ҚАЗАҚСТАН РЕСПУБЛИКАСЫ
БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РЕСПУБЛИКИ КАЗАХСТАН
MINISTRY OF EDUCATION AND SCIENCE
OF THE REPUBLIC OF KAZAKHSTAN

Л.Н. ГУМИЛЕВ АТЫНДАҒЫ
ЕУРАЗИЯ ҰЛТТЫҚ УНИВЕРСИТЕТІ
ЕВРАЗИЙСКИЙ НАЦИОНАЛЬНЫЙ
УНИВЕРСИТЕТ ИМ. Л.Н. ГУМИЛЕВА
GUMILYOV EURASIAN
NATIONAL UNIVERSITY

ЖАС ҒАЛЫМДАР КЕҢЕСІ

Студенттер мен жас ғалымдардың «Ғылым және білім - 2015» атты X Халықаралық ғылыми конференциясының БАЯНДАМАЛАР ЖИНАҒЫ

СБОРНИК МАТЕРИАЛОВ X Международной научной конференции студентов и молодых ученых «Наука и образование - 2015»

PROCEEDINGS of the X International Scientific Conference for students and young scholars «Science and education - 2015»

УДК 001:37.0
ББК72+74.04
Ғ 96

Ғ96
«Ғылым және білім – 2015» атты студенттер мен жас ғалымдардың X Халық. ғыл. конф. = X Межд. науч. конф. студентов и молодых ученых «Наука и образование - 2015» = The X International Scientific Conference for students and young scholars «Science and education - 2015». – Астана: http://www.enu.kz/ru/nauka/nauka-i-obrazovanie-2015/, 2015. – 7419 стр. қазақша, орысша, ағылшынша.

ISBN 978-9965-31-695-1

Жинаққа студенттердің, магистранттардың, докоранттардың және жас ғалымдардың жаратылыстану-техникалық және гуманитарлық ғылымдардың өзекті мәселелері бойынша баяндамалары енгізілген.

The proceedings are the papers of students, undergraduates, doctoral students and young researchers on topical issues of natural and technical sciences and humanities.

В сборник вошли доклады студентов, магистрантов, докторантов и молодых ученых по актуальным вопросам естественно-технических и гуманитарных наук.

УДК 001:37.0
ББК 72+74.04

ISBN 978-9965-31-695-1

… д-ра пед. наук. – СПб.,1994 [36].

2) Бороненко Т.А., Рыжова Н.И. Методика обучения информатике. Специальная методика: Учеб. пособие для студентов. – СПб.: РГПУ им. А.И. Герцена, 1997. [134]

3) Педагогика: Учебное пособие для студентов педагогических учебных заведений. 4-е изд. / В.А. Сластенин, И.Ф. Исаев, А.И. Мищенко, Е.Н. Шлянов. – М.: Школьная Пресса, 2002. [512]

4) Лебедева И.А. Методика отбора содержания обучения будущих учителей информатики конструированию компиляторов: Автореф. дис. … канд. пед. наук. – СПб., 1996. [19]

5) Пидкасистый П.И. Самостоятельная деятельность учащихся. – М.: Педагогика, 1972. [184]

УДК004.4

ВВЕДЕНИЕ В OBJECTIVE-C

Плалов Нурхат Турсынбекович
ЕНУ им. Л.Н.Гумилева, Астана, Казахстан
Научный руководитель – А.Альжанов

Objective-c - объектно-ориентированное расширение языка С в стиле языка Small-Talk. В настоящее время в основном используется для разработки приложения для MacOs и iOS (для айфона).

Есть три вариации основной библиотеки для Objective-c:

- Cocoa Api на MacOS;
- GNUStep;
- OpenStep;

Поэтому синтаксис языка будет рассмотрен в связке с этой библиотекой [1].

Первое, что мы встречаем это директиву **#import** - действие ее аналогично директиве **#include** языка С, с тем лишь отличием, что допускает множественные включения в код без использования конструкций директив**#ifdef**, без боязни рекурсивного включения одного и того-же файла. Но никто не запрещает Вам использовать**#include**, хотя врядли это будет удобным.

Для обозначения объектов используется специальный тип**id**. Переменная типа**id**фактически является указателем на произвольный объект. Для обозначения нулевого указателя на объект используется константа***nil***.

Вместо id можно использовать и более привычное обозначение с явным указанием класса. В частности последнее позволяет компилятору осуществлять некоторую проверку поддержки сообщения объектами - если компилятор из типа переменной не может сделать вывод о поддержке объектом данного сообщения, то он выдаст предупреждение. Тем самым язык поддерживает проверку типов, но в нестрогой форме (то есть найденные несоответствия возвращаются как предупреждения, а не ошибки).

Для посылки сообщений используется следующий синтаксис:

[receiver message];

receiver - указатель на объект

message - имя метода

(Немного похоже на С++ конструкцию указатель->метод).

Все директивы, используемые Objective-C начинаются с символа**@**

Классы.

Для объявления класса создается в заголовочном файле с расширением**.h**и использует синтаксис:

@interface имя_класса : super_Class

```
{
объявление переменных; (имеют зону видимости private)
}
объявление методов;
@end
```

Реализация класса создается в файле с расшерением**.m**и использует синтаксис:

```
#import "имя_класса.h"
@implementation имя_класса
реализация методов;
@end
```

Пример:

Файл first.h

```
#import <Cocoa/Cocoa.h>
@interface first : NSObject
{
int a;
int b;
}
- (id)init;
- (void)setA :(int)A andB :(int)B;
- (long)summa;
- (long)mul;
@end
```

Здесь мы видим объявление класса с именем**first**порожденный от класса**NSObject**(В Apple runtime библиотеке все классы являются наследниками класса NSObject, если мы будем писать программы с использованием Cocoa, то будет хорошей практикой наследовать свои классы также от NSObject, предоставляющего множество полезных возможностей).

Директива**#import**подключает к нашему объявлению заголовок Cocoa.h, содержащий объявления всех классов и протоколов пространства Cocoa.framework.

Далее следует объявление переменных типа Integer**a**и**b**, следом идут методы, начинающиеся с символа-(иногда с+), далее в скобках указывается возвращаемый тип данных, следом название метода, далее аргументы через пробел, перед скобками с типом аргумента ставится двоеточие, перед аргументом может стоять метка (в методе**setA**- меткой аргумента**B**является слово**andB**- наследие Smalltalk), используется для большей читабельности кода.

Файл first.m

```
#import "first.h"
@implementation first
- (id) init {
self = [super init];
if(self){
//инициализация вашего класса
}
return self;
}
- (void) setA :(int) A andB :(int)B{
a = A;
b = B;
}
- (long) summa{
return a+b;
}
```

```
- (long) mul{
return a*b;
}
@end
```

Начало файла реализации класса стандартно: с директивы**#import "first.h"**, загружающей объявление класса.

Директива**@implementation first**говорит компилятору, что далее следует реализация класса**first**

Далее следует функция**init**, которая служит для инициализации нашего класса. При создании экземпляра класса после выделения памяти всегда необходимо инициализировать класс, метод**init**- это перегруженный метод наследуемого класса(Супер класса) NSObject, поэтому в самом начале мы вызываем метод**init**наследуемого класса, для определения которого зарезервировано слово**super**. Значение, возвращаемое инициализатором, является указателем на наш класс, если инициализация неудачна, то возвращаемое значение равно*nil*.

self- зарезервированный указатель, который указывает на наш класс внутри самого класса.

Инициализатор можно и не перегружать, просто опустив его объявление и реализацию. Также класс может содержать несколько инициализаторов, также содержащих и аргументы.

Далее следует реализация методов, практически ничем не отличающаяся от обычного**C**.

В вышеуказанном примере переменные a и b имеют зону видимости только внутри своего класса. Мы написали метод setA:(int) andB:(int) для установки их значений. Если же нам необходимо прочитать значение a или b, то необходимо написать методы возвращающие их.

Например:

```
- (int)valueA{
return a;
}
```

Выглядит весьма запарно.

Свойства

Начиная с Objective C 2.0 появилось понятие - свойство, под свойство выделена директива**@property**, которая указывает, что данная переменная класса имеет метод для чтения и записи в нее. Данная директива объявляется в файле объявлении класса.

Сопутствующая директиве**@property**, директива говорящая компилятору, что необходимо создать методы для доступа к переменной носит наименование**@synthesize**и объявляется в файле реализации класса.

Синтаксис объявления свойства:>

@property (атрибуты, через запятую) тип имя;

Пример:

Файл myclass.h

```
@interface myclass
{
int a;
int b;
}
@property int a;
@property int b;

@end;
```

Как видим здесь объявляется класс myclass с двумя переменными a и b, доступными извне с помощью объявления свойств.

Файл myclass.m

```
#import "myclass.h"
@implementation myclass

@synthesize a;
@synthesize b;

@end
```

Вот так все стало проще. Теперь мы имеем для переменных методы, позволяющие считывать и записывать их значения.

Не надо забывать, что определение свойств сокращает лишь исходный код в редакторе, а компилятор создает методы, которые предоставляют механизм доступа к переменным класса, тем самым как раз увеличивая размер конечного приложения. Поэтому, если нет необходимости видимости переменной, то не стоит к ней объявлять и свойство.

Внимание.Начиная с Xcode 4.5 использование@synthesizeв файле реализации стало необязательным. В этом случае в файле - заголовке объявляется свойство@property NSInteger a;, доступ к самой переменной будет начинаться с символа подчеркивания_a, а доступ к сеттеру или геттеру внутри класса[self a].

Атрибуты свойств.

В случае необходимости можно переопределить методы установки или чтения переменной вместе называемые акцессорами (Accessor) и соответственно чтения - геттер (getter), записи - сеттер (setter):

getter=метод геттера

setter=метод сеттера

Бывает не всегда нужно предоставлять полный доступ к переменной класса, в этом случае нужно ограничить его в объявлении @property:

- **readwrite**- и чтение и запись
- **readonly**- только чтение

Свойство только для чтения будет выглядеть так: @property (readonly) int a;

Семантика сеттеров свойств.

strong- обозначает строгое переназначение объекта, делая указатель на объект владельцем этого объекта.

weak- в отличии от**strong**обозначает нестрогое соответствие и если объект был освобожден из памяти (из другого класса или потока), то значение установится в*nil*. Не поддерживается OS X v10.6 и iOS 4; используйте assign.

copy- указывает на то, что для присваивания будет использована копия переданного объекта. Первоначальному объекту посылается сообщение**release**. Может быть использовано только для объектов, поддерживающих протокол**NSCopying**, например NSString.

assign- для задания нового значения используется оператор присваивания. Используется только для скалярных типов (NSInteger и CGRect) либо для объектов, которыми мы не владеем.

retain- указывает на то, что для объекта, используемого в качестве нового значения instance-переменной, управление памятью происходит вручную (не забываем потом освободить память).

Atomic.

nonatomic- делает более быстрым способ доступа к объекту в немногозадачной среде, так-как в случае с**atomic**(по умолчанию все акцессоры **atomic**) и использованием strong, copy, или retain, во избежание коллизий с возможностью доступа к свойству из других потоков в сеттере строится код блокирующий переключение между потоками [2].

Список использованных источников

1. http://darkraha.com/rus/objective_c/index.php
2. http://macbug.ru/cocoa/objc1

УДК 003.09

ПРОБЛЕМА ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Польшикова Лариса Вячеславовна
Студентка 1-го курса, специальности Журналистика
ЕНУ им. Л.Н.Гумилева, Астана, Казахстан
Научный руководитель – Ж.Б.Ахаева, старший преподаватель

Весь мир связывает информация. Это любые сведения, которые беспрерывно передаются, обрабатываются, хранятся и используются каждым человеком. В современном обществе в связи с бурной информатизацией всё более актуальной становится проблема защиты этой информации. Обеспечение внутренней информационной безопасности является не только казахстанской, но и мировой проблемой. Если в первые годы с внедрением хакеров в локальные сети еще пытались справляться, то сегодня это становится всё труднее и труднее.

А обеспечить это - дорогое дело, и не только из-за затрат на закупку или установку средств защиты, но также из-за того, что трудно квалифицированно определить границы разумной безопасности и обеспечить соответствующее поддержание системы в работоспособном состоянии[1].

На сегодняшний день компьютерным системам доверяют самую ответственную работу, от качества выполнения которой зависит жизнь и благосостояние многих людей. ЭВМ управляют технологическими процессами на предприятиях и атомных электростанциях, управляют движением самолетов и поездов, выполняют финансовые операции, обрабатывают секретную и конфиденциальную информацию. И так можно сформулировать четыре принципа, которые должна обеспечить информационная безопасность[2,10]:

- целостность данных — защита от сбоев, ведущих к потере информации, а также зашита от неавторизованного создания или уничтожения данных;
- конфиденциальность информации;
- доступность информации для всех авторизованных пользователей;
- достоверность информации.

Для обеспечения национальной безопасности информации необходима система, включающая совокупность законодательных актов и созданных на их основе структур и механизмов взаимодействия по защите интересов субъектов правоотношений[3].

Также необходимо научное прогнозирование и моделирование решения возникающих проблем на альтернативной основе, законодательное регулирование и защита информационной деятельности при обеспечении достоверности, открытости и свободы информации, освобождение сознания людей от стереотипов тоталитаризма и прекращение психологической войны политического руководства против народа в любых формах ее проявления.

В итоге необходимо ускорение развития новых информационных технологий и их широкое распространение. А в Казахстане значит на основе международных стандартов создание своей системы безопасности. В арсенале специалистов по информационной безопасности Казахстана должен быть широкий спектр защитных мер: законодательных, морально-этических, административных, физических и технических средств. Все они обладают своими достоинствами и недостатками, которые необходимо знать и правильно учитывать при создании систем защиты. В Казахстане безопасность информации не