HOW "*FINAL*" IS FINAL?

MUSINGS ABOUT *FINALITY* IN THE JAVA LANGUAGE AND VM

Volker Simonis [Фолкер Симонис], SAP / volker.simonis@gmail.com


Живая и мертвая вода

JAVA'S DEFINITION OF “FINAL”

“A `final` variable may only be assigned to once..”

“Once a final variable has been assigned, it always contains the same value.”

The Java Language Specification, §4.12.4. final Variables

```
class Test0 {
  private final int f;
  public Test0() {
  }
}
```

Compile time error: *variable f might not have been initialized.*

```java
class TestA {
  private final int f = 0;
  public TestA(int f) {
    this.f = f;
  }
}
```

Compile time error: *cannot assign a value to final variable f*

```java
class TestB {
  private final int f;
  {
    f = 0; // Instance initializer
  }
  public TestB(int f) {
    this.f = f;
  }
}
```

Compile time error: *cannot assign a value to final variable f*

```java
class TestC {
  private final int f;
  public TestC(int f) {
    this.f = f;
  }
}
```

OK: since Java 1.1 (“*blank final*”).

```java
class TestF {
  private final int f;
  public TestF(int f) {
    if (f > 0) {
      this.f = f;
    } else {
      this.f = 42;
    }
  }
}
```

OK: `javac` does a certain amount of flow analysis..

```
class TestG {
  private final int f;
  public TestG(int f) {
    if (f > 0) {
      this.f = f;
    }
    if (f <= 0) {
      this.f = 42;
    }
  }
}
```

..but not too much!

Compile time error: *variable f might already have been initialized.*

```
class TestG1 {
  private final int f;
  public TestG1(int f) {
    try {
      this.f = f;
    }
    catch (Exception e) {

    }
  }
}
```

Compile time error: *variable f might not have been initialized.*

```
class TestG1 {
  private final int f;
  public TestG1(int f) {
    try {
      this.f = f;
    }
    catch (Exception e) {
      this.f = 42;
    }
  }
}
```

Compile time error: *variable f might already have been initialized.*

```java
class TestG1 {
  public int f;
  public TestG1(int f) {
    try {
      this.f = f;
    }
    catch (Exception e) {
      this.f = 42;
    }
  }
}
```

Without `final` OK

```
class TestG1 {
  public int f;
  public TestG1(int);
    0: aload_0
    1: invokespecial #1 // Object.<init>
    4: aload_0
    5: iload_1
    6: putfield      #2 // Field f:I
    9: goto          19
   12: astore_2
   13: aload_0
   14: bipush        42
   16: putfield      #2 // Field f:I
   19: return
 Exception table:
    from    to  target type
       4     9    12   Exception
```

```java
class TestG1 {
  public final int f;
  public TestG1(int f) {
    try {
      this.f = f;
    }
    catch (Exception e) {
      this.f = 42;
    }
  }
}
```

With `final` compile Error!

But we can generate the Bytecode:

```
class TestG1 {
  public final int f;
  public TestG1(int);
    0: aload_0
    1: invokespecial #1 // Object.<init>
    4: aload_0
    5: iload_1
    6: putfield      #2 // Field f:I
    9: goto          19
   12: astore_2
   13: aload_0
   14: bipush        42
   16: putfield      #2 // Field f:I
   19: return
 Exception table:
    from    to  target type
       4     9    12   Exception
```

THE JVM'S DEFINITION OF “FINAL”

> *“If the field is final the `putfield` instruction must occur in an instance initialization method (<init>) of the current class.”*
>
> The Java Virtual Machine Specification, §6.5.putfield

No or multiple assignmets within constructors are possible according to the JVMS.

Using jdk.internal.org.objectweb.asm

```
import jdk.internal.org.objectweb.asm.ClassWriter;
import jdk.internal.org.objectweb.asm.Label;
import jdk.internal.org.objectweb.asm.MethodVisitor;
import static jdk.internal.org.objectweb.asm.Opcodes.*;

static void testG1_ASM() throws Exception {
  ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);
  cw.visit(V1_8, ACC_PUBLIC, "TestG1", null, "java/lang/Object", null);
  cw.visitField(ACC_PUBLIC | ACC_FINAL, "f", "I", null, null);
  MethodVisitor constr = cw.visitMethod(ACC_PUBLIC, "<init>", "(I)V", null, null
  Label l_try_beg = new Label(), l_try_end = new Label();
  Label l_catch = new Label(), l_end = new Label();
  constr.visitCode();
  constr.visitVarInsn(ALOAD, 0);
  constr.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V");
  constr.visitLabel(l_try_beg);
  constr.visitVarInsn(ALOAD, 0);
  constr.visitVarInsn(ILOAD, 1);
  constr.visitFieldInsn(PUTFIELD, "TestG1", "f", "I");
```

```
constr.visitFieldInsn(PUTFIELD, "TestG1", "f", "I");
constr.visitLabel(l_try_end);
constr.visitJumpInsn(GOTO, l_end);
constr.visitLabel(l_catch);
constr.visitVarInsn(ASTORE, 2);
constr.visitVarInsn(ALOAD, 0);
constr.visitIntInsn(BIPUSH, 42);
constr.visitFieldInsn(PUTFIELD, "TestG1", "f", "I");
constr.visitLabel(l_end);
constr.visitInsn(RETURN);
constr.visitTryCatchBlock(l_try_beg, l_try_end, l_catch, "java/lang/Exception"
// max stack and max locals are automatically computed (because of the
// 'ClassWriter.COMPUTE_FRAMES' option) in the ClassWriter constructor,
// but we need this call nevertheless in order for the computation to happen!
constr.visitMaxs(0, 0);
constr.visitEnd();

// Get the bytes of the class..
byte[] b = cw.toByteArray();
// ..and write them into a class file (for debugging)
FileOutputStream fos = new FileOutputStream("TestG1.class");
fos.write(b);
fos.close();
```

```
  // Load the newly created class..
  Final f = new Final();
  Class<?> testG1Class = f.defineClass("TestG1", b, 0, b.length);
  // ..get the constructor..
  Constructor c = testG1Class.getConstructor(int.class);
  // ..and create a new "TestG1" object
  Object testG1 = c.newInstance(42);
  Field int_f = testG1Class.getDeclaredField("f");
  System.out.println("testG1.f = " + int_f.getInt(testG1));
}
```

Compile with “javac javac -XDignore.symbol.file=true”

```
class TestH {
  private final int f;
  {
    foo();
  }
  public TestH(int f) {
    this.f = f;
    System.out.println(this + ".f = " + this.f);
  }
  public void foo() {
    System.out.println(this + ".f = " + this.f);
  }
}
new TestH(42);
```

```
TestH@15db9742.f = 0
TestH@15db9742.f = 42
```

SEI CERT: TSM01-J. Do not let the "this" reference escape during object construction!

```
class TestH2 {
  private static final int f;
  static {
    f = foo();
  }
  public TestH2() {
    System.out.println(TestH2.class + ".f = " + f);
  }
  static int foo() {
    System.out.println(TestH2.class + ".f = " + f);
    return 42;
  }
}
new TestH2();
```

```
class Final$TestH2.f = 0
class Final$TestH2.f = 42
```

JDK9 Bug: 8087161: Fails to start up initialize system class loader on unsupported charset

```java
class TestI {
  public final int f;
  public TestI(int f) {
    this.f = f;
  }
  public void set(int f) {
    // change value of 'f'
  }
}

TestI i = new TestI(42);
i.set(99);
System.out.println("i.f = " + i.f);
```

```
i.f = 99
```

Question: How can we change the value of 'f' in Java?

JNI

> *"The JNI does not enforce class, field, and method access control restrictions that can be expressed at the Java programming language level through the use of modifiers such as private and final."*
>
> The Java Native Interface: Programmer's Guide and Specification, §10.9 Violating Access Control Rules

```
class TestI {
  ...
  public native void set(int f);
  static {
    System.loadLibrary("TestI1");
  }
}
```

```
extern "C" JNIEXPORT
void JNICALL Java_TestI1_set(JNIEnv *env, jobject obj, jint val) {

  jclass thisClass = env->GetObjectClass(obj);

  jfieldID fid = env->GetFieldID(thisClass, "f", "I");

  env->SetIntField(obj, fid, val);
}
```

JDK9 RFE: 8058164: final fields in objects need to support inlining optimizations

REFLECTION

```
static class TestI {
  ...
  public void set(int f) throws Exception {
    java.lang.reflect.Field field = this.getClass().getDeclaredField("f");
    field.setAccessible(true);
    field.setInt(this, f);
  }
}
```

REFLECTION (STATIC FINAL FIELDS)

```
public class TestJ2 {
  public static final int f;
  static { f = 42; }
  ...
  public static void set(int f) throws Exception {
    java.lang.reflect.Field field = TestJ2.class.getDeclaredField("f");
    field.setAccessible(true);
    field.setInt(null, f);
  }

  public static void main(String[] args) throws Exception {
    System.out.println("TestJ2.f = " + TestJ2.f);
    TestJ2.set(99);
    System.out.println("TestJ2.f = " + TestJ2.f);
  }
}
```

REFLECTION (STATIC FINAL FIELDS)

```
Exception in thread "main" java.lang.IllegalAccessException:
  Can not set static final int field TestJ2.f to (int)99

at sun.reflect.UnsafeFieldAccessorImpl.throwFinalFieldIllegalAccessException(Uns
at sun.reflect.UnsafeFieldAccessorImpl.throwFinalFieldIllegalAccessException(Uns
at sun.reflect.UnsafeQualifiedStaticIntegerFieldAccessorImpl.setInt(UnsafeQuali1
at java.lang.reflect.Field.setInt(Field.java:949)
at TestJ2.set(TestJ2.java:19)
at TestJ2.main(TestJ2.java:24)
```

REFLECTION (STATIC FINAL FIELDS)

```
public class TestJ2 {
  public static final int f;
  static { f = 42; }
  ...
  public static void set(int f) throws Exception {
    java.lang.reflect.Field field = TestJ2.class.getDeclaredField("f");

    Field modifiersField = Field.class.getDeclaredField("modifiers");
    modifiersField.setAccessible(true);
    modifiersField.setInt(field, field.getModifiers() & ~Modifier.FINAL);

    // field.setAccessible(true);
    field.setInt(null, f);
  }
  ...
}
```

REFLECTION (?$!%§)

```
static class TestJ3 {
    ...
    public static void set(String s) throws Exception {
      java.lang.reflect.Field field = TestJ3.class.getDeclaredField("f");

      Field modifiersField = Field.class.getDeclaredField("modifiers");
      modifiersField.setAccessible(true);
      modifiersField.setInt(field, field.getModifiers() & ~Modifier.FINAL);

      Field typeField = Field.class.getDeclaredField("type");  // Change
      typeField.setAccessible(true);                           // the field's
      typeField.set(field, String.class);                      // type!

      field.setAccessible(true);
      field.set(null, s);
    ...
```

REFLECTION (?$!%§)

```
...
public static void main(String[] args) throws Exception {
  System.out.println("TestJ3.f = " + TestJ3.f);
  TestJ3.set("Volker");
  System.out.println("TestJ3.f = " + TestJ3.f);
}
```

```
TestJ3.f = 42
TestJ3.f = -667891199
```

JDK-8055530: assert(_exits.control()->is_top() || !_gvn.type(ret_phi)->empty()) failed:
return value must be well defined

REFLECTION (?$!%§)

```
# To suppress the following error report, specify this argument
# after -XX: or in .hotspotrc:  SuppressErrorAt=\g1SATBCardTableModRefBS.cpp:45
#
# A fatal error has been detected by the Java Runtime Environment:
#
#  Internal Error (C:\Software\OpenJDK\jdk9-dev\hotspot\src\share\vm\gc\g1\g1SA
#  assert(pre_val->is_oop(true)) failed: Error
#
# JRE version: OpenJDK Runtime Environment (9.0) (build 1.9.0-internal-debug-d04
# Java VM: OpenJDK 64-Bit Server VM (1.9.0-internal-d046063_2015_10_07_23_02-b0
# Core dump will be written. Default location: C:\Users\D046063\public_html\hots
#
# An error report file with more information is saved as:
# C:\Users\D046063\public_html\hotspot\Joker2015\examples\hs_err_pid21520.log
```

METHOD HANDLES

```
static class TestI {
  ...
  public void set(int f) throws Throwable {
    java.lang.reflect.Field field = this.getClass().getDeclaredField("f");
    field.setAccessible(true);
    java.lang.invoke.MethodHandle setter =
      java.lang.invoke.MethodHandles.lookup().unreflectSetter(field);
    setter.invokeExact(this, f);
  }
}
```

sun.misc.Unsafe

```
static class TestI {
  ...
  static sun.misc.Unsafe UNSAFE;
  static {
    try {
      java.lang.reflect.Constructor<sun.misc.Unsafe> unsafeConstructor =
        sun.misc.Unsafe.class.getDeclaredConstructor();
      unsafeConstructor.setAccessible(true);
      UNSAFE = unsafeConstructor.newInstance();
    } catch (Exception e) {}
  }
  public void set(int f) throws Exception {
    java.lang.reflect.Field field = this.getClass().getDeclaredField("f");
    UNSAFE.putInt(this, UNSAFE.objectFieldOffset(field), f);
  }
}
```

```
static class TestM {
  public final int f = 42;
  public TestM() {

  }
  public void set(int f);  // Change 'f'
}

TestM testM = new TestM();
System.out.println((testM.f == 42 ? "Unchanged " : "Changed   ") +
                   TestM.class.getDeclaredField("f").getInt(testM));
testM.set(99);
System.out.println((testM.f == 42 ? "Unchanged " : "Changed   ") +
                   TestM.class.getDeclaredField("f").getInt(testM));
```

```
Unchanged 42
Unchanged 99
```

```
...
23: bipush        42
25: bipush        42
27: if_icmpne     35
30: ldc           #14 // String Unchanged
32: goto          37
35: ldc           #15 // String Changed
37: invokevirtual #16 // Method java/lang/StringBuilder.append()
...
```

```java
static class TestM {
  public final int f;
  public TestM() {
    f = 42;
  }
  public void set(int f);  // Change 'f'
}

TestM testM = new TestM();
System.out.println((testM.f == 42 ? "Unchanged " : "Changed   ") +
                   TestM.class.getDeclaredField("f").getInt(testM));
testM.set(99);
System.out.println((testM.f == 42 ? "Unchanged " : "Changed   ") +
                   TestM.class.getDeclaredField("f").getInt(testM));
```

```
Unchanged 42
Changed   99
```

```
...
18: aload_1            // Object testM:TestM
19: getfield      #2   // Field f:I
22: bipush        42
24: if_icmpne     32
27: ldc           #13  // String Unchanged
29: goto          34
32: ldc           #14  // String Changed
34: invokevirtual #15  // Method java/lang/StringBuilder.append()
...
```

CONSTANT VARIABLES

> *"We call a variable, of primitive type or type String, that is `final` and initialized with a compile-time constant expression a constant variable."*
>
> The Java Language Specification, §4.12.4. final Variables

> *"References to fields that are constant variables are resolved at compile time to the constant value that is denoted. No reference to such a constant field should be present in the code..."*
>
> The Java Language Specification, §13.1. The Form of a Binary

Constant variables are inlined by `javac` !

J. Bloch, N. Gafter - "Java Puzzlers", Nr. 93

```
class Test {
  static final Test test =
    new Test(42);
  static void set_test(Test t);
  final int f;
  void set_f(int f);

  public Test(int f) {
    this.f = f;
  }

  public static int get_f() {
    if (test.f == 42) {
      return 42;
    }
    return test.f;
  }
  ...
}
```

```
System.out.println(Test.get_f());  // -> 42

Test.set_test(new Test(99));
System.out.println(Test.get_f());  // -> 99

Test test42 = new Test(42);
Test.set_test(test42);
for (int n = 0; n < 20000; n++)
  if (Test.get_f() != 42)          // -> 42
    System.out.println("!!!");

Test.set_test(new Test(99));
System.out.println(Test.get_f());  // -> 42

Test.test.set_f(99);
System.out.println(Test.get_f());  // -> 42

test42.set_f(99);
System.out.println(Test.get_f());  //  ???
```

With -XX:-TrustFinalNonStaticFields -> 99

```
0x00007f5a6cd674e0: mov    %eax,-0x16000(%rsp) ; stack bang
0x00007f5a6cd674e7: push   %rbp                ; push frame
0x00007f5a6cd674e8: sub    0x10,%rsp

0x00007f5a6cd674ec: mov    0xeb6ac950,%r10     ; {oop(a 'Test')}
0x00007f5a6cd674f6: mov    0xc(%r10),%r11d     ;*getfield f
                                               ; - Test::get_f@3
0x00007f5a6cd674fa: cmp    0x2a,%r11d          ; 42
0x00007f5a6cd674fe: jne    0x00007f5a6cd67511  ;*if_icmpne
                                               ; - Test::get_f@8
0x00007f5a6cd67500: mov    0x2a,%eax           ; 42
0x00007f5a6cd67505: add    0x10,%rsp
0x00007f5a6cd67509: pop    %rbp                ; pop frame
0x00007f5a6cd6750a: test   %eax,0xa7d5af0(%rip); 0x00007f5a7753d000
                                               ;   {poll_return}
0x00007f5a6cd67510: ret
```

With -XX:+TrustFinalNonStaticFields -> 42

```
0x00007f6b01196e60: mov    %eax,-0x16000(%rsp) ; stack bang
0x00007f6b01196e67: push   %rbp                ; push frame
0x00007f6b01196e68: sub    0x10,%rsp

0x00007f6b01196e6c: mov    0x2a,%eax           ; 42
0x00007f6b01196e71: add    0x10,%rsp
0x00007f6b01196e75: pop    %rbp                ; pop frame
0x00007f6b01196e76: test   %eax,0xc55f184(%rip); 0x00007f6b0d6f6000
                                               ;   {poll_return}
0x00007f6b01196e7c: ret
```

JDK 7: 6912065: final fields in objects need to support inlining optimizations for JSR 292

By default only enabled for `java.lang.invoke` and `sun.invoke`.

The `@Stable` Annotation

- Only available for `java.lang.invoke` package.
- Treat fields as final if they change their value at most once.
- `-XX:+FoldStableValues`
- `-XX:+UseImplicitStableValues`
- 8001107: @Stable annotation for constant folding of lazily evaluated variables
- 8024042: Add verification support for @Stable into VM
- 8134758: Final String field values should be trusted as stable