Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
14.05.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка приложения для учета посещаемых мест «Foot-Court»

БГУИР КР 1-40 05 01-10 № 149 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
14.05.2024
_____
(подпись студента)

2024

Реферат

БГУИР КР 1-40 05 01-10 № 149 ПЗ, гр. 814303

, Разработка приложения для учета посещаемых мест «Foot-Court», Минск: БГУИР - 2024.

Пояснительная записка 162437 с., 0 рис., 0 табл.

Ключевые слова:

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

Содержание

Введение

Введение Мобильные телефоны давно перестали быть чем-то необычным и великолепно справляются со своей функцией – являются средством коммуникации между людьми. При этом, недавно появившиеся, но уже прочно вошедшие в нашу жизнь смартфоны настолько функциональны, что трудно сказать, чего они не умеют: это и плеер, и фотоаппарат, и возможность использования Интернет-ресурсов, и прочее. По сути, все смартфоны стали небольшой копией компьютера, который постоянно можно иметь при себе. В наше время все больше и больше смартфонов, коммуникаторов, планшетных ПК и других видов устройств, удобных для использования как в повседневной жизни, так и в заграничных поездках в частности, выпускаются на базе двух самых распространённых операционных системах: ОС Android и iOS. И вопрос, какая операционная система лучше, породил множества конфликтов и споров. Однако для себя я выделил несколько основополагающих преимуществ ОС от Apple по сравнению с ОС Android. Качество приложений Приложения практически всегда выглядят привлекательнее, удобнее на iPhone и iPad. Быстрое обновление Пользователям iPhone и iPad не нужно ждать, когда производители удосужатся подготовить обновление для их устройства после релиза новой версии iOS.. Обновления для iOS выходят регулярно и в один момент становятся доступны для всех совместимых устройств. Продолжительная поддержка старых устройств Период поддержки мобильных телефонов Apple составляет 48 месяцев. Доступ к iOS 13 получили даже пользователи смартфона iPhone 5s. Конечно, на устройстве доступны не все возможности ОС, однако пользователи могут использовать ряд важных функций мобильной платформы и запускать приложения из App Store, совместимые только с iOS 13. Лучшие приложения доступны первыми Большинство разработчиков решают сначала выпустить приложение на iPhone и iPad и только через некоторое время запустить его на Android. Это связано с хорошо проработанным инструментарием разработки для iOS. Например, Instagram, одна из крупнейших социальных сетей мира, более года была доступна только на iPhone. И только потом ее запустили для Android. Экосистема Apple Сегодня при выборе смартфона или планшета главными являются не различные технические характеристики вроде емкости батареи, разрешения камеры и т.д. Они более или менее схожи на всех современных устройствах. Главное для пользователя – это мобильная экосистема. И на iOS она гораздо более развита. Дружественный интерфейс iPhone стал массовым среди людей потому, что он удобен. Люди, чье время дорого, делают такой выбор, потому что iPhone позволяет экономить

время, получать удовольствие от использования аппарата и сосредоточиться на том, что хочется делать: общаться с нужными людьми, писать сообщения, пользоваться социальными сетями, и делать это все беспрепятственно. А не сражаться с интерфейсом. И аппараты, и программное обеспечение iPhone выглядят достаточно хорошо. Надежность iPhone значительно надежнее, так как производитель годами оттачивает нюансы производства одной модели и одной операционной системы, каждый апгрейд которой становится надежнее, проще, мощнее, производительнее. Масштаб продаж в сотни миллионов устройств позволил Apple отточить технологии производства и ПО до мелочей. Например, даже старый iPhone порадует достаточно быстрой работой и полным отсутствием глюков, в то время как аппараты Android после полутора-двух лет службы начинают тормозить, зависать. Семейный доступ «Семейный доступ» – очень полезная функция, которая позволяет экономить на покупках в онлайн-сервисах Apple. Так, объединив до шести Apple ID, пользователи могут загружать купленные приложения, музыку, песни на все подключенные устройства. Таким образом, человек тратит деньги на контент один раз, а пользоваться им могут несколько людей с разными аккаунтами. Большая безопасность Эксперты утверждают, что мобильная платформа iOS намного защищеннее операционной системы Google перед большинством видов существующих атак. Связано это в немалой степени со строгим цензурированием App Store. Google относится к этому более свободно, поэтому большинство вредоносных программ для Android интегрируется непосредственно в официальные приложения. Аналитики прогнозируют, что количество активных Android-зловредов будет увеличиваться стремительными темпами. Continuity Одной из полезных функций iOS является возможность объединить работу обновленных до последней операционной системы мобильных устройств. Пользователи могут отвечать на входящие звонки с iPhone через iPad, если и смартфон, и планшет подключены к одному Apple ID. Можно начать веб-серфинг, набирать SMS-сообщение или электронное письмо на iPad, а закончить – на iPhone. Другим немаловажным бонусом является возможность использовать iPhone в качестве модема, если тот находится поблизости от того же iPad. Для этого даже не нужно включать на смартфоне режим модема. Учитывая все вышеперечисленные плюсы, в данной курсовой работе я решил разрабатывать приложение для учета посещаемых мест именно для платформы от компании Apple, а не Android. Цель данной работы – создать удобное, юзабельное и приятное глазу приложение-заметки, которое могло бы запоминать и хранить посещаемые пользователем места, а после удобно их отображать.

1 Разработка приложения для учета посещаемых мест «Foot-Court»

Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем Дисциплина «Объектно-ориентированное программирование»

К защите допустить:

Руководитель курсовой работы

старший преподаватель кафедры

_____А. В. Михалькевич

___.___. 20___

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе на тему

Разработка приложения для учета посещаемых мест «Foot-Court»

БГУИР КР 1-40 05 01-10 №_

Студент Вареник А. А.

(подпись студента)

Курсовая работа

представлена на проверку

___.___. 20___

(подпись студента)

Минск 2020

Оглавление

1. Описание проекта

При запуске приложения на экран отображается список раннее сохраненных пользователем мест (рис. 1). Ячейки списка отображают описание сохраненных мест. Описание представлено фотографией, названием, типом места и рейтингом пользователя. Также на главном экране есть кнопки сортировки по алфавиту (вкладка Name), строка поиска мест из списка заранее сохраненных, кнопка добавления нового места (вверху справа) и кнопка загрузки топ мест в текущей геолокации. При взмахивании влево по ячейке появляется кнопка удаления места (красная кнопка delete).

Рис. 1 – Список сохраненных мест

При нажатии на ячейку с описанием сохраненного места появляется экран изменения описания выбранного места (рис. 2). В одноименных полях можно изменить имя, локацию, тип места и рейтинг по мнению пользователя. При клике по фотографии всплывет окно с выбором места новой фотографии (галерея или сделать новый снимок) т. е. от куда мы можем взять новое фото.

Рис. 2 – Экран изменения описания места

Также в поле изменения локации есть кнопка, при нажатии на которую откроется карта. (рис. 3). Это предусмотрено для того, чтобы пользователь мог выбрать ему удобный способ ввода локации (вручную или через карту).

Рис. 3 – Карта ввода новой локации

Если кликнуть по иконке с изображением карты на экране изменения описания места, в области отображения фотографии, откроется карта с расположением локации места. Можно при нажатии на соответствующие кнопки найти свое местоположение на карте и построить маршрут до текущего места (рис. 4).

Рис. 4 – Построение маршрута к выбранному месту

Изменение можно сохранить нажатием на кнопку Save на экране изменения описания места.

При нажатии кнопки добавление нового места на экране списка сохраненных мест (плюс) снизу выпадает экран добавления нового места (рис. 5). По функционалу он идентичен экрану изменения описания места. Но существенное отличие состоит в том, что новую запись

невозможно добавить без заполнения поля имени места.

Рис. 5 – Экран добавления нового места

Также при нажатии кнопки загрузки топа мест в текущей геолокации на экране списка сохраненных мест отобразиться подпорка мест с сайта tripadvisor.com, которые располагаются поблизости.

2. Выбранные технологии

Для разработки мобильных приложений компания Apple предлагает два языка программирования: более новый Swift и достаточно старый Objective – C.

Objective - C — компилируемый объектно-ориентированный язык программирования, используемый корпорацией Apple, построенный на основе языка Си и парадигм Smalltalk. В частности, объектная модель построена в стиле Smalltalk — то есть объектам *посылаются сообщения*.

Язык Objective-C является надмножеством языка Си, поэтому Си-код полностью понятен компилятору Objective-C.

Компилятор Objective-C входит в GCC и доступен на большинстве основных платформ. Язык используется в первую очередь для Mac OS X (Cocoa) и GNUstep — реализаций объектно-ориентированного интерфейса OpenStep. Также язык используется для iOS (Cocoa Touch).

Objective-C является расширением C: любая программа на C является программой на Objective-C.

Одной из отличительных черт Objective-C является динамичность: решения, обычно принимаемые на этапе компиляции, здесь откладываются до этапа выполнения.

Objective-C — message-oriented-язык, в то время как C++ — function-oriented: в Objective-C вызовы метода интерпретируются не как вызов функции (хотя к этому обычно все сводится), а как посылка сообщения (с именем и аргументами) объекту, подобно тому, как это происходит в Smalltalk.

Любому объекту можно послать любое сообщение. Объект может вместо обработки сообщения переслать его другому объекту для обработки (делегирование), в частности, так можно реализовать распределённые (то есть находящиеся в различных адресных пространствах и даже на разных компьютерах) объекты.

Привязка сообщения к соответствующей функции происходит на этапе выполнения.

Язык Objective-C поддерживает работу с метаинформацией — так, на этапе выполнения можно узнать класс объекта, список его методов (с типами передаваемых аргументов) и instance-переменных, проверить, является ли класс потомком заданного и поддерживает ли он заданный протокол и т. п.

В языке есть поддержка протоколов (понятия интерфейса объекта и протокола четко разделены). Поддерживается наследование (не множественное); для протоколов поддерживается множественное наследование. Объект может быть унаследован от другого

объекта и сразу нескольких протоколов (хотя это скорее не наследование протокола, а его поддержка).

На данный момент язык Objective-C поддерживается компиляторами Clang и GCC (под управлением Windows используется в составе MinGW или cygwin).

Некоторые функции языка перенесены в runtime-библиотеку и сильно зависят от неё. Вместе с компилятором gcc поставляется минимальный вариант такой библиотеки. Также можно свободно скачать runtime-библиотеку компании Apple: Apple's Objective-C runtime.

Эти две runtime-библиотеки похожи (основные отличия в именах методов).

Swift — открытый мультипарадигмальный компилируемый язык программирования общего назначения. Создан компанией Apple в первую очередь для разработчиков iOS и macOS. Swift работает с фреймворками Cocoa и Cocoa Touch и совместим с основной кодовой базой Apple, написанной на Objective-C. Swift задумывался как более лёгкий для чтения и устойчивый к ошибкам программиста язык, нежели предшествовавший ему Objective-C. Программы на Swift компилируются при помощи LLVM, входящей в интегрированную среду разработки Xcode 6 и выше. Swift может использовать рантайм Objective-C, что делает возможным использование обоих языков (а также C) в рамках одной программы.

Swift заимствовал довольно многое из Objective-C, однако он определяется не указателями, а типами переменных, которые обрабатывает компилятор. По аналогичному принципу работают многие скриптовые языки. В то же время, он предоставляет разработчикам многие функции, которые прежде были доступны в C++ и Java, такие как определяемые наименования, обобщения и перегрузка операторов.

Часть функций языка выполняется быстрее по сравнению с другими языками программирования. Например, сортировка комплексных объектов выполняется в 3,9 раз быстрее, чем в Python, и почти в 1,5 раза быстрее, чем в Objective-C.

Код, написанный на Swift, может работать вместе с кодом, написанным на языках программирования C и Objective-C в рамках одного и того же проекта[2].

Для написания данного приложения я выбрал Swift по следующим причинам:

1. Для написания приложения требуется меньше кода, хотя бы просто потому, что здесь реализован упрощенный принцип работы с повторяющимися строками и заявлениями;
2. Удобен для чтения. Стандартное достоинство любого современного языка;
3. Больше возможностей по сравнению с Objective-C, в частности возможность управлять памятью;
4. Полноценное взаимодействие с кодом, написанным на Objective-C;
5. Повышенная безопасность. Это выражается и в обработке указателей, и в дотошности компилятора, и в том, что в саму компиляцию можно встроить опциональную переменную nil для обеспечения обратной связи.
6. Swift изначально был разработан для замены Objective – C. В связи с этим Apple стремиться больше поддерживать и развивать Swift по сравнению с Objective – C.

Уже в экосистеме Swift были выбраны следующие Фреймворки для написания приложения: UIKit и Foundation для создания внешнего вида и логики перехода между экранами, Cosmos Kit

для удобного интегрирования шкалы рейтинга в приложение и Фреймворк MapKit для использования карты в приложении.

Основным и самым используемым Фреймворком в разработанном приложении послужил UIKit. Ниже представленные основные достоинства, по которым он был выбран:

- хранение всех UI-компонентов в одном месте облегчает их поиск; соответственно, работа становится более организованной;

освобождение классов от зависимостей помогает сократить время компиляции;
устранение зависимостей способствует повторному использованию компонентов и тоже ускоряет компиляцию;
обновляя компонент в UIKit, мы обновляем его повсюду; если приложение использует компоненты из UIKit, упрощается процесс внесения изменений в компоненты по всему приложению;
изолированные компоненты гораздо легче тестировать;
отдельный фреймворк при необходимости можно использовать и в других приложениях (например, при создании приложения — каталога всех компонентов данного фреймворка).

3. Основные инструменты

Для написания программного кода под данное приложение использовалась официально поддерживаемая Apple среда разработки XCode 11.

Xcode — интегрированная среда разработки (IDE) программного обеспечения для платформ macOS, iOS, watchOS и tvOS, разработанная корпорацией Apple. Первая версия выпущена в 2003 году. Стабильные версии распространяются бесплатно через Mac App Store. Зарегистрированные разработчики также имеют доступ к бета-сборкам через сайт Apple Developer.

Xcode включает в себя большую часть документации разработчика от Apple и Interface Builder — приложение, использующееся для создания графических интерфейсов.

Пакет Xcode включает в себя изменённую версию свободного набора компиляторов GNU Compiler Collection и поддерживает языки C, C++, Objective-C, Objective-C++ (англ.)русск., Swift, Java, AppleScript, Python и Ruby с различными моделями программирования, включая (но не ограничиваясь) Cocoa, Carbon и Java. Сторонними разработчиками реализована поддержка GNU Pascal, Free Pascal, Ada, C#, Perl, Haskell и D. Пакет Xcode использует GDB в качестве back-end'а для своего отладчика.

В августе 2006 Apple объявила о том, что DTrace, фреймворк динамической трассировки от Sun Microsystems, выпущенный как часть OpenSolaris, будет интегрирован в Xcode под названием Xray. Позже Xray был переименован в Instruments.

3 июня 2019 года на WWDC 2019 была представлена бета-версия нового Xcode 11. Появилась поддержка портирования специализированных приложений созданных для интернет-планшета iPad на настольную macOS. Были доработаны и усовершенствованны функции основных API, например такие как: фреймворки для машинного обучения Core ML (англ.)русск. и Create ML (англ.)русск., фреймворк для работы с GPU Metal (англ.)русск. и

другие. Для разработчиков дополненной реальности появились: новое приложение Reality Composer (англ.)русск. и новый высокоуровневый фреймворк RealityKit (англ.)русск., а также новая версия фреймворка ARKit (англ.)русск. . Появился совершенно новый API FileProvider (англ.)русск. для провайдеров облачных хранилищ, для высокопроизводительного способа бесшовной интеграции с Finder без расширения ядра и для повышения безопасности.

Для хранение выбранных в приложении мест была выбрана база данных Realm.

Realm - это система управления объектной базой данных с открытым исходным кодом , изначально для мобильных устройств (Android / iOS), также доступная для таких платформ, как Xamarin или React Native, и другие, включая настольные. приложений (Windows) и лицензируется по лицензии Apache .

В сентябре 2016 года была объявлена **мобильная платформа Realm** , за которой последовал первый стабильный выпуск в январе 2017 года. Она позволяет осуществлять двустороннюю синхронизацию между Realm Object Server, и базами данных на стороне клиента, которые принадлежат указанному журналу. -в пользователе. Была выпущена как разработка, так и коммерческое издание вместе с бизнес-лицензией для интеграции с другими системами управления базами данных, такими как PostgreSQL.

24 апреля 2019 года Realm объявил, что они заключили окончательное соглашение, которое будет приобретено MongoDB.

Наиболее заметные особенности Realm:

> Поскольку Realm является хранилищем объектов, его типизированные API для конкретных языков отображают типизированные объекты непосредственно в файл Realm, поэтому классы используются в качестве определения схемы.
> Отношения между объектами разрешены через «ссылки». Каждая «ссылка» создает «обратную ссылку» как обратную связь с какими-либо объектами, связывающимися с текущим объектом.
> Результаты запроса, возвращаемые Realm, являются локальными представлениями потока для текущей «версии базы данных» (поскольку Realm обрабатывает параллелизм с архитектурой MVCC), и эти представления «автоматически обновляются», когда транзакция фиксируется из *любого потока* , если Realm в состоянии обновить версию своего экземпляра (что возможно в потоках, которые могут получать уведомления об изменениях). Когда это происходит, Realm вызывает изменения слушателей, которые добавляются к его результатам запроса (если они изменились).
> Каждое локальное представление потока возвращает прокси-объекты, которые только читают / записывают в базу данных при вызове метода доступа, что означает, что весь доступ к базе данных загружен с отложенной загрузкой. Обратите внимание, что запись разрешена только во время транзакции записи.
> Поскольку каждый результат запроса и каждый прокси-объект является представлением базовых данных, любое изменение, внесенное в базу данных, отражается во всех объектах, которые указывают на одни и те же данные. Realm обычно называет это поведение «архитектурой с нулевым копированием» (наряду с ранее упомянутым доступом к данным с отложенной загрузкой).

Realm разрабатывается в основном с открытым исходным кодом на GitHub , где доступен

исходный код для привязок к конкретному языку и основная база данных. Однако аспекты Realm, относящиеся к функции синхронизации, являются собственностью закрытого источника.

Ну и наконец для сохранности кода и , в ходе разработки, возможности вернуться к предыдущей версии приложения в случае чего, использовалась система контроля версий Git.

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером оболочки к репозиториям Git, а StGit использует Git для управления коллекцией исправлений (патчей).

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и даёт возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создаёт в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «`.git`» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создаётся рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда `commit`).

Полный код данного приложение можно найти, перейдя по следующей ссылке: https://github.com/AnVaran/Foot-Court

4. Объектно-ориентированное программирование

История ООП

ООП возникло в результате развития идеологии процедурного программирования, где данные и подпрограммы (процедуры, функции) их обработки формально не связаны. Для дальнейшего развития объектно-ориентированного программирования часто большое значение имеют понятия события (так называемое событийно-ориентированное программирование) и компонента (компонентное программирование, КОП).

Взаимодействие объектов происходит посредством сообщений. Результатом дальнейшего развития ООП, по-видимому, будет агентно-ориентированое программирование, где *агенты* — независимые части кода на уровне выполнения. Взаимодействие агентов происходит посредством изменения *среды*, в которой они находятся.

Языковые конструкции, конструктивно не относящиеся непосредственно к объектам, но

сопутствующие им для их безопасной (исключительные ситуации, проверки) и эффективной работы, инкапсулируются от них в аспекты (в аспектно-ориентированном программировании). Субъектно-ориентированное программирование расширяет понятие объекта посредством обеспечения более унифицированного и независимого взаимодействия объектов. Может являться переходной стадией между ООП и агентным программированием в части самостоятельного их взаимодействия.

Первым языком программирования, в котором были предложены основные понятия, впоследствии сложившиеся в парадигму, была Симула, но термин «объектная ориентированность» не использовался в контексте использования этого языка. В момент его появления в 1967 году в нём были предложены революционные идеи: объекты, классы, виртуальные методы и др., однако это всё не было воспринято современниками как нечто грандиозное. Фактически, Симула была «Алголом с классами», упрощающим выражение в процедурном программировании многих сложных концепций. Понятие класса в Симуле может быть полностью определено через композицию конструкций Алгола (то есть класс в

Симуле — это нечто сложное, описываемое посредством примитивов).

Взгляд на программирование «под новым углом» (отличным от процедурного) предложили Алан Кэй и Дэн Ингаллс в языке Smalltalk. Здесь понятие класса стало основообразующей идеей для всех остальных конструкций языка (то есть класс в Смолтоке является примитивом, посредством которого описаны более сложные конструкции). Именно он стал первым широко распространённым объектно-ориентированным языком программирования.

В настоящее время количество прикладных языков программирования (список языков), реализующих объектно-ориентированную парадигму, является наибольшим по отношению к другим парадигмам. Наиболее распространённые в промышленности языки (C++, Delphi, C#, Java и др.) воплощают объектную модель Симулы. Примерами языков, опирающихся на модель Смолтока, являются Objective-C, Python, Ruby.

Технологии программирования

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Необходимо обратить внимание на следующие важные части этого определения:

1. объектно-ориентированное программирование использует в качестве основных логических конструктивных элементов объекты, а не алгоритмы;
2. каждый объект является экземпляром определенного класса;
3. классы образуют иерархии. Программа считается объектно-ориентированной, только если выполнены все три указанных требования. В частности, программирование, не использующее наследование, называется не объектно-ориентированным, а программированием с помощью абстрактных типов данных.

Объектно-ориентированное программирование (ООП) позволяет разложить проблему на составные части, каждая из которых становится самостоятельным объектом. Каждый из объектов содержит свой собственный код и данные, которые относятся к этому объекту.

Любая программа, написанная на языке ООП, отражает в своих данных состояние физических предметов либо абстрактных понятий – объектов программирования, для работы, с которыми она предназначена.

Все данные об объекте программирования и его связях с другими объектами можно объединить в одну структурированную переменную. В первом приближении ее можно назвать объектом.

С объектом связывается набор действий, иначе называемых методами. С точки зрения языка программирования набор действий или методов – это функции, получающие в качестве обязательного параметра указатель на объект и выполняющие определенные действия с данными объекта программирования. Технология ООП запрещает работать с объектом иначе, чем через методы, таким образом, внутренняя структура объекта скрыта от внешнего пользователя.

Описание множества однотипных объектов называется классом.

Объект – это структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии.

Класс – это описание множества объектов программирования (объектов) и выполняемых над ними действий.

Класс можно сравнить с чертежом, согласно которому создаются объекты. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области решаемой задачи.

Любая функция в программе представляет собой метод для объекта некоторого класса.

Класс должен формироваться в программе естественным образом, как только в ней возникает необходимость описания новых объектов программирования. Каждый новый шаг в разработке алгоритма должен представлять собой разработку нового класса на основе уже существующих.

Вся программа в таком виде представляет собой объект некоторого класса с единственным методом *run* (выполнить).

Программирование «от класса к классу» включает в себя ряд новых понятий. Основными понятиями ООП являются

инкапсуляция;
наследование;
полиморфизм.

Инкапсуляция данных (от "капсула") – это механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В ООП код и данные могут быть объединены вместе (в так называемый «черный ящик») при создании объекта.

Внутри объекта коды и данные могут быть закрытыми или открытыми.

Закрытые коды или данные доступны только для других частей того же самого объекта и,

соответственно, недоступны для тех частей программы, которые существуют вне объекта.

Открытые коды и данные, напротив, доступны для всех частей программы, в том числе и для других частей того же самого объекта.

Наследование. Новый, или производный класс может быть определен на основе уже имеющегося, или базового класса.

При этом новый класс сохраняет все свойства старого: данные объекта базового класса включаются в данные объекта производного, а методы базового класса могут быть вызваны для объекта производного класса, причем они будут выполняться над данными включенного в него объекта базового класса.

Иначе говоря, новый класс наследует как данные старого класса, так и методы их обработки.

Если объект наследует свои свойства от одного родителя, то говорят об одиночном наследовании. Если объект наследует данные и методы от нескольких базовых классов, то говорят о множественном наследовании.

Пример наследования – определение структуры, отдельный член которой является ранее определенной структурой.

Рисунок 13 – Структура «Наследование»

Полиморфизм – это свойство, которое позволяет один и тот же идентификатор (одно и то же имя) использовать для решения двух и более схожих, но технически разных задач.

Целью полиморфизма, применительно к ООП, является использование одного имени для задания действий, общих для ряда классов объектов. Такой полиморфизм основывается на возможности включения в данные объекта также и информации о методах их обработки (в виде указателей на функции).

Будучи доступным в некоторой точке программы, объект, даже при отсутствии полной информации о его типе, всегда может корректно вызвать свойственные ему методы.

Полиморфная функция – это семейство функций с одним и тем же именем, но выполняющие различные действия в зависимости от условий вызова.

Например, нахождение абсолютной величины в языке Си требует трех разных функций с разными именами:

Int abs(int); long int labs(long int); double fabs(double);

В C++ можно описать полиморфную функцию, которая будет иметь одинаковое имя и разные типы и наборы аргументов.

Объектно-ориентированная технология создания программ основывается на так называемом объектном подходе. Одним из проявлений этого подхода является то, что сначала довольно долго создаются и оптимизируются объектная модель и иные модели и лишь затем осуществляется кодирование.

Обычно проектируемая программная система первоначально представляется в виде трех взаимосвязанных моделей:

1. объектной модели, которая представляет статические, структурные аспекты системы;

2. динамической модели, которая описывает работу отдельных частей системы;
3. функциональной модели, в которой рассматривается взаимодействие отдельных частей системы (как по данным, так и по управлению) в процессе ее работы.

Эти три вида моделей должны позволить рассматривать три взаимно-ортогональных представления системы в одной системе обозначений.

Объектная модель на более поздних этапах проектирования дополняется моделями, отражающими как логическую (классы и объекты), так и физическую структуру системы (процессы и деление на компоненты, файлы или модули).

Поскольку при разработке объектно-ориентированного проекта используется множество моделей, которые необходимо увязать в единое целое, далее в гл. 10 рассматриваются средства автоматизации составления, верификации (проверки) и графической визуализации этих моделей.

Процесс построения *объектной модели* включает в себя следующие, возможно, повторяющиеся до достижения приемлемого качества модели этапы:

1. определение объектов;
2. подготовку словаря объектов с целью исключения схожих (синонимичных) понятий и уточнения имен, классификацию объектов, выделение классов;
3. определение взаимосвязей между объектами;
4. определение атрибутов объектов и методов (определение уровней доступа и проектирование интерфейсов классов);
5. исследование качества модели.

Теперь, используя функциональную модель, можно начинать работу с динамической моделью, наделяя объекты необходимыми методами и данными.

Модели, разработанные на первой фазе жизненного цикла системы, продолжают использоваться на всех последующих его фазах, облегчая программирование системы, ее отладку и тестирование, сопровождение и дальнейшую модификацию.

Объектная модель описывает структуру объектов, составляющих систему, их атрибуты, операции, взаимосвязи с другими объектами. В объектной модели должны быть отражены те понятия и объекты реального мира, которые важны для разрабатываемой системы. В объектной модели отражается прежде всего прагматика разрабатываемой системы, что выражается в использовании терминологии прикладной области, связанной с использованием разрабатываемой системы.

Прагматика определяется целью разработки программной системы: для обслуживания покупателей железнодорожных билетов, управления работой аэропорта, обслуживания чемпионата мира по футболу и т. п. В формулировке цели участвуют предметы и понятия реального мира, имеющие отношение к разрабатываемой программной системе.

Объектную модель можно описать следующим образом:

1. основные элементы модели — объекты и сообщения;
2. объекты создаются, используются и уничтожаются подобно динамическим переменным в обычных языках программирования;
3. выполнение программы заключается в создании объектов и передаче им последовательности

сообщений.

Объектная модель базируется на четырех главных принципах: абстрагировании; инкапсуляции; модульности; иерархии.

Эти принципы являются *главными* в том смысле, что без любого из них модель не будет по-настоящему объектно-ориентированной.

Абстрагирование концентрирует внимание на внешних особенностях объекта и позволяет отделить самые существенные особенности поведения от несущественных. Выбор правильного набора абстракций для заданной предметной области представляет собой главную задачу объектно-ориентированного проектирования.

Все абстракции обладают как статическими, так и динамическими свойствами. Например, файл как объект требует определенного объема памяти на конкретном устройстве, имеет имя и содержание. Эти атрибуты являются статическими свойствами. Конкретные же значения каждого из перечисленных свойств динамичны и изменяются в процессе использования объекта: файл можно увеличить или уменьшить, изменить его имя и содержимое.

Абстракция и инкапсуляция дополняют друг друга: абстрагирование направлено на наблюдаемое поведение объекта, а инкапсуляция занимается внутренним устройством. Чаще всего инкапсуляция дополняется сокрытием информации, т. е. маскировкой всех внутренних деталей, не влияющих на внешнее поведение. Объектный подход предполагает, что собственные ресурсы, которыми могут манипулировать только методы самого объекта, скрыты от внешних компонент.

При объектно-ориентированном проектировании необходимо физически разделить классы и объекты, составляющие логическую структуру проекта. Такое разделение делает возможным повторно использовать во все новых проектах код модулей, написанных ранее. Модулю в данном контексте соответствует отдельный файл исходного текста. На выбор разбиения на модули могут влиять и некоторые внешние обстоятельства. При коллективной разработке программ распределение работы осуществляется, как правило, по модульному принципу, и правильное разделение проекта минимизирует связи между участниками.

Таким образом, принципы абстрагирования, инкапсуляции и модульности являются взаимодополняющими. Объект логически определяет границы определенной абстракции, а инкапсуляция и модульность делают их физически незыблемыми.

Имея выявленные объекты, можно приступить к выявлению классов. Классы чаще всего строятся постепенно, начиная от простых родительских классов и заканчивая более сложными. Непрерывность процесса основана на наследовании. Каждый раз, когда из предыдущего класса производится последующий, производный класс наследует какие-то или все родительские качества, добавляя к ним новые. Завершенный проект может включать десятки и сотни классов, но часто все они произведены от считанного количества родительских классов.

Стиль решения задачи зависит от того, на сколько сложен будет алгоритм её решения. Если вы, к примеру, можете её решить выполнив одно действие потом другое, затем, если не получится сделать третье, то вы выполните четвёртое — это процедурный стиль или по-другому — процедурное программирование (ПП). Инструкции для решения задачи выполняются одна

за другой, сверху вниз, иногда возникают изменения в их последовательности. Среди команд нельзя выделить приоритетные, для решения задачи они просто должны быть выполнены.

Решение задачи в объектно-ориентированном стиле, применяя объектно-ориентированное программирование (ООП), оказывается ближе к реальности. Сложные задачи — это стихия ООП. В решении задачи участвуют программные объекты и ответственность за решение делится между ними. Часто, в процессе можно найти некоторые паттерны, поэтому при решении задачи некоторый участок кода используется многократно. Результат достигается путём распределения ответственности между объектами программы и повторным использования промежуточных решений.

Недостатки объектно-ориентированного программирования

1. ООП порождает огромные иерархии классов, что приводит к тому, что функциональность расползается или, как говорят, размывается по базовым и производным членам класса, и отследить логику работы того или иного метода становится сложно.
2. В некоторых языках все данные являются объектами, в том числе и элементарные типы, а это не может не приводить к дополнительным расходам памяти и процессорного времени.
3. Также, на скорости выполнения программ может неблагоприятно сказаться реализация полиморфизма, которая основана на механизмах позднего связывания вызова метода с конкретной его реализацией в одном из производных классов.
4. Психологический аспект. Многие считают, что ООП это круто и начинают использовать его подходы всегда и везде и без разбору. Все это приводит к снижению производительности программ в частности и дискредитации ООП в целом

Реализация объектно-ориентированных технологий программирования в современных программно-математических средах

Оптимизационную задачу формализуют и рассматривают как математическую. Для решения таких задач используются различные численные методы, которые реализуются на персональных компьютерах с помощью языков программирования высокого уровня (*Pascal, C*) или специализированного программного обеспечения: электронной таблицы *Excel*, математического пакета *MathCAD*.

MathCAD — система компьютерной алгебры из класса систем автоматизированного проектирования, ориентированная на подготовку интерактивных документов с вычислениями и визуальным сопровождением, отличается лёгкостью использования и применения для коллективной работы.

MathCAD — математически ориентированные универсальные системы. Помимо собственно вычислений они позволяют с блеском решать задачи, которые с трудом поддаются популярным текстовым редакторам или электронным таблицам. С их помощью можно не только качественно подготовить тексты статей, книг, диссертаций, научных отчетов, дипломных и курсовых проектов, они, кроме того, облегчают набор самых сложных математических формул

и дают возможность представления результатов, в изысканном графическом виде. Только *Mathematica* и *MathCAD* обладают современными средствами визуализации представления данных. И запись в системе *MathCAD* наиболее приближена к записи математических задач без применения компьютера.

Пакет *MatLab* представляет собой современное программное средство для матричных вычислений. Пакет развивался, ориентируясь на различных потребителей. В настоящее время – это продукт высокого качества, включающий в себя вычисления, визуализацию и программирование в удобном виде, где задачи и их решения выражаются в форме, близкой к математической.

MatLab представляет собой стандартный инструмент для работы в различных областях математики и других наук. В промышленности *MatLab* – это инструмент для исследований, разработки и анализа данных.

MathCAD содержит сотни операторов и встроенных функций для решения различных технических задач. Программа позволяет выполнять численные и символьные вычисления, производить операции со скалярными величинами, векторами и матрицами, автоматически переводить одни единицы измерения в другие.

В среде *MathCAD* фактически нет графиков функций в математическом понимании термина, а есть визуализация данных, находящихся в векторах и матрицах (то есть осуществляется построение как линий, так и поверхностей по точкам с интерполяцией), хотя пользователь может об этом и не знать, поскольку у него есть возможность использования непосредственно функций одной или двух переменных для построения графиков или поверхностей соответственно. Так или иначе, механизм визуализации *MathCAD* значительно уступает таковому у *Maple*, где достаточно иметь только вид функции, чтобы построить график или поверхность любого уровня сложности. По сравнению с *Maple*, графика *MathCAD* имеет ещё такие недостатки, как: невозможность построения поверхностей, заданных параметрически, с непрямоугольной областью определения двух параметров; создание и форматирование графиков только через меню, что ограничивает возможности программного управления параметрами графики. Среди возможностей *MathCAD* можно выделить:

- Решение дифференциальных уравнений, в том числе и численными методами
- Построение двумерных и трёхмерных графиков функций (в разных системах координат, контурные, векторные и т. д.)
- Использование греческого алфавита как в уравнениях, так и в тексте
- Выполнение вычислений в символьном режиме
- Выполнение операций с векторами и матрицами
- Символьное решение систем уравнений
- Аппроксимация кривых
- Выполнение подпрограмм
- Поиск корней многочленов и функций
- Проведение статистических расчётов и работа с распределением вероятностей
- Поиск собственных чисел и векторов
- Вычисления с единицами измерения
- Интеграция с САПР-системами, использование результатов вычислений в качестве управляющих параметров

С помощью *MathCAD* инженеры могут документировать все вычисления в процессе их проведения.

Несмотря на то, что эта программа, в основном, ориентирована на пользователей-непрограммистов, *MathCAD* также используется в сложных проектах, чтобы визуализировать результаты математического моделирования путём использования распределённых вычислений и традиционных языков программирования. Также *MathCAD* часто используется в крупных инженерных проектах, где большое значение имеет трассируемость и соответствие стандартам.

MathCAD достаточно удобно использовать для обучения, вычислений и инженерных расчетов. Открытая архитектура приложения в сочетании с поддержкой технологий *.NET* и *XML* позволяют легко интегрировать *MathCAD* практически в любые ИТ-структуры и инженерные приложения. Есть возможность создания электронных книг (*e-Book*).

5. MVC

Model-View-Controller (**MVC**, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

> ***Модель*** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
> ***Представление*** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
> ***Контроллер*** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Основная цель применения этой концепции состоит в отделении бизнес-логики (*модели*) от её визуализации (*представления*, *вида*). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи:

К одной *модели* можно присоединить несколько *видов*, при этом не затрагивая реализацию *модели*. Например, некоторые данные могут быть одновременно представлены в

виде электронной таблицы, гистограммы и круговой диаграммы;

Не затрагивая реализацию *видов*, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных) — для этого достаточно использовать другой *контроллер*;

Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики (*модели*), вообще не будут осведомлены о том, какое *представление* будет использоваться.

Концепция MVC позволяет разделить модель, представление и контроллер на три отдельных компонента:

Модель

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает, как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем), просто предоставляя доступ к данным и управлению ими.

Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «наблюдателей».

Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

Представление

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Функциональные возможности и расхождения

Поскольку MVC не имеет строгой реализации, то реализован он может быть по-разному. Нет общепринятого определения, где должна располагаться бизнес-логика. Она может находиться как в контроллере, так и в модели. В последнем случае, модель будет содержать все бизнес-объекты со всеми данными и функциями.

Некоторые фреймворки жестко задают где должна располагаться бизнес-логика, другие не имеют таких правил.

Также не указано, где должна находиться проверка введённых пользователем данных. Простая валидация может встречаться даже в представлении, но чаще они встречаются в контроллере или модели.

Интернационализация и форматирование данных также не имеет четких указаний по расположению.

Схема алгоритма, диаграмма последовательности и диаграмма состояний приведены в Приложении пояснительной записки.

5. Шаблон проектирования практических задач

Шаблон проектирования или **паттерн** (англ. *design pattern*) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют **архитектурные шаблоны**, они охватывают собой архитектуру всей программной системы.

Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи.

Плюсы

В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем. Шаблон даёт решению своё имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода. Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования.

Минусы

Хотя легкое изменение кода под известный шаблон может упростить понимание кода, по

мнению Стива Макконнелла, с применением шаблонов могут быть связаны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

Многие шаблоны проектирования в объектно-ориентированном проектировании можно рассматривать как идиоматическое воспроизведение *элементов* функциональных языков. Питер Норвиг утверждает, что 16 из 23 шаблонов, описанных в книге «Банды четырёх», в динамически-типизируемых языках реализуются существенно проще, чем в C++, либо оказываются незаметны. Пол Грэхэм считает саму идею шаблонов проектирования — антипаттерном, сигналом о том, что система не обладает достаточным уровнем абстракции, и необходима её тщательная переработка. Нетрудно видеть, что само определение шаблона как «*готового решения, но не прямого обращения к библиотеке*» по сути означает отказ от повторного использования в пользу дублирования. Это, очевидно, может быть *неизбежным* для сложных систем при использовании языков, не поддерживающих комбинаторы и полиморфизм типов, и это в принципе может быть *исключено* в языках, обладающих свойством гомоиконичности (хотя и не обязательно эффективно), так как любой шаблон может быть реализован в виде исполнимого кода.

Используемые источники

1. MVC: https://ru.wikipedia.org/wiki/Model-View-Controller
2. Swift: https://ru.wikipedia.org/wiki/Swift_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)
3. Objective – C: https://ru.wikipedia.org/wiki/Objective-C
4. Git: https://ru.wikipedia.org/wiki/Git
5. Realm: https://en.wikipedia.org/wiki/Realm_(database)
6. XCode: https://ru.wikipedia.org/wiki/Xcode
7. Сравнение Swift и Objective – C: https://geekbrains.ru/posts/swift_vs_obj_c
8. Шаблон проектирования: https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F
9. iOS против Android: https://www.digger.ru/news/ios-protiv-android-10-preimushhestv-operacionnoj-sistemy-apple

Фрагмент программного кода

```
import UIKit
import RealmSwift

class MainViewController: UITableViewController {

    let searchController = UISearchController(searchResultsController: nil)
    var places: Results<Place>!
    private var filteredPlaces: Results<Place>!
    var ascendingSorting = true
    private var searchBarIsEmpty: Bool {
        guard let text = searchController.searchBar.text else { return false }
        return text.isEmpty
    }
    private var isFiltering: Bool {
        return searchController.isActive && !searchBarIsEmpty
    }

    @IBOutlet weak var segmentedcontrol: UISegmentedControl!
    @IBOutlet weak var reversedSortingButton: UIBarButtonItem!
    override func viewDidLoad() {
        super.viewDidLoad()

        places = realm.objects(Place.self)

        // SetUp the serach Controller
        searchController.searchResultsUpdater = self
        searchController.obscuresBackgroundDuringPresentation = false
        searchController.searchBar.placeholder = "Search"
        navigationItem.searchController = searchController
        definesPresentationContext = true
    }

    // MARK: - Table view data source


    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
{
```

```
        if isFiltering {
            return filteredPlaces.count
        }
        return places.count
    }


    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath) as! CuctomTableViewCell

        let place = isFiltering ? filteredPlaces[indexPath.row] : places[indexPath.row]

        cell.nameLable.text = place.name
        cell.locationLable.text = place.location
        cell.typeLable.text = place.type
        cell.imageOfPlace.image = UIImage(data: place.imageData!)
        cell.cosmosView.rating = place.rating

        return cell
    }


    // MARK: - Table view delegate

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        tableView.deselectRow(at: indexPath, animated: true)
    }

    override func tableView(_ tableView: UITableView, editActionsForRowAt indexPath: IndexPath) -> [UITableViewRowAction]? {

        let place = places[indexPath.row]
        let deleteAction = UITableViewRowAction(style: .default, title: "delete") {
            (_, _) in

            StorageManager.deleteObject(place)
            tableView.deleteRows(at: [indexPath], with: .automatic)
        }
```

```
    return [deleteAction]
}

// MARK: - Navigation

override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

    if segue.identifier == "showDetail" {

        guard let indexPath = tableView.indexPathForSelectedRow else { return }

        let place = isFiltering ? filteredPlaces[indexPath.row] : places[indexPath.row]
        let newPlaceVC = segue.destination as! NewPlaceViewController
        newPlaceVC.currentPlace = place
    }
}

@IBAction func unwindSegue(_ segue: UIStoryboardSegue) {

    guard let newPlaceVC = segue.source as? NewPlaceViewController else { return }

    newPlaceVC.savePlace()
    tableView.reloadData()
}

@IBAction func sortSelection(_ sender: UISegmentedControl) {
  sorting()
}

@IBAction func reversedSorting(_ sender: Any) {
    ascendingSorting.toggle()
    if ascendingSorting {
        reversedSortingButton.image = #imageLiteral(resourceName: "AZ")
    } else {
        reversedSortingButton.image = #imageLiteral(resourceName: "ZA")
    }
    sorting()
}

private func sorting() {
    if segmentedcontrol.selectedSegmentIndex == 0 {
```

```
            places = places.sorted(byKeyPath: "date", ascending: ascendingSorting)
          } else {
            places = places.sorted(byKeyPath: "name", ascending: ascendingSorting)
          }
          tableView.reloadData()
        }
    }

    extension MainViewController: UISearchResultsUpdating {

        func updateSearchResults(for searchController: UISearchController) {
            filterContentForSearchText(searchController.searchBar.text!)
        }

        private func filterContentForSearchText(_ searchText: String) {

             filteredPlaces = places.filter("name CONTAINS[c] %@ OR location CONTAINS[c] %@",
searchText, searchText)

            tableView.reloadData()
        }
    }
```

Приложение

Блок-схема сортировки записей

Диаграмма последовательности

Диаграмма состояний

Заключение

6. Заключение Применение всех достоинств языка разработки, среды проектирования и необходимых Фреймворков, предоставленных компанией Apple, позволило качественно и быстро создать необходимое приложение. В ходе выполнения данной курсовой работы была достигнута поставленная цель: разработать красивое, удобное и быстрое приложение-помощник, которое может хранить и отображать вносимые пользователем посещаемые места.

Список использованных источников

Приложения

1. [электронный документ] 5ebd2abea2559_Курсовая работа ООП.docx