Add methods to classes

NSString-Extra.h

@interface NSString (NSString_Extras)
-
@end

NSString* rotate1;
@end

@end
#import "NSString-Extras.h"

@implementation NSString (NSString_Extras)

- (NSString *) rotate1 {
    int size = [self length];
    unichar rotatedChars[size];
    for (int k = 0;k < size;k++) {
        unichar currentChar = [self characterAtIndex: k];
        unichar rotateChar = currentChar + 1;
        if (currentChar == 'z') rotateChar = 'a';
        else if (currentChar == 'Z') rotateChar = 'A';
        rotatedChars[k] = rotateChar;
    }
    NSString * rotated = [[NSString alloc] initWithCharacters:rotatedChars length:size];
    [rotated autorelease];
    return rotated;
}
@end
Using the new NSString method

#import "NSString-Extras.h"

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    NSString * result = ["abc" rotate1];
    NSLog(@"Result = %@", result);
    [pool drain];
    return 0;
}

Output

@"bcd"
Aside - Characters

Strings hold unichar

typedef unsigned short unichar;

Literal 'a'
Reasons for Categories

Add methods to existing classes

Allow you to define a class in multiple files
   Each file containing related methods
Extensions - Motivation

One often has method that is private
   Needed by other methods to do some work
   But don't make sense as public method

All Objective-C methods are public

How to hide your private methods

   Use extension in implementation file
   Compiler requires extension methods
   People will not see method listed in header file
   Extension methods are still public
Extensions - Empty Categories

ClassExtensionExample.h

@interface ClassExtensionExample : NSObject {
    
- (void)normalMethod;
@end

ClassExtensionExample.m

#import "ClassExtensionExample.h"
@interface ClassExtensionExample ()
-(void) implementationExtension;
@end

@implementation ClassExtensionExample
- (void)normalMethod {
    NSLog(@"normal");
}
- (void)implementationExtension {
    NSLog(@"implementationExtension");
}
@end

Used to implement methods intended to be privately used in class. Can define the extension in either header file or implementation file. Common case is shown above
Motivation for Extensions

Old compiler

Compiler was one-pass

Private methods had to be Defined before used

Generated warning

@interface Foo : NSObject
-(void) bar;
@end

@implementation Foo
-(void) bar {
[self privateBar];
}
-(void) privateBar {
    NSLog(@"Private bar");
}
@end

Old compiler

Error

Warning

@interface Foo : NSObject
-(void) bar;
@end

@implementation Foo
-(void) privateBar {
    NSLog(@"Private bar");
}
@end
Motivation for Extensions

Old compiler
No Errors
Warning

Compiler was one-pass

Private methods had to be Defined before used

Generated warning

@interface Foo : NSObject
-(void) bar;
@end

@interface Foo()
-(void) privateBar;
@end

@implementation Foo
-(void) bar {
    [self privateBar];
}
-(void) privateBar {
    NSLog(@"Private bar");
}
@end
Motivation for Extensions

Current compiler

Multi-pass

Private methods could be Defined anywhere

No warning messsages

@interface Foo : NSObject
-(void) bar;
@end

@implementation Foo
-(void) bar {
    [self privateBar];
}
-(void) privateBar {
    NSLog(@"Private bar");
}
@end
Motivation for Extensions

Use of extensions a matter of style
Protocols

Like Java's interface

Phones.h

@protocol Phones
- (NSString*) mobile;
- (NSString*) work;
@end

#import "Phones.h"

@interface Person : NSObject <Phones>{
    NSString* fullName;
    int age;
}

@property int age;

- (id) initWithName: (NSString*) name;

@end
Using Protocol

#import "Person.h"

@implementation Person

@synthesize age;

- (NSString *) mobile {
    return @"619-999-9999";
}

- (NSString *) work {
    return @"619-999-9999";
}

etc.

Person* you = [[Person alloc] initWithName:@"Sam"];
NSString* cellNumber = [you mobile];
id <Phones> test = you;
Protocol Details

Can declare methods & properties

optional & required (default) methods

@protocol MyProtocol
- (void)requiredMethod;

@optional
- (void)anOptionalMethod;
- (void)anotherOptionalMethod;

@required
- (void)anotherRequiredMethod
@end
Blocks

Apple addition to C

Used for
- Assigning tasks to separate cores
- Completion handlers
- Notification handlers
- Error handlers
- Enumeration
- View animation
- Sorting

A closure

Function and
Referencing environment
Blocks - Simple Example

```cpp
int (^increase)(int) = ^(int amount) { return amount + 1;};

int result = increase(10);
```
Blocks - Accessing Environment Example

typedef int (^adder)(int);

    int increment = 3;
    adder increase = ^(int amount) { return amount + increment;};

int result = increase(10);
NSLog(@"result = %i", result);    //13

    increment = 20;
    result = increase(10);
NSLog(@"result = %i", result);    //13
Blocks - Read only variables

typedef void (^add)(int);
    int sum = 0;

    add increase = ^(int amount) { sum = sum + amount;}; // Compile error

    increase(2);
    increase(5);
    NSLog(@"result = %i", sum);
Blocks - Block variables

typedef void (^add)(int);
    __block int sum = 0;

    add increase = ^(int amount) { sum = sum + amount;};

    increase(2);
    increase(5);
    NSLog(@"result = %i", sum);
Blocks - In functions

```c
void foo(void (^adder)(int)) {
    int sum = 10;
    adder(9);
    NSLog(@"foo sum %i", sum);
}

int main (int argc, const char * argv[]) {
    typedef void (^add)(int);
    __block int sum = 0;
    add increase = ^(int amount) { sum = sum + amount;};
    foo(increase);
    NSLog(@"result = %i", sum);
    return 0;
}
```

Which sum is changed?