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

C01_e

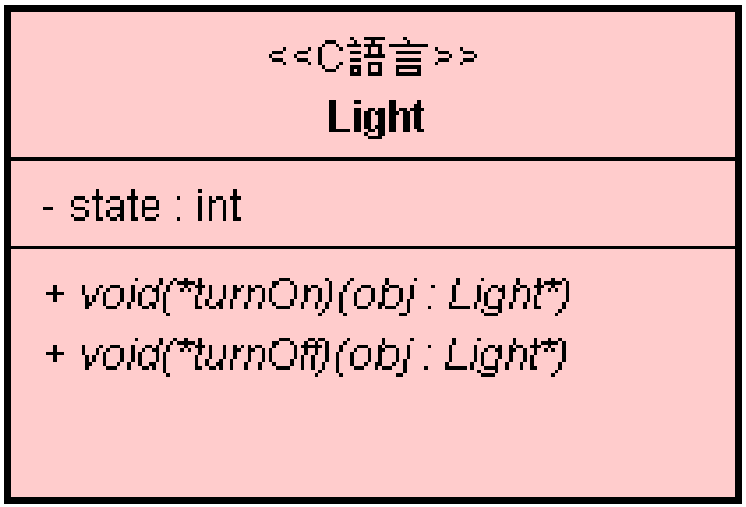
JNI架构原理：Java与C的对接(e)

By 高煥堂

EIT造形观点

- 基于熟悉的EIT造形，很容易理解重要的架构设计决策议题。

前言




```
// a.so 檔案(File)
void turnOn( Light *obj )
   {   obj->state = 1;   }
void turnOff( Light *obj )
   {   obj->state = 0;   }
```

C定義Light類

```
typedef struct Light Light;
struct Light {
    int state;
    void (*turnOn)(Light*);
    void (*turnOff)(Light*);   };
```

對接

C函數

```
// a.so 檔案(File)
void turnOn( Light *obj )
   {   obj->state = 1;   }
void turnOff( Light *obj )
   {   obj->state = 0;   }
```

C誕生對象&調用函數

```
struct Light *LightNew(){
   struct Light *t = (Light *)
            malloc(sizeof(Light));
   return (void*) t;
 }

void main() {
  Light *led = (Light*)LightNew();
  led->turnOn=turnOn; /*裝配C函數 */
  led->turnOff = turnOff;
  led->turnOn( led ); /* 啟動執行 */
  led->turnOff( led );    }
```

C定義Light類

```
// C代碼
```

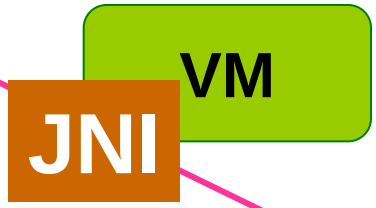
對接

C函數

```
// a.so 檔案(File)
void turnOn( Light *obj )
   {   obj->state = 1;   }
void turnOff( Light *obj )
   {   obj->state = 0;   }
```

C誕生對象& 調用函數

```
// C代碼
```



Java定義Light類

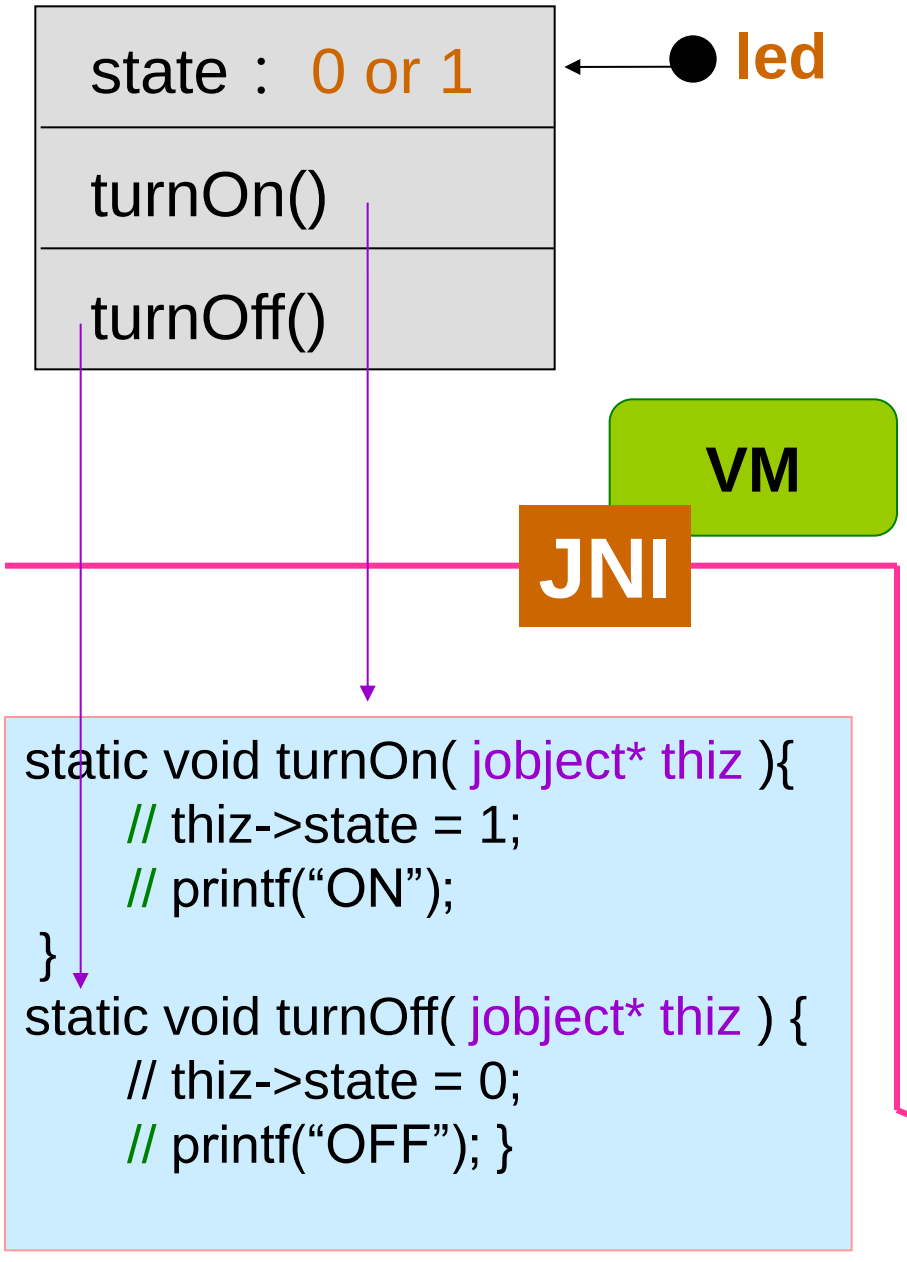
```
// Java代碼
```

C函數

```
static void turnOn( jobject* thiz ){
        // thiz->state = 1;
        // printf(“ON”);
 }
static void turnOff( jobject* thiz ) {
        // thiz->state = 0;
        // printf(“OFF”); }
```

Java誕生對象&調用函數

```
// Java代碼
```

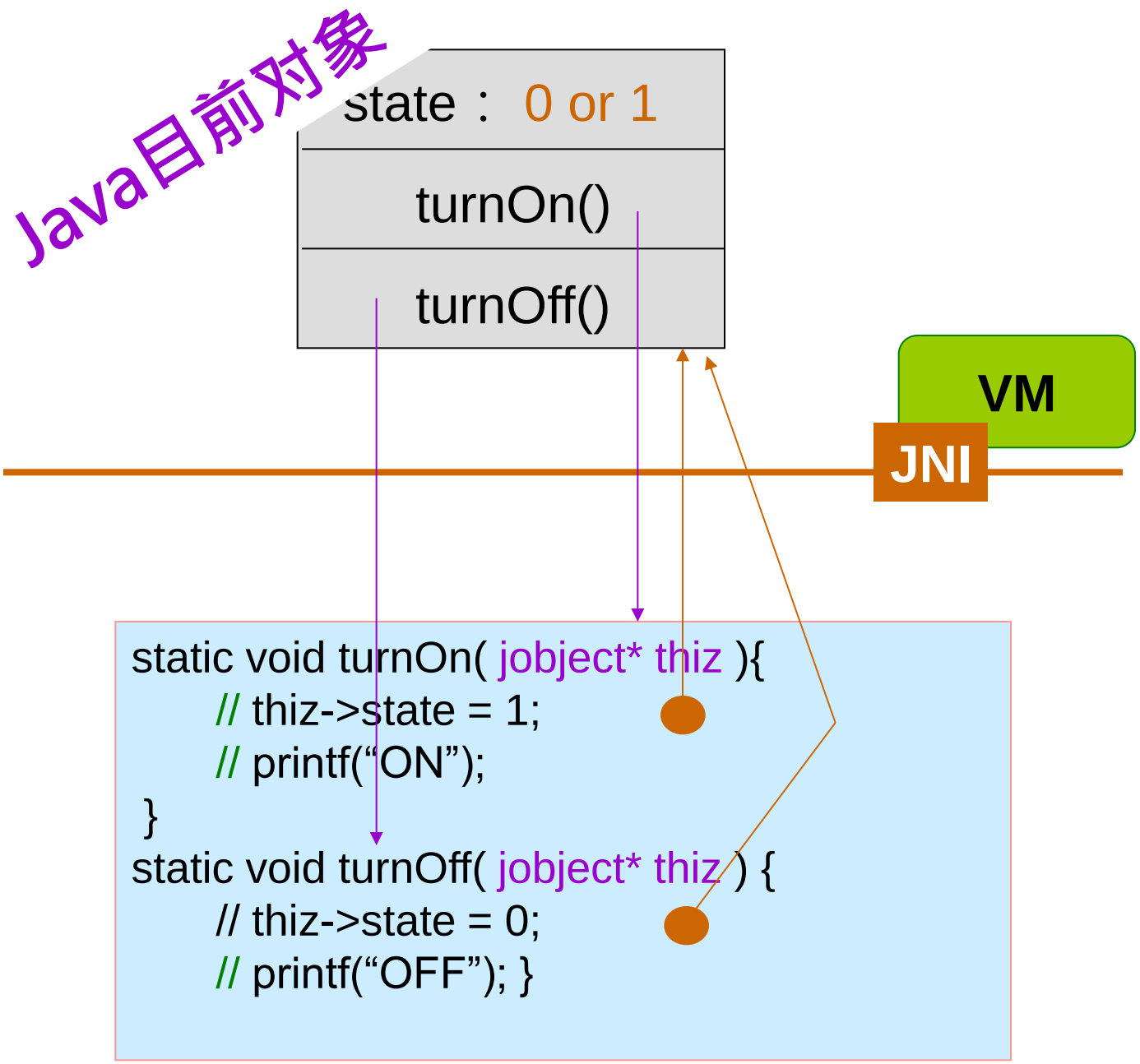


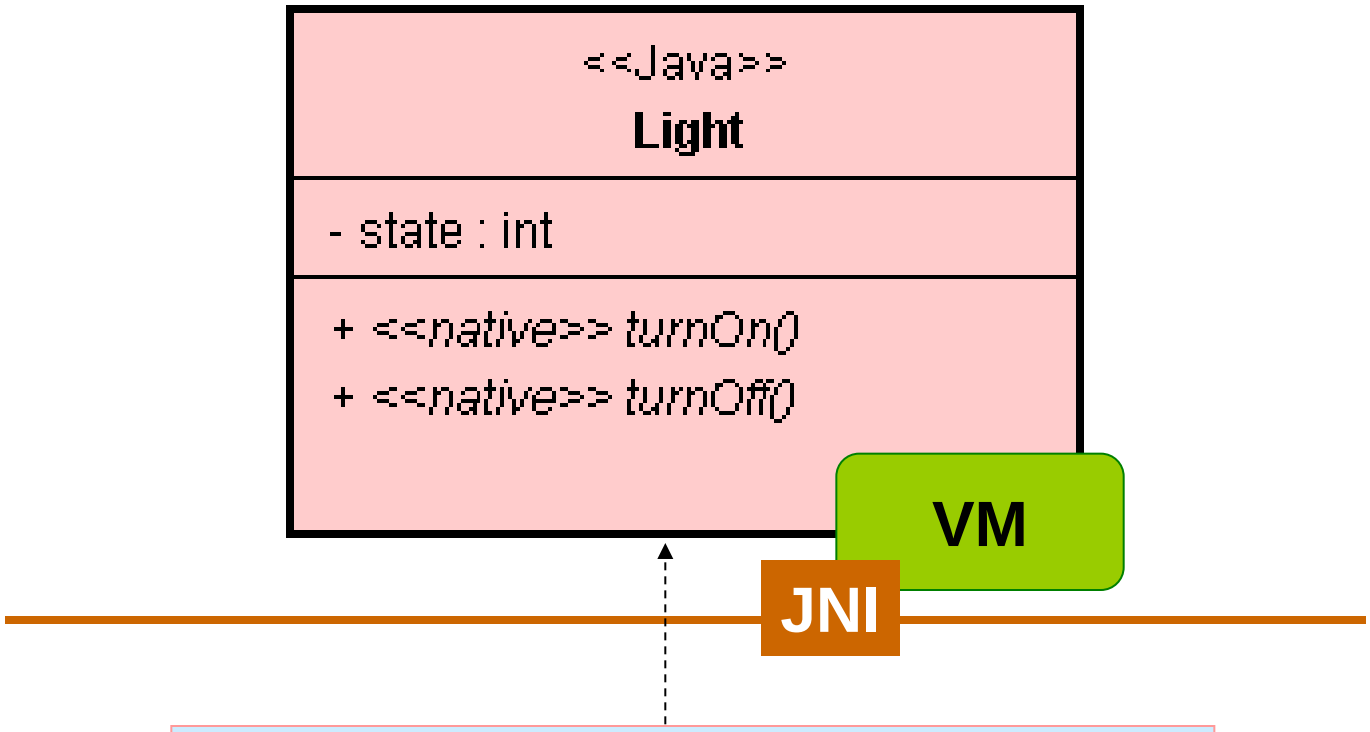
Java定義Light類

```
class Light {
    int state;
    native void turnOn();
    native void turnOff();
};
```

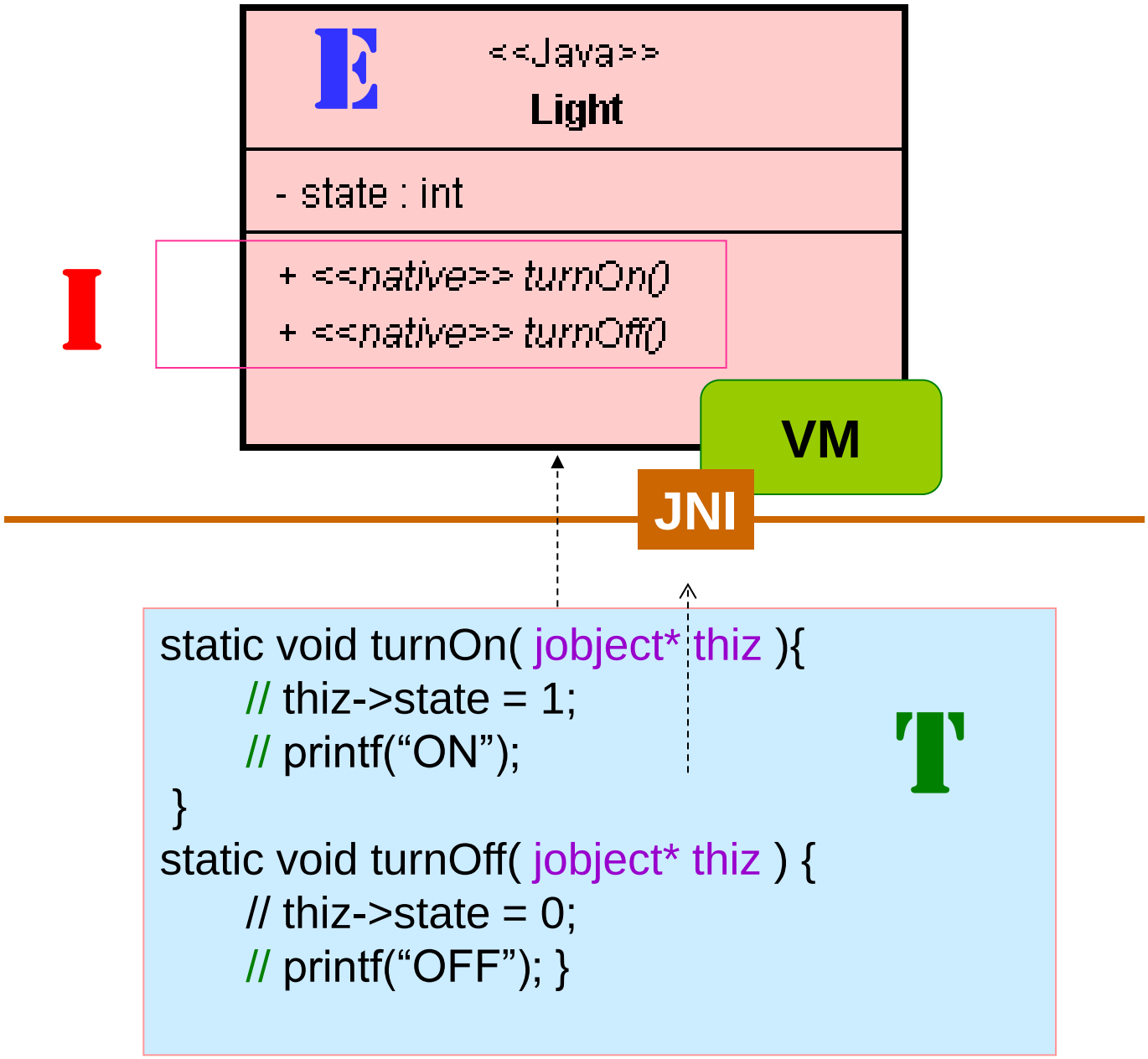
Java誕生對象&調用函數

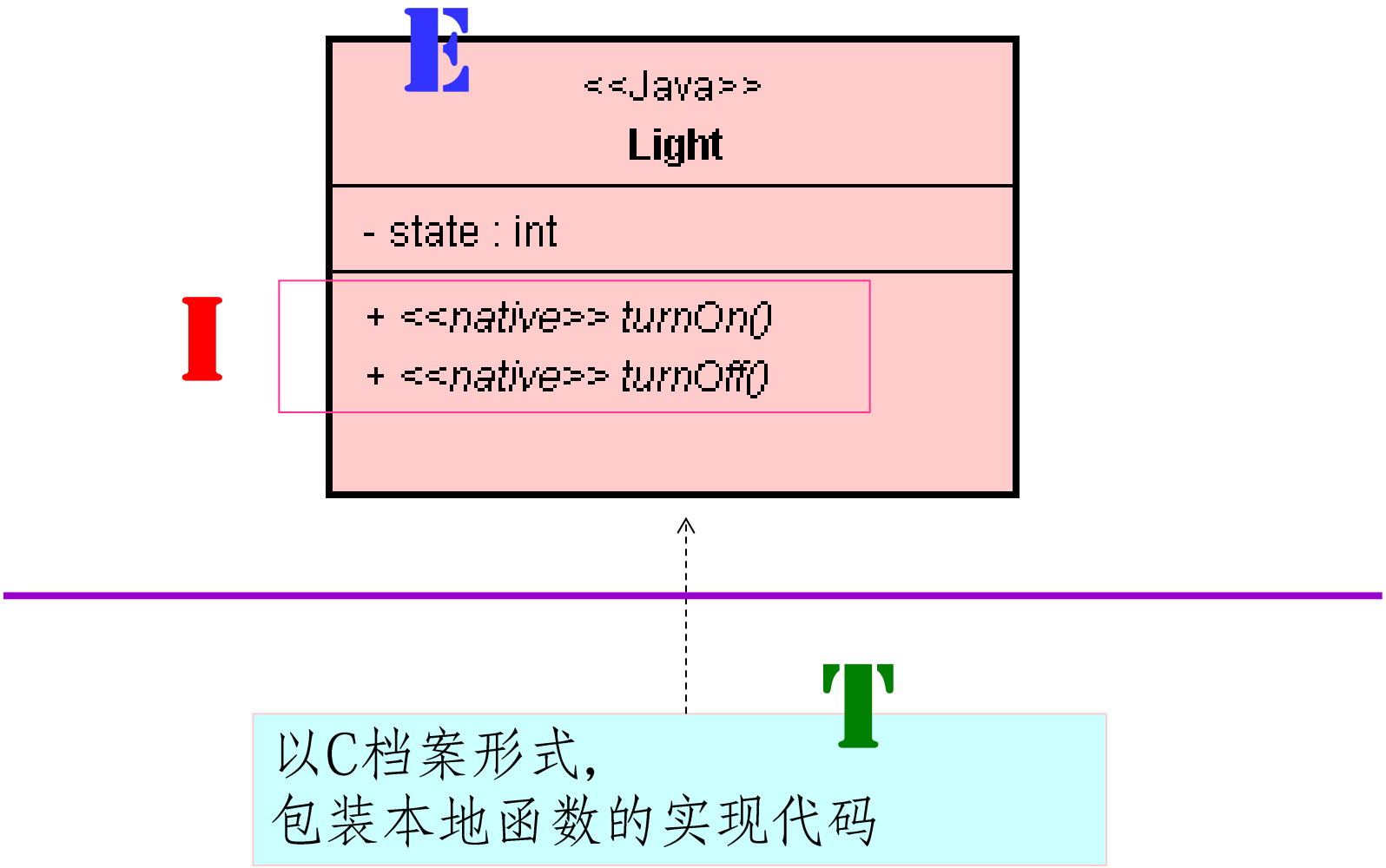
```
Light() {   state = -1;  }

void main() {
        Light led = new Light();
        led.turnOn();
        led.turnOff();
}
```

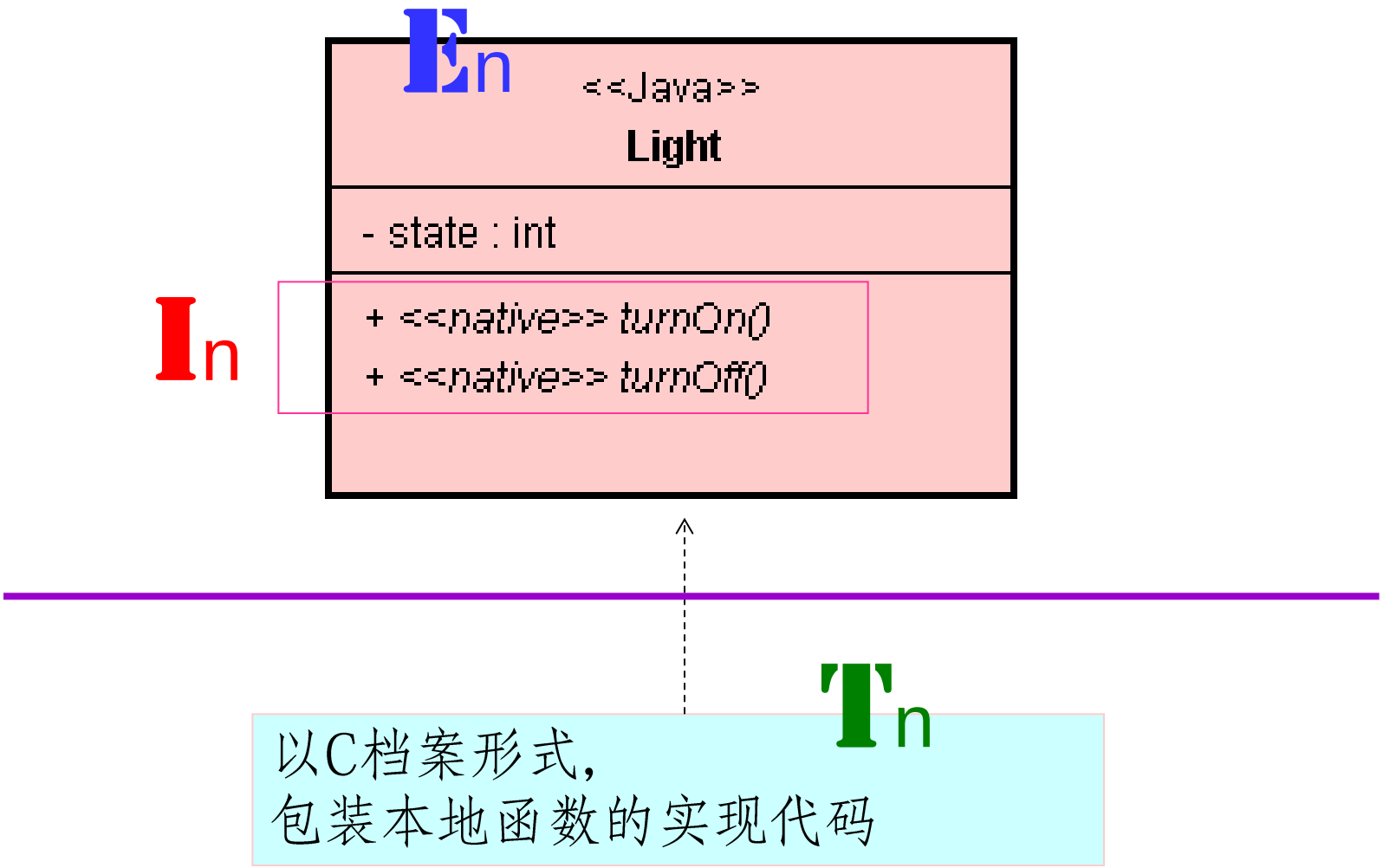

Java目前对象
state： 0 or 1
turnOn()
turnOff()
VM
JNI
static void turnOn(jobject* thiz){
// thiz->state = 1;
// printf("ON");
}
static void turnOff(jobject* thiz) {
// thiz->state = 0;
// printf("OFF"); }




```
static void turnOn( jobject* thiz ){
        // thiz->state = 1;
        // printf("ON");
 }
static void turnOff( jobject* thiz ) {
        // thiz->state = 0;
        // printf("OFF"); }
```


E
<<Java>>
Light
- state : int
+ <<native>> turnOn()
+ <<native>> turnOff()
I
VM
JNI
static void turnOn(jobject* thiz){
// thiz->state = 1;
// printf("ON");
}
static void turnOff(jobject* thiz) {
// thiz->state = 0;
// printf("OFF"); }
T


E
<<Java>>
Light
- state : int
I
+ <<native>> turnOn()
+ <<native>> turnOff()
T
以C档案形式，
包装本地函数的实现代码


En
<<Java>>
Light
- state : int
+ <<native>> turnOn()
+ <<native>> turnOff()
In
Tn
以C档案形式，
包装本地函数的实现代码

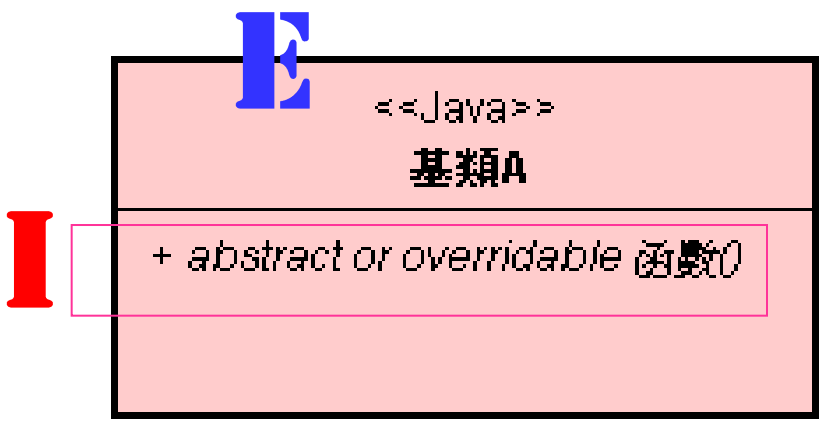
混合式EIT造形

- 一般EIT造形是同语言的。
- 也就是<E>、<I>和<T>都使用同一种语言撰写的，例如上述的Java、C/C++等。
- 于此，将介绍一个EIT造形的变形：

  <E&I>是以Java撰写的。

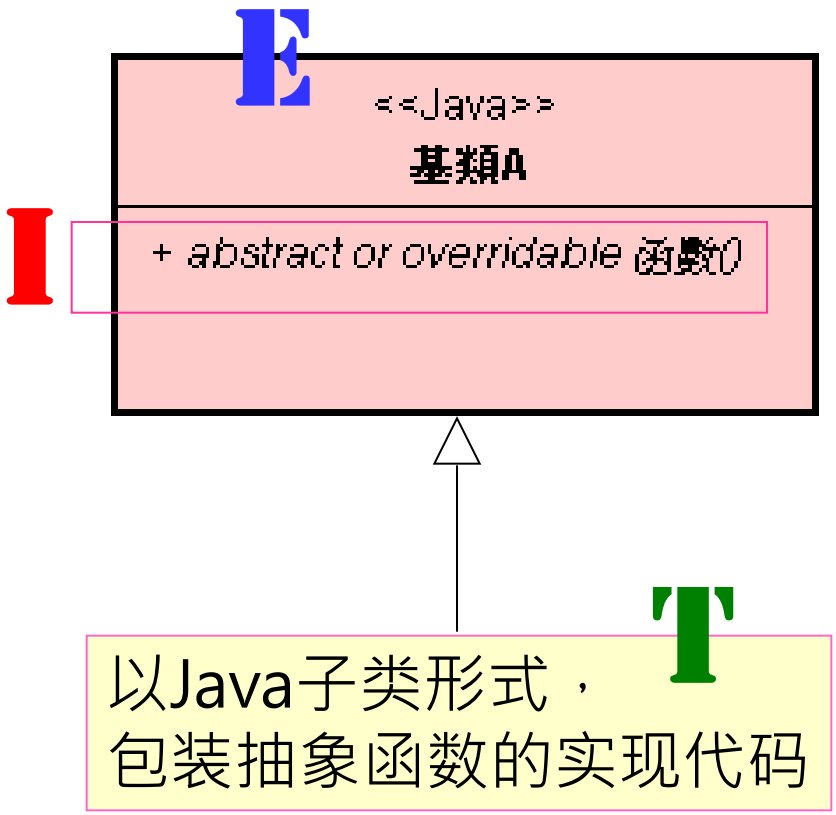
  <T>则是以C语言撰写的。

同語言的<E&I>

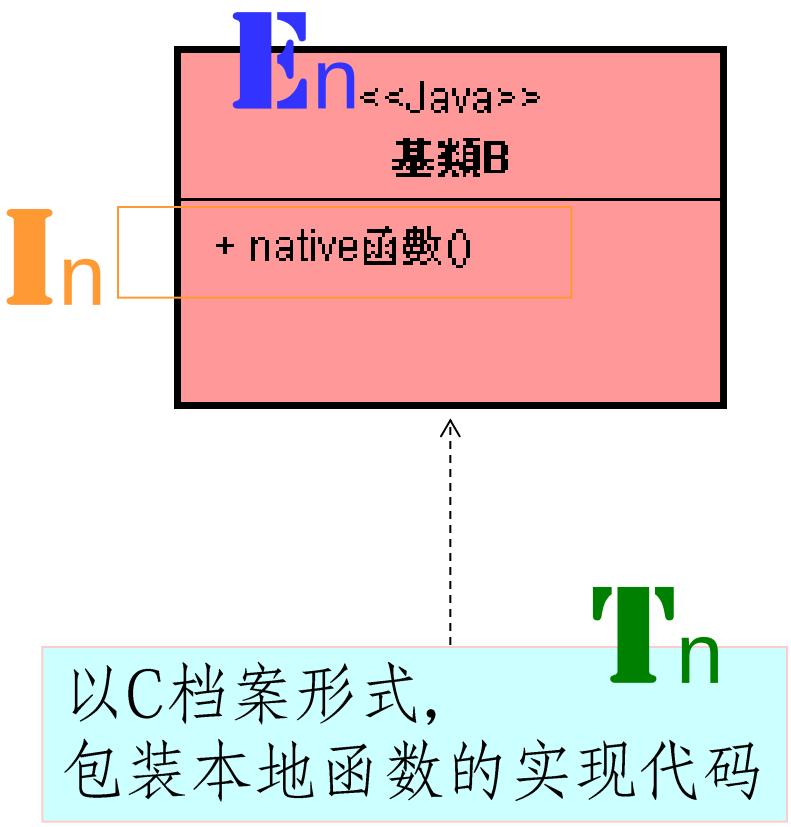


混合語言的<E&I>

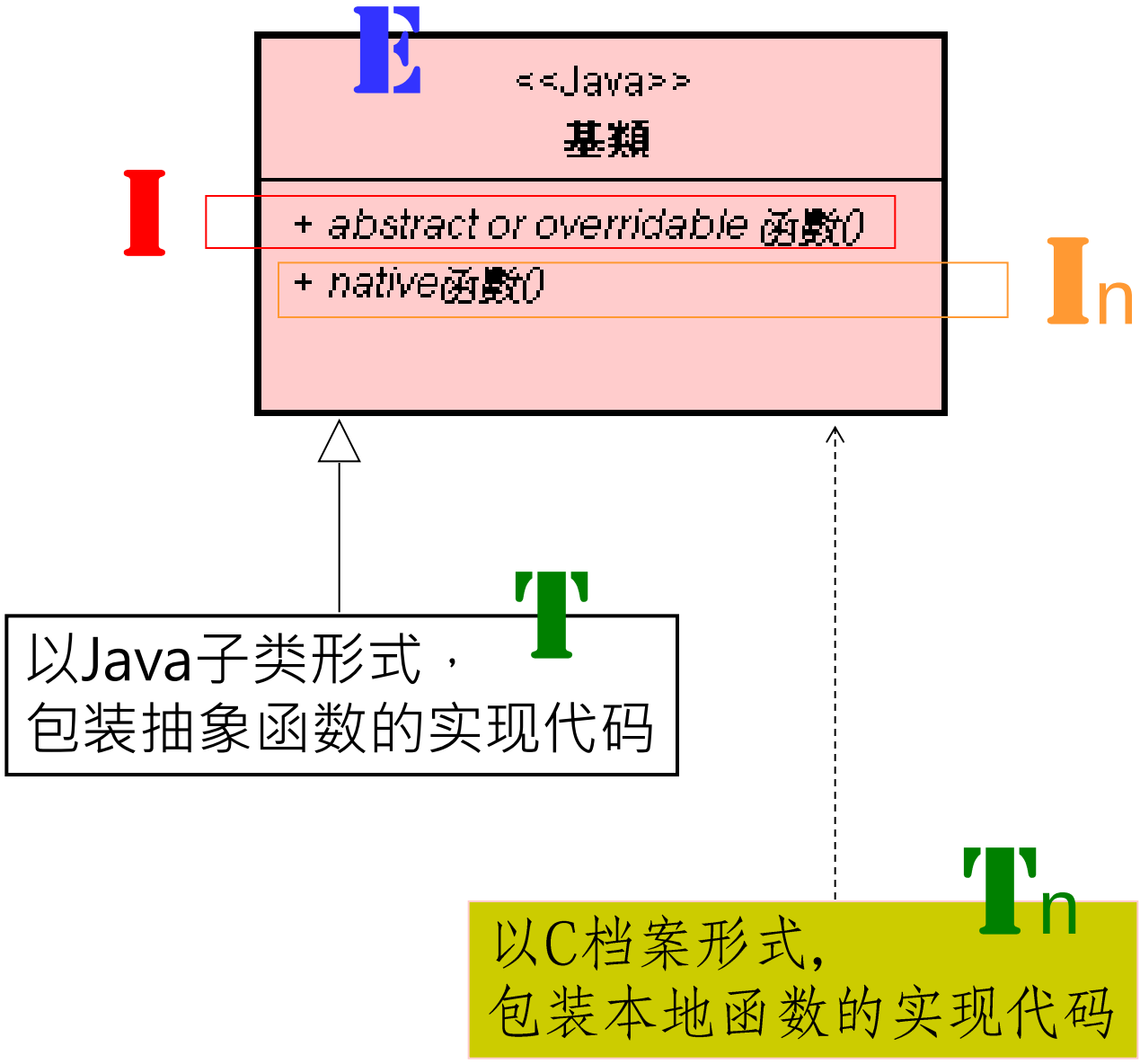
同語言的<E&I>

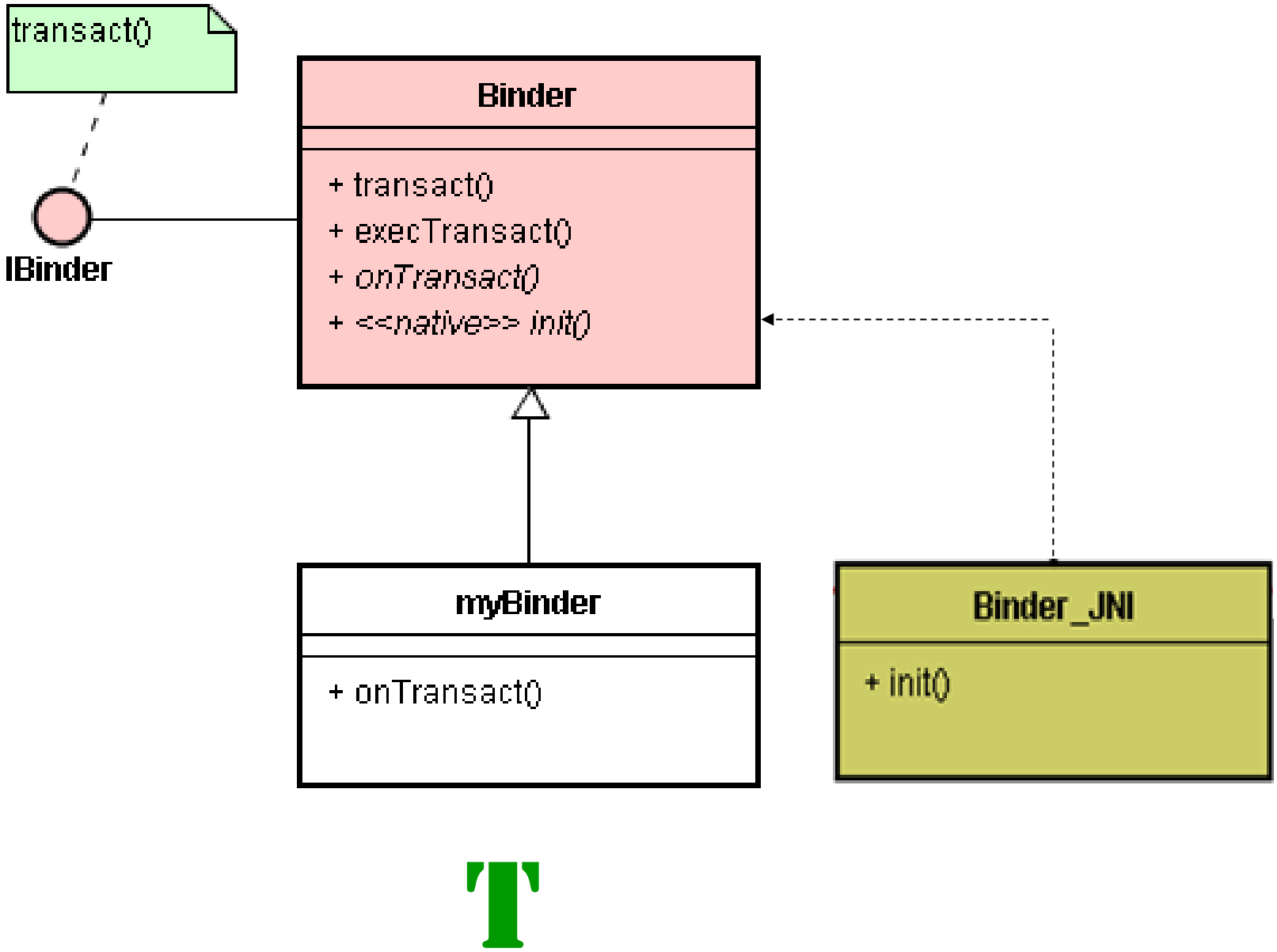


混合語言的<E&I>



两种EIT代码造形
常常合并存在


E
<<Java>>
基類
I
+ abstract or overridable 函數()
+ native函數()
In
T
以Java子类形式，
包装抽象函数的实现代码
Tn
以C档案形式，
包装本地函数的实现代码


transact()
IBinder
Binder
+ transact()
+ execTransact()
+ onTransact()
+ <<native>> init()
myBinder
+ onTransact()
Binder_JNI
+ init()

重要议题：
由谁来创建基类的对象呢?

- 答案是：通常，<Tn>不是App的一部分，而是基类(强龙撰写)的一部分。
- 创建子类<T>和创建基类<E>对象是App开发者(地头蛇)的事；将<T>与<E>装配起来，也是地头蛇的事。
- 因之，<Tn>可能是强龙开发的，或是第三方提供的。

- 结论：在本地C层，<Tn>开发者指需要撰写本地(native)函数的C代码实现即可。
- 创建(基类)对象和函数调用都是Java层的事。


Q&A

1. 为什么，由Java层定义类，创建对象和启动执行呢?
2. C函数是谁来撰写呢? 是强龙，或是地头蛇呢? 还是其它人?
3. 为什么，一个Java基类常常提供抽象函数，又提供本地函数呢?

Thanks...

高煥堂