MICROOH 麦可网

Android-从程序员到架构师之路

出品人：Sundy
讲师：高焕堂（台湾）

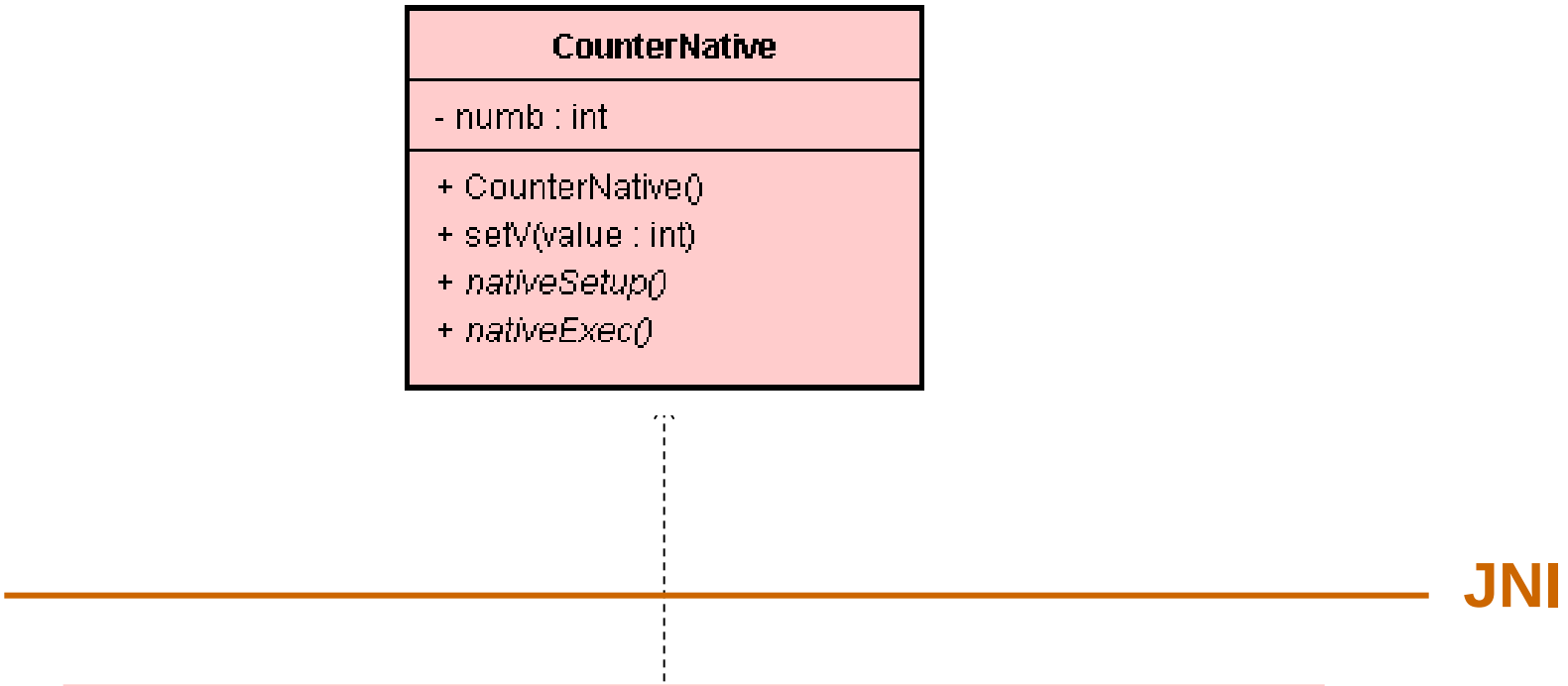
http://www.microoh.com

C03_f

JNI：从C调用Java函数 (f)

By 高煥堂

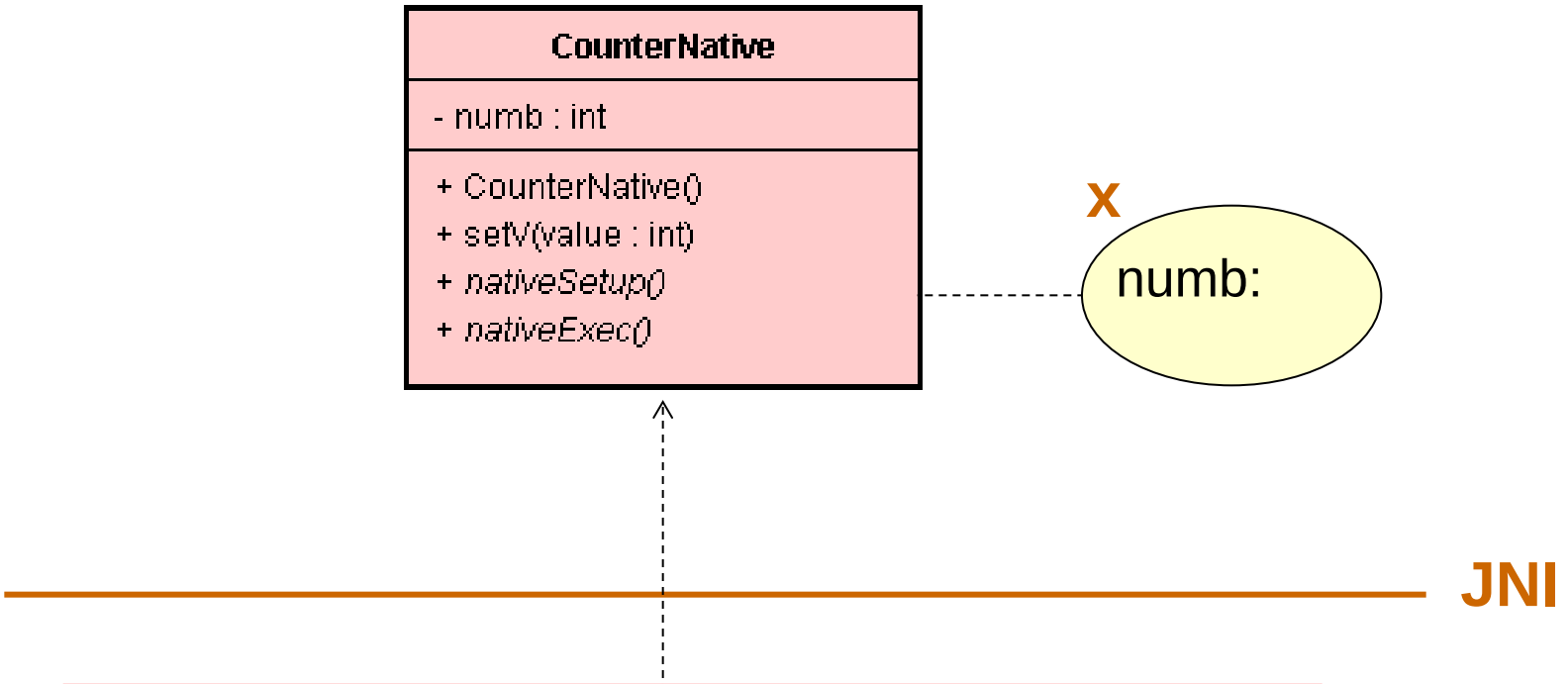
前言




```
/*  com.misoo.counter.CounterNative.c  */
// .....
JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeExec
        (JNIEnv *env, jobject thiz) {
                //.....
}
```

在Java层创建1个对象

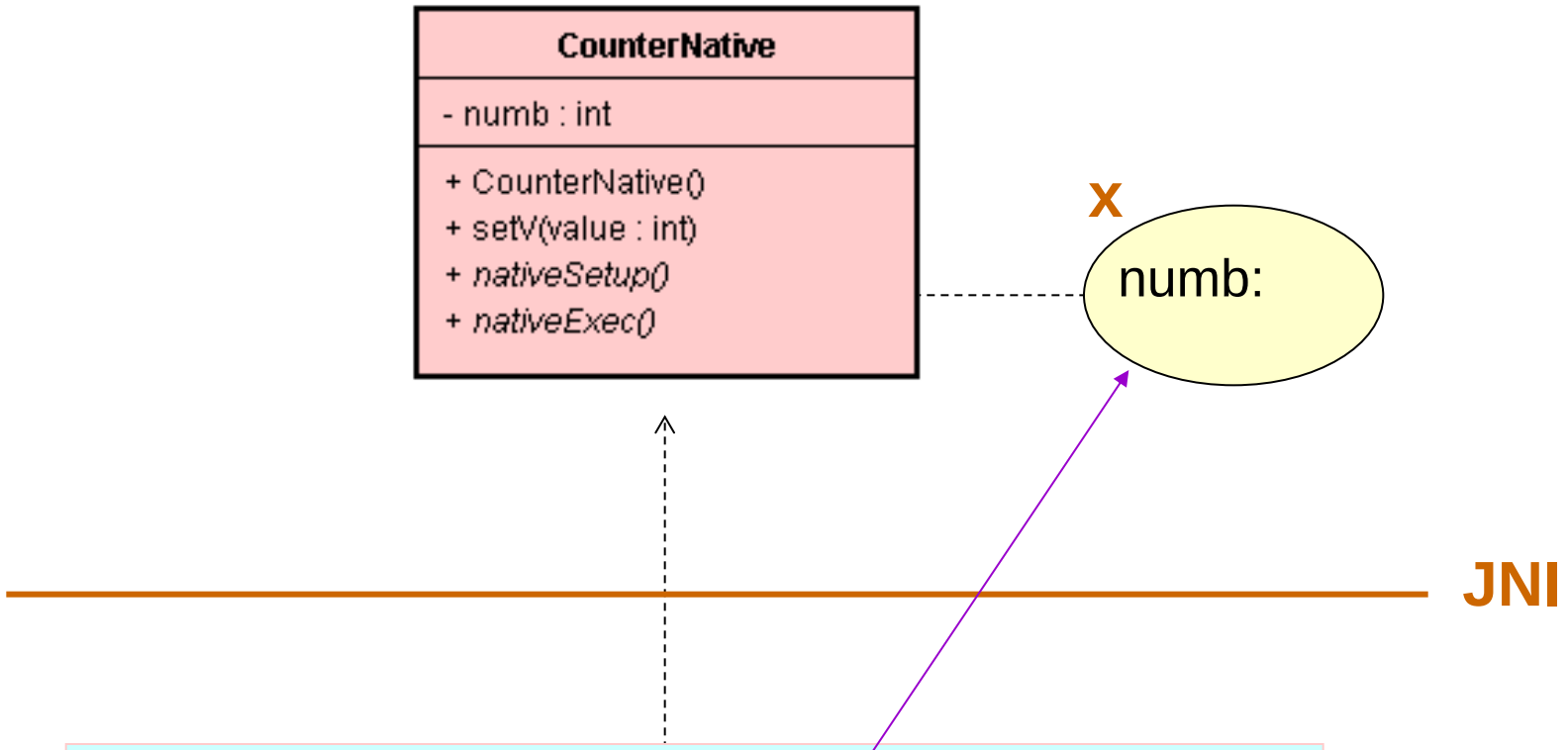
```
CounterNative x;
x = new CounterNative();
```



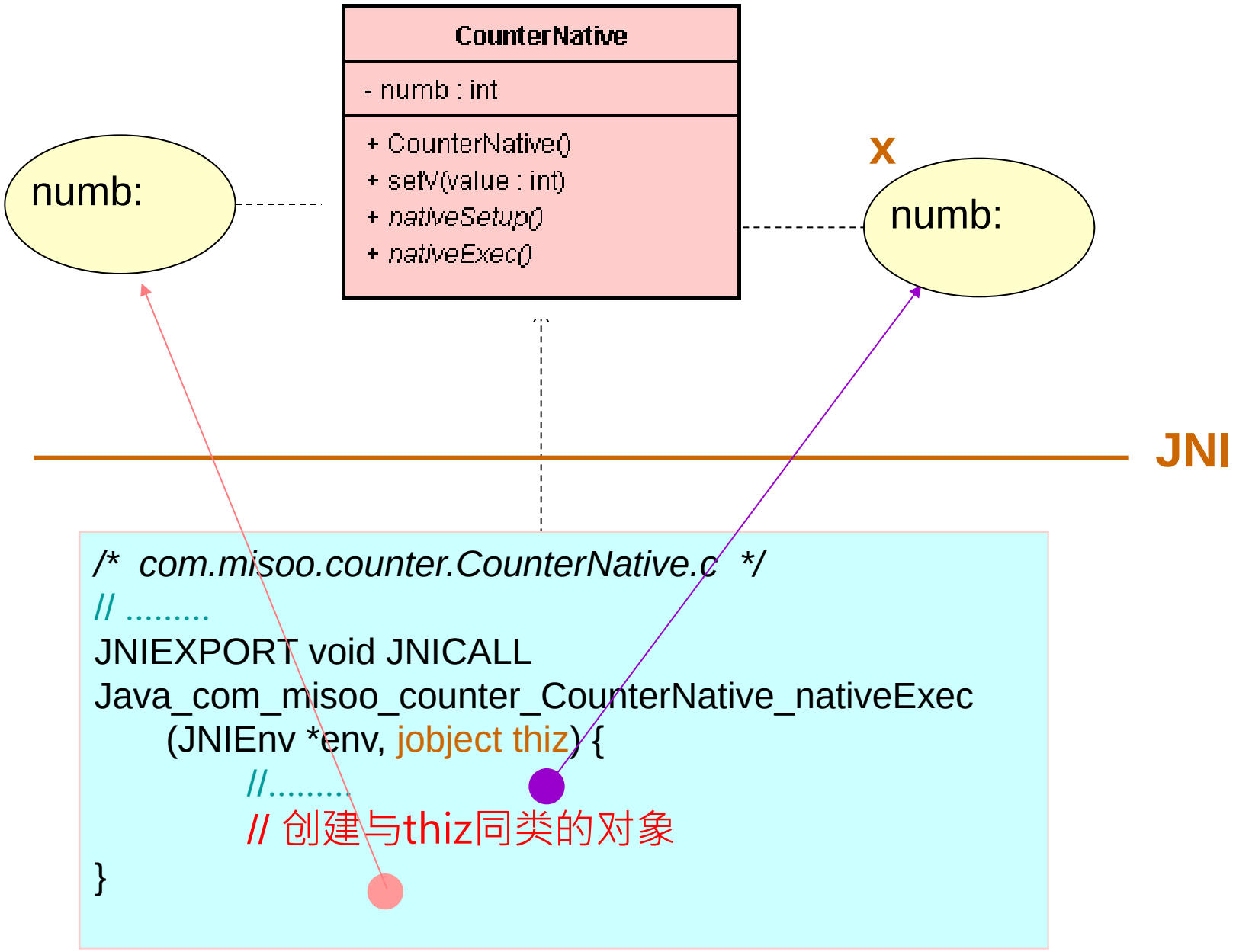

```
/*  com.misoo.counter.CounterNative.c  */
// .....
JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeExec
      (JNIEnv *env, jobject thiz) {
            //.....
}
```

从Java函数调用C函数

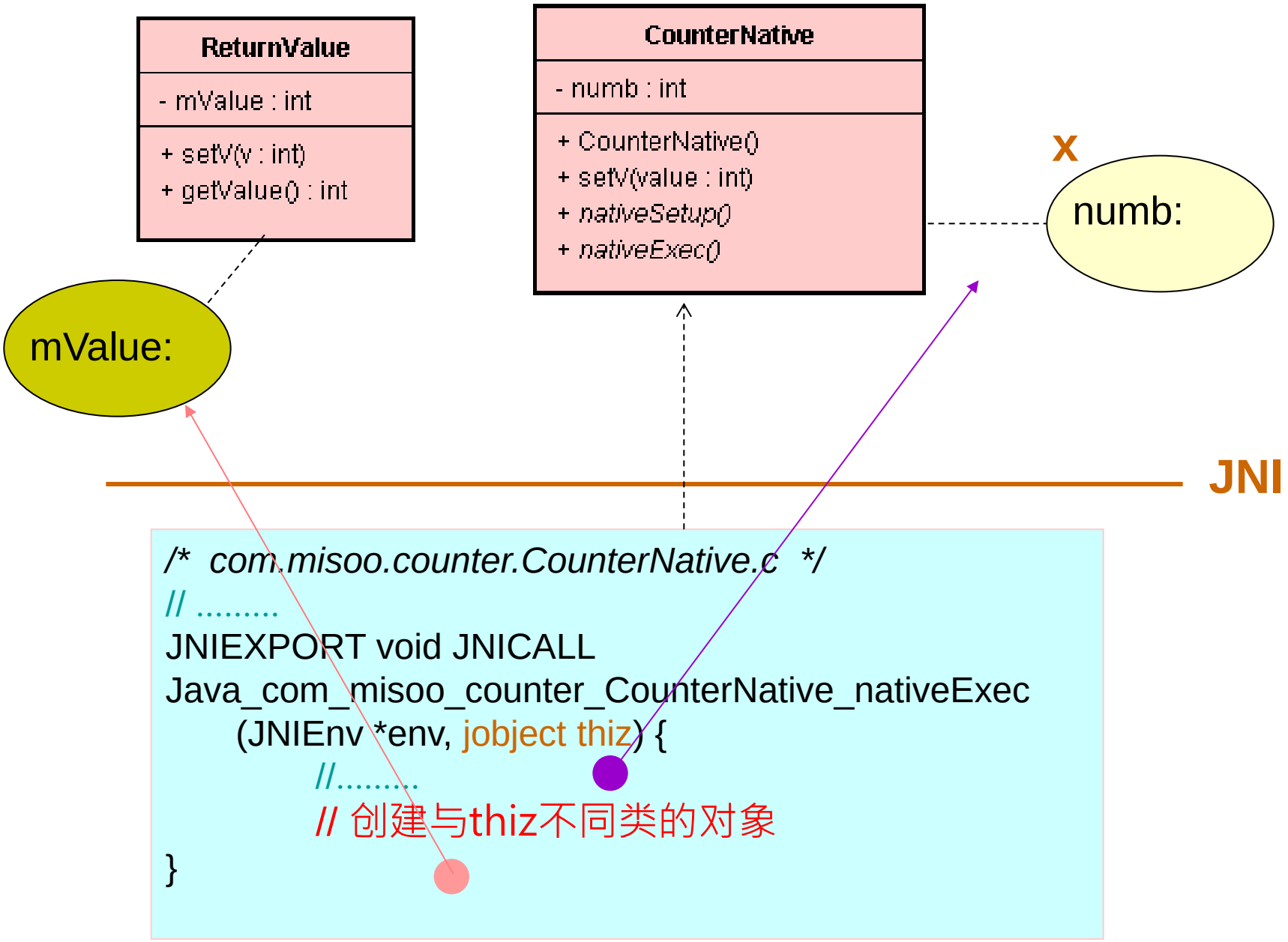
```
x.nativeExec();
```




```
/*  com.misoo.counter.CounterNative.c  */
// .....
JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeExec
      (JNIEnv *env, jobject thiz) {
            //.....
}
```


CounterNative
- numb : int
+ CounterNative()
+ setV(value : int)
+ nativeSetup()
+ nativeExec()
numb:
X
numb:
JNI
/* com.misoo.counter.CounterNative.c */
// .....
JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeExec
(JNIEnv *env, jobject thiz) {
//.....
// 创建与thiz同类的对象
}

1. 问这个对象thiz的类，得到clazz。
2. 问这个类里的<init>()构造式，
   得到methodID。
3. 基于methodID，调用构造式(创建对象)。


ReturnValue
- mValue : int
+ setV(v : int)
+ getValue() : int
CounterNative
- numb : int
+ CounterNative()
+ setV(value : int)
+ nativeSetup()
+ nativeExec()
x
numb:
mValue:
JNI
/* com.misoo.counter.CounterNative.c */
// .....
JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeExec
(JNIEnv *env, jobject thiz) {
//.....
// 创建与thiz不同类的对象
}

1.问特定的类，得到clazz。

```
jclass clazz = (*env)->FindClass(env,
                    "com/misoo/counter/ResultValue");
```

2. 问这个类里的<init>()构造式，
得到methodID。

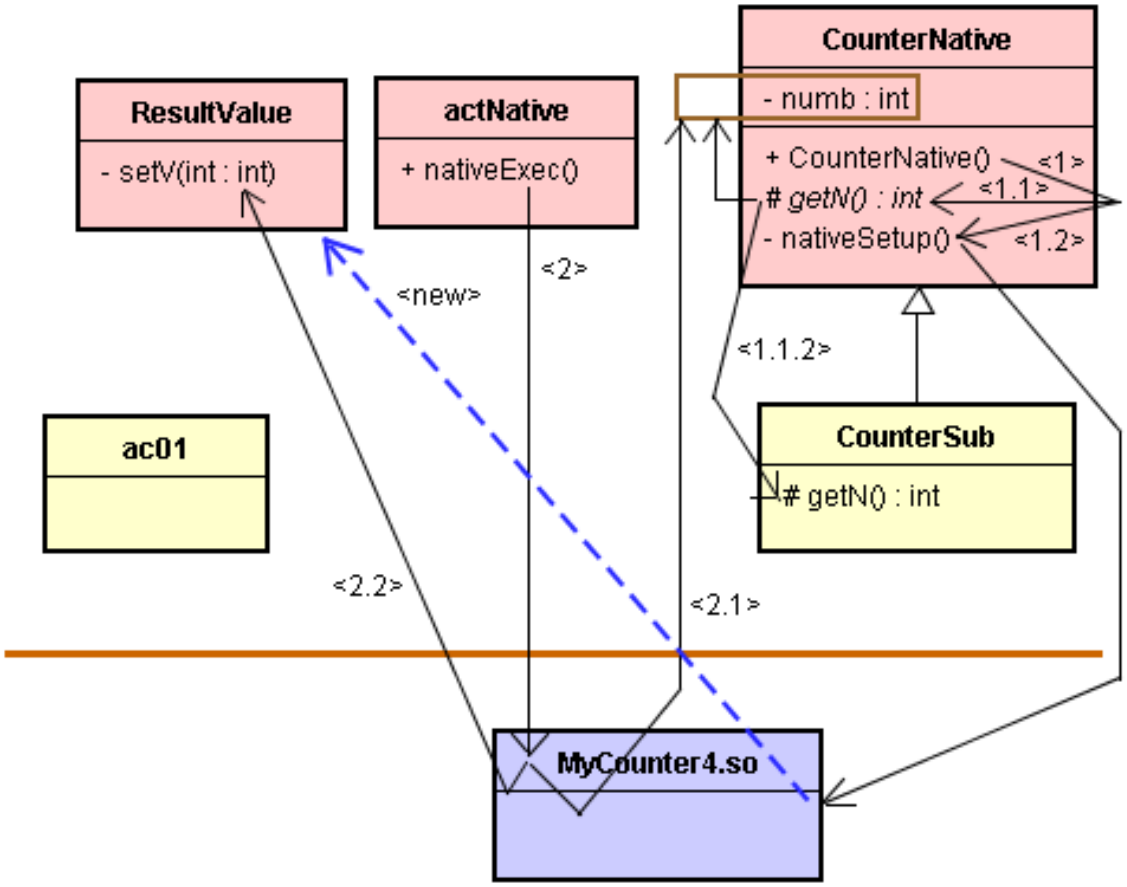
```
jmethodID constr =
        (*env)->GetMethodID(env, clazz,   "<init>", "()V");
```

3. 基于methodID，调用构造式(创建对象)。

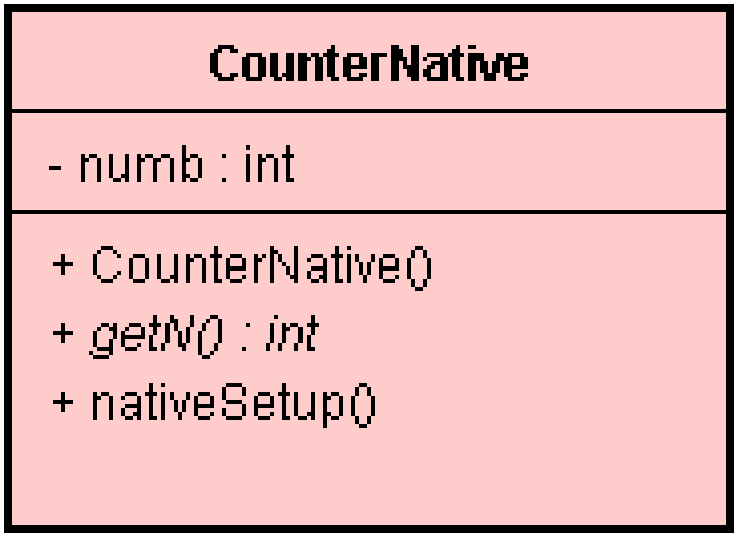
```
jobject ref = (*env)->NewObject(env, clazz, constr);
```

- 创建与Thiz同类的对象，对控制点的意义不大。因为Java层已经创建该类的对象，无法防止了。
- 创建与Thiz不同类的对象，有很大的控制涵意。

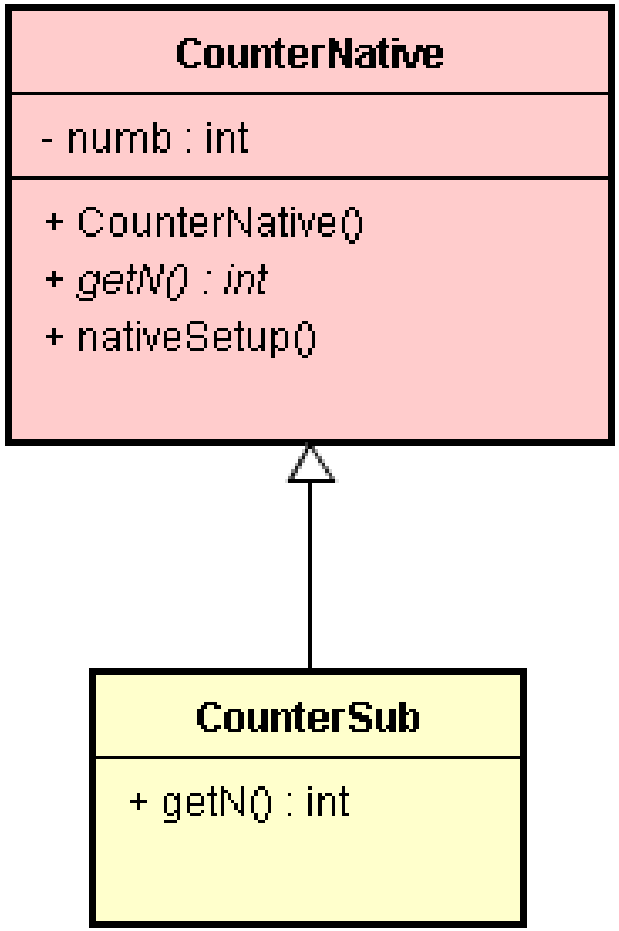
范例


CounterNative
- numb : int
+ CounterNative()
# getN() : int
- nativeSetup()
<1>
<1.1>
<1.2>
ResultValue
- setV(int : int)
actNative
+ nativeExec()
<new>
<2>
<1.1.2>
ac01
CounterSub
getN() : int
<2.2>
<2.1>
MyCounter4.so

范例代码


CounterNative
- numb : int
+ CounterNative()
+ getN() : int
+ nativeSetup()

- 这CounterNative类别里定义了1个抽象函数，以及1个本地函数。
- 抽象函数是由子类来实作；而本地函数则由C模块来实作。如下：

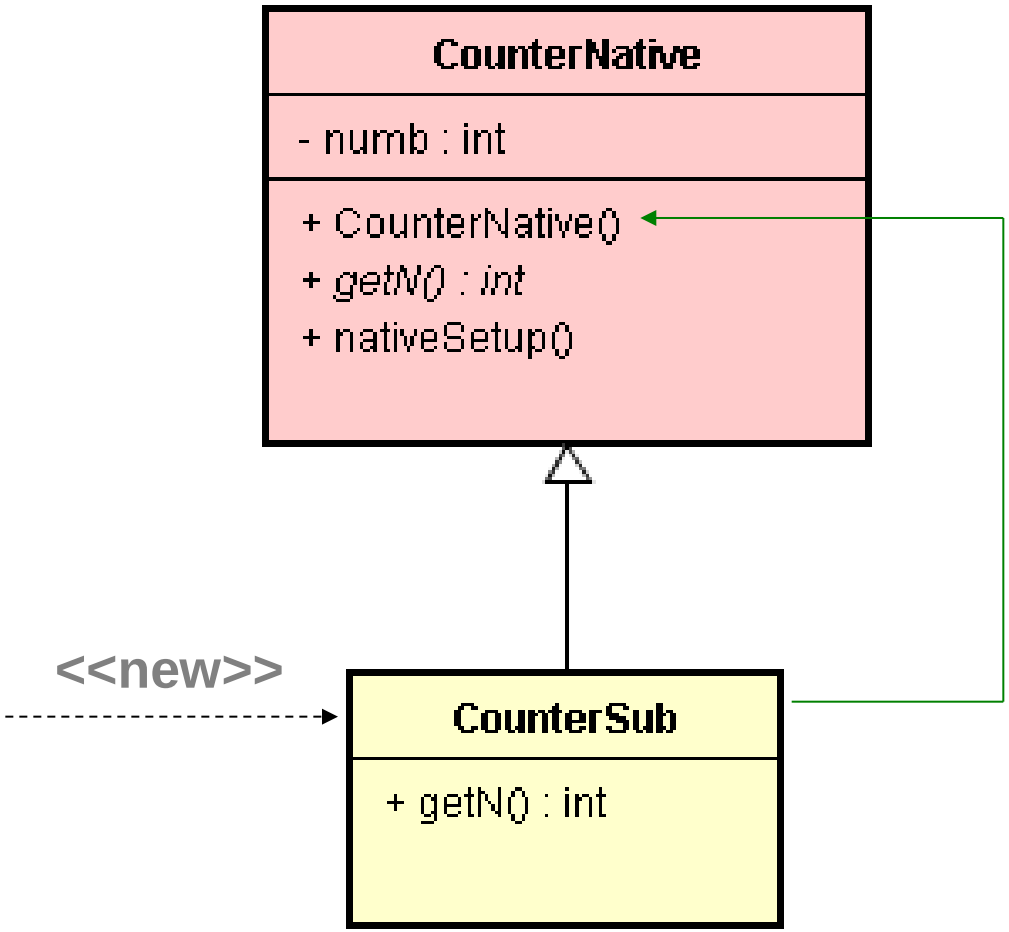

CounterNative
- numb : int
+ CounterNative()
+ getN() : int
+ nativeSetup()
CounterSub
+ getN() : int

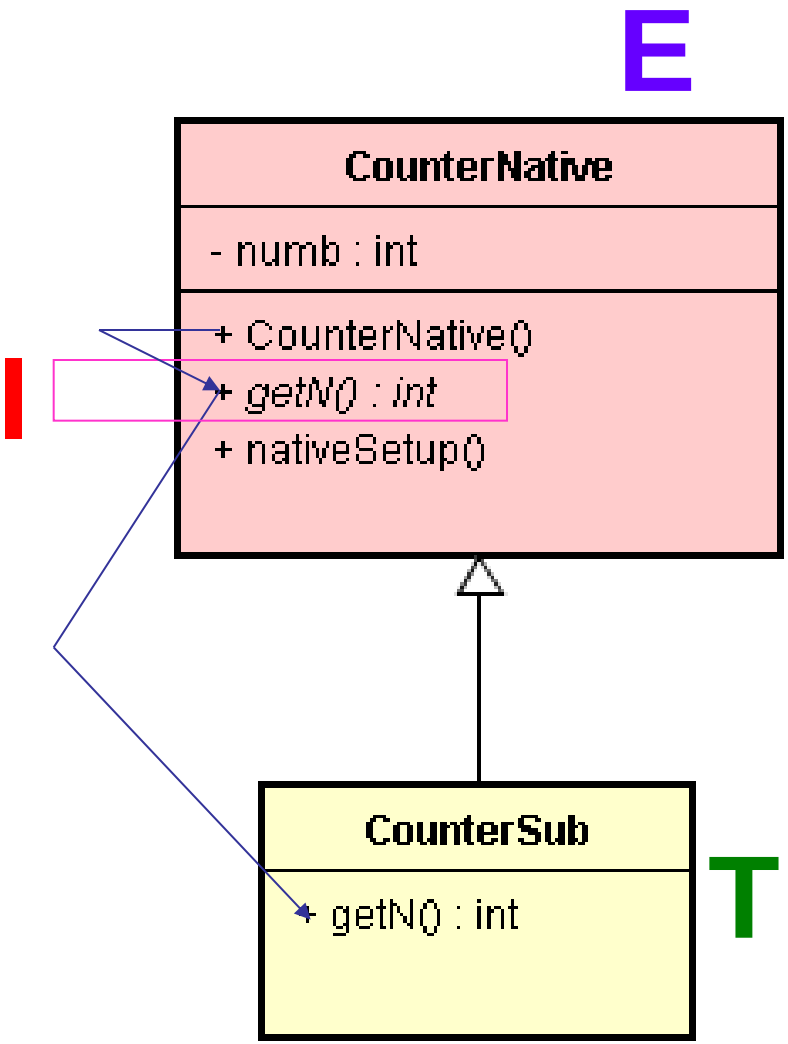
```
// CounterSub.java
// ……
public class CounterSub extends CounterNative{
        protected int getN()
                {  return 10;        }
}
```

```
// ac01.java
// ..........
public class ac01 extends Activity implements OnClickListener {
      private CounterNative cn;
      @Override
      public void onCreate(Bundle savedInstanceState){
            // .............
            cn = new CounterSub();     }
      @Override public void onClick(View v) {
             switch(v.getId()){
             case 101:  ResultValue rvObj
                                  = (ResultValue)actNative.nativeExec();
                              setTitle("Value = " + rvObj.getValue());   break;
             case 103:   finish();  break;
}}}
```

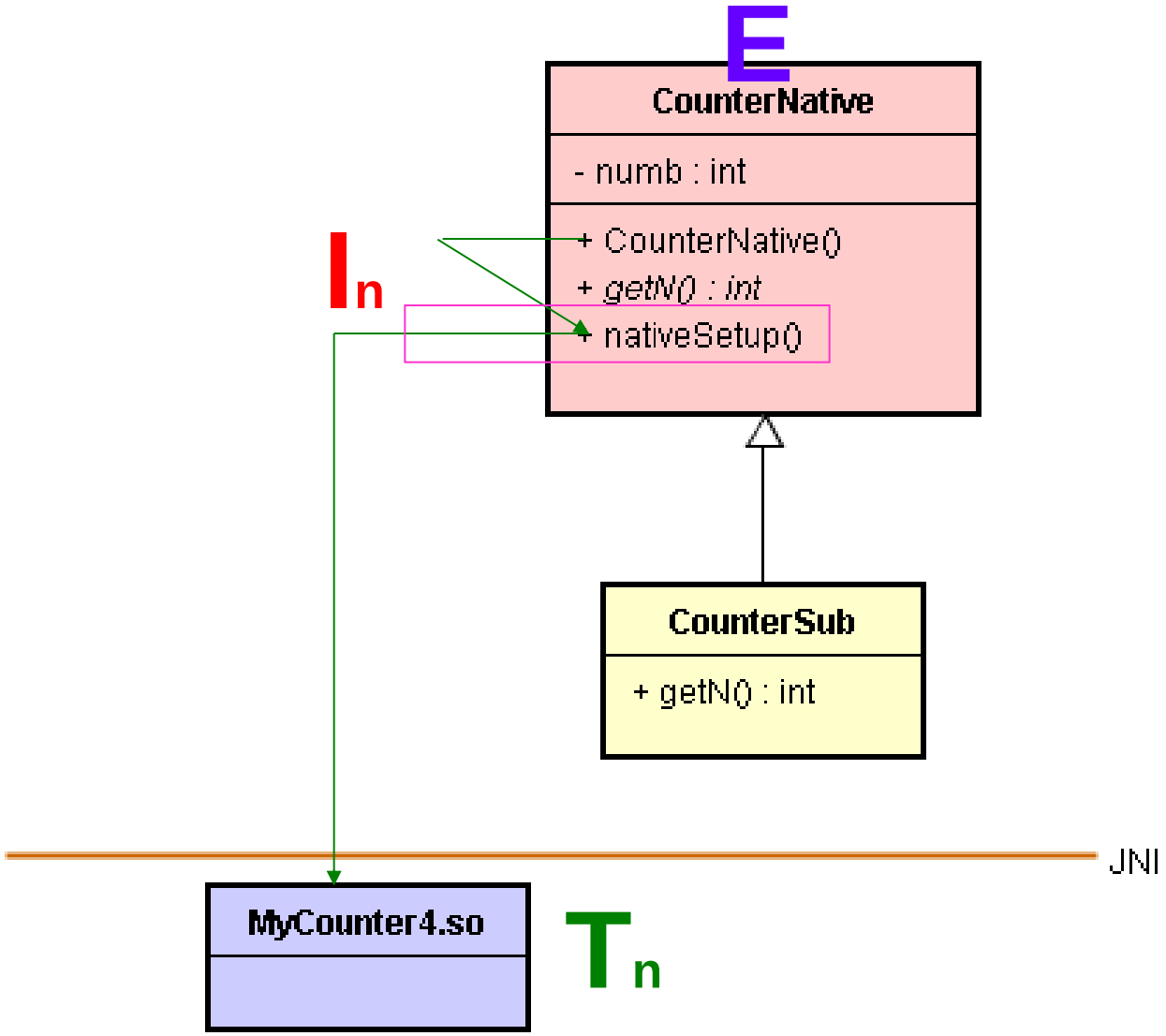
cn = new CounterSub();

- 就调用CounterSub子类别的建构函数，其调用CounterNative()父类别的建构函数。
- 此刻，先调用子类别的getN()函数，取得numb属性值。

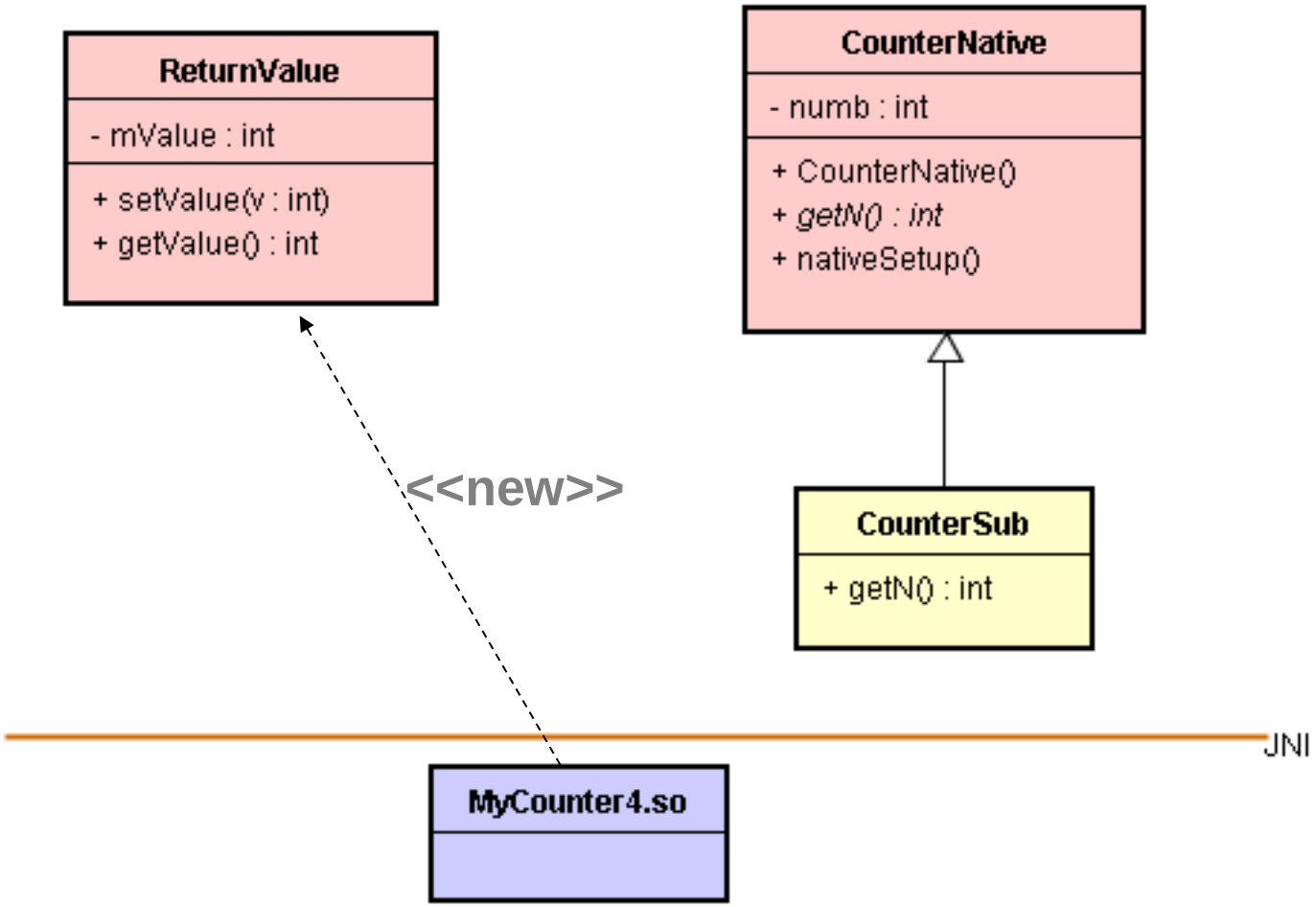

CounterNative
- numb : int
+ CounterNative()
+ getN() : int
+ nativeSetup()
<<new>>
CounterSub
+ getN() : int


E
CounterNative
- numb : int
+ CounterNative()
+ getN() : int
+ nativeSetup()
CounterSub
+ getN() : int
T

接着，调用本地的C函数


E
CounterNative
- numb : int
+ CounterNative()
+ getN() : int
+ nativeSetup()
In
CounterSub
+ getN() : int
JNI
MyCounter4.so
Tn

```
// CounterNative.java
// ........
abstract public class CounterNative {
      private int numb;
      static {
                System.loadLibrary("MyCounter5");   }
      public CounterNative(){
                  numb = getN();
                  nativeSetup();
            }
      abstract protected int getN();
      private native void nativeSetup();
}
```


ReturnValue
- mValue : int
+ setValue(v : int)
+ getValue() : int
CounterNative
- numb : int
+ CounterNative()
+ getN() : int
+ nativeSetup()
<<new>>
CounterSub
+ getN() : int
JNI
MyCounter4.so

- 透过VM而调用ResultValue类的建构函数去诞生ResultValue对象。

```
/* com.misoo.counter.Counter.c */
#include <android/log.h>
#include "com_misoo_counter_actNative.h"
#include "com_misoo_counter_CounterNative.h"
jobject   m_object, m_rv_object ;
jfieldID  m_fid;
jmethodID m_rv_mid;

JNIEXPORT void JNICALL
Java_com_misoo_counter_CounterNative_nativeSetup
    (JNIEnv *env, jobject thiz) {
        jclass clazz = (*env)->GetObjectClass(env, thiz);
        m_object = (jobject)(*env)->NewGlobalRef(env, thiz);
        m_fid = (*env)->GetFieldID(env, clazz, "numb", "I");
```

```
    jclass rvClazz = (*env)->FindClass(env,
                        "com/misoo/counter/ResultValue");
    jmethodID constr =
        (*env)->GetMethodID(env, rvClazz, "<init>", "()V");

    jobject ref = (*env)->NewObject(env, rvClazz, constr);

    m_rv_object = (jobject)(*env)->NewGlobalRef(env, ref);
    m_rv_mid
        = (*env)->GetMethodID(env, rvClazz, "setV", "(I)V");
    return;
}
```

```
JNIEXPORT jobject JNICALL
Java_com_misoo_counter_actNative_nativeExec
        (JNIEnv *env, jclass clazz) {
         int n, i, sum = 0;

         n = (int)(*env)->GetObjectField(env, m_object, m_fid);

         for(i=0; i<=n; i++)
                  sum+=i;

         (*env)->CallVoidMethod(env, m_rv_object, m_rv_mid, sum);

         return m_rv_object;
}
```

- 上述nativeSetup()函数里的指令：

```
jclass rvClazz = (*env)->FindClass(env,
                    "com/misoo/counter/ResultValue");
```

- 直接把"com/misoo/counter/ResultValue" 字符串写进去，VM的就可帮忙找到ResultValue类的参考(存于rvClazz内)。

- 执行指令：

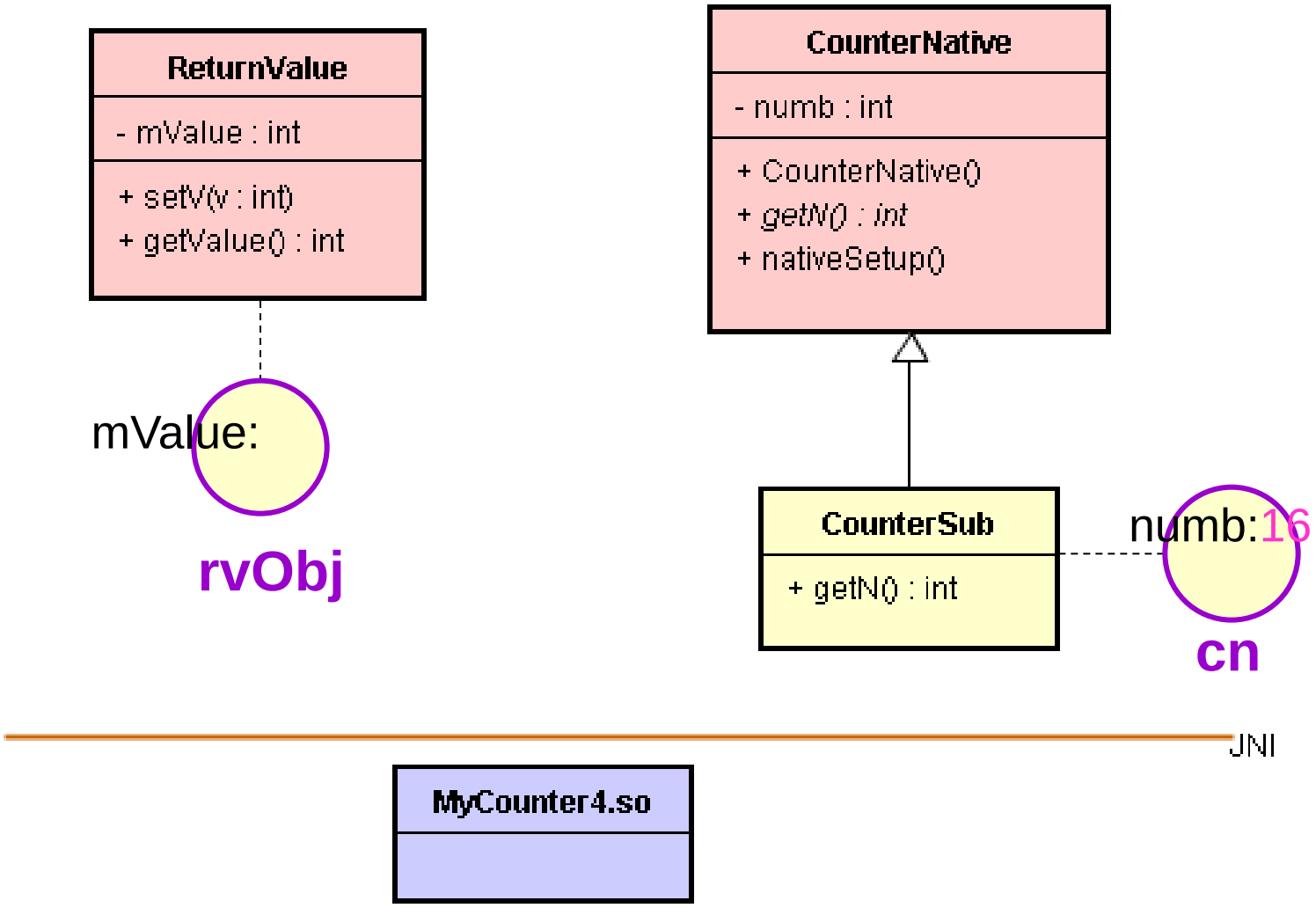
```
jmethodID constr = (*env)->GetMethodID(env,
                        rvClazz, "<init>", "()V");
```

- <init> 符号就代表构造式，VM的GetMethodID()函数取得构造式的ID，存于constr内。

- 执行到指令：

```
jobject ref =
    (*env)->NewObject(env, rvClazz, constr);
```

- 此时rvClazz代表ResultVlaue类，而constr是ResultValue类的构造式。
- 于是，以rvClazz和constr两者为参数，调用VM的NewObject()函数，诞生一个ResultVlaue对象了。


ReturnValue
- mValue : int
+ setV(v : int)
+ getValue() : int
mValue:
rvObj
CounterNative
- numb : int
+ CounterNative()
+ getN() : int
+ nativeSetup()
CounterSub
+ getN() : int
numb:16
cn
JNI
MyCounter4.so

- NewObject()诞生ResultVaue对象后，会将该对象参考回传给C模块。
- 由于Java层并没有这新对象的参考，所以此刻nativeExce()函数里的指令：

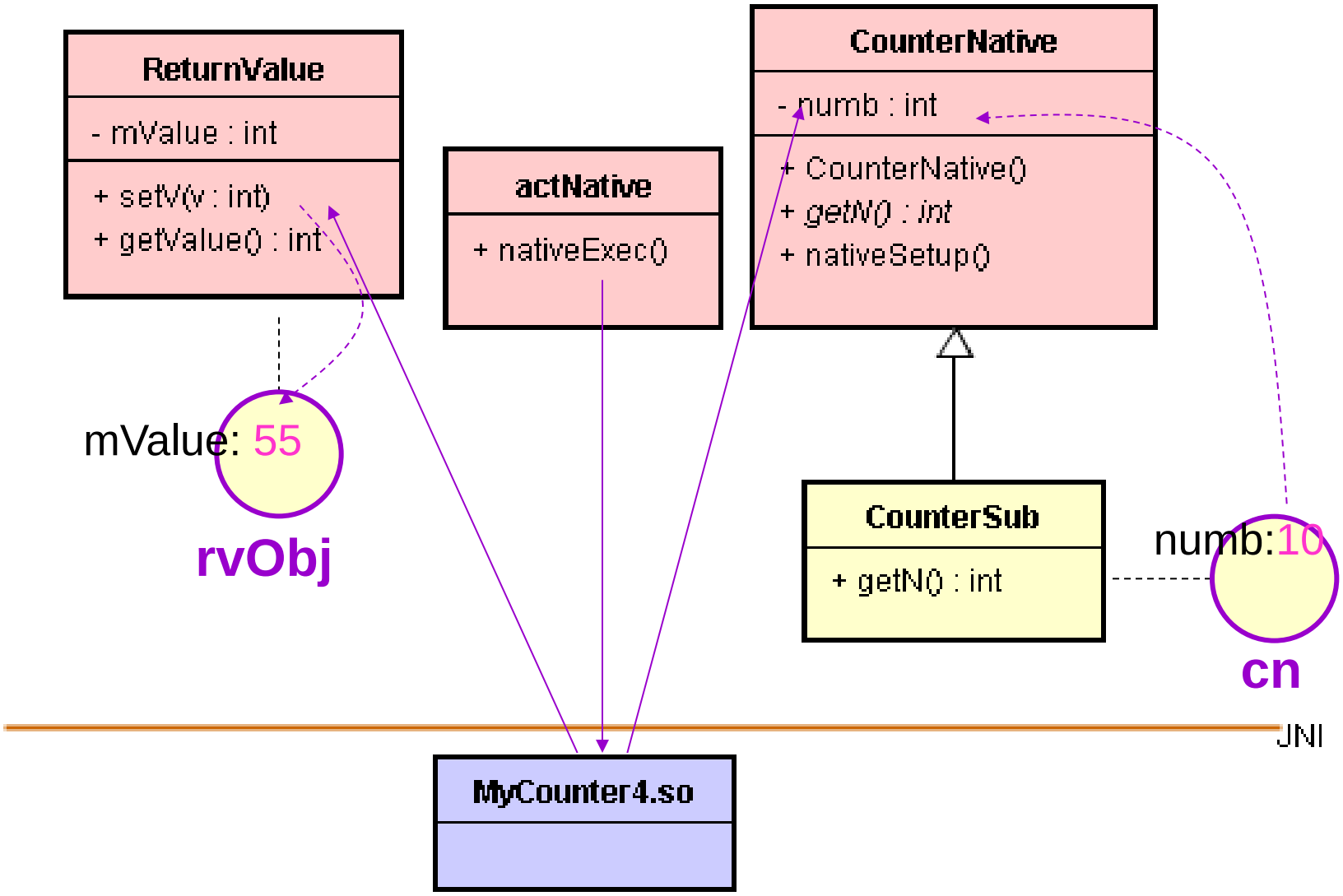
```
return m_rv_object;
```

- 就将新对象参考传递给Java层，让ac01类别能顺利读取对象里的数据。

```
// ResultValue.java
// ........
public class ResultValue {
        private int mValue;
        private void setV(int value)
                {   mValue = value;   }
        public int getValue()
                {   return mValue;   }
}
```

actNative.nativeExec();

```
// ac01.java
// ..........
public class ac01 extends Activity implements OnClickListener {
@Override public void onClick(View v) {
        // ........
        switch(v.getId()){
        case 101:  ResultValue rvObj
                        = (ResultValue) actNative.nativeExec();
                   setTitle("Value = " + rvObj.getValue());   break;
        case 103:   finish();  break;
}}}
```


ReturnValue
- mValue : int
+ setV(v : int)
+ getValue() : int
actNative
+ nativeExec()
CounterNative
- numb : int
+ CounterNative()
+ getN() : int
+ nativeSetup()
CounterSub
+ getN() : int
mValue: 55
rvObj
numb:10
cn
JNI
MyCounter4.so

```
// actNative.java
// ......
public class actNative {
        public static native Object nativeExec();
}
```

```
JNIEXPORT jobject JNICALL
Java_com_misoo_counter_actNative_nativeExec
    (JNIEnv *env, jclass clazz) {
```

结语

- 由于Android是开源开放的平台，我们才有将控制点往下，移到C/C++层的机会。
- 当你使用手机时，你所摸的都是硬件，例如触摸屏、键盘等。
- 你从来没有摸过软件，信不信，不然你说说软见摸起来感觉如何？摸起来像猫咪？像海绵？

- 因此C/C++层代码比Java层代码更早侦测到用户的事件，所以控制点往下移到C/C++层有效促进软硬整合，让硬件的创新迅速浮现出来，与Java层App代码紧密结合。

Thanks...

高煥堂