```
  #19 = Utf8               [Ljava/lang/String;
  #20 = Utf8               SourceFile
  #21 = Utf8               TestFinalClass.java
  #22 = NameAndType        #9:#10         // "<init>":()V
  #23 = NameAndType        #5:#6          // num:I
  #24 = Utf8               com/nx/day06/TestFinalClass
  #25 = Utf8               java/lang/Object
{
  public final int num;
    descriptor: I
    flags: ACC_PUBLIC, ACC_FINAL
    ConstantValue: int 10

  public com.nx.day06.TestFinalClass();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/lang/Object."<init>":()V
         4: aload_0
         5: bipush        10
         7: putfield      #2                  // Field num:I
        10: return
      LineNumberTable:
        line 3: 0
        line 5: 4
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
```

```
Local  Local (1)

  public com.nx.day06.TestFinalClass();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/lang/Object."<init>":()V
         4: aload_0
         5: bipush        10
         7: putfield      #2                  // Field num:I
        10: return
      LineNumberTable:
        line 3: 0
        line 5: 4
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0      11     0  this   Lcom/nx/day06/TestFinalClass;

  public static void main(java.lang.String[]);
    descriptor: ([Ljava/lang/String;)V
    flags: ACC_PUBLIC, ACC_STATIC
    Code:
      stack=0, locals=1, args_size=1
         0: return
      LineNumberTable:
        line 9: 0
      LocalVariableTable:
        Start  Length  Slot  Name   Signature
            0       1     0  args   [Ljava/lang/String;
}
```

src\share\vm\utilities\accessFlags.hpp

```
bool is_final       () const         { return (_flags & JVM_ACC_FINAL       ) != 0; }
```

src\cpu\x86\vm\templateInterpreter_x86.cpp

```
void TemplateTable::putfield(int byte_no) {
  putfield_or_static(byte_no, false);
}
void TemplateTable::putfield_or_static(int byte_no, bool is_static) {
```

```
      ..........................................
      __ bind(Done);

// Check for volatile store
__ testl(rdx, rdx);
__ jcc(Assembler::zero, notVolatile);
volatile_barrier(Assembler::Membar_mask_bits(Assembler::StoreLoad |
                                             Assembler::StoreStore));
__ bind(notVolatile);
}

// volatile-store-volatile-load case.  This final case is placed after
// volatile-stores although it could just as well go before
// volatile-loads.
void TemplateTable::volatile_barrier(Assembler::Membar_mask_bits
                                     order_constraint) {
  // Helper function to insert a is-volatile test and memory barrier
  if (os::is_MP()) { // Not needed on single CPU
    __ membar(order_constraint);
  }
}
```

src/cpu/x86/vm/assembler_x86.hpp

```
// Serializes memory and blows flags
void membar(Membar_mask_bits order_constraint) {
  if (os::is_MP()) {
    // We only have to handle StoreLoad
    if (order_constraint & StoreLoad) {
      // All usable chips support "locked" instructions which suffice
      // as barriers, and are much faster than the alternative of
      // using cpuid instruction. We use here a locked add [esp],0.
      // This is conveniently otherwise a no-op except for blowing
      // flags.
      // Any change to this code may need to revisit other places in
      // the code where this idiom is used, in particular the
      // orderAccess code.
```

```
      lock();
      addl(Address(rsp, 0), 0);// Assert the lock# signal here
    }
  }
}
```