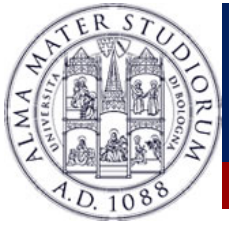
Programming with Android: Testing

Luca Bedogni

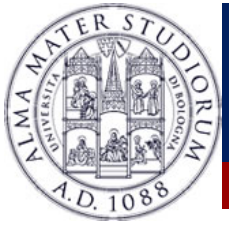
Dipartimento di Scienze dell'Informazione

Università di Bologna



Outline

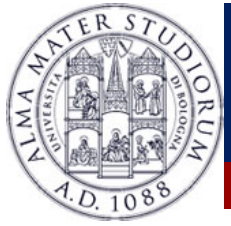
- Why **test**?
- Testing Android APPs: **Monkey**
- Testing Android APPs: **Monkeyrunner**
- Android Profiling: **CPU**
- Android Profiling: **Memory**
- Inspect layout with **LayoutInspector**
- Perform tests with **Espresso**



Where we are right now

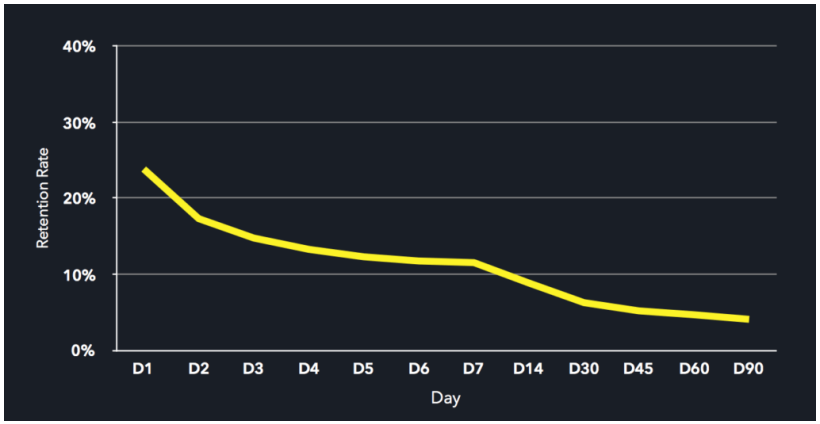
- We know how **Android is built**
- We know basic components such as **Activities**
- We know how to interact between Activities (**Intents**)
- We know how to handle **View events**
- We know how to place elements (**Layouts**)

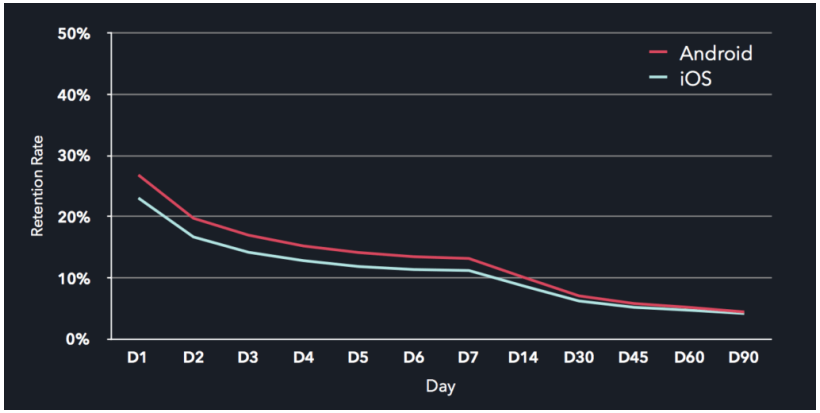
- We are ready to develop Android applications



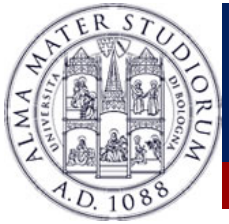
Retention rate

- Less than 25% of APP users return after the first day of use.
- Usually, **mobile gaming** experiences the **highest retention rate**
- **Social** APPs perform better on iOS
- Food and Beverages APPs experience a “weekly” retention rate
- Top 10 apps **lose 49%** of customers after 90 days
  - Top 5000 lose 91%
  - Average APPs lose 95%



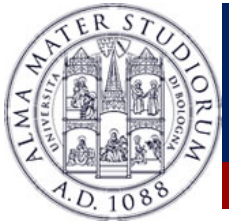


Source: Braze report



Who tests and what?

- Only 29% of Mobile developers do exploratory testing
- 67% of customers quit because of bad experiences
- Only 4% of unhappy customer complain
- Testing is expensive and time consuming
  - But ensures optimum performance
  - Stability of application
  - Reduces time and cost to market the application
  - Raises level of user experience



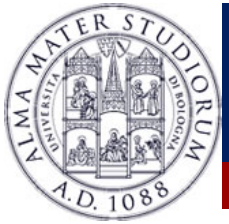
Testing with?

Real Device

- Have all the quirks present in real client hardware
- Hardware exception handling is possible
- Very expensive

Emulator

- Easier to manage
- Cost effective
- Do not have real faults



Different kind of testing

Unit Testing
- Test small pieces of your APP
- Each unit is tested separately from the others

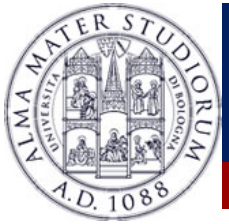
Integration
- To integrate single units together

Functionality
- Behave like black boxes
- Starting from inputs, check whether the outputs are those expected

Performance
- Evaluated in terms of response time and desired performance levels
- Responsiveness and stability
- Check whether battery, network, CPU, other applications affect your APP

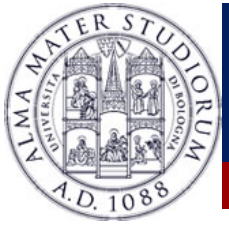
Stress
- To check APP behavior beyond normal usage levels

Usability
- Better to have thinner screens that perform well
- Instead of Bulky ones with lots of functionalities
- Check for different icon/images/text sizes



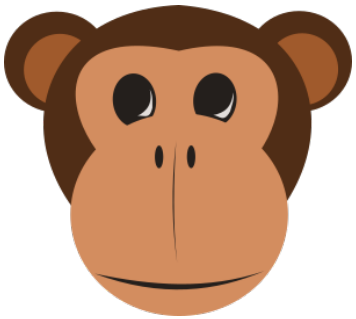
Testing the App

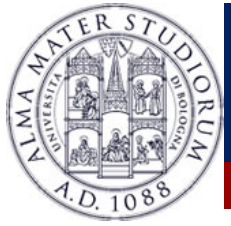
- Check for bugs
  - Test automation
- Profile the APP
  - To test for slow code
- Android provides several tools
  - Monkey
  - APP Profiler
  - LayoutInspector



How to test?

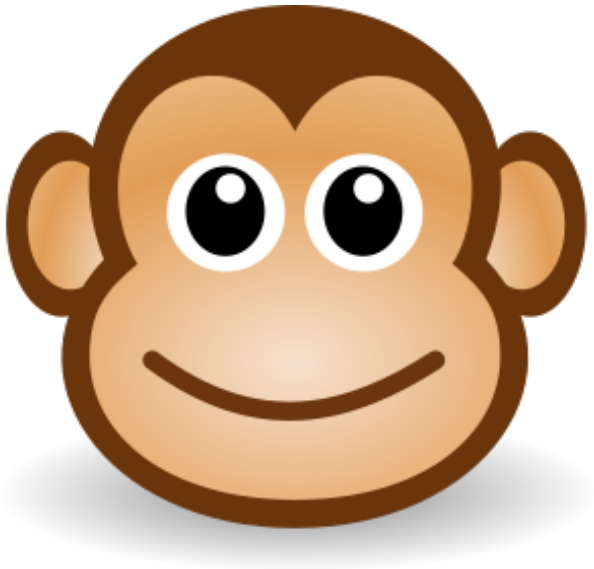
- Different smartphones, different possibilities
- Do it yourself: generate events on your application, see how it reacts.
  - Touch events, gestures
- Events can also come from the system
  - Calls, sms, notifications
- How to handle all possible events?
- How to repeat tests?
- Long and repetitive task, work for a monkey…

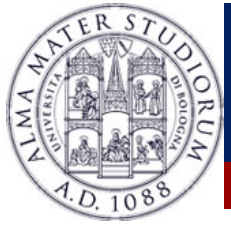




… so use a Monkey!

- The Monkey is a command line tool
  - Can run on the emulator or on the device
  - Sends events to the device
- Has several options
  - Basic options
  - Constraints
  - Kind of events and frequency



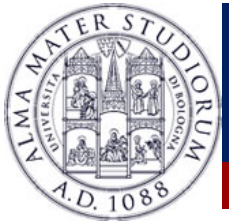


When the Monkey runs

- It sends events to the device
- And monitors it
  - To cope with constraints
  - To check for errors
  - To check for APP related blocking events
- Basic usage:

```
adb shell monkey -p my.package -v 500
```

  - Meaning: run the monkey on my.package generating 500 events



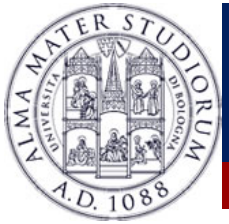
Monkey options: events and constraints

Events

Option	Meaning
-v	Verbosity level. Each v on the command line increases the level
-s	Seed. Use it to reproduce events
--throttle	Delay after events
--pct-{motion,trackball,touch,nav,majornav,syskeys,appswitch,anyevent}	Adjust the percentage of the specified event

Constraints

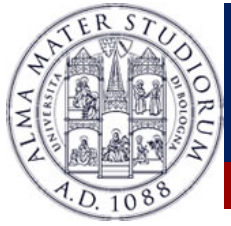
Option	Meaning
-p	Package or packages allowed to be visited.
-c	Category allowed to be visited



Monkey options: debugging

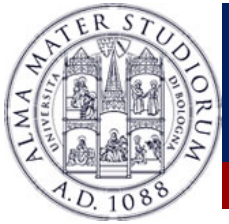
Debugging

Option	Meaning
--dbg-no-events	Only launch a test activity
--hprof	Generate profiling reports
--ignore-crashes	If something crashes, go on
--ignore-timeouts	If timeout, wait
--ignore-security-exceptions	If something requires a non granted permission, go on
--kill-process-after-error	If something crashes, then kill the process
--monitor-native-crashes	Watch and monitor system related crashes
--wat-dbg	Stop until a debugger is attached



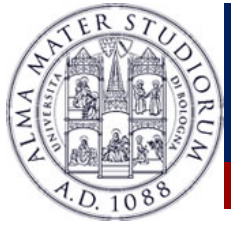
The monkey tool and monkeyrunner

- They are two different tools
  - The former runs inside adb
  - The latter may attach to multiple devices, and run specific tests
- The monkeyrunner runs a program written in Jython
- Can be extended with plugins



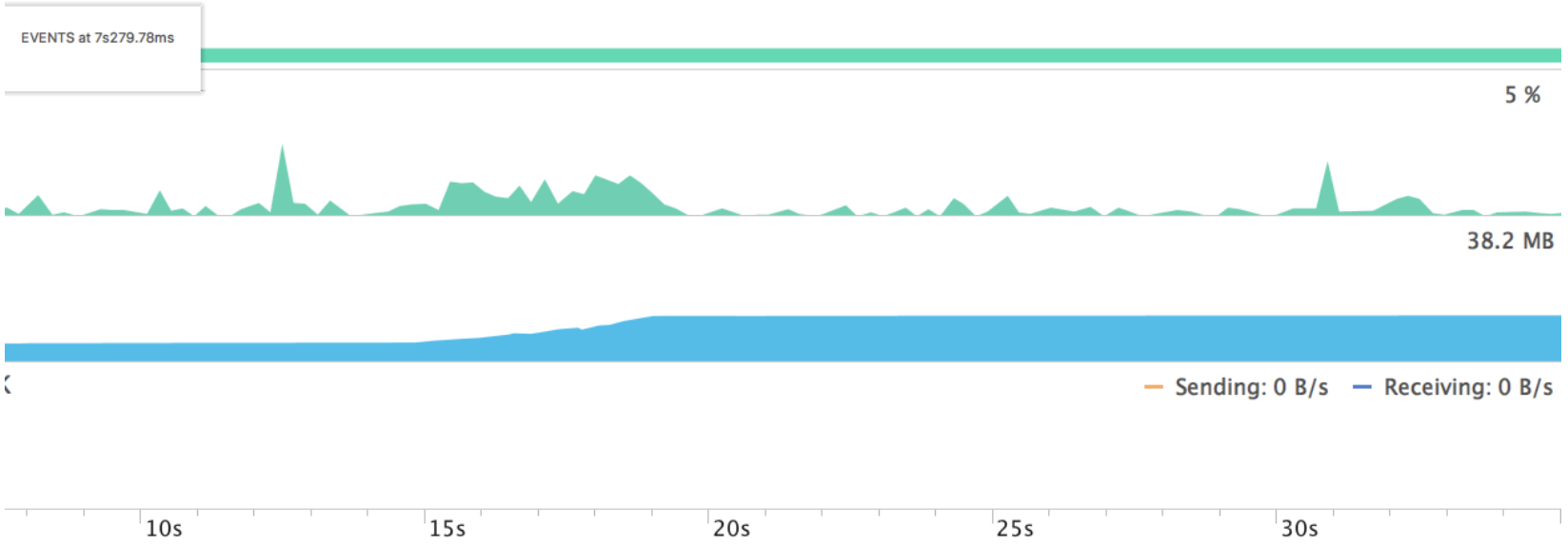
Monkeyrunner example

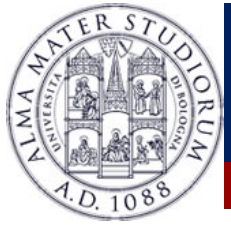
```
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice

device = MonkeyRunner.waitForConnection()

print "Launch WidgetExampleActivity”
device.startActivity(component='it.cs.android33/it.cs.android33.WidgetExampleActivity')

MonkeyRunner.sleep(1)
result = device.takeSnapshot()
result.writeToFile('screenshot.png','png')
print "Saved screenshot in screenshot.png”

device.touch(20,500,'DOWN_AND_UP')

MonkeyRunner.sleep(1)
result = device.takeSnapshot()
result.writeToFile('screenshot2.png','png')
print "Saved screenshot in screenshot2.png"
```



APP Profiling

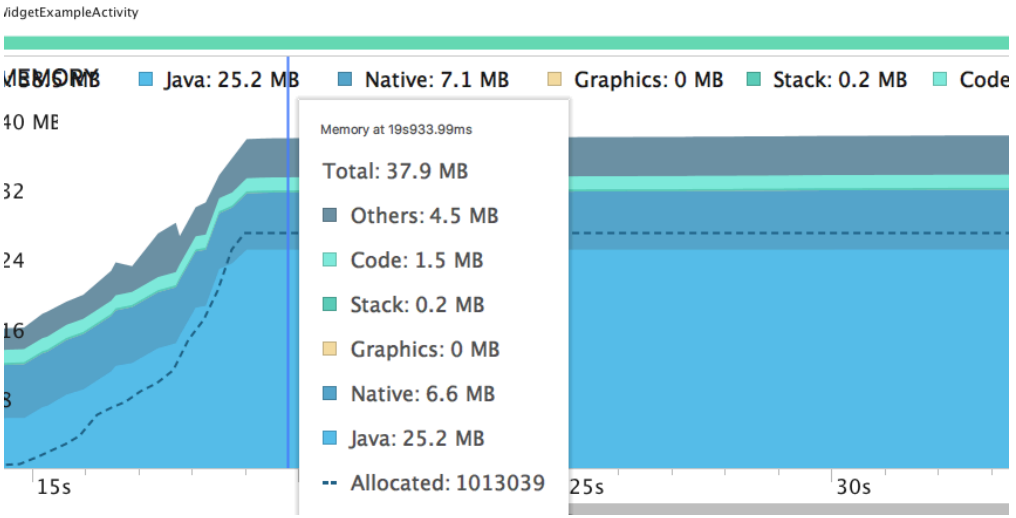
- Android provides several tools for APP monitoring
- One of them is the Android Profiler
  - Monitors CPU/MEMORY/NETWORK





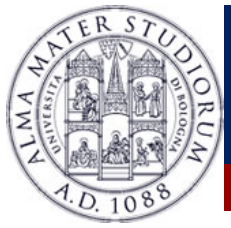
APP Profiling

- Can provide details on memory and CPU usage
- Use it to test parts of your code which slow down the APP

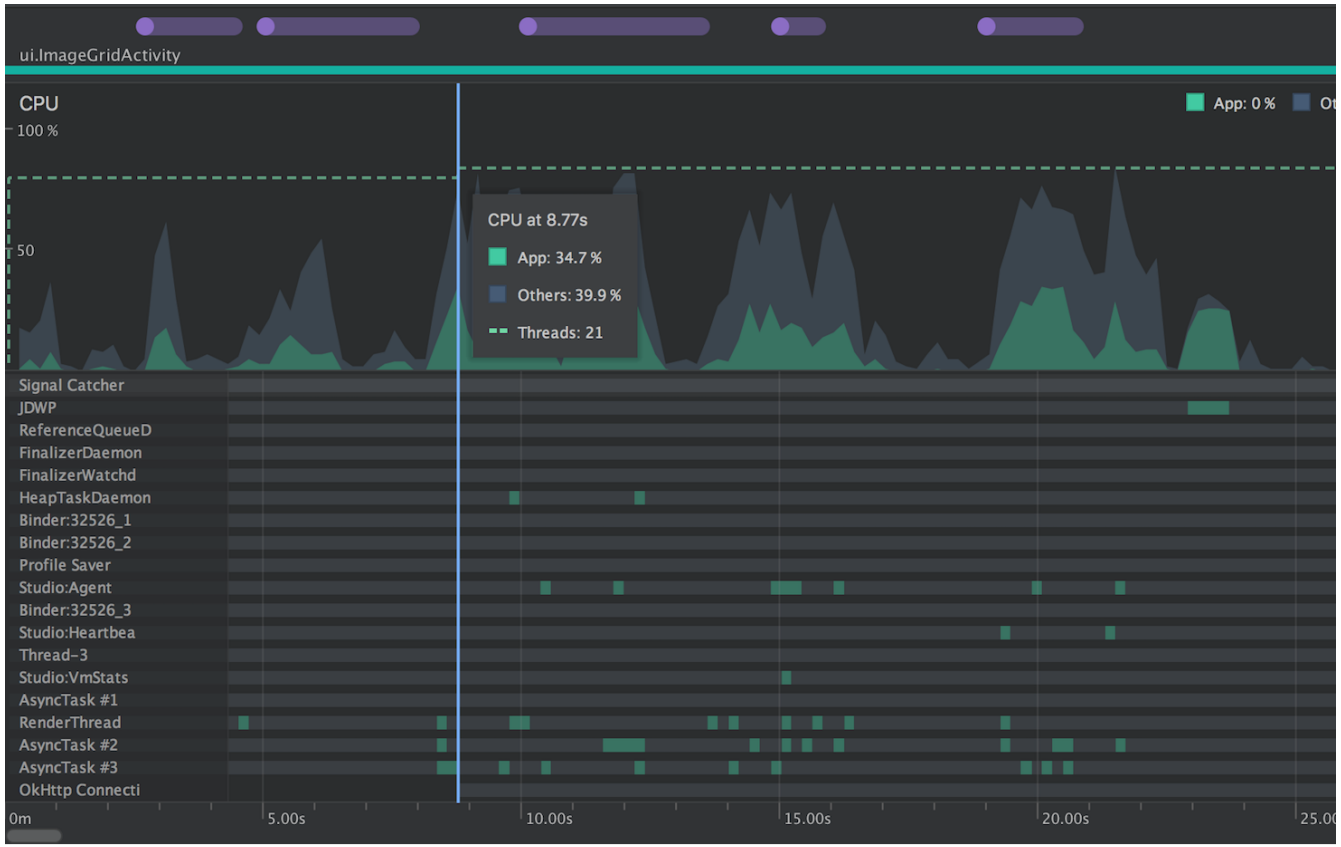


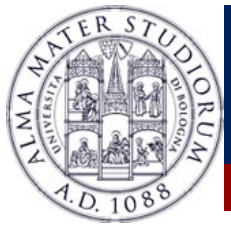
Recorded Allocations | default heap | Arrange by class | 00:09:23.210 – 00:09:28.005

Class Name	Allocations	Shallow Size
default heap	478	12192
char[]	55	2320
Class[] *(java.lang)*	84	1344
Constructor *(java.lang.reflect)*	84	1344
StringBuilder *(java.lang)*	40	1280
Field *(java.lang.reflect)*	56	896

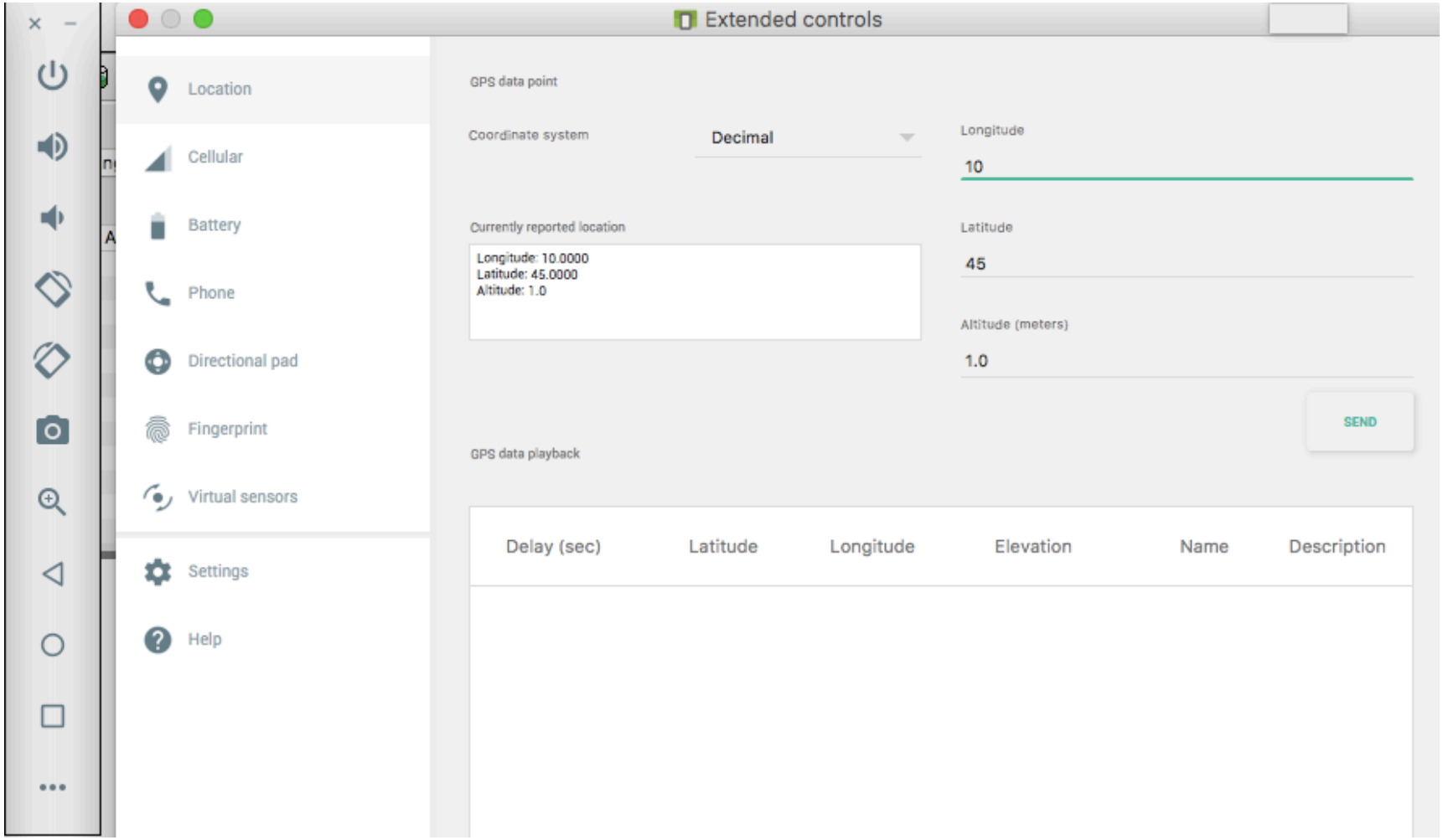


APP Profiling: CPU

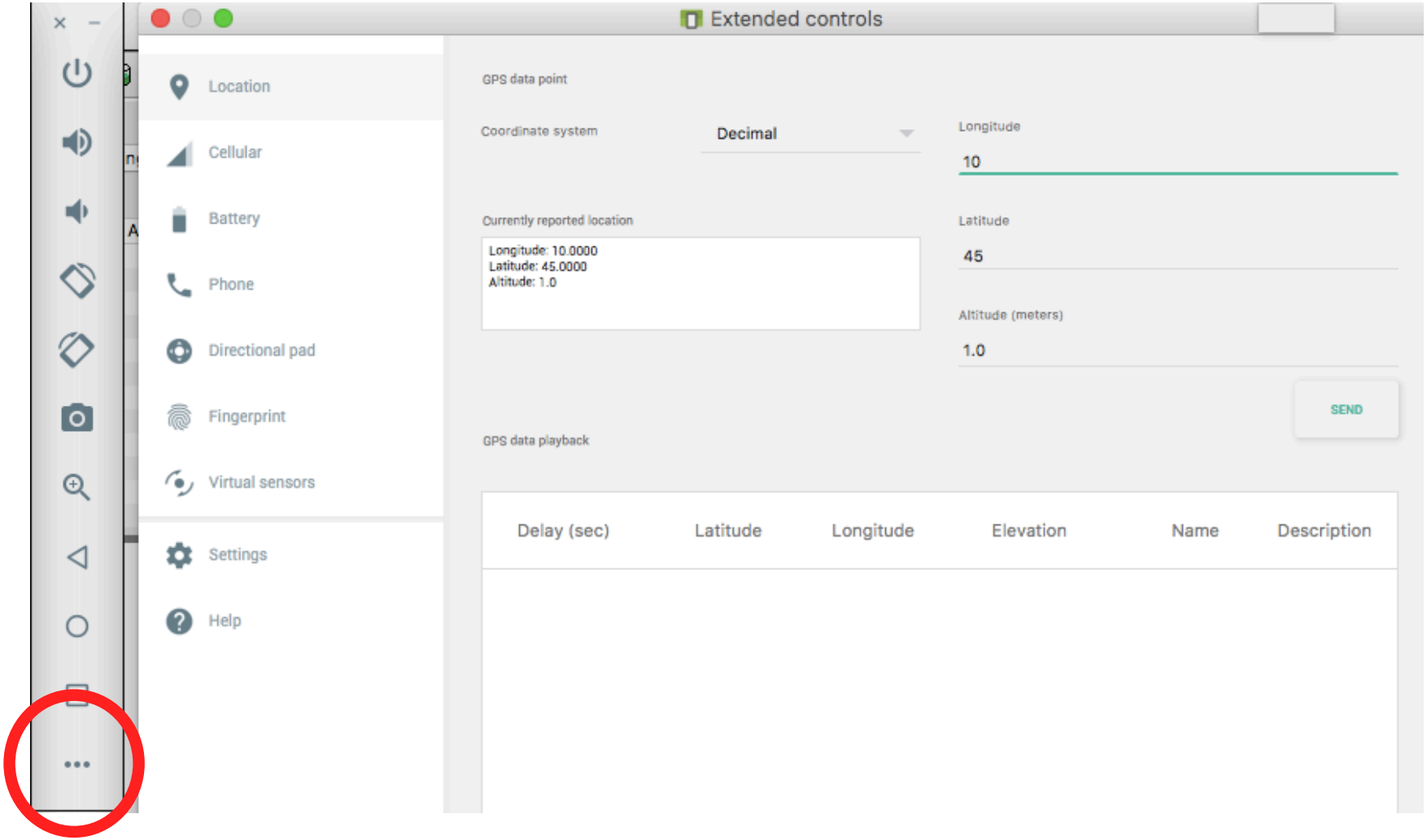


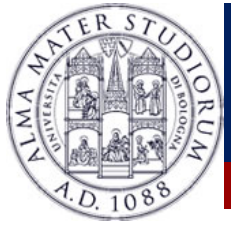


Device Control

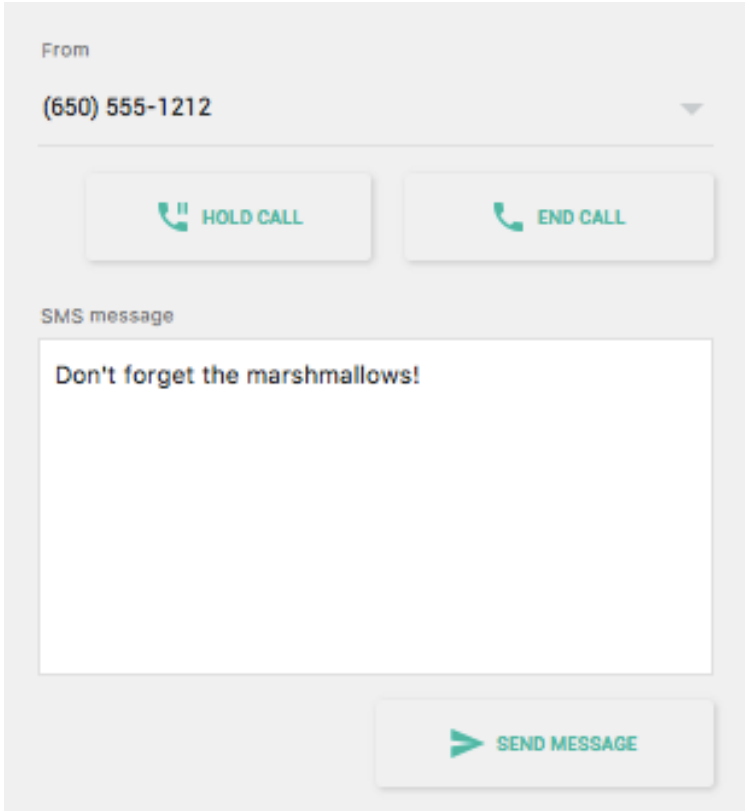


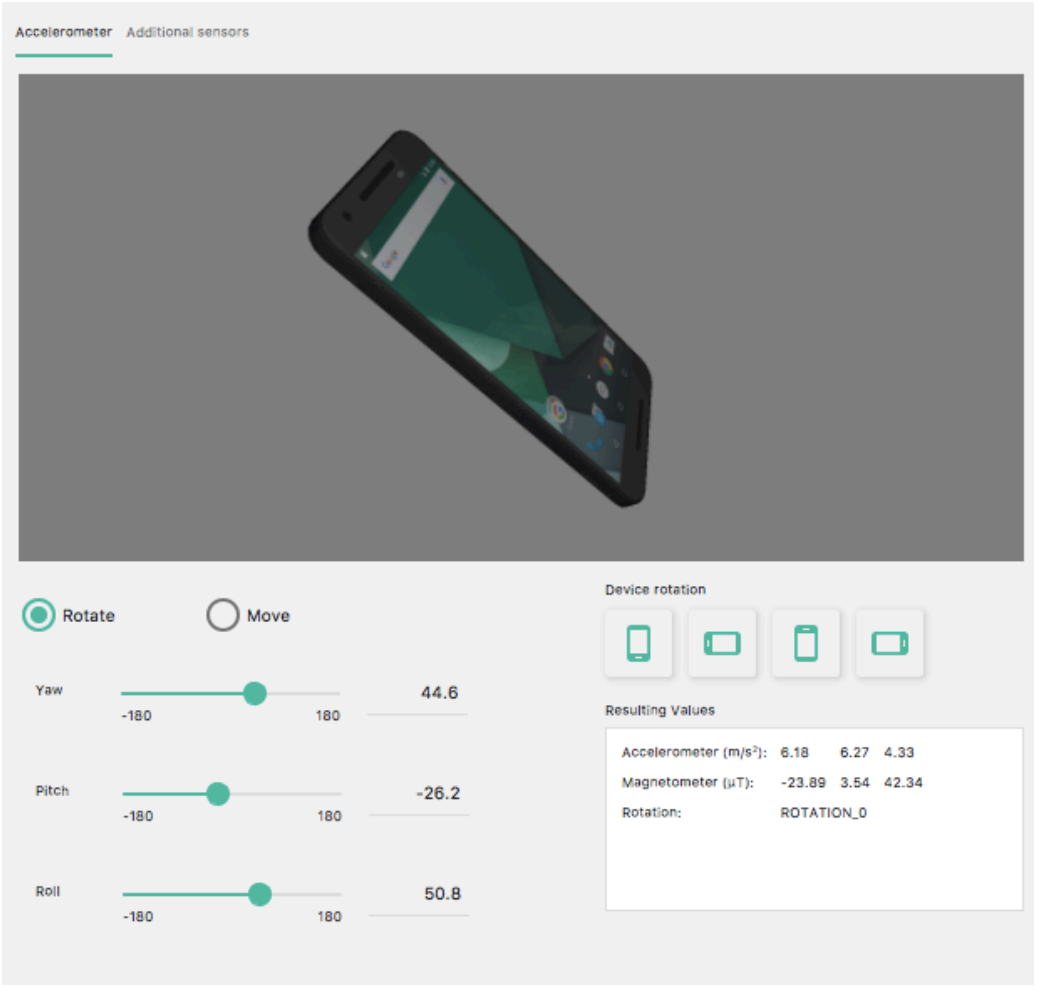
Device Control: Location

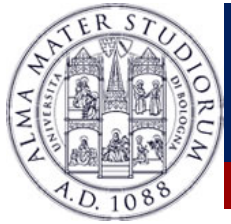




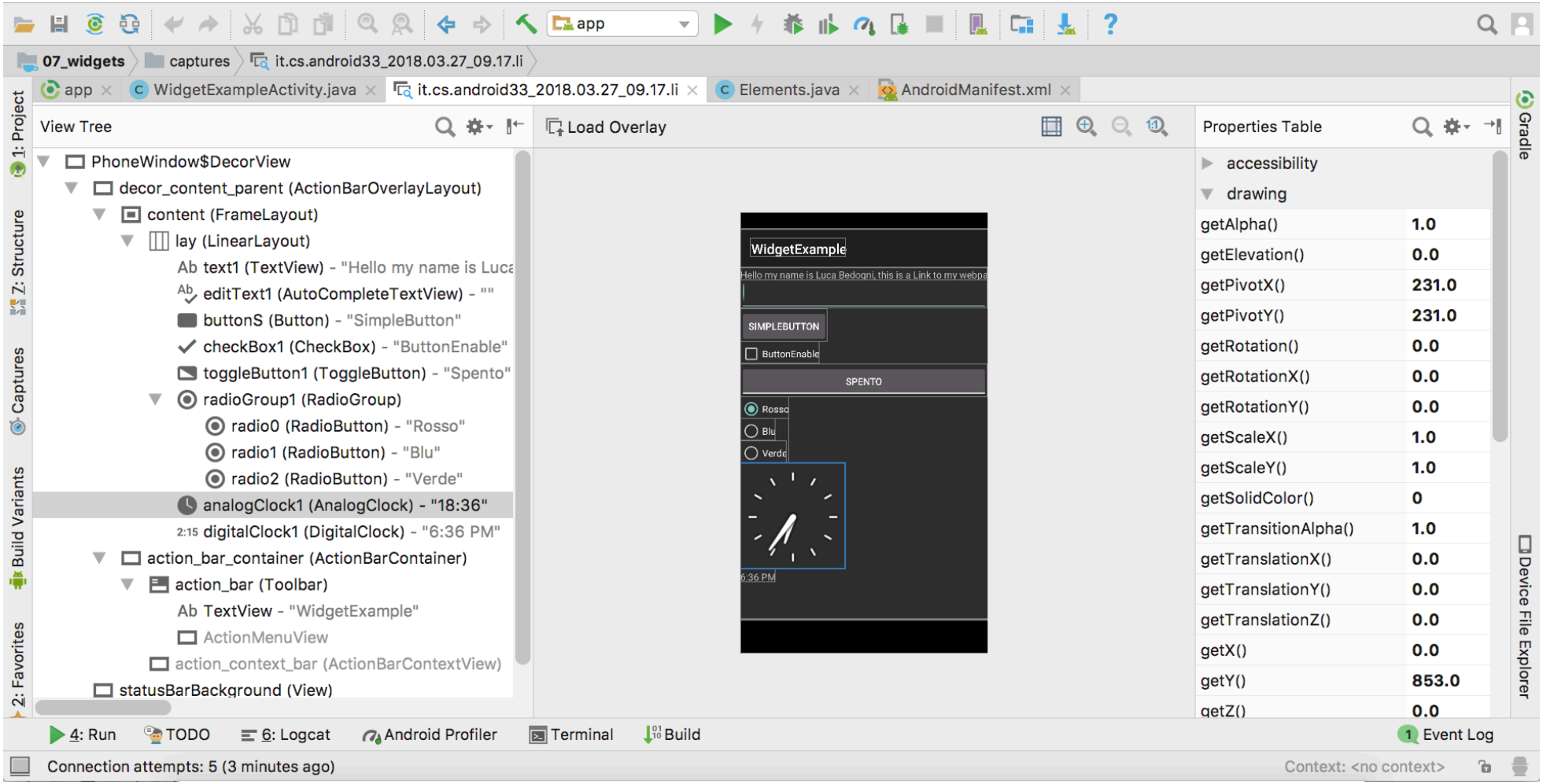
Device Control: Phone and Sensors

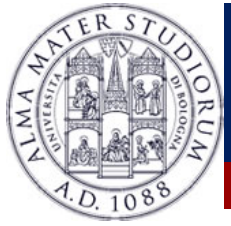






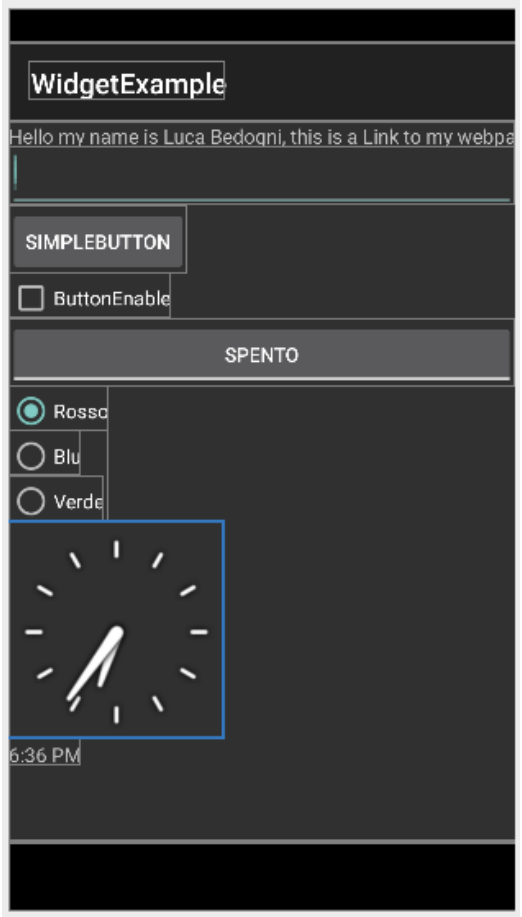
LayoutInspector

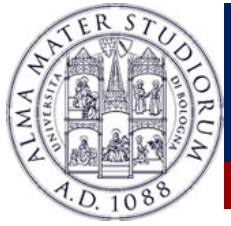




LayoutInspector

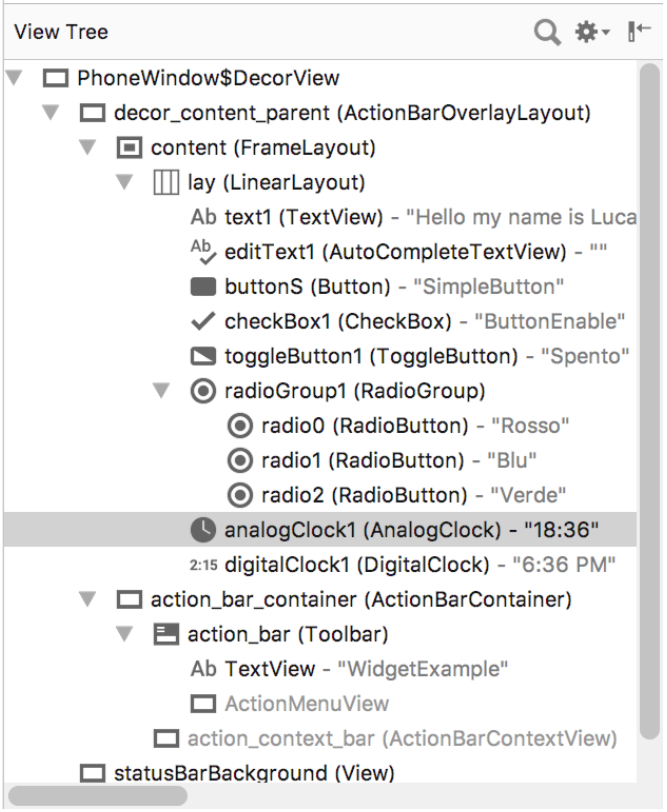
- Main Window:
  - Can see the whole layout
  - Each view is clearly separated from the others Double clicking on an item separates it from the whole picture
  - By clicking on items, you load specific attributes for such view

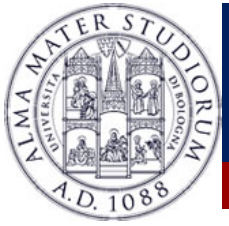




LayoutInspector

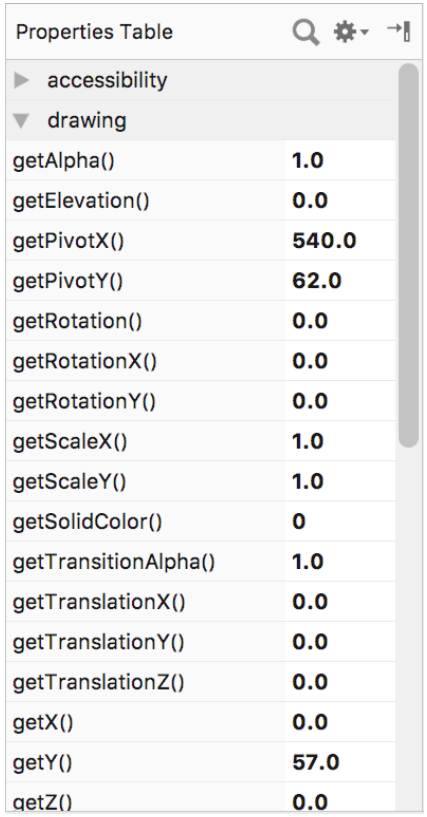
- On the left - view tree:
  - You get the whole hierarchy of the screen
  - Useful to understand how items are nested
  - For complex layout or small view, also used to select specific vies

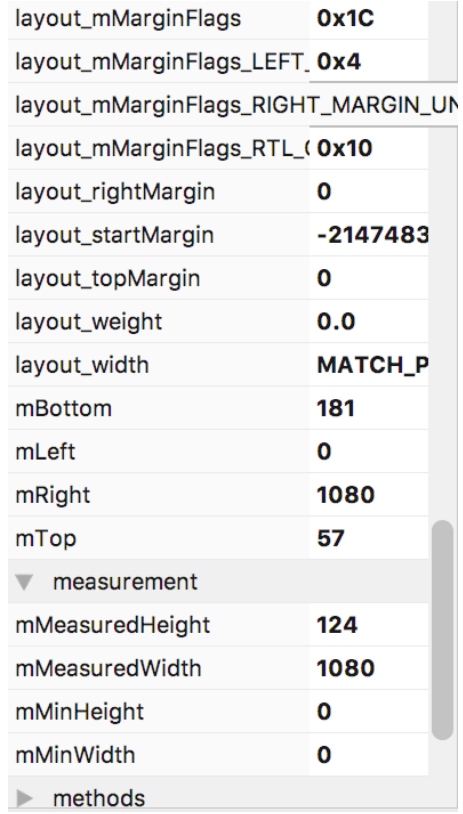


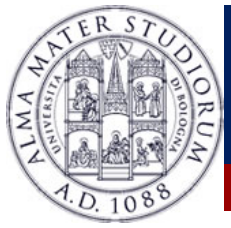


LayoutInspector

- On the right- properties:
  - When selecting a view, here we have the details
  - In case the layout is not seen as intended, used to understand which property is misbehaving





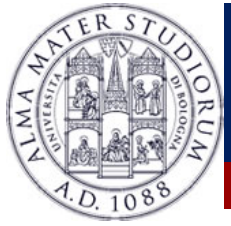


Espresso

- Espresso is needed for UI tests
- Idea:
  - Create a class with several methods
  - Each method represent a test
    - Can check for View contents, perform clicks etc.
  - Running the tests reports success or failure

» Example

```
onView(withId(R.id.my_view))
    .perform(click())
    .check(matches(isDisplayed()));
```



Espresso: how to configure it

- Add the following as dependencies

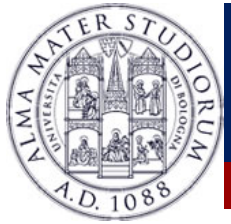
```
androidTestCompile 'com.android.support.test.espresso:espresso-core:3.0.1'
androidTestCompile 'com.android.support.test:runner:1.0.1'
```

- Add this in defaultConfig in build.gradle

```
testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
```

- Create a class in src/androidTest/java/my.package/

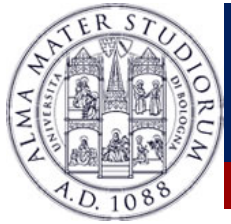
```
@RunWith(AndroidJUnit4.class)
@LargeTest
public class HelloWorldEspressoTest {

    @Rule
    public ActivityTestRule<MainActivity> mActivityRule =
            new ActivityTestRule(MainActivity.class);

    @Test
    public void listGoesOverTheFold() {
        onView(withText("Hello world!")).check(matches(isDisplayed()));
    }
}
```



The 4 Espresso building blocks

- **Espresso**
  - Main Entry point, needed to interact with views and perform view-independent actions
- **ViewMatchers**
  - A set of components through which it is possible to match certain views.

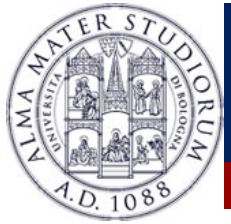
```
onView(withId(R.id.my_view))
```



The 4 Espresso building blocks

- **Espresso**
  - Main Entry point, needed to interact with views and perform view-independent actions
- **ViewMatchers**
  - A set of components through which it is possible to match certain views.
- **ViewActions**
  - A set of components to perform actions on views

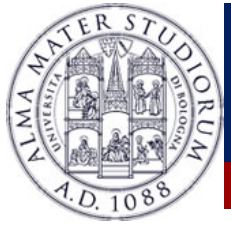
```
onView(withId(R.id.my_view))
    .perform(click())
```



The 4 Espresso building blocks

- **Espresso**
  - Main Entry point, needed to interact with views and perform view-independent actions
- **ViewMatchers**
  - A set of components through which it is possible to match certain views.
- **ViewActions**
  - A set of components to perform actions on views
- **ViewAssertions**
  - To check specific view properties

```
onView(withId(R.id.my_view))
    .perform(click())
    .check(matches(isDisplayed()));
```



The Espresso cheat sheet

```
onView(ViewMatcher)
    .perform(ViewAction)
    .check(ViewAssertion);
```

```
onData(ObjectMatcher)
    .DataOptions
    .perform(ViewAction)
    .check(ViewAssertion);
```

CHEAT SHEET espresso 2.1

View Matchers

USER PROPERTIES
```
withId(...)
withText(...)
withTagKey(...)
withTagValue(...)
hasContentDescription(...)
withContentDescription(...)
withHint(...)
withSpinnerText(...)
hasLinks()
hasEllipsizedText()
hasMultilineTest()
```

HIERARCHY
```
withParent(Matcher)
withChild(Matcher)
hasDescendant(Matcher)
isDescendantOfA(Matcher)
hasSibling(Matcher)
isRoot()
```

INPUT
```
supportsInputMethods(...)
hasIMEAction(...)
```

UI PROPERTIES
```
isDisplayed()
isCompletelyDisplayed()
isEnabled()
hasFocus()
isClickable()
isChecked()
isNotChecked()
withEffectiveVisibility(...)
isSelected()
```

CLASS
```
isAssignableFrom(...)
withClassName(...)
```

ROOT MATCHERS
```
isFocusable()
isTouchable()
isDialog()
withDecorView()
isPlatformPopup()
```

OBJECT MATCHER
```
allOf(Matchers)
anyOf(Matchers)
is(...)
not(...)
endsWith(String)
startsWith(String)
instanceOf(Class)
```

SEE ALSO
```
Preference matchers
Cursor matchers
Layout matchers
```

Data Options

```
inAdapterView(Matcher)
atPosition(Integer)
onChildView(Matcher)
```

View Actions

CLICK/PRESS
```
click()
doubleClick()
longClick()
pressBack()
pressIMEActionButton()
pressKey([int/EspressoKey])
pressMenuKey()
closeSoftKeyboard()
openLink()
```

GESTURES
```
scrollTo()
swipeLeft()
swipeRight()
swipeUp()
swipeDown()
```

TEXT
```
clearText()
typeText(String)
typeTextIntoFocusedView(String)
replaceText(String)
```

View Assertions

```
matches(Matcher)
doesNotExist()
selectedDescendantsMatch(...)
```

LAYOUT ASSERTIONS
```
noEllipseizedText(Matcher)
noMultilineButtons()
noOverlaps([Matcher])
```

POSITION ASSERTIONS
```
isLeftOf(Matcher)
isRightOf(Matcher)
isLeftAlignedWith(Matcher)
isRightAlignedWith(Matcher)
isAbove(Matcher)
isBelow(Matcher)
isBottomAlignedWith(Matcher)
isTopAlignedWith(Matcher)
```

Available at:
https://developer.android.com/training/testing/espresso/cheat-sheet.html

```
intended(IntentMatcher);
```

```
intending(IntentMatcher)
    .respondWith(ActivityResult);
```

Intent Matchers

INTENT
```
hasAction(...)
hasCategories(...)
hasData(...)
hasComponent(...)
hasExtra(...)
hasExtras(Matcher)
hasExtraWithKey(...)
hasType(...)
hasPackage()
toPackage(String)
hasFlag(int)
hasFlags(...)
isInternal()
```

URI
```
hasHost(...)
hasParamWithName(...)
hasPath(...)
hasParamWithValue(...)
hasScheme(...)
hasSchemeSpecificPart(...)
```

BUNDLE
```
hasEntry(...)
hasKey(...)
hasValue(...)
```

COMPONENT NAME
```
hasClassName(...)
hasPackageName(...)
hasShortClassName(...)
hasMyPackageName()
```