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

E01_b

OOPC与HAL的美妙结合(b)

By 高煥堂

LW_OOPC初版的宏(Macro)

```
/*  lw_oopc.h */  /* 这就高焕堂老师团队所设计的C */
#include "malloc.h"
#ifndef LOOPC_H
#define LOOPC_H

#define CLASS(type)\
typedef struct type type; \
struct type

#define CTOR(type) \
void* type##New() \
{ \
 struct type *t; \
 t = (struct type *)malloc(sizeof(struct type));
/*  continued  */
```

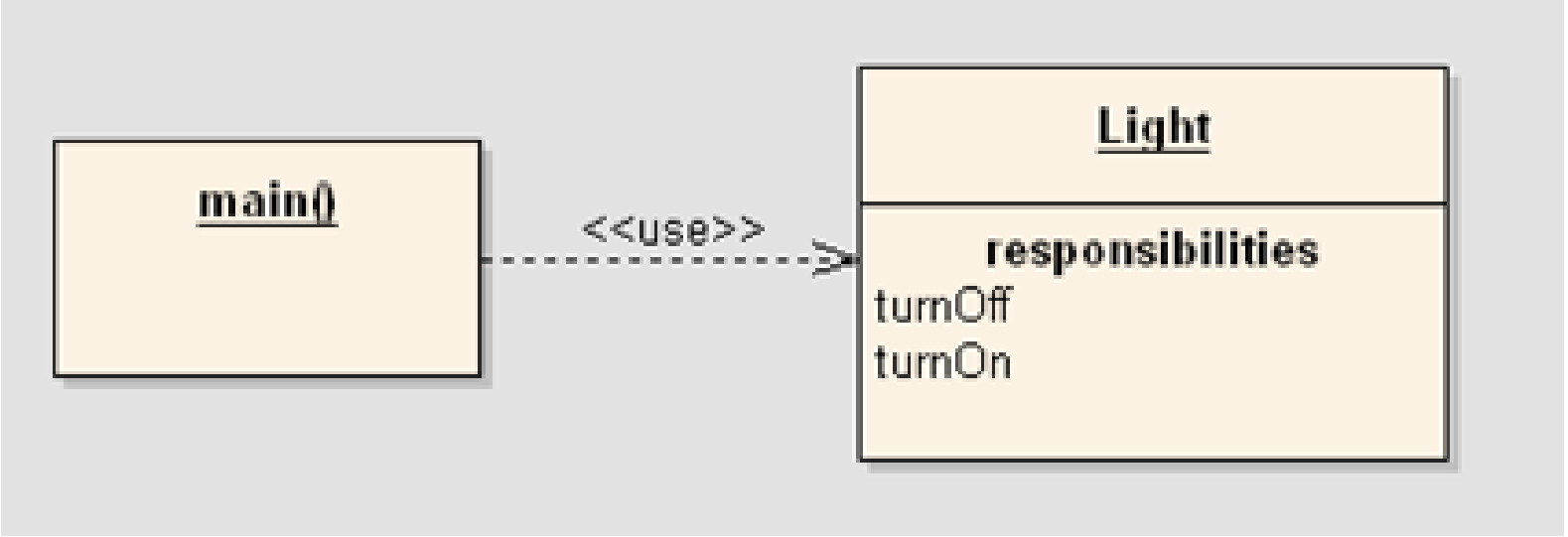
```
#define CTOR2(type, type2) \
void* type2##New() \
{ \
 struct type *t; \
 t = (struct type *)malloc(sizeof(struct type));

#define END_CTOR return (void*)t;  };
#define FUNCTION_SETTING(f1, f2)  t->f1 = f2;
#define IMPLEMENTS(type) struct type type
#define INTERFACE(type) struct type
#endif
/*     end     */
```

LW_OOPC简单示例

- 由于一般C语言并没有使用OO观念。
- 高老师将OO观念添加到C语言，让人们既以OO观念去分析及设计，并以OO观念去编写C程序，则从分析、设计到程序编写的过程就非常直截了当。如下述的步骤：

Step-1: 分析出一个类别叫Light，它提供两项服务，以UML表达如下：



Step-2：实现为LW_OOPC程序：

- 基于上述的lw_oopc.h就可以定义出类别了，例如定义一个Light类别，其light.h内容为：

```
/*  light.h   */
#include "lw_oopc.h“

CLASS(Light) {
    void (*turnOn)();
    void (*turnOff)();
 };
```

- 类别定义好了，就开始编写函数的内容：

```
/*  light.c   */
#include "stdio.h"
#include "light.h“

static void turnOn()
   {  printf("Light is ON\n");  }
static void turnOff()
   {  printf("Light is OFF\n");  }

CTOR(Light)
   FUNCTION_SETTING(turnOn, turnOn)
   FUNCTION_SETTING(turnOff, turnOff)
END_CTOR
```

- 这个 FUNCTION_SETTING(turnOn, turnOn)的用意是：让类别定义(.h文件)的函数名称能够与实现的函数名称不同。例如在light.c里可写为：

```
static void TurnLightOn()
        { ….. }

CTOR(Light)
      {
      FUNCTION_SETTING(turnOn, TurnLightOn);
      …..
      }
```

- 这是创造.c档案自由抽换的空间，这是实践接口的重要基础。
- 最后看看如何编写主程序：

```
#include "lw_oopc.h"
#include "light.h"
extern void* LightNew();

void main()
{
    Light* light = (Light*)LightNew();
    light->turnOn();
    light->turnOff();
    getchar();
    return;
}
```

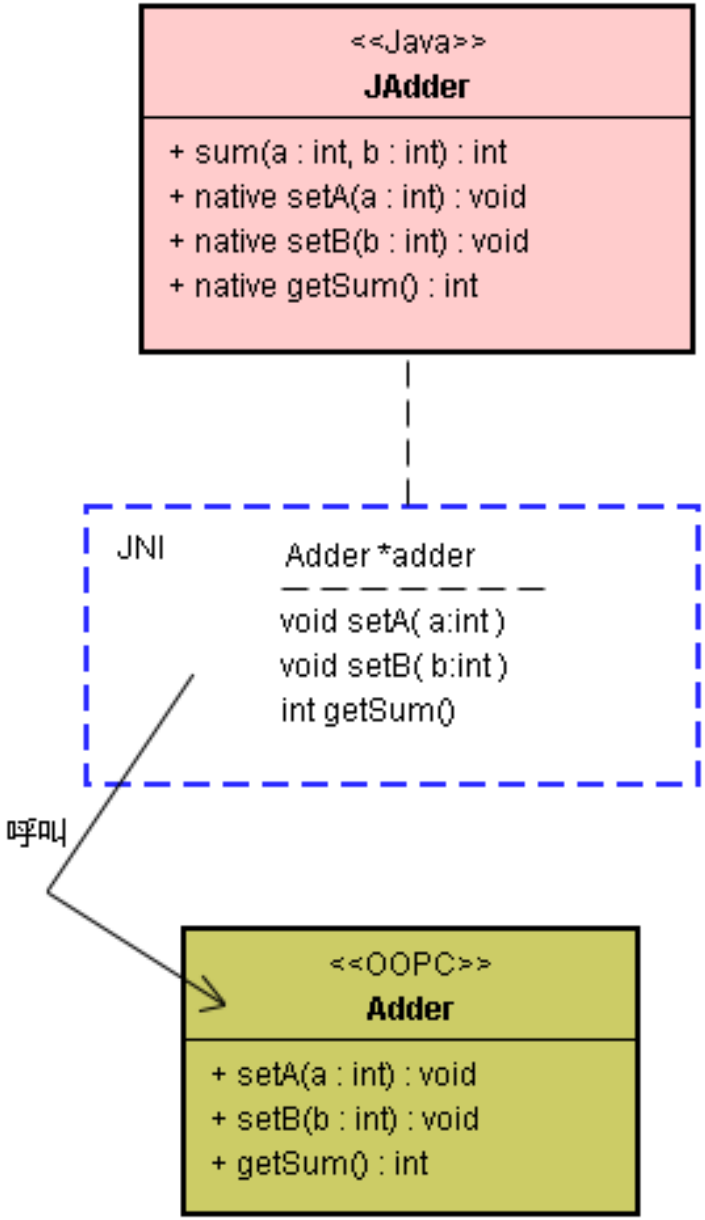
- LightNew()是由CTOR所生成的类别构造器(Constructor)。由于它是定义于别的档案，所以必需加上extern void* LightNew();指令。生成对象的基本格式为：

  类别名称* 对象指针 = (类别名称*)类别名称New();

  示例：Light* light = (Light*)LightNew()
- 然后就能透过对象指针去呼叫成员函数了。

3、演练：<LW_OOPC+JNI>代码范例

- Android 的以JNI为桥梁，将Java与C/C++结合起来。
- 虽然C++是个面向对象的语言，但如果有时不得不用 C来编写代码时，可使用OOPC来写出易读易懂的面向对象代码。
- 举个简单范例，说明如何以OOPC来撰写JNI本地代码。


<<Java>>
JAdder
+ sum(a : int, b : int) : int
+ native setA(a : int) : void
+ native setB(b : int) : void
+ native getSum() : int
JNI
Adder *adder
void setA(a:int)
void setB(b:int)
int getSum()
呼叫
<<OOPC>>
Adder
+ setA(a : int) : void
+ setB(b : int) : void
+ getSum() : int

撰写JAdder接口类别：

```
//  JAdder.java
// ……
public class JAdder {
        static {
                System.loadLibrary("adder_jni");
        }
        int sum(int a, int b) {
                setA(a);
                setB(b);
                return getSum();
        }
        private native void setA(int a);
        private native void setB(int b);
        private native int getSum();
}
```

撰写com_misoo_Adder_JAdder.c代码：

```
/*  com_misoo_Adder_JAdder.c   */
#include "com_misoo_Adder_JAdder.h"
#include <libadder/Adder.h>

static Adder* adder = 0;
static Adder* getAdder() {
	if (adder == 0)
		adder = Adder_new();
	return adder;
}
```

```
JNIEXPORT void JNICALL
Java_com_misoo_Adder_JAdder_setA
  (JNIEnv* env, jobject job, jint a) {
                getAdder()->setA(adder, a);
 }
JNIEXPORT void JNICALL
Java_com_misoo_Adder_JAdder_setB
  (JNIEnv* env, jobject job, jint b) {
                getAdder()->setB(adder, b);
}
JNIEXPORT jint JNICALL
Java_com_misoo_Adder_JAdder_getSum
 (JNIEnv* e, jobject job) {
                return getAdder()->getSum(adder);
}
```

撰写Adder.h和Adder.c代码(OOPC)：

```
/*  Adder.h   */
#ifndef ANDROID_MISOO_OOPC_ADDER_H
#define ANDROID_MISOO_OOPC_ADDER_H
#include <misoo/lw_oopc.h>
CLASS(Adder) {
        int a, b;
        void (*setA)(Adder* thiz, int a);
        void (*setB)(Adder* thiz, int b);
        int (*getSum)(Adder* thiz);
};
#endif // ANDROID_MISOO_OOPC_ADDER_H
```

```
/*    Adder.c    */
#include "Adder.h"
void setA(Adder* thiz, int a){
        thiz->a = a;
}
void setB(Adder* thiz, int b){
        thiz->b = b;
}
int getSum(Adder* thiz){
        return thiz->a + thiz->b;
}
CTOR(Adder)
        FUNCTION_SETTING(setA, setA);
        FUNCTION_SETTING(setB, setB);
        FUNCTION_SETTING(getSum, getSum);
END_CTOR
DTOR(Adder)
END_DTOR
```

- 由JNI的getAdder()来诞生Adder对象。
- 执行的结果是：输出两个整数的总和。

~ Continued ~