10026 - Designing and Building for Android Devices

SHARE in Orlando
10 August, 2011

Jim Willette
Sunrise e-Services

Dimension One Outline

- Software Engineering and Agile
  - A “simple” project: a 4-function calculator
  - The simplest sub-project
    - 1 function
    - 2 numbers
    - 4 (or5) buttons and a display
    - RPN (may not be absolute simplest, but close)
  - Screen / platform limitations (and features)
    - Now tablets as well as phones
  - Iteration plans (more RUP than Agile, but should be)

Dimension Two Outline

- Android Nuts and Bolts
  - Resources and Layouts for the UI / UX
  - Java (a large subset)
    - Not teaching Java; that’s another show
    - A Java surprise
  - Android specific APIs and mechanisms
    - One Androidism that I still don’t “get”
  - Eclipse IDE with Android plugin plus …
    - Android emulators and adb
    - Pros and cons
- I will mix these two dimensions chaotically

The Project

- An actual assignment from an Android course
  - Touches on the UI (Forms, anyway)
  - Explores an interesting kind of Java math
  - Uses simple data structures (with some variation)
  - Testable results
  - Can be expanded in future project(s)
  - Will follow my “lightbulb” moments
  - Code for the sub-project will be available (TBD how)
  - Illustrates relative device independence

Why a Sub-project

- Limited time
- How I actually handled the problem
- Troubleshooting is easier
- Exposes Agile philosophy (we can discuss or not)
- Once you have this done, the rest is repetition
- Lets us talk about larger issues

A One-function Calculator

- The numbers: 3, 6 {arbitrary, but not entirely}
- The function: Division (could have been subtraction)
- The fourth button: “Enter” {for RPN.}
  - What is RPN?
  - Why RPN?
    - Simple hardware (originally) e.g., HP35
    - Some work done “in your head” and in advance
    - It is how compilers work {Ref: B5500}
    - Needs Stack, but no registers (well, maybe one)
- The Decimal Point button (that makes 5)
- An optional “Clear” button (may talk about but not implement)

Android *Resources*

- XML based
- Provides many of the widgets you expect
- Includes “Layouts”
  - Absolute
  - Relative
  - Linear
  - Grid
    - May mix these somewhat, but results may surprise you □
- Automatically compiled {the R object}
- Tools available, but not truly reflective
- Resource IDs (AKA handles)
- Eclipse partially understands, and warns

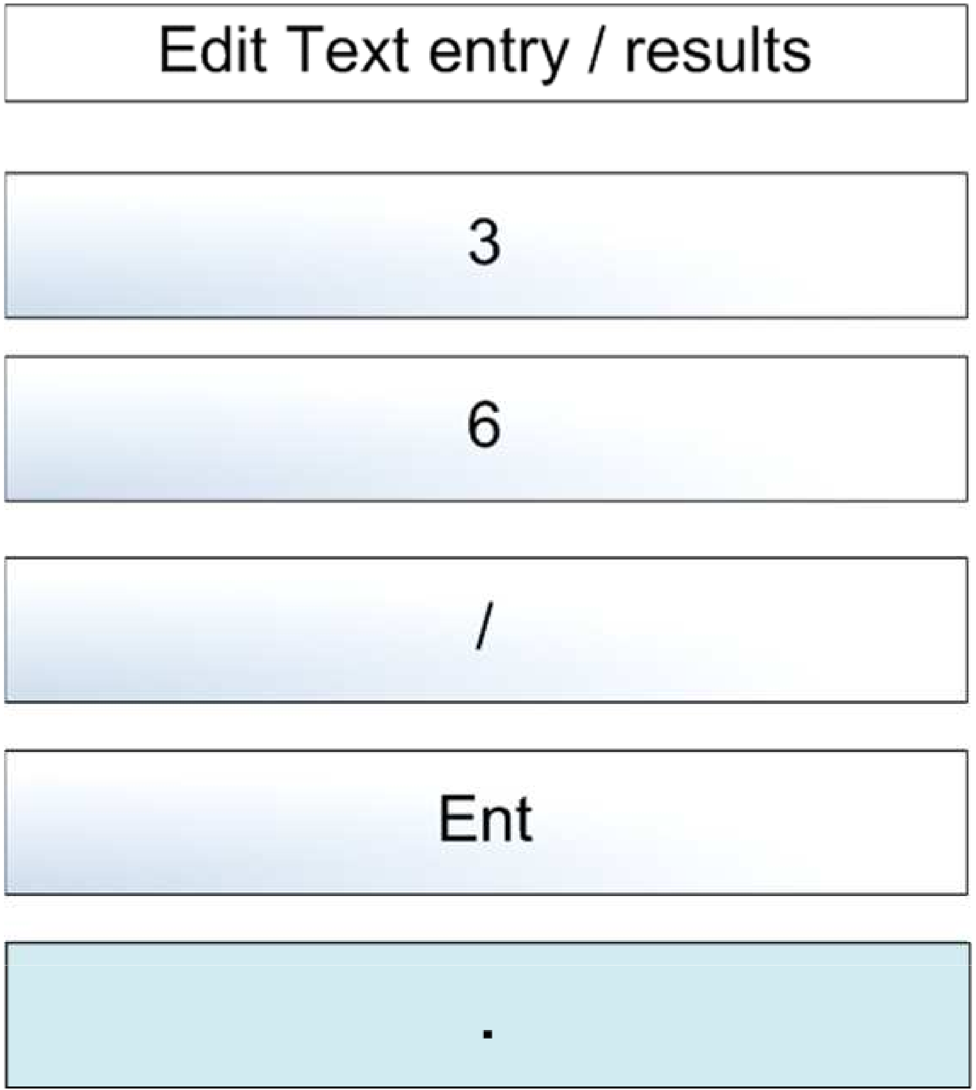
Resource Catalog

- Button
- Text View (captions)
- Edit Text
- Radio Button
  - Radio Group
- Check Box
- Table?
- Spinner
- Scrolling
- List View
- Media Controller
- Rating Bar (hybrid)
- Tab Host / Widget
- Search (this is from Google after all)
- Toast (like a bubble message)
- Zoom

The Screen (design)

- Simple layout
- Minimalist
- Not representative
- Not **exactly**

what you would see on Android



Edit Text entry / results

3

6

/

Ent

.

Calculator Resources 1

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:id="@+id/widget1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<EditText
    android:id="@+id/widget2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text=""
/>
<Button
    android:id="@+id/btn3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="3"
/>
```

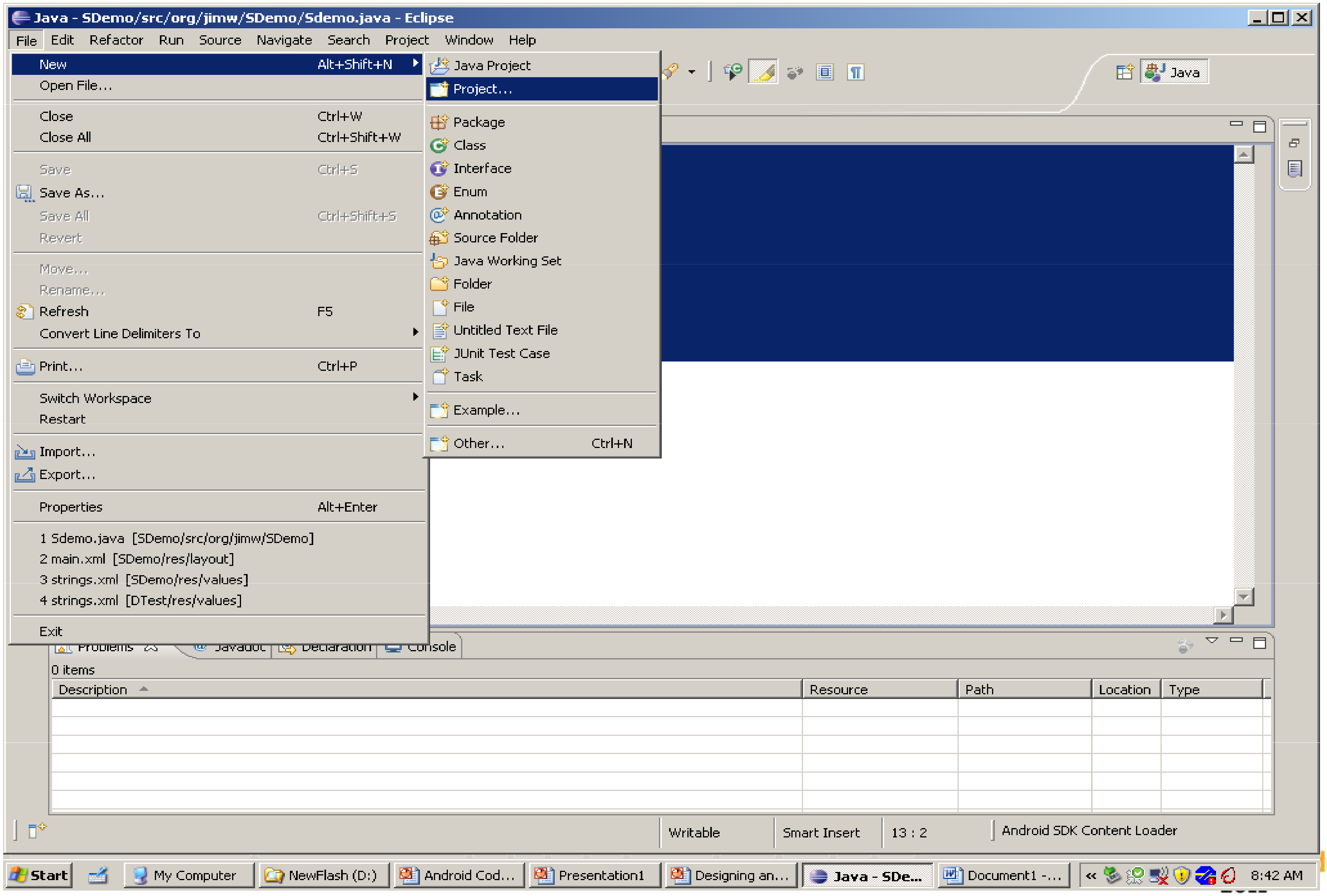
Calculator Resources 2

```
<Button
   android:id="@+id/btn6"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="6"
/>
<Button
   android:id="@+id/btndec"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="."
/>
<Button
   android:id="@+id/btnent"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Ent"
/>
<Button
   android:id="@+id/btndiv"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text=“/"
/>
</LinearLayout>
```

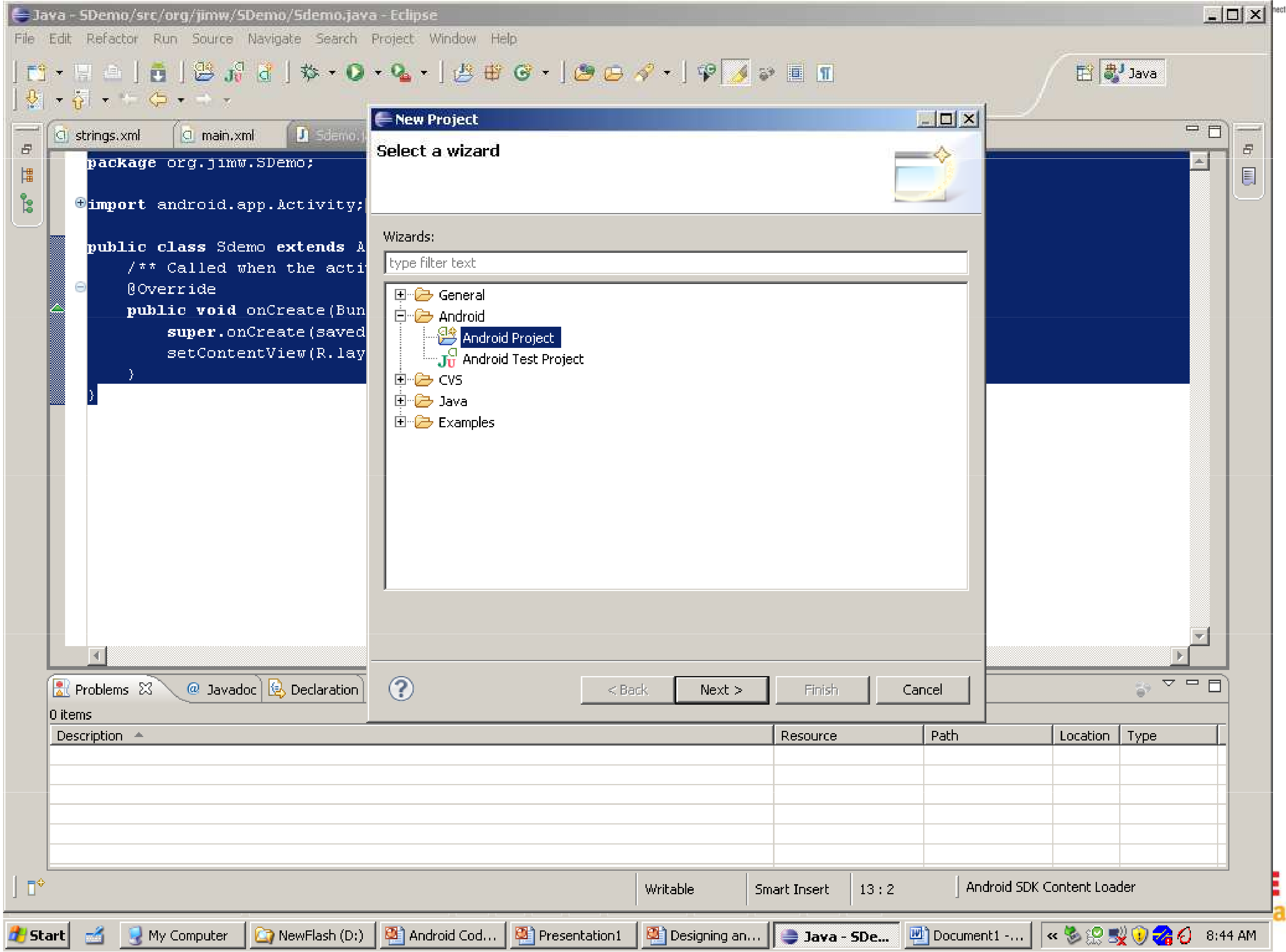
Android Automatic Project

- Builds a basic “activity” with “hello *app*” text resource and a layout with a text field (caption)
- Complete and runable, but vacuous
- Embellish and overlay for a real activity (app)
  - Add resources
  - Add code to display those resources
  - Add logic
    - Event handlers (for buttons in our case)
      - *Touch interface, as needed*

IDE at Startup



New Android Eclipse Project



```
package org.jimw.SDemo;

import android.app.Activity;

public class Sdemo extends A
    /** Called when the acti
    @Override
    public void onCreate(Bun
        super.onCreate(saved
        setContentView(R.lay
    }
}
```

New Project

Select a wizard

Wizards:

type filter text

- General
- Android
  - Android Project
  - Android Test Project
- CVS
- Java
- Examples

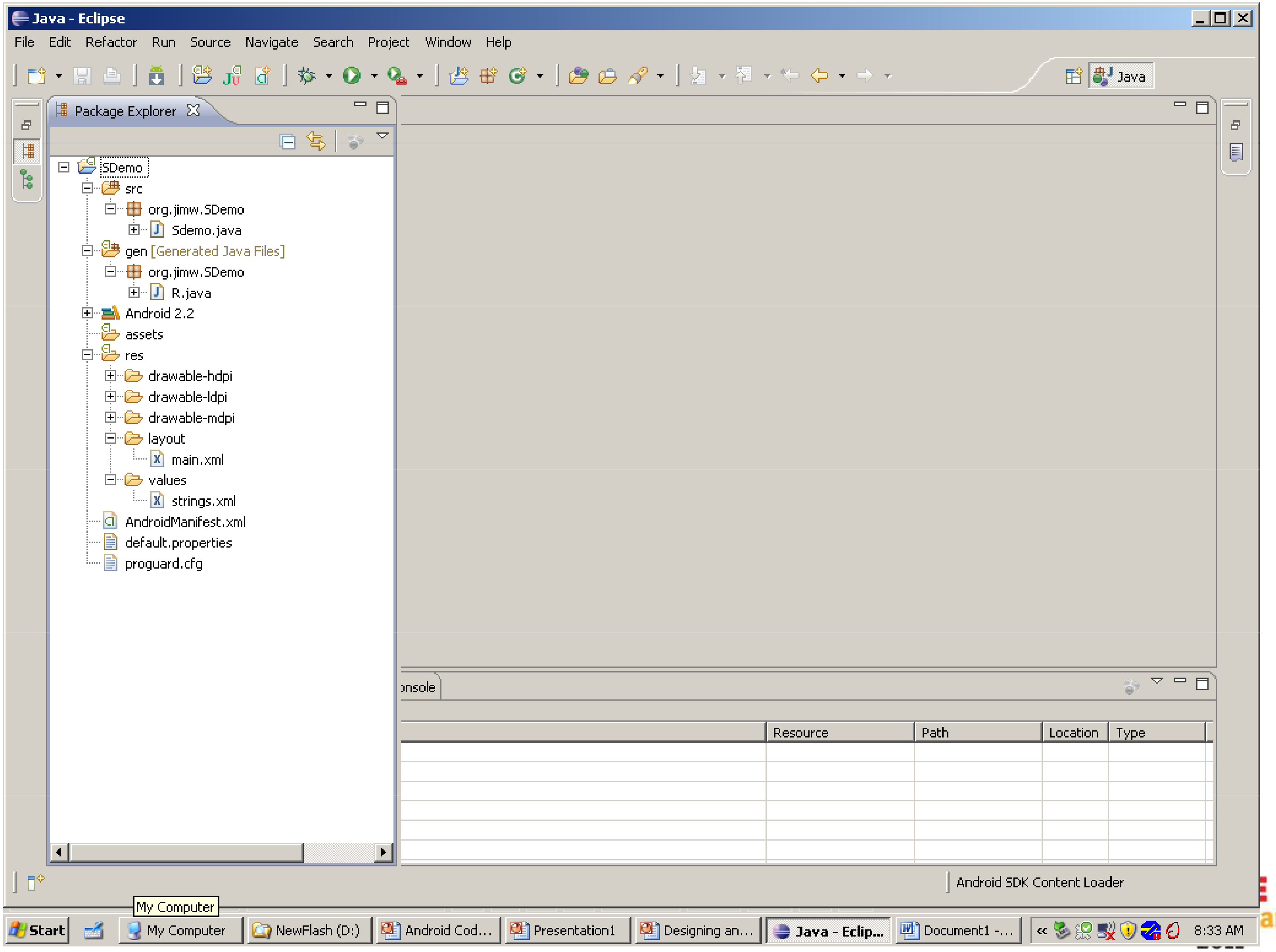
< Back | Next > | Finish | Cancel

0 items

Description	Resource	Path	Location	Type

Writable | Smart Insert | 13 : 2 | Android SDK Content Loader

Eclipse Workspace



```
SDemo
  src
    org.jimw.SDemo
      Sdemo.java
  gen [Generated Java Files]
    org.jimw.SDemo
      R.java
  Android 2.2
  assets
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    layout
      main.xml
    values
      strings.xml
  AndroidManifest.xml
  default.properties
  proguard.cfg
```

Automatic Resources

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Sdemo!</string>
    <string name="app_name">SHARE Demo</string>
</resources>
```

Automatic Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

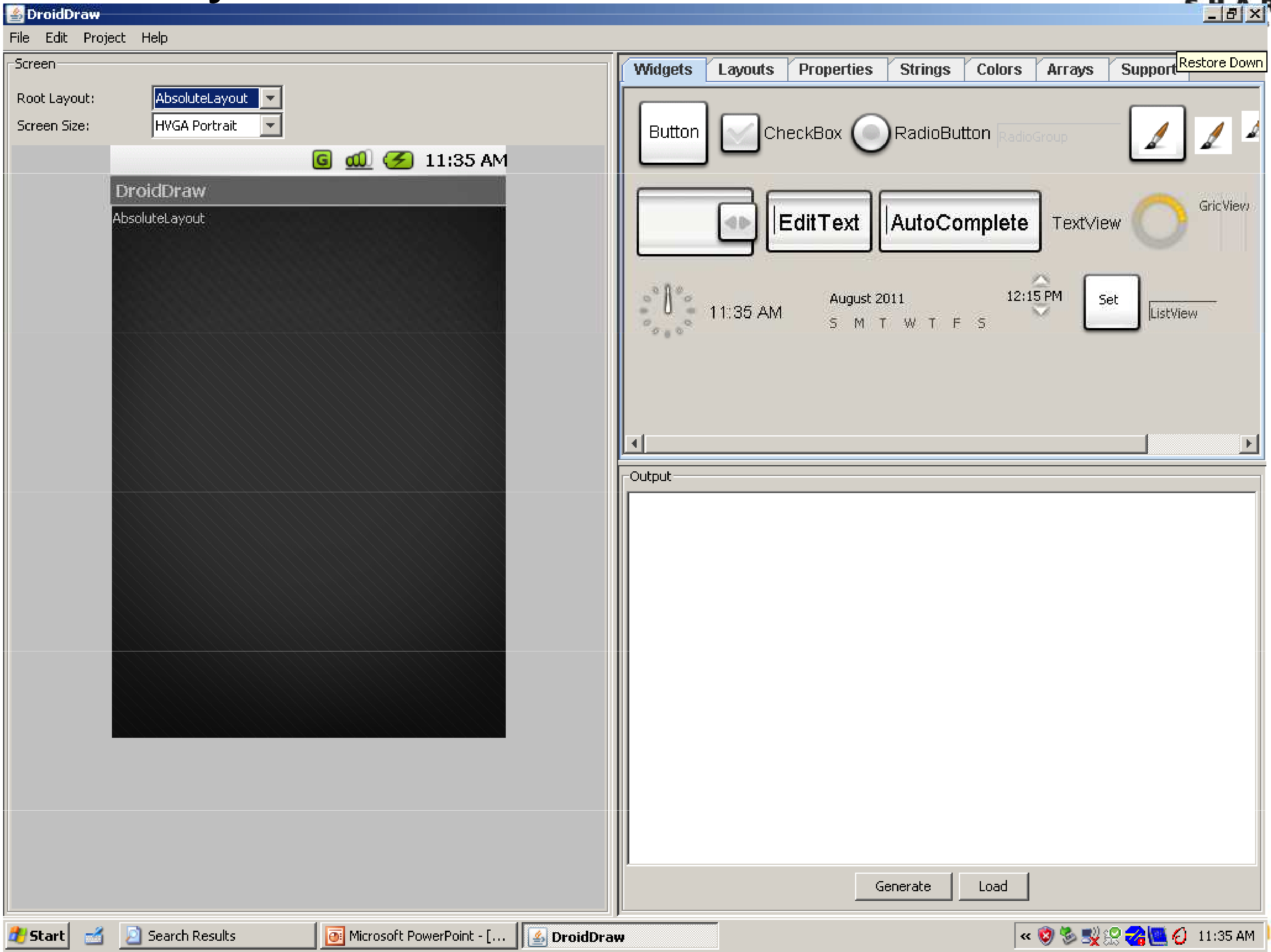
Automatic Code

```
package org.jimw.SDemo;
import android.app.Activity;
import android.os.Bundle;
public class Sdemo extends Activity
{
/** Called when the activity is first created. */
@Override    public void onCreate(Bundle savedInstanceState)
{
     super.onCreate(savedInstanceState);       setContentView(R.layout.main);
}
}
```

Making Life Easier

- Initially I had trouble creating a calculator layout
- The sub-project eliminated the need
- I still needed one – enter the tool “DroidDraw”
- Simple drag-and-drop had one together in no time, but it was not quite WYSIWYG (sigh)
  - Good enough; can fix later
    - “Good enough” is highest goal – “Perfect” is too good
  - Add IDs that match (or at least meaningful)
  - Copy to Layout page
  - Consider donating to author

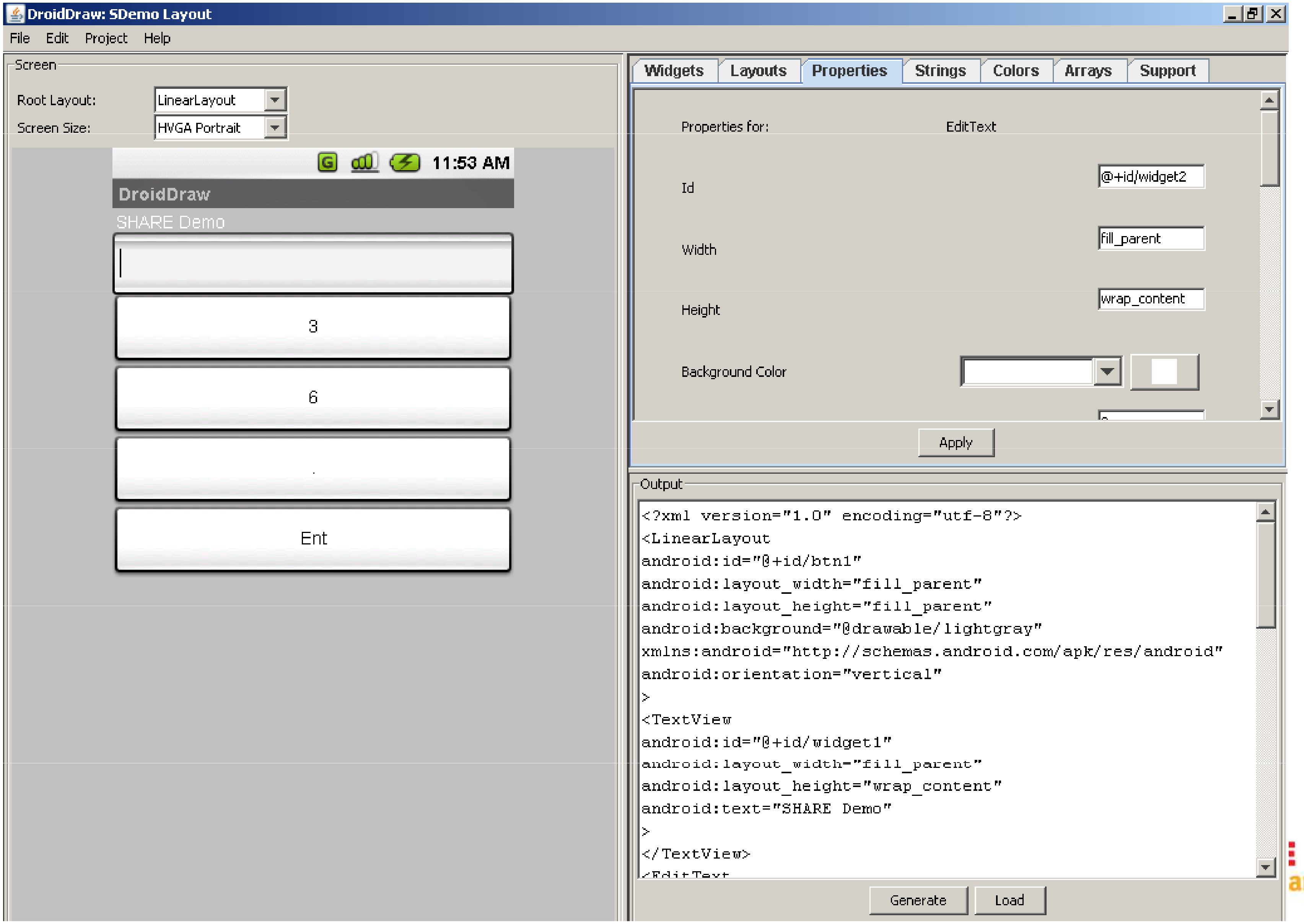
Droid Draw Layout



Root Layout: AbsoluteLayout

Screen Size: HVGA Portrait

Droid Draw Demo Layout



Root Layout: LinearLayout

Screen Size: HVGA Portrait

Properties for: EditText

Id: @+id/widget2

Width: fill_parent

Height: wrap_content

Background Color:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
android:id="@+id/btn1"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="@drawable/lightgray"
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
>
<TextView
android:id="@+id/widget1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="SHARE Demo"
>
</TextView>
<EditText
```

Layout Code

```
package org.jimw.SDemo;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

import java.math.BigDecimal;
import java.util.*;

public class SDemo extends Activity
{
    private EditText showTxt;
    private Button btn3;
    private Button btn6;
    private Button btndec;
    private Button btnent;
    String accum = new String("");
    Stack stk = new Stack();
    BigDecimal arg1;
    BigDecimal arg2;
    boolean ent = Boolean.TRUE;
}
```

Layout Code with Listeners

```
private void loadControls()
{
showTxt = (EditText) findViewById(R.id.widget2);
btn6 = (Button) findViewById(R.id.btn6);
btn3 = (Button) findViewById(R.id.btn3);
btnent = (Button) findViewById(R.id.btnent);
btndiv = (Button) findViewById(R.id.btndiv);
btn6.setOnClickListener(new Button.OnClickListener() { public void onClick (View v) {
    doNum(6); }});
btn3.setOnClickListener(new Button.OnClickListener() { public void onClick (View v) {
    doNum(3); }});
btnent.setOnClickListener(new Button.OnClickListener() { public void onClick (View v) {
    doEnt(); }});
btndiv.setOnClickListener(new Button.OnClickListener() { public void onClick (View v) {
    doDiv(); }});
}
```

Doing the Math

- Since we are doing Division, we need more than integers
  - Integer math fails silently when the largest value is exceeded (found this out when doing large Fibonacci series.)
- Floating point may not be enough
  - The requirement for number of digits was not specified
- Surprise: (Java.math) BigDecimal – arbitrarily large decimal numbers (not floating point, more like packed decimal mainframe arithmetic)

Doing the Arithmetic

- RPN = “Reverse Polish Notation” also called “Postfix”
- To divide two numbers:
  - Key in number (dividend)
  - Hit “Enter”
  - Key in number (divisor)
  - Hit “/” (divide)
  - The answer (quotient) appears
- What else
  - Only one decimal point per entry
  - New entry after each function button pressed

Test Driven Design (TDD)

- Simplest success:
  - Input a decimal number
- Simplest operation:
  - Divide two decimal numbers and …
  - Display the result
- That’s it!
- Simple failures:
  - Divide by zero (Display “Error” message)
  - Enter a second decimal point (silently ignore)
  - Divide without a second number (“Error” again)
  - Clearing the entry field without destroying calculations would be really nice here (next feature)

Button Listeners

```
private void doNum(int num)
{
if (ent)
accum = "";
accum = accum + num;
showTxt.setText(accum);
ent = Boolean.FALSE;
}

private void doDiv()
{
arg2 = new BigDecimal(showTxt.getText().toString());
arg1 = (BigDecimal)stk.pop();
arg1 = arg1.divide(arg2);
showTxt.setText(arg1.toString());
}
```

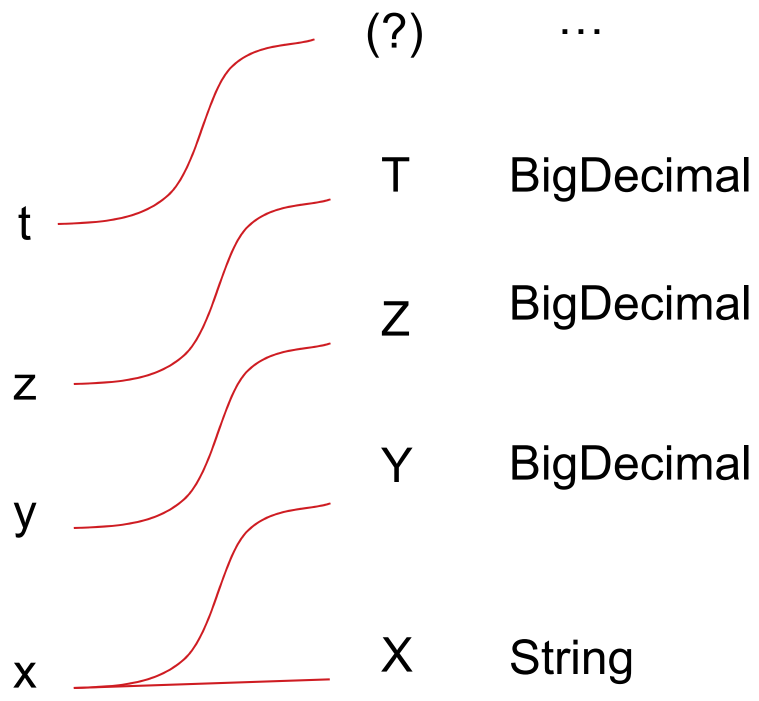
Startup Code

```
/** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        loadControls();
    }
}
```

My Weird Stack

- The top is a String (much easier to deal with when doing data entry
  - Could be considered a Register (accumulator)
  - Need flag for decimal point, and silently ignore more than one
  - Need flag for new number (any non-number key resets it)
- “Enter” pushes onto the regular stack
- Convert top of stack (String) to BigDecimal
- Pop off regular stack, and perform operation (for diadic operations; most are. Can you think of others?)
- Display results (on top of Stack (convert to String))
- Complicated calculations take only a few stack levels;
  - no operator precedent (that’s what you handle)

Stack Operation



	(?)	…
t	T	BigDecimal
z	Z	BigDecimal
y	Y	BigDecimal
x	X	String

Code Examples to Support Stack

```
private void doEnt()
{
stk.push(new BigDecimal(showTxt.getText().toString()));
ent = Boolean.TRUE;
}
/* You saw this before, but may make more sense now */
private void doDiv()
{
arg2 = new BigDecimal(showTxt.getText().toString());
arg1 = (BigDecimal)stk.pop();
arg1 = arg1.divide(arg2);
showTxt.setText(arg1.toString());
}
```

A Quick Test

- Enter a multi-digit integer, divide by another
- Use the same numbers backward
- Divide by zero (oops; need another button, and more code to handle that. Learn about throw and catch)
- Divide without a dividend (oops; need to check for stack underflow)

- Fix for those cases, and re-test
- Deliver app (internally)

Iteration Two, and Beyond

- You really need Droid Draw now!
- Lay out full keyboard (rest of the numbers, operations, what else)
  - 3 more functions (+, -, *)
  - Pi Button (it’s a number)
  - Change Sign (monadic)
  - Clear Entry Button (Cx)
  - Square Root (monadic, and a function)
  - Clear Button
  - Different number bases (octal, hexidecimal, binary…)
  - Additional Business or Scientific functions …
- Save App state (stack contents)
- Stack manipulation buttons (x <-> y)

App Distribution (local)

- Use adb to connect to your real Android device
  - Through TCP/IP (when on a wireless network)
  - Through the USB port
- adb will find all Android devices in range
- Upload your app file
- Try it out
- Rinse, Repeat

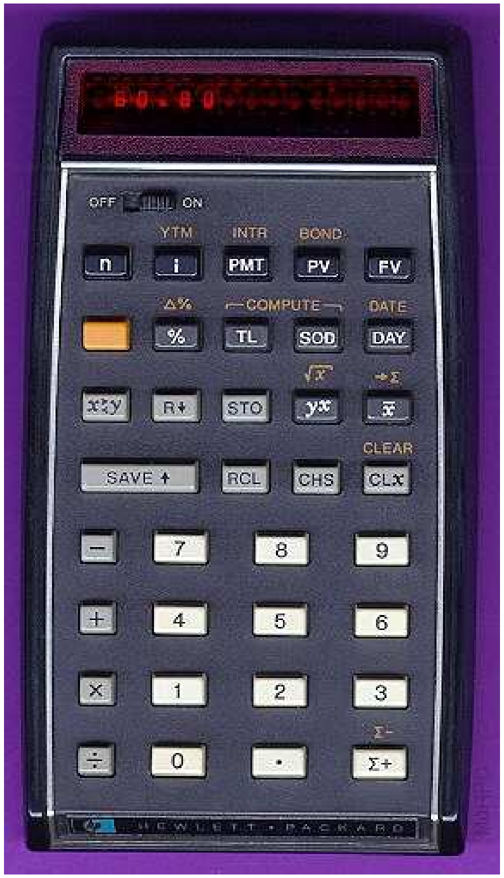
App Distribution

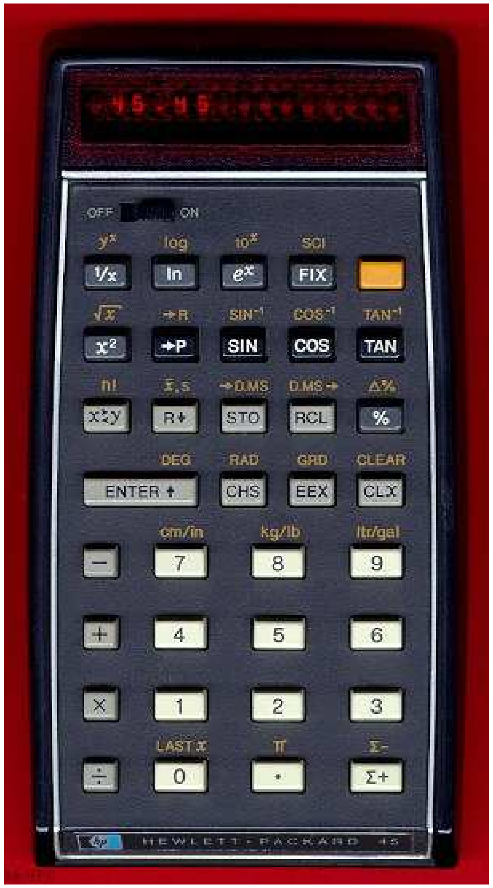
- Sign up for the Android App Store
- Sign up for the Barnes and Noble App Store (the Nook is an Android tablet underneath, Nook Color is more so)
- Learn Objective C and attack the Apple market
  - Check out the SHARE Proceedings online for the session that preceded this one (9774)

Questions?

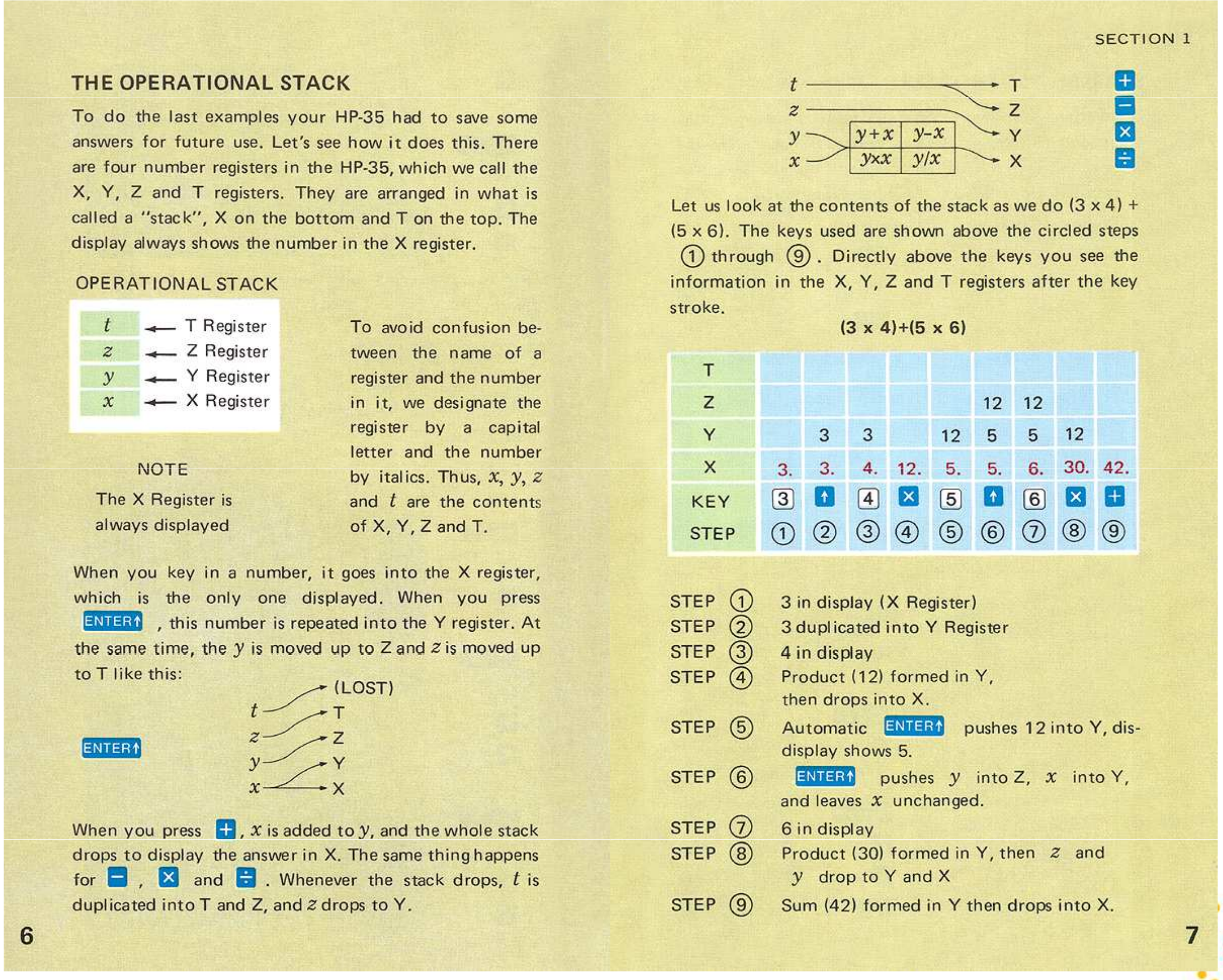
Thank You!

HP Calculators, My Inspiration





Stack and RPN in Action



THE OPERATIONAL STACK

To do the last examples your HP-35 had to save some answers for future use. Let's see how it does this. There are four number registers in the HP-35, which we call the X, Y, Z and T registers. They are arranged in what is called a "stack", X on the bottom and T on the top. The display always shows the number in the X register.

OPERATIONAL STACK

t	← T Register
z	← Z Register
y	← Y Register
x	← X Register

To avoid confusion between the name of a register and the number in it, we designate the register by a capital letter and the number by italics. Thus, *x*, *y*, *z* and *t* are the contents of X, Y, Z and T.

NOTE

The X Register is always displayed

When you key in a number, it goes into the X register, which is the only one displayed. When you press ENTER↑ , this number is repeated into the Y register. At the same time, the *y* is moved up to Z and *z* is moved up to T like this:

When you press +, *x* is added to *y*, and the whole stack drops to display the answer in X. The same thing happens for −, × and ÷. Whenever the stack drops, *t* is duplicated into T and Z, and *z* drops to Y.

Let us look at the contents of the stack as we do (3 x 4) + (5 x 6). The keys used are shown above the circled steps ① through ⑨. Directly above the keys you see the information in the X, Y, Z and T registers after the key stroke.

(3 x 4)+(5 x 6)

T									
Z						12	12		
Y		3	3		12	5	5	12	
X	3.	3.	4.	12.	5.	5.	6.	30.	42.
KEY	3	↑	4	×	5	↑	6	×	+
STEP	①	②	③	④	⑤	⑥	⑦	⑧	⑨

STEP ① 3 in display (X Register)
STEP ② 3 duplicated into Y Register
STEP ③ 4 in display
STEP ④ Product (12) formed in Y, then drops into X.
STEP ⑤ Automatic ENTER↑ pushes 12 into Y, dis-display shows 5.
STEP ⑥ ENTER↑ pushes *y* into Z, *x* into Y, and leaves *x* unchanged.
STEP ⑦ 6 in display
STEP ⑧ Product (30) formed in Y, then *z* and *y* drop to Y and X
STEP ⑨ Sum (42) formed in Y then drops into X.

References

- http://www.ibm.com/developerworks/opensource/tutorials/os-eclipse-android/