Static Analysis of Android Apps

Federico Maggi, Politecnico di Milano
fede@maggi.cc
@phretor

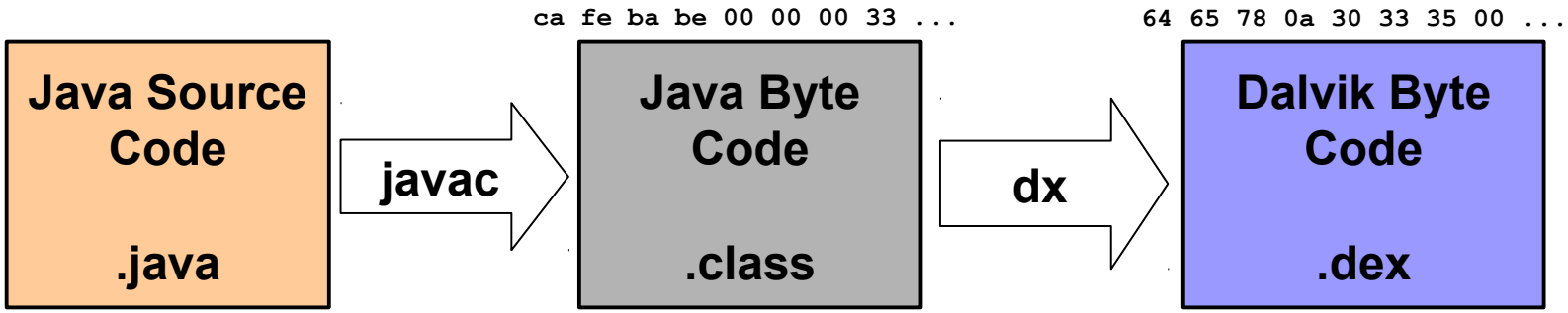
Outline

- Dalvik overview
- Dalvik executable file format
- Dalvik machine
- Disassembling
- Decompiling
- Real-world examples
- Limitations

Dalvik Overview

- high-level language: Java
  - translated in Dalvik bytecode
- different from Java bytecode
  - register based (DEX)
  - stack based (class)
- optimization
  - space
  - less instructions
  - each instruction carries a lot of semantic

Toolchain



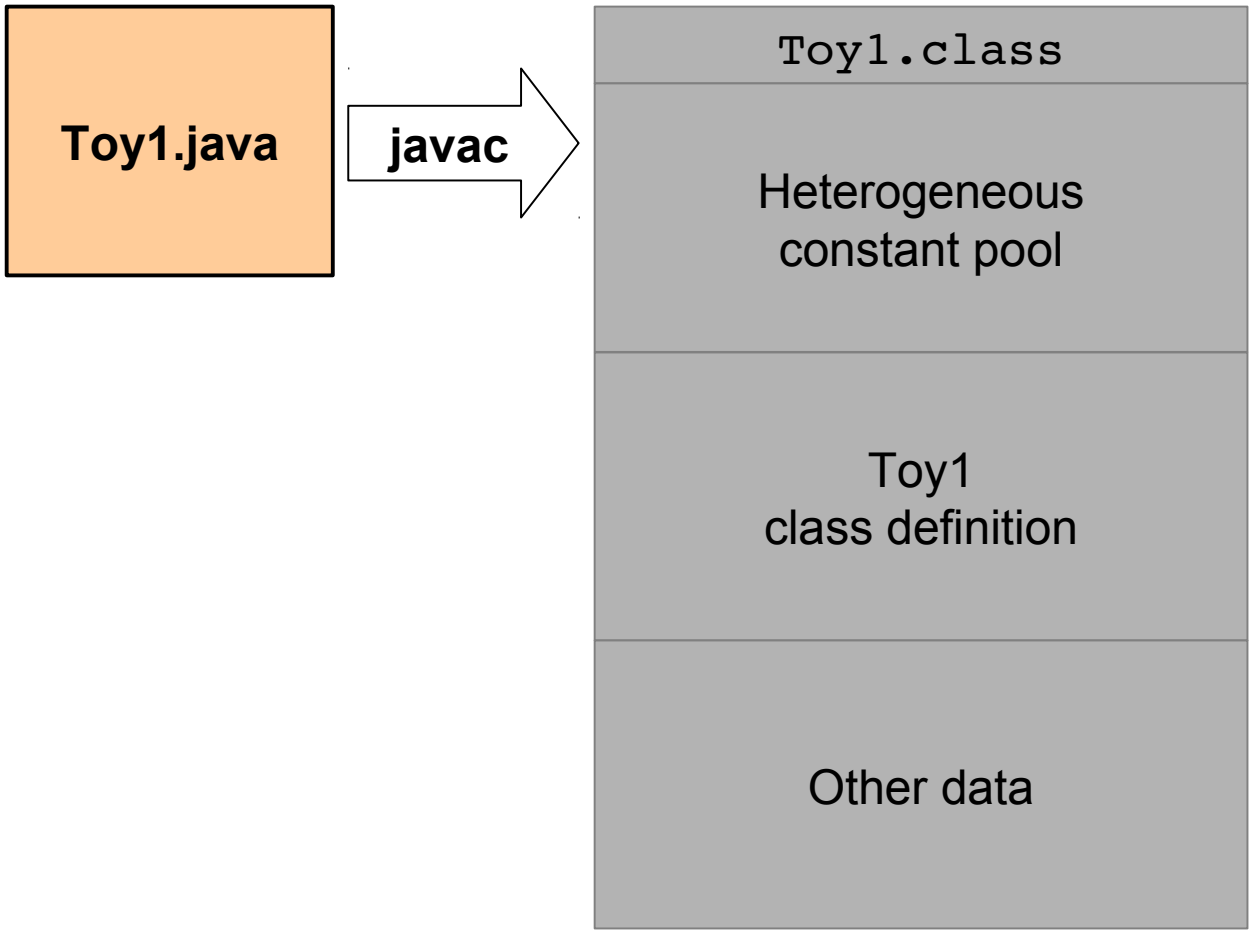

```
class Toy1 {
    static int a;
    static int b;
    static int c;

    public static void
hello() {
        a = 0;
    }

…



    public static void main
(String[] args) {
        hello();
        world();
        sum(a, b);

     System.out.println(c);
    }
}
```

```
class Toy1 {
  static int a;

  static int b;

  static int c;

  Toy1();
    Code:
       0: aload_0
       1: invokespecial #1
                // Method
java/lang/Object."<init>":
()V
       4: return
       // Field c:I
      21: invokevirtual #9
                // Method
java/io/PrintStream.println:
(I)V
      24: return
}
```

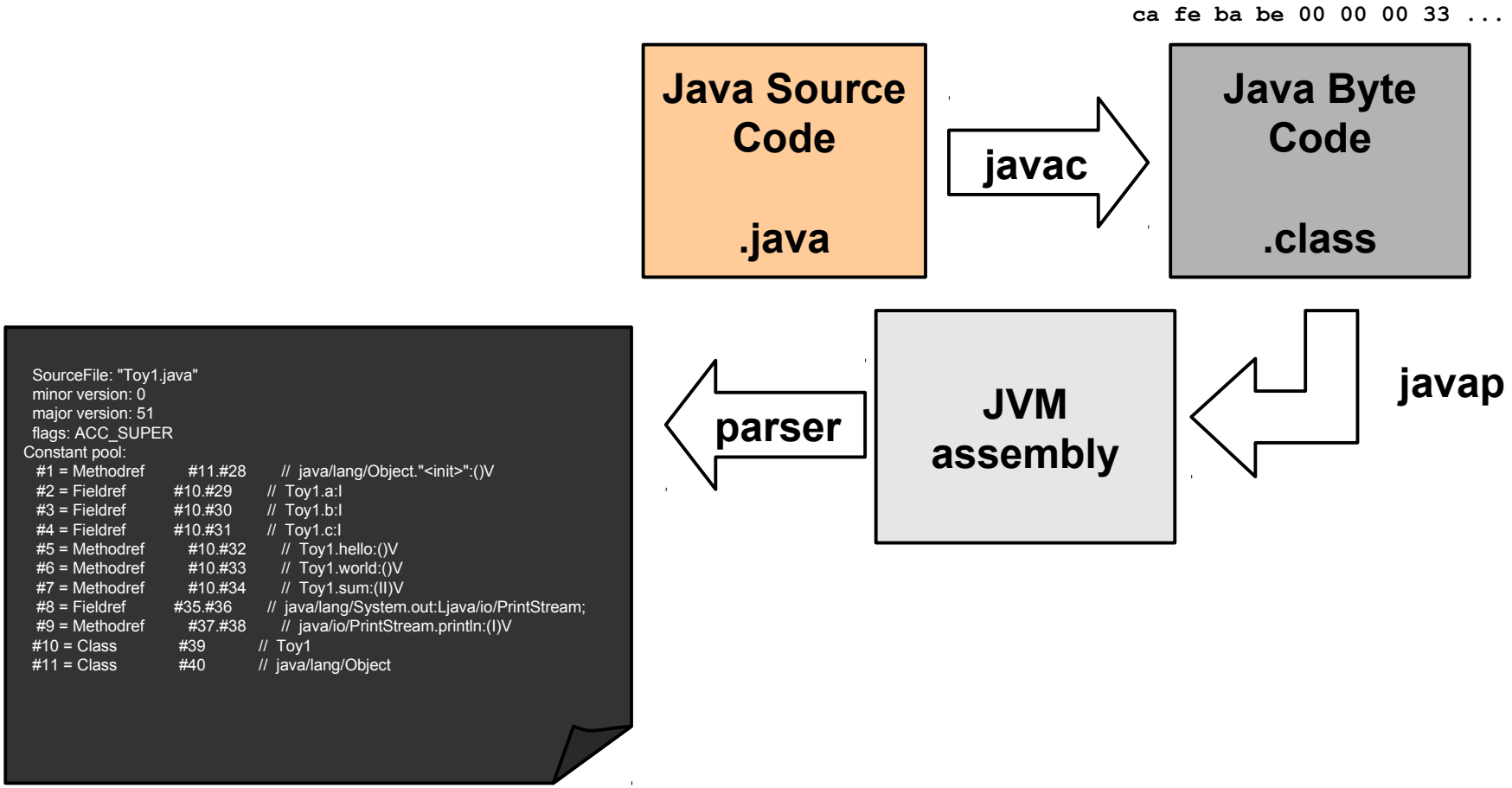
```
.class Toy1
.super java/lang/Object
.source Toy1.java

.field static a I
.field static b I
.field static c I


...

.line 19
        invoke-static
{},Toy1/hello    ; hello()V
;
        nop
.line 20
        invoke-static
{},Toy1/world    ; world()V
;
        nop
.line 21
```

```
class Toy1 {
    static int a;
    static int b;
    static int c;

    public static void hello() {
        a = 0;
    }

    public static void world() {
        b = 1;
    }

    public static void sum(int first, int second) {
        c = first + second;
    }

    public static void main (String[] args) {
        hello();
        world();
        sum(a, b);

        System.out.println(c);
    }
}
```

Java Bytecode Format



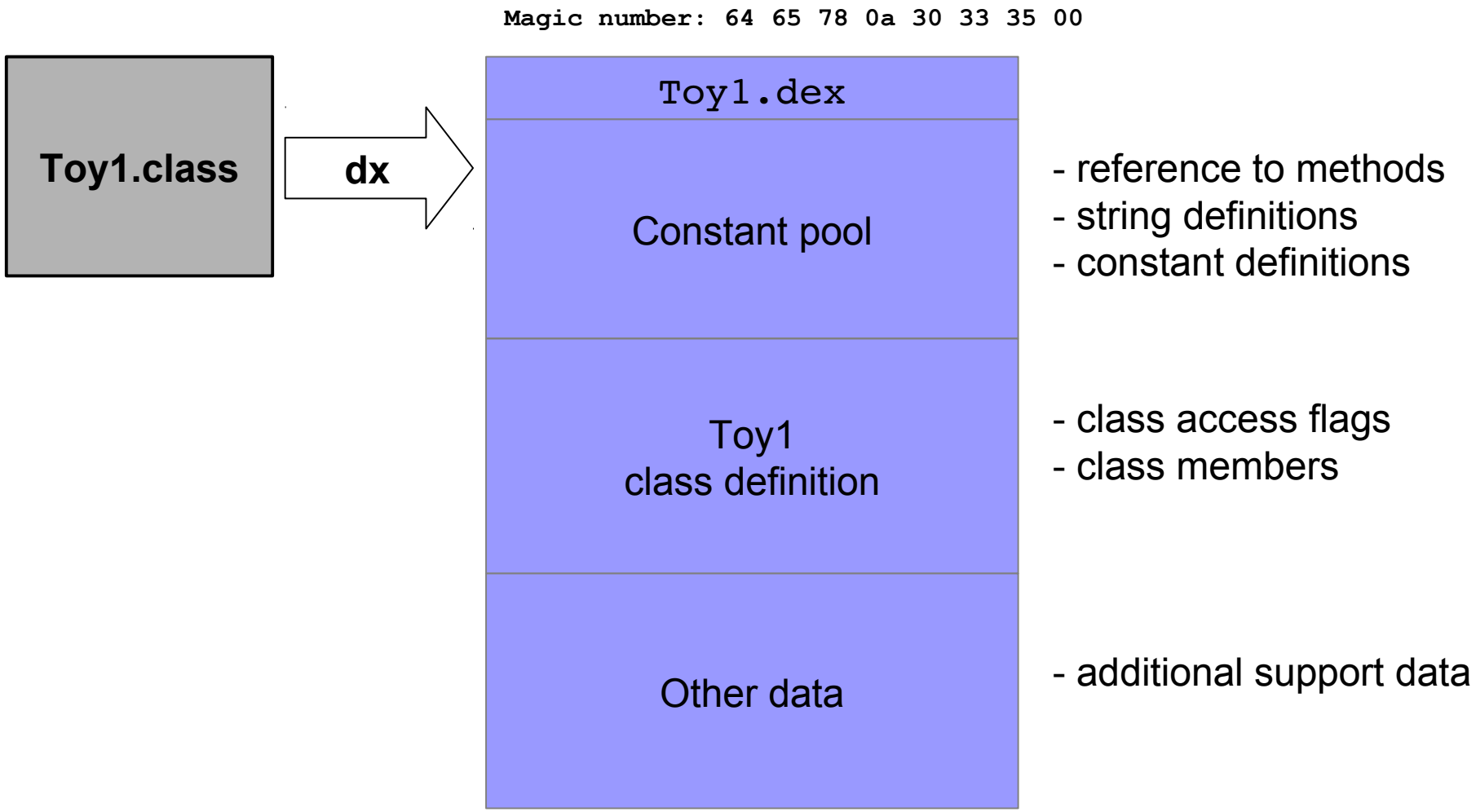
Disassembling Java Bytecode



```
→ toy1  javap -v -constants Toy1
Classfile /home/pfretor/Desktop/summerschool-2014/summerschool-2014/apps/session1/toy1/Toy1.class
  Last modified Sep 16, 2014; size 680 bytes
  MD5 checksum 87527efae82427cf620c192758fa90e9
  Compiled from "Toy1.java"
class Toy1
  SourceFile: "Toy1.java"
  minor version: 0
  major version: 51
  flags: ACC_SUPER
Constant pool:
   #1 = Methodref          #11.#28        //  java/lang/Object."<init>":()V
   #2 = Fieldref           #10.#29        //  Toy1.a:I
   #3 = Fieldref           #10.#30        //  Toy1.b:I
   #4 = Fieldref           #10.#31        //  Toy1.c:I
   #5 = Methodref          #10.#32        //  Toy1.hello:()V
   #6 = Methodref          #10.#33        //  Toy1.world:()V
   #7 = Methodref          #10.#34        //  Toy1.sum:(II)V
   #8 = Fieldref           #35.#36        //  java/lang/System.out:Ljava/io/PrintStream;
   #9 = Methodref          #37.#38        //  java/io/PrintStream.println:(I)V
  #10 = Class              #39            //  Toy1
  #11 = Class              #40            //  java/lang/Object
  #12 = Utf8               a
  #13 = Utf8               I
  #14 = Utf8               b
  #15 = Utf8               c
  #16 = Utf8               <init>
  #17 = Utf8               ()V
  #18 = Utf8               Code
  #19 = Utf8               LineNumberTable
  #20 = Utf8               hello
  #21 = Utf8               world
  #22 = Utf8               sum
  #23 = Utf8               (II)V
  #24 = Utf8               main
  #25 = Utf8               ([Ljava/lang/String;)V
  #26 = Utf8               SourceFile
```

```
public static void world();
  flags: ACC_PUBLIC, ACC_STATIC
  Code:
    stack=1, locals=0, args_size=0
       0: iconst_1
       1: putstatic     #3                  // Field b:I
       4: return
    LineNumberTable:
      line 11: 0
      line 12: 4

public static void sum(int, int);
  flags: ACC_PUBLIC, ACC_STATIC
  Code:
    stack=2, locals=2, args_size=2
       0: iload_0
       1: iload_1
       2: iadd
       3: putstatic     #4                  // Field c:I
       6: return
    LineNumberTable:
      line 15: 0
      line 16: 6

public static void main(java.lang.String[]);
  flags: ACC_PUBLIC, ACC_STATIC
  Code:
    stack=2, locals=1, args_size=1
       0: invokestatic  #5                  // Method hello:()V
       3: invokestatic  #6                  // Method world:()V
       6: getstatic     #2                  // Field a:I
       9: getstatic     #3                  // Field b:I
      12: invokestatic  #7                  // Method sum:(II)V
      15: getstatic     #8                  // Field java/lang/System.out:Ljava/io/PrintStream;
      18: getstatic     #4                  // Field c:I
      21: invokevirtual #9                  // Method java/io/PrintStream.println:(I)V
      24: return
```

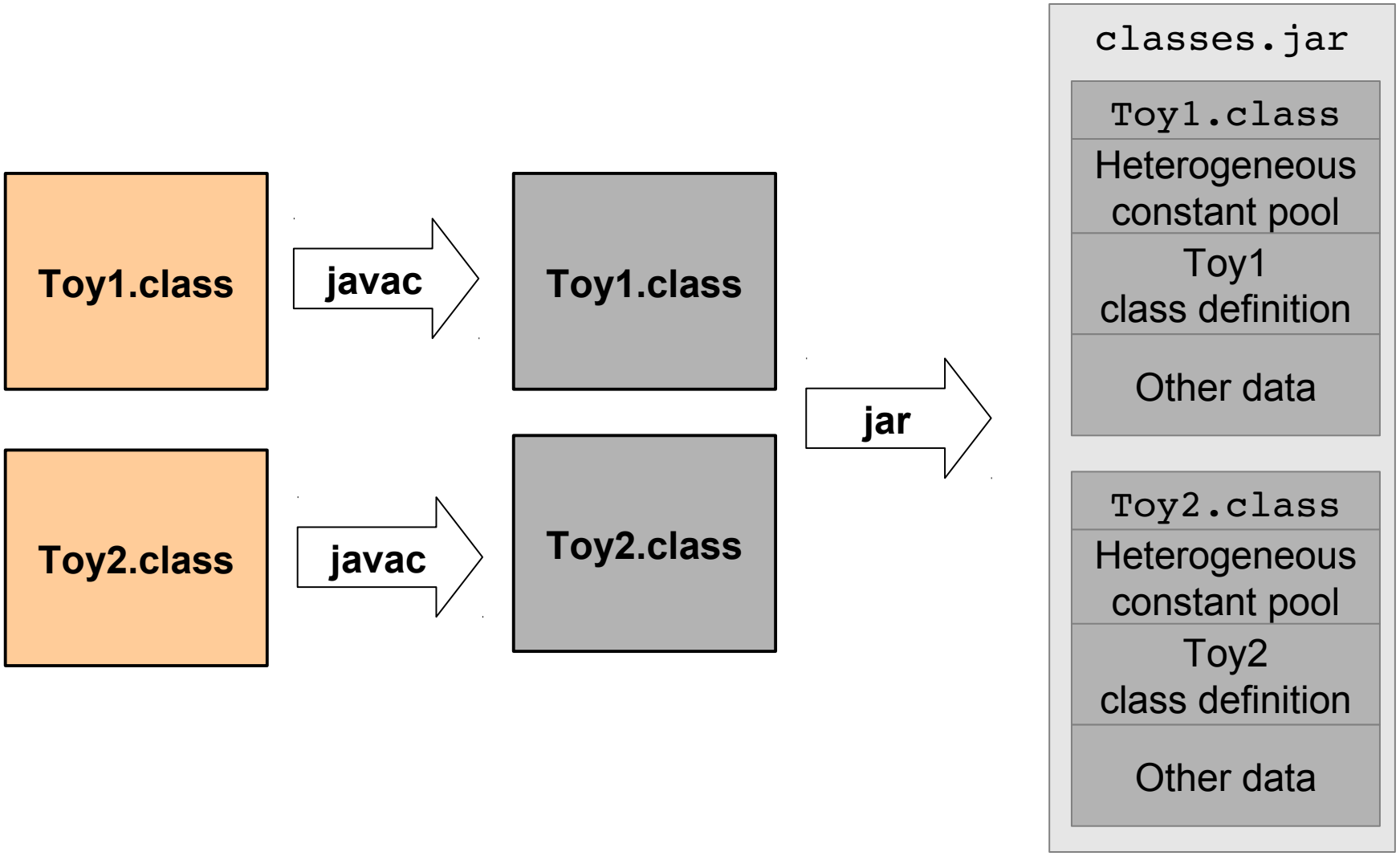
Java Bytecode to Dalvik Bytecode



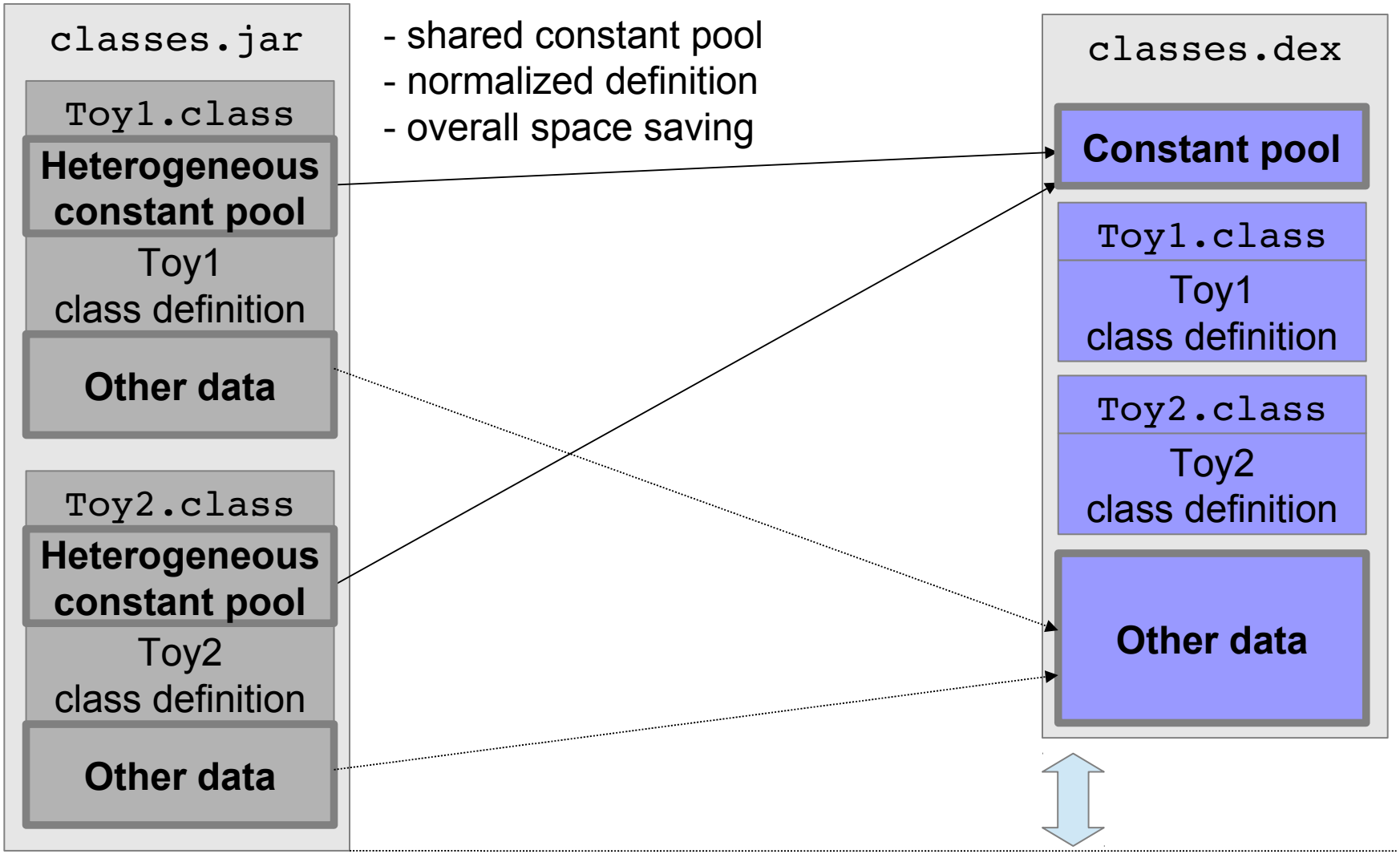
Demo (Toy1.java)

- Execute Dalvik code in the emulator

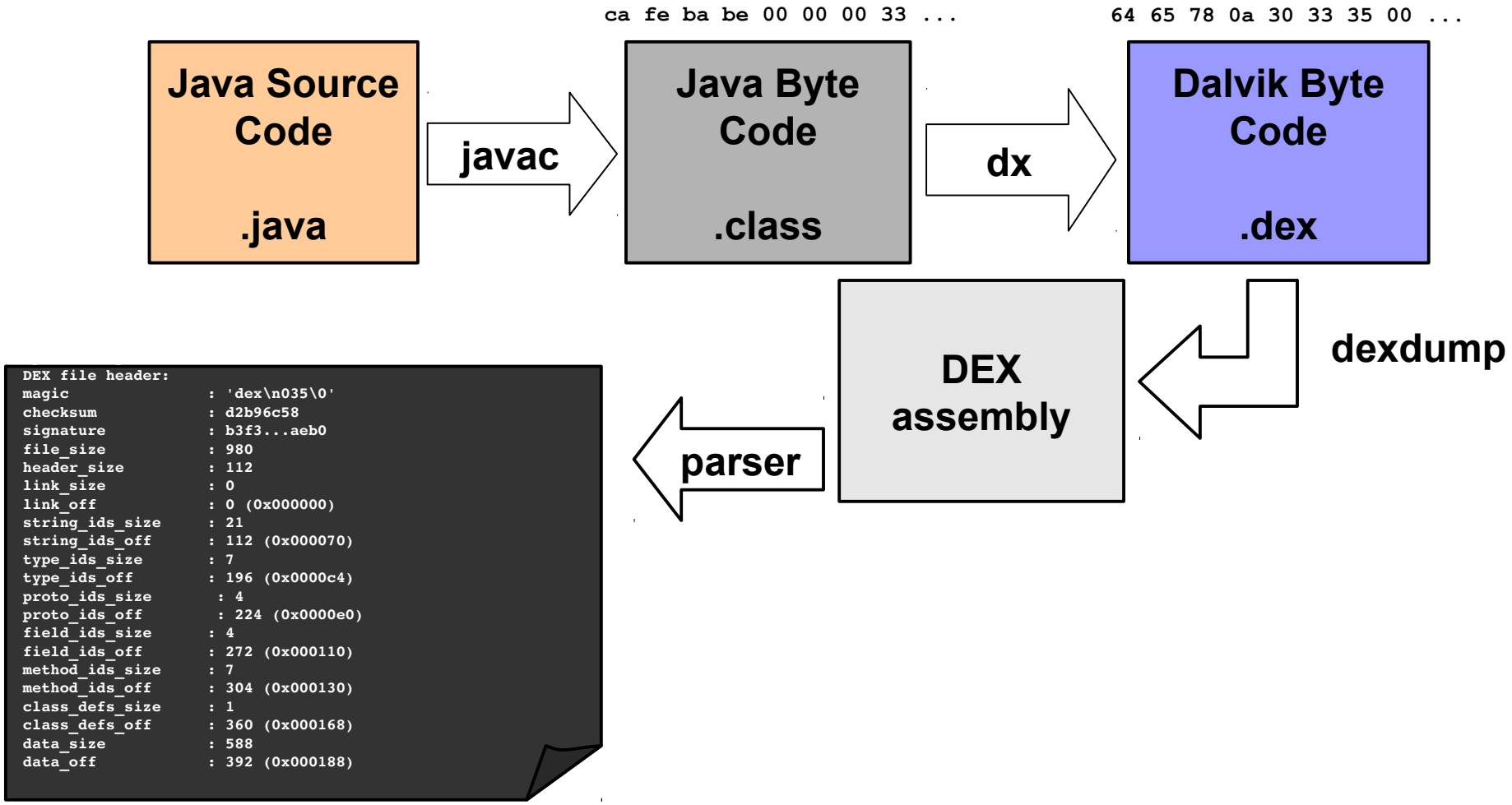
JAR File Format



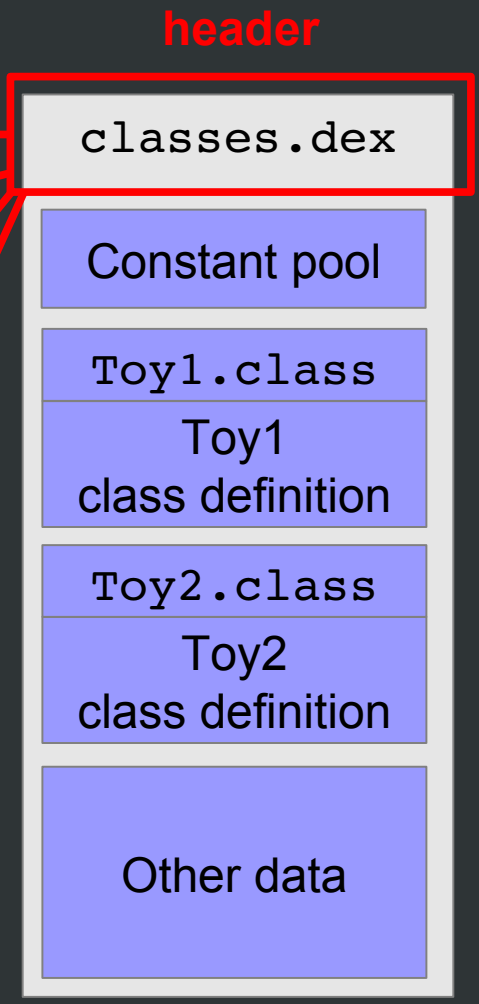
Sharing is Caring (for space)



Disassembling Dalvik Bytecode



```
→  toy1  dexdump -f Toy1.dex
Processing 'Toy1.dex'...
Opened 'Toy1.dex', DEX version '035'
DEX file header:
magic               : 'dex\n035\0'
checksum            : d2b96c58
signature           : b3f3...aeb0
file_size           : 980
header_size         : 112
link_size           : 0
link_off            : 0 (0x000000)
string_ids_size     : 21
string_ids_off      : 112 (0x000070)
type_ids_size       : 7
type_ids_off        : 196 (0x0000c4)
proto_ids_size      : 4
proto_ids_off       : 224 (0x0000e0)
field_ids_size      : 4
field_ids_off       : 272 (0x000110)
method_ids_size     : 7
method_ids_off      : 304 (0x000130)
class_defs_size     : 1
class_defs_off      : 360 (0x000168)
data_size           : 588
data_off            : 392 (0x000188)
```



class_def

```
Class #0            -
  Class descriptor  : 'LToy1;'
  Access flags      : 0x0000 ()
  Superclass        : 'Ljava/lang/Object;'
  Interfaces        -
  Static fields     -
    #0              : (in LToy1;)
      name          : 'a'
      type          : 'I'
      access        : 0x0008 (STATIC)
    #1              : (in LToy1;)
      name          : 'b'
      type          : 'I'
      access        : 0x0008 (STATIC)
    #2              : (in LToy1;)
      name          : 'c'
      type          : 'I'
      access        : 0x0008 (STATIC)
  Instance fields   -
  Direct methods    -
    #0              : (in LToy1;)
      name          : '<init>'
      type          : '()V'
      access        : 0x10000 (CONSTRUCTOR)
      code          -
      registers     : 1
      ins           : 1
      outs          : 1
      insns size    : 4 16-bit code units
000188:
000198: 7010 0600 0000
// method@0006
00019e: 0e00
      catches       : (none)
      positions     :
        0x0000 line=1
      locals        :
        0x0000 - 0x0004 reg=0 this LToy1;
```

constructor

```
class Toy1 {
    static int a;
    static int b;
    static int c;

    public static void hello() {
        a = 0;
    }

    public static void world() {
        b = 1;
    }

    public static void sum(int first, int second) {
        c = first + second;
    }

    public static void main (String[] args) {
        hello();
        world();
        sum(a, b);

        System.out.println(c);
    }
}
```

code

```
|[000188] Toy1.<init>:()V
|0000: invoke-direct {v0}, Ljava/lang/Object;.<init>:()V

|0003: return-void
```

Method Definition

```
    #3              : (in LToy1;)
      name          : 'sum'
      type          : '(II)V'
      access        : 0x0009 (PUBLIC STATIC)
      code          -
      registers     : 3
      ins           : 2
      outs          : 0
      insns size    : 5 16-bit code units
0001f4:                                        |[0001f4] Toy1.sum:(II)V
000204: 9000 0102                              |0000: add-int v0, v1, v2
000208: 6700 0200                              |0002: sput v0, LToy1;.c:I // field@0002
00020c: 0e00                                   |0004: return-void
```

```
#3 method ID          : (in LToy1;) method member
   name               : 'sum' method name
   type               : '(II)V' take two ints and return void
   access             : 0x0009 (PUBLIC STATIC)
   code               -
   registers          : 3 registers used: v0 v1 v2
```

other (e.g., local vars)

2nd parameter

1st parameter

Types

- reference types
  - objects **`Lpackage/name/ObjectName`**
  - arrays **`[I → int[], [[I → int[][], [Ljava/lang/String, ...`**
- primitive types (all the rest)
  - V: void
  - Z: boolean
  - B: byte
  - S: short
  - C: char
  - I: int
  - J long (64 bits)
  - F: float
  - D: double (64 bits)

Code

```
   #3              : (in LToy1;)
     name          : 'sum'
     type          : '(II)V'
     access        : 0x0009 (PUBLIC STATIC)
     code          -
     registers     : 3
     ins           : 2
     outs          : 0
     insns size    : 5 16-bit code units
0001f4:                                        |[0001f4] Toy1.sum:(II)V
000204: 9000 0102                              |0000: add-int v0, v1, v2
000208: 6700 0200                              |0002: sput v0, LToy1;.c:I // field@0002
00020c: 0e00                                   |0004: return-void
```

v0 ← v1 + v2

Sum the content of register v1 and v2, and move result in v0

```
|[0001f4] Toy1.sum:(II)V
|0000: add-int v0, v1, v2
|0002: sput v0, LToy1;.c:I // field@0002
|0004: return-void
```

Put content of v0 in static field

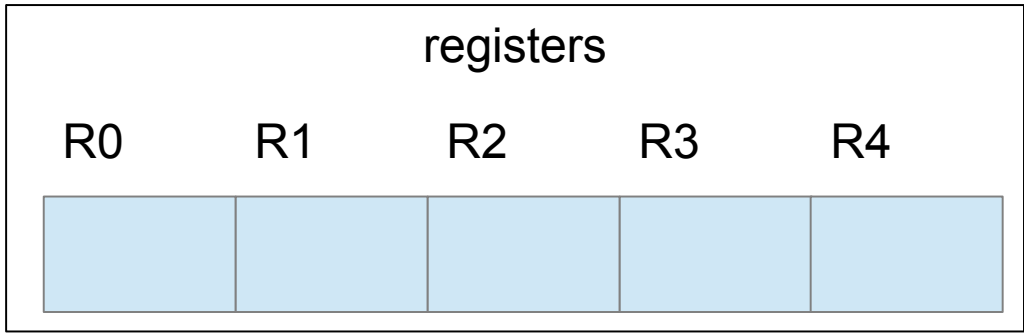
- 218 op-codes
- long instructions
- best reference: http://tinyurl.com/dalvik-opcodes

Dalvik Machine

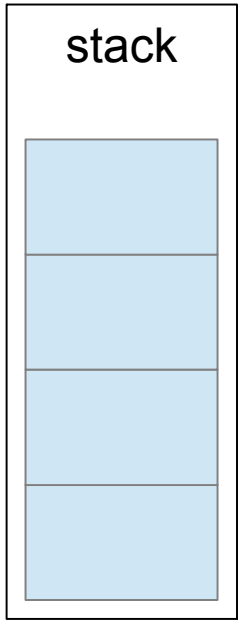
- register based (JVM is stack based)
- 2^64 registers (wow!)
- 32 bits registers
  - how about long/double? Just use two registers

Stack Based: JVM

```
int s = 3 + 4;
```

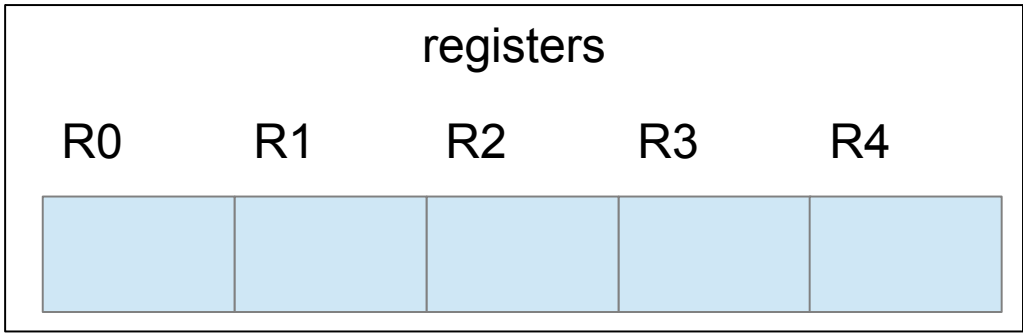


```
iconst_3
iconst_4
iadd
istore_0
```

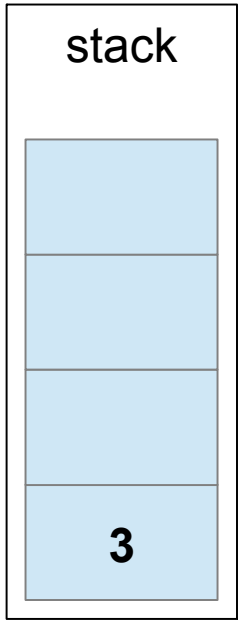


Stack Based: JVM

```
int s = 3 + 4;
```

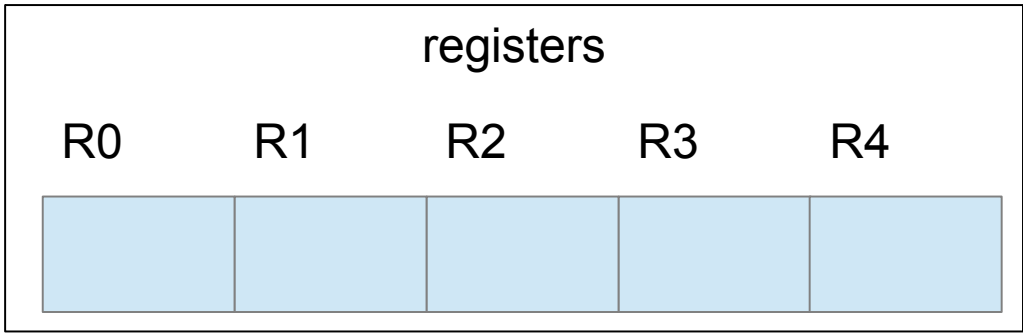



```
iconst_3
iconst_4
iadd
istore_0
```

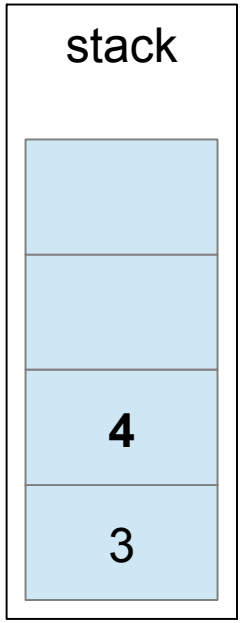


Stack Based: JVM

```
int s = 3 + 4;
```

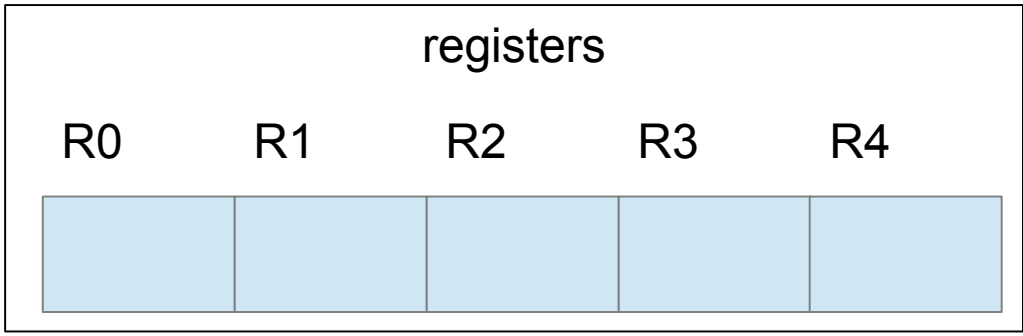



```
iconst_3
iconst_4
iadd
istore_0
```

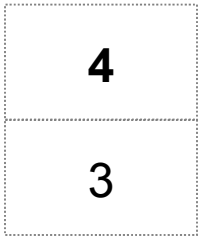


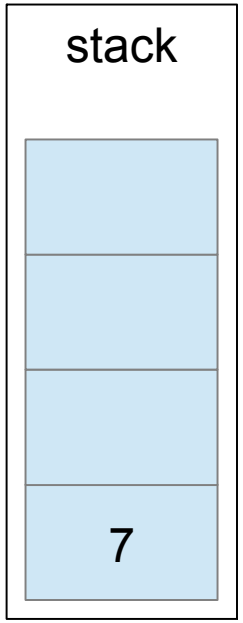
Stack Based: JVM

```
int s = 3 + 4;
```



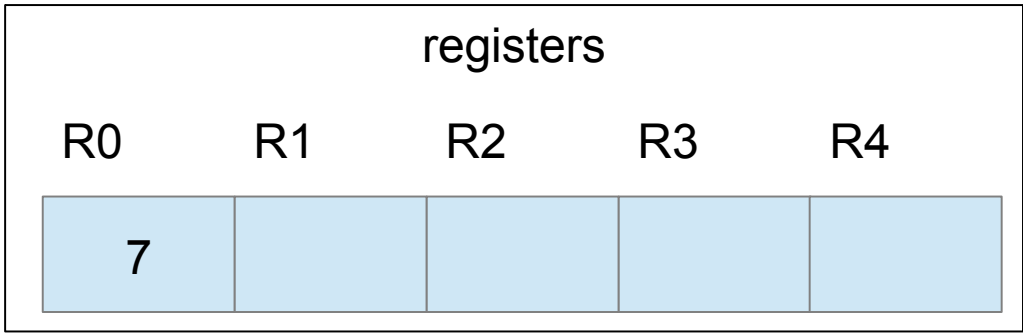

```
iconst_3
iconst_4
iadd
istore_0
```



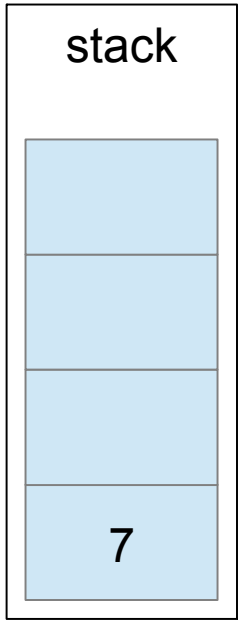


Stack Based: JVM

```
int s = 3 + 4;
```

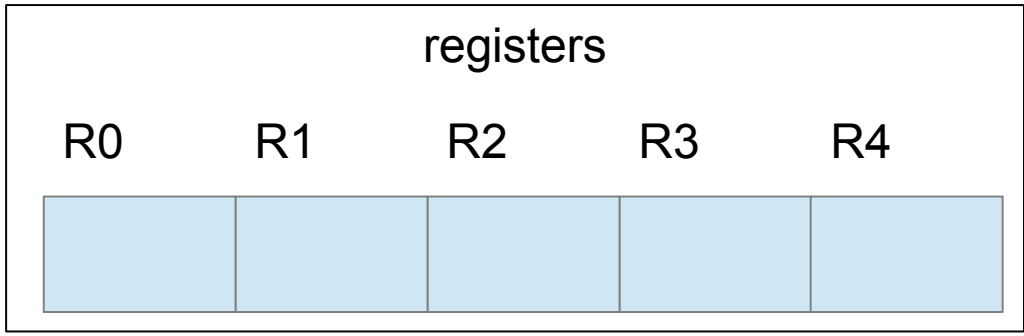



```
iconst_3
iconst_4
iadd
istore 0
```

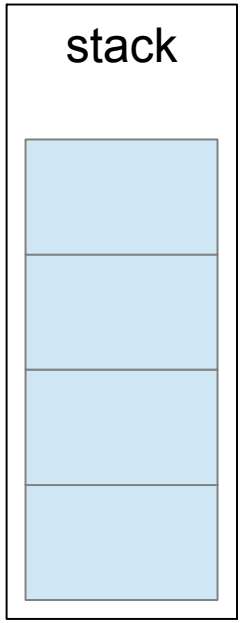


Register Based: Dalvik

```
int s = 3 + 4;
```

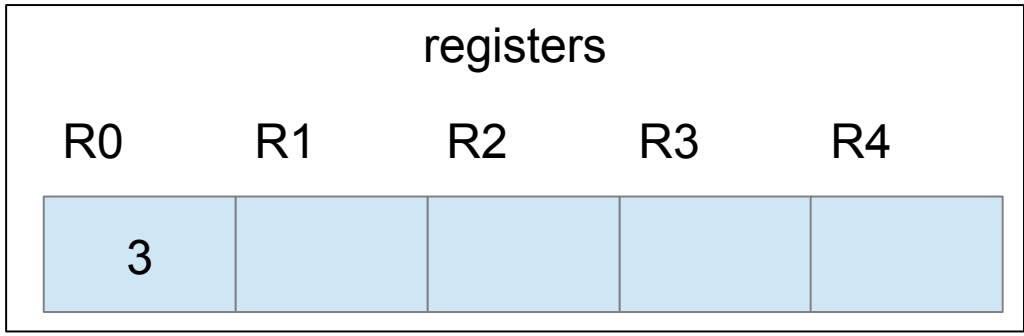


```
const/4 r0 3
add-int/lit8 r1 r0 4
```

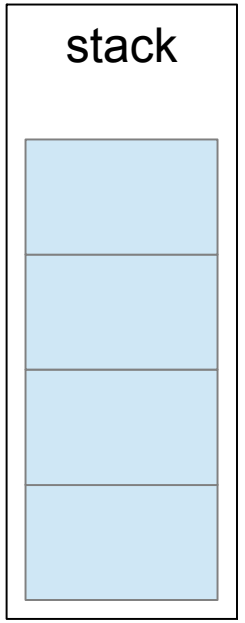


Register Based: Dalvik

```
int s = 3 + 4;
```

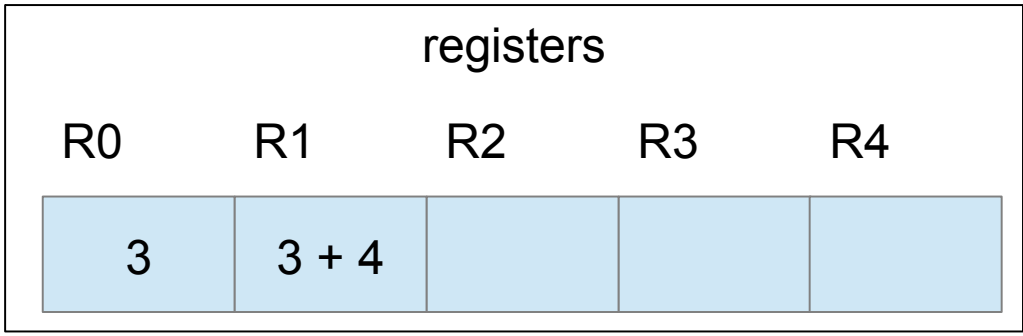



```
const/4 r0 3
add-int/lit8 r1 r0 4
```

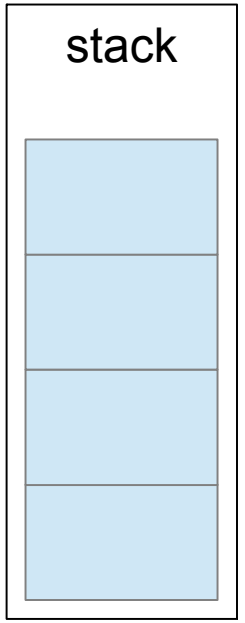


Register Based: Dalvik

```
int s = 3 + 4;
```

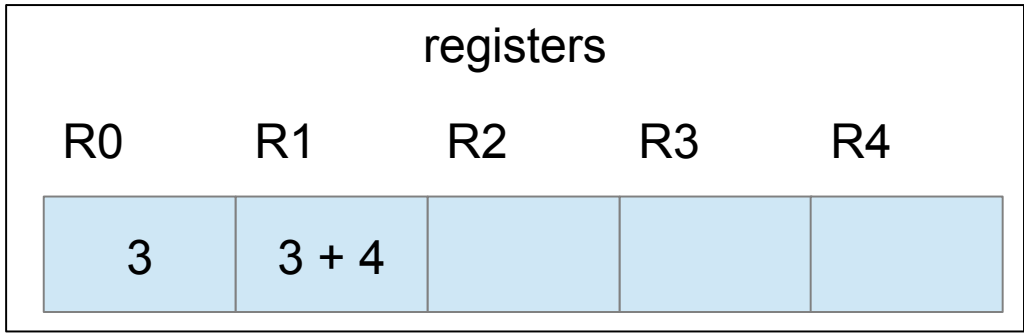


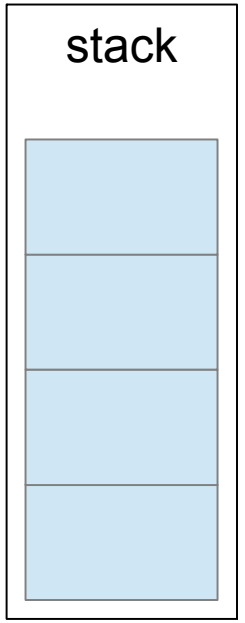

```
const/4 r0 3
add-int/lit8 r1 r0 4
```



Register Based: Dalvik

```
int s = 3 + 4;
```






```
const/4 r0 3
add-int/lit8 r1 r0 4
```

Disassembling: The smali language

- dexdump's output is rather verbose
- the smali language is an alternative and more readable intermediate language
- the concepts seen so far are the same

smali/baksmali Example

```
# virtual methods
.method public check(Ljava/lang/String;)Z
    .registers 3

    .prologue
    .line 5
    const-string v0, "syssecrulez"

    invoke-virtual {p1, v0}, Ljava/lang/String;->equals(Ljava/lang/Object;)Z

    move-result v0

    if-eqz v0, :cond_a

    .line 6
    const/4 v0, 0x1

    .line 7
    :goto_9
    return v0

    :cond_a
    const/4 v0, 0x0

    goto :goto_9
.end method
```

smali/baksmali Example

```
package foo;

public class PasswordChecker {
    public boolean check(String pass) {
        if (pass.equals("syssecrulez"))
            return true;
        return false;
    }
}
```

smali/baksmali Example

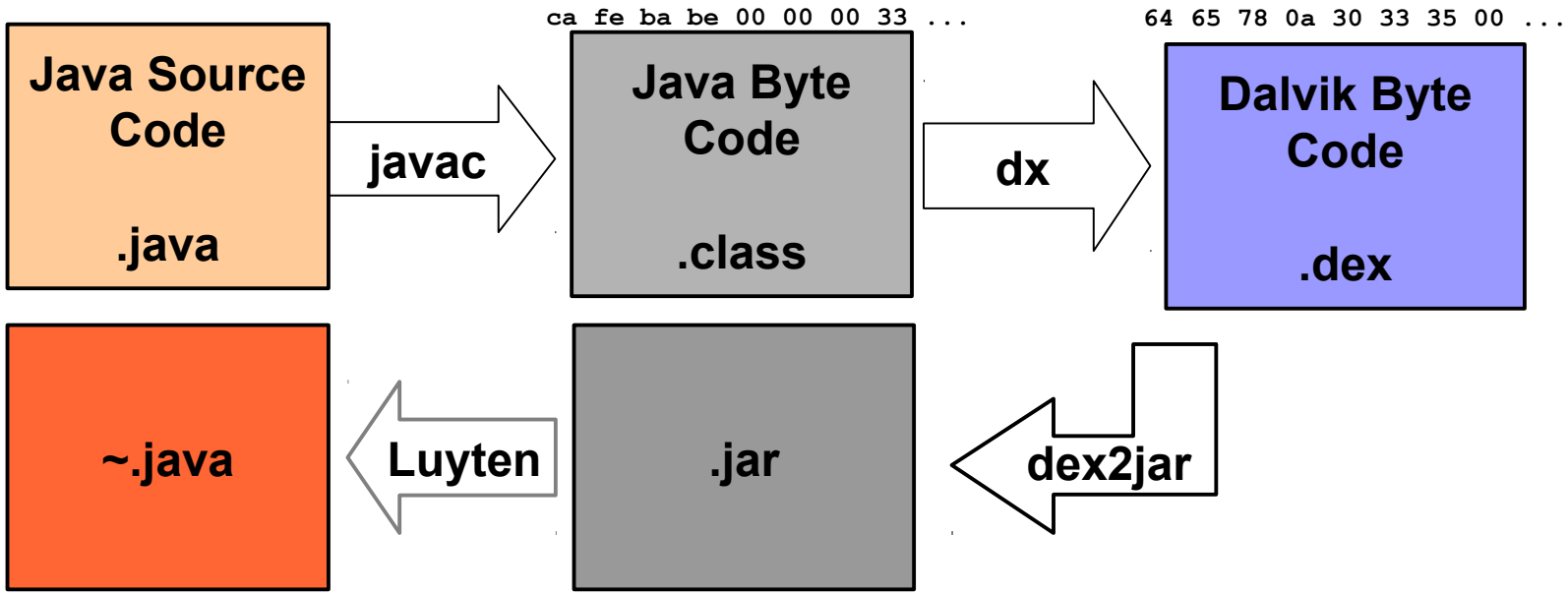
```
package foo;

import foo.PasswordChecker;

class Toy3 {
    public static void main (String[] args) {
        PasswordChecker checker = new PasswordChecker();
        if (args.length != 1)
            System.exit(0);

        if (checker.check(args[0]))
            System.out.println("p0wned");
    }
}
```

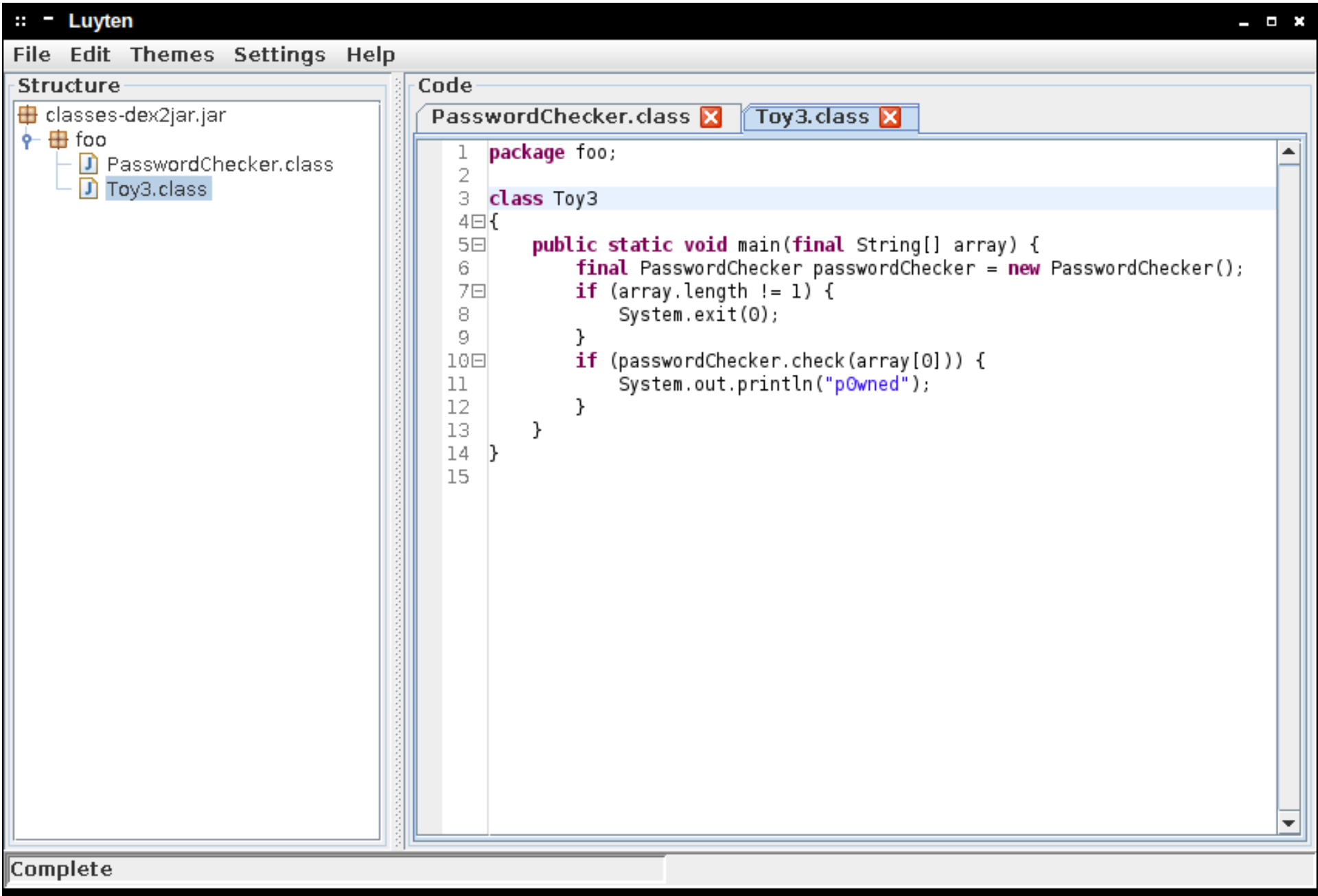
Simple Patching From smali Code

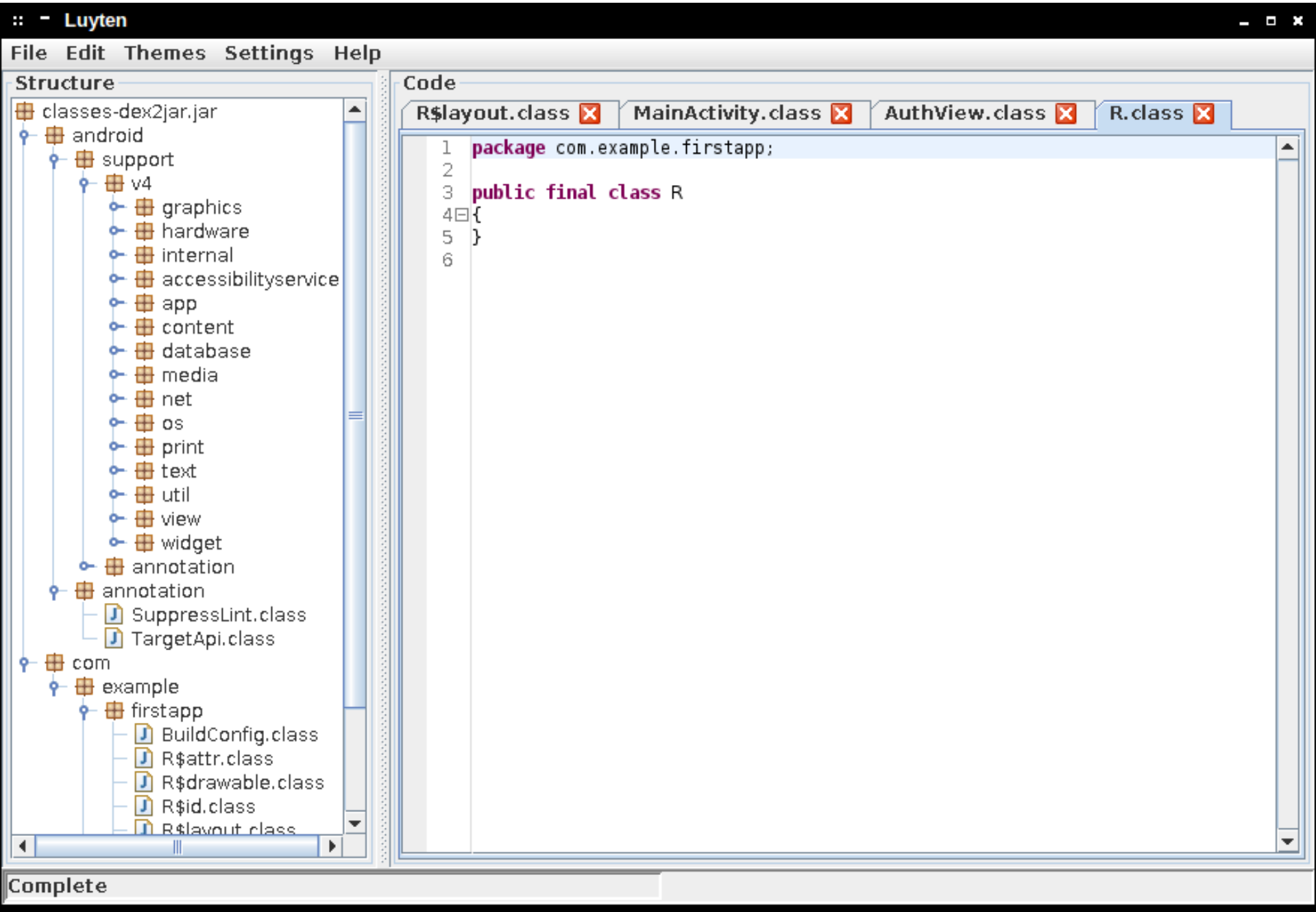
- **goal**: bypass password protection
  - let's suppose that the password is dynamic
  - so we can't just change/read it
  - we must modify the control flow
- edit the assembly code
- re-assemble it with smali *.class
- you will try this later ;-)

Decompiling

- from intermediate language to source code
- most of the times, decompiling is feasible




Luyten
File Edit Themes Settings Help
Structure
classes-dex2jar.jar
foo
PasswordChecker.class
Toy3.class
Code
PasswordChecker.class
Toy3.class
1 package foo;
2
3 class Toy3
4 {
5 public static void main(final String[] array) {
6 final PasswordChecker passwordChecker = new PasswordChecker();
7 if (array.length != 1) {
8 System.exit(0);
9 }
10 if (passwordChecker.check(array[0])) {
11 System.out.println("pOwned");
12 }
13 }
14 }
15
Complete


Luyten
File Edit Themes Settings Help
Structure
classes-dex2jar.jar
android
support
v4
graphics
hardware
internal
accessibilityservice
app
content
database
media
net
os
print
text
util
view
widget
annotation
annotation
SuppressLint.class
TargetApi.class
com
example
firstapp
BuildConfig.class
R$attr.class
R$drawable.class
R$id.class
R$layout.class
Code
R$layout.class
MainActivity.class
AuthView.class
R.class
1 package com.example.firstapp;
2
3 public final class R
4 {
5 }
6
Complete

Patching from Source Code

- it is possible, of course
- the decompilation process leaves “something” behind (e.g., var names, constants)
- libraries need to be re-imported and re-linked
- Eclipse will help you in this, but it's tedious
- just create a project and make sure that you solve all the warnings and errors you'll be fine

In practice

- recompiling from source is usually a lot of manual work
- best approach
  - decompile to source
  - decode the APK with apktool d Old.apk
  - figure out what the app does by looking at *.java
  - patch the *.smali
  - reassemble with apktool b Old/ New.apk

Limitations

- static analysis can be hindered by obfuscation
- sometimes even disassembling is difficult
- disassembling as well as decompiling strive to solve ambiguities
- sometimes such ambiguities can be introduced manually by the malicious guys

Summary

- what is JVM code and how does it look like
- what is Dalvik code and how it differs from JVM
- how to disassemble using dexdump and baksmali
- how to patch and re-assemble using smali
- how to decompile using dex2jar and Luyten
- how to recompile using a Java compiler
- Thanasis will tell you more about Dalvik ;-)

Practical Session: Part 1

- PlayMe.mp3 is just landed to your laptop, but you suspect it's not an mp3, maybe you want to use “file” or “hexdump” to take a look at its first bytes and see if you find something familiar

- Done already? Why don't you take a look at SecondApp.apk or Toy3.dex and see if you can do something there to p0wn them!

Practical Session: Part 2

- You just found out that your device has a suspicious SecondApp.apk file which seem to be coming out from nowhere...you better not execute it! It's maybe safer to take a look at its code.

- Done already? Why don't you take a look at Main.apk? Seems a nice battery-saving app. Not entirely sure though...

Practical Session: Part 3

Warning, warning! Looks like you received an application that looks malicious. You better fire up your favorite disassembler and decompiler and dig into the code before it's too late!