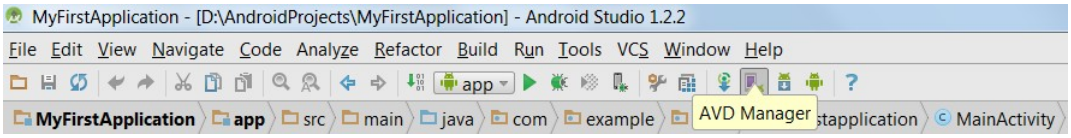


Virtual Devices:

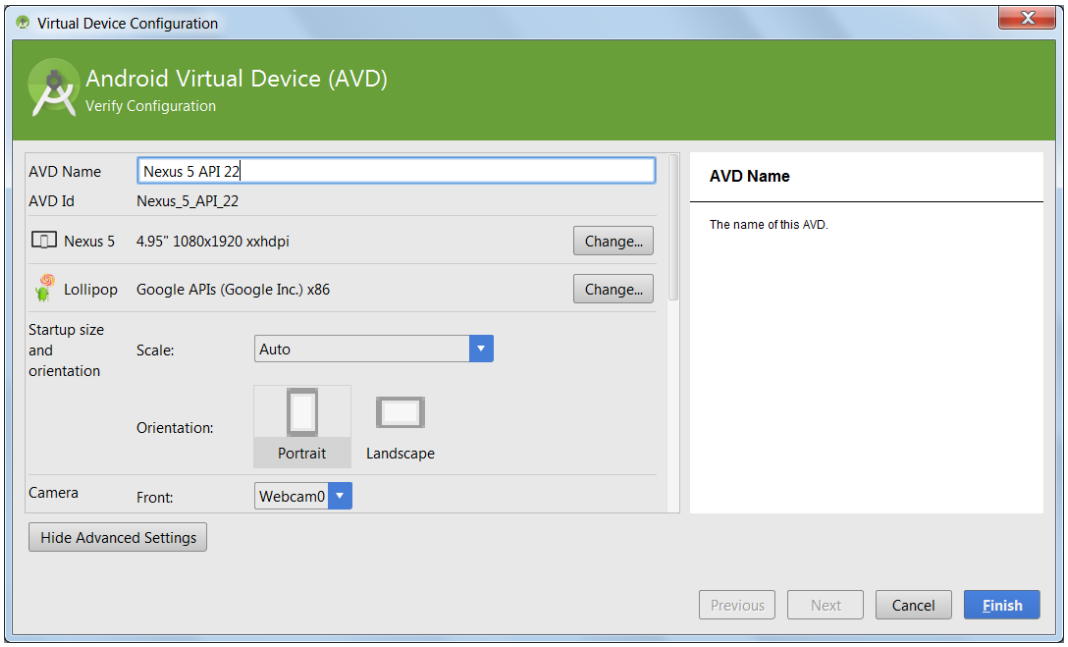
- Allows you to run your project in an emulator
  - A software simulation of an entire Android tablet, phone, watch.
  - When you click the "Run" button in Android Studio, it builds your app, installs it on the virtual device, and loads it.
- Must set up virtual device first in Android Studio.
- Alternative: install your app on your actual Android device!
  - Advantage: app will run faster, better test of real execution.
  - Disadvantage: requires Android device, must be plugged into the PC.

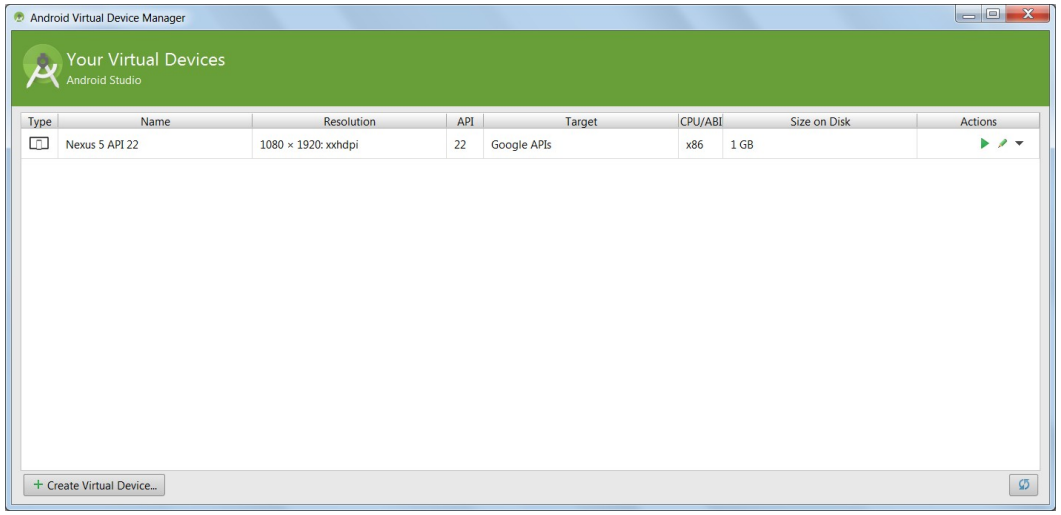
Creating a virtual device within Android Studio:

1. Open Android Studio.
2. Open the AVD manger:



3. Create your virtual device:

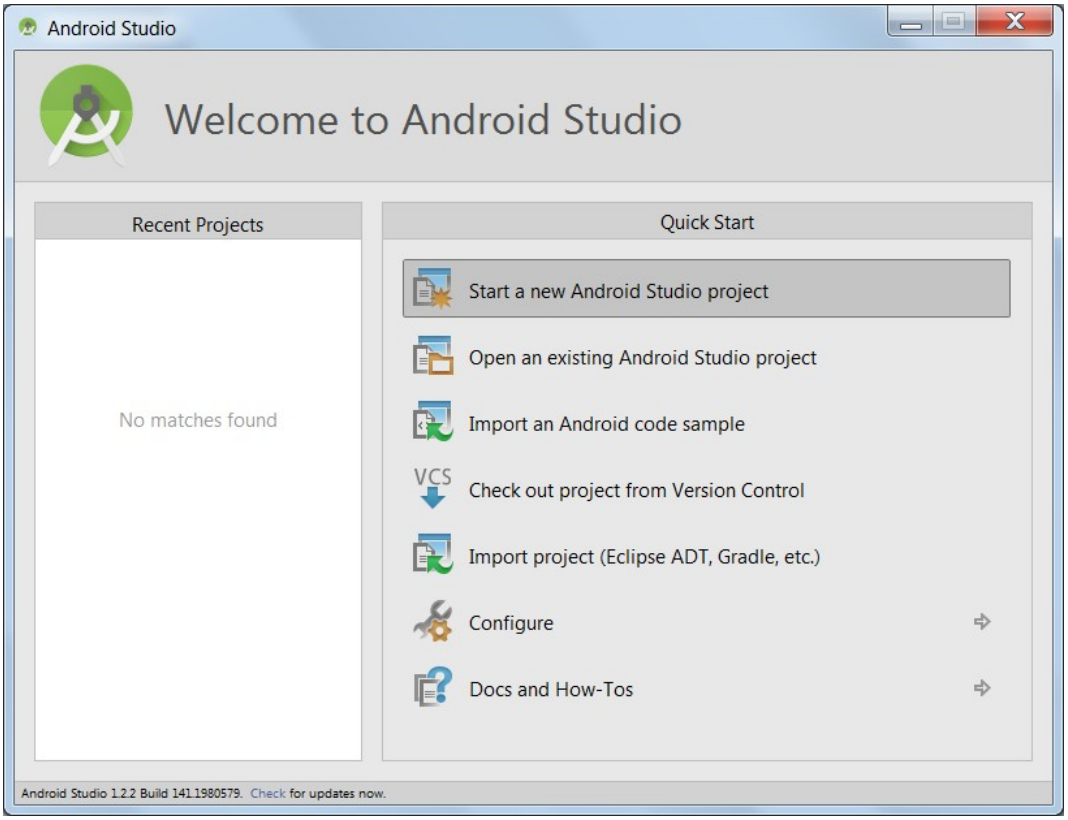




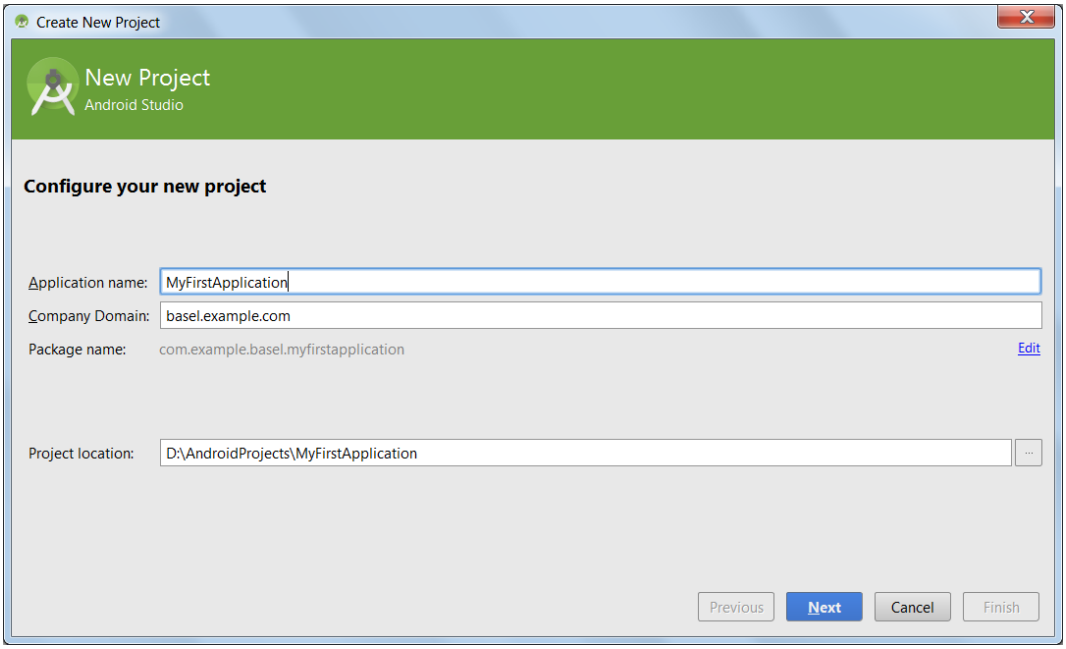
Creating new project using Android Studio:

- The Android Studio environment allows creating applications simply and quickly as the following steps show:

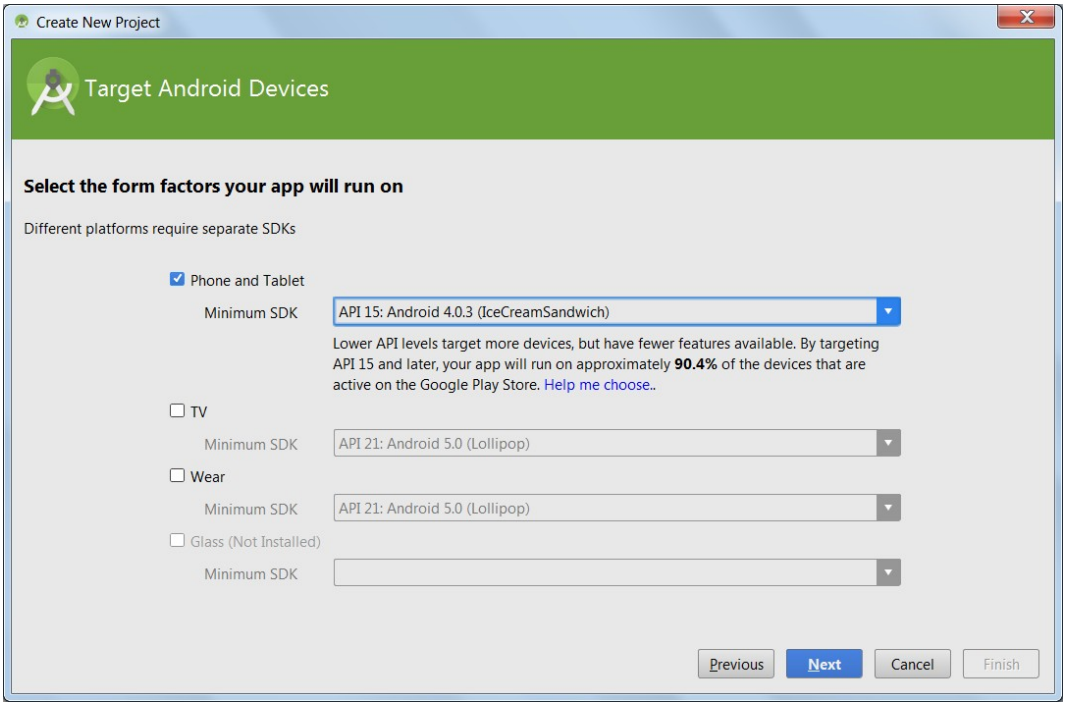
1. Open Android Studio
2. Start a new Android Studio project



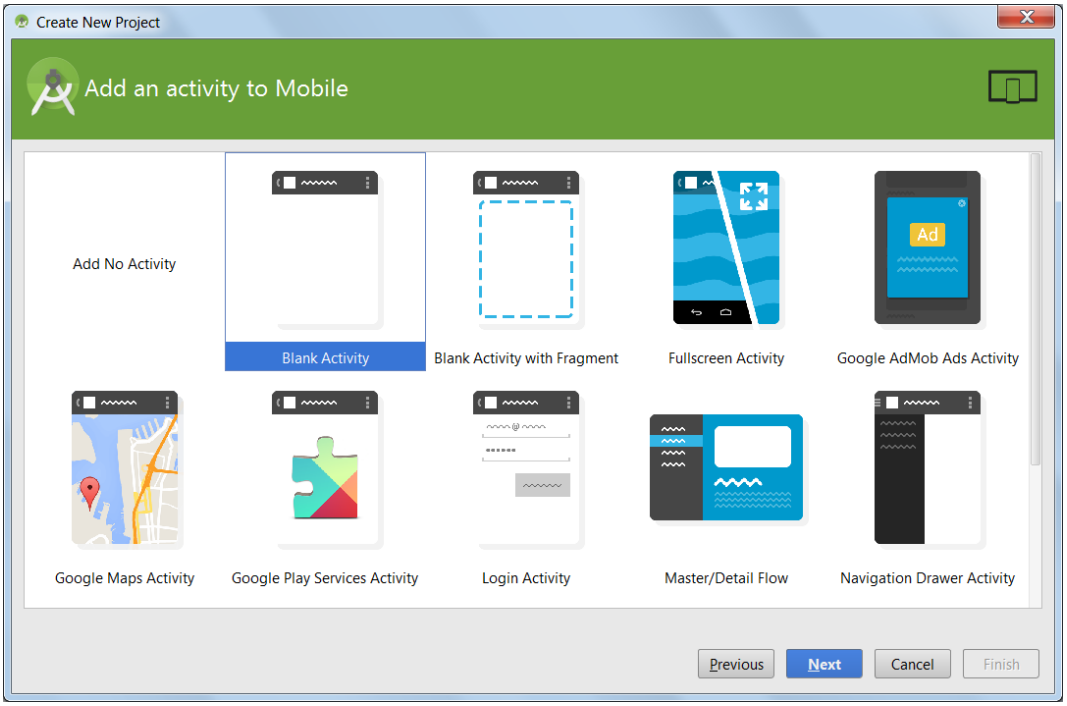
3. Select project location and name the project:



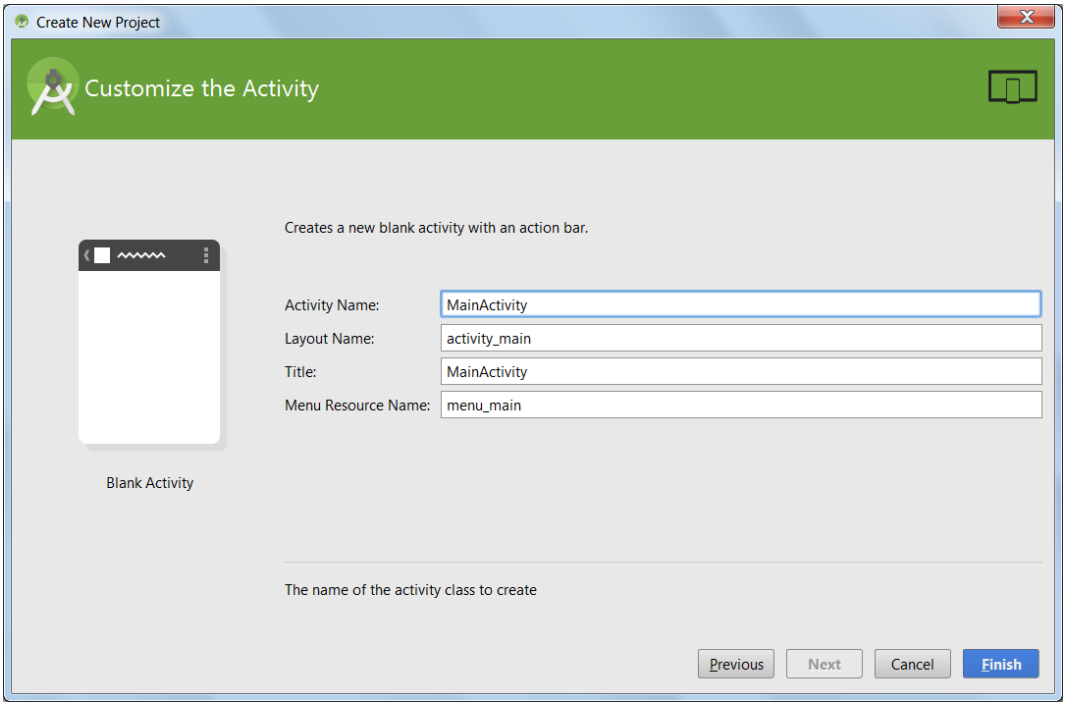
4. Keep the default setting to create phone project:



5. Select the default option to create Blank Activity:



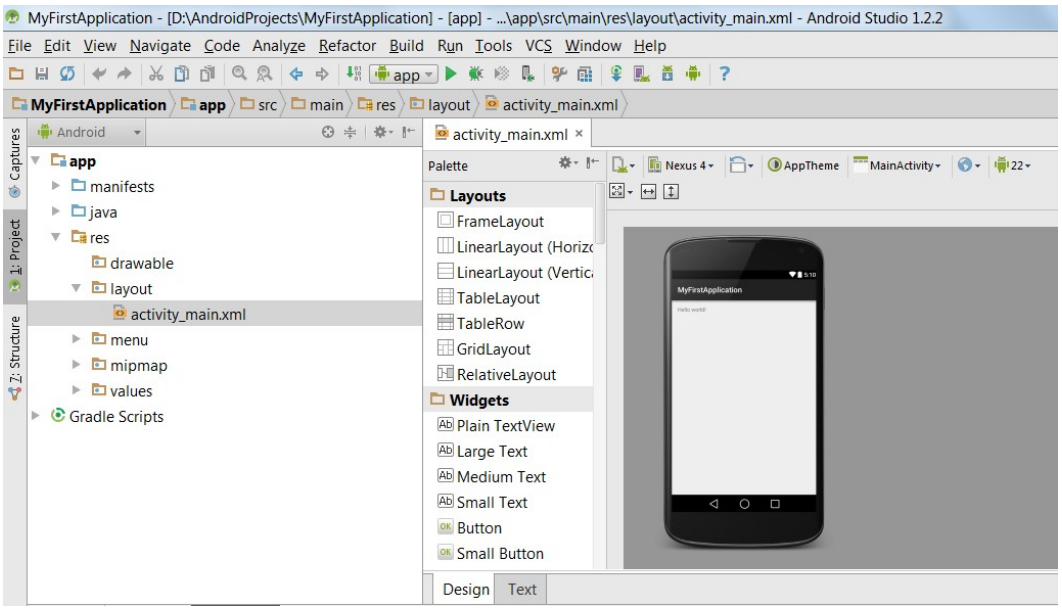
6. Keep the default names for the Activity, the Layout, the Title and the Menu Resource.



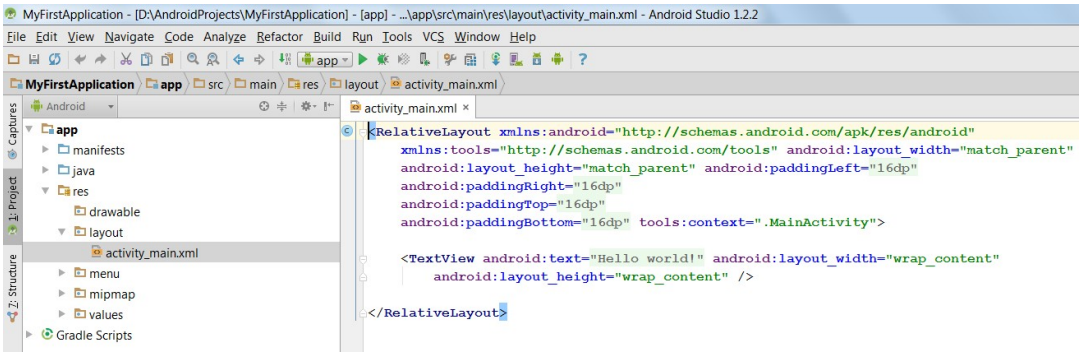
7. Open the activity design file in design view:

app / res / layout / activity_main.xml

(You might need to run the application once at least)



8. You can preview the previous file in XML format:

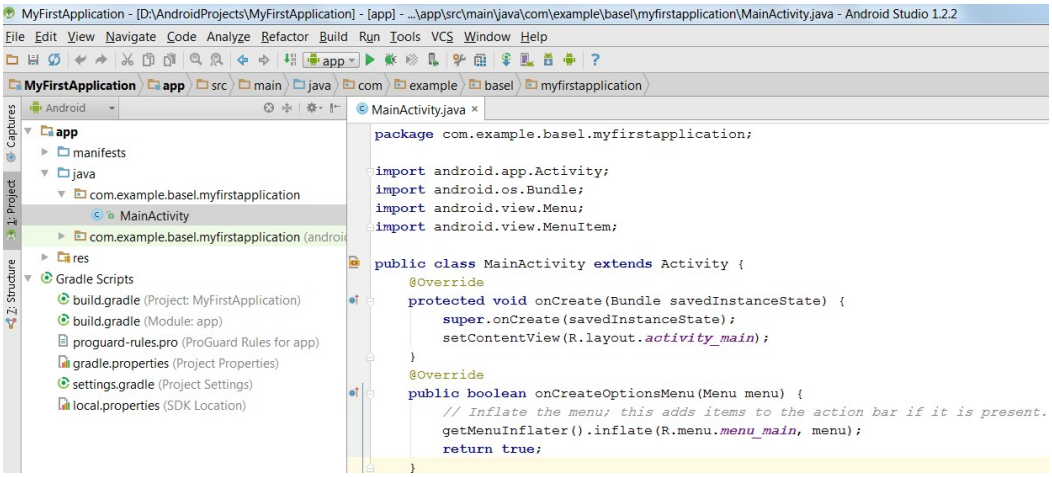



```
RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    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>
```

- The RelativeLayout specifies the layout type (positions of widgets).
- The TextView widget is used to display a text.

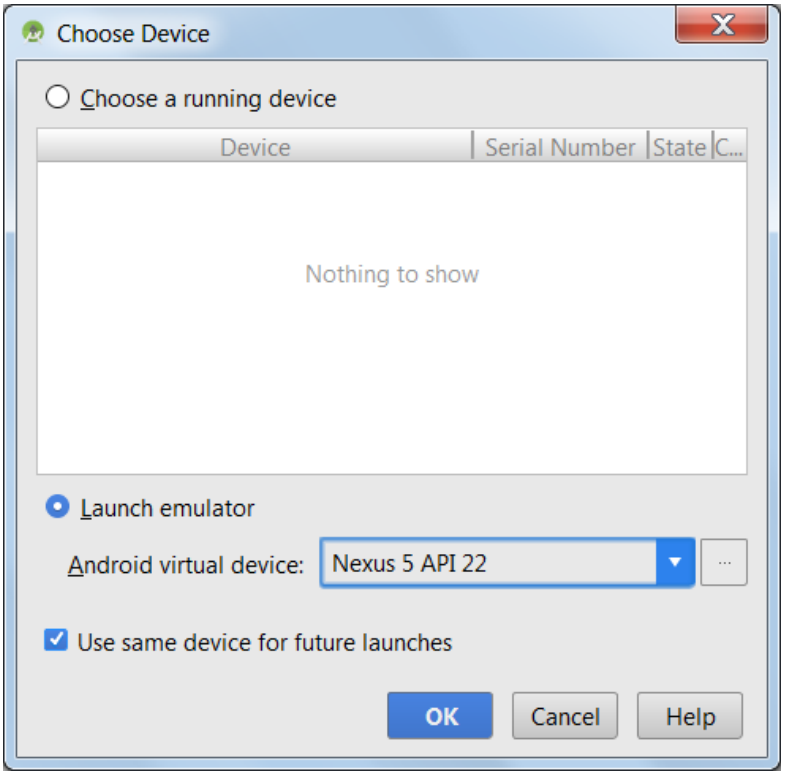
9. Open the activity code file:

app/java/MainActivity.java

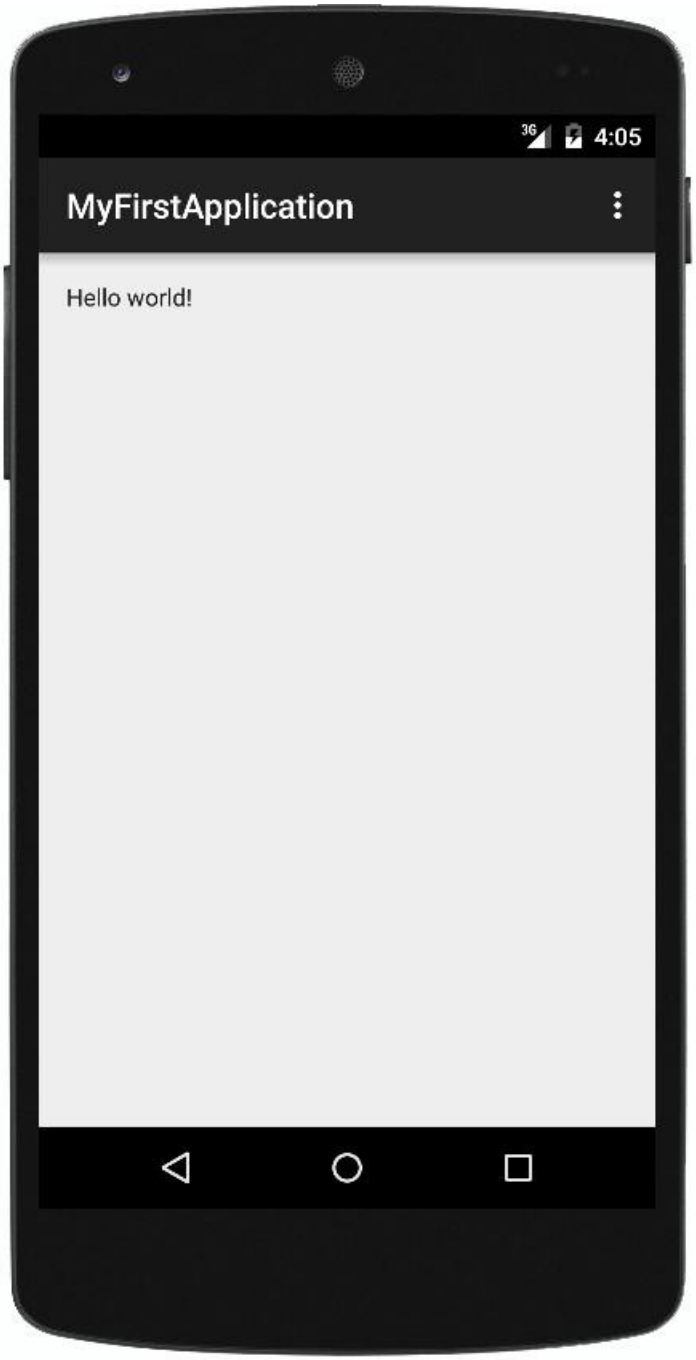


- Note that the call of the `setContentView` method is used to display the activity_main activity.

10. Run to display the Choose Device window:

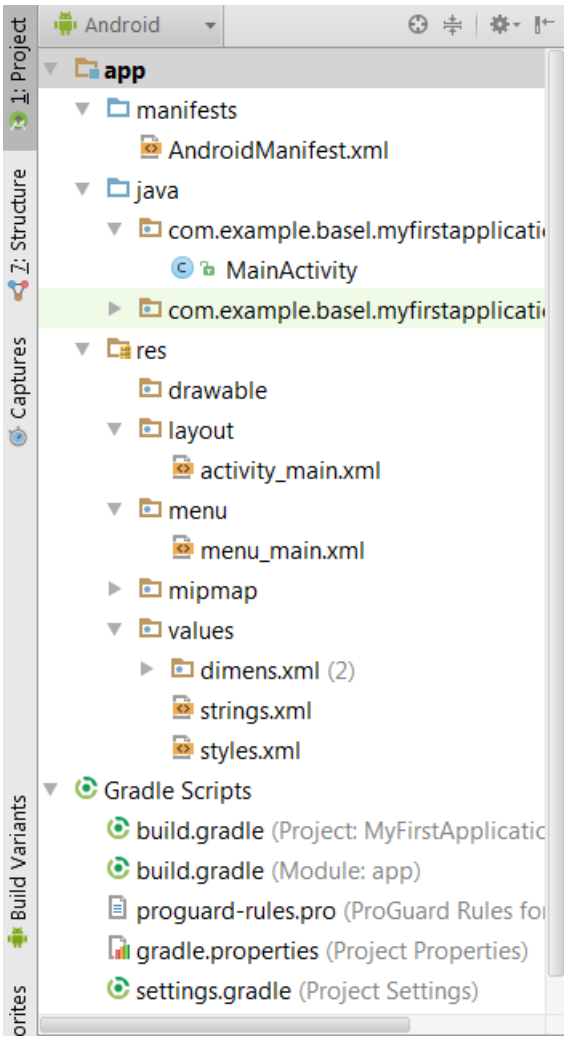


11. The execution output is a simple interface that shows a welcome message



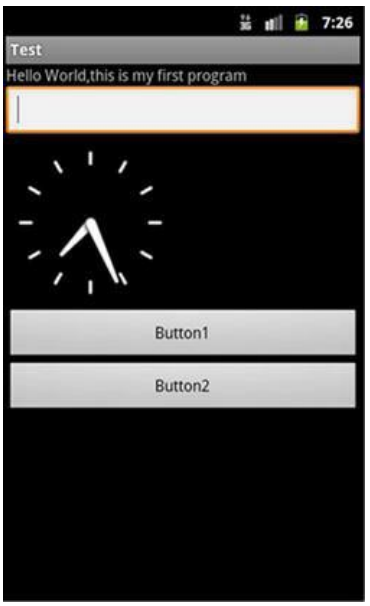
Android Project Structure:

- **AndroidManifest.xml**
  - overall project configuration and settings
- **src / java / .**
  - source code for your Java classes
- **res / … = resource files (many are XML)**
  - drawable/ = images
  - layout / = descriptions of GUI layout
  - menu / = overall app menu options
  - values / = constant values and arrays
  - strings = localization data
  - styles = general appearance styling
- **Gradle**
  - a build/compile management system
  - **build.gradle** = main build configuration file



Android terminology:

- **activity:** a single screen of UI that appears in your app
  - the fundamental units of GUI in an Android app
- **view:** items that appear onscreen in an activity
  - **widget:** GUI control such as a button or text field
  - **layout:** invisible container that manages positions/sizes of widgets
- **event:** action that occurs when user interacts with widgets
  - e.g. clicks, typing, scrolling
- **action bar:** a menu of common actions at top of app
- **notification area:** topmost system menu and icons

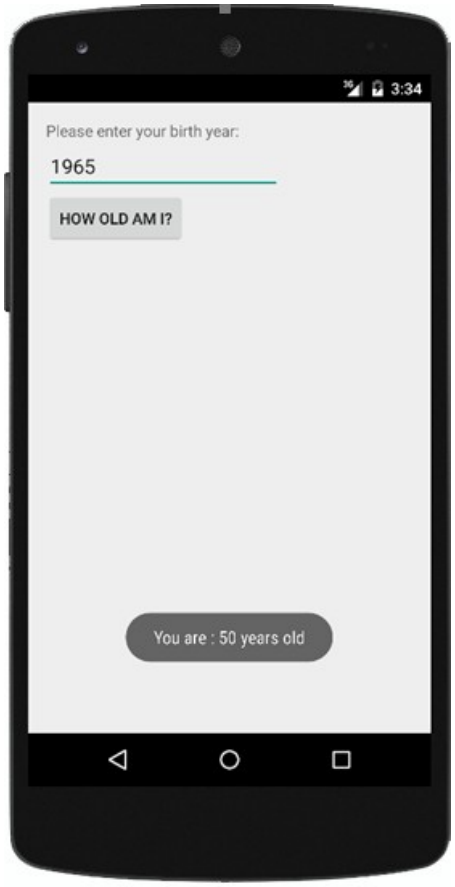


2. Example: How old am i ?

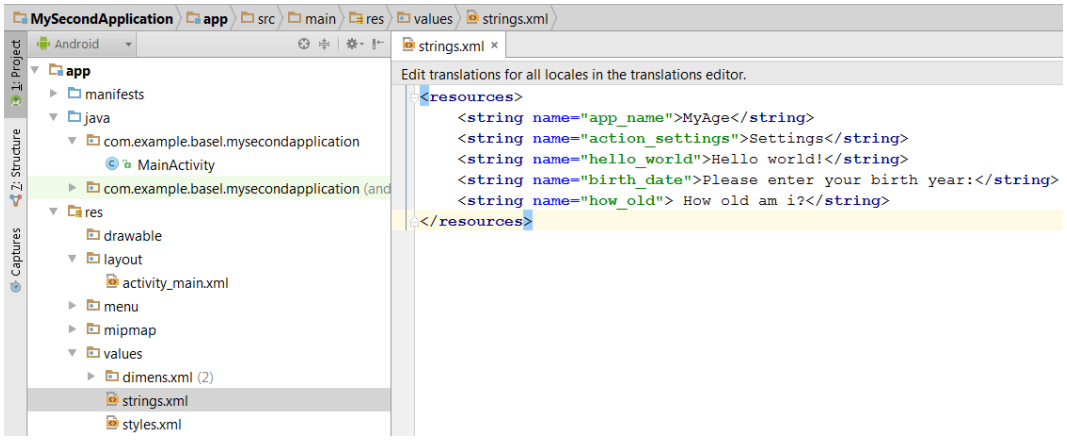
Learning outcomes:

Using main widgets and writing events.

We will modify the default application created by Android Studio (Hello World) to build an interface where the user can enter the year of his



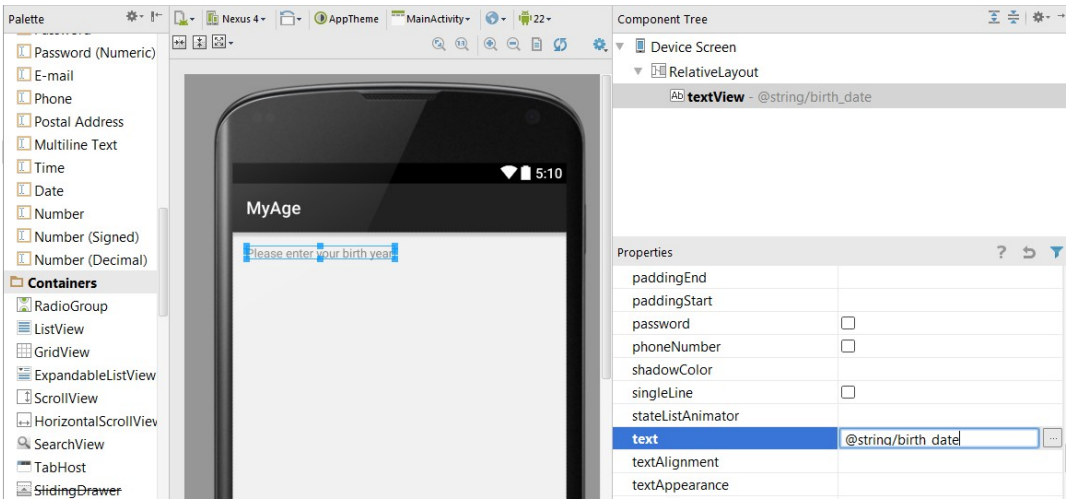
1. Create the default application (Hello World).
2. Open the constants file (app/res/values/strings.xml) and declare the following constants that will be used with the widgets:



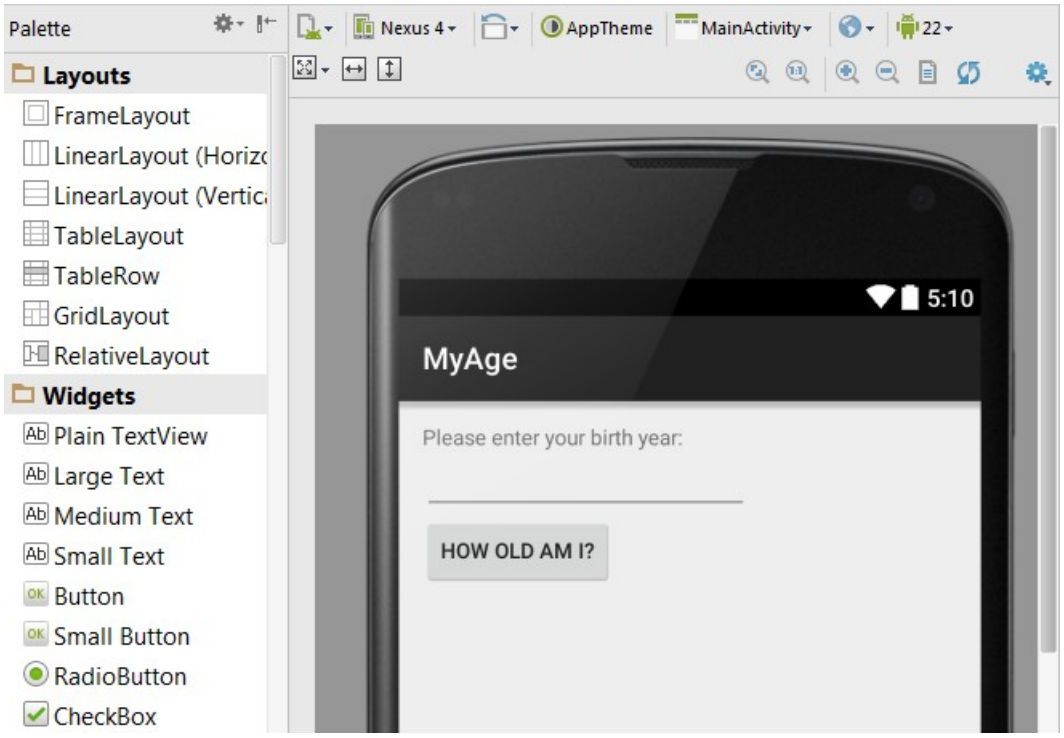

```
<resources>
    <string name="app_name">MyAge</string>
    <string
name="action_settings">Settings</string>
    <string name="hello_world">Hello
world!</string>
    <string name="birth_date">Please enter your
birth year:</string>
    <string name="how_old"> How old am i?</string>
</resources>
```

3. Do the following updates in the activity design file (app/res/layout/activity_main.xml):

- Adjust the text property of the TextView to @string/birth_date.



- Note the use of the prefix @string to the declared constants in the strings.xml file.
- Use the Palette to add text field (Text Fields/Number) and adjust id=@+id/etAge
- Note the use of the prefix @+id/ that asks to add a new resource to the resources file (R.java).
- Use the Palette to add a button with text: @string/how_old.



4. Open the activity code file app / java / MainActivity and write the following method:

```
public void DisplayAge(View v) {
    EditText etAge;
    etAge = (EditText) findViewById(R.id.etAge);
    int y;
    // check if the field is empty
    if (TextUtils.isEmpty(etAge.getText().toString())) {
        return;
    }

    y = Integer.parseInt  (etAge.getText().toString());
    int year = Calendar.getInstance().get(Calendar.YEAR);
    y= year -y ;

    Toast.makeText(this, "You are : " + y + " years old",
Toast.LENGTH_SHORT).show();
}
```

- Note the use of the method `findViewById` with R.id.*your_unique_ID to get a reference to a widget.*
- Note the use of the *`getText`* () to get the text contained in the widget.
- To get the current date, we use:

```
Calendar.getInstance().get(Calendar.YEAR);
```

Displaying pop–up messages:

The Toast pop–up message appears for a while and is suitable for showing messages in response to certain events.

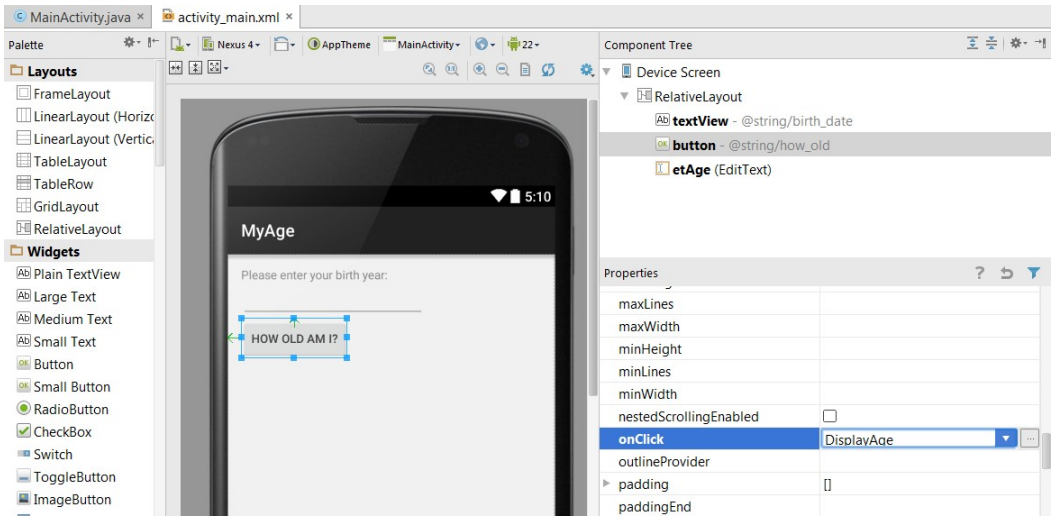
- Use:

```
Toast.makeText(this,"message",duration).show()
```

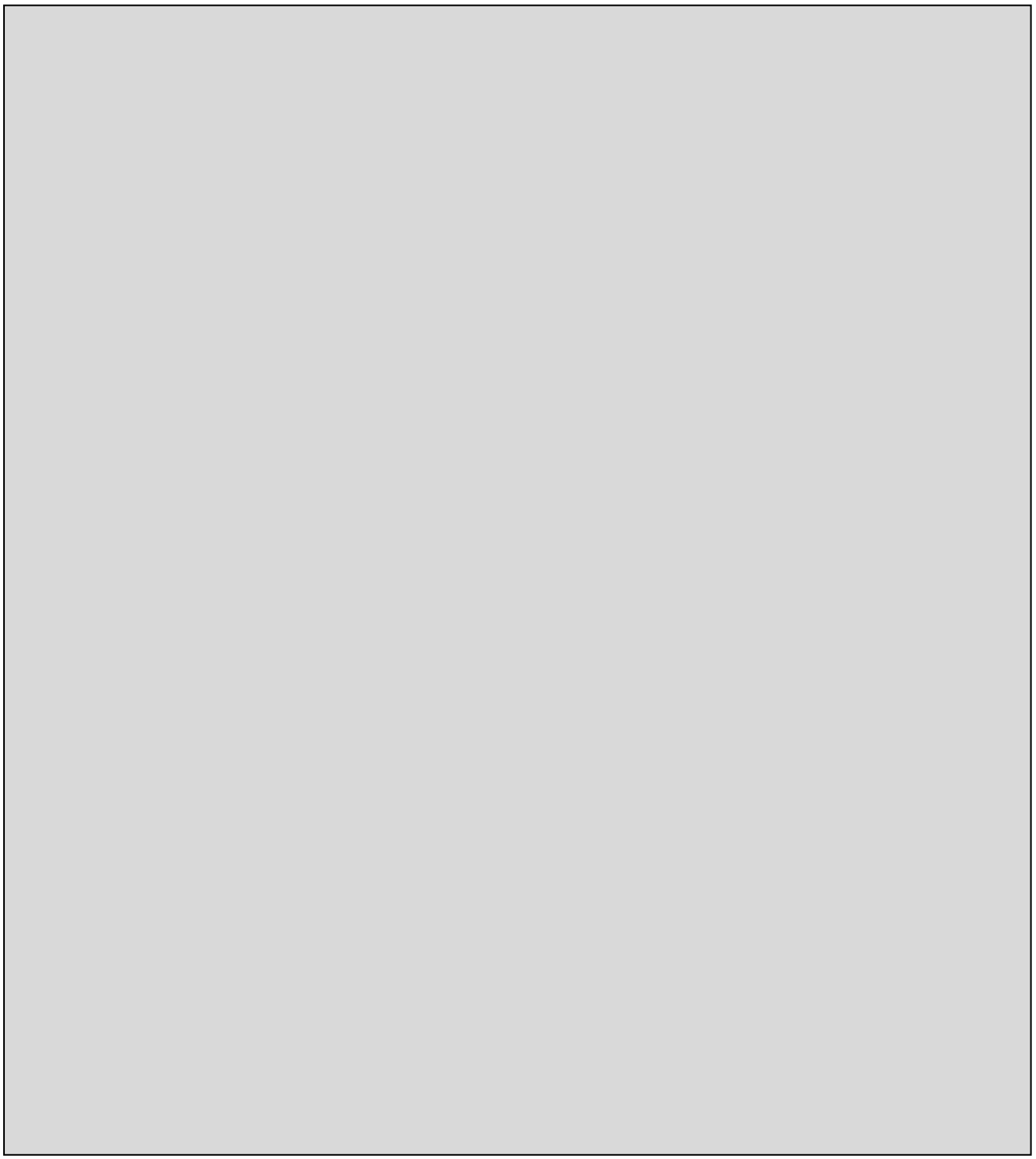
Where the duration parameter can be *`Toast.LENGTH_SHORT`* or *`Toast.LENGTH_LONG`*.

- Use the *onClick* property to call the previous method on the click event of the button:

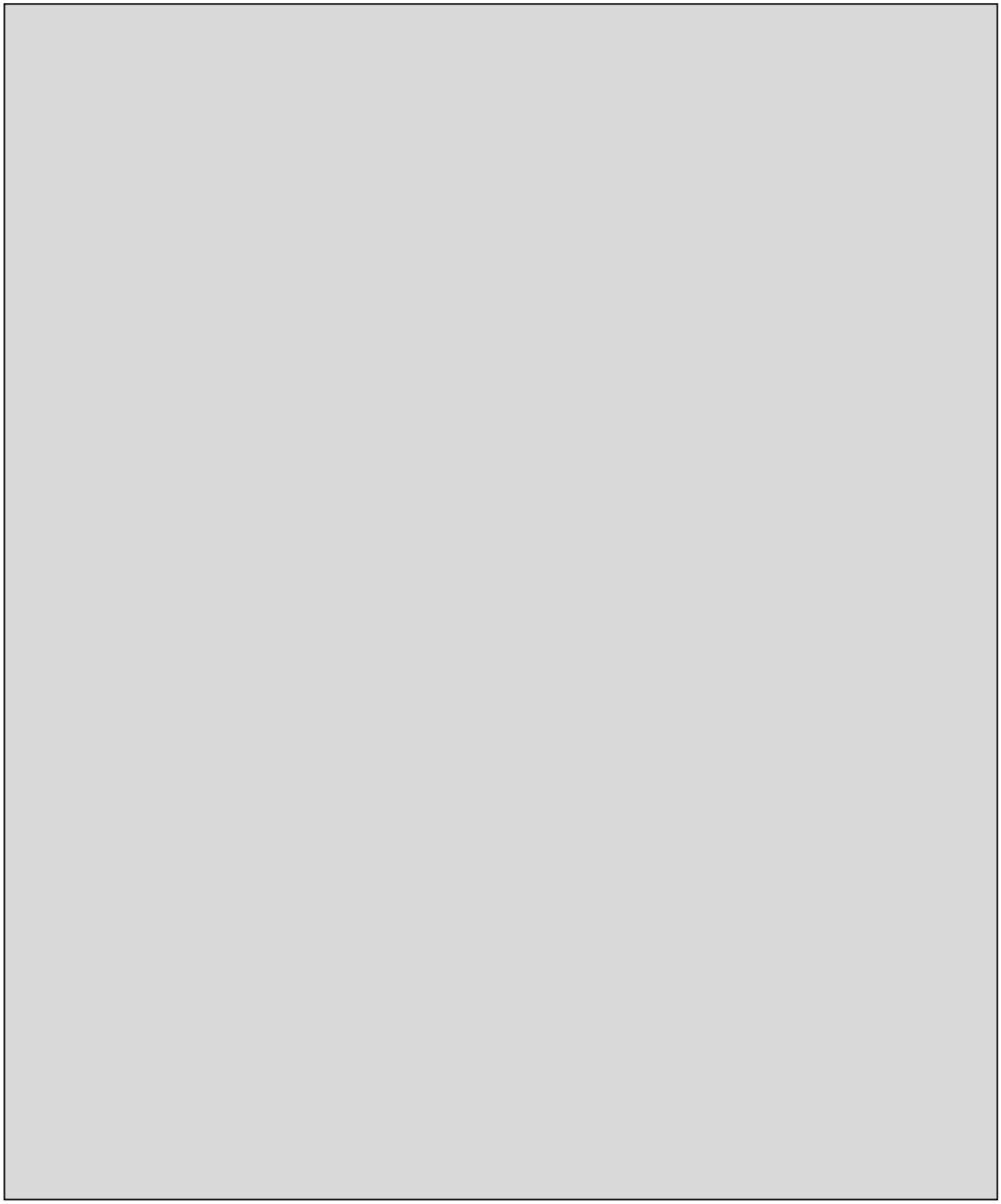
```
onClick: DisplayAge
```



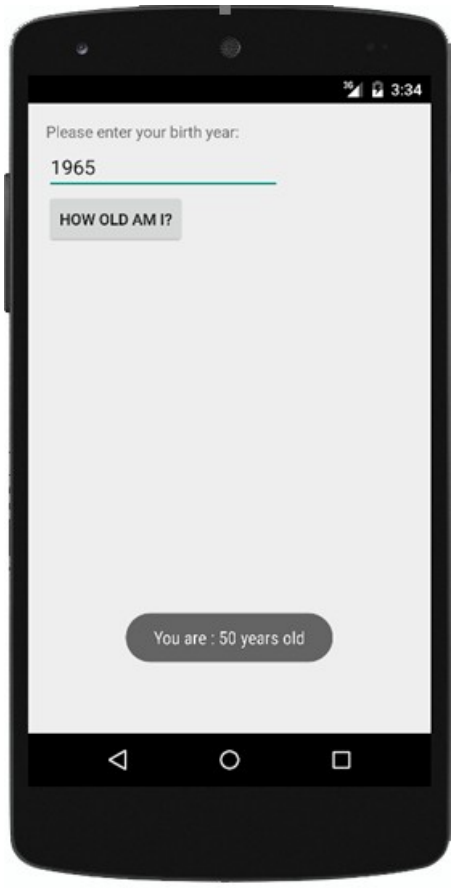
The activity design file *activity_main.xml* will be:



The activity code file *MainActivity.java* is:



Running the application shows for example:



Assigning methods to events:

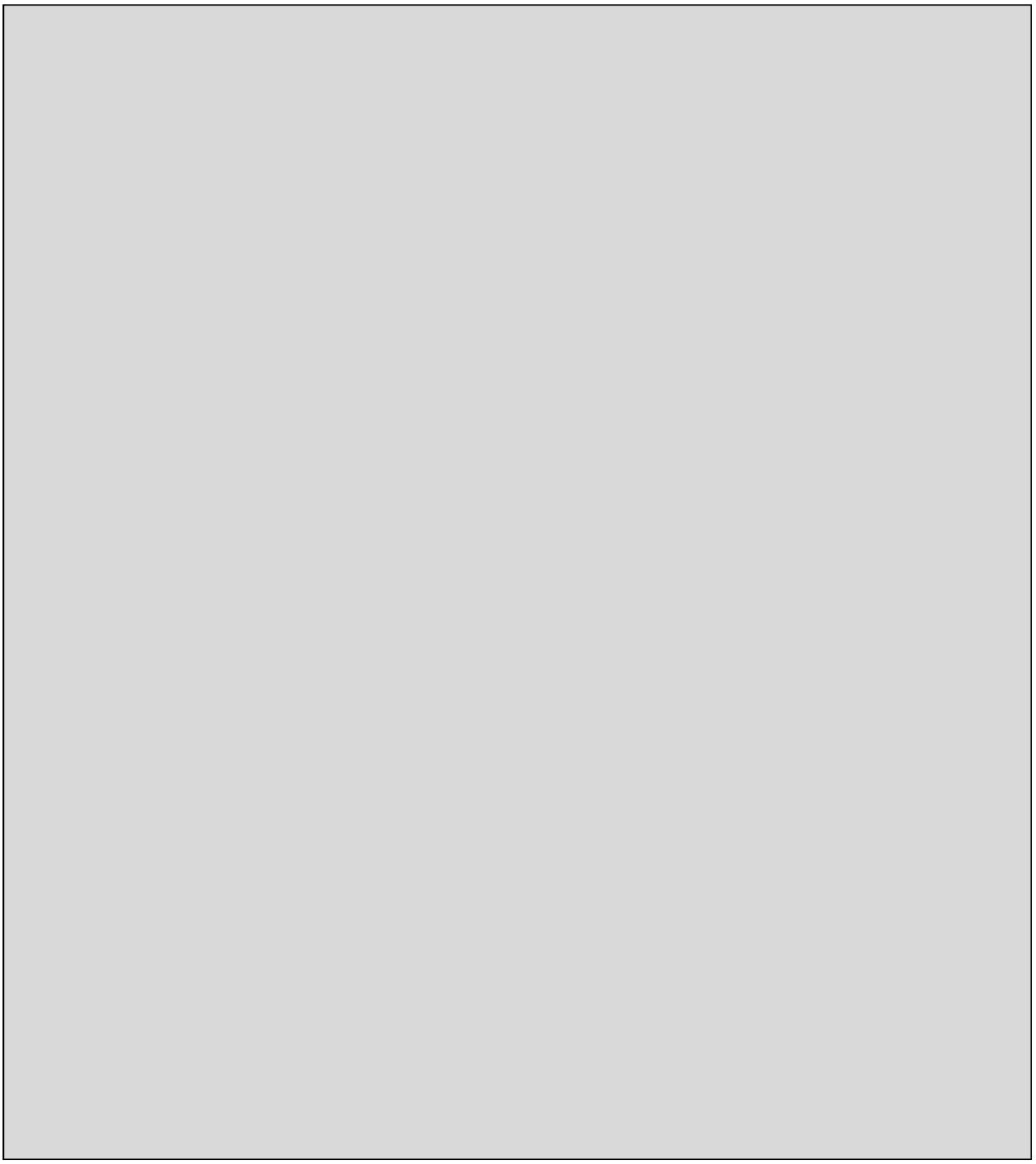
Note the use of the `onClick` property to assign the associated method with the click event of the button:

```
onClick= DisplayAge
```

The event method has the following signature:

```
public void DisplayAge(View v)
```

Assigning methods to events can be also done using an anonymous inner class as shown in the following example:



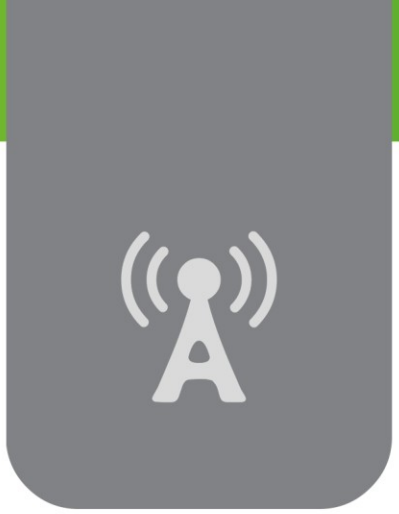
It can be done also as follows:

Exercise 1:

Create an application with an interface where the user enters a month number from 1 to 12, to show the name of the corresponding month.

Exercise 2:

Create an application with an interface where the user enters three numeric values: birthday day, birthday month, birthday year. to show the user's age in days.



Chapter 5:
Using main widgets, menus and events

Key Words:

Linear Layout, EditText, TextView, Buttons, Menus, Events.

Summary:

This unit explains the main widgets and the use of menus and events.

Outcomes:

Student will learn in this unit:

- Using main widgets.
- Using menus.
- Using events.

Plan:

2 Learning Objects

1. Application 1: Simple Calculator
2. Application 2: Simple Game

1. Application 1: Simple Calculator

Learning outcomes:

Using widgets, menus and events.

We will build an application that allows the user to enter two numbers and then perform calculations on them and show the result:

The application has a menu of two options:

- Reset to clear inputs
- Quit to close the application

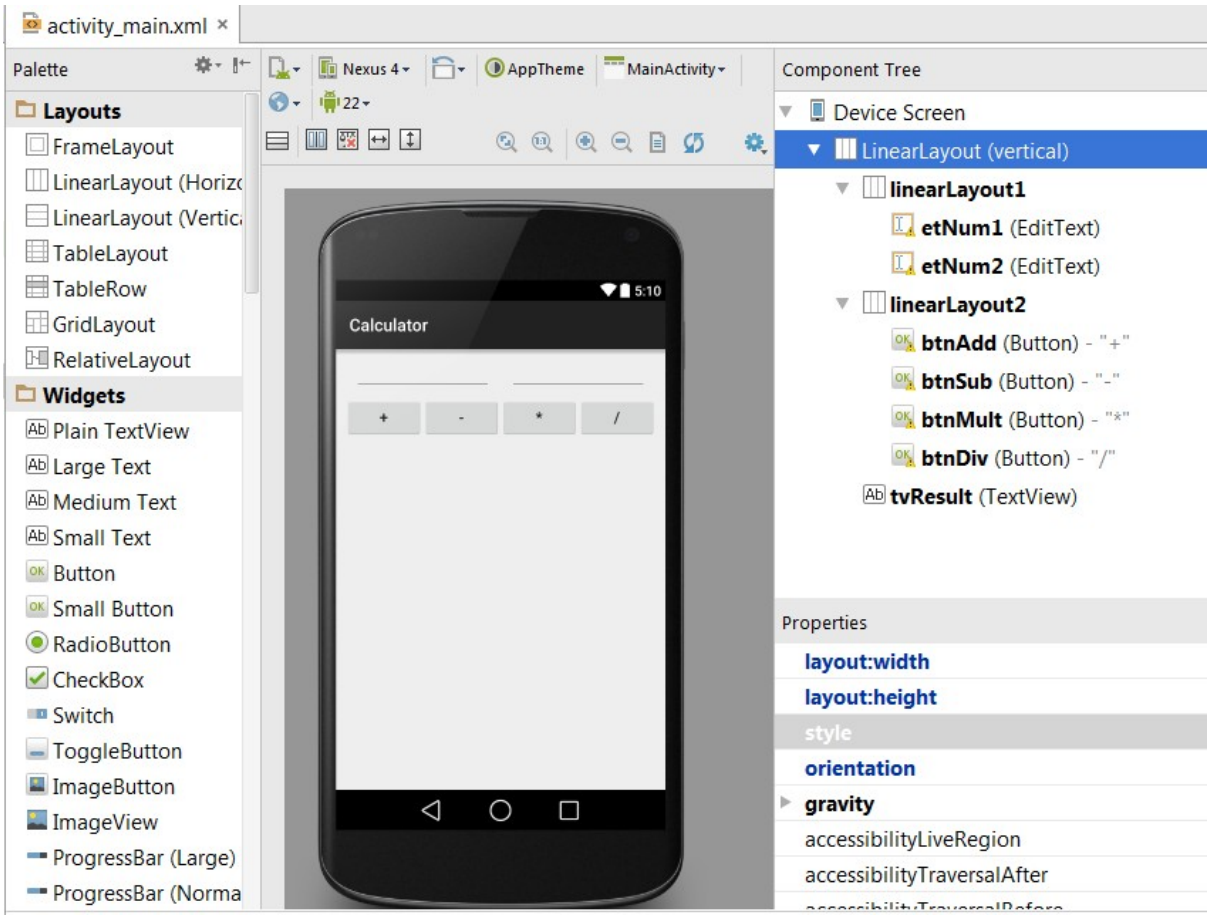
(Press F2 in the virtual device to display the menu)

Application Building Steps:

Activity Design

1. Add vertical linear layout (`LinearLayout (vertical)`) and then add tow horizontal layouts within the vertical layout.
2. Add to the first horizontal layout two `EditTexts` of type `Number(Decimal)` and set the identifiers as `etNum1, etNum2.`
3. Add four buttons to the second horizontal layout and set their identifiers: `btnAdd, btnSub, btnMult, btnDiv.`
4. Add a `TextView` to the vertical layout to display the result.

- The design will be as follow:



5. Use the `onClick` property of the buttons to link these buttons with the `Calc` method that will do the calculations.

- The activity design file activity_main.xml will be:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/linearLayout1"
            android:layout_marginLeft="10pt"
            android:layout_marginRight="10pt"
            android:layout_marginTop="3pt">
            <EditText
                android:layout_weight="1"
                android:layout_height="wrap_content"
                android:layout_marginRight="5pt"
                android:id="@+id/etNum1"
                android:layout_width="match_parent"
                android:inputType="numberDecimal">
            </EditText>
            <EditText
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:layout_marginLeft="5pt"
                android:id="@+id/etNum2"
                android:layout_width="match_parent"
                android:inputType="numberDecimal">
            </EditText>
        </LinearLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/linearLayout2"
            android:layout_marginTop="3pt"
            android:layout_marginLeft="5pt"
            android:layout_marginRight="5pt">
            <Button
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:text="+"
                android:textSize="8pt"
                android:id="@+id/btnAdd"
                android:onClick="Calc">
            </Button>
            <Button
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
```

```
                android:layout_weight="1"
                android:text="-"
                android:textSize="8pt"
                android:id="@+id/btnSub"
                android:onClick="Calc">
            </Button>
            <Button
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:text="*"
                android:textSize="8pt"
                android:id="@+id/btnMult"
                android:onClick="Calc">
            </Button>
            <Button
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:text="/"
                android:textSize="8pt"
                android:id="@+id/btnDiv"
                android:onClick="Calc">
            </Button>
        </LinearLayout>
        <TextView
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:layout_marginLeft="5pt"
            android:layout_marginRight="5pt"
            android:textSize="12pt"
            android:layout_marginTop="3pt"
            android:id="@+id/tvResult"
            android:gravity="center_horizontal">
        </TextView>
</LinearLayout>
```

Writing the Activity Code:

1. Open the activity code file MainActivity.java and declare the following variable in the class MainActivity. These variables will be used in many methods:

```
EditText etNum1;
EditText etNum2;
Button btnAdd;
Button btnSub;
Button btnMult;
Button btnDiv;
TextView tvResult;
```

2. Assign application widgets to the previous variable in the `onCreate` Assign application widgets to the previous variable in the `onCreate` method of the `MainActivity` class:

```
etNum1 = (EditText) findViewById(R.id.etNum1);
etNum2 = (EditText) findViewById(R.id.etNum2);
btnAdd = (Button) findViewById(R.id.btnAdd);
btnSub = (Button) findViewById(R.id.btnSub);
btnMult = (Button) findViewById(R.id.btnMult);
btnDiv = (Button) findViewById(R.id.btnDiv);
tvResult = (TextView) findViewById(R.id.tvResult);
```

3. Write the Calc method:
   - Note that if one of the edit texts is left empty, the method is ended without calculations.
   - You can check weather an edit text is empty:

```
TextUtils.isEmpty(etNum1.getText().toString())
```

   - As the same method is used with the four button, the following statement specify the clicked button:

```
v.getId()
```

- Where v is of type view and is the input of the method `Calc`, which indicates the object on which the event occurred.
- The `Calc` method is

```
public void Calc(View v) {

    float num1 = 0;
    float num2 = 0;
    float result = 0;

    // check if the fields are empty
            if (TextUtils.isEmpty(etNum1.getText().toString())
             || TextUtils.isEmpty(etNum2.getText().toString()))
{
        return;
    }
    // read EditText and fill variables with numbers
    num1 = Float.parseFloat(etNum1.getText().toString());
    num2 = Float.parseFloat(etNum2.getText().toString());
// defines the button that has been clicked and performs the
corresponding operation
// write operation into oper, we will use it later for output
    switch (v.getId()) {
        case R.id.btnAdd:
            oper = "+";
            result = num1 + num2;
            break;
        case R.id.btnSub:
            oper = "-";
            result = num1 - num2;
            break;
        case R.id.btnMult:
            oper = "*";
            result = num1 * num2;
            break;
        case R.id.btnDiv:
            oper = "/";
            result = num1 / num2;
            break;
        default:
            break;
    }
    // form the output line
    tvResult.setText(num1 + " " + oper + " " + num2 + " = " +
result);
}
```

4. Rewrite the `onCreateOptionsMenu` method:

```
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, MENU_RESET_ID, 0, "Reset");
    menu.add(0, MENU_QUIT_ID, 0, "Quit");
    return super.onCreateOptionsMenu(menu);
}
```

- The add method is used to add menu items:

```
add(int groupId, int itemId, int order, CharSequence title)
```

- The following constants are used with menu items:

```
final int MENU_RESET_ID = 1;
final int MENU_QUIT_ID = 2;
```

5. Rewrite the `onOptionsItemSelected` method:

```
public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()) {
        case MENU_RESET_ID:
            // clear the fields
            etNum1.setText("");
            etNum2.setText("");
            tvResult.setText("");
            break;
        case MENU_QUIT_ID:
            // exit the application
            finish();
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

- The `finish()` method ends the application.
- The `MainActivity.java` code file is:

```
package com.example.basel.calculator;

import android.app.Activity;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {

    EditText etNum1;
    EditText etNum2;
    Button btnAdd;
    Button btnSub;
    Button btnMult;
    Button btnDiv;
    TextView tvResult;
    String oper = "";
    final int MENU_RESET_ID = 1;
    final int MENU_QUIT_ID = 2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // find the elements
        etNum1 = (EditText) findViewById(R.id.etNum1);
        etNum2 = (EditText) findViewById(R.id.etNum2);
        btnAdd = (Button) findViewById(R.id.btnAdd);
        btnSub = (Button) findViewById(R.id.btnSub);
        btnMult = (Button) findViewById(R.id.btnMult);
        btnDiv = (Button) findViewById(R.id.btnDiv);
        tvResult = (TextView) findViewById(R.id.tvResult);
    }

public void Calc(View v) {

        float num1 = 0;
        float num2 = 0;
        float result = 0;

// check if the fields are empty
if (TextUtils.isEmpty(etNum1.getText().toString())
   || TextUtils.isEmpty(etNum2.getText().toString())) {
       return;
}
```

```
// read EditText and fill variables with numbers
num1 = Float.parseFloat(etNum1.getText().toString());
num2 = Float.parseFloat(etNum2.getText().toString());
// defines the button that has been clicked and performs the
corresponding operation
// write operation into oper, we will use it later for output
        switch (v.getId()) {
            case R.id.btnAdd:
                oper = "+";
                result = num1 + num2;
                break;
            case R.id.btnSub:
                oper = "-";
                result = num1 - num2;
                break;
            case R.id.btnMult:
                oper = "*";
                result = num1 * num2;
                break;
            case R.id.btnDiv:
                oper = "/";
                result = num1 / num2;
                break;
            default:
                break;
        }
// form the output line
tvResult.setText(num1 + " " + oper + " " + num2 + " = " +
result);
    }
       @Override
    // menu creation
    public boolean onCreateOptionsMenu(Menu menu) {

        menu.add(0, MENU_RESET_ID, 0, "Reset");
        menu.add(0, MENU_QUIT_ID, 0, "Quit");
        return super.onCreateOptionsMenu(menu);
    }
    // process menu item clicks
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId()) {
            case MENU_RESET_ID:
                // clear the fields
                etNum1.setText("");
                etNum2.setText("");
                tvResult.setText("");
                break;
            case MENU_QUIT_ID:
                // exit the application
```

```
                finish();
                break;
            }
        return super.onOptionsItemSelected(item);
    }
}
```

2. Application 2: Simple Game

Learning outcomes:

Using widgets, menus and events.

- We will build a simple game in which the user is asked to click on the larger number of two random numbers that appear on the interface.
- If the user clicks the larger number, a correct popup message appears, and a counter at the bottom of the screen counts the correct answers.
- If the user clicks the smaller number, an error message appears, and the counter is decreased.

In both cases, the game continues to show new numbers. And so on.

Building the application steps:

1. Start by building the following activity that **contains three text views and two buttons**.
2. Adjust the properties of the third text view to be displayed in the center of interface bottom.

```
text="Points: 0"
id="@+id/pointsTextView"
layout_alignParentBottom="true"
layout_centerHorizontal="true"
```

3. Adjust the properties of the two buttons to be displayed at left and the right of the interface.

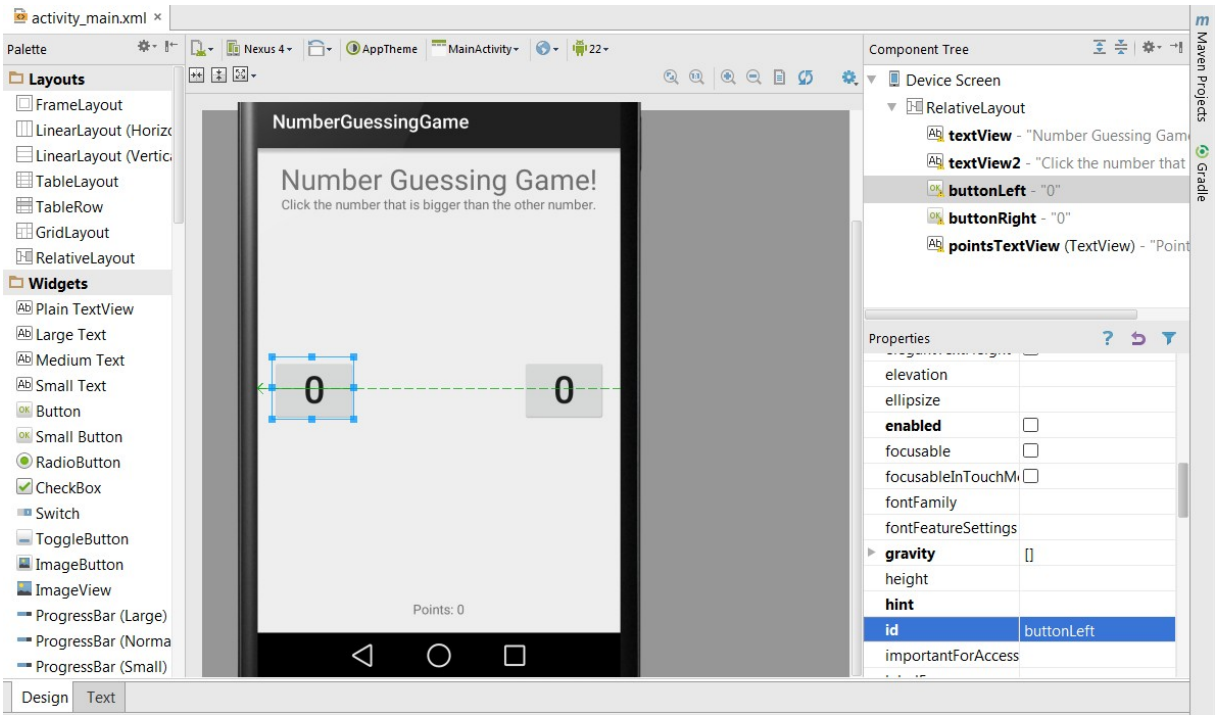
The left button:

```
text="0"
id="@+id/buttonLeft"
layout_centerVertical="true"
layout_alignParentLeft="true"
layout_alignParentStart="true"
textSize="40dp"
onClick="clickButton2"
```

The right button:

```
text="0"
id="@+id/buttonRight"
layout_centerVertical="true"
layout_alignParentRight="true"
layout_alignParentEnd="true"
textSize="40dp"
onClick="clickButton2"
```

The design is:



The activity_main.xml file:

```
<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:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Number Guessing Game!"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:textSize="30dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Click the number that is bigger than the
other number."
        android:id="@+id/textView2"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0"
        android:id="@+id/buttonLeft"
        android:layout_centerVertical="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:textSize="40dp"
        android:onClick="clickButton1" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0"
        android:id="@+id/buttonRight"
        android:layout_centerVertical="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:textSize="40dp"
        android:onClick="clickButton2" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Points: 0"
        android:id="@+id/pointsTextView"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_alignParentEnd="false" />

</RelativeLayout>
```

4. Declare the following variables in the `MainActivity` class:

```
private int num1; //the numbers on the left and right
buttons
private int num2;
private int points;   // player's point total; initially 0
```

5. Write the roll() method that generates two random numbers between 0 and 9 and put each generated number on one button

```
private void roll() {
    // pick two random numbers
    Random r = new Random();
    num1 = r.nextInt(9);
    num2 = r.nextInt(9);
    while (num2 == num1) {
        num2 = r.nextInt(9);
    }

// set the buttons to display the random numbers
Button left = (Button) findViewById(R.id.buttonLeft);
left.setText("" + num1); // "" + int -> converts int to String

Button right = (Button) findViewById(R.id.buttonRight);
right.setText("" + num2);
}
```

6. Write the *check* method that accepts two numbers: if the first number is greater than the second number, the *Correct!* message is shown and the *points* counter is increased by 1.

   If the first number is less than the second one, an error message is shown, and the counter is decreased by 1.

```
private void check(int a, int b) {
    if (a > b) {
        points++;
        Toast.makeText(this, "Correct!",
                        Toast.LENGTH_SHORT).show();
    } else {
        points--;
        Toast.makeText(this, "Wrong!",
                        Toast.LENGTH_SHORT).show();
    }

    TextView pointsView = (TextView)
                findViewById(R.id.pointsTextView);
    pointsView.setText("Points: " + points);
    roll();
}
```

7. Add a call to the `roll()` method in the `onCreate` method:

```
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 roll(); // <-- we added this line to set initial button
random numbers
}
```

8. Write the first button method that calls the check method with passing the first number as the first parameter and the second number as the second number.

```
public void clickButton1(View view) {
    check(num1, num2);
}
```

9. Write the second button method that calls the check method with passing the second number as the first parameter and the first number as the second number.

```
public void clickButton2(View view) {
    check(num2, num1);
}
```

10. Assign each of the above methods with the `onClick` event of each button.

The `MainActivity.java` code file is:

```
package com.example.basel.numberguessinggame;
import android.app.*;
import android.support.v7.app.*;
import android.os.*;
import android.view.*;
import android.widget.*;
import java.util.*;

public class MainActivity extends Activity {
    private int num1;     // the numbers on the left and right
buttons
    private int num2;
    private int points;  // player's point total; initially 0

    /*
     * Called when the player clicks the left number button.
     */
    public void clickButton1(View view) {
        check(num1, num2);
    }

    /*
     * Called when the player clicks the right number button.
     */
    public void clickButton2(View view) {
        check(num2, num1);
    }

    /*
     * Updates the player's score based on whether they
guessed correctly.
     * Also shows a 'toast' which is a brief popup message.
     */
private void check(int a, int b) {
        if (a > b) {
            points++;
            Toast.makeText(this, "Correct!",
                                Toast.LENGTH_SHORT).show();
        } else {
            points--;
            Toast.makeText(this, "Wrong!",
                                Toast.LENGTH_SHORT).show();
        }

        TextView pointsView = (TextView)
         findViewById(R.id.pointsTextView);
        pointsView.setText("Points: " + points);
        roll();
    }
```

```
    /*
     * Chooses new random integers to appear on the two
buttons.
     */
private void roll() {
        // pick two random numbers
        Random r = new Random();
        num1 = r.nextInt(9);
        num2 = r.nextInt(9);
        while (num2 == num1) {
            num2 = r.nextInt(9);
        }
        // set the buttons to display the random numbers
        Button left = (Button) findViewById(R.id.buttonLeft);
        left.setText("" + num1);   // "" + int : converts int
to String

        Button right =(Button)findViewById(R.id.buttonRight);
        right.setText("" + num2);
    }

@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        roll();   // <-- we added this line to set initial
button random numbers
    }
}
```

Exercise 1: Random number guessing game

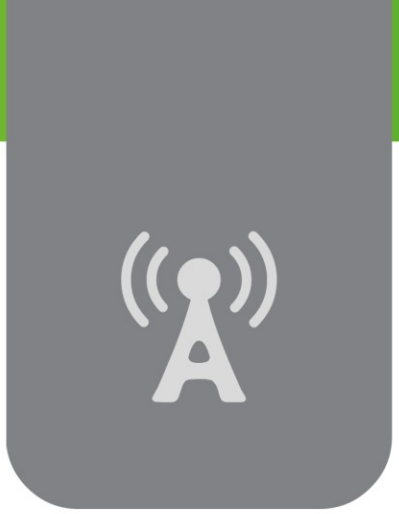
The required application generates a random number between 1 and 1000, and the user must find this number in at most five attempts: in each attempt the user type a number.

If the user finds the number correctly, a message is displayed and the game ends, else a message is shown indicating whether the entered number is grater or smaller than the required number.

Exercise 2: Grade computer

The user types a number (mark) between 0 and 100. The application gives the literal equivalent of the entered number according to the following table:

96-100	A+
91-95	A
86-90	A-
81-85	B+
76-80	B
71-75	B-
60-70	D
<60	F



Chapter 6:
Widgets

Key Words:

Button, ImageView, EditText, CheckBox, RadioButton, Spinner, Action Bar.

Summary:

This unit explains main widgets and the use of the action bar.

Outcomes:

Student will learn in this unit:

- Main widgets.
- Action bar.

Plan:

2 Learning Objects

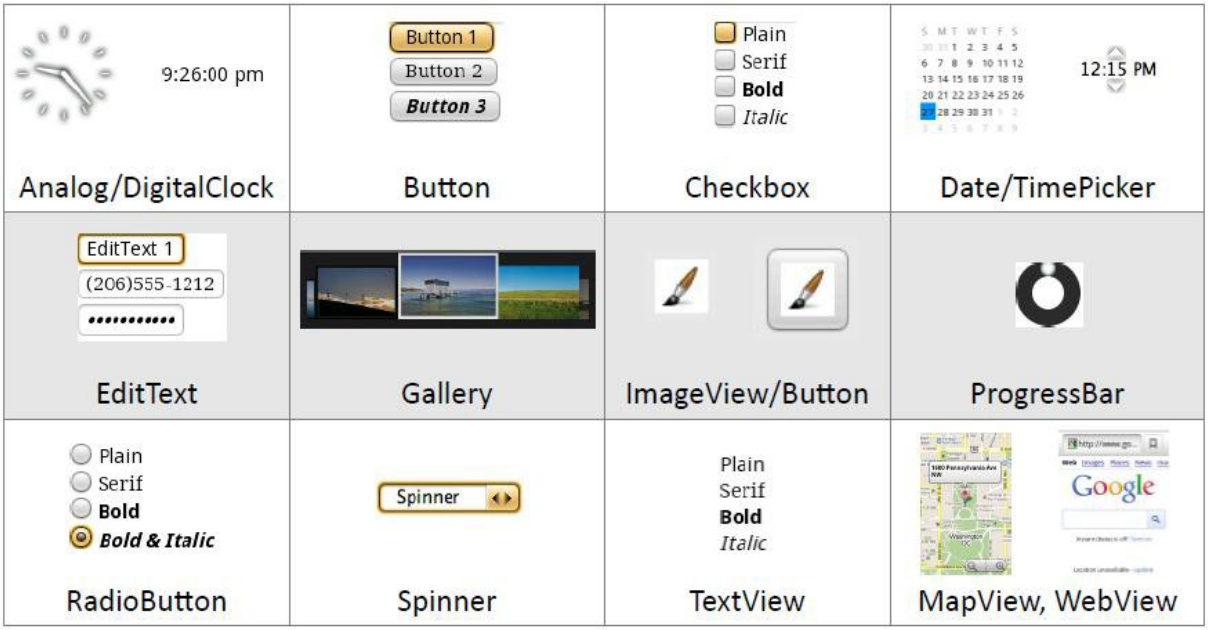
1. GUI Widgets
2. The Action Bar

1. GUI Widgets

Learning outcomes:

Widgets.

You can add many different widgets (controls) to your interfaces.



Button:

- A clickable widget with a text label.
- key attributes:

`android:clickable="`***`bool`***`"`	set to false to disable the button
`android:id="@+id/`***`theID`***`"`	unique ID for use in Java code
`android:onClick="`***`function`***`"`	function to call in activity when clicked public void fct (View arg)
`android:text="`***`text`***`"`	text to put in the button
represented by Button class in Java code: *`Button b = (Button) findViewById (R.id.`**`theID`**`);`*	

Image Button:

- A clickable widget with an image label.
- key attributes:

`android:clickable="bool"`	set to false to disable the button
`android:id="@+id/theID"`	unique ID for use in Java code
`android:onClick="function"`	function to call in activity when clicked public void fct (View arg)
`android:src="@drawable/img"`	image to put in the button (must correspond to an image resource)

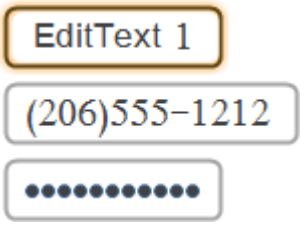
- to set up an image resource:
  - put image file in project folder app / src / main / res / drawable
  - use @drawable / foo to refer to foo.png
- use simple file names with only letters and numbers

ImageView:

- Displays an image without being clickable.
- key attributes

`android:id="@+id/theID"`	unique ID for use in Java code
`android:src="@drawable/img"`	image to put in the button (must correspond to an image resource)
to change the visible image, in Java code: • get the `ImageView` using `findViewById` • call its **`setImageResource`** method and pass `R.drawable.filename`	

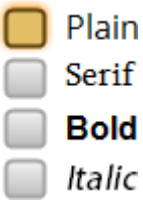
EditText:



- An editable text input box.
- key attributes:

`android:hint="`***`text`***`"`	gray text to show before user starts to type.
`android:id="@+id/`***`theID`***`"`	unique ID for use in Java code
`android:inputType="`***`type`***`"`	number, phone, date, time,…
`android:lines="`***`int`***`"`	number of visible lines (rows) of input
`android:maxLines="`***`int`***`"`	max lines to allow user to type in the box
`android:text="`***`text`***`"`	initial text to put in box (default empty)
`android:textSize="`***`size`***`"`	size of font to use (e.g. "20dp")
others: `capitalize, digits, fontFamily, letterSpacing, lineSpacingExtra, minLines, numeric, password, phoneNumber, singleLine, textAllCaps, textColor, typeface.`	

CheckBox:



- An individual toggleable on/off switch
- Key attributes:

`android:checked="`***`bool`***`"`	set to true to make it initially checked
`android:clickable="`***`bool`***`"`	set to false to disable the checkbox
`android:id="@+id/`***`theID`***`"`	unique ID for use in Java code
`android:onClick="`***`function`***`"`	function to call in activity when clicked public void fct (View arg)
`android:text="`***`text`***`"`	text to put next to the checkbox

In Java code:
`CheckBox cb = (CheckBox) findViewById(R.id.`***`theID`***`);`
`cb.toggle();`
`cb.setChecked(true);`
`cb.performClick();`	

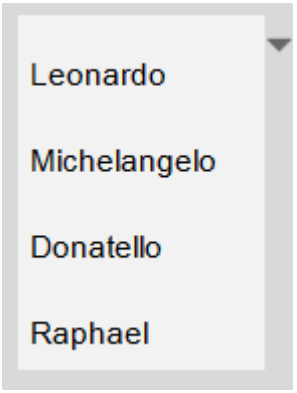
RadioButton:

- A toggleable on/off switch; part of a group
- Key attributes:

android:checked="***bool***"	set to true to make it initially checked
android:clickable="***bool***"	set to false to disable the button
android:id="@+id/***theID***"	unique ID for use in Java code
android:onClick="***function***"	function to call in activity when clicked public void fct (View arg)
android:text="***text***"	text to put next to the button

need to be nested inside a `RadioGroup` tag in XML
so that only one can be selected at a time

```
<LinearLayout ...
android:orientation="vertical"
android:gravity="center|top">
<RadioGroup ...
android:orientation="horizontal">
<RadioButton ... android:id="@+id/lions"
android:text="Lions"
android:onClick="radioClick" />
<RadioButton ... android:id="@+id/tigers"
android:text="Tigers"
android:checked="true"
android:onClick="radioClick" />
<RadioButton ... android:id="@+id/bears"
android:text="Bears, oh my!"
android:onClick="radioClick" />
</RadioGroup>
</LinearLayout>

// in MainActivity.java
public class MainActivity extends Activity {
public void radioClick(View view) {
// check which radio button was clicked
if (view.getId() == R.id.lions) {
// ...
} else if (view.getId() == R.id.tigers) {
// ...
} else {
// bears ...
}
}
```



Spinner:

- A drop–down menu of selectable choices
- Key attributes:

android:clickable="***bool***"	set to false to disable the spinner
android:id="@+id/***theID***"	unique ID for use in Java code
android:entries="@array/***array***"	set of options to appear in spinner (must match an array in strings.xml)
android:prompt="@string/***text***"	title text when dialog of choices pops up

also need to handle events in Java code:

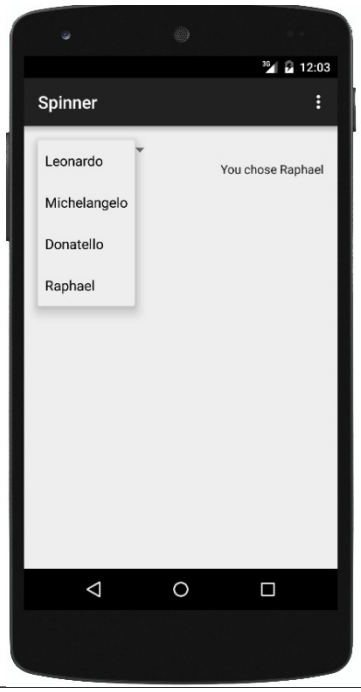
- must get the Spinner object using `findViewById`
- then call its `setOnItemSelectedListener` method

Example:

```
<Spinner
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tmnt"
    android:entries="@array/turtles"
    android:prompt="@string/choose_turtle" />

<TextView android:text=" "
android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/turtle_result" />
in res/values/strings.xml:
<string name="choose_turtle">Choose a turtle:</string>
<string-array name="turtles">
    <item>Leonardo</item>
    <item>Michelangelo</item>
    <item>Donatello</item>
    <item>Raphael</item>
</string-array>
                                                    // in MainActivity.java

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Spinner spin = (Spinner) findViewById(R.id.tmnt);

    spin.setOnItemSelectedListener(new
              AdapterView.OnItemSelectedListener() {
```

```
        public void onItemSelected
             (AdapterView<?> spin, View v, int i, long id) {
             TextView result = (TextView)
                         findViewById(R.id.turtle_result);
             result.setText("You chose " +
spin.getSelectedItem());
         }

         public void onNothingSelected(AdapterView<?> parent)
{
         } // empty
     });
}
```

The execution will show the selected item by the user in the text view:



String resources:

Declare constant strings and arrays in res/values/strings.xml:

```
<resources>

<string name="name">value</string>
<string name="name">value</string>

<string-array name="arrayname">
<item>value</item>
<item>value</item>
<item>value</item>  <!-- must escape ' as \' in values -->
...
<item>value</item>
</string-array>
</resources>
```

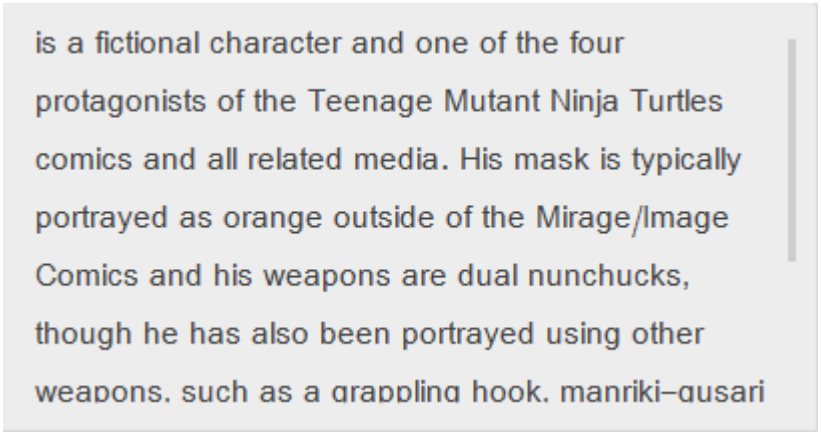
- Refer to them in Java code:
- as a resource: `R.string.name, R.array.name`

  as a string or array: `getResources ().getString (R.string.name), getResources ().getStringArray (R.array.name)`
- as a string or array: `getResources ().getString (R.string.name), getResources ().getStringArray (R.array.name)`

ScrollView:

A container with scrollbars around another widget or container

Example:




```
<LinearLayout ...>
...
<ScrollView
android:layout_width="wrap_content"
android:layout_height="wrap_content">
<TextView ... android:id="@+id/turtle_info" />
</ScrollView>
</LinearLayout>
```

List:

A visible menu of selectable choice

- lists are more complicated, so we'll cover them later

2. The Action Bar

Learning outcomes:

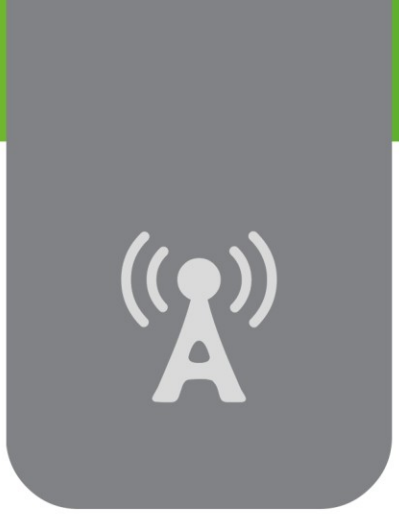
The action bar.

Action bar: top-level menu of app functions

- identifies current activity/app to user
- make common actions prominent and available
- make less common actions available through a drop-down menu

To create an action bar:

1. make activity class extend ActionBarActivity
2. write methods:
   - onCreateOptionsMenu: reads the menu_main.xml file to create menu items.
   - onOptionsItemSelected: react with the selection of a menu item.
3. declare the menu items in res/menu/menu_activity.xml
   decide which items have icons, which have text, which should appear in main menu, which in "overflow" submenu
   - need to place icon image files in res / drawable folder
4. handle events:
   - write code in onOptionsItemSelected to check what option was clicked and respond accordingly



Chapter 7:
Layouts

Key Words:

Linear Layout, Widget Box Model, Grid Layout, Nested Layout, Relative Layout, Frame Layout.

Summary:

This unit shows different possible layouts to position widgets within the interfaces.

Outcomes:

Student will learn in this unit:

- Layouts Types.
- Formatting widgets.

Plan:

1 Learning Object

1. Layouts

1. Layouts

Learning outcomes:

Layouts.

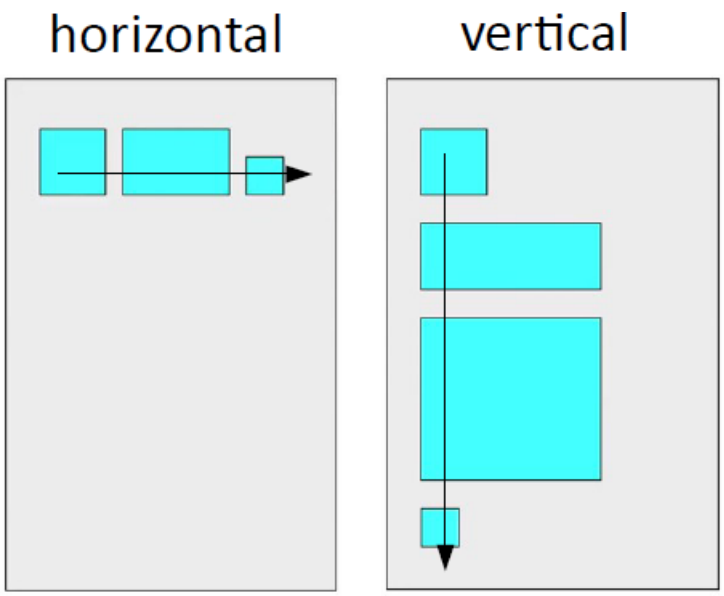
The layout decides where to position each component based on some general rules or criteria

Changing layouts:

- go to the **Text** view for your layout XML file
- modify the opening/closing tags to the new layout type, e.g. LinearLayout.
- now go back to **Design** view and add widgets

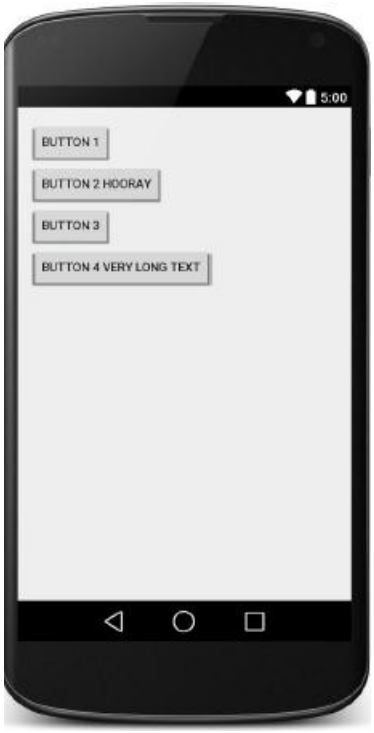
LinearLayout:

- lays out widgets/views in a single line
- **orientation** of horizontal (default) or vertical
- items do not wrap if they reach edge of screen!



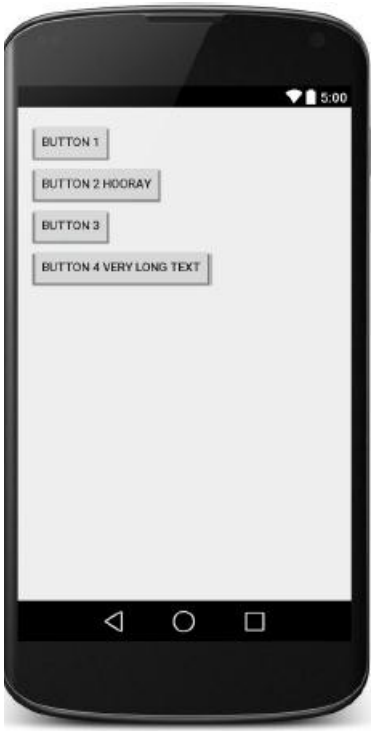
The following is an example of horizontal layout:

```
<LinearLayout ...
    android:orientation="horizontal"
    tools:context=".MainActivity">
<Button ... android:text="Button 1" />
<Button ... android:text="Button 2 Hooray" />
<Button ... android:text="Button 3" />
<Button ... android:text="Button 4 Very Long Text" />
</LinearLayout>
```



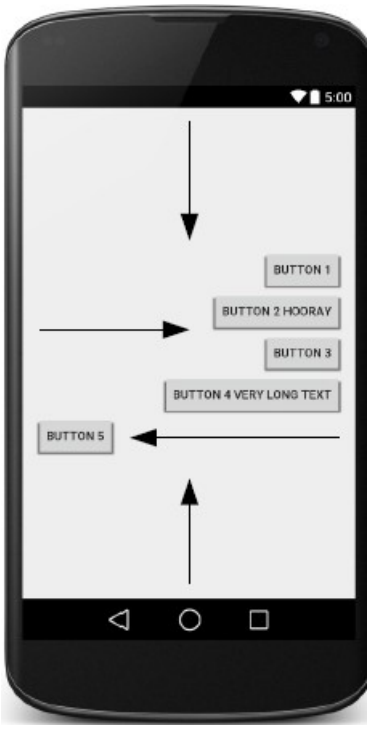
The following is an example of a vertical layout:

```
<LinearLayout ...
    android:orientation="vertical"
    tools:context=".MainActivity">
<Button ... android:text="Button 1" />
<Button ... android:text="Button 2  Hooray" />
<Button ... android:text="Button 3" />
<Button ... android:text="Button 4 Very Long Text" />
</LinearLayout>
```


5:00
BUTTON 1
BUTTON 2 HOORAY
BUTTON 3
BUTTON 4 VERY LONG TEXT

Gravity:

- **gravity**: alignment direction that widgets are pulled:
  - top, bottom, left, right, center
  - combine multiple with |
  - set gravity on the layout to adjust all widgets; set layout_gravity on an individual widget




```
<LinearLayout ...
    android:orientation="vertical"
    android:gravity="center|right">
<Button ... android:text="Button 1" />
<Button ... android:text="Button 2 Hooray" />
<Button ... android:text="Button 3" />
<Button ... android:text="Button 4 Very Long Text" />
<Button ... android:text="Button 5"
    android:layout_gravity="left" />
</LinearLayout>
```

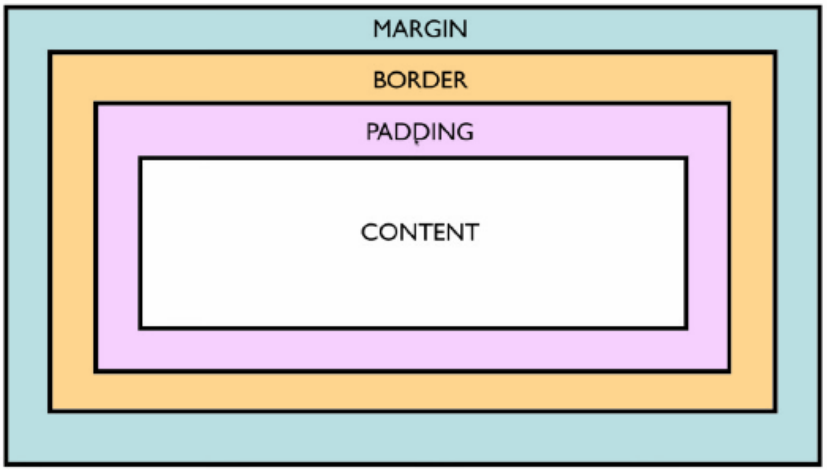
Weight:

- **weight:** gives elements relative sizes by integers
  - widget with weight **K** gets **K** / total fraction of total size
  - cooking analogy: "2 parts flour, 1 part water, …"


```
<LinearLayout ...
    android:orientation="vertical">
<Button ... android:text="B1"
    android:layout_weight="1" />
<Button ... android:text="B2"
    android:layout_weight="3" />
<Button ... android:text="B3"
    android:layout_weight="1" />
</LinearLayout>
```

Widget Box Model:

- **content:** every widget or view has a certain size (width x height) for its content, the widget itself
- **padding:** you can artificially increase the widget's size by applying padding in the widget just outside its content
- **border:** outside the padding, a line around edge of widget
- **margin:** separation from neighboring widgets on screen



Sizing:

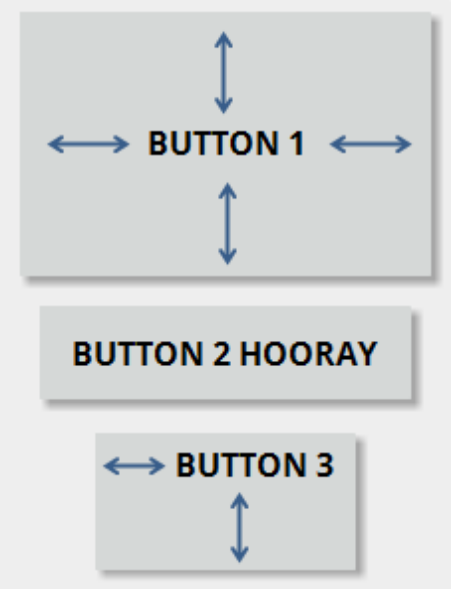
- **width** and **height** of a widget can be:
  - `wrap_content`: exactly large enough to fit the widget's content
  - `match_parent`: as wide or tall as 100% of the screen or layout
  - a specific fixed width such as 64dp (not usually recommended)
  - `dp`= device pixels; dip = device–independent pixels; `sp`= scaling pixels


```
<Button ...
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Padding:

- padding: extra space inside widget
- set padding to adjust all sides; `paddingTop, Bottom, Left, Right` for one side.
- usually set to specific values like 10dp (some widgets have a default value ~16dp)

```
<LinearLayout ...
    android:orientation="vertical">
<Button ... android:text="Button 1"
    android:padding="50dp" />
<Button ... android:text="Button 2 Hooray" />
<Button ... android:text="Button 3"
    android:paddingLeft="30dp"
    android:paddingBottom="40dp" />
</LinearLayout>
```



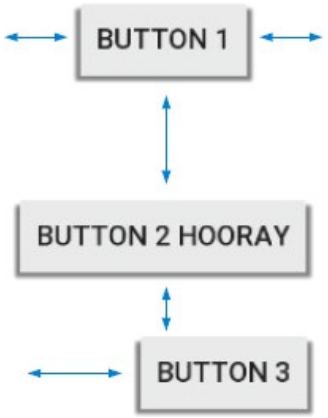
Margin:

margin: extra space outside widget to separate it from others

set `layout_margin` to adjust all sides; `layout_marginTop, Bottom, Left, Right`

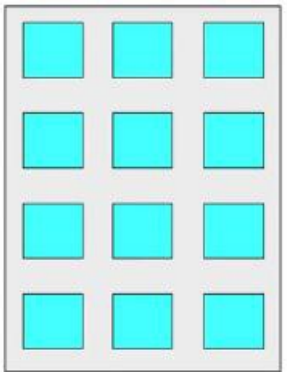
usually set to specific values like 10dp (set defaults in res/values/dimens.xml)

```
<LinearLayout ...
    android:orientation="vertical">
<Button ... android:text="Button 1"
    android:layout_margin="50dp" />
<Button ... android:text="Button 2 Hooray" />
<Button ... android:text="Button 3"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="40dp" />
</LinearLayout>
```



Grid Layout:

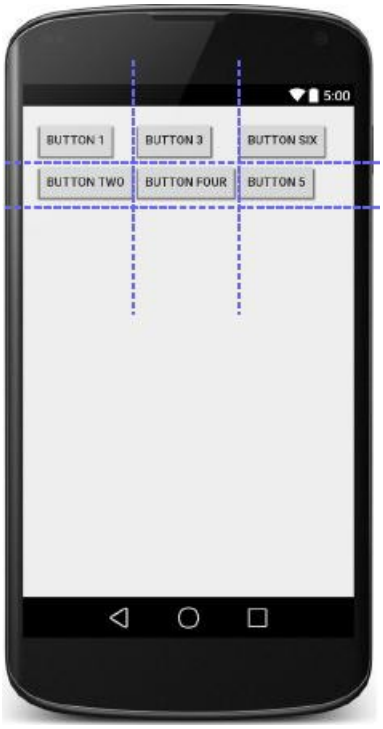
- lays out widgets / views in lines of **rows** and **columns**
  - orientation attribute defines row–major or column–major order
  - introduced in Android 4; replaces older TableLayout
- by default, rows and columns are equal in size
  - each widget is placed into "next" available row/column index unless it is given an explicit layout_row and layout_column attribute
  - grid of 4 rows, 3 columns:



```
<GridLayout ...
    android:rowCount="2"
    android:columnCount="3"
    tools:context=".MainActivity">
<Button ... android:text="Button 1" />
<Button ... android:text="Button Two" />
<Button ... android:text="Button 3" />
<Button ... android:text="Button Four" />
<Button ... android:text="Button 5" />
<Button ... android:text="Button Six" />
</GridLayout>
```

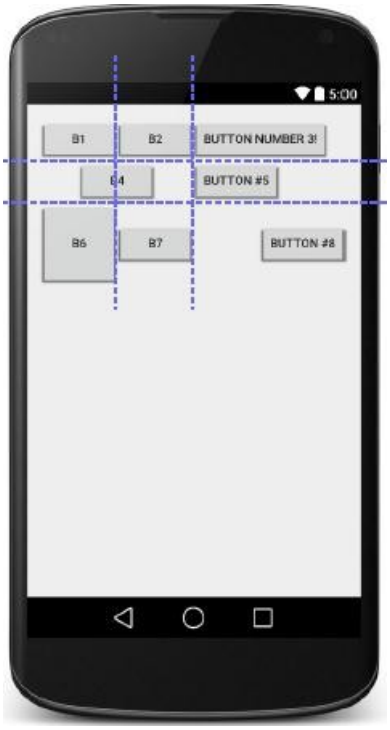
Example:

```
<GridLayout ...
    android:rowCount="2"
    android:columnCount="3"
    android:orientation="vertical">
<Button ... android:text="Button 1" />
<Button ... android:text="Button Two" />
<Button ... android:text="Button 3" />
<Button ... android:text="Button Four" />
<Button ... android:text="Button 5"
    android:layout_row="1"
    android:layout_column="2" />
<Button ... android:text="Button Six"
    android:layout_row="0"
    android:layout_column="2" />
</RelativeLayout>
```


5:00
BUTTON 1
BUTTON 3
BUTTON SIX
BUTTON TWO
BUTTON FOUR
BUTTON 5

Example:

```
<GridLayout ...
    android:rowCount="2"
    android:columnCount="3">
<Button ... android:text="B1" />
<Button ... android:text="B2" />
<Button ... android:text="Button Number 3!" />
<Button ... android:text="B4"
    android:layout_columnSpan="2"
    android:layout_gravity="center" />
<Button ... android:text="B5" />
<Button ... android:text="B6"
    android:layout_paddingTop="40dp"
    android:layout_paddingBottom="40dp" />
<Button ... android:text="B7" />
<Button ... android:text="Button #8"
    android:layout_gravity="right" />
</RelativeLayout>
```



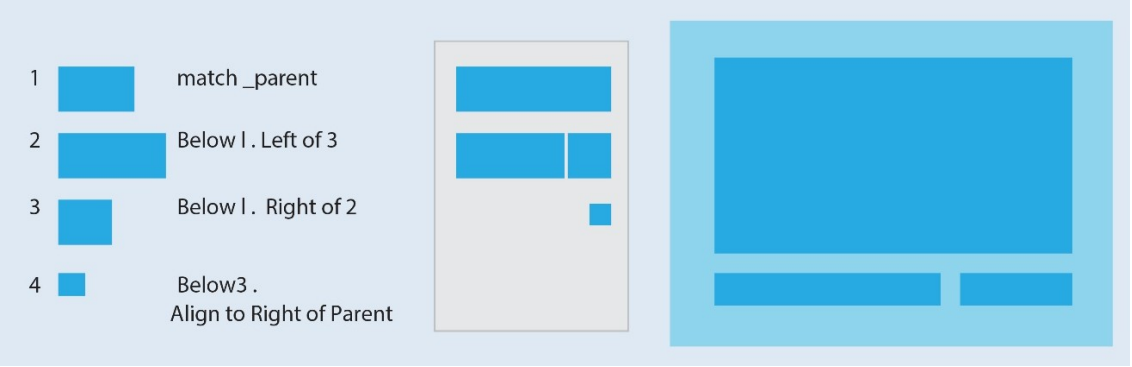
Nested layout:

- to produce more complicated appearance, use a nested layout (layouts inside layouts)


```
<OuterLayoutType ...>
<InnerLayoutType ...>
<Widget ... />
<Widget ... />
    </InnerLayoutType>
<InnerLayoutType ...>
<Widget ... />
<Widget ... />
    </InnerLayoutType>
<Widget ... />
<Widget ... />
</OuterLayoutType>
```

RelativeLayout:

- each widget's position and size are relative to other views
  - relative to "parent" (the activity itself)
  - relative to other widgets/views
  - x–positions of reference: left, right, center
  - y–positions of reference: top, bottom, center
- intended to reduce the need for nested layouts



Relative anchor points:

- properties for x/y relative to **another widget**:
  layout_below, above, toLeftOf, toRightOf

set these to the ID of another widget in the format "@id/theID"
(obviously, the given widget must have an ID for this to work)

- properties for x/y relative to layout **container** (the activity):
  layout_alignParentTop, Bottom, Left, Right

set these flags to a boolean value of "true" to enable them
layout_center Horizontal, Vertical, InParent

set these flags to "true" to center the control within its parent in a dimension

```
<RelativeLayout ... >
<Button ... android:id="@+id/b1" android:text="B1"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />
<Button ... android:id="@+id/b2" android:text="B2"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/b1" />
<Button ... android:id="@+id/b3" android:text="B3"
    android:layout_centerHorizontal="true"
    android:layout_below="@+id/b2" />
<Button ... android:id="@+id/b4" android:text="B4"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/b2" />
<TextView ... android:id="@+id/tv1"
    android:text="I'm a TextView!"
    android:layout_centerInParent="true" />
<Button ... android:id="@+id/b5" android:text="B5"
    android:padding="50dp"
    android:layout_centerHorizontal="true"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="50dp" />
</RelativeLayout>
```

FrameLayout:

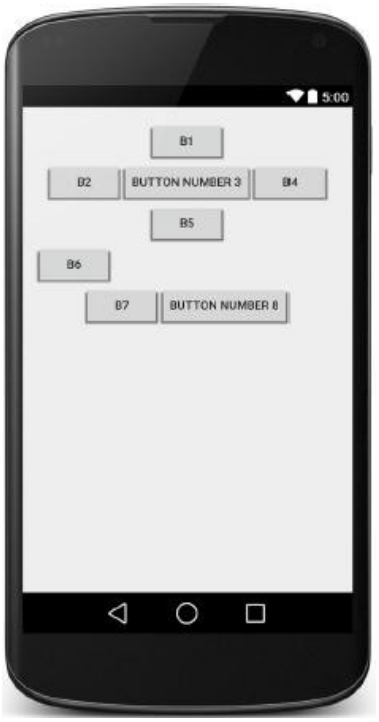
- meant to hold only a single widget inside, which occupies the entirety of the activity
  - most commonly used with layout fragments (seen later)
  - less useful for more complex layouts

(can put in multiple items and move them to "front" in Z–order)


```
<FrameLayout ... >
<ImageView
android:src="@drawable/jellybean"
... />
</FrameLayout>
```

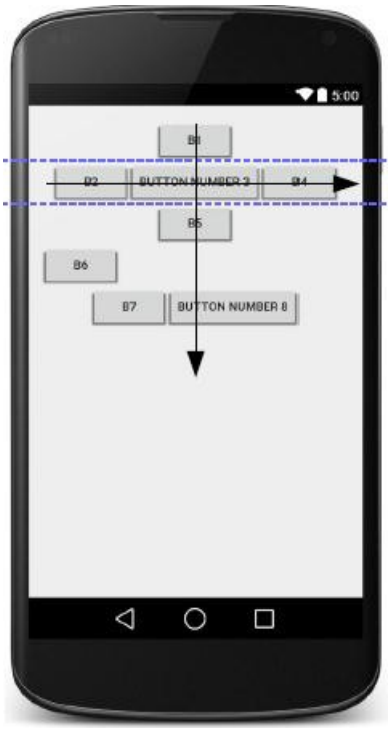
Exercise:

Write the necessary activity to display:



Solution:

```
<LinearLayout ...
    android:orientation="vertical"
android:gravity="center|top">
<Button ... android:text="B1" />
<LinearLayout ...
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="center|top">
<Button ... android:text="B2" />
<Button ... android:text="Button Number 3" />
<Button ... android:text="B4" />
</LinearLayout>
<Button ... android:text="B5" />
<Button ... android:text="B6" android:layout_gravity="left" />
<LinearLayout ...
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="center|top">
<Button ... android:text="B7" />
<Button ... android:text="Button Number 8" />
</LinearLayout>
</LinearLayout>
```



Chapter 8:
Activities and Intents

Key Words:

Apps, Memory, Storage, Log, Activity, Intent.

Summary:

This unit shows activities lifecycles and associated events, and how to communicate between activities using intents.

Outcomes:

Student will learn in this unit:

- Activity states.
- Activity events.
- Creating and using intents.

Plan:

3 Learning Objects

1. The Activity Lifecycle
2. Multiple activities and Intents
3. Example

1. The Activity Lifecycle

Learning outcomes:

Activity lifecycle.

Apps, memory, and storage:

- **Storage:** Your device has apps and files installed and stored on its internal disk, SD card, etc.

  Settings →Storage

- **memory:** Some subset of apps might be currently loaded into the device's RAM and are either running or ready to be run.
  - When the user loads an app, it is loaded from storage into memory.
  - When the user exits an app, it might be cleared from memory, or might remain in memory so you can go back to it later.
  - See which apps are in memory:

    Settings → Apps → Running

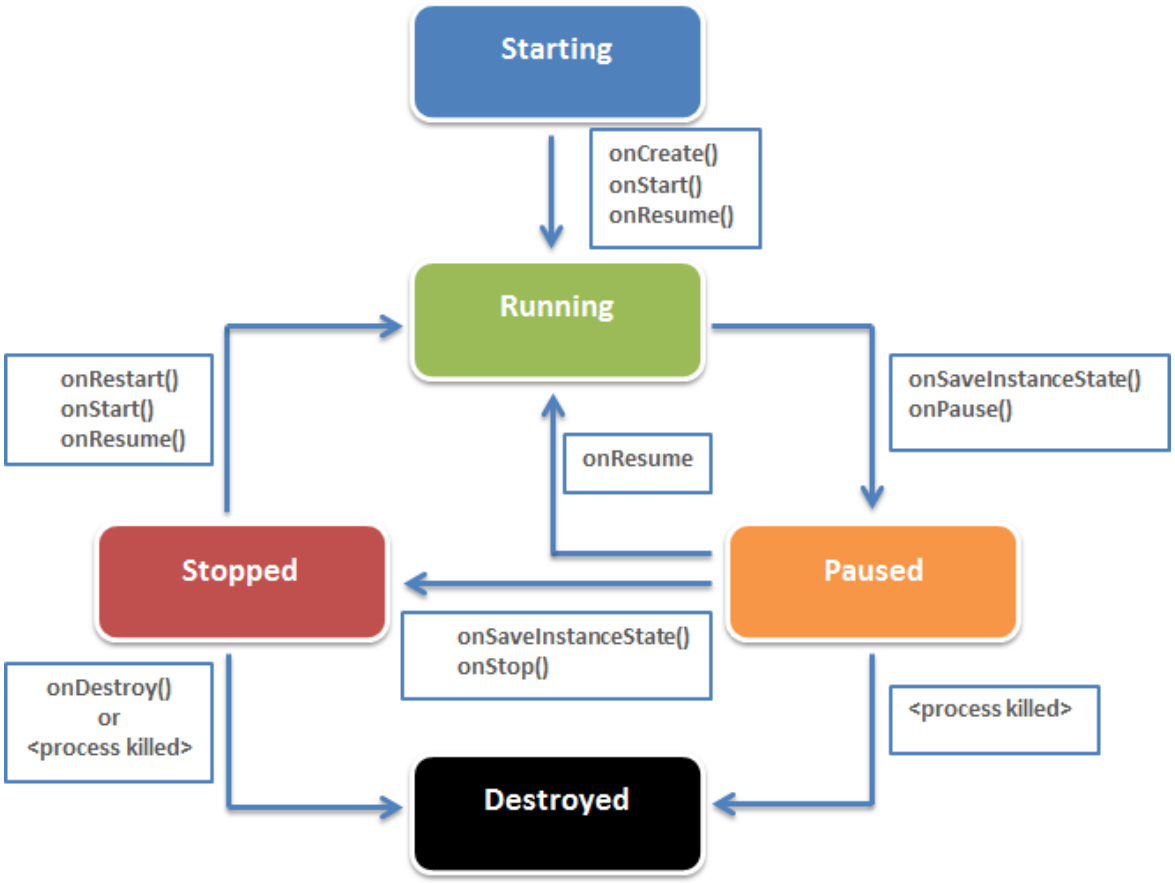
Activity State:

- An activity can be thought of as being in one of several states:

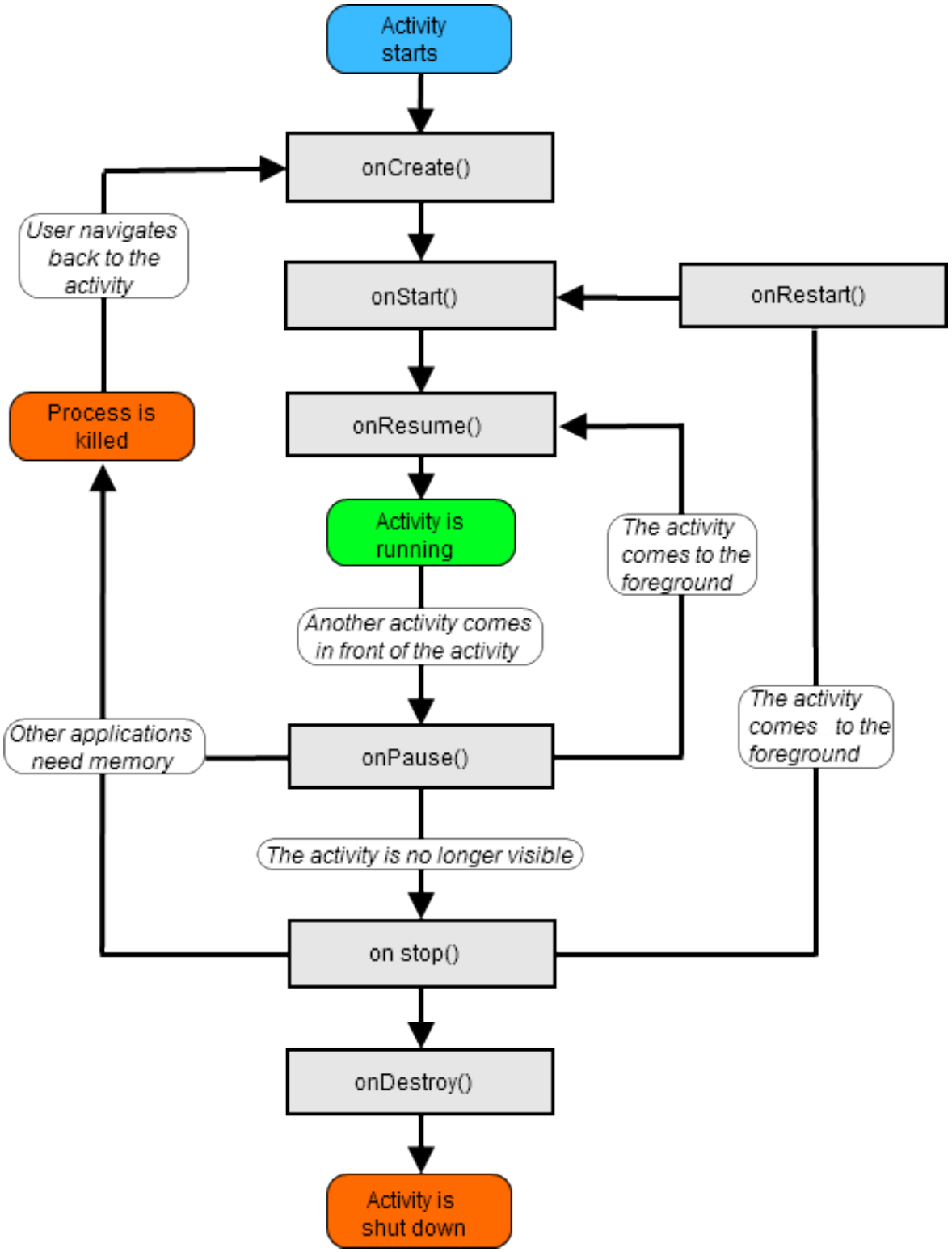
`starting`	In process of loading up, but not fully loaded.
`running`	Done loading and now visible on the screen.
`paused`	Partially obscured or out of focus, but not shut down.
`stopped`	No longer active, but still in the device's active memory.
`destroyed`	Shut down and no longer currently loaded in memory.

- Transitions between these states are represented by **events** that you can listen to in your activity code.
  - `onCreate, onPause, onResume, onStop, onDestroy, ...`

The following diagram shows activity lifecycle:

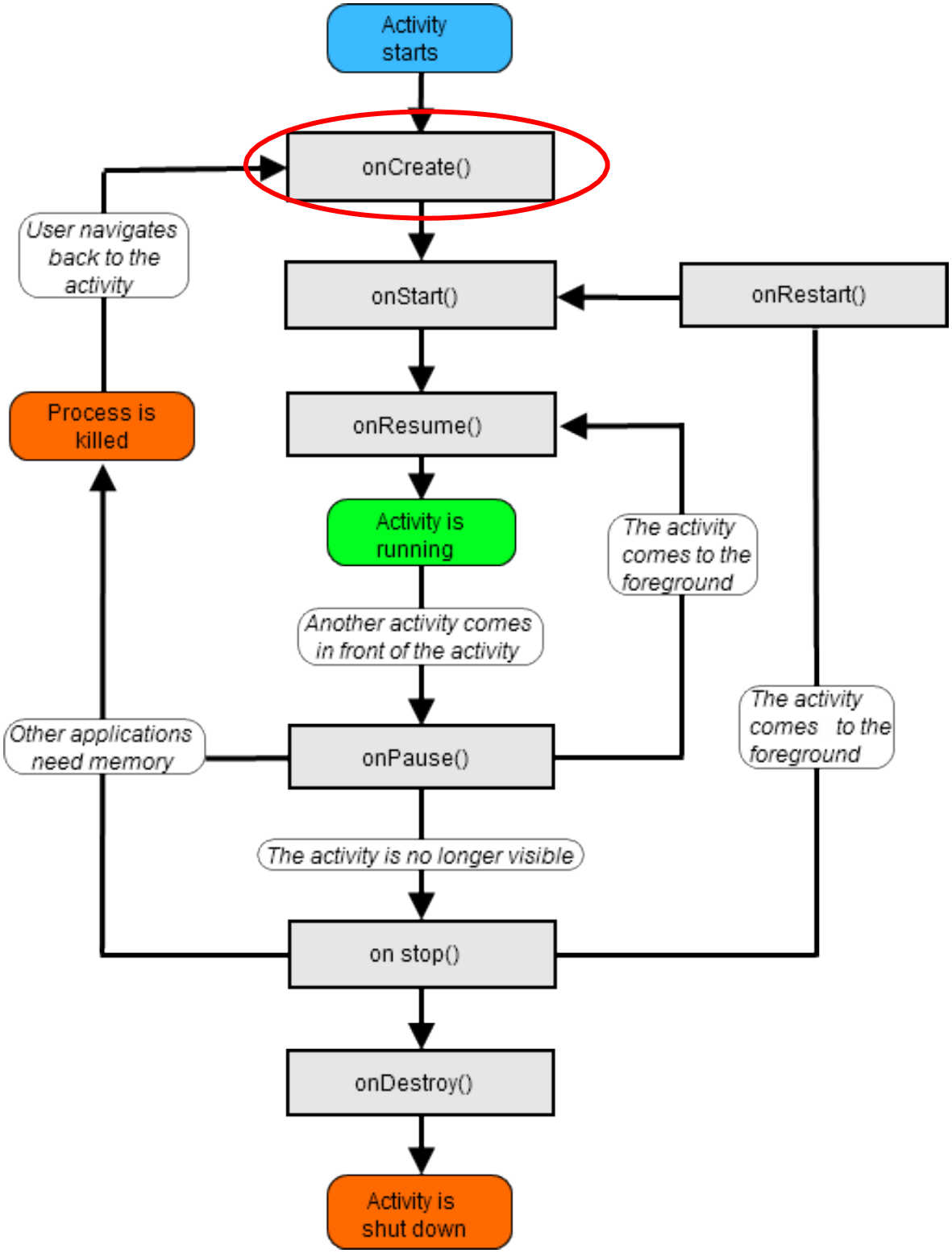


Activity lifecycle can be also represented by the following diagram:



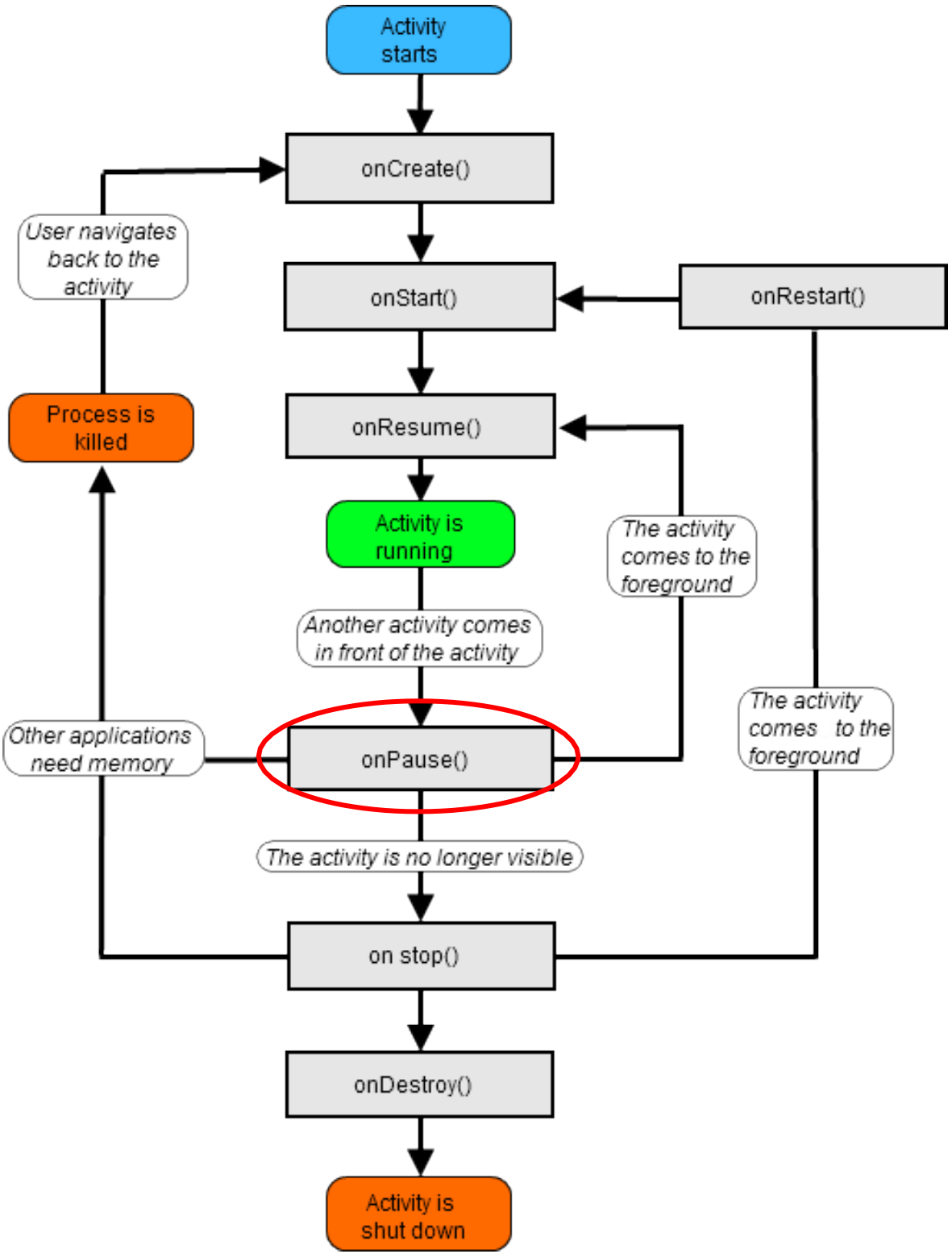
The OnCreate method:

- In `onCreate`, you create and set up the activity object, load any static resources like images, layouts, set up menus etc.
- after this, the Activity object exists
- think of this as the "constructor" of the activity


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
on stop()
onDestroy()
Activity is shut down

```
public class FooActivity extends Activity {
    ...
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); // always call super
setContentView(R.layout.activity_foo); // set up layout
any other initialization code; // anything else you need
    }
}
```

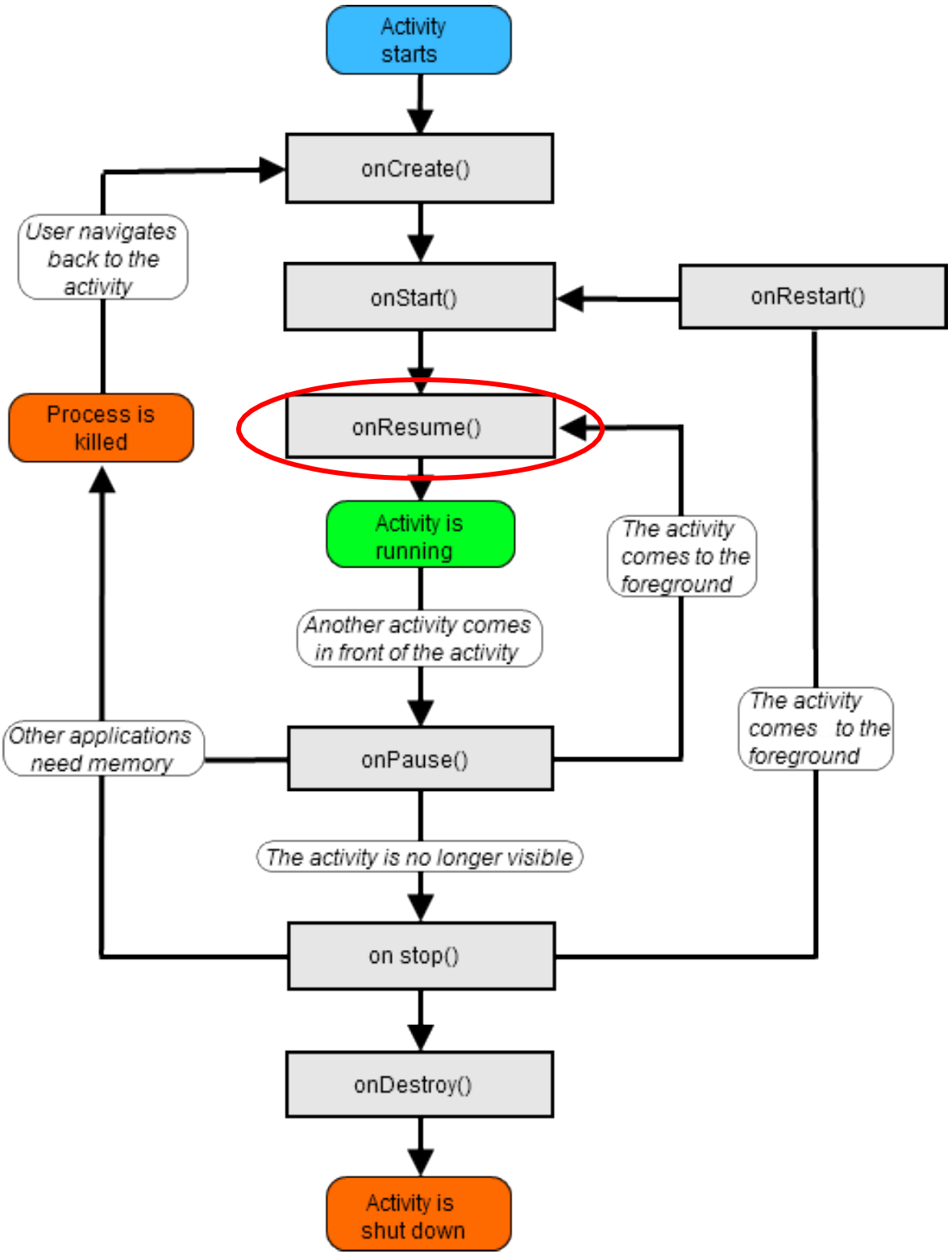
The OnPause method:



- When **`onPause`** is called, your activity is still partially visible.
- May be temporary, or on way to termination.
  - **Stop animations** or other actions that consume CPU.
  - **Commit unsaved changes** (e.g. draft email).
  - **Release system resources** that affect battery life.

```
public void onPause() {
        super.onPause(); // always call super
        if (myConnection != null) {
           myConnection.close(); // release resources
            myConnection = null;
        }
}
```

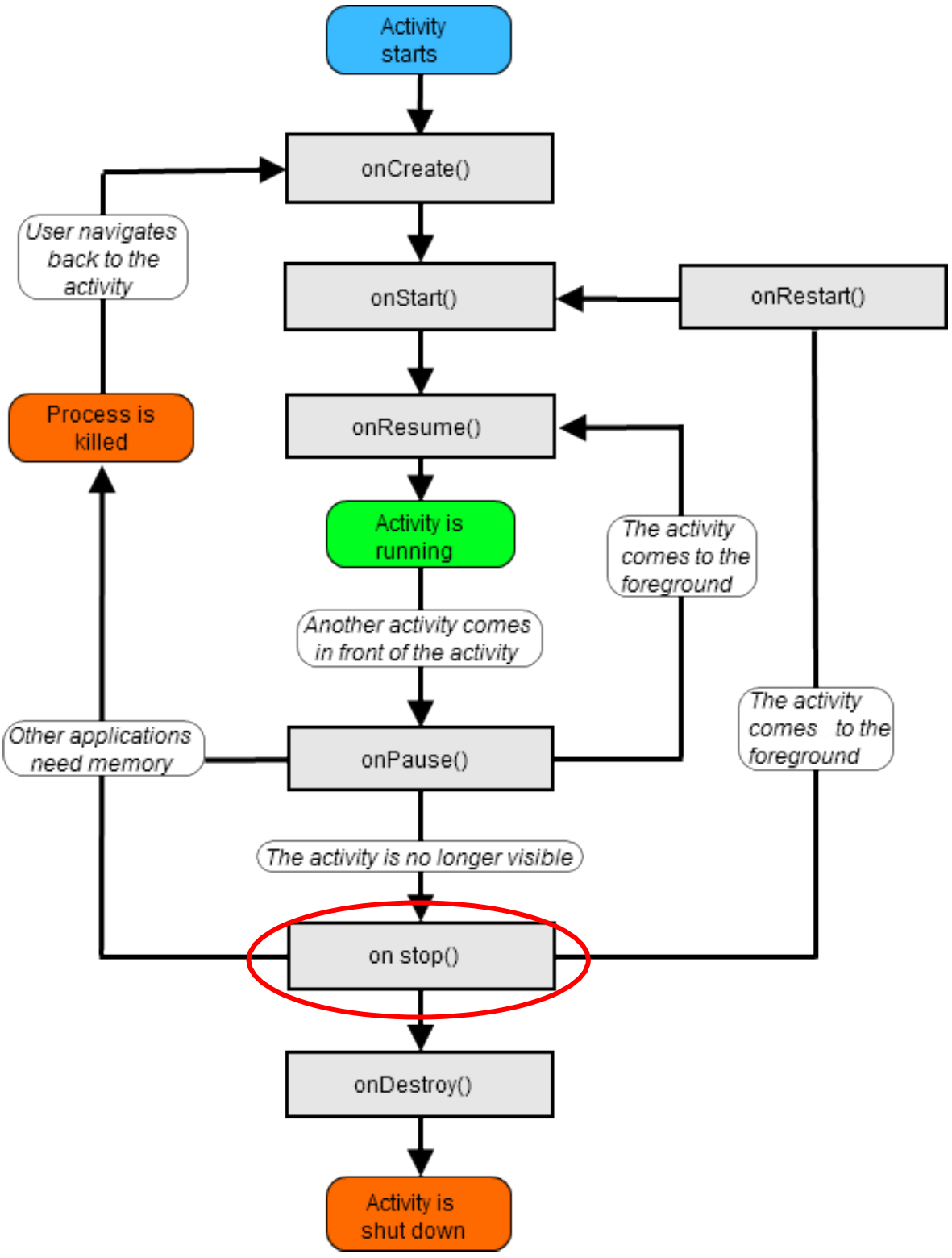
The OnResume method:



- When **`onResume`** is called, your activity is coming out of the Paused state and into the Running state again.
- Also called when activity is first created/loaded!
  - **Initialize resources** that you will release in `onPause`.
  - **Start / resume animations** or other ongoing actions that should only run when activity is visible on screen.

```
public void onResume() {
        super.onPause(); // always call super
        if (myConnection == null) {
         myConnection = new ExampleConnect(); //
init.resources
         myConnection.connect();
        }
 }
```

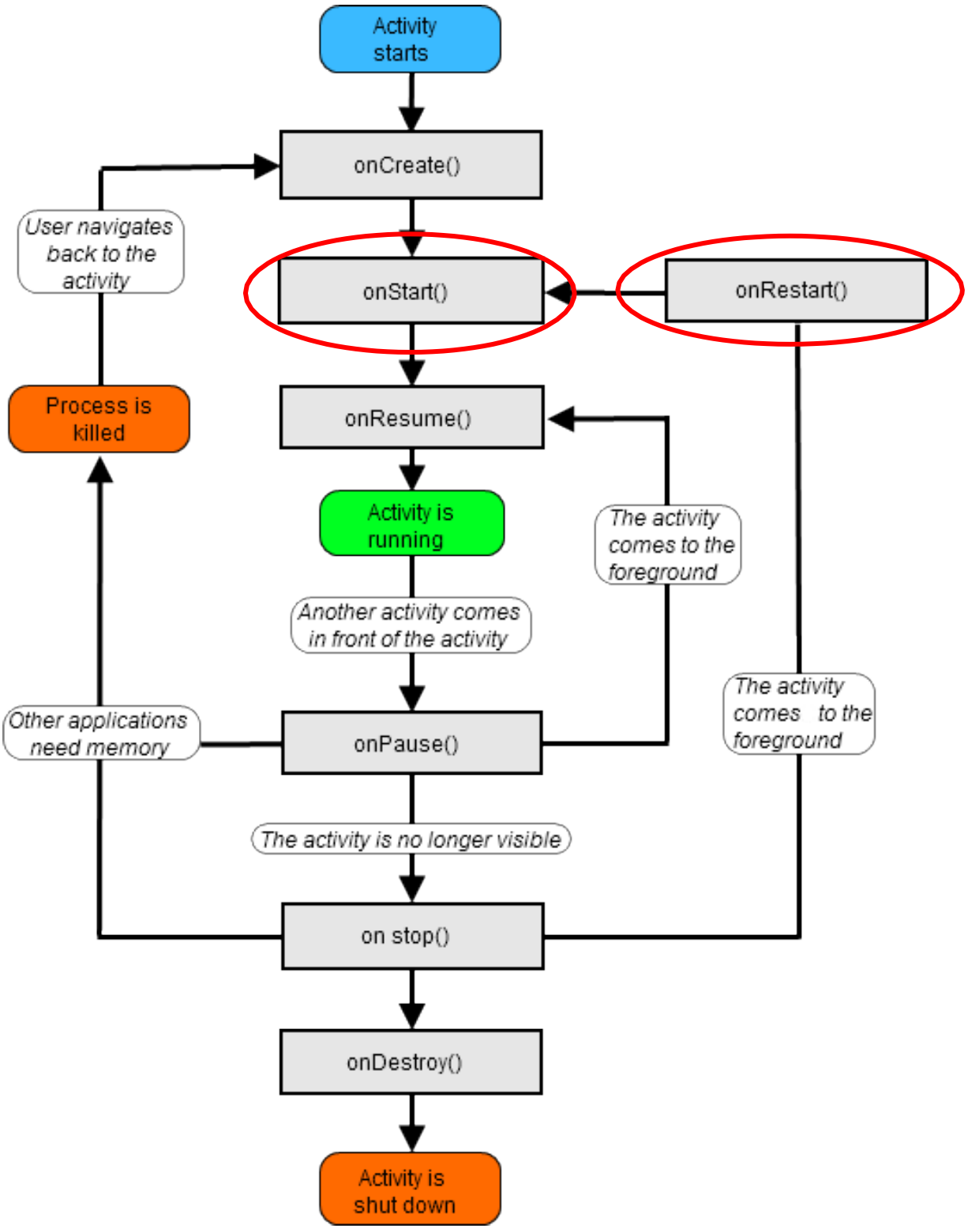
The OnStop method:



- When **`onStop`** is called, your activity is no longer visible on the screen:
  - User chose another app from **Recent Apps** window.
  - User starts **a different activity** in your app.
  - User receives a **phone call** while in your app.
- Your app might still be running, but that activity is not.
  - `onPause` is always called before `onStop`.
  - `onStop` performs heavy–duty shutdown tasks like writing to a database.

```
public void onStop() {
        super.onStop(); // always call super
        ...
        }
```

The OnStart and the OnRestart methods:



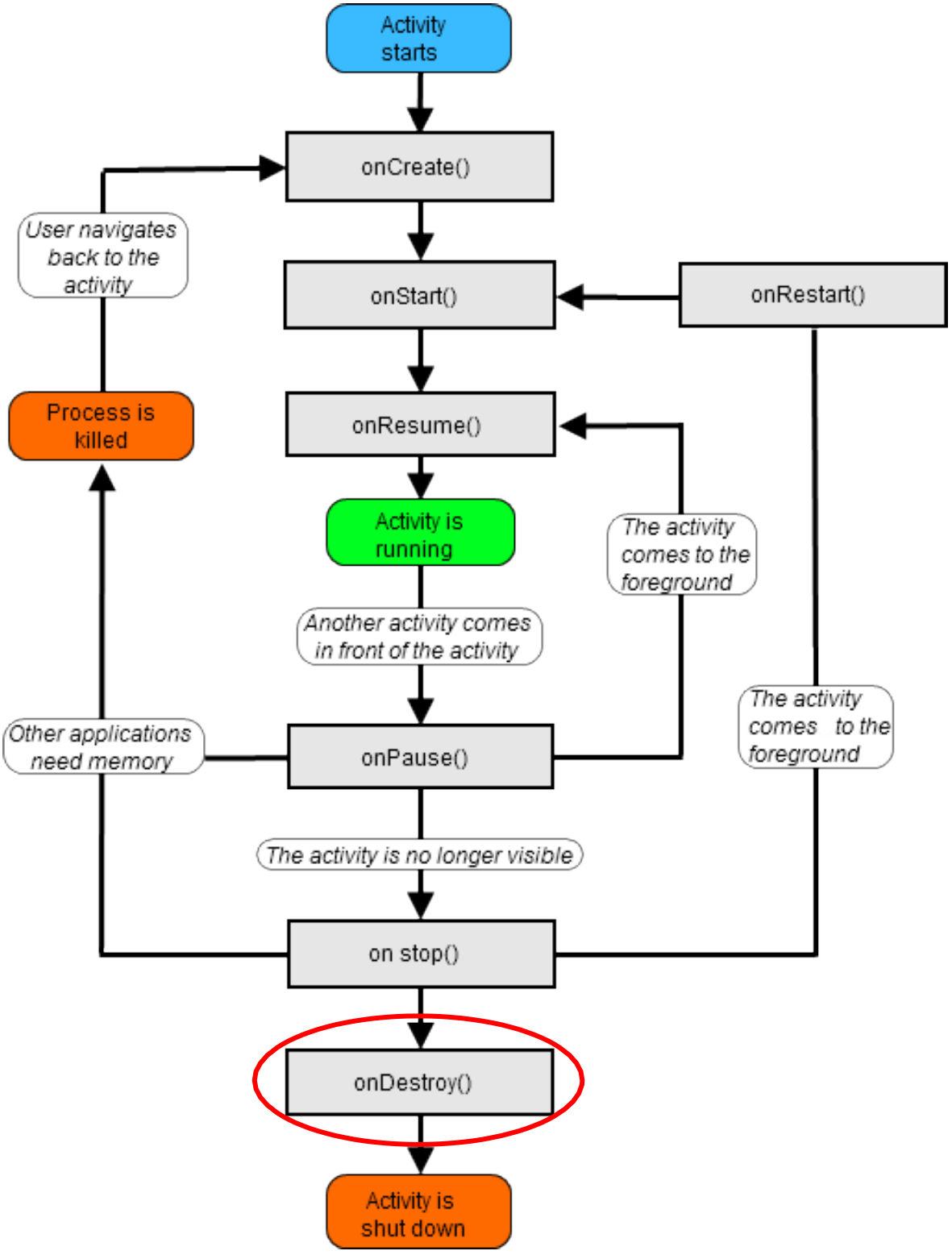
- **`onStart`** is called every time the activity begins.
- **`onRestart`** is called when activity was stopped but is started again later (all but the first start).
  - Not as commonly used; favor onResume.
  - Re–open any resources that onStop closed.

```
public void onStart() {
        super.onStart(); // always call super
        ...
        }
public void onRestart() {
        super.onRestart(); // always call super
        ...
        }
```

The OnDestroy method:

When **onDestroy** is called, your entire app is being shut down and unloaded from memory.

- Unpredictable exactly when/if it will be called.
- Can be called whenever the system wants to reclaim the memory used by your app.
- Generally favor `onPause` or `onStop` because they are called in a predictable and timely manner.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
on stop()
onDestroy()
Activity is shut down

```
public void onDestroy() {
        super.onDestroy(); // always call super
        ...
        }
```

Testing activity states:

- Use the `LogCat` system for logging messages when your app changes states:
  - analogous to `System.out.println` debugging for Android apps
  - appears in the `LogCat` console in Android Studio

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Log.d("testing", "onCreate got called");
}
```

Log methods:

`Log.d("tag", "message")`	debug message (for debugging)
`Log.e("tag", "message")`	error message (fatal error)
`Log.i("tag", "message")`	info message (low–urgency FYI)
`Log.v("tag", "message")`	verbose message (rarely shown)
`Log.w("tag", "message")`	warning message (non–fatal error)
`Log.wrf("tag", "message")`	log stack trace of an exception

Each method can also accept an optional exception argument:

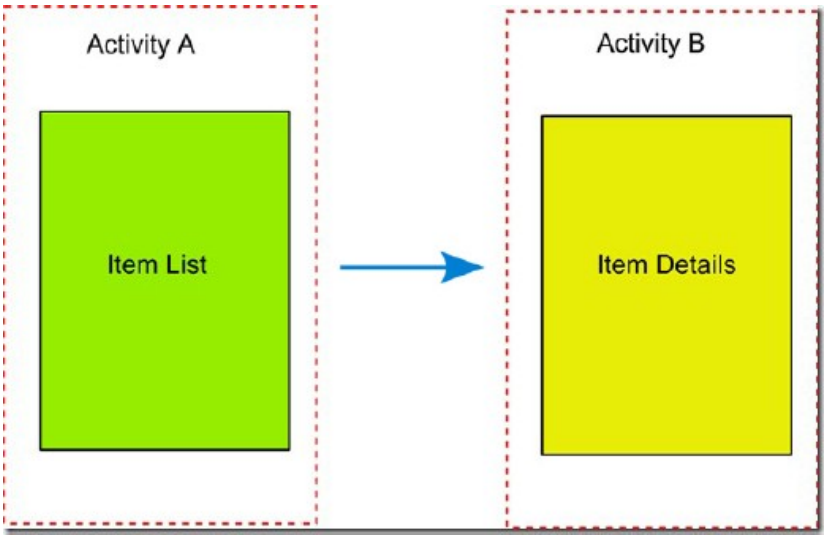
```
try { someCode(); }
        catch (Exception ex) {
        Log.e("error4", "something went wrong", ex);
}
```

2. Multiple activities and Intents

Learning outcomes:

Intents.

- Many apps have **multiple activities**.
  - Example: In an address book app, the main activity is a list of contacts, and clicking on a contact goes to another activity for viewing details.
  - An activity A can launch another activity B in response to an event.
  - The activity A can pass data to B.
  - The second activity B can send data back to A when it is done.

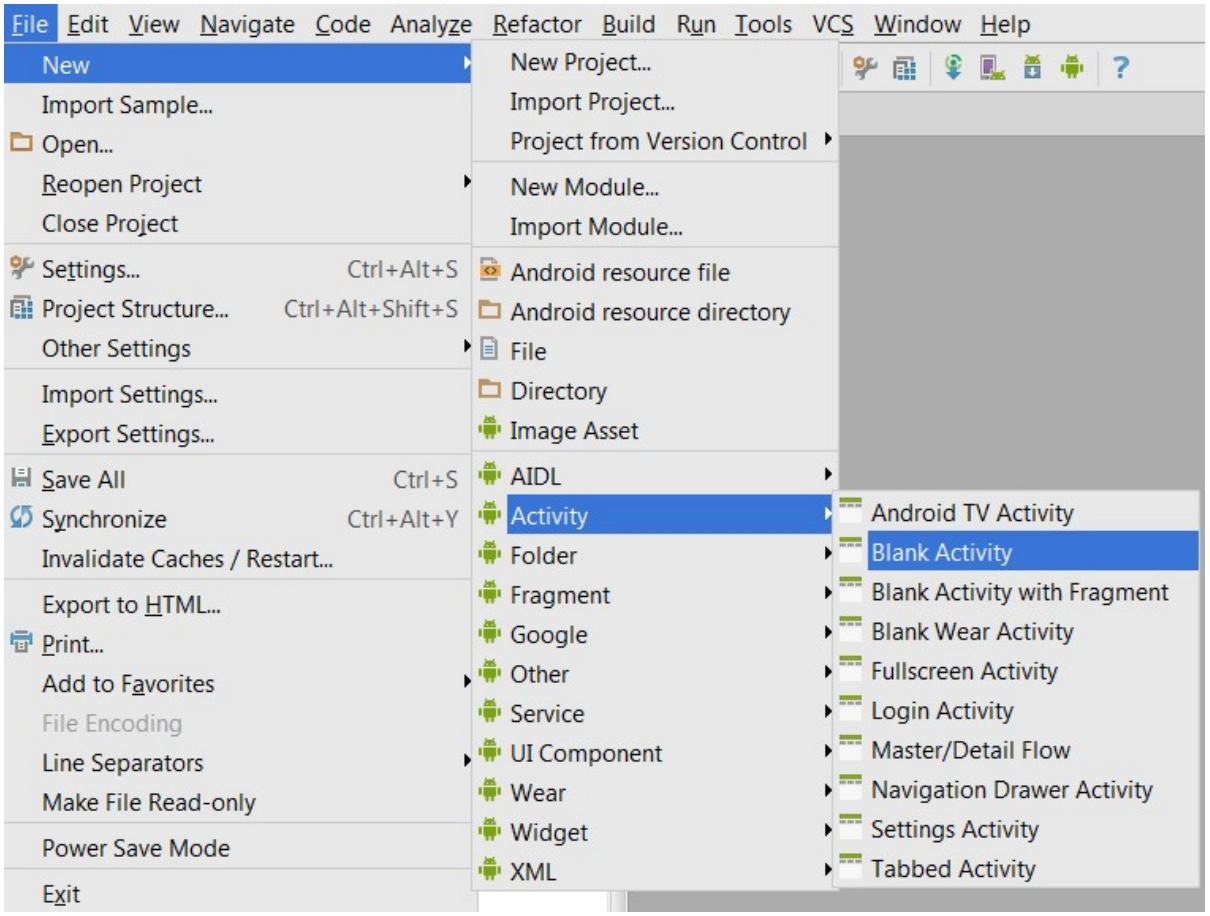


Adding an activity:

- in Android Studio:

File → New → Activity

  - creates a new **.XML file** in **res/layouts**
  - creates a new **.java class** in **src/java**
  - adds information to **AndroidManifest.xml** about the activity (without this information, the app will not allow the activity)

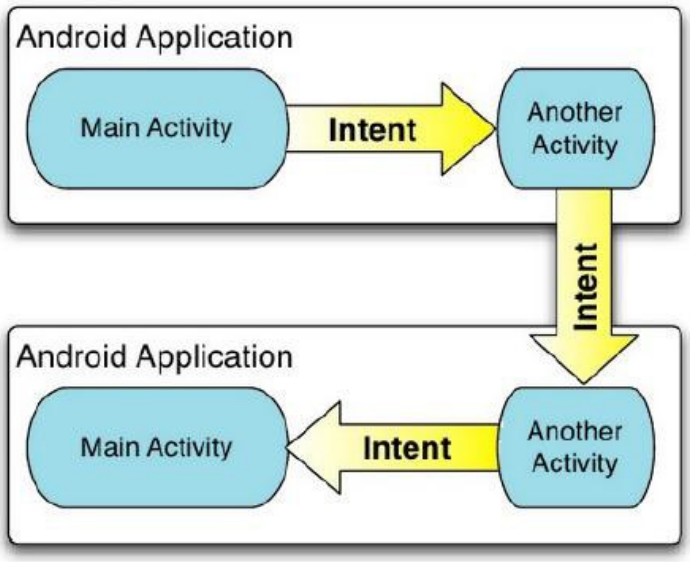


Every activity has an entry in project's **AndroidManifest.xml**, added automatically by Android Studio:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.basel.intents" >

<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
<activity
  android:name=".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=".DetailsActivity"
   android:label="@string/title_activity_details" >
</activity>
</application>
</manifest>
```

Intents:

- **intent:** a bridge between activities; a way for one activity to invoke another
  - the activity can be in the same app or in a different app
  - can store **extra data** to pass as "parameters" to that activity
  - second activity can "**return**" information back to the caller if needed



Creating an Intent:

- To launch another activity (usually in response to an event), create an Intent object and call startActivity with it:

```
Intent intent = new Intent(this, ActivityName.class);
startActivity(intent);
```

- If you need to pass any parameters or data to the second activity, call putExtra on the intent.

  It stores "extra" data as key/value pairs, not unlike a Map.

```
Intent intent = new Intent(this, ActivityName.class);
intent.putExtra("name1", value1);
intent.putExtra("name2", value2);
startActivity(intent);
```

Extracting extra data:

- In the second activity that was invoked, you can grab any extra data that was passed to it by the calling act.
  - You can access the Intent that spawned you by calling `getIntent`.
  - The Intent has methods like `getExtra`, `getIntExtra`, `getStringExtra`, etc. to extract any data that was stored inside the intent.

```
public class SecondActivity extends Activity {
    ...
    public void onCreate(Bundle savedState) {
        super.onCreate(savedState);
        setContentView(R.layout.activity_second);
        Intent intent = getIntent();
        String extra = intent.getExtra("name");
        ...
    }
}
```

Waiting for a result:

- In the second activity that was invoked, you can grab any extra data that was passed to it by the calling act.
  - You can access the Intent that spawned you by calling `getIntent`.
  - The Intent has methods like `getExtra`, `getIntExtra`, `getStringExtra`, etc. to extract any data that was stored inside the intent.

Sending back a result:

- In the second activity that was invoked, send data back:
  - Need to create an Intent to go back.
  - Store any extra data in that intent; call setResult and finish.

```
public class SecondActivity extends Activity {
    ...
    public void myOnClick(View view) {
        Intent intent = new Intent();
        intent.putExtra("name", value);
        setResult(RESULT_OK, intent);
        finish(); // calls onDestroy
    }
}
```

Grabbling the result:

The following statements show how to get the results in the first activity:

```
public class FirstActivity extends Activity {
 private static final int REQ_CODE = 123; // MUST be 0-65535
 public void myOnClick(View view) {
  Intent intent = getIntent(this, SecondActivity.class);
  startActivityForResult(intent, REQ_CODE);
}
protected void onActivityResult
   (int requestCode, int resultCode, Intent intent) {
super.onActivityResult(requestCode, resultCode, intent);
if (requestCode == REQ_CODE) {
// came back from SecondActivity
  String data = intent.getStringExtra("name");
  Toast.makeText(this, "Got back: " + data,
                    Toast.LENGTH_SHORT).show();
        }
    }
}
```

Implicit intent:

- implicit intent: One that launches another app, without naming that specific app, to handle a given type of request or action.

examples: invoke default browser; load music player to play a song

```
// make a phone call
Uri number = Uri.parse("tel:5551234");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
// go to a web page in the default browser
Uri webpage =  Uri.parse("http://www.stanford.edu/");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
// open a map pointing at a given latitude/longitude (z=zoom)
Uri location = Uri.parse("geo:37.422219,-122.08364?z=14");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

Playing audio with MediaPlayer:

Find sound files such as .WAV, .MP3

- put sound files in project folder app / **src** / **main** / **res** / **raw**
- in Java code, refer to audio file as R.raw.**filename**
  - (don't include the extension; R.raw.foo for foo.mp3)
  - use simple file names with only letters and numbers

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.filename);
mp.start();
```

other methods: `stop`, `pause`, **`isLooping`**, **`isPlaying`**, **`getCurrentPosition`**, **`release`**, **`seekTo`**, **`setDataSource`**, **`setLooping`**

3. Example

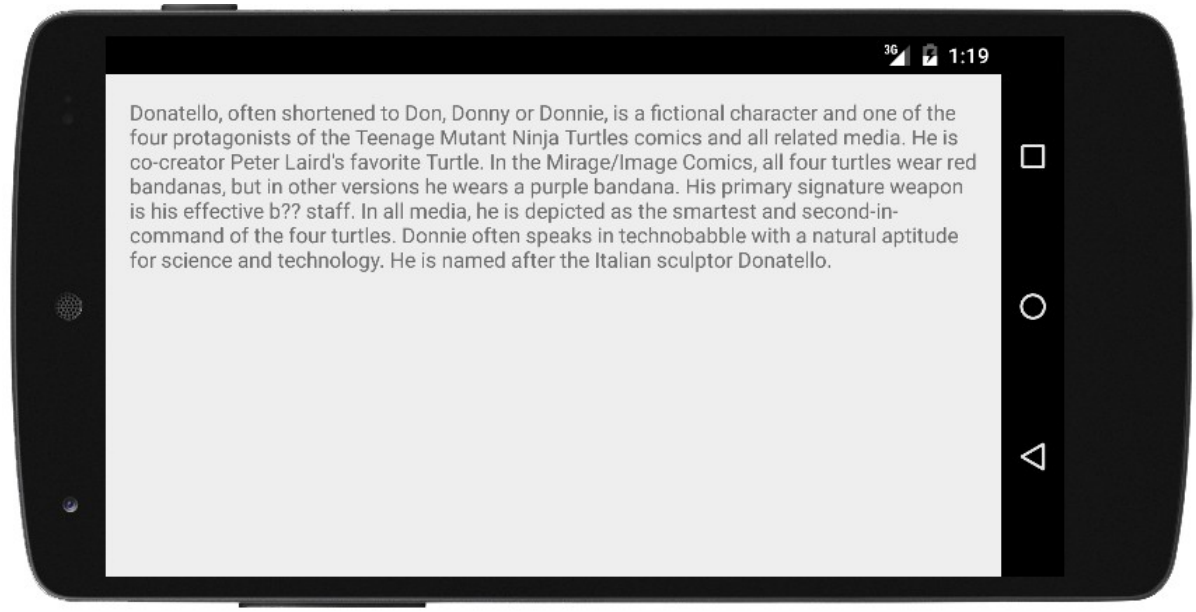
Learning outcomes:

Example

- When the application launches, a music track starts playing and the next interface appears:

- When you click one of the radio buttons, the corresponding profile picture appears:

- When you click on a profile, you go to a second interface to view the profile data:



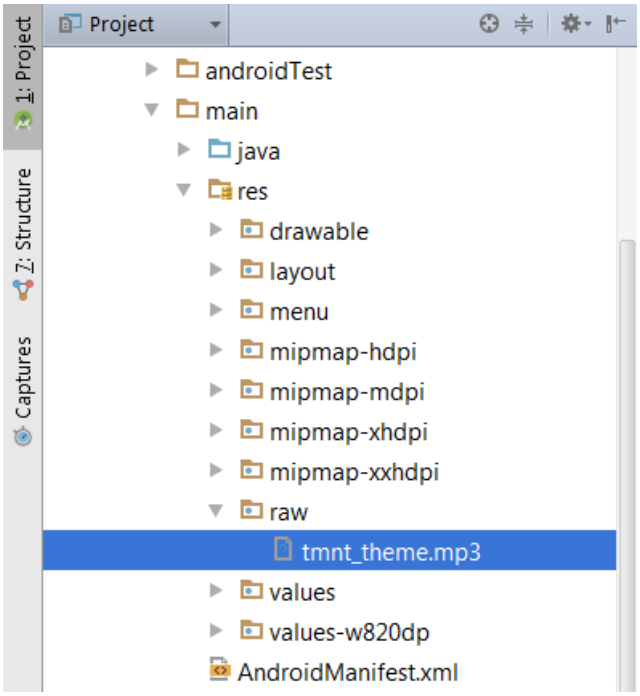
First Interface:

The first interface contains four radio buttons and an image button.

The activity_main.xml file is:

```
<LinearLayout
  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:gravity="top|center" android:orientation="vertical"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

<RadioGroup android:id="@+id/turtle_group"
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
<RadioButton android:id="@+id/leo" android:onClick="pickTurtle"
            android:text="Leo" android:checked="true"
             android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

<RadioButton android:id="@+id/mike" android:onClick="pickTurtle"
android:text="Mike" android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
<RadioButton android:id="@+id/don" android:onClick="pickTurtle"
android:text="Don" android:layout_width="wrap_content"
 android:layout_height="wrap_content"/>

<RadioButton android:id="@+id/raph" android:onClick="pickTurtle"
android:text="Raph" android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
</RadioGroup>

<ImageButton android:id="@+id/turtle"
        android:onClick="onClickTurtleImage"
        android:src="@drawable/tmntleo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

The sound file tmnt_theme.mp3 is contained within the res \ raw folder.



The MainActivity.java code file is:

```
package com.example.basel.intents;
import android.app.Activity;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ImageButton;
import android.widget.RadioGroup;
public class MainActivity extends Activity {
private MediaPlayer player;
/*
* Called when the activity first gets created.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        player = MediaPlayer.create(this,
                                    R.raw.tmnt_theme);

        Log.d("testing", "onCreate got called");
    }
public void onStart() {
```

```
        super.onStart();
        Log.d("testing", "onStart got called");
    }
public void onResume() {
        super.onResume();
        if (player != null) {
            player.setLooping(true);
            player.start();
        }
        Log.d("testing", "onResume got called");
    }
public void onPause() {
        super.onPause();
        player.pause();
        Log.d("testing", "onPause got called");
    }
public void onStop() {
        super.onStop();
        Log.d("testing", "onStop got called");
    }
public void onDestroy() {
        super.onDestroy();
        Log.d("testing", "onDestroy got called");
    }
/*
* Called when the Details activity finishes running and comes
back to here.
*/
@Override
protected void onActivityResult
   (int requestCode, int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);

    }
/*
* Called when the user clicks on the large TMNT image button.
* Loads the DetailsActivity for more information about that
turtle.
*/
public void onClickTurtleImage(View view) {
Intent intent = new Intent(this, DetailsActivity.class);

RadioGroup group = (RadioGroup)
                findViewById(R.id.turtle_group);
int id = group.getCheckedRadioButtonId();
intent.putExtra("turtle_id", id);
startActivity(intent);
}
/*
* This method is called when the user chooses one of the
turtle radio buttons.
```

```
* In this code we set which turtle image is visible on the
screen in the ImageView.
*/
public void pickTurtle(View view) {
ImageButton img = (ImageButton)
                    findViewById(R.id.turtle);
        if (view.getId() == R.id.leo) {
            img.setImageResource(R.drawable.tmntleo);
        } else if (view.getId() == R.id.mike) {
            img.setImageResource(R.drawable.tmntmike);
        } else if (view.getId() == R.id.don) {
            img.setImageResource(R.drawable.tmntdon);
        } else if (view.getId() == R.id.raph) {
            img.setImageResource(R.drawable.tmntraph);
        }
    }
}
```

Second Interface:

The second interface contains only a text view control.

The activity_details.xml file is:

```
<LinearLayout
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:gravity="top|center"
    android:orientation="vertical"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"

tools:context="com.example.stepp.layoutfun.DetailsActivity">
<TextView android:id="@+id/turtle_info"
        android:text=""
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

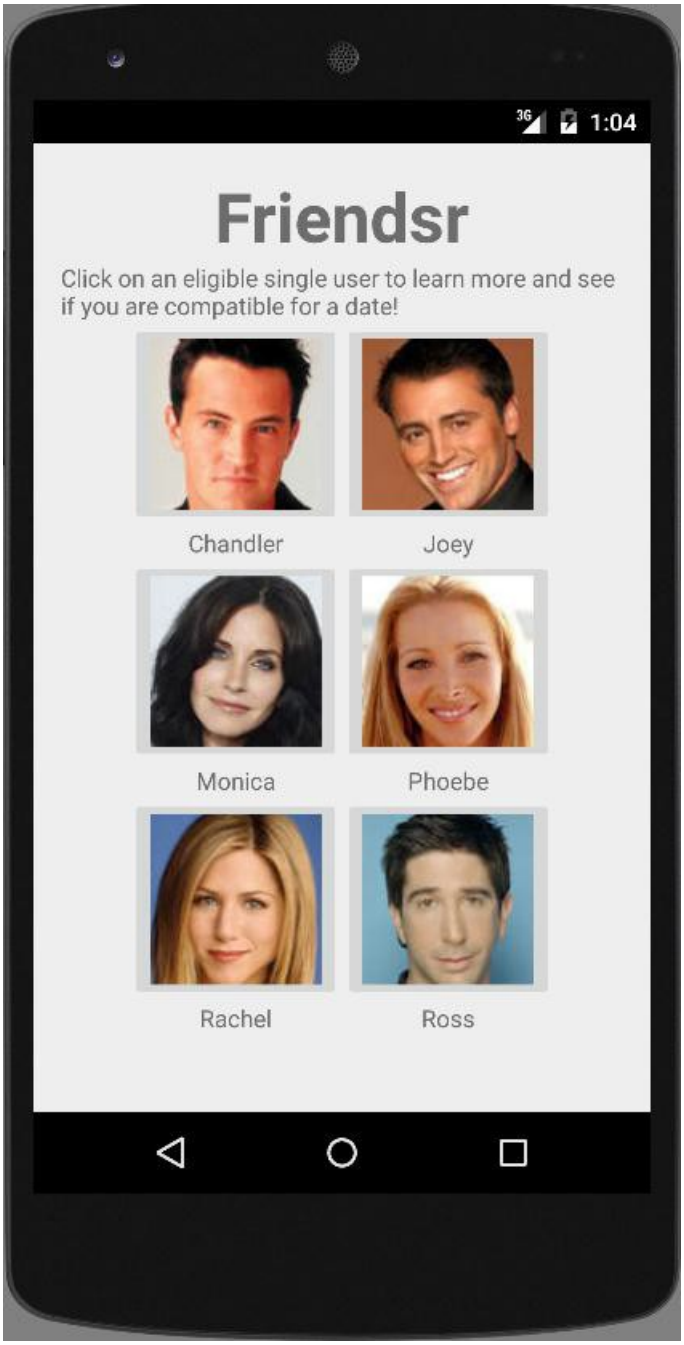
And the DetailsActivity.java code file:

```
package com.example.basel.intents;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class DetailsActivity extends Activity {
/*
* Constant array of data about each of the four turtles.
*/
private static final String[] TURTLE_DETAILS = {
            "Leonardo, or Leo,.... ",
            "Michelangelo, Mike or Mikey ….
            "Donatello, often shortened to Don.... ",
            "Raphael, or Raph, is a….. "
    };

/*
* Called when the activity first gets created.
```

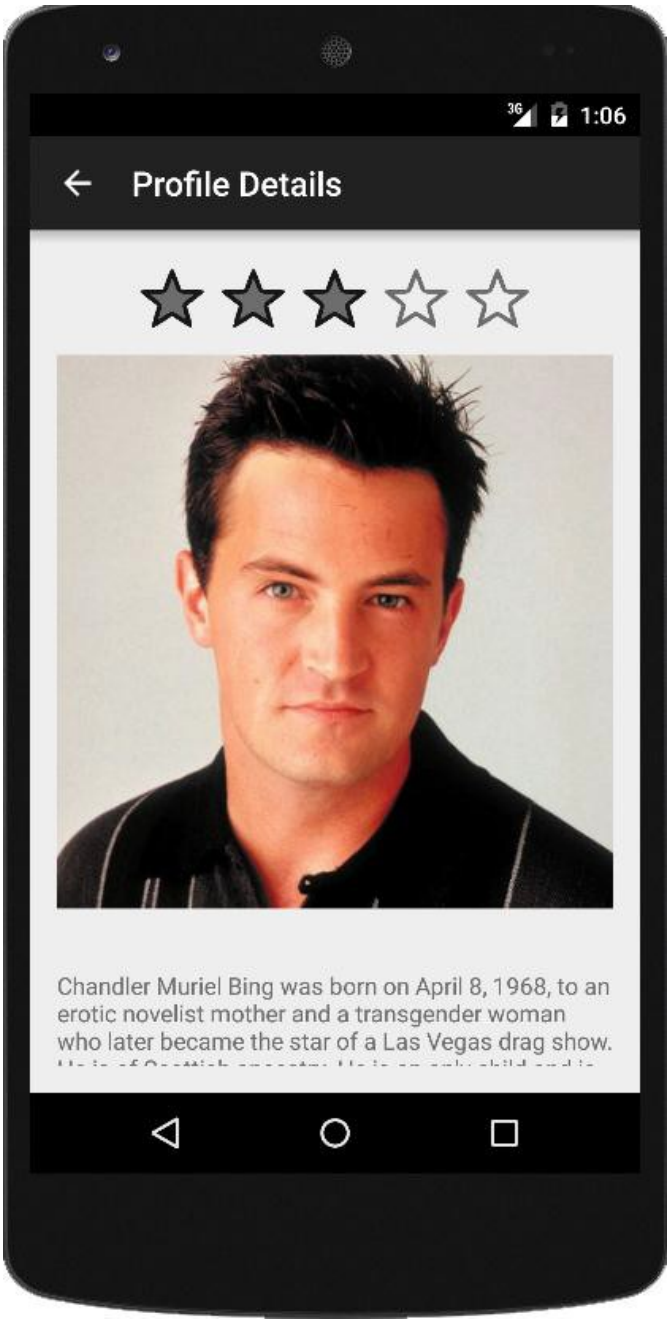
```
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details);

// pull the turtle's ID out of the intent that the
MainActivity used to load me
Intent intent = getIntent();
int id = intent.getIntExtra("turtle_id", R.id.leo);
String text = "";
if (id == R.id.leo) {
            text = TURTLE_DETAILS[0];
        } else if (id == R.id.mike) {
            text = TURTLE_DETAILS[1];
        } else if (id == R.id.don) {
            text = TURTLE_DETAILS[2];
        } else { // if (id == R.id.raph)
            text = TURTLE_DETAILS[3];
 }
TextView tv = (TextView) findViewById(R.id.turtle_info);
tv.setText(text);
    }
}
```

Exercise: online dating application:

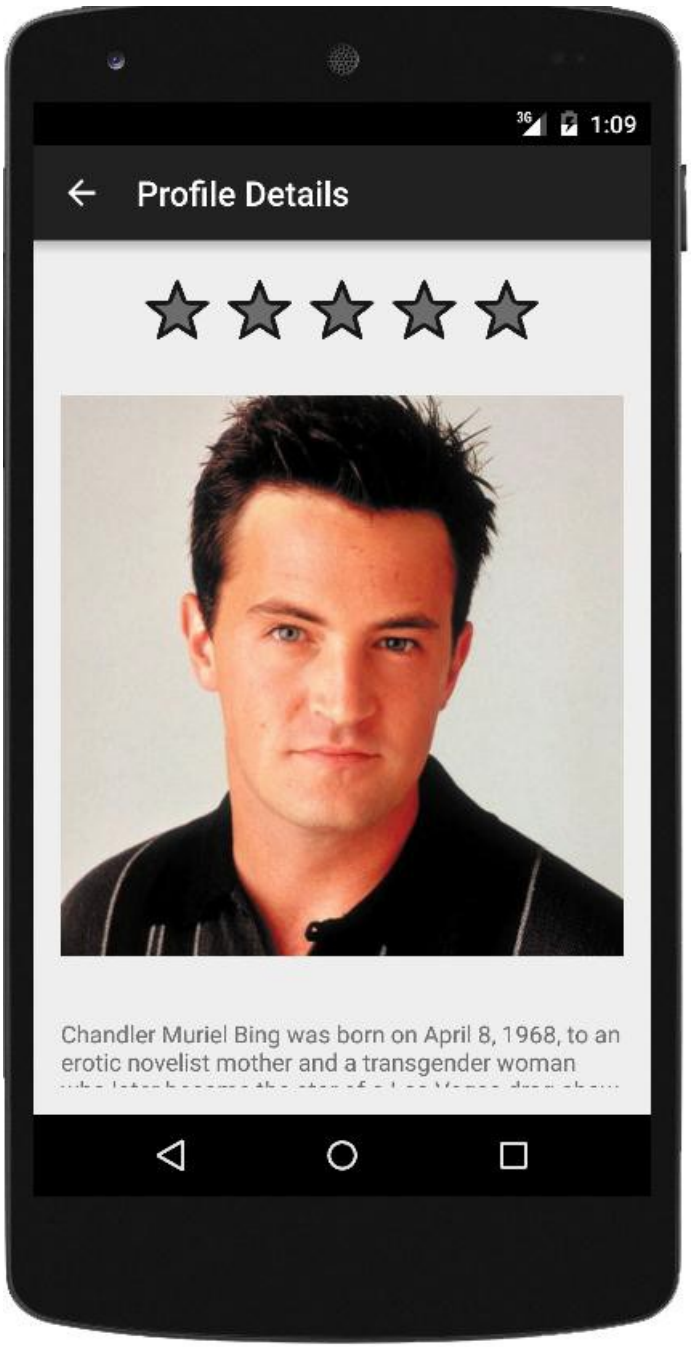
When you open the application, an interface containing the images of the candidates appears:

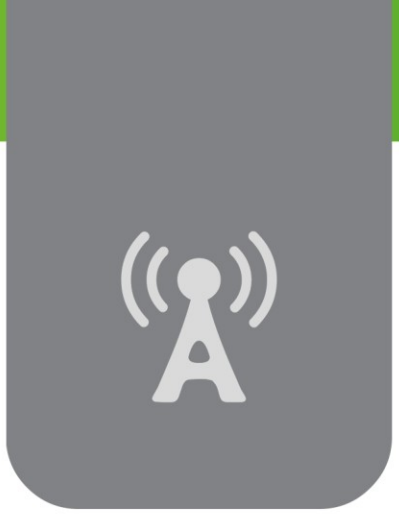


When a picture is clicked, a second interface appears with a large picture of the same person and some information about it:



A rating can be given to a person by clicking on a number of stars:





Chapter 9:
Activity State, Preferences, Files and Camera

Key Words:

Instance state, non-persistent/permanent state, preferences, storage, Camera.

Summary:

This unit shows how to handle activity state, using internal and external storage, using the camera.

Outcomes:

Student will learn in this unit:

- Maintaining activity state.
- Using files and streams.
- Using the camera.

Plan:

4 Learning Objects

1. Storage
2. Activity State
3. The Camera
4. Example

1. Activity State

Learning outcomes:

Activity state.

- **instance state**: Current state of an activity.
  - Which boxes are checked
  - Any text typed into text boxes
  - Values of any private fields

  …
- Example: In the app at right, the instance state is that the Don checkbox is checked, and the Don image is showing.

Lost activity state:

- Several actions can cause your activity state to be lost:
  - When you go from one **activity** to another and back, within same app
  - When you launch another **app** and then come back
  - When you rotate the device's **orientation** from portrait to landscape

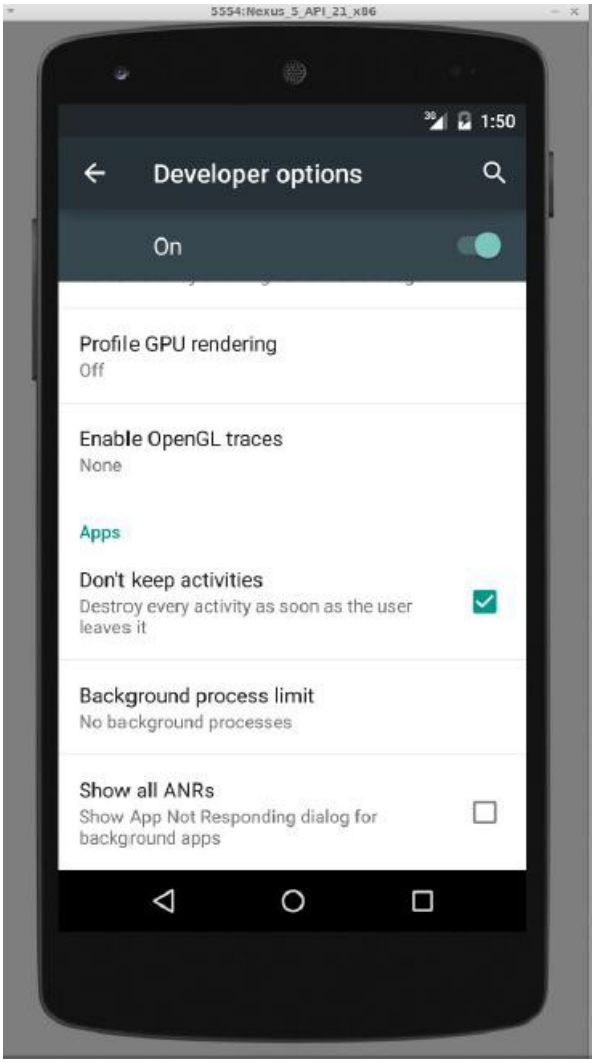
    …

Simulating state change in AVD:

- Testing orientation change: press Ctrl-F11
- Testing activity shutdown (`onDestroy`):
  - Settings → Developer options → **Don't keep activities**
  - Developer options → Background process limit → **No bg processes**

- تذكر أنو يمكنك تغيير حالة االتجاه)طولي أم عرضي(بضغط Ctrl+F11.
- الختبار إنهاء النشاطات)الحدث onDestroy(، قم بالدخول إلى اإلعدادات:

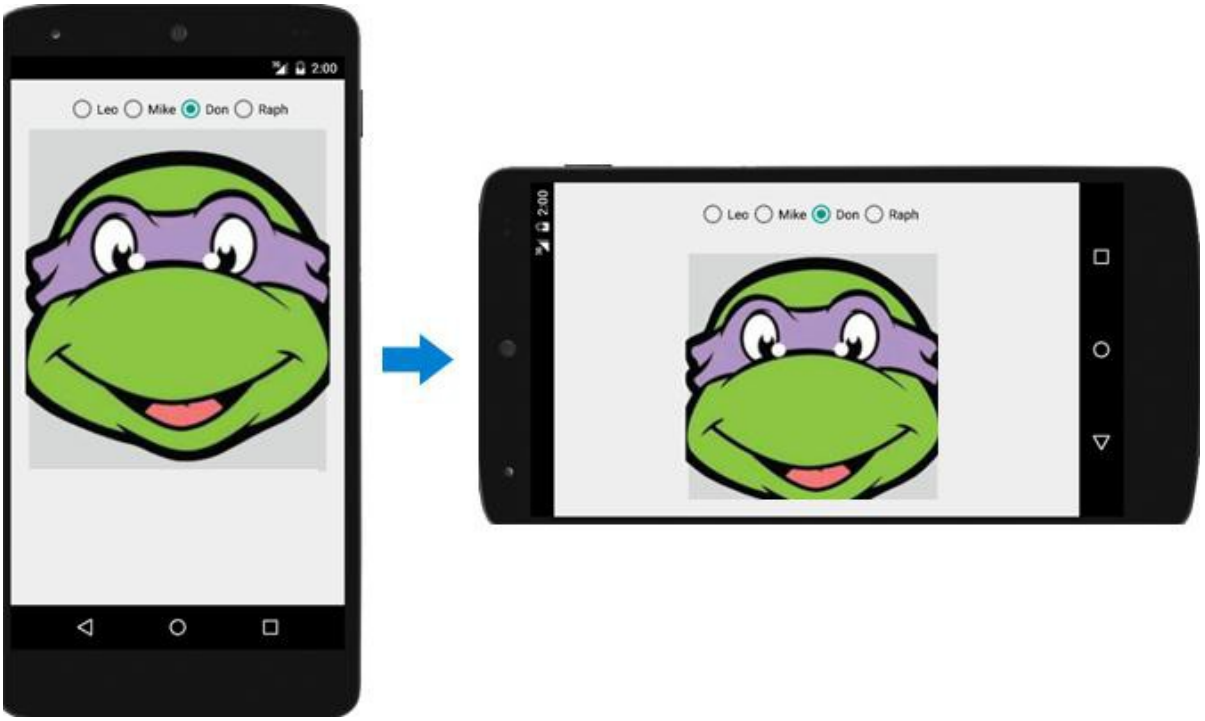
Settings → Developer options → Don't keep activities
Developer options → Background process limit → No background processes



Handling rotation:

- A quick way to retain your activity's GUI state on rotation is to set the `configChanges` attribute of the activity in **AndroidManifest.xml**.
  - This doesn't solve the other cases like loading other apps / activities.

```
<activity android:name=".MainActivity"
android:configChanges="orientation|screenSize"...>
```



non-persistent/permanent state:

A non-permanent state of an activity is the state that does not last after the application is terminated, like controls values in general for example.

Values that we want to maintain permanently, are called permanent like the application settings for example.

non-persistent state:

The non-permanent state of the activity is maintained by using the *onSaveInstanceState* procedure and restored by using the *onRestoreInstanceState* procedure.

- When an activity is being destroyed, the event method **onSaveInstanceState** is also called.
  - This method should save any "non-persistent" state of the app.
  - **non-persistent state: Stays** for now, but lost on shutdown/reboot.
- Accepts a **Bundle** parameter storing key/value pairs.
  - Bundle is passed back to activity if it is recreated later.

```
public void onSaveInstanceState(Bundle outState) {
super.onSaveInstanceState(outState); // always call super
outState.putInt("name", value);
outState.putString("name", value);
...
}
```

- `public void` **`onRestoreInstanceSt`** When an activity is recreated later, the event method `onRestoreInstanceState` is called. *
  - This method can restore any "non-persistent" state of the app.
  - **Bundle** from `onSaveInstanceState` from before is passed back in.

```
ate(Bundle inState) {
super.onRestoreInstanceState(inState); // always call super
int name = inState.getInt("name");
String name = inState.getString("name");
...
}
```

Saving your own classes:

- By default, your own classes can't be put into a Bundle.
- You can make a class able to be saved by implementing the (methodless) `java.io.Serializable` interface.

```
public class Date implements Serializable {
...
}
public class MainActivity extends Activity {
public void onSaveInstanceState(Bundle outState) {
super.onSaveInstanceState(outState);
Date d = new Date(2015, 1, 25);
outState.putSerializable("today", d);
   }
}
```

Example:

We save in the following example the selected radio value:

```
protected void onSaveInstanceState(Bundle outState) {
super.onSaveInstanceState(outState);
Log.d("testing", "onSaveInstanceState got called");
RadioGroup group =
       (RadioGroup)findViewById(R.id.turtle_group);
int id = group.getCheckedRadioButtonId();
outState.putInt("id", id);
}
```

Value are retrieved:

```
protected void onRestoreInstanceState
                        (Bundle savedInstanceState) {
super.onRestoreInstanceState(savedInstanceState);
Log.d("testing", "onRestoreInstanceState got called");

int id = savedInstanceState.getInt("id");
RadioGroup group =
        (RadioGroup)findViewById(R.id.turtle_group);
group.check(id);
updateTurtleImage();

}
```

Preferences:

- `SharedPreferences` object can store permanent settings and data for your app.
  - stores key / value pairs similar to a Bundle or Intent
  - pairs added to SharedPreferences persist after shutdown / reboot (unlike savedInstanceState bundles)
- Two ways to use it:
  - per–activity (`getPreferences`)
  - per–app (`getSharedPreferences`)
- Saving per activity:

```
SharedPreferences prefs = getPreferences(MODE_PRIVATE);
SharedPreferences.Editor prefsEditor = prefs.edit();
prefsEditor.putInt("name", value);
prefsEditor.putString("name", value);
...
prefsEditor.apply(); // or commit();
```

- Restoring values:

```
SharedPreferences prefs = getPreferences(MODE_PRIVATE);
int i = prefs.getInt("name", defaultValue);
String s = prefs.getString("name", "defaultValue");
...
```

- Saving per app:

```
SharedPreferences prefs = getSharedPreferences
                                    ("filename", MODE_PRIVATE);
SharedPreferences.Editor prefsEditor = prefs.edit();
prefsEditor.putInt("name", value);
prefsEditor.putString("name", value);
...
prefsEditor.commit();
```

- Restoring values:

```
SharedPreferences prefs = getSharedPreferences
                                    ("filename", MODE_PRIVATE);
int i = prefs.getInt("name", defaultValue);
String s = prefs.getString("name", "defaultValue");
...
```

2. Storage

Learning outcomes:

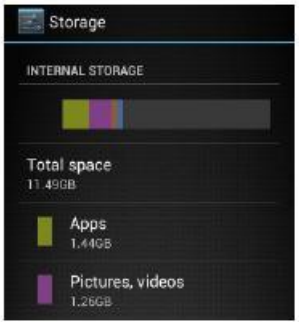
using storage.

Internal and external storage:

- Android can read/write files from two locations:
  - **internal** and **external** storage.
  - Both are **persistent** storage; data remains after power–off / reboot.

Internal storage:

- Built into the device.
  - guaranteed to be present
  - typically smaller (~1–4 gb)
  - can't be expanded or removed
  - specific and private to each app
  - wiped out when the app is uninstalled



File:

- `java.io.File` – Objects that represent a file or directory.

methods: `canRead`, `canWrite`, `create`, `delete`, `exists`, `getName`, `getParent`, `getPath`, `isFile`, `isDirectory`, `lastModified`, `length`, `listFiles`, `mkdir`, `mkdirs`, `renameTo`

Streams:

- `java.io.InputStream, OutputStream` – Stream objects represent flows of data bytes from/to a source or destination.
  - Could come from a file, network, database, memory, …
  - Normally not directly used; they only include low–level methods for reading/writing a byte (character) at a time from the input.
  - Instead, a stream is often passed as parameter to other objects like `java.util.Scanner, java.io.BufferedReader, java.io.PrintStream` to do the actual reading / writing.

Using internal storage:

- An activity has methods you can call to read/write files:
  - `getFilesDir ()` – returns internal directory for your app
  - `getCacheDir ()` – returns a "temp" directory for scrap files
  - `getResources ().openRawResource (R.raw.id)`
    read an input file from res/raw/)
  - `openFileInput("name", mode)`
    opens a file for reading
  - `openFileOutput("name", mode)`
    opens a file for writing
- You can use these to read/write files on the device.
  - many methods return standard `java.io.File` objects
  - some return `java.io.InputStream` or `OutputStream` objects, which can be used with standard classes like `Scanner, BufferedReader,` and `PrintStream` to read/write files (see Java API)

Examples:

- The following example shows how to read a source file and display its content:

```
// read a file, and put its contents into a TextView
// (assumes hello.txt file exists in res/raw/ directory)
Scanner scan = new Scanner
                (getResources().openRawResource(R.raw.hello));
String allText = ""; // read entire file
while (scan.hasNextLine()) {
 String line = scan.nextLine();
 allText += line;
}
myTextView.setText(allText);
scan.close();
```

- Read / Write example:

```
// write a short text file to the internal storage
PrintStream output = new PrintStream
                 )openFileOutput("out.txt", MODE_PRIVATE));
output.println("Hello, world!");
output.println("How are you?");
output.close();
 ...
// read the same file, and put its contents into a TextView
Scanner scan = new Scanner
    (openFileInput("out.txt", MODE_PRIVATE));
String allText = ""; // read entire file
while (scan.hasNextLine()) {
        String line = scan.nextLine();
        allText += line;
}
myTextView.setText(allText);
scan.close();
```

External Storage:

- external storage: Card that is inserted into the device.
  (such as a MicroSD card)
  - can be much larger than internal storage (~8–32 gb)
  - can be removed or transferred to another device if needed
  - may not be present, depending on the device
  - read/writable by other apps and users; not private to your app
  - not wiped when the app is uninstalled, except in certain cases
- If your app needs to read/write the device's external storage, you must explicitly request **permission** to do so in your app's **AndroidManifest.xml** file.

```
<manifest ...>
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
...
</manifest>
```

- On install, the user will be prompted to confirm your app permissions.

- Methods to read / write external storage:
  - `getExternalFilesDir ("name")` – returns "private" external directory for your app with the given name
  - `Environment.getExternalStoragePublicDirectory(name)` – returns public directory for common files like photos, music,
- pass constants for name such as Environment.DIRECTORY_ALARMS, `DIRECTORY_DCIM`, `DIRECTORY_DOWNLOADS`, `DIRECTORY_MOVIES`, `DIRECTORY_MUSIC`, `DIRECTORY_NOTIFICATIONS`, `DIRECTORY_PICTURES`, `DIRECTORY_PODCASTS`, `DIRECTORY_RINGTONES`
- You can use these to read/write files on the external storage.
  - the above methods return standard `java.io.File` objects
  - these can be used with standard classes like `Scanner`, `BufferedReader`, and `PrintStream` to read/write files (see Java API)

- **Example:**

```
// write short data to app-specific external storage
File outDir = getExternalFilesDir(null); // root dir
File outFile = new File(outDir, "example.txt");
PrintStream output = new PrintStream(outFile);
output.println("Hello, world!");
output.close();
// read list of pictures in external storage
        File picsDir =
        Environment.getExternalStoragePublicDirectory(
        Environment.DIRECTORY_PICTURES);
        for (File file : picsDir.listFiles()) {
        ...
        }
```

Accessing web data:

- To read data from the web, first request the `INTERNET` permission in your **AndroidManifest.xml:**

```
<uses-permission
android:name="android.permission.INTERNET" />
```

- Then you can use the standard `java.net.URL` class to connect to a file or page at a given URL and read its data:

```
URL url = new URL("http://foobar.com/example.txt");
Scanner scan = new Scanner(url.openStream());
while (scan.hasNextLine()) {
String line = scan.nextLine();
...
}
```

3. The Camera

Learning outcomes:

Using the camera:

- If your app uses the device camera,
- you must explicitly state that it will use that hardware in your app's **AndroidManifest.xml** file.

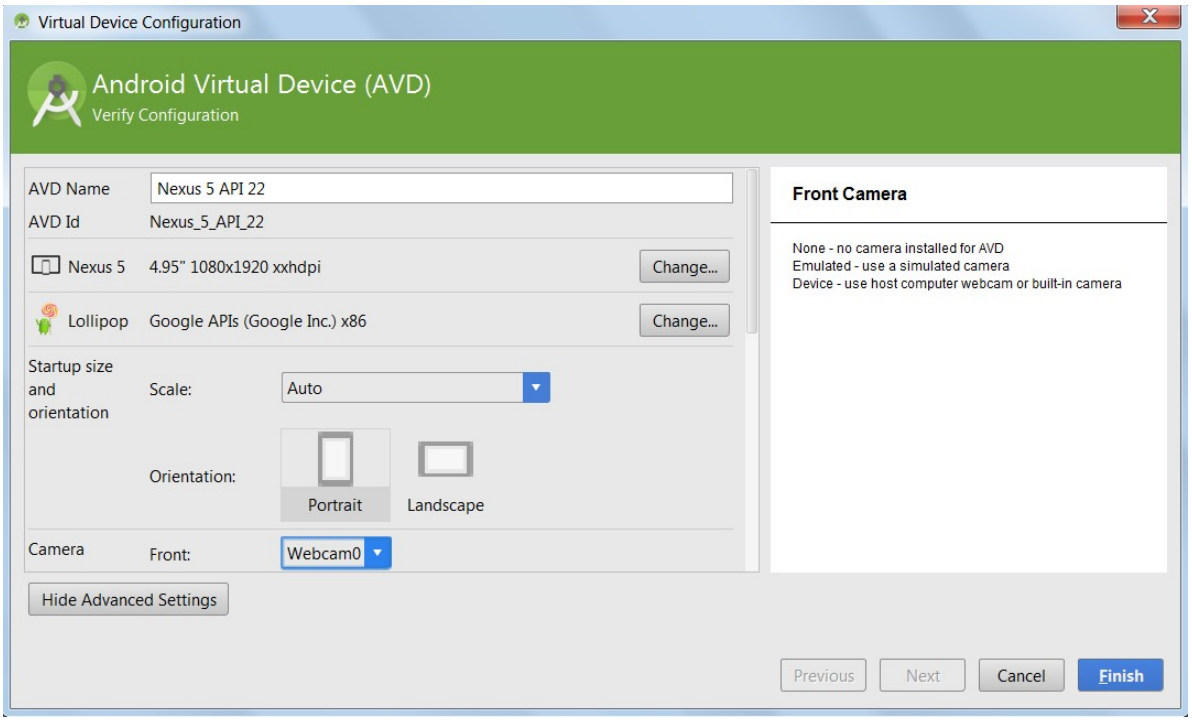
```
<manifest ...>
<uses-permission
android:name="android.permission.CAMERA" />
<uses-feature
android:name="android.hardware.camera"
android:required="true" />
...
</manifest>
```

AVD and camera:

- It is difficult to emulate a camera in the Android Virtual Device.
- In AVD Manager, you can try setting the device to use your laptop's webcam as the virtual camera, but it may crash.

  AVD Manager → Edit → Advanced Settings → Camera

Alternatively, just deploy your app to a physical device to test camera functionality.


Virtual Device Configuration
Android Virtual Device (AVD)
Verify Configuration
AVD Name
Nexus 5 API 22
AVD Id
Nexus_5_API_22
Nexus 5
4.95" 1080x1920 xxhdpi
Change...
Lollipop
Google APIs (Google Inc.) x86
Change...
Startup size and orientation
Scale:
Auto
Orientation:
Portrait
Landscape
Camera
Front:
Webcam0
Hide Advanced Settings
Front Camera
None - no camera installed for AVD
Emulated - use a simulated camera
Device - use host computer webcam or built-in camera
Previous
Next
Cancel
Finish

Example:

The following example takes a picture and places it into an ImageView:

```
public class MainActivity extends Activity {
    // these "request codes" are used to identify sub-
activities that return results
    private static final int REQUEST_CODE_DETAILS_ACTIVITY =
1234;
    private static final int REQUEST_CODE_TAKE_PHOTO = 4321;

    private static final int REQ_CODE_TAKE_PICTURE = 90210;
    ...
    Intent picIntent = new
Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    startActivityForResult(picIntent, REQ_CODE_TAKE_PICTURE);
    ...
 @Override
protected void onActivityResult
                          (int requestCode, int resultCode,
Intent intent) {
        if (requestCode == REQ_CODE_TAKE_PICTURE
                && resultCode == RESULT_OK) {
            Bitmap bmp = (Bitmap)
intent.getExtras().get("data");
            ImageView img = (ImageView)
findViewById(R.id.camera_image);
            img.setImageBitmap(bmp);
        }
    }
```

- In our previous example, we used the built–in photo–taking activity using an Intent.
  - This is simple but allows very little customization.
- For more precise camera control, look up the Camera class:

```
Camera cam = Camera.open();
Camera.Parameters params = cam.getParameters();
params.setPreviewSize(w, h);
cam.setParameters(params);
cam.setDisplayOrientation(degrees); // degrees
cam.startPreview();
cam.takePicture(...); // pass "callbacks" to run later
cam.stopPreview();
cam.release();
```

4. Example

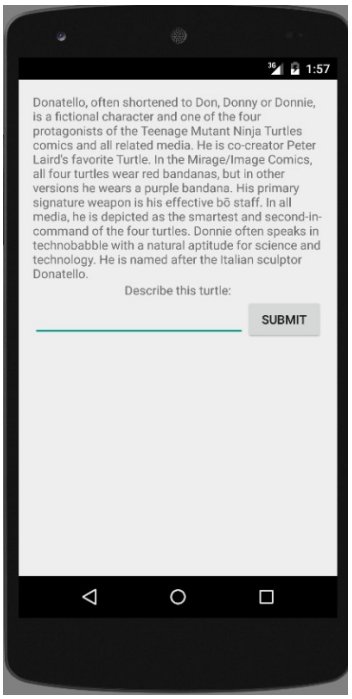
Learning outcomes:

Example

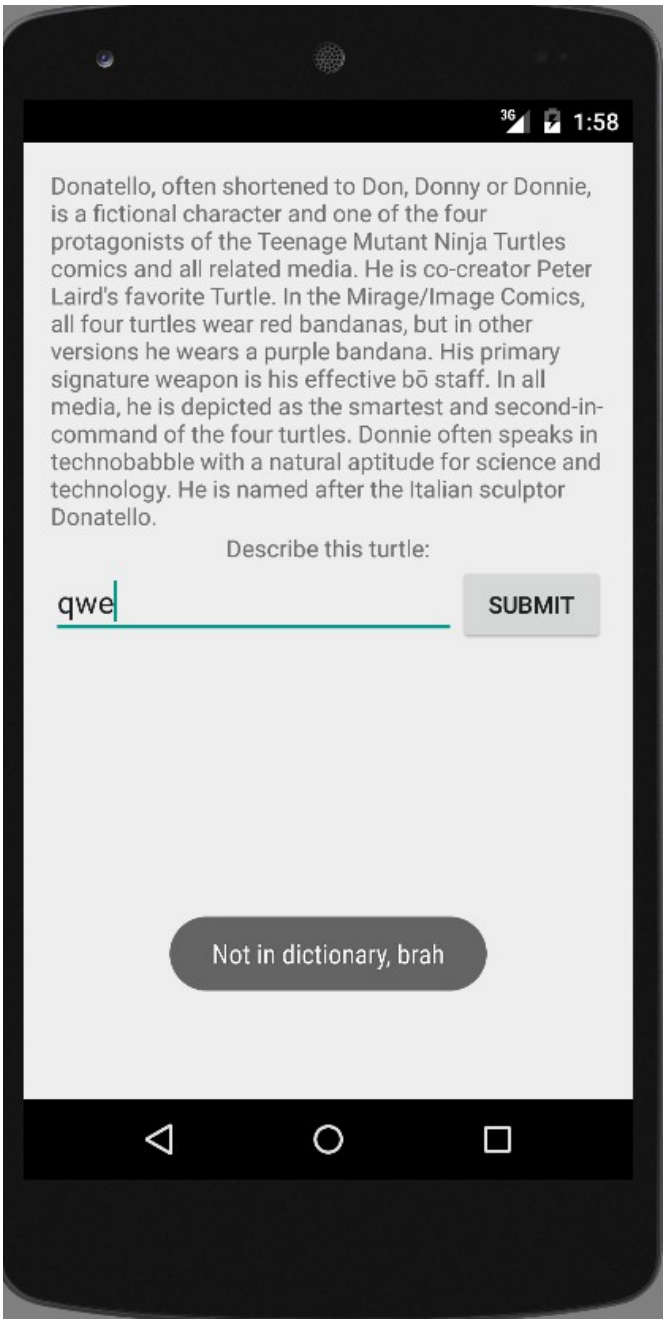
- When you run the application, the following interface appears:

- When you click on a radio button, the picture of the associated avatar appears:

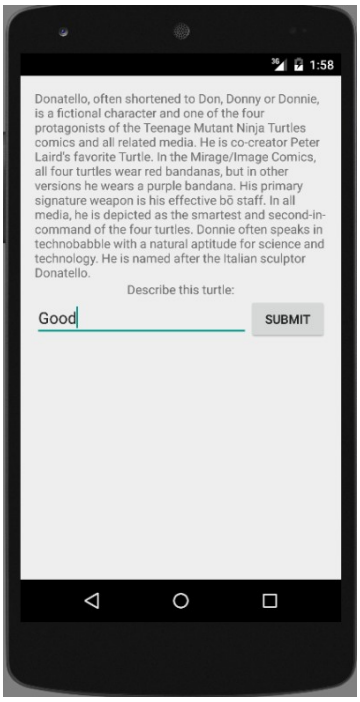
- When you click on an avatar, you move to the second interface where the avatar data is displayed and you are asked to write some comment.



- If you type a word that is not in the application dictionary, you will receive an error message:



- If you type a correct word that is in the application dictionary:

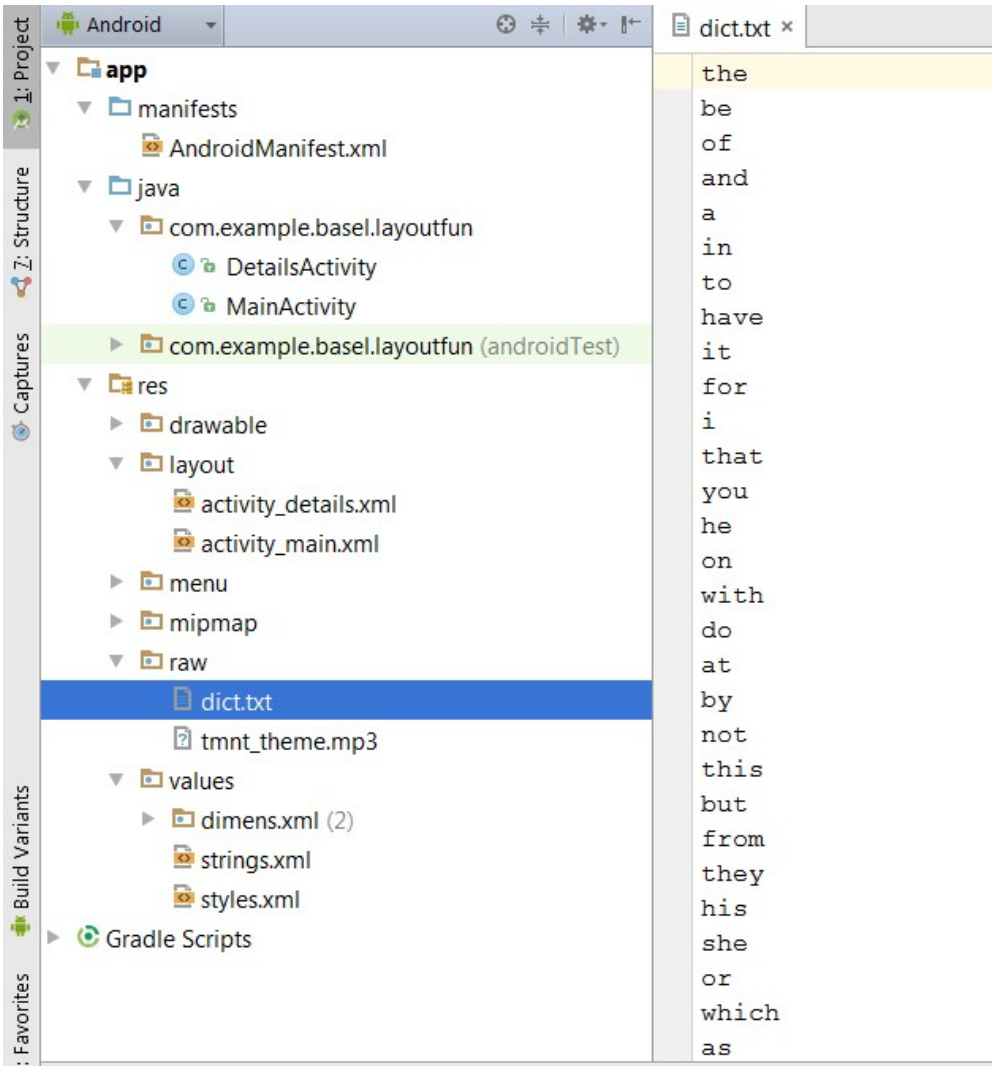


- You will return back to the first interface with your comment shown:

- You can also take a photo with the TAKE PHOTO button.

Building the application:

- The file raw\dict.txt contains some English word (the dictionary).



- The activity_main.xml design file is:

```
<LinearLayout
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:gravity="top|center"
    android:orientation="vertical"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"

    tools:context=".MainActivity">

<RadioGroup
        android:id="@+id/turtle_group"
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
<RadioButton
            android:id="@+id/leo"
            android:onClick="pickTurtle"
            android:text="Leo"
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

<RadioButton
            android:id="@+id/mike"
            android:onClick="pickTurtle"
            android:text="Mike"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

<RadioButton
            android:id="@+id/don"
            android:onClick="pickTurtle"
            android:text="Don"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

<RadioButton
            android:id="@+id/raph"
            android:onClick="pickTurtle"
            android:text="Raph"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content" />
</RadioGroup>

<ImageButton
        android:id="@+id/turtle"
        android:onClick="onClickTurtleImage"
        android:src="@drawable/tmntleo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

<Button
        android:text="Take Photo"
        android:onClick="onClickTakePhoto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

- The MainActivity.java code file:

```
package com.example.basel.layoutfun;
/*
 * onSaveInstanceState to make sure that the turtle is
 * not forgotten when the user leaves and returns to the
activity.
 */

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Bitmap;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.provider.MediaStore;
import android.util.Log;
import android.view.View;
import android.widget.ImageButton;
import android.widget.RadioGroup;
import android.widget.Toast;

public class MainActivity extends Activity {
    // these "request codes" are used to identify sub-
activities that return results
private static final int REQUEST_CODE_DETAILS_ACTIVITY = 1234;
private static final int REQUEST_CODE_TAKE_PHOTO = 4321;

private MediaPlayer player;    // media player for playing TMNT
music

/*
* Called when the activity first gets created.
```

```
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        player = MediaPlayer.create(this, R.raw.tmnt_theme);
        Log.d("testing", "onCreate got called; Bundle=" +
savedInstanceState);
    }

public void onPause() {
        super.onPause();
        player.stop();
        Log.d("testing", "onPause got called");
    }

public void onResume() {
        super.onResume();
        if (player != null) {
            player.setLooping(true);
            player.start();
        }
        Log.d("testing", "onResume got called");
    }

public void onStart() {
        super.onStart();
        Log.d("testing", "onStart got called");
    }

public void onStop() {
        super.onStop();
        Log.d("testing", "onStop got called");
    }

public void onDestroy() {
        super.onDestroy();
        Log.d("testing", "onDestroy got called");
    }

/*
* Called when the activity is stopped and wants to save its
state.
* We save which turtle is selected and showing so that the app
stays on that turtle
* when the user comes back later.
*/
@Override
protected void onSaveInstanceState(Bundle outState) {
super.onSaveInstanceState(outState);
Log.d("testing", "onSaveInstanceState got called");
```

```
RadioGroup group= (RadioGroup)findViewById(R.id.turtle_group);
int id = group.getCheckedRadioButtonId();
outState.putInt("id", id);
}

/*
* Called when the activity returns to an active running state
and wants to restore its state.
* We restore which turtle was previously selected.
*/
@Override
protected void onRestoreInstanceState

          (Bundle savedInstanceState) {
super.onRestoreInstanceState(savedInstanceState);
Log.d("testing", "onRestoreInstanceState got called");
int id = savedInstanceState.getInt("id");
RadioGroup group (RadioGroup)findViewById(R.id.turtle_group);
group.check(id);
updateTurtleImage();
    }

/*
* Called when the user clicks on the large TMNT image button.
* Loads the DetailsActivity for more information about that
turtle.
*/
public void onClickTurtleImage(View view) {
Intent intent = new Intent(this, DetailsActivity.class);
RadioGroup group=(RadioGroup) findViewById(R.id.turtle_group);
 int id = group.getCheckedRadioButtonId();
 intent.putExtra("turtle_id", id);
 startActivityForResult(intent,REQUEST_CODE_DETAILS_ACTIVITY);
}

/*
* Event handler that is called when a sub-activity returns.
* We can extract the data that came back from the other
activity
* (packed into the Intent that it sends back) and use it here.
*/
@Override
protected void onActivityResult(int requestCode, int
resultCode, Intent intent) {
super.onActivityResult(requestCode, resultCode, intent);
if (requestCode == REQUEST_CODE_DETAILS_ACTIVITY &&
                resultCode == RESULT_OK) {
// returned from DetailsActivity; user sent us a dictionary
word, so show it as a Toast
String word = intent.getStringExtra("the_word");
Toast.makeText(this, "You typed: " + word,
```

```
Toast.LENGTH_SHORT).show();
        }

else if (requestCode == REQUEST_CODE_TAKE_PHOTO
                && resultCode == RESULT_OK) {
// returned from taking a photo with the camera; grab the
photo and show it
Bitmap bmp = (Bitmap) intent.getExtras().get("data");
ImageButton img = (ImageButton)findViewById(R.id.turtle);
img.setImageBitmap(bmp);
        }
}

/*
* This method is called when the user chooses one of the
turtle radio buttons.
* In this code we set which turtle image is visible on the
screen in the ImageView.
*/
public void pickTurtle(View view) {
        updateTurtleImage();
    }

/*
* Called when the "Take Photo" button is clicked.
* Launches a new activity to take a photo using the camera.
*/
public void onClickTakePhoto(View view) {
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, REQUEST_CODE_TAKE_PHOTO);
}

/*
* Called by various event handlers to update which turtle
image is showing
* based on which radio button is currently checked.
*/
private void updateTurtleImage() {
ImageButton img = (ImageButton) findViewById(R.id.turtle);
RadioGroup group=(RadioGroup)findViewById(R.id.turtle_group);
        int checkedID = group.getCheckedRadioButtonId();
        if (checkedID == R.id.leo) {
            img.setImageResource(R.drawable.tmntleo);
        } else if (checkedID == R.id.mike) {
            img.setImageResource(R.drawable.tmntmike);
        } else if (checkedID == R.id.don) {
            img.setImageResource(R.drawable.tmntdon);
        } else if (checkedID == R.id.raph) {
            img.setImageResource(R.drawable.tmntraph);
        }
    }
```

```
}
```

- The activity_details.xml design file:

```
<LinearLayout
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:gravity="top|center"
    android:orientation="vertical"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"

tools:context="com.example.basel.layoutfun.DetailsActivity">

<TextView
        android:id="@+id/turtle_info"
        android:text=""
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

<TextView
        android:text="Describe this turtle:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

<LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <EditText
            android:id="@+id/the_word"
            android:layout_weight="1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

<Button
            android:text="Submit"
            android:onClick="onclickSubmit"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
</LinearLayout>
</LinearLayout>
```

The DetailsActivity.java code file:

```
package com.example.basel.layoutfun;
/*
 * This file represents the Java code for the second activity.
 * This version adds a dictionary for looking up words.
 */

import android.app.Activity;
import android.content.*;
import android.os.*;
import android.support.v7.app.*;
import android.view.*;
import android.widget.*;
import java.util.*;

public class DetailsActivity extends Activity {
private Set<String> dictionary;// dictionary of words to look
up

/*
* Constant array of data about each of the four turtles.
* (This is not the most idiomatic way to store such
information,
*/
private static final String[] TURTLE_DETAILS = {
            "Leonardo, or Leo, is one of the four protagonists
of the Teenage Mutant Ninja Turtles comics and all related
media. In the Mirage/Image Comics, all four turtles wear red
bandanas, but in other versions, he wears a blue bandana. His
signature weapons are two ninjato. Throughout the various
media, he is often depicted as the eldest and leader of the
four turtles, as well as the most disciplined. He is named
after Leonardo da Vinci. In the 2012 series, he is the only
turtle who harbors strong romantic affections for Karai,
considering her his love interest.",
            "Michelangelo, Mike or Mikey (as he is usually
called), is a fictional character and one of the four
protagonists of the Teenage Mutant Ninja Turtles comics and
all related media. His mask is typically portrayed as orange
outside of the Mirage/Image Comics and his weapons are dual
nunchucks, though he has also been portrayed using other
weapons, such as a grappling hook, manriki-gusari, tonfa, and
a three section staff (in some action figures).",
            "Donatello, often shortened to Don, Donny or
Donnie, is a fictional character and one of the four
protagonists of the Teenage Mutant Ninja Turtles comics and
all related media. He is co-creator Peter Laird's favorite
Turtle. In the Mirage/Image Comics, all four turtles wear red
```

```
bandanas, but in other versions he wears a purple bandana. His
primary signature weapon is his effective bō staff. In all
media, he is depicted as the smartest and second-in-command of
the four turtles. Donnie often speaks in technobabble with a
natural aptitude for science and technology. He is named after
the Italian sculptor Donatello.",
            "Raphael, or Raph, is a fictional character and
one of the four protagonists of the Teenage Mutant Ninja
Turtles comics and all related media. In the Mirage/Image
Comics, all four turtles wear red bandanas over their eyes,
but unlike his brothers in other versions, he is the only one
who keeps the red bandana. Raphael wields twin sai as his
primary weapon. (In the Next Mutation series, his sai stick
together to make a staff-like weapon.) He is generally the
most likely to experience extremes of emotion, and is usually
depicted as being aggressive, sullen, maddened, and
rebellious. The origin of Raphael's anger is not always fully
explored, but in some incarnations appears to stem partly from
the realization that they are the only creatures of their kind
and ultimately alone. He also has a somewhat turbulent
relationship with his older brother Leonardo because Leonardo
is seen as the group's leader. Raphael is named after the
16th-century Italian painter Raphael. In 2011 Raphael placed
23rd on IGN's Top 100 Comic Book Heroes, a list that did not
feature any of his brothers."
    };

/*
* Called when the activity first gets created.
* Shows detail text about the selected ninja turtle.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details);

// pull the turtle's ID out of the intent that the
MainActivity used to load me
        Intent intent = getIntent();
        int id = intent.getIntExtra("turtle_id", R.id.leo);
        String text = "";
        if (id == R.id.leo) {
            text = TURTLE_DETAILS[0];
        } else if (id == R.id.mike) {
            text = TURTLE_DETAILS[1];
        } else if (id == R.id.don) {
            text = TURTLE_DETAILS[2];
        } else { // if (id == R.id.raph)
            text = TURTLE_DETAILS[3];
        }
    TextView tv = (TextView) findViewById(R.id.turtle_info);
```

```
        tv.setText(text);

        if (dictionary == null) {
            loadDictionary();
        }
    }

/*
* Called when the user presses the Submit button.
* Checks whether the user has typed a legal dictionary word in
the text box,
* and if so, closes this activity and returns to the
MainActivity.
*/
public void onclickSubmit(View view) {
// if user has typed a valid dictionary word, accept it and
send it back to main activity
 EditText edit = (EditText) findViewById(R.id.the_word);
 String text = edit.getText().toString().trim().toLowerCase();
        if (dictionary.contains(text)) {
            Intent intent = new Intent();
            intent.putExtra("the_word", text);
            setResult(RESULT_OK, intent);
            finish();
        } else {
            Toast.makeText(this, "Not in dictionary, brah",
Toast.LENGTH_SHORT).show();
        }
    }

/*
* Loads the dictionary words from the provided text file,
dict.txt.
*/
private void loadDictionary() {
dictionary = new HashSet<String>();
Scanner scan = new
Scanner(getResources().openRawResource(R.raw.dict));
        while (scan.hasNextLine()) {
            String word = scan.nextLine();
            dictionary.add(word);
        }
    }

// AUTO-GENERATED CODE

@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;
    }

@Override
public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar
will
        // automatically handle clicks on the Home/Up button,
so long
        // as you specify a parent activity in
AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

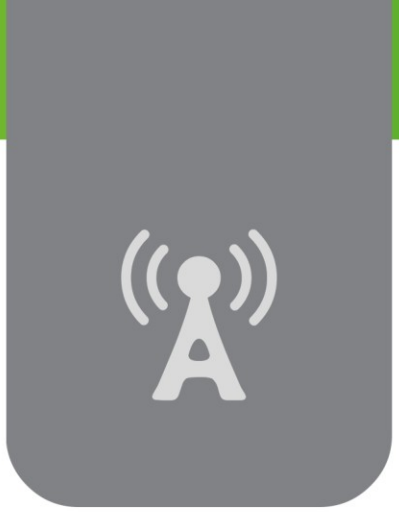
Exercise 1: Preferences

- Use the preferences to build the following application:

- Save: to save data.
- Get: to retrieve saved data.
- Clear: to clear texts.

You can use the following link for more help:

http://www.compiletimeerror.com/2015/02/android-shared-preferences-example-and-tutorial.html#.VeL68vnoSGQ



Chapter 10:
Lists

Key Words:

Lists, Text to Speech, Speech to Text.

Summary:

This unit shows how to build lists, and how to use text to speech and speech to text functions.

Outcomes:

Student will learn in this unit:

- Building lists.
- Text to speech and speech to text tools.

Plan:

3 Learning Objects

1. Lists
2. Text to Speech and Speech to Text
3. Example

1. Lists

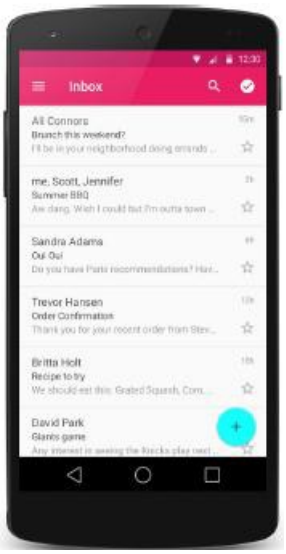
Learning outcomes:

Using lists.

ListView:

An ordered collection of selectable choices

Main properties:



`android:clickable="bool"`	set to false to disable the list
`android:id="@+id/theID"`	unique ID for use in Java code
`android:entries="@array/array"`	set of options to appear in the list (must match an array in *strings.xml*)

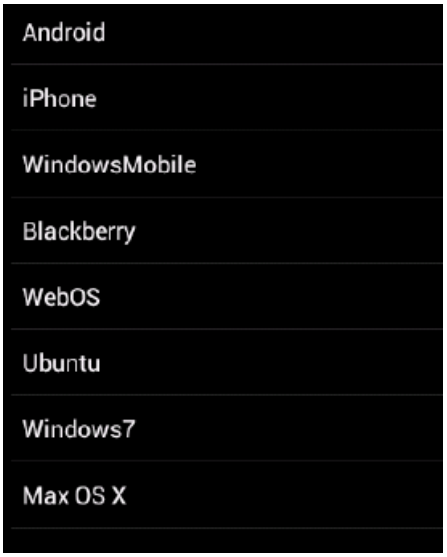
Static Lists:

- **static list**: Content is fixed and known before the app runs.
  - Declare the list elements in the **strings.xml** resource file.

```
<!-- res/values/strings.xml -->
<resources>
<string-array name="oses">
<item>Android</item>
<item>iPhone</item>
        ...
<item>Max OS X</item>
</string-array>
</resources>
```

Used in the activity:

```
<!-- res/layout/activity_main.xml -->
<ListView ... android:id="@+id/mylist"
        android:entries="@array/oses" />
```



Dynamic Lists:

- **Dynamic list**: Content is read or generated as the program runs.
  - Comes from a data file, or from the internet, etc.
  - Must be set in the Java code.
  - Suppose we have the following file and want to make a list from it:

```
// res/raw/oses.txt
Android
iPhone
...
Max OS X
```

List Adapter:

- **Adapter**: Helps turn list data into list view items.
  - common adapters: `ArrayAdapter, CursorAdapter`
- Syntax for creating an adapter:

```
ArrayAdapter<String> name =
        new ArrayAdapter<String>(activity, layout, array);
```

- the **activity** is usually this
- the default **layout** for lists is

  android.R.layout.simple_list_item_1
- get the **array** by reading your file or data source of choice

  (it can be an array like `String[]`, or a list like `ArrayList<String>`)
- Once you have an adapter, you can attach it to your list by calling the `setAdapter` method of the `ListView` object in the Java code:

```
ArrayList<String> myArray = ...; // load data from file
ArrayAdapter<String> adapter =
        new ArrayAdapter<String>(
        this,
        android.R.layout.simple_list_item_1,
        myArray);
ListView list = (ListView) findViewById(R.id.mylist);
list.setAdapter(myAdapter);
```

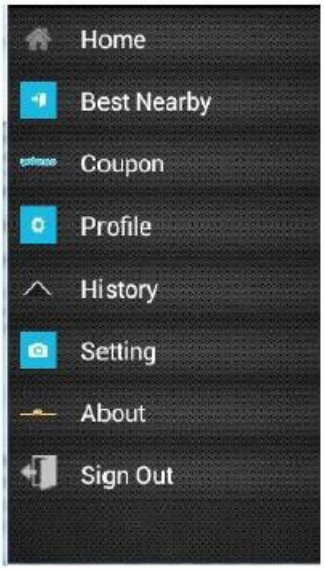
List events:

- **setOnItemClickListener** (AdapterView.OnItemClickListener)
  //Listener for when an item in the list has been clicked.
- **setOnItemLongClickListener** (AdapterView.OnItemLongClickListener)
  //Listener for when an item in the list has been clicked and held.
- **setOnItemSelectedListener** (AdapterView.OnItemSelectedListener)
  //Listener for when an item in the list has been selected.
- onDrag
- onFocusChanged
- onHover
- onKey
- onScroll
- onTouch

Custom Formatting:

- If you want your list to look different than the default appearance (of just a text string for each line), you must:
  - Write a short **layout XML file** describing the layout for each row.
  - Write **a subclass of `ArrayAdapter`** that overrides the `getView` method to describe what view must be returned for each row.

- Example:



The activity file:

```
<!-- res/layout/mylistlayout.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ... android:orientation="horizontal">
<ImageView ... android:id="@+id/list_row_image"
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:src="@drawable/smiley" />
<TextView ... android:id="@+id/list_row_text"
    android:textStyle="bold"
    android:textSize="22dp"
    android:text=""
    android:background="#336699" />
</LinearLayout>
```

The code file:

```
// MyAdapter.java
public class MyAdapter extends ArrayAdapter<String> {
private int layoutResourceId;
private List<String> data;

public MyAdapter
   (Context context, int layoutId, List<String> list) {
        super(context, layoutResourceId, data);
        layoutResourceId = layoutId;
        data = list;
}
@Override
public View getView(int index, View row, ViewGroup parent) {
row = getLayoutInflater().inflate
            (layoutResourceId, parent, false);
TextView text = (TextView)
row.findViewById(R.id.list_row_text);
text.setText(data.get(index));
return row;
    }
}
```

2. Text to Speech and Speech to Text

Learning outcomes:

Text to Speech and Speech to Text tools

Text to Speech:

- text–to–speech: Allows Android device to speak an audible message based on a text string.
  - Not installed by default on some devices. To install, click: Settings → Language and Input → Text to speech output → Google text–to–speech engine "settings" icon → Install voice data → Languages
- In general, text–to–speech on Android is simple:
  - create a TextToSpeech object
  - call its speak method
- But there are a few details that require us to discuss some advanced features of Java

TextToSpeech class:

- `new TextToSpeech(activity, listener)` - constructor
- `getVoice, getVoices, setVoice` – change speaking voice
- `getLanguage, setLanguage` – sets language used
- `getPitch, setPitch` – sets vocal tone used
- `isSpeaking` – returns true if speaking
- `shutdown` - kills TTS engine
- `speak(text, mode, params)` – speaks given text aloud
- `stop` - halts any speech
- `synthesizeToFile (text, params, filename)` – speaks to file

Initialization and listener:

- The `TextToSpeech` service can take a while to initialize.
  - So its constructor forces you to pass a listener object.
  - The listener will be notified when the TTS service is done loading.
  - This helps keep the main UI from freezing up during TTS load time.

(The code below uses a Java **anonymous inner class**.)

```
TextToSpeech tts = new TextToSpeech(this,
        new TextToSpeech.OnInitListener() {
@Override
public void onInit(int status) {
// code to run when done loading
        }
});
```

Waiting for initialization:

- You must wait until the text-to-speech listener's onInit method has been called before trying to speak any text.
  - Otherwise the app will crash with an exception.
- Typical usage pattern:
  - create a boolean flag in your activity
  - have your listener set it to true when the initialization is complete
  - only call speak on the TTS object if the flag is set to true

speak:

- The speak method accepts three parameters:
  - the text to speak aloud, as a String.
  - the mode to use for speaking, one of:
  - `TextToSpeech.QUEUE_ADD`: Speak after any other text is done.
  - `TextToSpeech.QUEUE_FLUSH`: Stop any other text and speak immediately.
  - a Map of parameters (we don't need any, so we can pass null).

```
// speak text aloud, if my init boolean flag is set
if (myTTSisReady) {
        tts.speak("Hello, world!",
                TextToSpeech.QUEUE_FLUSH, null);
        }
```

Speech to Text:

- **`speech-to-text`**: User talks; Android records, turns into a String.
- Similar to the camera, Android has a built-in activity for capturing speech into text.
- You can call it using an Intent and wait for the result.

```
Intent intent = new Intent(
        RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
        Locale.getDefault());
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
        RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
// prompt text is shown on screen to tell user what to say
intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "text");
startActivityForResult(intent, requestCode);
```

Receiving speech-to-text data:

- When the speech-to-text activity comes back, its Intent gives you all text spoken by the user in an `ArrayList`.
  - Usually the first element (index 0) contains the string you want.

```
@Override
protected void onActivityResult(int requestCode,
        int resultCode, Intent intent) {
super.onActivityResult(requestCode, resultCode, intent);
ArrayList<String> list = intent.getStringArrayListExtra(
        RecognizerIntent.EXTRA_RESULTS);
String spokenText = list.get(0);
// ...
}
```

- Some devices do not have speech-to-text capability.
  - In these cases, it will throw an exception when you try to use it.
  - To handle such situations, you can **try** / **catch** the exception.

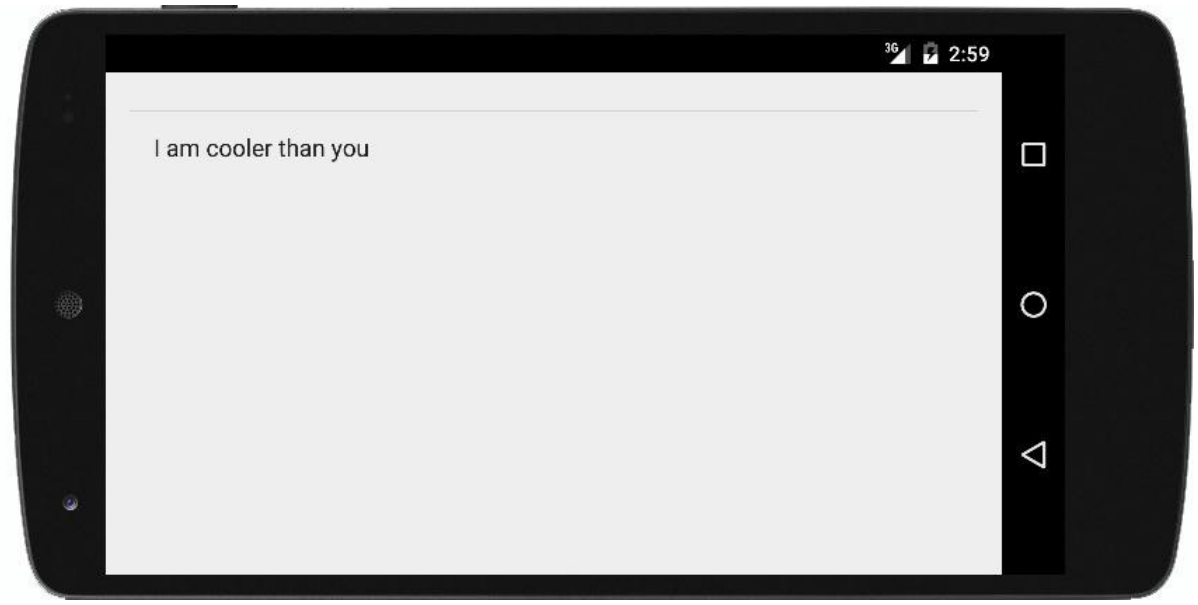
```
Intent intent = new Intent(
        RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        ...
        try {
        startActivityForResult(intent, requestCode);
        } catch (ActivityNotFoundException anfe) {
         // code to handle the exception
        }
```

3. Example

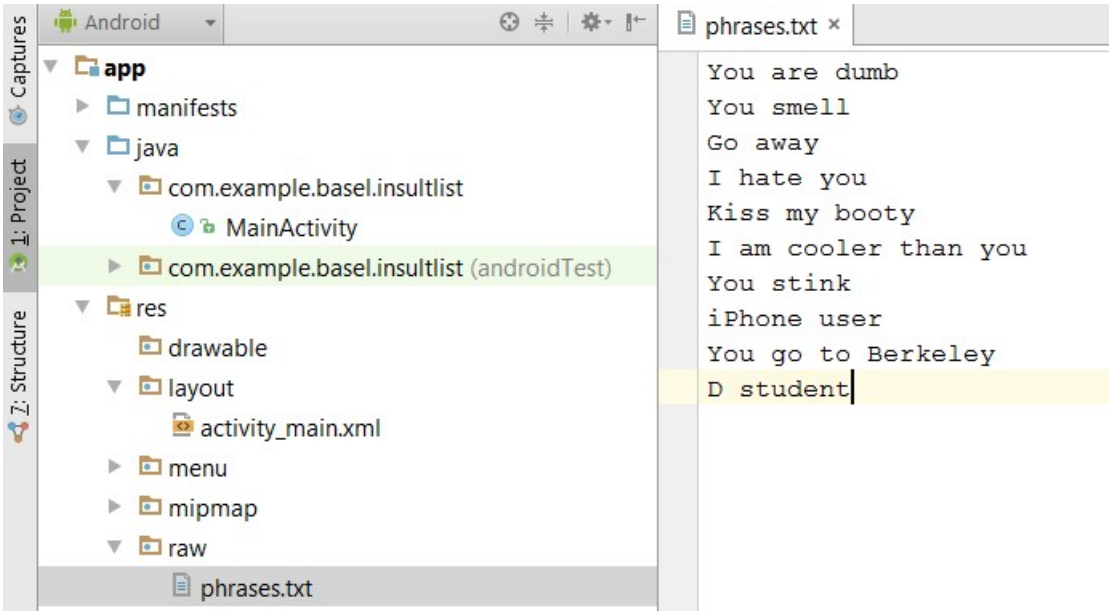
Learning outcomes:

Example

When you run the app, a sliding list of reprimand phrases appears. When you click on a phrase, the device will speak it.



- he phrases are saved in the raw/phrases.txt file:



The activity_main.xml file contains a `ListView` with a `ScrollView`:

```
<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">
<ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <ListView
            android:id="@+id/list_of_insults"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:theme="@android:style/Animation">
        </ListView>
</ScrollView>
</RelativeLayout>
```

The MainActivity.java code file is:

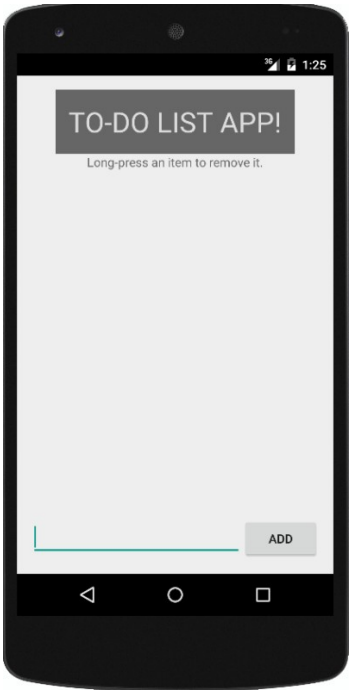
```
package com.example.basel.insultlist;
/*
 * This file implements the main activity for the insult list
app.
 * It shows a list of insult phrases and speaks them aloud
when
 * the user clicks on each one.
 */
import android.app.Activity;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import java.util.ArrayList;
import java.util.Scanner;

public class MainActivity extends Activity {
private ArrayList<String> lines;       // lines of file of
insults
private TextToSpeech tts;              // TTS engine
```

```
private boolean speechReady = false;// true when TTS engine is
loaded

/*
* Initializes the state of the activity.
*/
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

// read input file
lines = readEntireFile(R.raw.phrases);


// set up the ListView to use the lines from the file
ListView myList = (ListView)
                findViewById(R.id.list_of_insults);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
         this, android.R.layout.simple_list_item_1, lines);
myList.setAdapter(adapter);


// set up event listening for clicks on the list
myList.setOnItemClickListener(new
      AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent,
                            View view, int index, long id) {
                handleClick(index);
            }
        });

// set up text-to-speech engine
tts = new TextToSpeech(this, new TextToSpeech.OnInitListener()
{
            @Override
            public void onInit(int status) {
                speechReady = true;
            }
        });
    }

/*
* Handles a click on the list item at the given 0-based index.
* Speaks the given insult aloud using text-to-speech.
*/
private void handleClick(int index) {
        String text = lines.get(index);
        if (speechReady) {
            tts.speak(text, TextToSpeech.QUEUE_FLUSH, null);
```

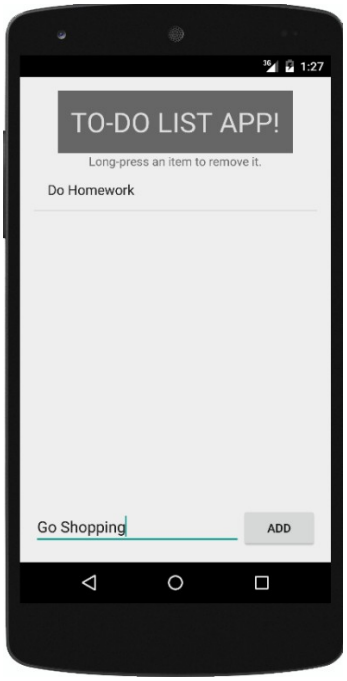
```
        }
}

/*
* Reads the lines of the file with the given resource ID,
* returning them as an array list of strings.
* Assumes that the file with the given ID exists in res/raw
folder.
*/
private ArrayList<String> readEntireFile(int id) {
        ArrayList<String> list = new ArrayList<String>();
        Scanner scan = new
                  Scanner(getResources().openRawResource(id));
        while (scan.hasNextLine()) {
            String line = scan.nextLine();
            list.add(line);
        }
        return list;
    }
}
```

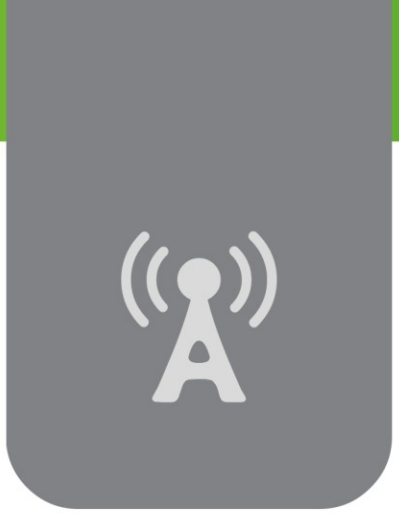
Exercise:

You are requested to build the (to-do list) application where the main interface has initially an empty ListView:



The user can enter an item in the list by typing the item in the Edit text and then clicking the ADD button. To delete an item, he has to click it for a while.





Chapter 11:
Fragments

Key Words:

Fragments, Situation-Specific Folders, Portrait, Landscape, Fragment lifecycle, Fragment template.

Summary:

This unit show how to use fragments as reusable blocks.

Outcomes:

Student will learn in this unit:

- Creating fragments.
- Fragment lifecycle.
- Fragment template.
- Fragment parameters.

Plan:

2 Learning Objects

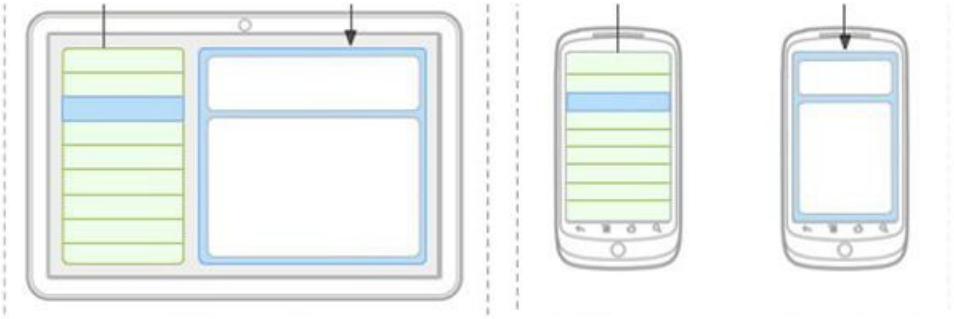
1. Fragments
2. Example

1. Fragments

Learning outcomes:

Using fragments:

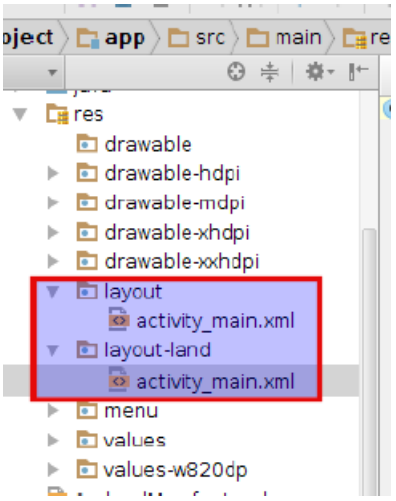
- Your app can use different layout in different situations:
  - different device type (tablet vs phone vs watch)
  - different screen size
  - different orientation (portrait vs. landscape)
  - different country or locale (language, etc.)



Situation-Specific folders:

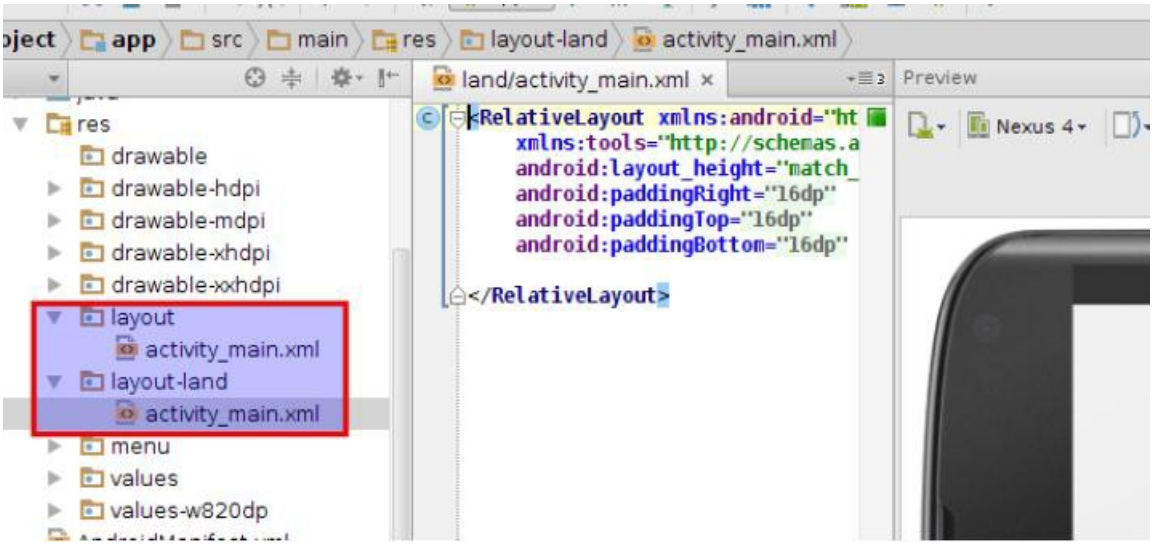
Your app will look for resource folder names with suffixes:

- screen density (e.g. **drawable-hdpi**)
  - xhdpi: 2.0 (twice as many pixels/dots per inch)
  - hdpi: 1.5
  - mdpi: 1.0 (baseline)
  - ldpi: 0.75
- screen size (e.g. layout-large)
  - small, normal, large, xlarge
- orientation (e.g. layout-land)
  - portrait (), land (landscape)



Portrait / Landscape:

- To create a different layout in landscape mode:
  - create a folder in your project called res/layout–land
  - place another copy of your activity's layout XML file there
  - modify it as needed to represent the differences

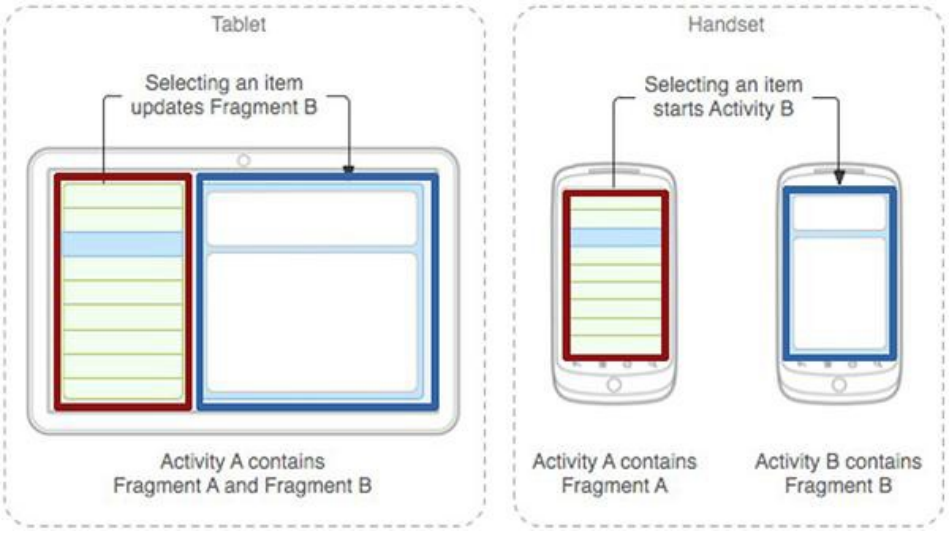


Redundency:

- With situational layout you begin to encounter **redundancy**.
  - The layout in one case (e.g. portrait or medium) is very similar to the layout in another case (e.g. landscape or large).
  - You don't want to represent the same XML or Java code multiple times in multiple places.
- You sometimes want your code to behave **situationally**.
  - In portrait mode, clicking a button should launch a new **activity**.
  - In landscape mode, clicking a button should launch a new **view**.

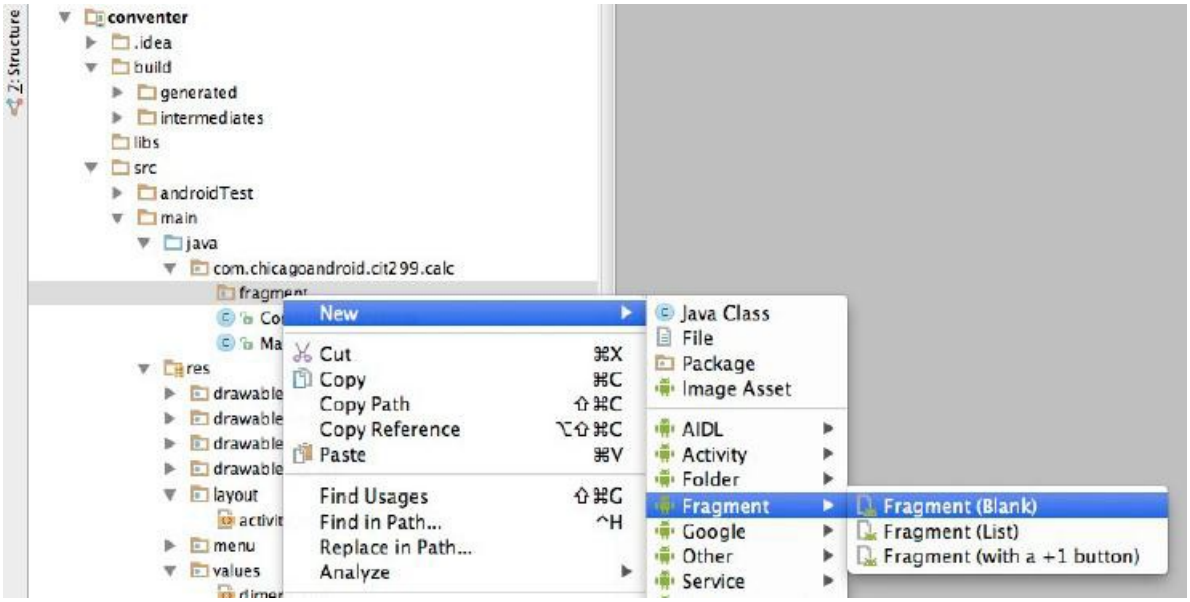
Fragments:

- **fragment:** A reusable segment of Android UI that can appear in an activity.
  - can help handle different devices and screen sizes
  - can reuse a common fragment across multiple activities
- The following shows an example where the activity A on a tablet contains two fragments A and B. Selecting an item from the fragment A updates the fragment B.
- The same application on a handset consists of two activities A and B. The activity A contains the fragment A, and the activity B contains the fragment B. Selecting an item from the fragment A opens the activity B.



- To create a fragment:

  New → Fragment → Fragment (blank)



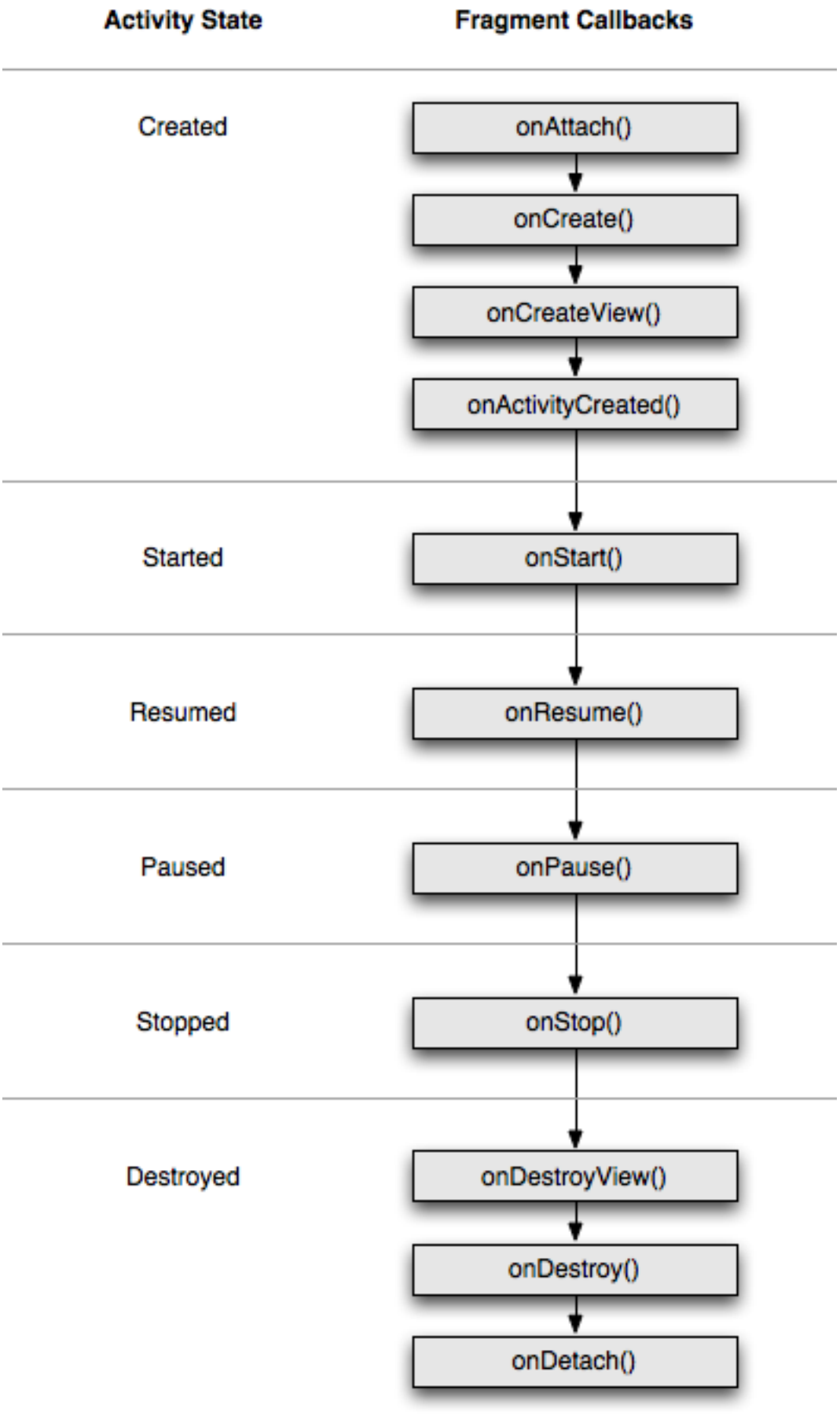
- An activity can have more than one fragment:

```
<!-- activity_name.xml -->
<LinearLayout ...>
<fragment ...
android:id="@+id/id1"
android:name="ClassName1"
tools:layout="@layout/name1" />
<fragment ...
android:id="@+id/id2"
android:name="ClassName2"
tools:layout="@layout/name2" />
</LinearLayout>
```

Fragment lifecycle:

- Fragments have a similar life cycle and events as activities.

`onAttach`	to glue fragment to its surrounding activity
`onCreate`	when fragment is loading
`onCreateView`	method that must return fragment's root UI view
`onActivityCreated`	method that indicates the enclosing activity is ready
`onPause`	when fragment is being left/exited
`onDetach`	just as fragment is being deleted


Activity State
Fragment Callbacks
Created
onAttach()
onCreate()
onCreateView()
onActivityCreated()
Started
onStart()
Resumed
onResume()
Paused
onPause()
Stopped
onStop()
Destroyed
onDestroyView()
onDestroy()
onDetach()

Fragment Template:

The fragment has the following template:

```
public class Name extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater,
                              ViewGroup vg, Bundle bundle) {
        // load the GUI layout from the XML
        return inflater.inflate(R.layout.id, vg, false);
    }
 public void onActivityCreated(Bundle savedState) {
        super.onActivityCreated(savedState);
// ... any other GUI initialization needed
    }
// any other code (e.g. event-handling)
}
```

Fragment vs. activity:

Fragment code is similar to activity code, with a few changes:

- Many activity methods aren't present in the fragment, but you can call `getActivity` to access the activity the fragment is inside of.

```
Button b = (Button) findViewById(R.id.but);
Button b = (Button) getActivity().findViewById (R.id)
```

- Sometimes also use `getView` to refer to the activity's layout
- Event handlers cannot be attached in the XML any more.
  - Must be attached in Java code instead.
- Passing information to a fragment (via Intents/Bundles) is trickier.
  - The fragment must ask its enclosing activity for the information.
- Fragment initialization code must be mindful of order of execution.
  - Does it depend on the surrounding activity being loaded? Etc.
  - Typically move `onCreate` code to `onActivityCreated`.

Example: Declaring the click event:

For example, the activity has:

```
<Button android:id="@+id/b1"
android:onClick="onClickB1" ... />
```

and in the fragment:

```
Button android:id="@+id/b1" ... />
```

We can write the following code:

```
// in fragment's Java file
Button b = (Button) getActivity().findViewById(r.id.b1);
b.setOnClickListener(new View.OnClickListener() {
@Override public void onClick(View view) {
// whatever code would have been in onClickB1
            }
});
```

Passing parameters:

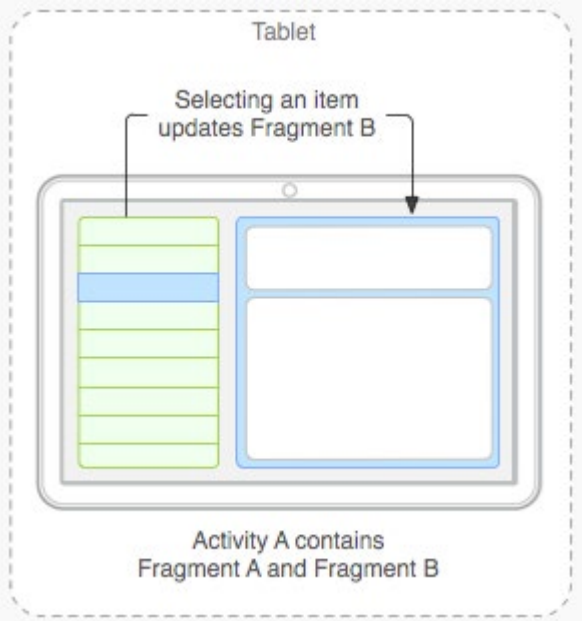
In the activity:

```
public class MainActivity extends Activity {
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

public class Name extends Activity {
@Override
protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.name);
               // extract parameters passed to activity from
intent
                Intent intent = getIntent();
                int name1 = intent.getIntExtra("id1",
default);
                String name2 = intent.getStringExtra("id2",
"default");
              // use parameters to set up the initial state
                ...
            }
            ...
}
```

In the fragment:

```
public class Name extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater,
                            ViewGroup container, Bundle
savedInstanceState) {
   return inflater.inflate(R.layout.name, container, false);
    }
@Override
public void onActivityCreated(Bundle savedState) {
        super.onActivityCreated(savedState);
// extract parameters passed to activity from intent
        Intent intent = getActivity().getIntent();
        int name1 = intent.getIntExtra("id1", default);
        String name2 = intent.getStringExtra("id2",
"default");
// use parameters to set up the initial state
        ...
    }
```

Communication between fragments:

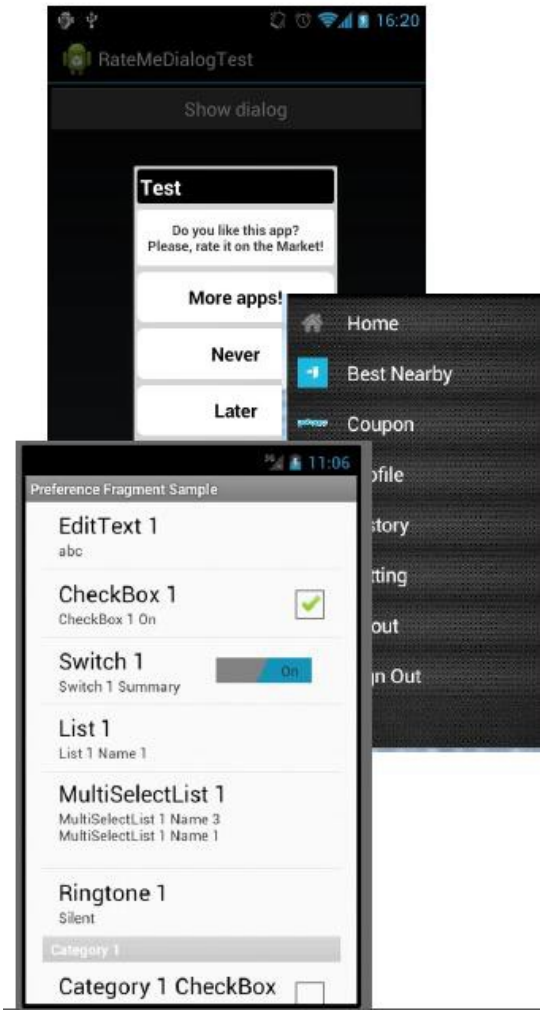


- One activity might contain multiple fragments.
- The fragments may want to talk to each other.
  - Use activity's getFragmentManager method.
  - its findFragmentById method can access any fragment that has an id.
- Example:

```
Activity act = getActivity();
        if (act.getResources().getConfiguration().orientation ==
        Configuration.ORIENTATION_LANDSCAPE) {
        // update other fragment within this same activity
        FragmentClass fragment = (FragmentClass)
        act.getFragmentManager().findFragmentById(R.id.id);
        fragment.methodName(parameters);
        }
```

Fragment subclasses:

- `DialogFragment` – a fragment meant to be shown as a dialog box that pops up on top of the current activity.
- `ListFragment` – a fragment that shows a list of items as its main content.
- `PreferenceFragment` – a fragment whose main content is meant to allow the user to change settings for the app.



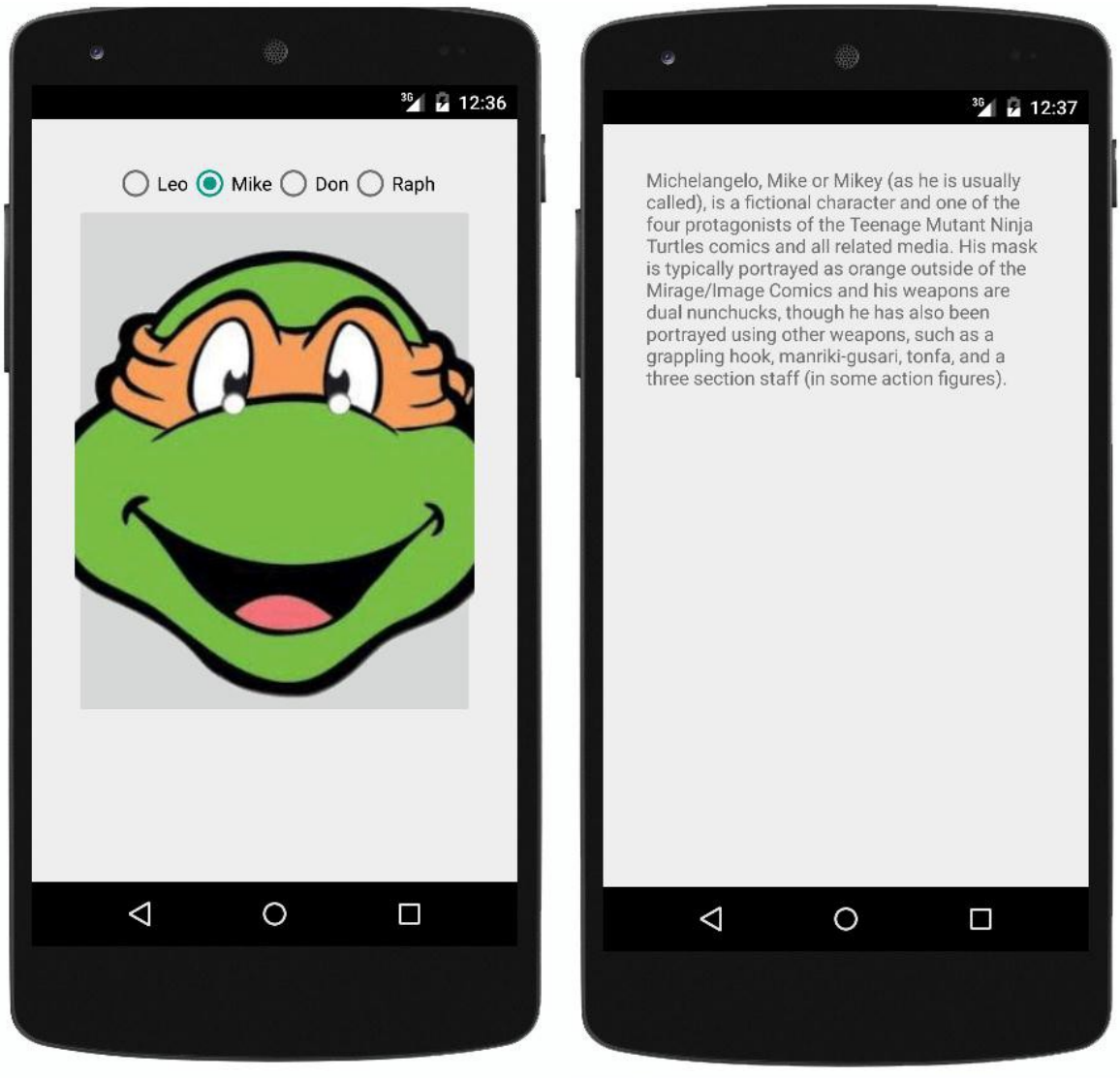
2. Example

Learning outcomes:

Example:

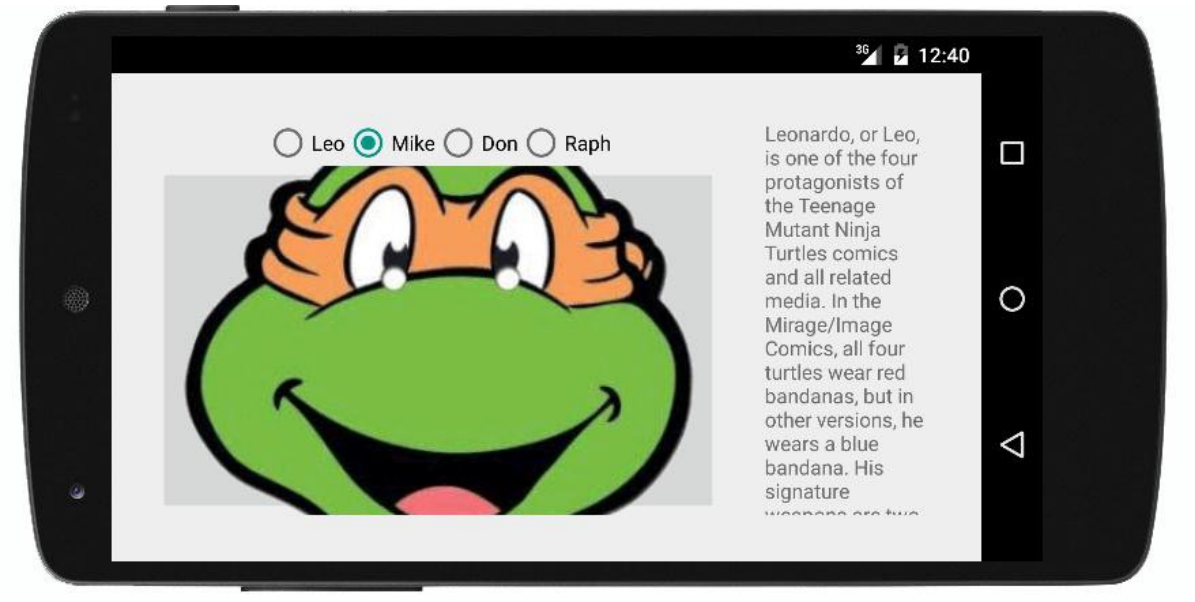
In the following example, we change the behavior of the application according to the orientation of the device, portrait, or landscape.

When you click on an image in the portrait mode, a new activity is opened to show the clicked character profile.



In the landscape direction the profile is displayed in the same activity:

(to reverse the emulator direction, press CTRL + F11),

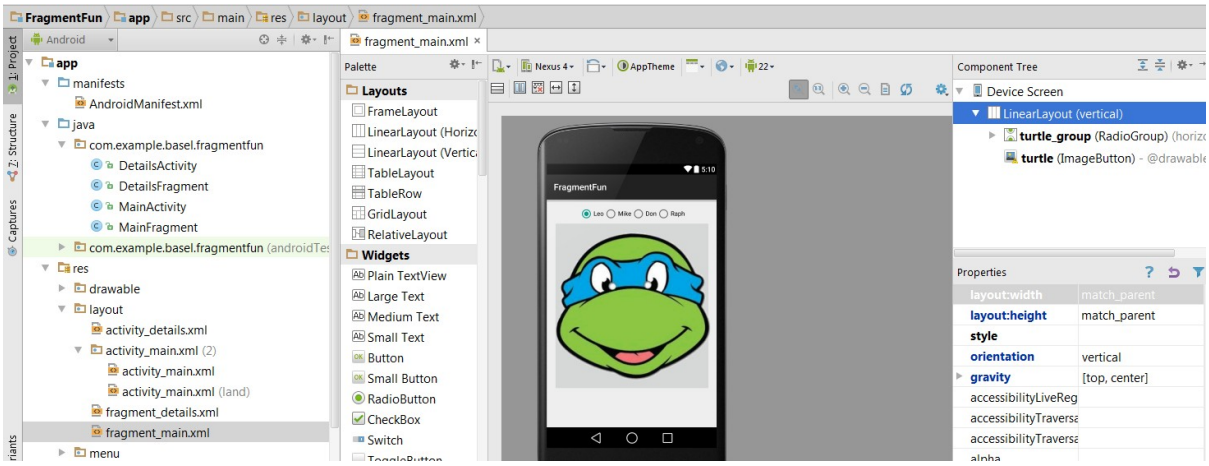


Building the application steps:

1. Create new application (has MainActivity.java and activity_main.xml).
2. Add new fragment:

New→Fragment→Fragment (Blank)

Name the fragment fragment_main.xml, and make the following design that contains four radio buttons and an image button

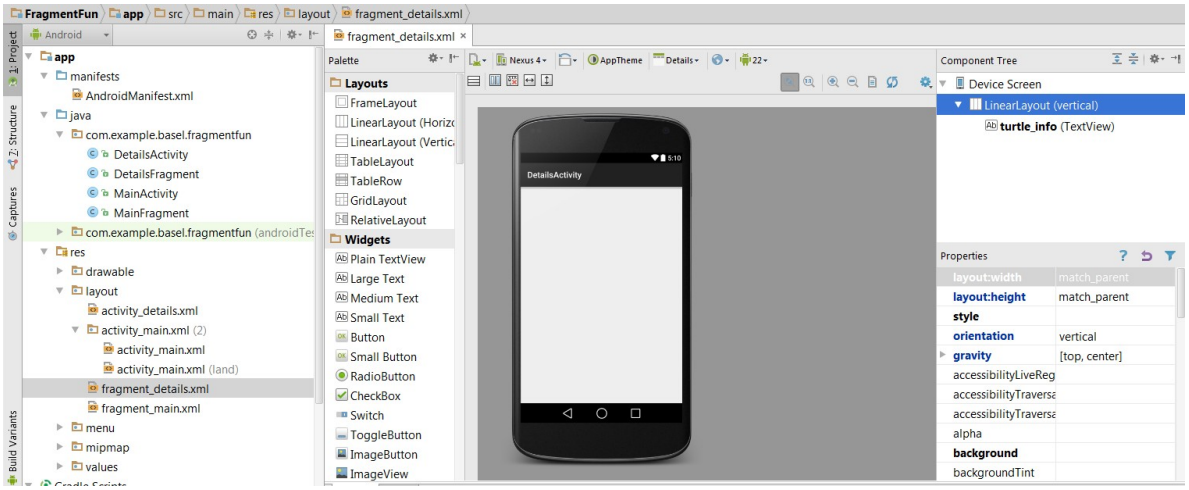


The fragment_main.xml file is:

```
<LinearLayout
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:gravity="top|center"
    android:orientation="vertical"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin">
<RadioGroup android:id="@+id/turtle_group"
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <RadioButton android:id="@+id/leo"
            android:text="Leo"
            android:checked="true"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <RadioButton android:id="@+id/mike"
            android:text="Mike"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <RadioButton android:id="@+id/don"
            android:text="Don"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <RadioButton android:id="@+id/raph"
            android:text="Raph"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
</RadioGroup>
<ImageButton android:id="@+id/turtle"
        android:src="@drawable/tmntleo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

3. Add new fragment (name it fragment_details.xml), and make the following design (that contains a TextView):



The fragment_details.xml file is:

```
<LinearLayout
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:gravity="top|center"
    android:orientation="vertical"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context="com.example.basel.fragmentfun.DetailsActivity">

<TextView android:id="@+id/turtle_info"
        android:text="" android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

4. Make the main activity activity_main.xml contains the first fragment:

```
<LinearLayout
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:gravity="top|center"
    android:orientation="vertical"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

<fragment android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/fragment1"
     android:name="com.example.basel.fragmentfun.MainFragment"
        tools:layout="@layout/fragment_main"/>
</LinearLayout>
```

5. Create a new activity activity_details.xml that contains the second activity:

```
<LinearLayout
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:gravity="top|center"
    android:orientation="vertical"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context="com.example.basel.fragmentfun.DetailsActivity">

<fragment android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/fragment2"
  android:name="com.example.basel.fragmentfun.DetailsFragment"
        tools:layout="@layout/fragment_details"/>

</LinearLayout>
```

6. In the folder res, make a new folder and name it layout-land and put a copy of the file activity_main.xml in this folder. Update the file activity_main.xml (land) to contain the two fragments.

```
<LinearLayout
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:gravity="top|center"
    android:orientation="horizontal"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">


<fragment android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/fragment1"
     android:name="com.example.basel.fragmentfun.MainFragment"
        tools:layout="@layout/fragment_main"/>
```

```
<fragment android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/fragment2"
  android:name="com.example.basel.fragmentfun.DetailsFragment"
        tools:layout="@layout/fragment_details"/>

</LinearLayout>
```

7. Leave the MainActivity.java without update:

```
package com.example.basel.fragmentfun;

import android.app.Activity;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
        }
    }
```

8. Leave the file DetailsActivity.java without update:

```
package com.example.basel.fragmentfun;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class DetailsActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details);
    }
}
```

9. Update the MainFragment.java file:

- Declare the radio button click event in the `onActivityCreated` method.

```
// attach event listener to radio button group
RadioGroup group = (RadioGroup)
getActivity().findViewById(R.id.turtle_group);
group.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
@Override
public void onCheckedChanged(RadioGroup group, int checkedId)
{
        updateTurtleImage();
    }
});
```

- The `updateTurtleImage()` method puts the character image in the image button.
- Declare the image button click event in the `onActivityCreated` method.

```
// attach click event listener to turtle image button
ImageButton img = (ImageButton)
getActivity().findViewById(R.id.turtle);
img.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        showDetailsAboutTurtle();
    }
});
```

- Declare the `showDetailsAboutTurtle()` method that shows the character information as follow:
  - In the landscape orientation, the information are shown in the second fragment in the same activity `DetailsFragment`.
  - In the portrait orientation, the activity `DetailsActivity` is shown and the information are displayed in this activity.

```
package com.example.basel.fragmentfun;
import android.content.Intent;
import android.content.res.Configuration;
import android.os.Bundle;
import android.app.Fragment;
```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageButton;
import android.widget.RadioGroup;

public class MainFragment extends Fragment {
// these "request codes" are used to identify sub-activities
that return results
private static final int REQUEST_CODE_DETAILS_ACTIVITY = 1234;

/*
* This method initialize the fragment's layout.
*/
@Override
public View onCreateView
        (LayoutInflater inflater, ViewGroup container,
                                Bundle savedInstanceState) {
      return inflater.inflate
             (R.layout.fragment_main, container, false);
}

/*
* This method is called after the containing activity is done
being created.
* This is a good place to attach event listeners and do other
initialization.
*/
@Override
public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

// attach event listener to radio button group
RadioGroup group = (RadioGroup)
       getActivity().findViewById(R.id.turtle_group);

group.setOnCheckedChangeListener(new
            RadioGroup.OnCheckedChangeListener() {
 @Override
 public void onCheckedChanged(RadioGroup group, int checkedId)
{
                updateTurtleImage();
            }
        });

// attach click event listener to turtle image button
ImageButton img = (ImageButton)
                    getActivity().findViewById(R.id.turtle);
img.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                showDetailsAboutTurtle();
            }
        });
    }

/*
* Shows more detailed information about the currently selected
ninja turtle.
* In portrait mode, this pops up a second activity.
* In landscape mode, this updates the DetailsFragment within
the same activity.
*/
public void showDetailsAboutTurtle() {
RadioGroup group = (RadioGroup)
         getActivity().findViewById(R.id.turtle_group);
        int id = group.getCheckedRadioButtonId();

        if (getResources().getConfiguration().orientation ==
                Configuration.ORIENTATION_LANDSCAPE) {
            // show in same activity
            DetailsFragment frag = (DetailsFragment)
       getFragmentManager().findFragmentById(R.id.fragment2);
            frag.setTurtleId(id);
        } else {
            // launch details as its own activity
            Intent intent = new Intent(getActivity(),
                           DetailsActivity.class);
            intent.putExtra("turtle_id", id);
            startActivityForResult(intent,
                  REQUEST_CODE_DETAILS_ACTIVITY);
        }
    }

/*
* Updates which turtle image is showing
* based on which radio button is currently checked.
*/
private void updateTurtleImage() {
        ImageButton img = (ImageButton)
                  getActivity().findViewById(R.id.turtle);
        RadioGroup group = (RadioGroup)
            getActivity().findViewById(R.id.turtle_group);
        int checkedID = group.getCheckedRadioButtonId();
        if (checkedID == R.id.leo) {
            img.setImageResource(R.drawable.tmntleo);
        } else if (checkedID == R.id.mike) {
            img.setImageResource(R.drawable.tmntmike);
        } else if (checkedID == R.id.don) {
            img.setImageResource(R.drawable.tmntdon);
        } else if (checkedID == R.id.raph) {
            img.setImageResource(R.drawable.tmntraph);
```

```
        }

        if (getResources().getConfiguration().orientation ==
                Configuration.ORIENTATION_LANDSCAPE) {
            showDetailsAboutTurtle();
        }
    }
}
```

10. Write the DetailsFragment.java methods:

```
package com.example.basel.fragmentfun;

import android.content.Intent;
import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class DetailsFragment extends Fragment {
    /*
     * Constant array of data about each of the four turtles.
     */
    private static final String[] TURTLE_DETAILS = {
            "Leonardo, or Leo, is one of the four protagonists
of the Teenage Mutant Ninja Turtles comics and all related
media. In the Mirage/Image Comics, all four turtles wear red
bandanas, but in other versions, he wears a blue bandana. His
signature weapons are two ninjato. Throughout the various
media, he is often depicted as the eldest and leader of the
four turtles, as well as the most disciplined. He is named
after Leonardo da Vinci. In the 2012 series, he is the only
turtle who harbors strong romantic affections for Karai,
considering her his love interest.",
            "Michelangelo, Mike or Mikey (as he is usually
called), is a fictional character and one of the four
protagonists of the Teenage Mutant Ninja Turtles comics and
all related media. His mask is typically portrayed as orange
outside of the Mirage/Image Comics and his weapons are dual
nunchucks, though he has also been portrayed using other
weapons, such as a grappling hook, manriki-gusari, tonfa, and
a three section staff (in some action figures).",
            "Donatello, often shortened to Don, Donny or
Donnie, is a fictional character and one of the four
protagonists of the Teenage Mutant Ninja Turtles comics and
all related media. He is co-creator Peter Laird's favorite
Turtle. In the Mirage/Image Comics, all four turtles wear red
```

```
bandanas, but in other versions he wears a purple bandana. His
primary signature weapon is his effective b?? staff. In all
media, he is depicted as the smartest and second-in-command of
the four turtles. Donnie often speaks in technobabble with a
natural aptitude for science and technology. He is named after
the Italian sculptor Donatello.",
            "Raphael, or Raph, is a fictional character and
one of the four protagonists of the Teenage Mutant Ninja
Turtles comics and all related media. In the Mirage/Image
Comics, all four turtles wear red bandanas over their eyes,
but unlike his brothers in other versions, he is the only one
who keeps the red bandana. Raphael wields twin sai as his
primary weapon. (In the Next Mutation series, his sai stick
together to make a staff-like weapon.) He is generally the
most likely to experience extremes of emotion, and is usually
depicted as being aggressive, sullen, maddened, and
rebellious. The origin of Raphael's anger is not always fully
explored, but in some incarnations appears to stem partly from
the realization that they are the only creatures of their kind
and ultimately alone. He also has a somewhat turbulent
relationship with his older brother Leonardo because Leonardo
is seen as the group's leader. Raphael is named after the
16th-century Italian painter Raphael. In 2011 Raphael placed
23rd on IGN's Top 100 Comic Book Heroes, a list that did not
feature any of his brothers."
    };

    /*
     * This method initialize the fragment's layout.
     */
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                           Bundle savedInstanceState) {
   return inflater.inflate(R.layout.fragment_details,
container, false);
    }

/*
* This method is called after the containing activity is done
being created.
* This is a good place to attach event listeners and do other
initialization.
*/
@Override
public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        // pull out the turtle ID to show from the activity's
intent
        Intent intent = getActivity().getIntent();
```

```
        int id = intent.getIntExtra("turtle_id", R.id.leo);
        setTurtleId(id);
    }
/*
* Sets the actively selected ninja turtle text based on the
given resource ID.
*/
    public void setTurtleId(int id) {
        int index;
        if (id == R.id.leo) {
            index = 0;
        } else if (id == R.id.mike) {
            index = 1;
        } else if (id == R.id.don) {
            index = 2;
        } else { // if (id == R.id.raph)
            index = 3;
        }

        String text = TURTLE_DETAILS[index];
        TextView tv = (TextView)

      getActivity().findViewById(R.id.turtle_info);
        tv.setText(text);
    }
}
```

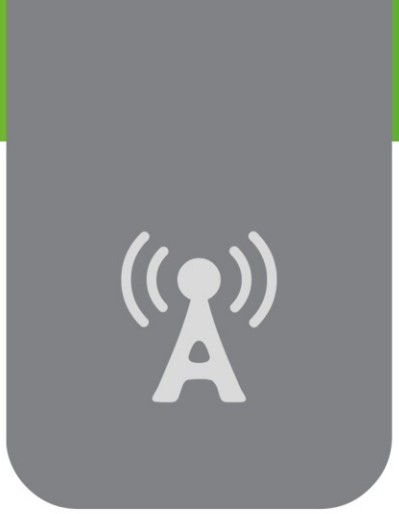
Exercise:

Build the following application, which has two buttons.

When you click the first button the first fragment appears:

When you click the second button the second fragment appears:

Update the application to show the fragments in a second activity if the orientation is landscape.



Chapter 12:
Using Google Maps

Key Words:

Google Maps, API key, Tracking user location.

Summary:

This unit provides an overview of dealing with maps.

Outcomes:

Student will learn in this unit:

- Getting API key.
- Including maps in the applications.
- Adding items to maps.
- Tracking user location.

Plan:

2 Learning Objects

1. Google Maps
2. Example

1. Google Maps

Learning outcomes:

Using Google Maps.

Google won't allow you to fetch map data without an **API key**.

Getting API Key:

The following steps show how to get an API key:

1. Copy the file debug.keystore from the folder:

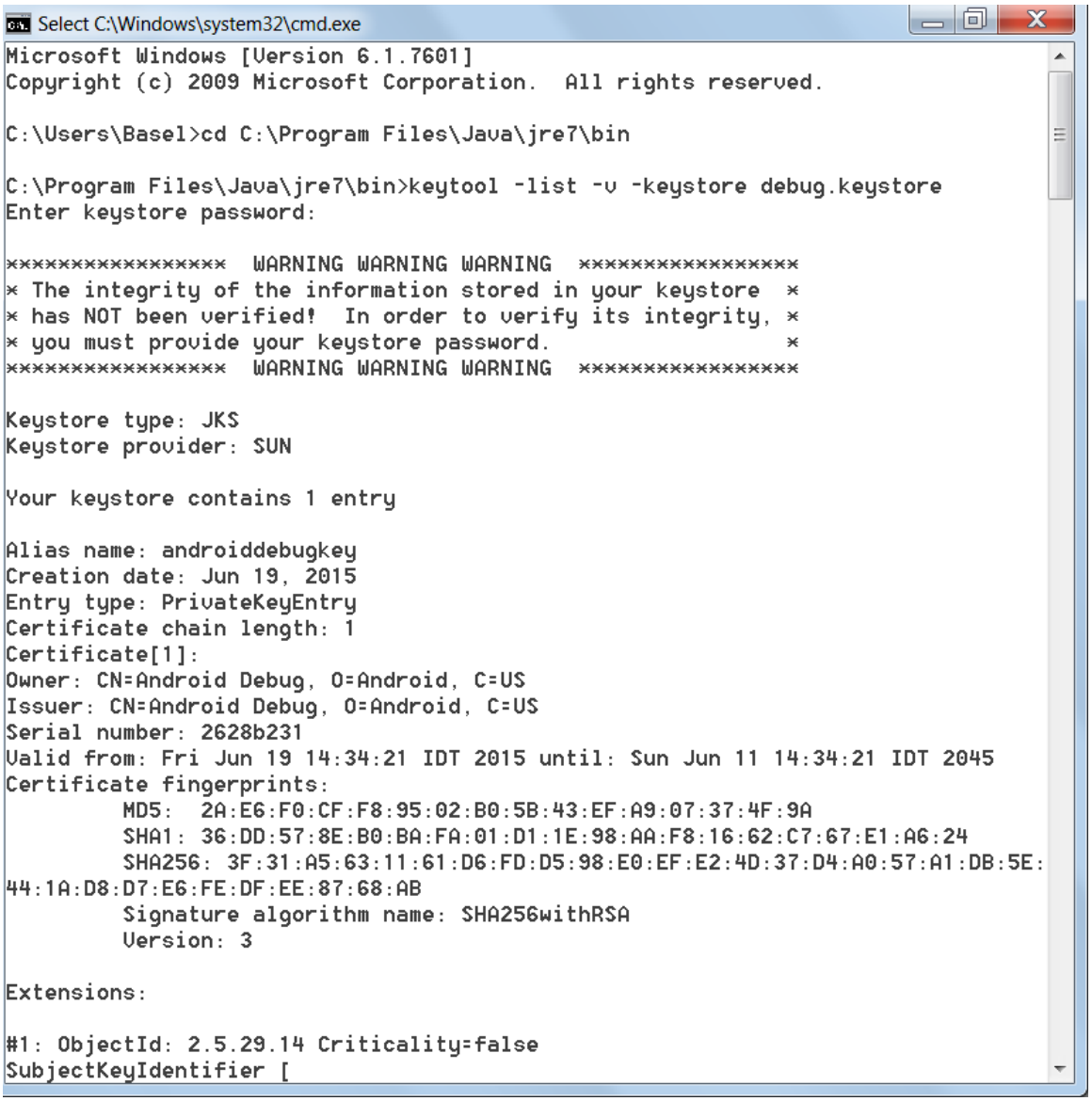
   C:\Windows\USERNAME\.Android

   To the folder:

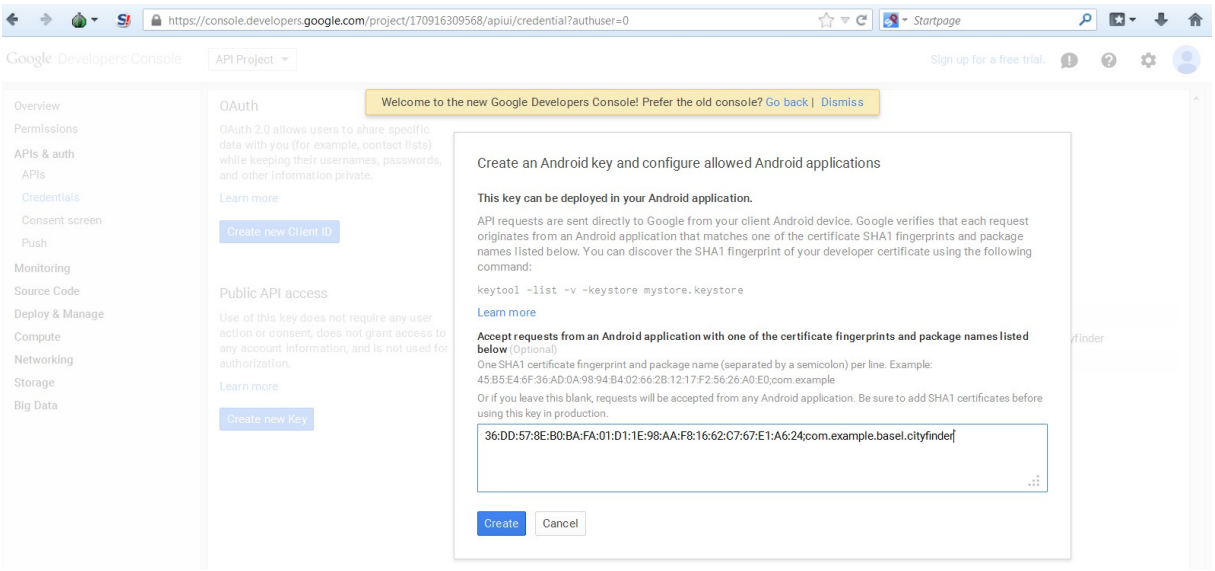
   C:\Program Files\Java\jre7\bin
2. Open a Terminal and move to the previous folder:

   cd C:\Program Files\Java\jre7\bin
3. Execute the command:

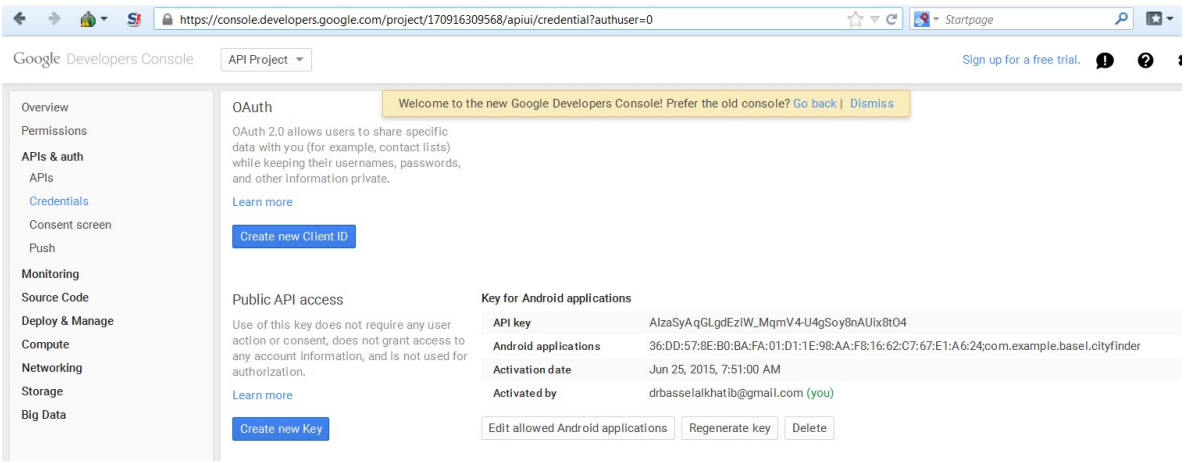
   keytool –list –v –keystore debug.keystore


Select C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\Basel>cd C:\Program Files\Java\jre7\bin
C:\Program Files\Java\jre7\bin>keytool -list -v -keystore debug.keystore
Enter keystore password:
********** WARNING WARNING WARNING **********
* The integrity of the information stored in your keystore *
* has NOT been verified! In order to verify its integrity, *
* you must provide your keystore password. *
********** WARNING WARNING WARNING **********
Keystore type: JKS
Keystore provider: SUN
Your keystore contains 1 entry
Alias name: androiddebugkey
Creation date: Jun 19, 2015
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 2628b231
Valid from: Fri Jun 19 14:34:21 IDT 2015 until: Sun Jun 11 14:34:21 IDT 2045
Certificate fingerprints:
MD5: 2A:E6:F0:CF:F8:95:02:B0:5B:43:EF:A9:07:37:4F:9A
SHA1: 36:DD:57:8E:B0:BA:FA:01:D1:1E:98:AA:F8:16:62:C7:67:E1:A6:24
SHA256: 3F:31:A5:63:11:61:D6:FD:D5:98:E0:EF:E2:4D:37:D4:A0:57:A1:DB:5E:
44:1A:D8:D7:E6:FE:DF:EE:87:68:AB
Signature algorithm name: SHA256withRSA
Version: 3
Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [

4. Find the line with your "Certificate fingerprint" for "SHA-1". It should contain a long string in this format (36:DD:...). Copy it down.
5. Open the developer web page:
   https://code.google.com/apis/console/
6. Ask for Android key:
   Credentials → Create new Key



7. Enter the Certificate fingerprint followed by semi column and the package name.
   For example:
   36:DD:....;com.example.basel.cityfinder:)
   A new page will show containing the required key (API key)

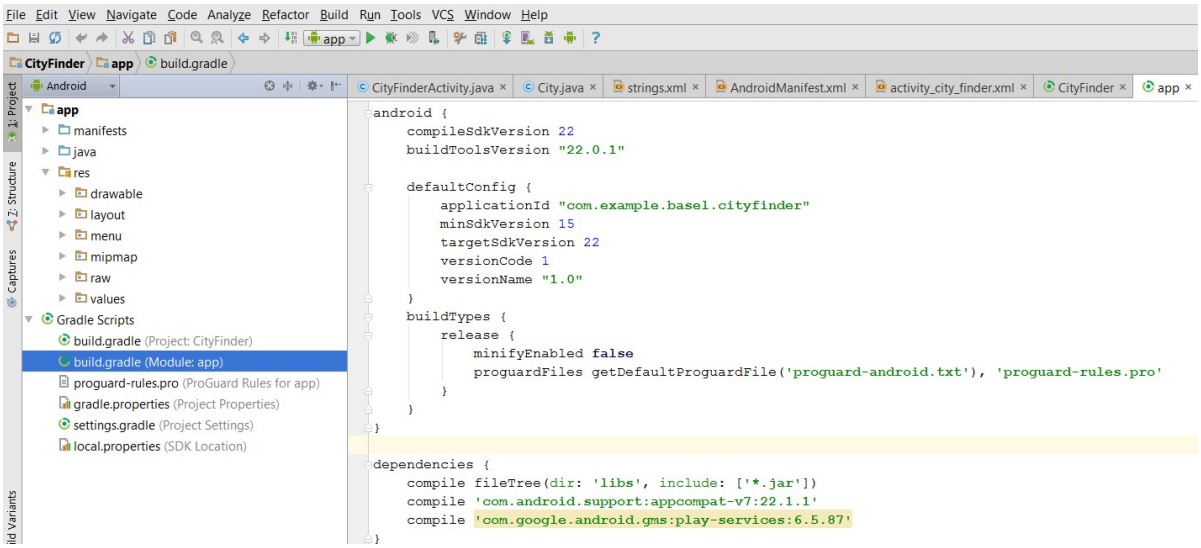


8. Update the Androidmanifest.xml file to enter the API key:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.basel.cityfinder">
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-feature android:glEsVersion="0x00020000"
        android:required="true"/>
<application android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
android:theme="@style/AppTheme">
<meta-data android:name="com.google.android.gms.version"

android:value="@integer/google_play_services_version"/>
<meta-data android:name="com.google.android.maps.v2.API_KEY"
android:value="AIzaSyAqGLgdEzlW_MqmV4-U4gSoy8nAUix8tO4"/>

<activity
android:name="com.example.basel.cityfinder.CityFinderActivity"
            android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
</manifest>
```

9. Add Google Play to the .build.gradle file

```
dependencies {
compile fileTree(dir: 'libs', include: ['*.jar'])
compile 'com.android.support:appcompat-v7:21.0.3'
compile 'com.google.android.gms:play-services:6.5.87'
}
```



MapFragment:

Google Maps API provides a fragment class named MapFragment for displaying a map within an activity.

```
<LinearLayout ...
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:map="http://schemas.android.com/apk/res-auto"
tools:ignore="MissingPrefix">
<fragment ...
android:name="com.google.android.gms.maps.MapFragment"
android:id="@+id/ID" />
</LinearLayout>
```

Waiting for map to be ready:

Map loading usually takes some time. We do the following:

```
public class Name extends Activity
      implements OnMapReadyCallback, GoogleMap. OnMapLoadedCallback {
   private GoogleMap map = null;

   @Override
   protected void onCreate (Bundle savedInstanceState) {
      …
      MapFragment mf = (MapFragment) get FragmentAnager().
   findFragmentById(R.id.ID);
      mf.getMapAsync(this);                    //calls onMapReady when loaded
   }

   @Override
   public void onMapReady(GoogleMap map) { // map is loaded but not laid out yet
       map.setOnMapLoadedCallback(this);   //calls onMapLoaded when layout done
   }

   @Override
   public void onMapLoaded() {
   code to run when the map has loaded;
   }
}
```

Google Map methods:

Placing items on the map

- `addCircle`, `addGroundOverlay`, **`addMarker`**, `addPolygon`, `addPolyline`, **`addTileOverlay`**
- **`clear`** – Removes all markers, polylines/polygons, overlays

Manipulating the camera:

- `getCameraPosition`, **`moveCamera`**, **`animateCamera`**, `stopAnimation`

Map settings and appearance:

- `setBuildingsEnabled`, `setIndoorEnabled`, `setMapType`, `setPadding`, `setTrafficEnabled`

`Snapshot` – take a screen shot of the map as a bitmap

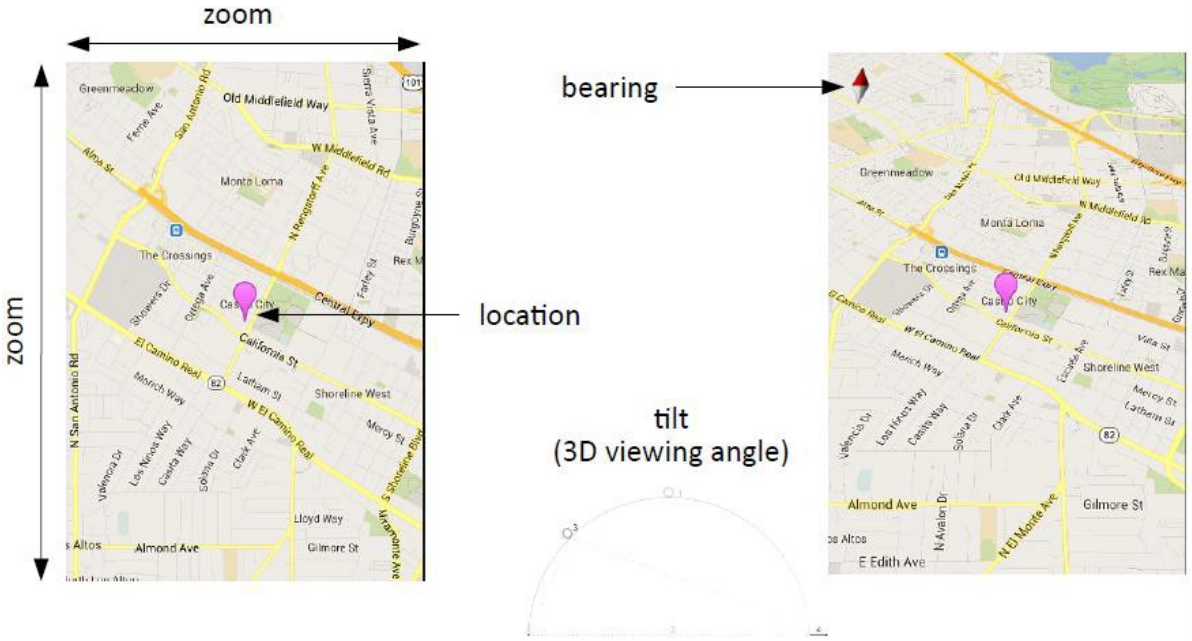
Event listeners:

- `setOnCameraChangeListener`, **`setOnMapClickListener`**, `setOnMapLoadedCallback`, `setOnMapLongClickListener`, **`setOnMarkerClickListener`**, `setOnMarkerDragListener`, `setOnMyLocationChangeListener`

The map camera:

The current viewing window of a map's camera is defined by:

- target location(latitude/longitude)
- zoom (2.0 - 21.0)
- bearing (orientation/rotation)
- tilt (degrees)



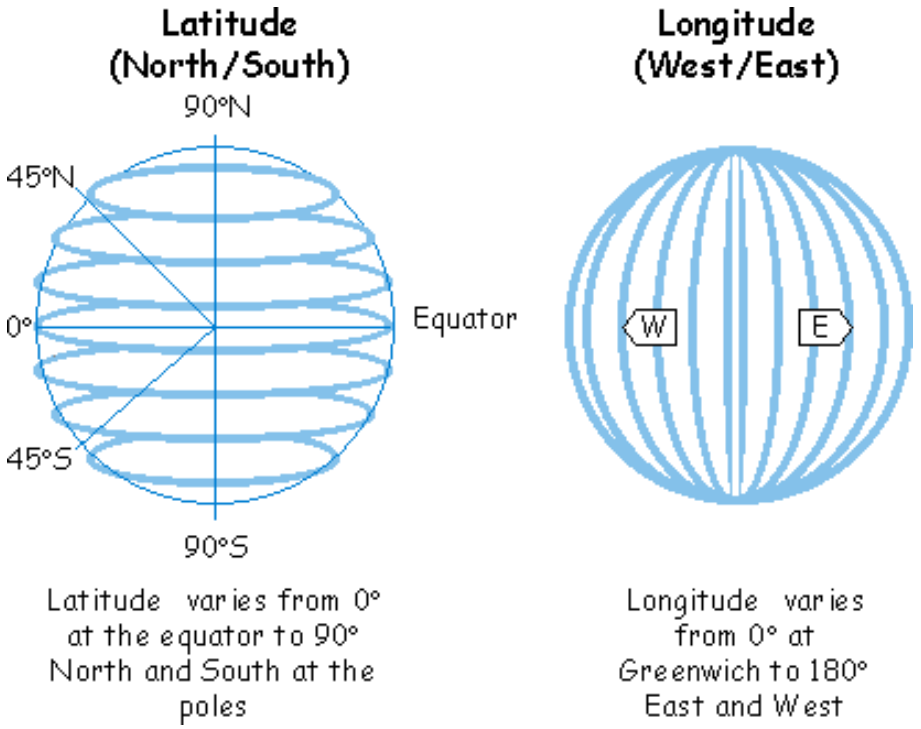
Latitude and longitude:

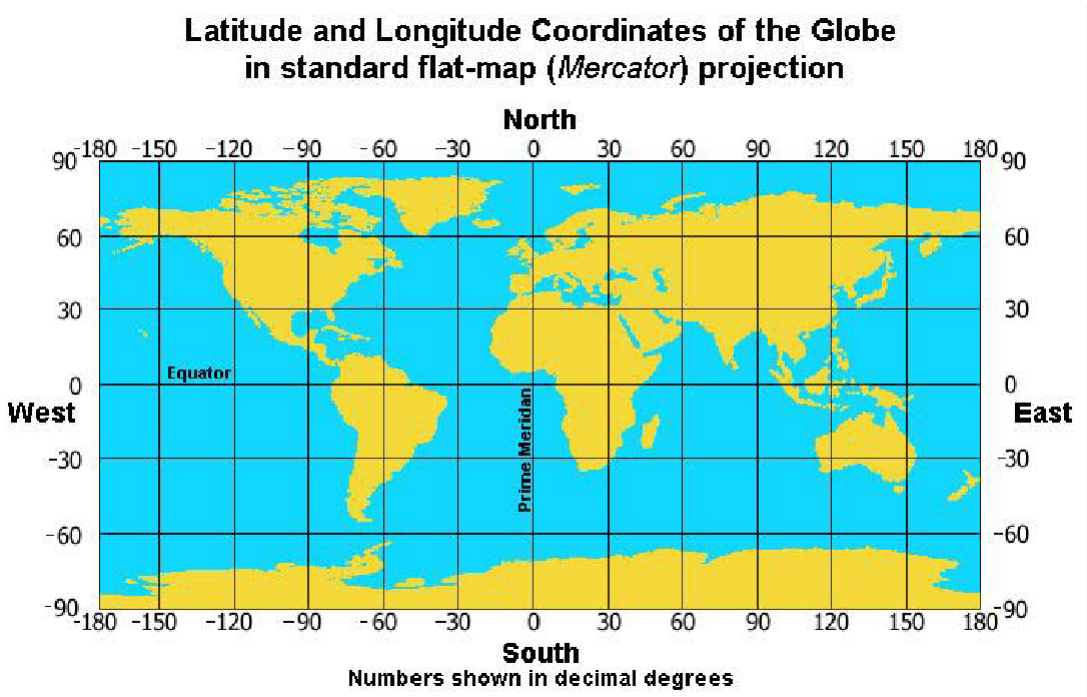
- **latitude:** N / S angle relative to the equator

North pole = +90; South pole = −90

- **longitude:** E/W angle relative to prime meridian

West = 0 → −180; East = 0 → 180





Set camera in XML:

Set initial map settings and camera position in the layout XML:

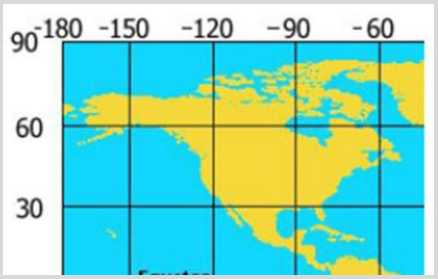
```
<fragment ...
android:name="com.google.android.gms.maps.MapFragment"
android:id="@+id/ID"
map:cameraBearing="112.5"
map:cameraTargetLat="-33.796923"
map:cameraTargetLng="150.922433"
map:cameraTilt="30"
map:cameraZoom="13"
map:mapType="normal"
map:uiCompass="false"
map:uiRotateGestures="true"
map:uiScrollGestures="false"
map:uiTiltGestures="true"
map:uiZoomControls="false"
map:uiZoomGestures="true" />
```

Set camera in Java Code:

Use `CameraUpdateFactory` methods:

- `newLatLng(new LatLng(`***lat, lng***`))`
- `newLatLngBounds(new LatLngBounds(`***SW, NE***`),` ***padding***`)`
- `newLatLngZoom(new LatLng(`***lat, lng***`),` ***zoom***`)`
- `newCameraPosition(`***CameraPosition***`)`
- others:

```
// example; show roughly the entire USA
 LatLngBounds bounds = new LatLngBounds(
 new LatLng(20, -130.0), // SW
 new LatLng(55, -70.0)); // NE
 map.moveCamera
CameraUpdateFactory.newLatLngBounds(bounds, 50));
```



Placing Markers:

A `GoogleMap` object has an `addMarker` method that can let you put "push pin" markers at locations on the map.

- The marker's methods return the marker, so you can chain them.
- options: `alpha, draggable, icon, position, rotation, title, visible, ...`

```
map.addMarker(new MarkerOptions()
.position(new LatLng(40.801, -96.691))
.title("Lincoln, NE")
);
// to modify/remove the marker later
Marker mark = map.addMarker
                (new MarkerOptions()
...);
mark.remove();
```

Lines and paths:

A `GoogleMap` object has an `addPolyline` method that can let you put lines between locations on the map.

- options: `color, visible, width, zIndex, ...`

```
map.addPolyline(new PolylineOptions()
.add(new LatLng(40.801, -96.691)) // Lincoln, NE
.add(new LatLng(34.020, -118.412)) // Los Angeles,
CA
.add(new LatLng(40.703, -73.980)) // New York, NY
);
// to modify/remove the line later
Polyline polly = map.addPolyline(...);
polly.remove();
```

Accessing phone's location:

Android `LocationManager` gives you the phone's position:

- GPS provider provides highest accuracy
- Network provider is a fallback in case GPS is disabled / not present

```
getSystemService(Context.LOCATION_SERVICE);
Location loc = locationManager.getLastKnownLocation(
LocationManager.GPS_PROVIDER);
if (loc == null) {
// fall back to network if GPS is not available
loc = locationManager.getLastKnownLocation(
LocationManager.NETWORK_PROVIDER);
}
if (loc != null) {
double myLat = loc.getLatitude();
double myLng = loc.getLongitude();
...
// other methods: getAltitude, getSpeed, getBearing, ...
```

AndroidManifest.xml:

Because of privacy issues, to access phone's current location, must request permission in AndroidManifest.xml:

```
<manifest ...>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
<application ...>
...
</application>
</manifest>
```

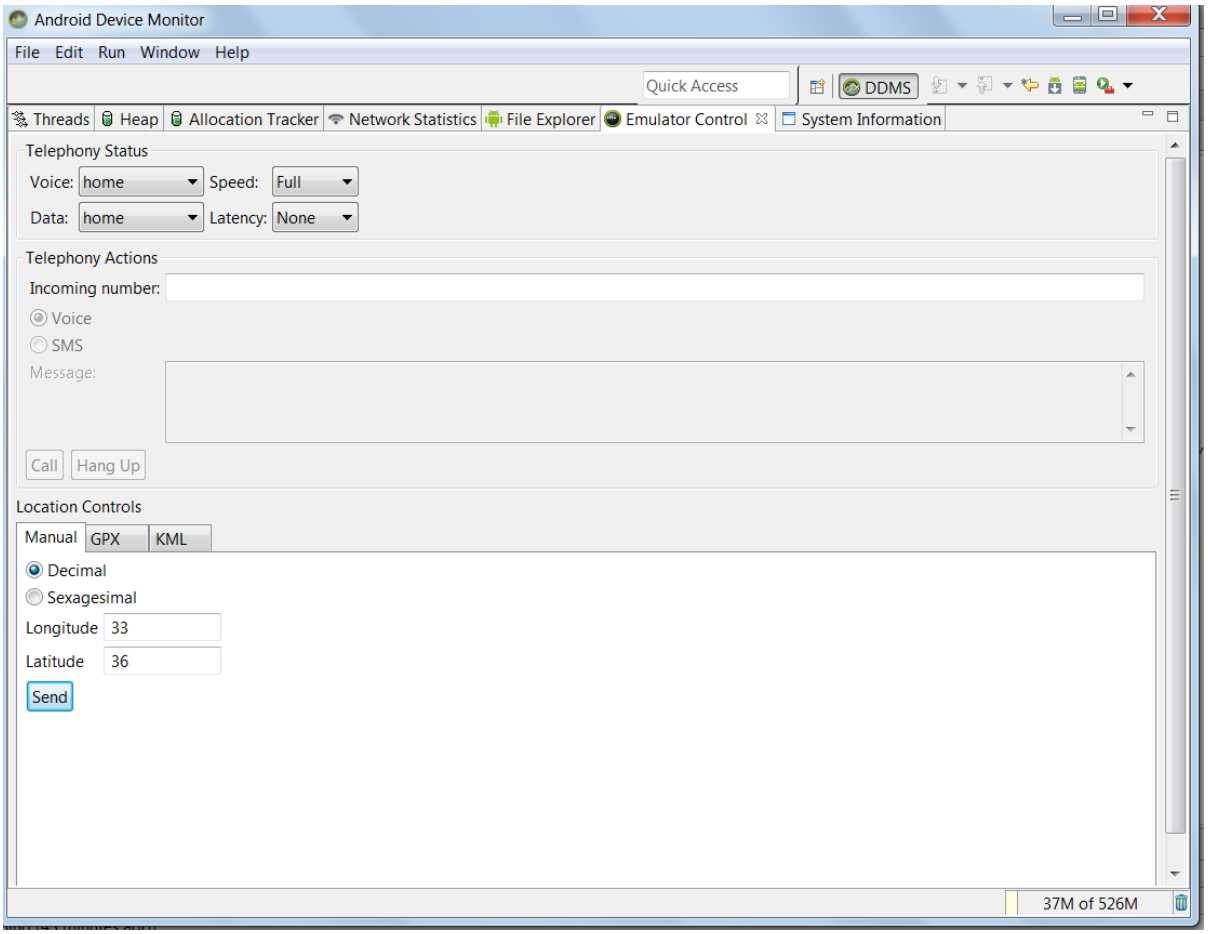
Location update events:

You can track user location using the following:

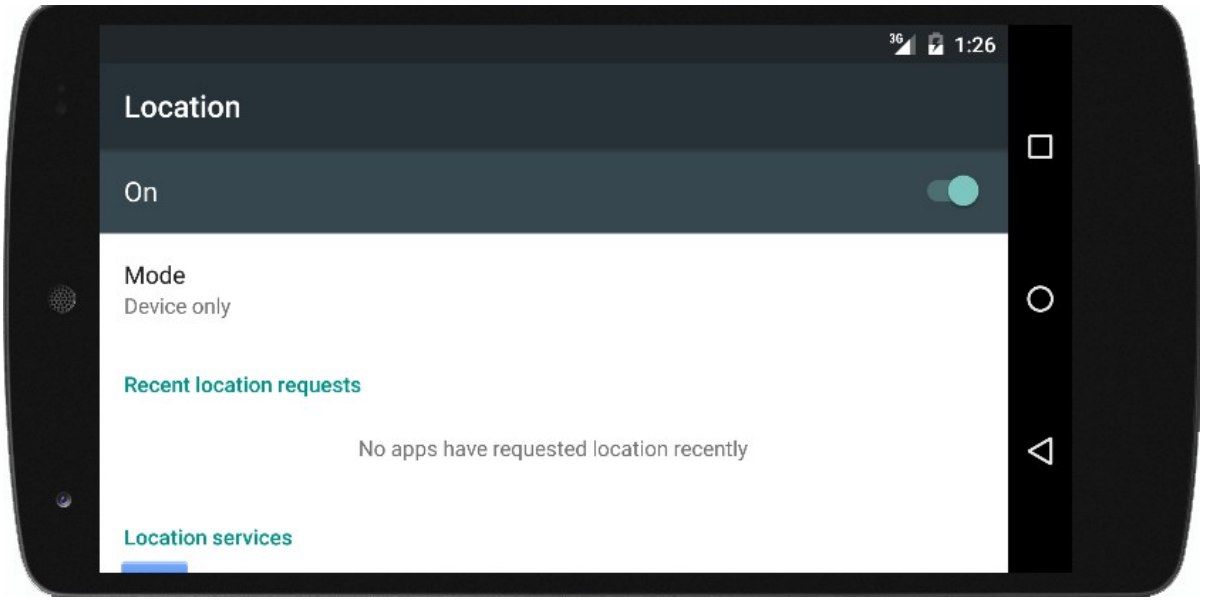
```
LocationManager locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
locationManager.requestLocationUpdates(
LocationManager.GPS_PROVIDER, 0, 0, // provider, min
time/distance
new LocationListener() {
public void onLocationChanged(Location location) {
// code to run when user's location changes
}
public void onStatusChanged(String prov, int stat, Bundle b){}
public void onProviderEnabled(String provider) {}
public void onProviderDisabled(String provider) {}
}
);
```

Emulator location:

You can set the emulator location using Android Device Monitor:



Use Google Settings to make Location=ON in the device.



2. Example

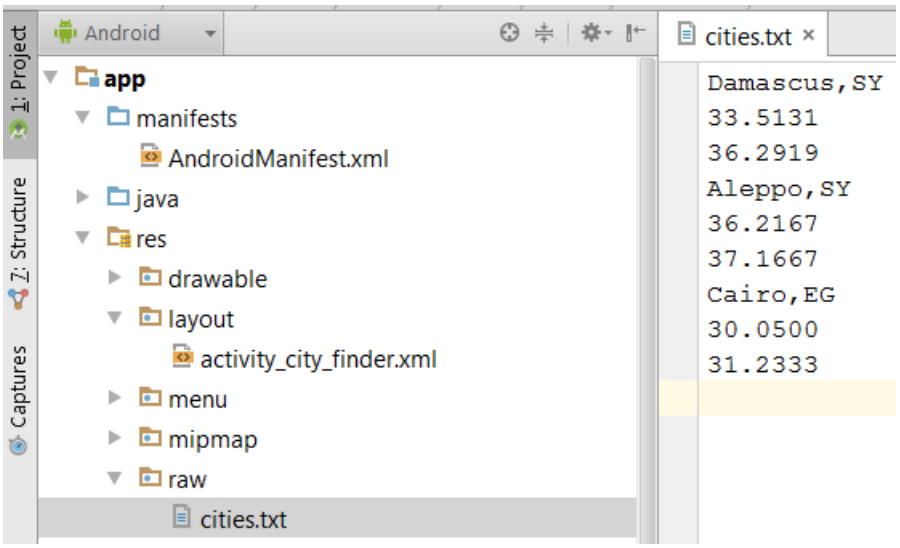
Learning outcomes:

Example:

When you run the app, the map appears with visible signs on some cities:

The file raw/cities.txt contains some cities data:

The file raw/cities.txt contains some cities data:



The activity file is:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="top|center"
    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:ignore="MissingPrefix"
    tools:context=".CityFinderActivity">
<TextView android:text="CityFinder!" android:textSize="30sp"
        android:layout_marginBottom="10dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
<fragment
android:name="com.google.android.gms.maps.MapFragment"
        android:id="@+id/the_map"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        map:cameraTargetLat="33"
        map:cameraTargetLng="36"
        map:cameraZoom="3"/>
<TextView android:text="Click a city to get started."
```

```
        android:layout_marginTop="10dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

The code file is:

```
package com.example.basel.cityfinder;
import android.app.*;
import android.content.*;
import android.location.*;
import android.os.*;
import android.widget.*;

import com.example.basel.cityfinder.R;
import com.google.android.gms.maps.*;
import com.google.android.gms.maps.model.*;
import java.util.*;

public class CityFinderActivity extends Activity
        implements OnMapReadyCallback,
GoogleMap.OnMapLoadedCallback, GoogleMap.OnMarkerClickListener
{

    private GoogleMap map;
    private LatLng myLocation;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_city_finder);

        MapFragment mf = (MapFragment)
        getFragmentManager().findFragmentById(R.id.the_map);
        mf.getMapAsync(this);  // calls onMapReady when loaded

    }

    @Override
    public void onMapReady(GoogleMap map) {
        // map is loaded but not laid out yet
        this.map = map;
        map.setOnMapLoadedCallback(this);
        // calls onMapLoaded when layout done
    }

    @Override
    public void onMapLoaded() {
        // code to run when the map has loaded
```

```
        readCities();
        map.setOnMarkerClickListener(this);

        // read user's current location, if possible
        myLocation = getMyLocation();
        if (myLocation == null) {
            Toast.makeText(this, "Unable to access your
location. Consider enabling Location in your device's
Settings.", Toast.LENGTH_LONG).show();
        } else {
            map.addMarker(new MarkerOptions()
                          .position(myLocation)
                          .title("ME!")
            );
        }
    }

    /*
     * Reads a list of cities from a text file and draws
       a marker for each one on the map.
     */
    private void readCities() {
      Scanner scan = new
        Scanner(getResources().openRawResource(R.raw.cities));
        while (scan.hasNextLine()) {
            String name = scan.nextLine();
            if (name.isEmpty()) break;
            double lat = Double.parseDouble(scan.nextLine());
            double lng = Double.parseDouble(scan.nextLine());
            map.addMarker(new MarkerOptions()
                          .position(new LatLng(lat, lng))
                          .title(name)
            );
        }
    }

    /*
     * Returns the user's current location as a LatLng object.
     * Returns null if location could not be found (such as in
an AVD emulated virtual device).
     */
private LatLng getMyLocation() {
// try to get location three ways: GPS, cell/wifi network, and
'passive' mode
  LocationManager locationManager = (LocationManager)
                getSystemService(Context.LOCATION_SERVICE);
  Location loc = locationManager.getLastKnownLocation
                          (LocationManager.GPS_PROVIDER);
        if (loc == null) {
            // fall back to network if GPS is not available
            loc = locationManager.getLastKnownLocation
```

```
                        (LocationManager.NETWORK_PROVIDER);
        }
        if (loc == null) {
// fall back to "passive" location if GPS and network are not
available
            loc = locationManager.getLastKnownLocation
                        (LocationManager.PASSIVE_PROVIDER);
        }

        if (loc == null) {
            return null;   // could not get user's location
        } else {
            double myLat = loc.getLatitude();
            double myLng = loc.getLongitude();
            return new LatLng(myLat, myLng);
        }
    }

    /*
     * Called when user clicks on any of the city map markers.
     * Adds a line from the user's location to that city.
     */
    @Override
    public boolean onMarkerClick(Marker marker) {
        if (myLocation != null) {
            LatLng markerLatLng = marker.getPosition();
            map.addPolyline(new PolylineOptions()
                        .add(myLocation)
                        .add(markerLatLng)
            );
            return true;
        } else {
            return false;
        }
    }
}
```

The class City is defined as:

```
package com.example.basel.cityfinder;

import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;

public class City {
    public String name;
    public double lat;
    public double lng;
```

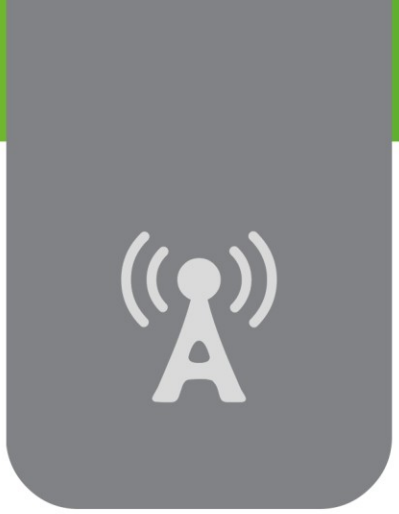
```
    public LatLng latLng;
    public Marker marker;

    public City(String name, double lat, double lng) {
        this.name = name;
        this.lat = lat;
        this.lng = lng;
        this.latLng = new LatLng(lat, lng);
    }

    public String toString() {
        return name;
    }
}
```

Exercise:

You are requested to build an application that allows showing maps with a layer representing some weather layer: Temperature, Wind, Pressure, …

For help, visit the link:

http://www.survivingwithandroid.com/2015/03/android-google-map-add-weather-data-tile.html



Chapter 13:
Mobile Databases

Key Words:

DataBase, Table, Query.

Summary:

This unit provides an overview of dealing with databases.

Outcomes:

Student will learn in this unit:

- Create databases.
- Create tables
- Add/Edit/Delete records.
- Querying data.

Plan:

1 Learning Object

1. Mobile Databases

1. Mobile Databases

Learning outcome:

using mobile database: SQLite.

The class SQLiteDatabase from the package android.database.sqlite and the class Cursor from the package android.database provide all necessary methods for creating database tables and then making necessary queries.

Creating database:

You can use the method `openOrCreateDatabase` to create a database as shown in the following example:

```
SQLiteDatabase db;
db=openOrCreateDatabase("StudentDB", Context.MODE_PRIVATE,
null);
```

The method `openOrCreateDatabase` opens the database `StudentDB` if it is already exists, if the database does not exist, it creates a new database named `StudentDB`.

- The first parameter of the method specifies the database name to open or to create.
- The second parameter `Context.MODE_PRIVATE` specifies that the database can be accessed only by the calling application or any another application that has the user ID.
- The third parameter is a Cursor factory object and can be left null if it is not required.

Creating table:

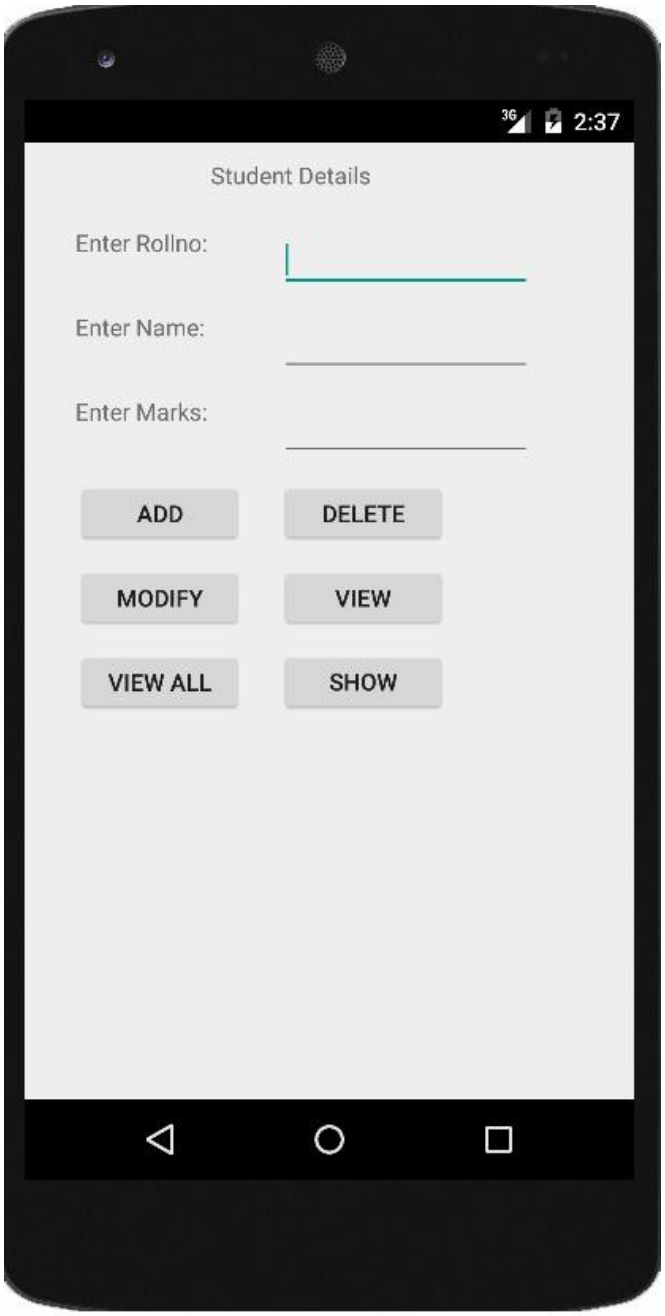
You can use the method `execSQL` of the class `SQLiteDatabase` to execute SQL statements on the database.

The following example creates the table student that has three fields:

1. Student rolling number `rollno`
2. Student name `name`
3. Students marks `marks`

```
db.execSQL("CREATE TABLE IF NOT EXISTS student(rollno
VARCHAR,name VARCHAR,marks VARCHAR);");
```

Example: Simple Students Management System:



- The application interface contains three text views to enter student data.
- The application interface provides the following buttons:
  - `ADD`: to insert the entered student data to the students table.
  - `DELETE`: to delete the student with the specified `rollno`.
  - `MODIFY`: to update student data (name and marks) that has the specified `rollno`.
  - `VIEW`: to display student data that has the specified `rollno`.
  - `VIEW ALL`: to display all students' data.
  - `SHOW`: to display information message.

The following is the activity_main.xml file:

```
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLayout"
    android:stretchColumns="0"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:text="@string/title"
        android:layout_x="110dp"
        android:layout_y="10dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView android:text="@string/roll_no"
        android:layout_x="30dp"
        android:layout_y="50dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText android:id="@+id/editRollno"
        android:inputType="number"
        android:layout_x="150dp"
        android:layout_y="50dp"
        android:layout_width="150dp"
        android:layout_height="40dp"/>
    <TextView android:text="@string/name"
        android:layout_x="30dp"
        android:layout_y="100dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText android:id="@+id/editName"
        android:inputType="text"
        android:layout_x="150dp"
        android:layout_y="100dp"
        android:layout_width="150dp"
```

```
        android:layout_height="40dp"/>
    <TextView android:text="@string/marks"
        android:layout_x="30dp"
        android:layout_y="150dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText android:id="@+id/editMarks"
        android:inputType="number"
        android:layout_x="150dp"
        android:layout_y="150dp"
        android:layout_width="150dp"
        android:layout_height="40dp"/>
    <Button       android:id="@+id/btnAdd"
        android:text="@string/add"
        android:layout_x="30dp"
        android:layout_y="200dp"
        android:layout_width="100dp"
        android:layout_height="40dp"
        android:onClick="onClick" />
    <Button       android:id="@+id/btnDelete"
        android:text="@string/delete"
        android:layout_x="150dp"
        android:layout_y="200dp"
        android:layout_width="100dp"
        android:layout_height="40dp"
        android:onClick="onClick" />
    <Button    android:id="@+id/btnModify"
        android:text="@string/modify"
        android:layout_x="30dp"
        android:layout_y="250dp"
        android:layout_width="100dp"
        android:layout_height="40dp"
        android:onClick="onClick" />
    <Button    android:id="@+id/btnView"
        android:text="@string/view"
        android:layout_x="150dp"
        android:layout_y="250dp"
        android:layout_width="100dp"
        android:layout_height="40dp"
        android:onClick="onClick" />
    <Button    android:id="@+id/btnViewAll"
        android:text="@string/view_all"
        android:layout_x="30dp"
        android:layout_y="300dp"
        android:layout_width="100dp"
        android:layout_height="40dp"
        android:onClick="onClick" />
    <Button    android:id="@+id/btnShowInfo"
        android:text="@string/show_info"
        android:layout_x="150dp"
```

```
        android:layout_y="300dp"
        android:layout_width="100dp"
        android:layout_height="40dp"
        android:onClick="onClick" />
</AbsoluteLayout  >
```

We start in the activity file MainActivity.java by declaring associated variables with the different widgets; we declare also the variable db of `SQLiteDatabase` type to deal with the database.

```
public class MainActivity extends  Activity {
EditText editRollno,editName,editMarks;
Button btnAdd,btnDelete,btnModify,btnView,btnViewAll,
       btnShowInfo;
SQLiteDatabase db;
```

In the `OnCreate` method, we access the widgets, and create the `StudentDB` database (or open it, if it is already exists).

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Initializing controls
    editRollno=(EditText)findViewById(R.id.editRollno);
    editName=(EditText)findViewById(R.id.editName);
    editMarks=(EditText)findViewById(R.id.editMarks);
    btnAdd=(Button)findViewById(R.id.btnAdd);
    btnDelete=(Button)findViewById(R.id.btnDelete);
    btnModify=(Button)findViewById(R.id.btnModify);
    btnView=(Button)findViewById(R.id.btnView);
    btnViewAll=(Button)findViewById(R.id.btnViewAll);
    btnShowInfo=(Button)findViewById(R.id.btnShowInfo);

    // Creating database and table
    db=openOrCreateDatabase("StudentDB", Context.MODE_PRIVATE,
null);
    db.execSQL("CREATE TABLE IF NOT EXISTS student(rollno
VARCHAR,name VARCHAR,marks VARCHAR);");
}
```

All interface buttons are linked (using the onClick property) with the following method:

```
public void onClick(View view)
{
    // Adding a record
    if(view==btnAdd)
    {
        // Checking empty fields

if(editRollno.getText().toString().trim().length()==0||

editName.getText().toString().trim().length()==0||

editMarks.getText().toString().trim().length()==0)
        {
            showMessage("Error", "Please enter all values");
            return;
        }
        // Inserting record
        db.execSQL("INSERT INTO student
VALUES('"+editRollno.getText()+"','"+editName.getText()+
                "','"+editMarks.getText()+"');");
        showMessage("Success", "Record added");
        clearText();
    }
    // Deleting a record
    if(view==btnDelete)
    {
        // Checking empty roll number
        if(editRollno.getText().toString().trim().length()==0)
        {
            showMessage("Error", "Please enter Rollno");
            return;
        }
        // Searching roll number
        Cursor c=db.rawQuery("SELECT * FROM student WHERE
                  rollno='"+editRollno.getText()+"'", null);
        if(c.moveToFirst())
        {
            // Deleting record if found
            db.execSQL("DELETE FROM student WHERE
                        rollno='"+editRollno.getText()+"'");
            showMessage("Success", "Record Deleted");
        }
        else
        {
            showMessage("Error", "Invalid Rollno");
        }
```

```
            clearText();
        }
        // Modifying a record
        if(view==btnModify)
        {
            // Checking empty roll number
            if(editRollno.getText().toString().trim().length()==0)
            {
                showMessage("Error", "Please enter Rollno");
                return;
            }
            // Searching roll number
            Cursor c=db.rawQuery("SELECT * FROM student WHERE
                        rollno='"+editRollno.getText()+"'", null);
            if(c.moveToFirst())
            {
                // Modifying record if found
                db.execSQL("UPDATE student SET
name='"+editName.getText()+"',marks='"+editMarks.getText()+
                        "' WHERE
rollno='"+editRollno.getText()+"'");
                showMessage("Success", "Record Modified");
            }
            else
            {
                showMessage("Error", "Invalid Rollno");
            }
            clearText();
        }
        // Viewing a record
        if(view==btnView)
        {
            // Checking empty roll number
            if(editRollno.getText().toString().trim().length()==0)
            {
                showMessage("Error", "Please enter Rollno");
                return;
            }
            // Searching roll number
            Cursor c=db.rawQuery("SELECT * FROM student WHERE
                        rollno='"+editRollno.getText()+"'", null);
            if(c.moveToFirst())
            {
                // Displaying record if found
                editName.setText(c.getString(1));
                editMarks.setText(c.getString(2));
            }
            else
            {
                showMessage("Error", "Invalid Rollno");
```

```
                clearText();
            }
        }
        // Viewing all records
        if(view==btnViewAll)
        {
            // Retrieving all records
            Cursor c=db.rawQuery("SELECT * FROM student", null);
            // Checking if no records found
            if(c.getCount()==0)
            {
                showMessage("Error", "No records found");
                return;
            }
            // Appending records to a string buffer
            StringBuffer buffer=new StringBuffer();
            while(c.moveToNext())
            {
                buffer.append("Rollno: "+c.getString(0)+"\n");
                buffer.append("Name: "+c.getString(1)+"\n");
                buffer.append("Marks: "+c.getString(2)+"\n\n");
            }
            // Displaying all records
            showMessage("Student Details", buffer.toString());
        }
        // Displaying info
        if(view==btnShowInfo)
        {
            showMessage("Student Management Application",
"Developed By BK");
        }
}
```

We explain main parts of the previous method:

Adding student data:

```
db.execSQL("INSERT INTO student
VALUES('"+editRollno.getText()+"','"+editName.getText()+
        "','"+editMarks.getText()+"');");
```

Note the use of the `execSQL` to execute the SQL statement.

Delete student:

```
// Searching roll number
Cursor c=db.rawQuery("SELECT * FROM student WHERE
rollno='"+editRollno.getText()+"'", null);
if(c.moveToFirst())
{
    // Deleting record if found
    db.execSQL("DELETE FROM student WHERE
                        rollno='"+editRollno.getText()+"'");
    showMessage("Success", "Record Deleted");
}
else
{
    showMessage("Error", "Invalid Rollno");
}
clearText();
```

Note that we start first by searching for the student that has the entered `rollno`.

The method `rawQuery` is used to execute a query that returns some records. This method returns a cursor on the retrieved records.

The method `moveToFirst()` moves the cursor to the first record and returns true if some records are found, else it returns false.

An error message is shown if there is no student with the specified `rollno`, else the specified student is deleted using the following SQL statement:

```
db.execSQL("DELETE FROM student WHERE
rollno='"+editRollno.getText()+"'");
```

Updating student data:

```
Cursor c=db.rawQuery("SELECT * FROM student WHERE
              rollno='"+editRollno.getText()+"'", null);
if(c.moveToFirst())
{
    // Modifying record if found
    db.execSQL("UPDATE student SET
name='"+editName.getText()+"',marks='"+editMarks.getText()+
            "' WHERE rollno='"+editRollno.getText()+"'");
    showMessage("Success", "Record Modified");
}
else
{
    showMessage("Error", "Invalid Rollno");
}
```

We look for the student that has the entered `rollno`. If there is no student with the specified `rollno`, an appropriate message is shown, else we update student data (name and marks) using the following statement:

```
db.execSQL("UPDATE                    student                    SET
name='"+editName.getText()+"',marks='"+editMarks.getText()+
            "' WHERE rollno='"+editRollno.getText()+"'");
```

Preview student data:

```
Cursor c=db.rawQuery("SELECT * FROM student WHERE
rollno='"+editRollno.getText()+"'", null);
if(c.moveToFirst())
{
    // Displaying record if found
    editName.setText(c.getString(1));
    editMarks.setText(c.getString(2));
}
else
{
    showMessage("Error", "Invalid Rollno");
    clearText();
}
```

We use the `rawQuery` method with an SQL statement that queries the student according to the entered `rollno`. This method returns a cursor on the returned records. If the result of the query does not return any record, the `moveToFirst()` method returns false.

Note that we use the `getString` method to access the value of the desired column (the parameter of the method is the column number in the table).

Display all students data:

```
// Retrieving all records
Cursor c=db.rawQuery("SELECT * FROM student", null);
// Checking if no records found
if(c.getCount()==0)
{
    showMessage("Error", "No records found");
    return;
}
// Appending records to a string buffer
StringBuffer buffer=new StringBuffer();
while(c.moveToNext())
{
    buffer.append("Rollno: "+c.getString(0)+"\n");
    buffer.append("Name: "+c.getString(1)+"\n");
    buffer.append("Marks: "+c.getString(2)+"\n\n");
}
// Displaying all records
showMessage("Student Details", buffer.toString());
```

We first use the `rawQuery` method to query all students and return a cursor on the returned records.

Note that the `getCount()` method returns the number of retuned records.
To iterate over the returned records, we use the `moveToNext()` method which returns false when it reaches the end of the records.

The following method is used to display alert messages:

```
public void showMessage(String title,String message)
{
    Builder builder=new Builder(this);
    builder.setCancelable(true);
    builder.setTitle(title);
    builder.setMessage(message);
    builder.show();
}
```

The following method is used to clear text views:

```
public void clearText()
{
    editRollno.setText("");
    editName.setText("");
    editMarks.setText("");
    editRollno.requestFocus();
}
```

Exercise: Dictionary:

You are requested to build an interface to look for the translation of English words into Arabic.

The user writes a word in English and the application shows its translation in Arabic.