1. @ sign immediately precedes a constant **NSString** object, if no @ is specified, you are writing a constanct C-style string.
2. How to call methods defined in objective c.

myCar = [Car new];

[myCar getName];

[myCar setName:@”hello world”];

1. How to define one class

The @interface section describes the class and its methods, and the @implementation section describes the data (the instance variables that objects from the class will store) and contains the actual code that implements the methods declared in the interface section.

**@interface Fraction: NSObject //Parent class NSObject**

**-(void) print;**

**-(void) setNumerator: (int) n;**

**-(void) setDenominator: (int) m;**

**@end**

**// here – means it’s an instance method, + means it’s a class method**

**@implementation Fraction**

**{**

**int numerator;**

**int denominator;**

**}**

**-(void) print**

**{**

**NSLog(@”%i / %i ”, numerator, denominator);**

**}**

**-(void) setNumerator: (int) n**

**{**

**numerator = n;**

**}**

**-(void) setDenominator: (int) m**

**{**

**denominator = m;**

**}**

**@end**

When a method takes an argument, you also append a colon to the method name when referring to the method. Therefore, setNumerator: and setDenominator: is the correct way to identify these two methods, each of which takes a single argument.

1. Some datatype and qualifiers

Type: int float double char

Qualifiers: long, long long, short , signed, unsigned

With these, you can get long int, long long int, long double etc which will have greater range.

With these qualifiers, the output format will change also:

char %c

short char %sc

int %i, %x, %o

long int %li

long long int %lli

id %p

1. Type id is used to store an object of any type, so it’s a generic object type, which is also the **basis** for very important features in Objective-C known as **polymorphism** and **dynamic binding**.
2. Synthesized Accessor methods

As of Objective-C 2.0 you can have your setter/getter methods automatically generated for you.

* Use @property directive in your interface section to identify your properties.

@property int numerator, denominator;

* Use @synthesize numerator, denominator in implementation part.

@synthesize numerator, denominator;

You don’t need to use the @synthesize directive, using the @property is enough. If you leave out the @synthesize, then the compiler will generate the variable with an underscore(\_) as the first character of its name.

1. Accessing properties using the dot operator

[myFraction numerator]

[myFraction setNumerator: 1]

🡺

int a = myFraction.numerator;

myFraction.numerator = 1;

1. Multiple arguments to methods

You can define methods that take multiple arguments simply by listing each successive argument followed by a colon. This becomes part of the method name. For example, the method named **addEntryWithName:andEmail:** takes two arguments.

-(void) setTo: (int)n over: (int) d;

[myFraction setTo:1 over: 3];

1. Methods without argument names

When creating the name for a method, the argument names are actually optional, you can declare a method like this:

-(int) set: (int) n :( int) d;

[myFraction set:1 :3];

1. @class XYPoint

@class directive is used to declare some type before hand so that we no need to import the header file here.

It’s more efficient because the compiler doesn’t need to import and therefore process the entire XYPoint.h file.

1. Polymorphism: Same name, different class

The capability to share the same name across different classes is known as polymorphism, which enables you to develop a set of classes that each can response to the same method name.

1. Dynamic binding and the id type

Id datatype type can hold any type of object, when you invoke methods on id datatype, the system knows which datatype is called actually. It because the objective-c system always keeps track of the class to which an object belongs.

The polymorphism and the dynamic binding , id type make the core part of Objective-C programing.

1. Asking questions about classes

In some cases, we need to know the information of the class.

Like is it a Rectangle? Does this object support a print method etc.

Here are some methods, class-object is a class object (typically generated by the class method), and selector is a value of type SEL(typically created with @selector directive)

isKindOfClass: is the object a member of class-object or a descendant

isMemberOfClass: is the object a member of class-object

respondsToSelector: selector Can object respond to method

instancesRepondToSelector: can instance of class respond to selector

isSubclassOfClass: is it a subclass?

performSelector apply the method

[myFraction class] // get the class object

@selector(alloc) //get selector of alloc method

SEL action;

id graphicObject;

Action = @selector(draw);

[graphicObject performSelector: action];

1. Exception handling

@try {

}

@catch(NSException\* exception){

@throw ; // throw your own exception

}

@finally {

}

1. Define methods to initialize your objects, something like constructor in C++

-(instancetype) initWith: (int) a over: (int) b;

-(instancetype) initWith: (int)a over: (int) b

{

self =[super init];

if (self ) {

[self setTo:a over:b];

}

return self;

}

1. Enum datatype

enum direction { up, down, left, right };

1. Typedef can help you to define another datatype

typedef int Counter;

Counter a = 3;

1. Categories

Sometimes you want to add method to a class implementation, category is good solution for it, you can even add new method to some framework classes.

A category provides an easy way for you to modularize the definition of a class into groups or categories of related methods. It also gives you an easy way to extend an existing class definition without even having access to the original source code for the class and without having to create a subclass.

@interface Fraction (MathOps)

@end

Here is the definition for category MathOps to Fraction class.

The category files must be saved like this:

Fraction+MathOps.h

Fraction+MathOps.m

1. Extension is a special case of Category, which doesn’t have a name. When you define an unnamed category, you can extend the class by adding **additional instance variables and properties.**  This is not allowed for named categories. Methods declared in a class extension are implemented in the main implementation section for the class and not in a separated implementation section.

**Class extension is useful, because their methods are private.**

extension 提供了一种数据封装的手段， extension里定义的东西在主要类实现提供， 但是类的接口里不提供这些东西， 因此类的使用者无法看见这些数据！！！！

extension 定义只是定义接口， 实现还必须是在类实现里。

A category can override another method in the class, but this is bad practice. It because after you override a method, you can no longer access the original method.

1. Protocols and Delegation

A protocol is a list of methods that is shared among classes. The methods listed in the protocol don’t have corresponding implementations, something like interface in C++.

methods under @optional directive are optional, you can ignore them at all in your class.

@protocol NSCopying //by default it’s @required

-(void) paint;

-(void) erase;

@optional

-(void) outline

@end

@interface AddressBook: NSObject <NSCopying, NSCoding>

@end

We can also test if an object conforms to a protocol.

If( [currentObject conformsToProtocol: @protocol(Drawing)] == YES )

{

}

id<Drawing> aa;

id<NSCopying, NSCoding> bb;

protocol can also extend another protocol.

@protocol Drawing3D <Drawing>

@end

A category can also adopt a protocol like

@interface Fraction(Stuff) <NSCopying, NSCoding>

@end

An extension

@interface Fraction () <NSCopying>

@end

1. You can think of a protocol as an interface definition between two classes. The class defines the protocol can be thought of as **delegating** the work defined by the methods in the protocol to the class that implements them.
2. Block look and act a lot like functions, blocks can be passed as arguments to functions.

void (^printMessage)(int) =

^(int a )

{

NSLog(@"hello world %i", a);

};

printMessage();

1. 函数指针

void exchange(int \*ptr1, int \*ptr2)

{

}

typedef void (\*stanley)(int \*p1, int \* p2);

stanley aa = exchange;

aa(3, 4);

1. D
2. D
3. D
4. d