# 定义OC的类和创建OC的对象

* 接下来就在OC中模拟现实生活中的情况，创建一辆车出来。首先要有一个车子类，然后再利用车子类创建车子对象
* 要描述OC中的类稍微麻烦一点，分2大步骤：类的声明、类的实现（定义）。跟函数类似，函数有分声明和定义

## 类的声明

1. 代码编写

* 定义一个Car类，拥有2个属性：轮子数、时速，1个行为：跑
* 类名\属性的命名规则：标示符的规则
* 类名的命名规范：有意义、驼峰标识、首字母大写

#import <Foundation/Foundation.h>

// 类的声明

@interface Car : NSObject

{

@public

int wheels; // 多少个轮子

int speed; // 时速

}

- (void)run; // 跑的行为

@end

1. 成员变量

* @interface的大括号{}中声明的变量：wheels、speed
* @interface的大括号和函数的大括号是不一样的
* 默认会初始化为0

1. @public

@public可以让Car对象的wheels和speed属性被外界访问

局部变量、全局变量都有自己的作用域，成员变量也不例外

* @private：只能在当前类的对象方法中直接访问
* @protected：可以在当前类以及子类的对象方法中直接访问
* @public：任何地方都可以直接访问
* @package：同一个“体系内”（框架）可以访问，介于@private和@public之间

1. NSObject

加上:NSObject的目的是让Car类具备创建对象的能力

## 类的实现

// 类的实现

@implementation Car

- (void) run

{

NSLog(@"%i个轮子，%i时速的车子跑起来了", wheels, speed);

}

@end

## 创建对象

1. 代码编写

// 主函数

int main()

{

// 创建车子对象

Car \*c = [Car new];

c->wheels = 3;

c->speed = 300;

[c run];

return 0;

}

1. main函数的代码分析、内存分析（对象在内存中有成员）

* [Car new]每次都会创建出新的对象，并且返回对象的地址，那么就应该用一个指针变量保存对象的地址

Car \*c = [Car new];

用一个指针变量c指向内存中的Car对象

* 设置车子对象的属性

跟用指向结构体的指针访问结构体属性一样，用->

c->wheels = 3;

c->speed = 300;

## 创建多个Car对象

* 分别只设置wheels、speed属性

Car \*c1 = [Car new];

c1->wheels = 4;

Car \*c2 = [Car new];

c2->speed = 250;

[c1 run];

* 1个赋值给另一个，然后修改属性

Car \*c1 = [Car new];

c1->wheels = 4;

c1->speed = 250;

Car \*c2 = c1;

c2->wheels = 3;

[c1 run];

## 面向对象封装的好处

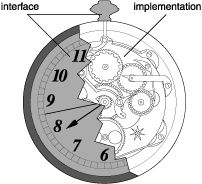
* 更加接近人类的思考方式
* 只需要关注对象，不需要关注步骤

## 对象与函数参数

* 对象成员变量作为函数参数
* 指向对象的指针作为函数参数
* 修改指向指向对象的成员
* 修改指针的指向

# 类的声明和实现

## @interface和@implementation的分工



* @interface就好像暴露在外面的时钟表面
* @implementation就好像隐藏在时钟内部的构造实现

## 声明和定义多个类

## 常见错误

* 只有类的声明，没有类的实现
* 漏了@end
* @interface和@implementation嵌套
* 两个类的声明嵌套
* 成员变量没有写在括号里面
* 方法的声明写在了大括号里面

## 语法细节

* 成员变量不能在{}中进行初始化、不能被直接拿出去访问
* 方法不能当做函数一样调用
* 成员变量\方法不能用static等关键字修饰，别跟C语言混在一起（暂时忽略）
* 类的实现可用写在main函数的后面，主要在声明后面就行了

## OC方法和函数的区别

* OC方法只能声明在@interface和@end之间，只能实现在@implementation和@end之间。也就是说OC方法不能独立于类存在
* C函数不属于类，跟类没有联系，C函数只归定义函数的文件所有
* C函数不能访问OC对象的成员
* 低级错误：方法有声明，但是实现的时候写成了函数

## OC的方法注意

* 方法只有声明，没有实现（经典错误）
* 方法没有声明，只有实现（编译器警告，但是能调用，OC的弱语法）
* 编译的时候：访问没有的成员变量直接报错，访问没有的方法，只是警告

## @implementation

* 没有@interface，只有@implementation，也是能成功定义一个类的

@implementation Car : NSObject

{

@public

int wheels; // 多少个轮子

int speed; // 时速

}

- (void) run

{

NSLog(@"%i个轮子，%i时速的车子跑起来了", wheels, speed);

}

@end

* @implementation中不能声明和@interface一样的成员变量

# 方法

设计一个Caculator计算器类，它拥有计算的功能（行为）

## 不带参数的方法

* 设计一个返回PI的方法

// 方法声明

- (double)pi;

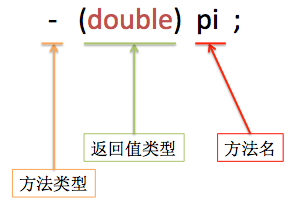
// 方法实现

- (double)pi

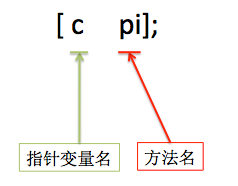
{

return 3.14;

}



* 方法调用



## 带一个参数的方法

* 设计一个计算平方的方法

// 方法声明

- (double)square:(double)number;

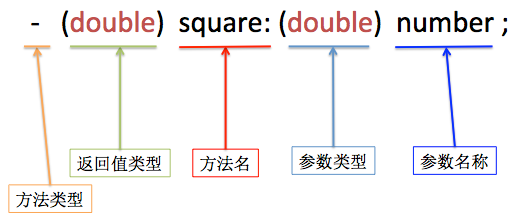
// 方法实现

- (double)square:(double)number

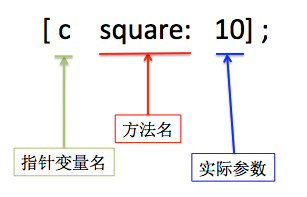
{

return number \* number;

}



* 方法调用



## 带多个参数的方法

* 设计一个计算和的方法

// 方法声明

- (double)sumOfNum1:(double)num1 andNum2:(double)num2;

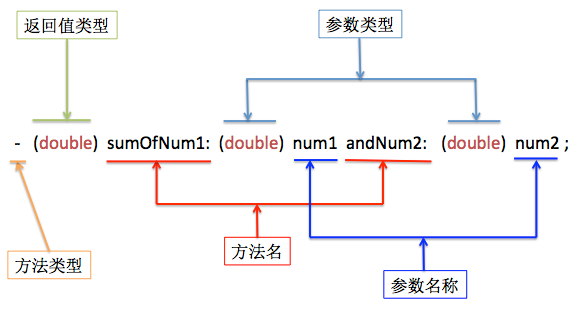
// 方法实现

- (double)sumOfNum1:(double)num1 andNum2:(double)num2

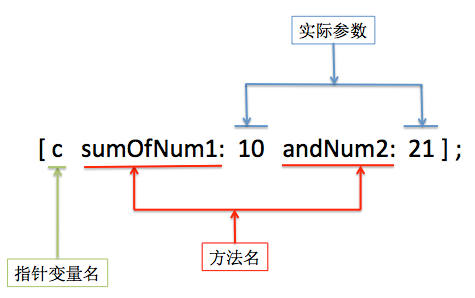
{

return num1 + num2;

}



* 方法调用



## 方法名注意

* 冒号也是方法名的一部分
* 同一个类中不允许两个对象方法同名

## 习题

给Car类设计一个方法，用来和其他车比较车速，如果本车速度快，就返回1，如果本车速度慢，就返回-1，速度相同就返回0

# 匿名对象

* 属性访问

[Car new]->speed = 200;

* 方法调用

[ [Car new] run];