ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ РФ   
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ   
**УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**им. А.М.Горького**  
Математико-механический факультет

Кафедра высокопроизводительных компьютерных технологий   
  
  
  
 МАРШРУТИЗАЦИЯ В ГОРОДСКИХ ИНФОРМАЦИОННЫХ СИСТЕМАХ НА БАЗЕ МОБИЛЬНЫХ ИНТЕРФЕЙСОВ

|  |  |  |
| --- | --- | --- |
| «Допустить к защите» Заведующий кафедрой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_2011 г. |  | Дипломная работа  студента V курса  Мальцева А.С.  Научный руководитель - кандидат физ.-мат. наук,  доцент кафедры информатики  и процессов управления  Авербух В.Л. |

Екатеринбург  
2011

**РЕФЕРАТ**

Мальцев А.С. МАРШРУТИЗАЦИЯ В ГОРОДСКИХ ИНФОРМАЦИОННЫХ СИСТЕМАХ НА БАЗЕ МОБИЛЬНЫХ ИНТЕРФЕЙСОВ, дипломная работа: стр.107, рис. 14, прил. 1

Ключевые слова: ИНФОРМАЦИОННАЯ СИСТЕМА, ЭЛЕКТРОННАЯ КАРТА, ANDROID, ИНФОРМАЦИОННАЯ ВИЗУАЛИЗАЦИЯ, GPS, МАРШРУТ, НАВИГАЦИЯ.

Объект исследования – информационные системы для мобильных устройств.

Цель работы – расширение электронной туристической карты до «электронного туристического гида» для мобильных устройств на платформе Android.

В процессе работы были проведены теоретические исследования средств информационной визуализации для мобильных приложений, изучены возможности практического применения GPS-навигации в информационных системах.

Результатом работы стала прототипная реализация приложения (электронный туристический гид) для мобильных устройств на платформе Android.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc295048549)

[1. ИНФОРМАЦИОННАЯ ВИЗУАЛИЗАЦИЯ ДЛЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ 8](#_Toc295048550)

[2. ГИС И СПУТНИКОВЫЕ ТЕХНОЛОГИИ 15](#_Toc295048551)

[2.1 GPS-навигация в мобильных устройствах 17](#_Toc295048552)

[2.2 Построение оптимальных маршрутов 25](#_Toc295048553)

[3. ОСНОВНЫЕ ТРЕБОВАНИЯ К ПРИЛОЖЕНИЮ 30](#_Toc295048554)

[4. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ 31](#_Toc295048555)

[4.1 История операционной системы Android 31](#_Toc295048556)

[4.2 Архитектура OS Android 33](#_Toc295048557)

[4.3 Среда разработки под OS Android 40](#_Toc295048558)

[5. РЕАЛИЗАЦИЯ 41](#_Toc295048559)

[5.1 Структура программы 41](#_Toc295048560)

[5.2 Программная реализация 48](#_Toc295048561)

[ЗАКЛЮЧЕНИЕ 59](#_Toc295048562)

[ЛИТЕРАТУРА 60](#_Toc295048563)

[ПРИЛОЖЕНИЕ 61](#_Toc295048564)

# ВВЕДЕНИЕ

В настоящее время мобильные устройства при всем желании нельзя отнести к разряду экзотики. Мы видим модели класса «все в одном» на прилавках салонов связи, у знакомых и коллег. Многие из нас владеют такими аппаратами лично. Традиционные мобильные телефоны «умнеют» на глазах, а карманные компьютеры, не оснащенные возможностью мобильной связи, готовятся сойти со сцены.

История массовых моделей «карманных помощников» насчитывает уже более пяти лет — учитывая современные темпы развития электроники, это весьма солидный срок. За прошедшее время смартфоны и коммуникаторы возмужали и приобрели ощутимую респектабельность. Размеры корпусов последних моделей определяются лишь задачей размещения достаточно большого экрана и удобной клавиатуры; их полезным и нужным возможностям нет числа — начиная от тривиальной способности сделать напоминание о запланированной встрече и заканчивая функцией GPS, которая поможет сориентироваться в незнакомом городе. Если же не полениться и поинтересоваться существующими опциями внимательнее, можно найти еще более удивительные инструменты. Знаете ли вы, например, что смартфон может выступать в качестве вашего агента на бирже (отслеживать заданные параметры котировок и в нужный момент совершать те или иные операции)?

В общем, наверняка через пару-тройку лет не останется ни одного десктопного приложения, которое нельзя было бы запустить с помощью смартфона или коммуникатора.

Чем же могут порадовать современного пользователя, избалованного гигагерцевыми частотами настольных ПК и широкополосным доступом в Интернет, такие внешне скромные устройства как смартфоны и коммуникаторы? Включаем смартфон. На первый взгляд, все как в обычном телефоне — записная книжка, отправка коротких сообщений, мелкие утилиты вроде будильника и конвертера валют... Возможность просмотра и отсылки электронной почты. Но многие обычные телефоны давно обладают подобной функцией. Казалось бы, современный мобильный телефон способен удовлетворить все мыслимые потребности. Однако у смартфонов есть козырь в рукаве — поддержка работы с мобильными версиями бизнес-приложений, которую они имеют практически с самого начала пути своего развития. Эта функция востребована в большинстве крупных компаний; благодаря возможности прямой синхронизации с десктопной версией отпадает необходимость производить дополнительный перенос информации из одной программы в другую, что существенно упрощает ведение электронного документооборота.

Теснейшим образом связан с электронной почтой и мобильный Интернет. Если раньше на портативных устройствах можно было просматривать только WAP-ресурсы, то сегодня производители обеспечили пользователей смартфонов функцией просмотра полноценных веб-страниц.

Для удобства работы вне офиса существует возможность редактирования документов Word и Exсel на смартфоне или коммуникаторе, причем вне зависимости от типа ОС.

Итак, чем же современные смартфоны и коммуникаторы отличаются от своих предтеч? Что мы сможем увидеть лет через пять?

Общий курс можно обозначить одним словом — развитие. Это и усложнение, и совершенствование, и увеличение числа всевозможных функций.

Важной тенденцией в эволюции мобильных устройств является то, что их интерфейс становится все более привычным для пользователей. В области управления несомненным прогрессом является появление смартфонов с сенсорными экранами.

Что ждет нас впереди? Казалось бы, все функции изобретены, все, что только можно усовершенствовать, усовершенствовано. Однако производители не собираются останавливаться — ведь чтобы подогревать потребительский спрос на продукцию, необходимо выпускать новинки.

Хотелось бы сказать несколько слов о тех, на кого ориентируются производители коммуникаторов и смартфонов. Кому может понадобиться объединение огромного количества функций мобильных гаджетов в одном корпусе?

Несомненно, наибольшую выгоду от использования коммуникаторов получают бизнесмены; они же и составляют целевую аудиторию этого класса портативных интеллектуальных устройств. Кстати говоря, следствием такого серьезного позиционирования коммуникаторов является их высокая цена.

Смартфоны — вещь гораздо более дешевая и массовая, а потому пользуются ими сейчас практически все, кого интересуют мобильные технологии. Так, их зачастую приобретают либо ради моды и престижа, либо для чтения книг в транспорте. Дело в том, что покупка КПК даже начального уровня в довесок к уже имеющемуся телефону — это достаточно ощутимый удар по кошельку. А вот если вы собрались менять телефон, то приобретение смартфона представляется вполне разумным решением. Такой аппарат послужит прекрасным дополнением к настольному компьютеру.

Ниже в дипломной работе проводится исследование в области маршрутизация в городских информационных системах на базе мобильных интерфейсов. Будет проведено изучение возможностей практического применения GPS-навигации в информационной визуализации с использованием мобильных сервисов. При этом особое внимание будет уделено особенностям построения оптимального пути следования объекта в городском пространстве. Результатом дипломной работы станет реализация прототипа приложения, расширяющего электронную туристическую карту до «электронного туристического гида» для мобильных устройств на платформе Android. В заключении рассматриваются перспективы данного исследования.

# ИНФОРМАЦИОННАЯ ВИЗУАЛИЗАЦИЯ ДЛЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Определим термин «информационная визуализация», который относится к визуальному описанию и представлению абстрактной информации, получаемой в результате процесса сбора и обработки данных различного типа и назначения. Это процесс формирования ментальной модели данных, посредством чего обеспечивается представление об их внутренней структуре. Задача информационной визуализации − представление и выявление (возможно скрытых) взаимоотношений, структур и отдельных характеристик изучаемых данных.

Если научная визуализация служит для представления данных, сгенерированных по ходу научных вычислений, то информационная визуализация связана с обработкой разнотипной информации, используемой для анализа в различных областях человеческой деятельности, часто не имеющих отношения к науке и технике.

Особое значение информационная визуализация приобрела в связи с необходимостью представления огромных информационных ресурсов, появившихся во «всемирной паутине» интернета. Если использовать термины теории информации, то можно сказать, что, в значительной мере, WWW содержит не информацию, а шум. Поэтому задачей информационной визуализации является выявление полезных знаний из залежей сырых данных. Ее можно рассматривать как составляющую такой дисциплины как data mining (она же knowledge discovery). Абстрактный характер визуализируемой информации определяет необходимость поиска и разработки сложных методов представления.

На базе исследований и опыта использования (в том числе и на докомпьютерной стадии развития) визуальных систем можно описать принципы построения эффективных видов отображения.

Результаты исследований и разработок в области информационной визуализации активно используются не только в первоначально предназначенных для этого сферах (например, бизнесе, медицине или управлении), но и играют важную роль при разработке методов представления для систем визуализации программного обеспечения, в частности для визуализации данных в системах отладки и настройки производительности параллельных вычислений.

Как уже упоминалось выше, информационную визуализацию можно рассматривать как составляющую такой дисциплины как data mining.

Под технологией Data Mining (она же Knowledge Discovery in Database) понимается набор методов, поддерживающих процесс обнаружения и извлечения ранее неизвестных, практически полезных, поддающихся интерпретации знаний из сырых данных, хранящихся в базах данных. Эти знания могут затем использоваться для принятия решений в различных сферах человеческой деятельности.

В основу современной технологии Data Mining положена концепция шаблонов (patterns), отражающих фрагменты многоаспектных взаимоотношений в данных. Эти шаблоны представляют собой закономерности, свойственные подвыборкам данных, которые могут быть компактно выражены в понятной форме. При этом поиск шаблонов не ограничивается рамками априорных предположений о структуре выборки и виде распределений значений анализируемых показателей.

Более точное определение этой дисциплины выглядит следующим образом:

Data Mining − процесс обнаружения значимых корреляций, зависимостей и тенденций в результате анализа хранимой информации с применением методов анализа и выявлений отношений различного типа (ассоциаций, последовательностей аналогий, кластеров, etc.) между данными. Применяются различные (в том числе и связанные с искусственным интеллектом) методы выделения и извлечения информации, которые позволяют выявить систематизированные структуры данных (patterns − шаблоны) и вывести из них правила для принятия решений и прогнозирования их последствий.

В Data Mining используется следующие основные инструменты:

- нейронные сети;

- деревья решений;

- индуктивные правила;

- визуализация данных.

Методы Data Mining позволяют выявить следующие типы закономерности в данных:

- ассоциация, когда выявляются события, имеющие между собой некоторые семантические зависимости;

- последовательность, когда выявляется цепочка связанных во времени событий;

- классификация, когда выделяются признаки, характеризующие группу, которой принадлежит тот или иной объект;

- кластеризация, которая позволяет самостоятельно выделять различные однородные и заранее не заданные группы данных;

- прогнозирование, основой для которого служат временные ряды, хранящиеся в базах данных. Шаблоны, адекватно отражающие динамику поведения каких-либо показателей, могут помочь в предсказании поведения системы в будущем.

Визуализация в рамках Data Mining (как и любая информационная визуализация) понимается в нескольких смыслах.

Во-первых, имеет место визуализация, поддерживающая подготовку данных для Data Mining.

Во-вторых, визуализация используется для представления «сырых» и промежуточных данных и результатов проведенного анализа.

В-третьих, визуализация и автоматические методики Data Mining объединяются в рамках одного инструментария.

Внимание специалистов привлекает описание видов отображения информационной визуализации и их использования в практике представления типов данных и конкретных данных. При этом, рассматриваются динамические виды отображения, включающие образы вместе с их возможной динамикой и возможным взаимодействием пользователей с создаваемой картинкой, например, с коническими деревьями или с информационной стеной. К этой проблематике можно в принципе отнести и взаимодействие с информацией посредством WWW. Специальный интерес вызывает визуализация многомерных информационных пространств и связанная с ней проблематика извлечения знаний. Отдельной темой является визуализация медицинской и биологической информации, также как и визуализация документов. Рассказывается о разработке и использовании визуализации в таких областях как электронная коммерция, географическая визуализация, информационная поддержка на поле боя. Естественно, большой внимание уделяется разработке конкретных систем, а также вопросам оценки качества визуализации и удобству пользователей систем. Идет постоянный поиск новых метафор и видов отображения, пригодных для информационной визуализации. Достаточно высокий уровень развития информационной визуализации подтверждается тем, что в литературе появляются теоретические работы по основаниям дисциплины, по визуализационной эстетике и проблемам восприятия визуальных образов.

По сравнению с другими областями компьютерной визуализации в литературе по информационной визуализации уделяется большее внимание видам отображения, как таковым, их классификации и методикам применения для различных типов организации данных. В теоретических работах по информационной визуализации рассказывается о принципах разработки эффективных видов отображения и проводится (правда, первичный) анализ того, как достигается эффективность.

Рассмотрим общие правила компоновки графических средств представления информации:

1. Принцип лаконичности

Графическое средство представления информации должно содержать лишь те элементы, которые необходимы для сообщения наблюдателю существенной информации, точного понимания её значения или принятия соответствующего оптимального (в каком-то смысле) решения. Бесполезно стремиться направить внимание на важнейшие характеристики, если они окружены лишними, не относящимися к ним визуальными раздражителями, мешающими восприятию главного.

2. Принцип обобщения и унификации

Основные формы графического средства представления информации не следует излишне дробить, включая в них элементы, обозначающие несущественные с точки зрения отображаемой информации детали изображаемых объектов; их форма должна быть рационально обобщена. Кроме этого, в пределах всего комплекса графических средств представления информации образы, использованные для обозначения одних и тех же объектов, должны быть обязательно унифицированы - иметь единое графическое решение.

3. Принцип акцента на основных смысловых элементах

На графических средствах отображения информации следует выделять размерами, формой, цветом в первую очередь те элементы, которые наиболее существенны с точки зрения восприятия наблюдателем передаваемой информации. В отдельных случаях допустимо даже нарушение пропорций между размерами образов и соответствующих им реальных объектов.

4. Принцип автономности

Части графического средства представления информации, передающие относительно автономные сообщения следует обособить и чётко отграничить от других частей. Разбиение сложной графической информации на отдельные простые изображения значительно облегчает её восприятие и понимание.

5. Принцип структурности

Каждая автономная часть комплекса графических средств отображения информации, занимающая в общем изложении некоторое центральное положение, должна иметь чёткую, легко запоминающуюся и дифференцирующуюся от других структуру, отражающую характер каждого сообщения.

6. Принцип стадийности

В зависимости от стадий изложения информации должен выбираться состав сообщений, отображаемых на отдельных графических средствах. Этот принцип основан на методах борьбы с вредной, лишней в конкретном случае информацией путём пространственного и временного разделения всей информации и её последовательного восприятия.

7. Принцип использования привычных ассоциаций и стереотипов

При создании графических средств представления информации должны учитываться устойчивые, привычные ассоциации между образами и обозначаемыми ими объектами и явлениями. Желательно применять не абстрактные условные знаки, а образы, привычно ассоциирующиеся с соответствующими объектами и явлениями. Однако следует учитывать, что слишком натуралистичное изображение внешнего вида объектов фиксирует мысль наблюдателя именно на внешнем сходстве с объектом и мешает осознанию более существенных с точки зрения представляемой информации признаков данного объекта.

При разработке видов отображения необходимо фокусироваться на содержании данных, а не на методиках визуализации. Необходимо показывать данные, причем пользователь должен обращать внимание на суть визуализируемых данных, а не на особенности графических программ, алгоритмы визуализации и т.п. Успех визуализации основан на глубоком знании и обеспечении сути данных, их качественной природы и целостности их содержания.

# ГИС И СПУТНИКОВЫЕ ТЕХНОЛОГИИ

Материал для данного раздела взят из [1], [2], [3], [4].

C развитием высоких технологий в нашем быту происходят стремительные перемены. Кто десять-пятнадцать лет назад мог предположить, что мобильная связь станет общедоступной?

А как вам понравится вместо атласа автомобильных дорог электронный планшет, в котором не только имеются карты или планы местности в любом масштабе, но и указываются ваши местоположение и направление движения? С его помощью можно определить, как лучше добраться до интересующего «адреса» (детализация - вплоть до поэтажного плана здания), получить информацию о проводимых дорожных работах и т. п.

Например, для английских автомобилистов такие электронные атласы стали уже привычными. Придумываются сотни новых применений для навигационных приемников. Куда их только не пытаются встроить! Практически все известные автомобилестроители комплектуют ими свои машины.

Сегодня работают две спутниковые системы - американская Navstar (Navigation System using Timing And Ranging), больше известная нам как GPS (Global Positioning System), и отечественная «ГЛОНАСС». Принципы их работы во многом схожи. Главная задача спутников, входящих в состав этих систем, заключается в постоянной передаче сигналов, которые принимают наземные (авиационные, корабельные, автомобильные, ручные и т. д.) приемники. Системы построены так, что приемник одновременно «видит» несколько спутников. Сравнивая задержки в приходе сигналов от разных спутников, приемник вычисляет расстояние от себя до них, а затем решает систему уравнений, чтобы определить свои координаты.

Возникает вопрос: «Ну и что с этими координатами я буду делать? Удовлетворять свое любопытство? Неужели я без спутников дорогу домой не найду?»

Никто и не утверждает, что навигационные приемники нужны всем и всегда. Спутниковая навигация требуется тем областям, в которых без знания координат не обойтись, - авиации, морскому и автомобильному транспорту, диспетчерским и геоинформационным системам, проектированию и строительству, службам спасения, туризму, охоте, рыбалке и т. д.

Имея навигационный приемник, можно с точностью до нескольких метров определять свои координаты, скорость перемещения и направление движения. В сочетании с электронной картой местности (ГИС) и средствами мобильной связи получается мощнейший инструмент для построения диспетчерских систем. Первыми из гражданских пользователей возможности спутниковой навигации оценили банкиры: были созданы системы слежения за перевозкой денежных ценностей. Диспетчер банка видит на мониторе всю картину - на какой улице и возле какого дома находится автомобиль инкассации, куда движется, есть ли проблемы у экипажа и т.д. Аналогичные системы разворачиваются для «экстренных» служб, туристических операторов экстремального туризма за рубежом.

На сегодняшний день геоинформационные технологии постепенно завоевывают Российский рынок. Создание земельного кадастра позволит на основе его карт строить другие, предметно ориентированные карты и дополнять их соответствующим атрибутивным наполнением. Для создания туристско-ориентированных ГИС потребуется объединение усилий всех заинтересованных сторон, это необходимо для создания информационного контента баз данных, постоянного поддержания его актуальности и соответствия действительности. Также необходима финансовая и законодательная поддержка со стороны государства, ввиду высокой дороговизны ГИС проектов.

Таким образом, мы являемся свидетелями рождения и развития ГИС разных масштабов: мегаполисов – Москва, С-Петербург, областных, городских.

Применение технологии ГИС является огромным подспорьем в деле организации и проведения туров, а также сопутствующего сервиса.

## 2.1 GPS-навигация в мобильных устройствах

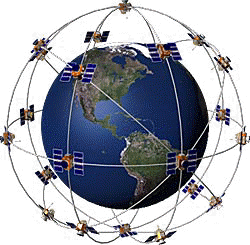
Конвергенция мобильных телефонов и навигационных решений стала одной из самых мощных тенденций последнего времени. Время специализированных GPS-устройств, предназначенных для массового рынка, уходит. Современные навигационные чипы компактны и недороги, что позволяет оснащать функцией глобального позиционирования даже мобильные телефоны. Ведущие компании-производители тратят огромные средства на реализацию навигационной составляющей в максимально большом количестве своих продуктов.

История развития спутниковой навигации

Идея использовать космические аппараты для навигации начала развиваться в Америке после того, как СССР в 1957 году запустил первый искусственный спутник Земли. США поставили задачу слежения за советскими спутниками посредством приема их сигналов на наземном пункте с известными координатами, определения доплеровского сдвига несущей частоты и дальнейшего расчета параметров их движения. Следом вполне естественно возникла и возможность решения обратной задачи: расчета координат наземного приемника на основе обработки принятого сигнала от спутника с известной орбитой и скоростью движения.

Затем военные стали развивать идею: для пуска баллистических ракет с подводных лодок в 1964 году была создана доплеровская спутниковая радионавигационная система первого поколения Transit. Впрочем, она уже в 1967 году была рассекречена и приспособлена для коммерческого использования, причем число гражданских потребителей вскоре существенно превысило количество военных.

После того, как в 1960 году были разработаны атомные часы, стало возможным использовать для навигации сеть точно синхронизированных передатчиков, где координаты рассчитывались по временным задержкам.

В 1973 году все программы ВВС и ВМФ Соединенных Штатов были объединены в общую Навигационную технологическую программу, позднее превратившуюся в Navstar-GPS. Первая штатная орбитальная группировка современной системы разворачивалась с июня 1989 по март 1994 года. При этом на орбиту были выведены 24 навигационных спутника Block II, находящихся на 6 различных круговых орбитах, которые расположены под углом 60 град друг к другу. Период обращения одного спутника - 12 часов. Вес каждого спутника около 787 кг, размер более 5 м, включая солнечные батареи. На борту каждого спутника установлены атомные часы, обеспечивающие точность 10-9 сек, вычислительно кодирующее устройство и передатчик мощностью 50 Вт, излучающий на частоте 1575.42 МГц. Полностью GPS-система была запущена в эксплуатацию в 1995 году. В настоящее время она обслуживается Министерством обороны США. В самом конце прошлого века к системе было добавлено еще четыре спутника следующего поколения.

Механизм работы системы GPS (сокращение от NAVSTAR GPS - англ. Navigation Satellits providing Time And Range Global Positioning System, то есть глобальная система позиционирования) заключается в определении местоположения устройства при помощи измерения расстояний до него от точек с известными и постоянными координатами, то есть спутников. Система разрабатывалась под эгидой Министерства обороны США, да и сегодня имеет не только гражданское предназначение (более того, вплоть до 2000 года "благодаря" нуждам Пентагона нарочно занижалась точность гражданских GPS-приемников, а в том или ином виде контроль за функционированием GPS со стороны военного ведомства сохраняется до сих пор).

Решение о разработке GPS было принято в 1973 году, когда США и СССР противостояли друг другу не только на земле, воде и в воздухе, но и в космосе. Сегодня данная навигационная система используется по всему миру и занимает лидирующие позиции, несмотря на наличие ряда локальных проектов спутниковой навигации в отдельных странах (в частности, азиатских).

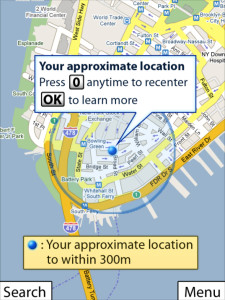
Существуют также и другие навигационные системы, претендующие на звание глобальных. Одна из таких систем - отечественная разработка ГЛОНАСС (Глобальная навигационная спутниковая система), идея которой появилась в те же времена, что и конкурирующей GPS, и по тем же самым причинам. К сожалению, функциональное состояние системы оставляет желать лучшего, поскольку до сих пор не выведено на орбиту необходимое для покрытия территории всей Земли количество спутников (по причине устаревания уже действующих спутников это количество должно быть увеличено еще на несколько единиц).

Наконец, детище объединенной Европы - проект "Галилео" (европейская система позиционирования) пока также можно считать развивающимся, реальных результатов определения местоположения при помощи "Галилео" нет (за неимением сети спутников, оборудования и проч.).

Спутниковая навигация в мобильных устройствах.

На рынке мобильных решений наблюдается та же картина. Доминирует система спутниковой навигации GPS, устройств с ее поддержкой сегодня достаточное количество, в то время как ГЛОНАСС-навигаторы пока в диковинку (да и первенец, GloSpace, является комбинированным GPS/ГЛОНАСС-решением).

Однако нахождение местоположения при помощи мобильных устройств не всегда определяется наличием либо отсутствием GPS-приемника в телефоне или смартфоне. В последнее время стали весьма популярными различные сервисы, позволяющие с удовлетворительной точностью (десятки-сотни метров) находить свои координаты - принцип работы основан на получении информации от базовых станций (БС) операторов сотовой связи (здесь БС играют роль спутников, поскольку их координаты известны). Причем такая возможность известна уже довольно давно, операторы предлагали подобную функциональность и раньше. Однако с выходом Google Maps 2.0 операторы "встрепенулись" и сегодня наперебой предлагают абонентам услуги по определению местоположения (причем не только своего - можно находить координаты друзей, ребенка и т.д.). Продолжают развиваться и собственно картографические онлайн-сервиcы, вроде того же Google Maps, "Яндекс.Карты" и проч.



Специализированное навигационное ПО для смартфонов и коммуникаторов также не стоит на месте. В большинстве случаев навигационный софт имеет версии как для специализированных навигаторов, так и для мобильных устройств - те же "Навител Навигатор", iGO и т.д. Это обстоятельство несколько уравнивает возможности различных типов устройств, хотя, несомненно, специализированные устройства-навигаторы выглядят куда предпочтительнее - например, при использовании в автомобиле 2,8-дюймовая (а то и меньше) диагональ современных коммуникаторов менее удобна, чем порядка 4,3 дюйма у навигаторов.

Сами навигаторы также представляют собой весьма разношерстную компанию - здесь и профессиональные автономные навигаторы (идеальный вариант для охотников), и автомобильные навигаторы, и даже внешние GPS-приемники, соединяющиеся с другими устройствами посредством Bluetooth.

В то же время, поддержку спутниковой навигации первыми из мобильных устройств получили КПК. Это стало одной из главных вех в развитии спутниковой навигации. Важно понимать, что вначале спутниковая навигация в КПК воспринималась лишь как одна из дополнительных возможностей. Да и появились подобные решения в то время, когда конкуренция на рынке КПК вынуждала компании-производителей предпринимать какие-то новые шаги (например, внедрение VGA-экранов, беспроводных интерфейсов Bluetooth и Wi-Fi и проч.).

Для платформы Palm OS первым навигационным КПК стала модель Garmin iQue 3600. А для более привычной нам сегодня платформы Windows Mobile первенство принадлежит модели хорошо известного на рынке навигаторов игрока - MiTAC Mio 168, КПК образца 2004 года.

В то же время, выход в сегмент навигационных КПК крупных производителей состоялся более года спустя - лишь в 2005 году появились КПК со встроенным GPS: ASUS MyPal A636, Fujitsu Siemens Pocket LOOX N520 и проч.

К этому времени направление коммуникаторов стало уже теснить КПК, в результате чего вполне предсказуемо на рынке появились первые навигационные коммуникаторы.

Следующим важным этапом в развитии мобильной спутниковой навигации стало появление и, начиная с 2005 года, массовое использование нового чипсета для GPS-устройств - SiRF Star III. Основными преимуществами перед старыми решениями, такими как SiRF Star II/LP, стали улучшенная чувствительность, работа одновременно до 20 каналов, появившаяся возможность работы в помещении, обновление позиции по одному спутнику, а также пониженное энергопотребление.

За короткий период времени количество навигационных решений, оснащенных SiRF Star III, увеличилось на порядок, на долгое время новый чип стал гарантом качества навигационного "железа".

А что же мобильные телефоны? В 2007 году стали появляться устройства на основе операционной системы Symbian OS. В то же время, это были первые смартфоны со встроенным GPS-приемником. Компания Nokia, менявшая в то время свою стратегию, сделала ставку на навигационные устройства. Всего за год появилось сразу несколько сильных моделей, специально для них разрабатывался и доводился до ума навигационный сервис smart2go (позднее переименованный в Nokia Maps).

При этом, начиная с первых смартфонов с GPS-функциональностью, продукты Nokia неизменно получали поддержку A-GPS, и это стало еще одним знаковым событием в сегменте мобильных устройств со спутниковой навигацией.

Что интересно, в моделях Nokia был использован вовсе не SiRF Star III, а решение от Texas Instruments - TI GPS5300 NaviLink. Это стало одним из первых предвестников дальнейшей интеграции GPS-чипов в аппаратные платформы. В дальнейшем навигационные решения будут включаться в "шасси" устройств изначально, равно как и беспроводные интерфейсы.

Наконец, последним действительно знаковым событием в сегменте "мобильной" навигации стал выход второй редакции онлайн-сервиса Google Maps. Вторые "Карты" от Google предоставили возможность не только отображать карту местности, но и определять свои координаты по базовым станциям сотовых операторов. Точность при этом, разумеется, существенно ниже, чем у специализированных устройств, однако для приблизительного позиционирования на местности подобные приложения подходят идеально. К тому же, в отличие от большинства GPS-навигаторов, координаты определяются, даже если вы находитесь в помещении.

Еще один весомый аргумент в пользу важности таких сервисов - интеграция приложения в прошивку ряда мобильных телефонов, когда новое решение уже изначально предоставляет пользователю подобную функциональность. Все это ведет к еще большему распространению навигационных систем, изменяя тем самым рынок мобильных устройств.

Вслед за Google Maps подтянулись и другие решения, кроме того, стали проявлять активность и операторы сотовой связи. Российские операторы также предлагают сегодня подобные услуги, либо самостоятельно, либо совместно с навигационными сервисами - в частности, проектом "Яндекс.Карты" (МТС). Также можно вспомнить кооперацию "Билайн" и компании Global Searching Systems (проект "Мобильная навигация" NaviFon).

Более того, некоторые операторы предлагают определить не только ваше местонахождение, но и ваших знакомых и родственников (услуги "Ребенок под присмотром" и "Локатор" от того же МТС) - естественно, с их ведома и согласия.

Все это на самом деле представляет большой интерес, поскольку расширяет возможности традиционных GPS-систем, делает их более интересными для конечных пользователей. В то же время, рынок программного обеспечения и навигационных сервисов сегодня настолько обширен, что требует отдельного рассмотрения - поскольку, если количество баз картографических данных ограничено (тот же TeleAtlas, другие проекты), то программ на их основе - бесчисленное множество. Только в России актуальными являются более десятка сервисов - в числе лидеров такие проекты, как "Навител Навигатор", City Guide и другие.

Таким образом, на сегодняшний день средства навигации охватывают широчайший спектр мобильных устройств, от классических автономных и автомобильных навигаторов до обычных мобильных телефонов.

Все больше смартфонов и коммуникаторов среднего ценового диапазона постепенно обзаводятся встроенными GPS-приемниками, не в последнюю очередь, благодаря интеграции навигационных чипов в аппаратные платформы устройств.

Кстати, если до 2007 года львиную долю GPS-решений занимали Windows Mobile-устройства, то с массовым выходом на данный рынок компаний Nokia и Samsung доля навигационных Symbian-смартфонов резко увеличилась.

Данное обстоятельство, кстати, несколько расширило возможности средств спутниковой навигации в мобильных устройствах. Наряду со ставшим уже привычным использованием GPS-навигаторов и коммуникаторов со встроенным GPS в автомобилях, появилась и набирает популярность так называемая пешеходная навигация. Точнее, она стала более массовой - не стоит забывать об узкой нише профессиональных автономных навигаторов для путешественников, спортсменов и т.д.

Еще более массовой делает навигацию в мобильных устройствах постепенное проникновение в данный сегмент обычных мобильных телефонов. Они могут быть даже лишены встроенных GPS-приемников - местоположение определяется по координатам базовых станций операторов сотовой связи.

Настоящей революцией в данном направлении стал выход навигационного сервиса Google Maps - теперь любой телефон может с достаточно высокой точностью показывать местоположение владельца (при движении карты в режиме реального времени подгружаются из сети). Однако именно выход Google Maps 2.0 послужил тем толчком, который побудил других разработчиков к выпуску собственных решений. В частности, на отечественном мобильном рынке известность получил сервис "Яндекс.Карты".

Если говорить о дальнейшем развитии программного обеспечения для навигационных решений, то здесь, прежде всего, необходимо отметить появление новых возможностей, сервисов, повышающих удобство работы с GPS-устройствами. Это и увеличение количества информационных сервисов, и насыщение баз данных "точек по интересам" POI, а также интеграция навигационных и мультимедийных сервисов - в частности, функция геотеггинга (географических "меток" фотографий), когда координаты места снимка заносятся в EXIF-информацию файла.

В результате, навигационное направление в развитии мобильных устройств сегодня действительно представляется одним из самых перспективных. Рынок пока далек от насыщения, в ближайшем будущем нас наверняка ожидает появление новых игроков, с еще более интересными устройствами.

## 2.2 Построение оптимальных маршрутов

Алгоритмы нахождения оптимальных путей описаны по работам [5], [6].

Задача нахождения оптимальных путей по уровню транспортных затрат актуальна для ряда технических приложений. В частности, к ним относятся: оценка транспортной доступности для территориально-распределенных систем охраны, планирование оптимальных маршрутов движения робототехнических систем на пересеченной местности, моделирование прокладки маршрутов в тренажерах мобильных систем и компьютерные играх.

Задача планирования оптимального пути в общей постановке формулируется следующим образом. На карте местности необходимо определить маршрут движения от стартового множества точек к множеству конечных точек обладающий минимальными транспортными затратами. В такой постановке начальные и конечные точки заведомо не известны и определяются в процедуре расчета. В частной постановке возможны следующие варианты задач:

* проложить оптимальный маршрут от множества стартовых точек к заданной конечной точке;
* проложить оптимальный маршрут от множества стартовых точек к заданному множеству конечных точек;
* проложить оптимальный маршрут от заданной стартовой точки к заданной конечной точке;
* построить фронт транспортной доступности с заданным уровнем затрат по отношению к множеству стартовых точек.

Известный базовый алгоритм прокладки оптимальных маршрутов основан на методе динамического планирования Форда-Беллмана на взвешенных графах (1956-1958 гг). Для транспортной сети вершинами графа является узлы транспортной сети, а дуги соответствуют переходам между узлами. Множество алгоритмов предложенных в последующие годы (алгоритмы Дейкстры, Калаба, A-звезда и др.) в основном являются вариациями базового алгоритма для частных постановок, благодаря чему достигается более высокая вычислительная эффективность данных алгоритмов по сравнению с базовым алгоритмом.

Алгоритм Дейкстры

Алгори́тм Де́йкстры (Dijkstra’s algorithm) — алгоритм на графах, изобретённый нидерландским ученым Э. Дейкстрой в 1959 году. Находит кратчайшее расстояние от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм широко применяется в программировании и технологиях, например, его использует протокол OSPF для устранения кольцевых маршрутов.

**Формальное определение**

Дан взвешенный ориентированный граф *G(V,E)* без петель и дуг отрицательного веса. Найти кратчайшие пути от некоторой вершины *a* графа *G* до всех остальных вершин этого графа.

**Неформальное объяснение**

Каждой вершине из *V* сопоставим метку — минимальное известное расстояние от этой вершины до *a*. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация. Метка самой вершины *a* полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от *a* до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина *u*, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых *u* является предпоследним пунктом. Вершины, в которые ведут рёбра из *u*, назовем соседями этой вершины. Для каждого соседа вершины *u*, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки *u* и длины ребра, соединяющего *u* с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину *u* как посещенную и повторим шаг алгоритма.

**Алгоритм**

Обозначения:

* *V* — множество вершин графа
* *E* — множество ребер графа
* *w[ij]* — вес (длина) ребра *ij*
* *a* — вершина, расстояния от которой ищутся
* *U* — множество посещенных вершин
* *d[u]* — по окончании работы алгоритма равно длине кратчайшего пути из a до вершины *u*
* *p[u]* — по окончании работы алгоритма содержит кратчайший путь из *a* в *u*

**Псевдокод**

Присвоим d[a] \gets 0,\ p[a] \gets
 a

Для всех u \in Vотличных от *a*

присвоим d[u] \gets \infty

Пока \exists v \notin U

Пусть v \notin U— вершина с минимальным *d*[*v*]

Для всех u \notin Uтаких, что vu \in E

если *d*[*u*] > *d*[*v*] + *w*[*vu*] то

изменим d[u] \gets d[v] + w [vu]

изменим p[u] \gets p[v], u

**Описание**

В простейшей реализации для хранения чисел *d[i]* можно использовать массив чисел, а для хранения принадлежности элемента множеству *U* — массив булевых переменных.

В начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом (бо́льшим максимального возможного пути в графе). Массив флагов заполняется нулями. Затем запускается основной цикл.

На каждом шаге цикла мы ищем вершину с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины. Если в ней расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его. Цикл завершается когда флаги всех вершин становятся равны 1, либо когда у всех вершин c флагом 0 d[i] = \infty. Последний случай возможен тогда и только тогда, когда граф *G* не связан.

**Сложность алгоритма**

Сложность алгоритма Дейкстры зависит от способа нахождения вершины *v*, а также способа хранения множества непосещенных вершин и способа обновления меток. Обозначим через *n* количество вершин, а через *m* — количество ребер в графе *G*.

* В простейшем случае, когда для поиска вершины с минимальным *d*[*v*] просматривается все множество вершин, а для хранения величин *d* — массив, время работы алгоритма есть *O*(*n*2 + *m*). Основной цикл выполняется порядка *n* раз, в каждом из них на нахождение минимума тратится порядка *n* операций, плюс количество релаксаций (смен меток), которое не превосходит количества ребер в исходном графе.
* Для разреженных графов (то есть таких, для которых *m* много меньше *n²*) непосещенные вершины можно хранить в двоичной куче, а в качестве ключа использовать значения *d*[*i*], тогда время извлечения вершины из \overline Uстанет *logn*, при том, что время модификации *d*[*i*] возрастет до log*n*. Так как цикл выполняется порядка *n* раз, а количество релаксаций не больше *m*, скорость работы такой реализации *O(nlogn + mlogn)*
* Если для хранения непосещенных вершин использовать фиббоначиеву кучу, для которой удаление происходит в среднем за *O(logn),* а уменьшение значения в среднем за *O(1),* то время работы алгоритма составит *O(nlogn + m)*. Однако, скрытые константы в асимптотических оценках трудоемкости велики и использование фибоначчиевых куч редко оказывается целесообразным: обычные двоичные (*d*-ичные) кучи на практике эффективнее.

Альтернативами им служат толстые кучи, тонкие кучи и кучи Бродала, обладающие теми же асимптотическими оценками, но меньшими константами.

# ОСНОВНЫЕ ТРЕБОВАНИЯ К ПРИЛОЖЕНИЮ

Целью работы является расширение туристической карты до электронного туристического гида для мобильных устройств, работающих на платформе Android.

Данное приложение должно быть «электронным помощником» туриста, используя которое он смог бы без труда познакомиться с достопримечательностями города.

Туристический гид должен обладать простым и понятным интерфейсом, в то же время обладать широкими функциональными возможностями.

Основываясь на исследованиях, приведенных в предыдущих главах, я пришел к выводу, что приложение должно иметь следующие особенности:

* Мобильность — можно воспользоваться приложением, где бы вы ни находились и в любое время.
* Полноценная, постоянно обновляемая карта города.
* Возможность определить свое местоположение.
* Визуализация перемещения пользователя с мобильным устройством по городу
* Построение маршрута от текущего местоположения до определенного объекта
* Возможность составления пользователем собственных туристических маршрутов

# ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ

Во время проведения исследований было решено разрабатывать наше приложение для мобильных устройств работающих под управлением платформы Android, т.к. данная платформа по мнению аналитиков и экспертов захватывает рынок мобильных устройств постепенно вытесняя с него общепризнанных лидеров.

Архитектура данной платформы и её основные особенности описано по работам [7], [8], [9].

## 4.1 История операционной системы Android

Android – платформа для мобильных телефонов, смартфонов, коммуникаторов на базе ОС Linux, которая поддерживается альянсом Open Handset Alliance (OHA), который инициировала компания Google.

Компания Google купила Android Inc. в июле 2005 года, которая представляла собой небольшую стартап-компанию, базировавшуюся в Пало-Альто, Калифорния. Среди основателей Android'а были Энди Рабин (Andy Rubin, основатель Danger), Рич Майнер (Rich Miner, основатель Wildfire), Ник Сирс (Nick Sears, бывший вице-президент компании T-Mobile), и Крис Уайт (Chris White, один из первых инженеров в WebTV). Все они пошли работать в Google. В то время мало что было известно про Android Inc., кроме того что они занимаются разработкой программного обеспечения для мобильных телефонов. Такое развитие событий привело к появлению слухов, что Google планирует войти на рынок мобильных телефонов, но не было ясно что именно он планирует там делать.

В Google, группа, во главе с Рубином, разработала операционную систему на основе Linux (ядро v2.6), которую они предлагают разработчикам телефонов и операторам мобильной связи.

5 ноября 2007 года, консорциум Open Handset Alliance заявил про намерение разработать открытые стандарты для мобильных устройств. В период формирования Open Handset Alliance, было также заявлено о их первом продукте, платформы для мобильных телефонов (смартфонов) на основе ОС Linux - Android.

Платформа Android легко приспосабливается для использования на VGA, 2D графических библиотек разработанных на основе спецификации OpenGL ES 1.0, а также традиционных инструментов для смартфонов. Android использует базу данных для структурированных данных. Android поддерживает большое количество технологий, которые обеспечивают связь, включая: GSM, Bluetooth, EDGE, 3G и WiFi. Для обмена сообщениями доступны как SMS, так и MMS сервисы, включая и потоковые сообщения. На Android доступен браузер разработанный на основе WebKit application framework, а также есть возможность запускать программы на Java. Android поддерживает следующие форматы аудио/видео данных и изображений: MPEG-4, H.264, MP3, и AAC, AMR, JPG, PNG, GIF. Из нестандартного оборудования присутствует поддержка видеокамеры, фотоаппарата, тачскрина, GPS, компаса, акселерометра, ускорителя 3D графики.

21 октября 2008 года альянс OHA опубликовал исходный код платформы Android. В релиз вошел весь стек Android: и операционная система, и промежуточное ПО (middleware), и основные конечные приложения, написанные на Java.

Первым смартфоном на платформе Android стало устройство T-Mobile G1 (HTC G1), презентация которого состоялась 23 сентября 2008 года. Вскоре последовали многочисленные анонсы других производителей смартфонов о намерении выпустить устройства с Android.

Основные черты Android

* Для выполнения приложений используется виртуальная машина Dalvik, оптимизированная для мобильных телефонов. Файлы типа Java Class преобразуются обычным Java-компилятором в классы другого формата (.dex) для запуска в Dalvik. По лицензионным соображениям в Dalvik вместо OpenJDK используется Apache Harmony.
* Реализована поддержка 2D/3D-графики, изображений, аудио и видео.
* Для хранения данных используется популярная легковесная СУБД SQLite.
* Доступна поддержка GSM, EDGE, 3G, Bluetooth, Wi-Fi, фото- и видеокамеры, GPS, компаса, акселерометра.
* В меню контакта присутствует не только информация, но и история переписки SMS, E-Mail, звонков и т.д.
* Возможность массовой рассылки SMS или E-Mail.
* На рабочий стол в качестве ярлыка можно вынести множество действий - звонок конкретному абоненту, смс конкретному абоненту, проигрывание плей-листа
* Наличае Android-сервисов Google.
* При каждом нажатии клавиши на экранной клавиатуре аппарат слегка вибрирует.

## 4.2 Архитектура OS Android

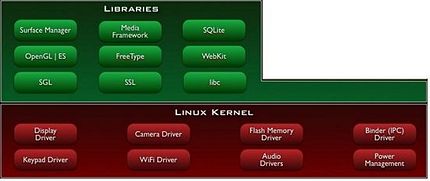
Android - это программный стек для мобильных устройств, который включает операционную систему, программное обеспечение промежуточного слоя (middleware), а также основные пользовательские приложения (e-mail-клиент, календарь, карты, браузер, контакты и другие).

Архитектура Android построена на основе ядра Linux версии 2.6. Оно отвечает за такие системные службы, как управление безопасностью, памятью, процессами, включает сетевой стек и модель драйверов. Кроме того, это ядро также играет роль абстрактного слоя между аппаратным уровнем и остальной частью программного стека.

[](http://wiki.androidfan.ru/%D0%A4%D0%B0%D0%B9%D0%BB:Arch_1.jpg)

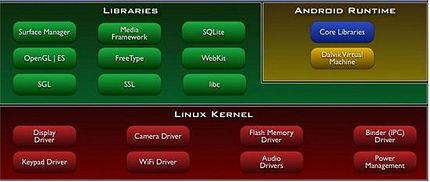
**Рис. 1** Ядро Linux

Следующий уровень в иерархической системе - библиотеки, написанные на C/C++. В медиабиблиотеках (Media Libraries) заложена поддержка основных форматов и кодеков, включая MPEG4, H.264, MP3, WMA, WAV, AAC, AMR, JPG, PNG. Графические библиотеки (3D Libraries) поддерживают API OpenGL ES 1.0 и обеспечивают поддержку аппаратных 3D-ускорителей. Как отмечается, в одном приложении можно одновременно использовать двух- и трехмерную графику. Еще одной примечательной библиотекой является LibWebCore (WebKit) - движок для web-браузера Android, используемый также в популярном браузере Safari от Apple.

[](http://wiki.androidfan.ru/%D0%A4%D0%B0%D0%B9%D0%BB:Arch_2.jpg)

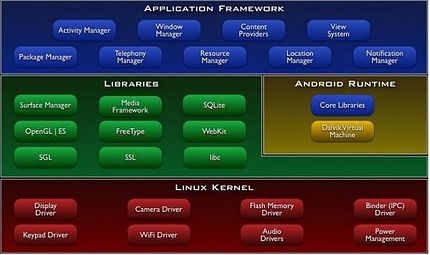
**Рис. 2** Уровень библиотек

Одной из важнейших частей стека является Android Runtime, состоящая из виртуальной Java-машины Dalvik и набора библиотек ядра. Dalvik позволяет поддерживать одновременную работу нескольких приложений и выполняет файлы в специальном формате .dex, оптимизированном для устройств с малым количеством памяти. Библиотеки ядра написаны на языке Java и включают большой набор классов, которые поддерживают широкий диапазон функциональных возможностей.

[](http://wiki.androidfan.ru/%D0%A4%D0%B0%D0%B9%D0%BB:Arch_3.jpg)

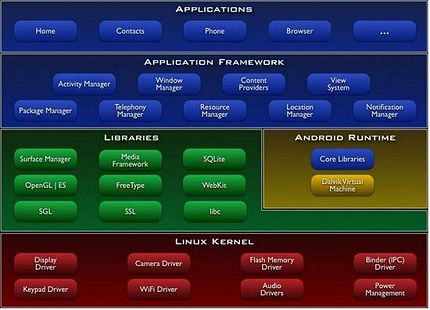
**Рис. 3** Виртуальная Java-машина Dalvik и набор библиотек ядра

Следующий уровень - Application Framework (каркас приложений). Этот уровень фактически представляет собой инструментарий, которым пользуются все приложения. Здесь стоит выделить такой компонент, как Content Providers (провайдеры данных), который является уникальной особенностью Android. Провайдеры данных позволяют, например, объединять информацию из интернета с данными телефона - контактной информацией или географической локацией, - чтобы реализовать новые функции. Кроме того, данные могут совместно использоваться разными приложениями.

[](http://wiki.androidfan.ru/%D0%A4%D0%B0%D0%B9%D0%BB:Arch_4.jpg)

**Рис. 4** Уровень каркаса приложений

И, наконец, на вершине пирамиды - уровень приложений (Applications). Что интересно, Android не делает разницы между основными приложениями телефона и сторонним программным обеспечением - таким образом, ключевые компоненты, такие как набор номера, рабочий стол или почтовый клиент GMail, можно заменить альтернативными аналогами. Приложения для Android пишутся на языке Java.

[](http://wiki.androidfan.ru/%D0%A4%D0%B0%D0%B9%D0%BB:Arch_5.jpg)

**Рис. 5** Уровень приложений

Прежде всего, система Android предназначена для мобильных устройств. Это означает, что наивысший приоритет отдается сохранению энергии батареи и эффективному управлению ограниченными ресурсами памяти. В системе Android можно выделить пять слоев:

Ядро Linux, работающее на архитектуре ARM (Acorn RISC Machine), является базой для остальных слоев. Linux - проверенная и высоконадежная технология, а семейство процессоров ARM известно своей высокой производительностью при низком энергопотреблении.

Набор библиотек предоставляет общий повторно используемый код низкого уровня для основных функций, таких как кодеки, позволяющие кодировать и декодировать цифровой звук и видео, функции для отображения качественной графики на маленьких дисплеях, поддержка зашифрованного трафика TCP/IP в облаке, поддержка компонентов для Web-серфинга (WebKit), функции SQL-СУБД (SQLite) и стандартная для Linux-систем функциональность библиотек C.

Интерпретатор байт-кода реального времени Dalvik, имеющий большое сходство с интерпретатором Java™, отличается несколькими свойствами, обеспечивающими Android уникальную безопасность и минимизацию потребления энергии. Каждое запущенное в текущий момент приложение имеет свой собственный пользовательский идентификатор и собственную копию интерпретатора, что обеспечивает строгое разграничение процессов для повышения надежности и безопасности.

Структура приложений Android позволяет использовать и заменять компоненты так, как вы считаете нужным. Эти интегрируемые компоненты являются высокоуровневыми Java-классами, составляющими Android API.

Стандартные приложения Android включают в себя браузер WebKit, календарь Google, клиент Gmail, приложение Gmaps, SMS-мессенджер и e-mail клиент. Приложения для Android написаны на Java, и вы всегда можете загрузить дополнительные приложения из магазина Android.

Далее, каждое приложение Android можно разделить на следующие функциональные модули:

* Activities (Действия)
* Intents (Намерения)
* Services (Службы)
* Security model (Модель безопасности)

Действия (Activities)

Действия (Activities) - это компоненты приложения Android, расширяющие базовый класс Activity и определяющие интерфейс, в который входит Представление (View), отвечающее на различные События (Events). Если приложение состоит из трех окон (например, окно авторизации, окно отображения текста и окно просмотра файлов), каждое из них, как правило, представляется отдельным классом Activity.

Android сохраняет историю действий для каждого приложения, запущенного со стартового экрана, и вы можете с помощью кнопки Back передвигаться назад по этой истории.

Намерения (Intents)

Намерения (Intents), подобно действиям, являются специальными классами в коде приложения, которые определяют и описывают запросы приложения на выполнение каких-либо операций. Намерения добавляют слой, позволяющий оперировать компонентами с целью их повторного использования и замещения. Например, приложение может иметь кнопку Клиенты, которая, будучи нажатой, отображает список клиентских контактов. С помощью Намерения вы можете использовать любой компонент для их отображения вместо стандартного.

В некоторых случаях это может быть очень мощным средством интеграции приложений. Например, топографическая карта в определенных случаях может быть лучше, чем изображение карты по умолчанию.

Классы наподобие BroadcastReceiver определяют код, который выполняется после возникновения внешних событий. Так можно отслеживать события типа срабатывания таймера или входящего звонка. Как правило, такой код не отображает никакого окна, но с помощью класса NotificationManager можно сообщить пользователю информацию, требующую его внимания.

Службы (Services)

Службой является приложение, запущенное в фоновом режиме и не имеющее графического интерфейса. Обычно подобные программы работают в фоне в течение долгого времени. Отличным примером является медиаплеер, проигрывающий треки из списка. Несмотря на то, что у такого приложения есть интерфейс, позволяющий пользователям составлять свои плейлисты, в процессе проигрывания песен оно может работать в автоматическом режиме службы.

Модель безопасности (Security model)

Модель безопасности Android позволяет программам иметь собственную область данных. Если разработчик хочет обобществить данные между несколькими различными программами, он может использовать поставщики данных (content providers).

Пакет android.provider содержит набор классов и интерфейсов, обеспечивающих доступ к многочисленным встроенным базам данных Android. Приложения с подобными интерфейсами могут с легкостью работать с контактами, фотографиями и музыкой.

Силу модели Open Source, как и силу свободного творчества, нельзя недооценивать. Не ограниченная закрытыми API и интересами собственника, которые зачастую препятствуют прогрессу программного обеспечения, платформа Android сформировала большое и очень активное сообщество, суммарная ценность которого больше ценности его отдельных частей.

Сердцем Android является ARM Linux. Одно это должно вселять уверенность в возможность быстрого роста этой платформы. Linux является быстрой и защищенной операционной системой, с которой знакомы тысячи программистов. Многие основанные на Linux системы работают без отказов на протяжении многих лет, будучи подключенными к облаку и обслуживая его запросы — и это является лучшим доказательством их надежности.

## Среда разработки под OS Android

Есть несколько вариантов выбора среды разработки для Android: для Microsoft® Windows® XP или Vista®, Macintosh OS X (V10.4.8 или более поздняя версия, только для архитектуры x86), а также Desktop Linux (предпочтительно Ubuntu). Для всех этих операционных систем можно загрузить комплект программ разработчика (SDK).

Можно работать в IDE с графическим интерфейсом типа Eclipse или NetBeans от Sun Microsystems или использовать метод "настоящего программиста", пользуясь командной строкой и стандартным текстовым редактором.

Сообщество Android поддерживает обмен полезной информацией; к тому же стремятся и компании, стоящие за созданием Android — Google, HTC, T-Mobile и другие участники проекта Open Handset Alliance. Open Handset Alliance действительно означает открытое общение между людьми (программистами, пользователями, маркетологами и т.п.), открытые протоколы и открытые API.

Однако для Anroid можно предлагать и закрытые программные продукты – выбор возможных лицензий достаточно широк. Такой подход дает возможность коммерческим компаниям не игнорировать платформу, но при этом не ограничивает выбор пользователей. В выигрыше оказываются все.

# РЕАЛИЗАЦИЯ

## 5.1 Структура программы

Приложения для Android состоят из компонентов, который система может запускать и управлять так, как ей необходимо. Для этого система должна быть в состоянии запустить процесс для приложения в котором находятся требуемые компоненты, и инициализировать нужные ей объекты. Одним из компонентов Android-приложения является деятельность (activity).

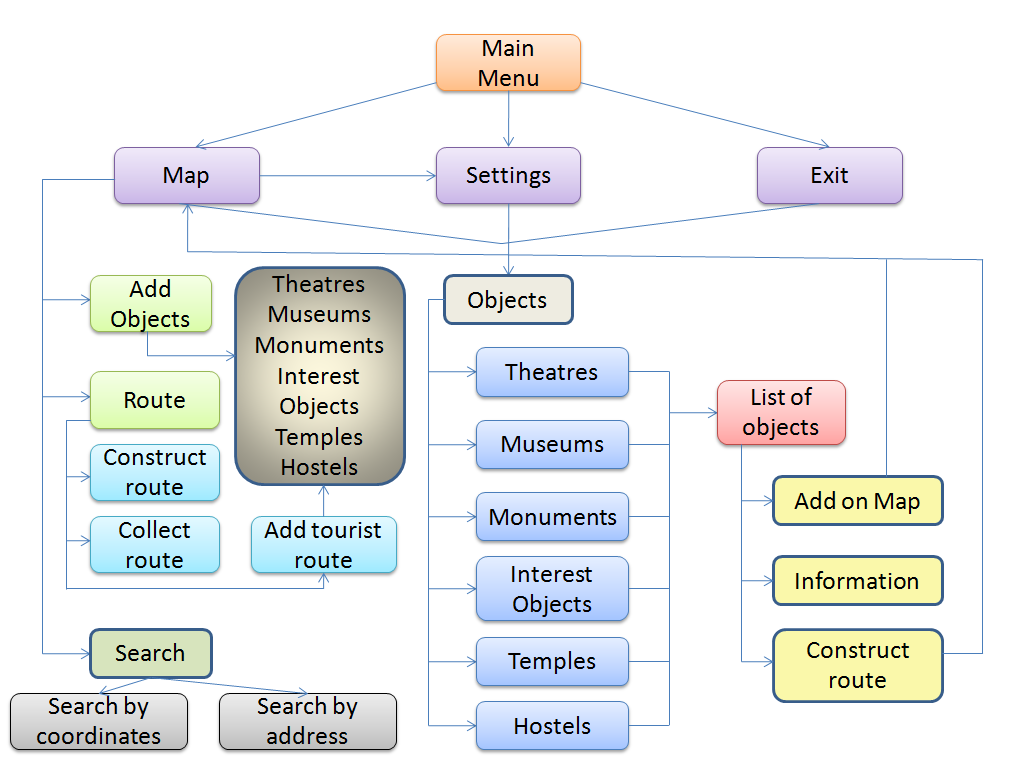
Activity представляет собой визуальный интерфейс (отдельный экран) для одного действия, которое пользователь может совершить.

Приложение может состоять из одного activity или из нескольких. Это зависит от типа приложения и его дизайна. Одно activity может вызвать другое.

Каждое activity задаёт окно для отображения, которое, обычно, занимает весь экран, но может быть меньше и плавать поверх других окон. Activity может использовать дополнительные окна, например, всплывающий диалог, который требует промежуточного ответа пользователя, или окно, которое отображает пользователям важную информацию при выборе элемента, заслуживающего особого внимания.

Визуальный интерфейс строится на основе иерархии визуальных компонентов — объектов, производных от базового класса View. Android имеет ряд готовых к использованию компонентов, включая кнопки, текстовые поля, полосы прокрутки, меню, флажки и многое другое.

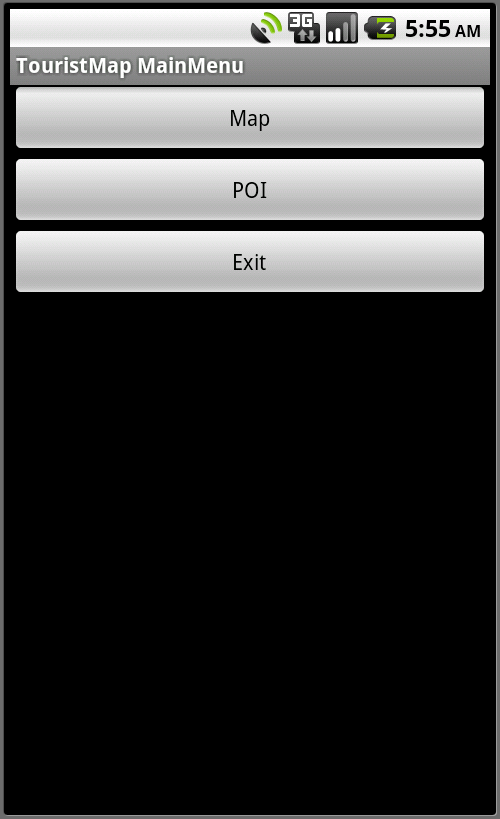
На рисунке 6 показано дерево взаимодействия Activities (деятельностей) программы.



**Рис. 6** Дерево взаимодействия Activities (деятельностей) программы.

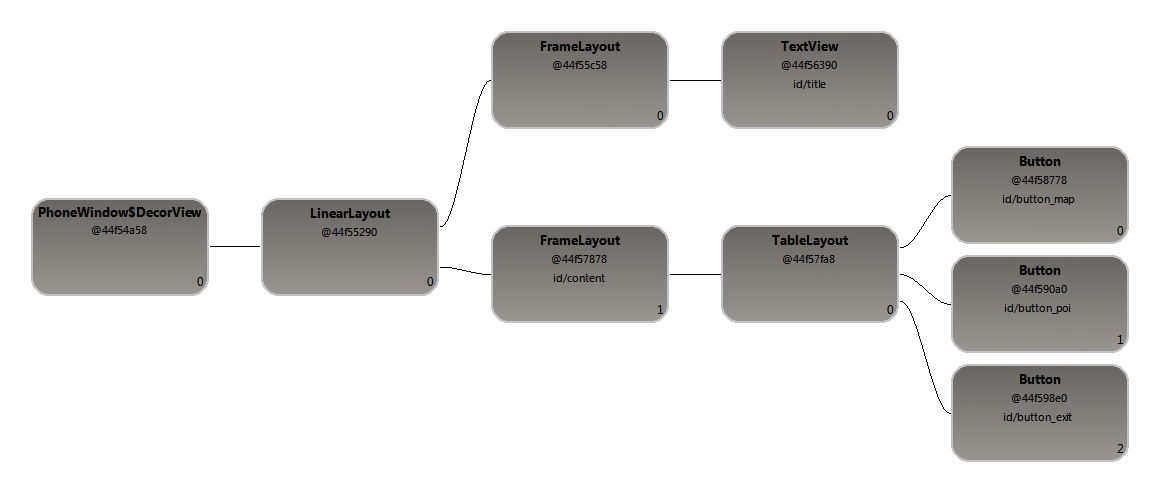
Главная деятельность (Main Menu)

Рассмотрим структуру главной деятельности. Внешний вид окна представлен на рисунке 7.



**Рис. 7** Главное окно программы

Рисунок 8 отображает иерархию разметки главного окна программы и её свойства.



**Рис. 8** Иерархия разметки главного окна.

Как видно из рисунка конечными элементами разметки, которые видит пользователь, являются следующие элементы: текстовое поле (Title), 3 кнопки (button\_map, button\_poi, button\_exit).

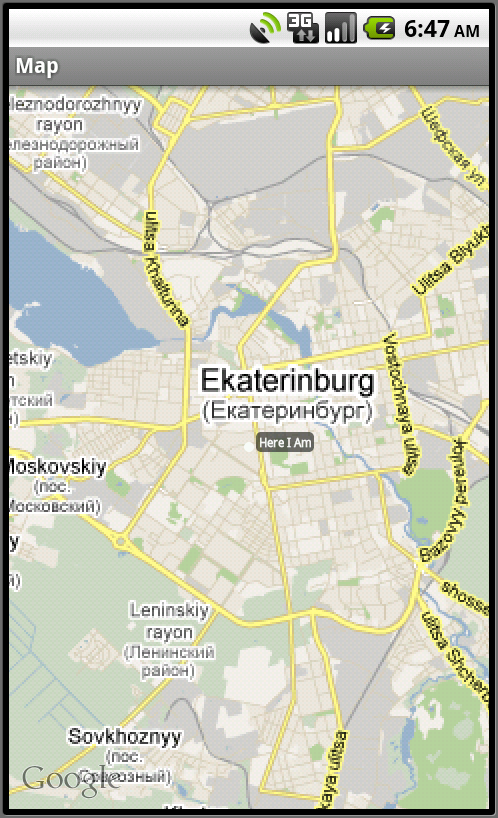
При нажатии на кнопку <Map> открывается активити содержащую карту Google Map.

При нажатии на кнопку <POI> открывается активити содержащее список групп туристических объектов города Екатеринбурга.

При нажатии на кнопку <Exit> программа закрывается.

Деятельность Map

Рассмотрим структуру деятельности Map. Внешний вид окна представлен на рисунке 9.

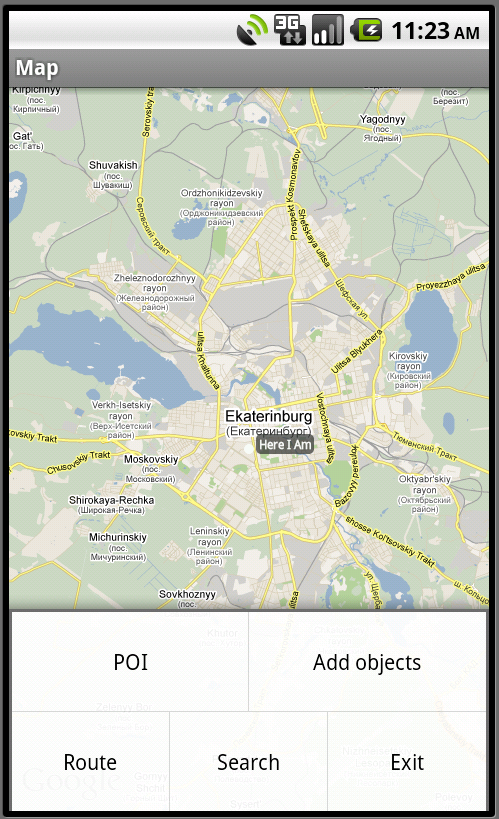


**Рис.9** Окно Map

При запуске данной активити определяется текущее местоположение мобильного устройства с помощью встроенного GPS-приёмника. Карта центрируется по полученным координатам. Также поддерживается возможность изменения масштаба карты и передвижения центра экрана по карте.

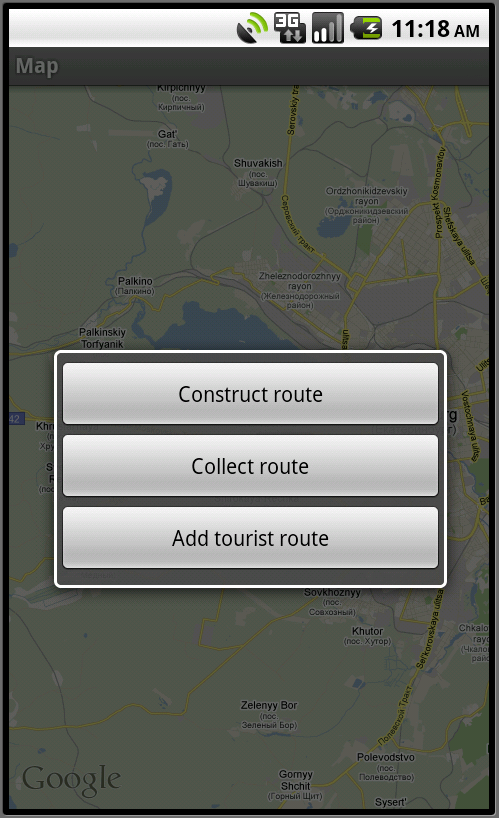
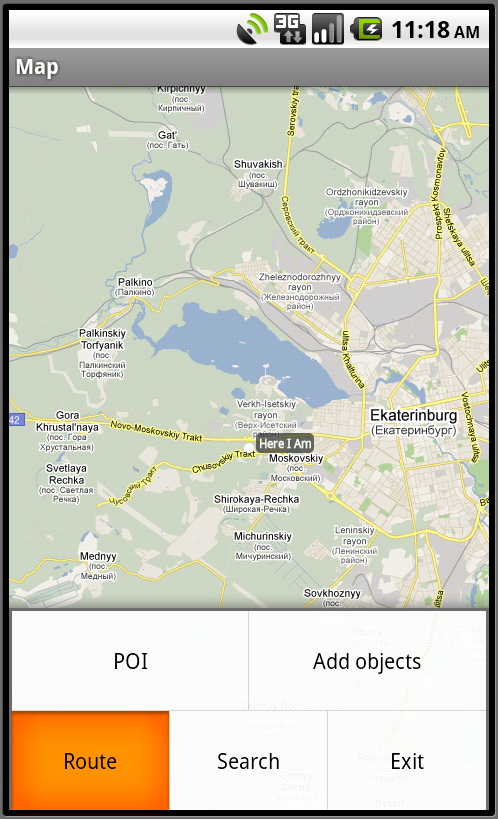
При движении мобильного устройства координаты текущего местоположения постоянно обновляются, при этом маркер указывающий местоположение мобильного устройства перемещается по карте в соответствии с новыми координатами.

По нажатию на кнопку <Menu> мобильного устройства появляется меню выбора опций, содержащее 5 пунктов: POI, Add objects, Route, Search, Exit (рисунок 10).



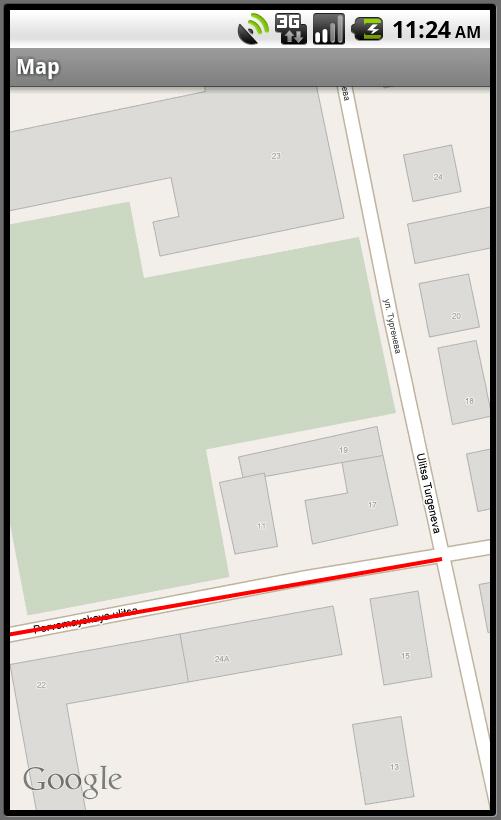
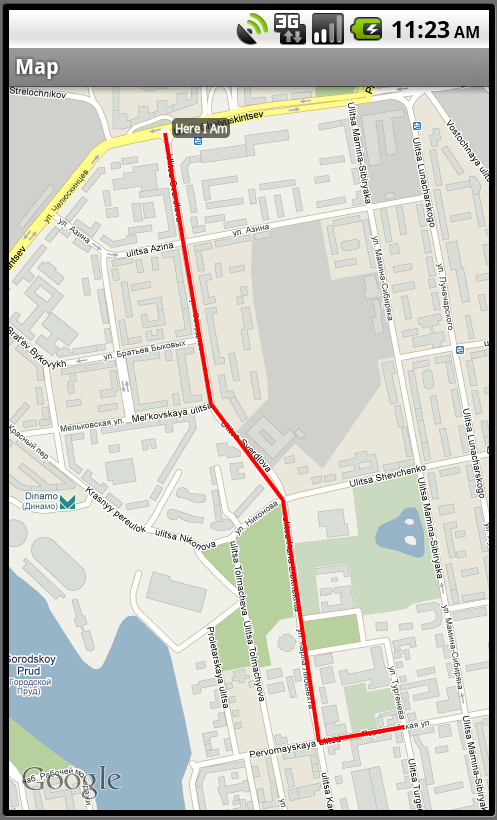
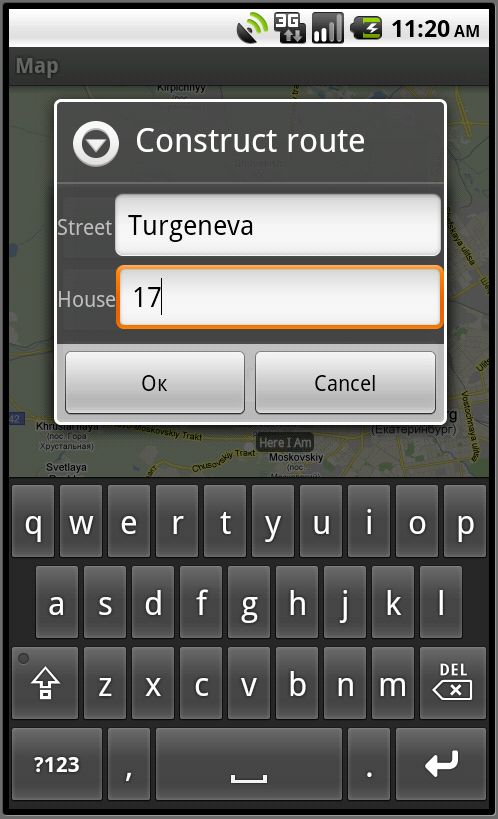
**Рис. 10** Меню окна Map

При выборе пункта Route открывается всплывающее окно содержащее три кнопки (Construct route, Collect route, Add tourist route) (рисунок 11).



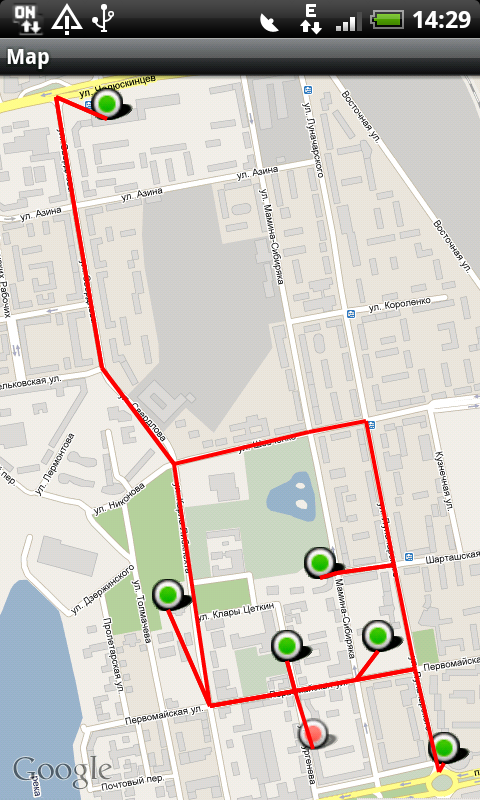
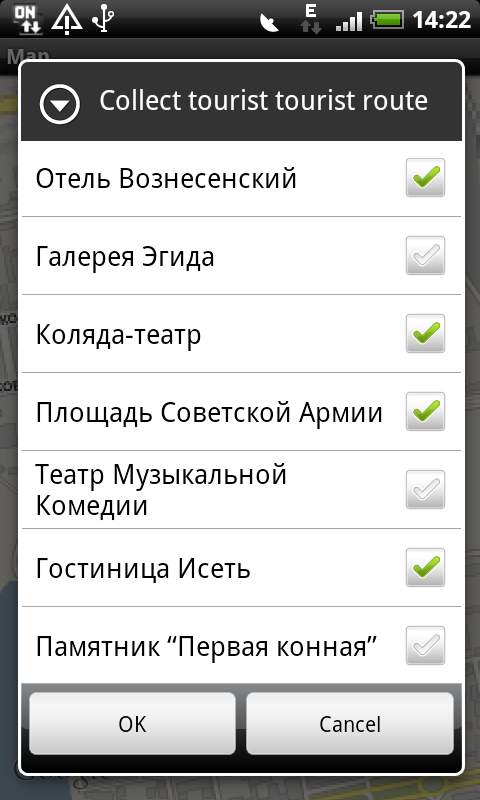
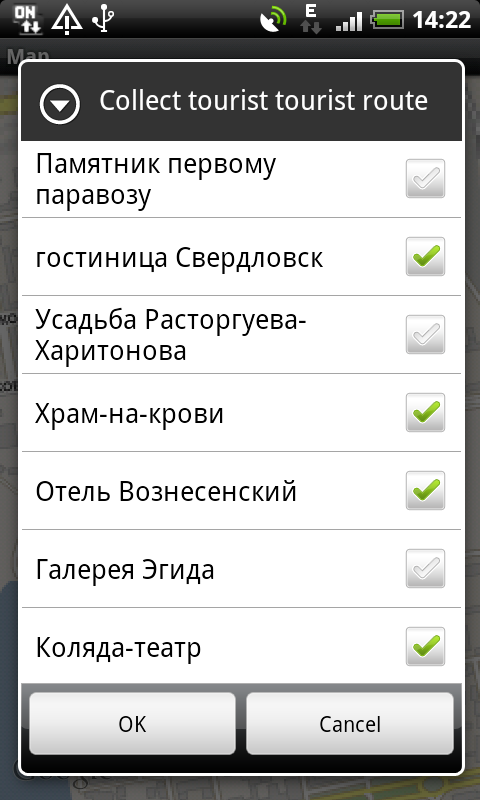
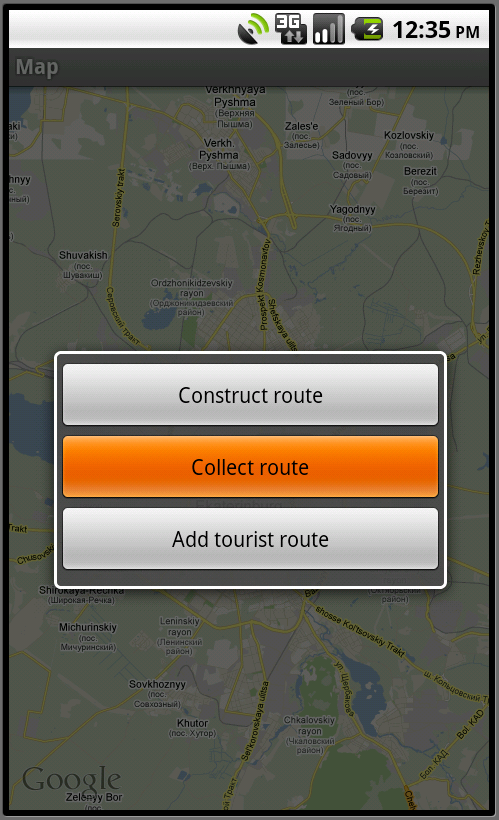
**Рис.11** Всплывающее окно Route

При нажатии на кнопку Construct route, появляется форма с двумя текстовыми полями для ввода адреса до которого будет строится маршрут. Маршрут прокладывается от текущего местоположения мобильного устройства (рисунок 12)



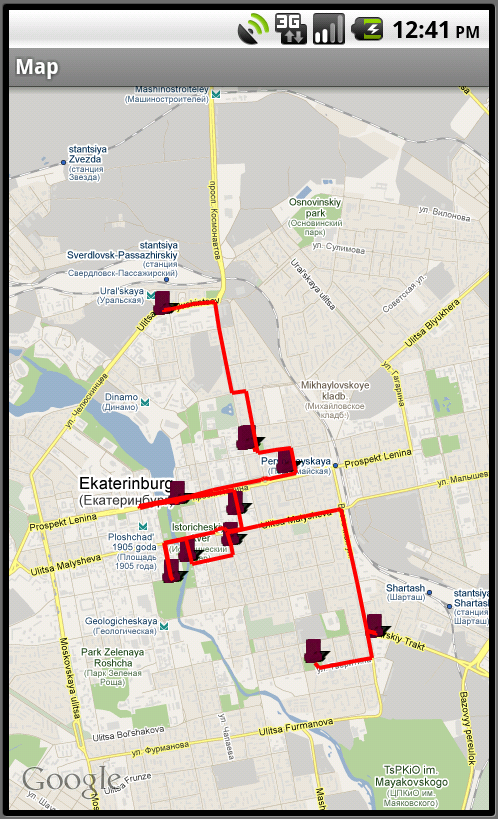
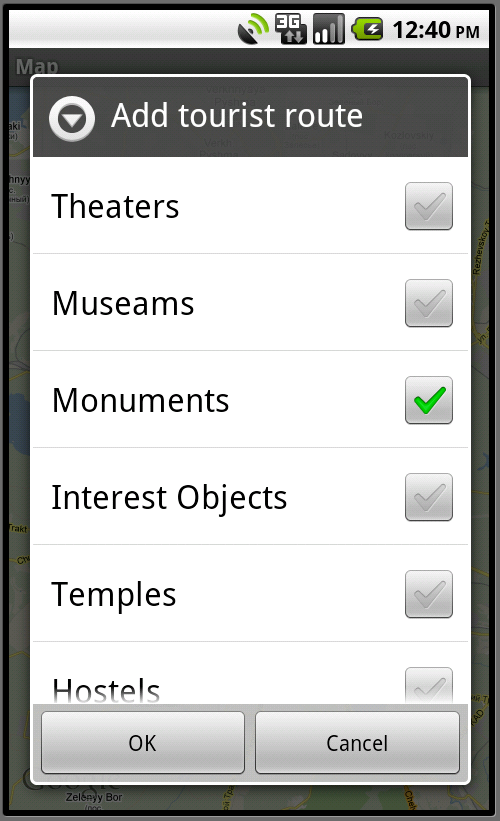
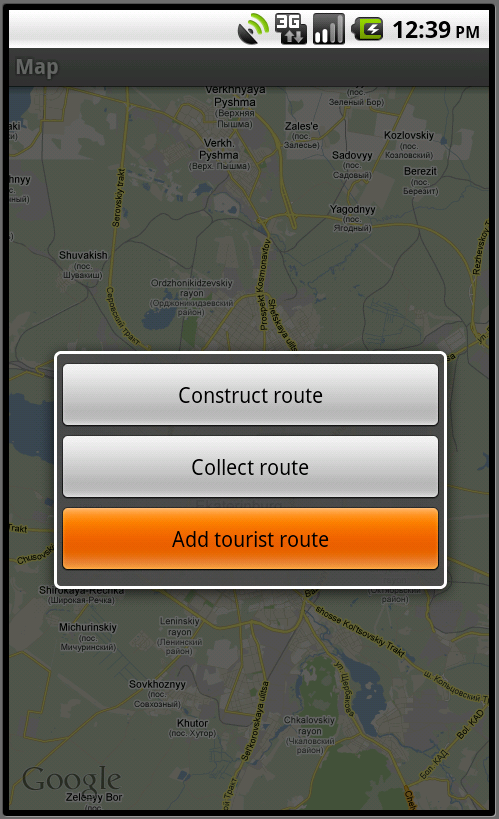
**Рис. 12** Ввод адреса и построение маршрута

При нажатии на кнопку Collect route появляется всплывающее окно содержащее список объектов с флажками. Пользователь может выбрать несколько объектов и построить маршрут проходящий через выбранные точки (рисунок 13)



**Рис. 13** Всплывающее окно Collect route

При нажатии на кнопку Add tourist route открывается всплывающее окно содержащее список групп объектов, которые можно отобразить на карте при активации флажков (рисунок 14).



**Рис. 14** Выбор группы объектов и отображение маршрута на карте

## 5.2 Программная реализация

Структура проекта представляет собой дерево каталогов. При компиляции в каталоге проекта создается папка bin/. Откомпилированный код Java-классов вместе с файлами данных и ресурсов помещается в архивный файл с расширением .apk. Этот файл используется для распространения приложения и установки его на мобильных устройствах.

Каталог ресурсов

В данном каталоге хранятся используемые в приложении статические файлы ресурсов: изображения, строки, файлы разметки и т.д.

В подкаталоге **res/drawable/** размещены все графические файлы, используемые в приложении. Например: файл **icon.png** - значок приложения, который будет отображаться в меню запуска установленных на телефоне приложений.

В подкаталоге **res/values/** хранятся общие константы для всего приложения. Например:

<string name=*"menu\_info"*>Info</string>

<string name=*"search\_name"*>Name</string>

Подкаталог **res/layout/** содержит файлы разметки в формате XML, которые определяют внешний вид окна деятельности и расположение на нем элементов управления. Преимущество объявления пользовательского интерфейса в XML – файле состоит в том, что это дает возможность отделить структуру интерфейса приложения от программного кода, который управляет поведением приложения.

Дополнительно, объявление разметки в XML – файле облегчает возможность по визуализации структуры пользовательского интерфейса, что упрощает отладку приложения.

Рассмотрим структуру файла разметки на примере файла **objects.xml.**

Файл разметки должен содержать только один элемент, который должен быть объектом View или ViewGroup.

<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:orientation=*"vertical"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"fill\_parent"* >

После определения корневого элемента, добавляются дополнительные объекты, тем самым формируется иерархия элементов, которую определяет создаваемая разметка.

<ListView

android:layout\_height=*"fill\_parent"*

android:id=*"@android:id/list"*

android:layout\_width=*"fill\_parent"*

android:drawSelectorOnTop=*"false"*/>

Элемент <ListView> представляет собой вертикальный список с прокруткой. Связанные со списком данные ListView получает от объекта ListAdater.

Файлы map.xml, object\_info.xml, object\_menu.xml, object\_view.xml, objects.xml, search\_address.xml, search\_coord.xml, search\_menu.xml, search.xml устроены похожим образом.

Структура файла R.java

Когда проект компилируется в первый раз, среда создает класс R и помещает его в файл R.java. Этот класс используется в коде программы для обращения к ресурсам, которые находятся в каталоге res/.

Класс R содержит набор внутренних классов с идентификаторами ресурсов, которые он создает в зависимости от внутреннего содержимого каталога res/: drawable, layout, id, string.

Например:

**public static final** **class** drawable {

**public static final** **int** *androidmarker*=0x7f020000;

**public static final** **int** *h\_hotel*=0x7f020001;

…

**public static final** **class** id {

**public static final** **int** *ObjectAddress*=0x7f07000b;

**public static final** **int** *ObjectName*=0x7f07000a;

**public static final** **int** *address*=0x7f070006;

**public static final** **int** *button\_add*=0x7f070007;

…

Структура файла AndroidManfest.xml

Манифест – структурный XML – файл, задающий конфигурацию приложения. Он объявляет компоненты приложения, перечисляет любые библиотеки, связанные с приложением, а также объявляет разрешения, которые приложение предоставляет.

Файл манифеста инкапсулирует всю структуру Android-приложения, его функциональные возможности и конфигурацию.

Структура манифеста многоуровневая. Порядок расположения элементов на одном уровне произвольный. Элемент <application> является основным элементом манифеста и содержит множество дочерних элементов, определяющих структуру и работу приложения. Элементы <manifest>, <application> и <uses-sdk> являются обязательными.

<manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*

**package**=*"com.touristmap"* android:versionCode=*"1"* android:versionName=*"1.0."*>

<uses-sdk android:minSdkVersion=*"8"* />

<application android:icon=*"@drawable/icon"* android:label=*"@string/app\_name"*>

Элемент <application> содержит описание компонентов приложения, доступных в пакете. У данного элемента есть дочерние, которые объявляют каждый из компонентов, входящих в состав приложения.

<provider android:name=*".providers.TemplesProvider"*

android:authorities=*"com.touristmap.TemplesProvider"*>

</provider>

…

<uses-library android:name=*"com.google.android.maps"* />

<activity android:name=*".activities.MainActivity"*

android:label=*"TouristMap MainMenu"*

android:launchMode=*"singleTask"*>

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

</activity>

…

Элемент <uses-permission> запрашивает разрешения, которые приложению должны быть предоставлены системой для его нормального функционирования.

<uses-permission android:name=*"android.permission.INTERNET"* />

Файлы деятельностей

Файл деятельности представляет собой класс содержащий код который реализует логику работы приложения в данном окне. В каждом из таких классов обязательно должен определяться метод обратного вызова onCreate. Этот метод вызывается при создании деятельности. Внутри метода настраивают статический интерфейс деятельности – создают представления, связывают данные со списками и т.д. Этот метод принимает один параметр – объект Bundle, содержащий предыдущее состояние деятельности (если это состояние было сохранено)

В файле MainActivity.java задается функциональность главного окна программы: определяются действия происходящие по нажатию кнопок, а именно запуск активити соответствующих кнопкам.

**final** Button button\_map = (Button) findViewById (R.id.*button\_map*);

button\_map.setOnClickListener(**new** View.OnClickListener() {

@Override

**public void** onClick(View v) {

Intent mapsActivity = **new** Intent(MainActivity.**this**,MapsActivity.**class**);

startActivity(mapsActivity);

}

});

Также проверяется включен ли GPS приемник на устройстве, если GPS выключен, то программа предлагает его включить, т.к. это необходимое условие для работы приложения.

LocationManager locationManager = (LocationManager)getSystemService(Context.*LOCATION\_SERVICE*);

**if** (!locationManager.isProviderEnabled(LocationManager.*GPS\_PROVIDER*)) {

**final** AlertDialog.Builder builder = **new** AlertDialog.Builder(**this**);

builder.setMessage("GPS выключен")

.setCancelable(**false**)

.setPositiveButton("Настройки GPS", **new** DialogInterface.OnClickListener() {

**public void** onClick(**final** DialogInterface dialog, **final** **int** id) {

Intent intent = **new** Intent(Settings.*ACTION\_LOCATION\_SOURCE\_SETTINGS*);

startActivityForResult(intent, 1);

}

})

.setNegativeButton("Отмена", **new** DialogInterface.OnClickListener() {

**public void** onClick(**final** DialogInterface dialog, **final** **int** id) {

finish();

}

});

**final** AlertDialog alert = builder.create();

alert.show();

}

Файл MapsActivity.java описывает функционал для работы с картой:

* Отображение карты (получение контроля над картой, определение вида отображения карты, добавление кнопок масштабирования, загрузка слоев)
* Создание критериев работы с GPS, получение GPS провайдеров, получение координат последнего местоположения (если оно известно), получение координат текущего местоположения (с заданным интервалом обновления)

LocationManager locationManager;

String context = Context.*LOCATION\_SERVICE*;

locationManager = (LocationManager) getSystemService(context);

Criteria criteria = **new** Criteria();

criteria.setAccuracy(Criteria.*ACCURACY\_FINE*);

criteria.setAltitudeRequired(**false**);

criteria.setBearingRequired(**false**);

criteria.setCostAllowed(**true**);

criteria.setPowerRequirement(Criteria.*POWER\_LOW*);

String provider = locationManager.getBestProvider(criteria, **true**);

Location location = locationManager.getLastKnownLocation(provider);

updateWithNewLocation(location);

locationManager.requestLocationUpdates(provider, 1500, 3,

locationListener);

* Добавление маркера текущего местоположения

positionOverlay = **new** MyPositionOverlay();

List<Overlay> overlays = myMapView.getOverlays();

overlays.add(positionOverlay);

* Процедура считывания данных из текстового файла ReadInTxt(String in). В файле задается граф улиц. Тип представления графа – список предшествующих вершин.
* Процедура считывания координат точек из текстового файла ReadCoord(String coord). В файле хранятся координаты точек, входящих в граф улиц.
* Процедура записи в текстовый файл кратчайшего пути в графе WriteOutTxt(String out).
* Процедура записи в текстовый файл координат точек кратчайшего пути в графе WriteCoord(String out\_coord).
* Процедура отыскания ближайшей точки маршрута к точке текущего местоположения устройства min\_point(**double** lt, **double** lg).
* Процедура выбора наименьшего числа MINIM(**int** a, **int** b).
* Вспомогательная процедура для алгоритма Дейкстры MIN(TreeSet<Integer> F, **int**[] d).
* Процедура отыскания кратчайшего пути в графе с помощью алгоритма Дейкстры DeikstraMin().
* Процедура обновления текущего местоположения updateWithNewLocation(Location location).
* Создание всплывающего меню в процедуре onCreateOptionsMenu(Menu menu). Создание обработчика событий выбора пункта меню onOptionsItemSelected(MenuItem item).
* Создание всплывающего диалога построения маршрутов.
* Процедура добавлении группы объектов на карту addGroup(Uri uri, Drawable drawable)
* Процедура добавления точки на карту isAddLocationPoint(**double** netlat, **double** netlng, String name)
* Процедуры вызова диалогов построения маршрутов ConstructRouteDialog(),CollectRouteDialog(),AddTouristRouteDialog()

В файле ObjectsEditorActivity.java описывается функциональность окна POI:

* Создается список групп объектов
* Создание обработчика событий выбора элементов списка (групп объектов), а именно переходы на новые активити содержащие списки объектов данной группы.

В файле TheatersViewActivity.java описывается функциональность окна Theaters:

* Создается список объектов, данные о которых берутся из соответствующей базы данных
* Создание обработчика событий выбора элементов списка (всплывающее диалоговое окно)
* Процедура построения маршрута от текущего местоположения до выбранного объекта ConstructRouteToObject().

Файлы MuseamViewActivity.java, ArhitectureViewActivity.java, InterestViewActivity.java, TemplesViewActivity.java, HostelViewActivity.java аналогичным образом определяют структуру соответствующих окон.

Вспомогательные файлы для работы с базой данных

Рассмотрим структуру данных файлов на примере ObjectDbHelper.java. Класс ObjectDbHelper, расширяющий класс SQLiteOpenHelper представляет таблицу Object базы данных. В нем объявлена структура таблицы и в методе onCreate() производится создание таблицы и заполнение её текстовыми данными при первом запуске приложения на устройстве.

**public void** onCreate(SQLiteDatabase db){

db.execSQL("CREATE TABLE " + *TABLE\_NAME* + " (\_id INTEGER PRIMARY KEY AUTOINCREMENT, "+ *NAME* + " TEXT, " + *ADDRESS* + " TEXT, " + *LAT* + " TEXT, " + *LON* + " TEXT);");

ContentValues values = **new** ContentValues();

values.put(*NAME*, "Театр Эстрады");

values.put(*ADDRESS*, "8 Марта 15");

values.put(*LAT*, "56.835662");

values.put(*LON*, "60.600392");

db.insert(*TABLE\_NAME*, *NAME*, values);

}

Файлы MuseamDbHelper.java, ArhitectureDbHelper.java, InterestDbHelper.java, TemplesDbHelper.java, HostelDbHelper.java устроены аналогичным образом.

Провайдеры

Рассмотрим структуру провайдера на примере файла класса провайдера ObjectViewProvider.java.

Класс ObjectViewProvider данного файла расширяющий базовый класс ContentProvider представляет логику доступа к содержимому базы данных Object.

В методе обратного вызова onCreate() который вызывается системой при создании экземпляра ContentProvider, инициализируется объект SQLiteDatabase.

**private** SQLiteDatabase db;

…

@Override

**public boolean** onCreate() {

db = (**new** ObjectDbHelper(getContext())).getWritableDatabase();

**return** (db == **null**) ? **false** : **true**;

}

Метод query(Uri url, String[] projection,

String selection, String[] selectionArgs, String sort)реализует запрос к БД.

Метод insert(Uri url, ContentValues inValues) используется для вставки данных и возвращает клиенту контент-провайдера Uri вставляемой строки.

Метод update(Uri url, ContentValues values,

String where, String[] whereArgs) реализует модификацию данных, а метод delete(Uri url, String where, String[] whereArgs) – удаление данных.

Файлы MuseamProvider.java, InterestProvider.java, ArhitectureProvider.java, TemplesProvider.java, HostelProvider.java устроены аналогичным образом.

# ЗАКЛЮЧЕНИЕ

Подведем итоги нашей работы. Была поставлена задача создания туристической карты для мобильных устройств, работающих на платформе Android. Основной идеей было использование функции спутниковой навигации позволяющей расширить специализированую активную карту до электронного гида. Был разработан прототип программы, реализующий нашу идею. Программа представляет собой «электронный туристический гид», используя который турист может познакомиться с достопримечательностями города. На следующем этапе необходимы консультации со специалистами в области туризма и аккуратное опробование на реальных пользователях.

Следует отметить, что данный прототип нуждается в развитии. Перечислим некоторые идеи для дальнейшей доработки нашей системы:

* Создание локальной карты города, с которой можно работать без подключения к интернету.
* Локализация программы на различные языки.
* Расширение базы данных туристических объектов.
* Расширение области построения маршрутов.
* Разработка более быстродействующих алгоритмов поиска маршрутов.
* Создание возможности прокладки пеших маршрутов.
* Добавление голосовых подсказок.

# ЛИТЕРАТУРА

* 1. Ципилева Т.А. - Геоинформационные системы.
  2. <http://www.dataplus.ru/> - ООО "ДАТА+" – распространение в России и других странах СНГ передовых геоинформационных технологий ESRI.
  3. <http://ru.wikipedia.org/>
  4. <http://www.3dnews.ru/> - Навигация в мобильных устройствах – GPS и не только
  5. Дорогов А.Ю., Лесных В.Ю., Раков И.В., Титов Г.С. Алгоритмы оптимального движения мобильных объектов по пересеченной местности и транспортной сети
  6. Степанова Д. М. - Построение оптимального маршрута обзорной экскурсии по памятникам культурного наследия с использованием геоинформационных технологий.
  7. Голощапов А.Л. - Google Android. Программирование для мобильных устройств
  8. Reto Meier – Professional Android™ 2 Application development
  9. Rick Rogers, John Lombardo, Zigurd Mednieks, and Blake Meike - Android Application Development

# ПРИЛОЖЕНИЕ

**Файл AndroidManifest.xml**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*

**package**=*"com.touristmap"* android:versionCode=*"1"* android:versionName=*"1.0."*>

<uses-sdk android:minSdkVersion=*"8"* />

<application android:icon=*"@drawable/icon"* android:label=*"@string/app\_name"*>

<provider android:name=*".providers.ObjectViewProvider"*

android:authorities=*"com.touristmap.ObjectViewProvider"*>

</provider>

<provider android:name=*".providers.MuseamProvider"*

android:authorities=*"com.touristmap.MuseamProvider"*>

</provider>

<provider android:name=*".providers.ArhitectureProvider"*

android:authorities=*"com.touristmap.ArhitectureProvider"*>

</provider>

<provider android:name=*".providers.InterestProvider"*

android:authorities=*"com.touristmap.InterestProvider"*>

</provider>

<provider android:name=*".providers.TemplesProvider"*

android:authorities=*"com.touristmap.TemplesProvider"*>

</provider>

<provider android:name=*".providers.HostelProvider"*

android:authorities=*"com.touristmap.HostelProvider"*>

</provider>

<uses-library android:name=*"com.google.android.maps"* />

<activity android:name=*".activities.MainActivity"*

android:label=*"TouristMap MainMenu"*

android:launchMode=*"singleTask"*>

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

</activity>

<activity android:name=*".activities.MapsActivity"*

android:label=*"Map"* />

<activity android:name=*".activities.ObjectsEditorActivity"*

android:label=*"Objects"*>

</activity>

<activity android:name=*".activities.TheatersViewActivity"*

android:label=*"@string/title\_view"*>

</activity>

<activity android:name=*".activities.MuseamViewActivity"*

android:label=*"@string/title\_view"*>

</activity>

<activity android:name=*".activities.ArhitectureViewActivity"*

android:label=*"@string/title\_view"*>

</activity>

<activity android:name=*".activities.InterestViewActivity"*

android:label=*"@string/title\_view"*>

</activity>

<activity android:name=*".activities.TemplesViewActivity"*

android:label=*"@string/title\_view"*>

</activity>

<activity android:name=*".activities.HostelViewActivity"*

android:label=*"@string/title\_view"*>

</activity>

<activity android:name=*".activities.ViewObjectOnMapActivity"*

android:label=*"@string/title\_view"*>

</activity>

</application>

<uses-permission android:name=*"android.permission.INTERNET"* />

<uses-permission android:name=*"android.permission.ACCESS\_COARSE\_LOCATION"* />

<uses-permission android:name=*"android.permission.ACCESS\_FINE\_LOCATION"* />

</manifest>

**Файл R.java**

/\* AUTO-GENERATED FILE. DO NOT MODIFY.

\*

\* This class was automatically generated by the

\* aapt tool from the resource data it found. It

\* should not be modified by hand.

\*/

**package** com.touristmap;

public **final** class R {

**public static final** class attr {

}

**public static final** class drawable {

**public static final** int androidmarker=0x7f020000;

**public static final** int h\_hotel=0x7f020001;

**public static final** int h\_memorial=0x7f020002;

**public static final** int h\_museum=0x7f020003;

**public static final** int h\_restaurant=0x7f020004;

**public static final** int h\_shelter=0x7f020005;

**public static final** int h\_theatre=0x7f020006;

**public static final** int h\_view\_point=0x7f020007;

**public static final** int icon=0x7f020008;

}

**public static final** class id {

**public static final** int ObjectAddress=0x7f07000b;

**public static final** int ObjectName=0x7f07000a;

**public static final** int address=0x7f070006;

**public static final** int button\_add=0x7f070007;

**public static final** int button\_addr=0x7f070011;

**public static final** int button\_coord=0x7f070012;

**public static final** int button\_exit=0x7f070002;

**public static final** int button\_info=0x7f070008;

**public static final** int button\_map=0x7f070000;

**public static final** int button\_name=0x7f070010;

**public static final** int button\_poi=0x7f070001;

**public static final** int button\_route=0x7f070009;

**public static final** int house=0x7f07000d;

**public static final** int latitude=0x7f07000e;

**public static final** int longitude=0x7f07000f;

**public static final** int mapView=0x7f070003;

**public static final** int name=0x7f070005;

**public static final** int root=0x7f070004;

**public static final** int search\_record=0x7f070013;

**public static final** int street=0x7f07000c;

}

**public static final** class layout {

**public static final** int main=0x7f030000;

**public static final** int map=0x7f030001;

**public static final** int object\_info=0x7f030002;

**public static final** int object\_menu=0x7f030003;

**public static final** int object\_view=0x7f030004;

**public static final** int objects=0x7f030005;

**public static final** int search=0x7f030006;

**public static final** int search\_address=0x7f030007;

**public static final** int search\_coord=0x7f030008;

**public static final** int search\_menu=0x7f030009;

**public static final** int search\_menu2=0x7f03000a;

}

**public static final** class menu {

**public static final** int main\_menu=0x7f060000;

}

**public static final** class string {

**public static final** int add=0x7f050005;

**public static final** int app\_name=0x7f050000;

**public static final** int menu\_info=0x7f050003;

**public static final** int no\_records=0x7f050007;

**public static final** int search=0x7f050009;

**public static final** int search\_hint=0x7f050008;

**public static final** int search\_name=0x7f050004;

**public static final** int text=0x7f050006;

**public static final** int title\_info=0x7f050002;

**public static final** int title\_view=0x7f050001;

}

**public static final** class xml {

**public static final** int searchable=0x7f040000;

}

}

**Файл main.xml**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<TableLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:orientation=*"vertical"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"fill\_parent"*

>

<Button

android:id=*"@+id/button\_map"*

android:text=*"Map"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

/>

<Button

android:id=*"@+id/button\_poi"*

android:text=*"POI"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

/>

<Button

android:id=*"@+id/button\_exit"*

android:text=*"Exit"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

/>

</TableLayout>

**Файл MainActivity.java**

**package** com.touristmap.activities;

**import** com.touristmap.R;

**import** android.app.Activity;

**import** android.app.AlertDialog;

**import** android.content.Context;

**import** android.content.DialogInterface;

**import** android.content.Intent;

**import** android.location.LocationManager;

**import** android.os.Bundle;

**import** android.provider.Settings;

**import** android.view.View;

**import** android.widget.Button;

**public class** MainActivity extends Activity {

/\*\* Called when the activity is first created. \*/

@Override

**public void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.main);

LocationManager locationManager = (LocationManager)getSystemService(Context.LOCATION\_SERVICE);

if (!locationManager.isProviderEnabled(LocationManager.GPS\_PROVIDER)) {

**final** AlertDialog.Builder builder = new AlertDialog.Builder(this);

builder.setMessage("GPS выключен")

.setCancelable(false)

.setPositiveButton("Настройки GPS", new DialogInterface.OnClickListener() {

**public void** onClick(**final** DialogInterface dialog, **final** int id) {

Intent intent = new Intent(Settings.ACTION\_LOCATION\_SOURCE\_SETTINGS);

startActivityForResult(intent, 1);

}

})

.setNegativeButton("Отмена", new DialogInterface.OnClickListener() {

**public void** onClick(**final** DialogInterface dialog, **final** int id) {

finish();

}

});

**final** AlertDialog alert = builder.create();

alert.show();

}

**final** Button button\_map = (Button) findViewById (R.id.button\_map);

button\_map.setOnClickListener(new View.OnClickListener() {

@Override

**public void** onClick(View v) {

Intent mapsActivity = new Intent(MainActivity.this,MapsActivity.class);

startActivity(mapsActivity);

}

});

**final** Button button\_settings = (Button) findViewById (R.id.button\_poi);

button\_settings.setOnClickListener(new View.OnClickListener() {

@Override

**public void** onClick(View v) {

Intent objectsEditorActivity = new Intent(MainActivity.this,ObjectsEditorActivity.class);

startActivity(objectsEditorActivity);

}

});

**final** Button button\_exit = (Button) findViewById (R.id.button\_exit);

button\_exit.setOnClickListener(new View.OnClickListener() {

@Override

**public void** onClick(View v) {

finish();

}

});

}

}

**Файл MapsActivity.java**

**package** com.touristmap.activities;

**import** android.net.Uri;

**import** android.os.Bundle;

**import** com.touristmap.DirectionPathOverlay;

**import** com.touristmap.HelloItemizedOverlay;

**import** com.touristmap.MyPositionOverlay;

**import** com.touristmap.ObjectDbHelper;

**import** com.touristmap.R;

**import** com.google.android.maps.GeoPoint;

**import** com.google.android.maps.MapActivity;

**import** com.google.android.maps.MapController;

**import** com.google.android.maps.MapView;

**import** com.google.android.maps.Overlay;

**import** com.google.android.maps.OverlayItem;

**import** com.touristmap.activities.MapsActivity;

**import** com.touristmap.providers.ArhitectureProvider;

**import** com.touristmap.providers.HostelProvider;

**import** com.touristmap.providers.InterestProvider;

**import** com.touristmap.providers.MuseamProvider;

**import** com.touristmap.providers.ObjectViewProvider;

**import** com.touristmap.providers.TemplesProvider;

**import** android.app.AlertDialog;

**import** android.app.Dialog;

**import** android.content.Context;

**import** android.content.DialogInterface;

**import** android.content.Intent;

**import** android.database.Cursor;

**import** android.graphics.drawable.Drawable;

**import** android.location.Address;

**import** android.location.Criteria;

**import** android.location.Geocoder;

**import** android.location.Location;

**import** android.location.LocationListener;

**import** android.location.LocationManager;

**import** android.view.LayoutInflater;

**import** android.view.Menu;

**import** android.view.MenuItem;

**import** android.view.MotionEvent;

**import** android.view.View;

**import** android.view.View.OnClickListener;

**import** android.widget.Button;

**import** android.widget.EditText;

**import** android.widget.Toast;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.OutputStreamWriter;

**import** java.util.\*;

**public class** MapsActivity extends MapActivity {

MapController mapController;

MyPositionOverlay positionOverlay;

**private** String[] street;

**private** int[] house;

**private** **double**[] lath;

**private** **double**[] lonh;

**private** int sizeSTR=-1;

**private** String streetIsc="ленина";

**private** int houseIsc = 47;

**private** **double** latIsc=0;

**private** **double** lonIsc=0;

**public static final** int IDM\_SETTINGS = 101;

**public static final** int IDM\_ADD\_OBJ = 102;

**public static final** int IDM\_ADD\_ROUTE = 103;

**public static final** int IDM\_PATH\_TEST = 104;

**public static final** int IDM\_SEARCH = 105;

**public static final** int IDM\_EXIT = 106;

**private** **final** static int IDD\_OBJECTS = 0;

**private** **final** static int IDD\_ADD\_TOURIST\_ROUTE = 1;

**private** **final** static int IDD\_COLLECT\_ROUTE=2;

**final** CharSequence[] mObjects = { "Theaters", "Museams", "Monuments", "Interest Objects", "Temples", "Hostels" };

**final** boolean[] mCheckedObjItems = { false, false, false, false, false, false };

**final** boolean[] mCheckedPathItems = { false, false, false, false, false, false };

**final** CharSequence[] mRouteObj = { "Памятник первому паравозу", "гостиница Свердловск", "Усадьба Расторгуева-Харитонова", "Храм-на-крови", "Отель Вознесенский", "Галерея Эгида", "Коляда-театр", "Площадь Советской Армии", "Театр Музыкальной Комедии", "Гостиница Исеть", "Памятник “Первая конная”" };

**final** boolean[] mCheckedRouteObjItems = { false, false, false, false, false, false, false, false, false, false, false};

**private** static **final** String[] mCoord = new String[] { ObjectDbHelper.\_ID,

ObjectDbHelper.NAME, ObjectDbHelper.LAT, ObjectDbHelper.LON };

**public static** String NLAT = "nlat";

**public static** String NLNG = "nlng";

List<Overlay> mapOverlays;

HelloItemizedOverlay itemizedOverlay;

MapView myMapView;

**double** lat;

**double** lng;

**private** **double**[] latiC;

**private** **double**[] longC;

**private** String[] nameC;

Drawable theatres;

Drawable museams;

Drawable monuments;

Drawable interestObjects;

Drawable temples;

Drawable hostels;

String path = "/mnt/sdcard/OutCoord.txt";

String coord = "/mnt/sdcard/coord.txt";

String in = "/mnt/sdcard/in.txt";

String out = "/mnt/sdcard/out.txt";

String out\_coord = "/mnt/sdcard/OutCoord.txt";

**private** int[][] A;

**private** int s, t;

**private** int size = -1;

**int**[] d;

**private** Stack<Integer> st;

**private** Stack<Integer> st1;

**private** int stackSize = 0;

**public** LinkedList<Integer> l = new LinkedList<Integer>();

**private** void ReadInTxt(String in) {

try {

BufferedReader ibr = new BufferedReader(new InputStreamReader(

new FileInputStream(in)));

String str;

Scanner scanner;

size = Integer.parseInt(ibr.readLine());

A = new int[size][size];

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

A[i][j] = 32767;

}

A[i][i] = 32767;

}

int ver;

for (int i = 0; i < size; i++) {

str = ibr.readLine();

scanner = new Scanner(str);

while ((ver = scanner.nextInt()) != 0) {

A[ver - 1][i] = scanner.nextInt();

}

}

t = Integer.parseInt(ibr.readLine());

ibr.close();

}

catch (IOException ae) {

}

}

// Считывание координат из файла

**public void** ReadCoord(String coord) {

try {

BufferedReader ibr = new BufferedReader(new InputStreamReader(

new FileInputStream(coord)));

String str;

Scanner scanner;

sizeC = Integer.parseInt(ibr.readLine());

latC = new **double**[sizeC];

lonC = new **double**[sizeC];

for (int i = 0; i < sizeC; i++) {

latC[i] = -1000;

lonC[i] = -1000;

}

for (int i = 0; i < sizeC; i++) {

str = ibr.readLine();

scanner = new Scanner(str);

scanner.useLocale(Locale.US);

scanner.useDelimiter(" ");

latC[i] = scanner.next**Double**();

lonC[i] = scanner.next**Double**();

}

ibr.close();

}

catch (IOException ae) {

}

}

**private** void WriteOutTxt(String out) {

try {

BufferedWriter obr = new BufferedWriter(new OutputStreamWriter(

new FileOutputStream(out), "UTF8"));

if (!st.empty()) {

obr.write("Y\r\n");

while (!st.empty()) {

obr.write(st.pop() + " ");

stackSize++;

}

obr.write("\r\n");

obr.write(Integer.toString(d[t - 1]));

} else {

obr.write("N");

}

obr.close();

} catch (IOException ae) {

}

}

**private** void WriteCoord(String out\_coord) {

int cr;

try {

BufferedWriter obr = new BufferedWriter(new OutputStreamWriter(

new FileOutputStream(out\_coord), "UTF8"));

if (!st1.empty()) {

obr.write(stackSize + "\r\n");

while (!st1.empty()) {

cr = st1.pop();

obr.write(latC[cr - 1] + " ");

obr.write(lonC[cr - 1] + "\r\n");

}

obr.write("\r\n");

}

obr.close();

} catch (IOException ae) {

}

}

**private** int min\_point(**double** lt, **double** lg) {

int num = -1;

**double** dist = -1, distM = 1000000;

Location locationB = new Location("point B");

locationB.setLatitude(lt \* 1E6);

locationB.setLongitude(lg \* 1E6);

for (int i = 0; i < sizeC; i++) {

Location locationA = new Location("point A");

locationA.setLatitude(latC[i] \* 1E6);

locationA.setLongitude(lonC[i] \* 1E6);

String str = locationA.distanceTo(locationB) / 1000 + "";

dist = **Double**.parse**Double**(str);

if (dist < distM) {

distM = dist;

num = i + 1;

}

}

**return** num;

}

**private** int MIN(TreeSet<Integer> F, int[] d) {

int min = F.first();

Iterator<Integer> itr = F.iterator();

int temp;

while (itr.hasNext()) {

temp = itr.next();

if (d[temp - 1] < d[min - 1]) {

min = temp;

}

}

**return** min;

}

**private** int MINIM(int a, int b) {

if (a <= b) {

**return** a;

} else {

**return** b;

}

}

**private** void DeikstraMin() {

int temp;

int w;

d = new int[size];

int[] prev = new int[size];

st = new Stack<Integer>();

st1 = new Stack<Integer>();

TreeSet<Integer> S = new TreeSet<Integer>();

TreeSet<Integer> T = new TreeSet<Integer>();

d[s - 1] = 32767;

S.add(s);

for (int i = 0; i < size; i++) {

T.add((i + 1));

}

T.remove(s);

Iterator<Integer> itr = T.iterator();

while (itr.hasNext()) {

temp = itr.next();

d[temp - 1] = A[s - 1][temp - 1];

prev[temp - 1] = s;

}

while (!S.contains(t)) {

w = this.MIN(T, d);

T.remove(w);

S.add(w);

Iterator<Integer> itr1 = T.iterator();

while (itr1.hasNext()) {

temp = itr1.next();

if (d[temp - 1] > MINIM(d[temp - 1], d[w - 1]

+ A[w - 1][temp - 1])) {

d[temp - 1] = MINIM(d[temp - 1], d[w - 1]

+ A[w - 1][temp - 1]);

prev[temp - 1] = w;

}

}

}

if (d[t - 1] > -32767) {

int v;

st.addElement(t);

st1.addElement(t);

v = t;

while (v != s) {

w = prev[v - 1];

st.addElement(w);

st1.addElement(w);

v = w;

}

}

}

/\*\* Called when the activity is first created. \*/

@Override

**public void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*map*);

myMapView = (MapView) findViewById(R.id.*mapView*);

mapController = myMapView.getController();

myMapView.setStreetView(**true**);

myMapView.setBuiltInZoomControls(**true**);

mapOverlays = myMapView.getOverlays();

mapController.setCenter(*START\_POINT*);

mapController.setZoom(12);

// Добавляем маркер своего местоположения

positionOverlay = **new** MyPositionOverlay();

List<Overlay> overlays = myMapView.getOverlays();

overlays.add(positionOverlay);

theatres = getResources().getDrawable(R.drawable.*h\_theatre*);

museams = getResources().getDrawable(R.drawable.*h\_museum*);

monuments = getResources().getDrawable(R.drawable.*h\_memorial*);

interestObjects = getResources().getDrawable(R.drawable.*h\_view\_point*);

temples = getResources().getDrawable(R.drawable.*h\_shelter*);

hostels = getResources().getDrawable(R.drawable.*h\_hotel*);

LocationManager locationManager;

String context = Context.*LOCATION\_SERVICE*;

locationManager = (LocationManager) getSystemService(context);

Criteria criteria = **new** Criteria();

criteria.setAccuracy(Criteria.*ACCURACY\_FINE*);

criteria.setAltitudeRequired(**false**);

criteria.setBearingRequired(**false**);

criteria.setCostAllowed(**true**);

criteria.setPowerRequirement(Criteria.*POWER\_LOW*);

String provider = locationManager.getBestProvider(criteria, **true**);

Location location = locationManager.getLastKnownLocation(provider);

updateWithNewLocation(location);

locationManager.requestLocationUpdates(provider, 1500, 3,

locationListener);

}

**private** **final** LocationListener locationListener = **new** LocationListener() {

**public void** onLocationChanged(Location location) {

updateWithNewLocation(location);

}

**public void** onProviderDisabled(String provider) {

updateWithNewLocation(**null**);

}

**public void** onProviderEnabled(String provider) {

}

**public void** onStatusChanged(String provider, **int** status, Bundle extras) {

}

};

**private** **int** sizeC;

**private** **double**[] latC;

**private** **double**[] lonC;

// Обновление местоположения

**private** **void** updateWithNewLocation(Location location) {

**if** (location != **null**) {

// Update my location marker

positionOverlay.setLocation(location);

// Update the map location.

**Double** geoLat = location.getLatitude() \* 1E6;

**Double** geoLng = location.getLongitude() \* 1E6;

GeoPoint point = **new** GeoPoint(geoLat.intValue(), geoLng.intValue());

mapController.animateTo(point);

}

}

@Override

**protected** **boolean** isRouteDisplayed() {

// **TODO** Auto-generated method stub

**return false;**

}

@Override

**public boolean** onCreateOptionsMenu(Menu menu) {

menu.add(Menu.*NONE*, *IDM\_SETTINGS*, Menu.*NONE*, "POI")

.setAlphabeticShortcut('p');

menu.add(Menu.*NONE*, *IDM\_ADD\_OBJ*, Menu.*NONE*, "Add objects")

.setAlphabeticShortcut('a');

menu.add(Menu.*NONE*, *IDM\_ADD\_ROUTE*, Menu.*NONE*, "Route")

.setAlphabeticShortcut('r');

menu.add(Menu.*NONE*, *IDM\_SEARCH*, Menu.*NONE*, "Search")

.setAlphabeticShortcut('s');

menu.add(Menu.*NONE*, *IDM\_EXIT*, Menu.*NONE*, "Exit").setAlphabeticShortcut(

'x');

**return** (**super**.onCreateOptionsMenu(menu));

}

@Override

**public boolean** onOptionsItemSelected(MenuItem item) {

**switch** (item.getItemId()) {

**case** *IDM\_SETTINGS*:

Intent objectsEditorActivity = **new** Intent(MapsActivity.**this**, ObjectsEditorActivity.**class**);

startActivity(objectsEditorActivity);

item.setChecked(**true**);

**break;**

**case** *IDM\_ADD\_ROUTE*:

LayoutInflater inflater\_route = LayoutInflater.*from*(**this**);

View root\_route = inflater\_route.inflate(R.layout.*search\_menu*, **null**);

**final** Button button\_collect = (Button) root\_route.findViewById(R.id.*button\_addr*);

**final** Button button\_add = (Button) root\_route.findViewById(R.id.*button\_coord*);

**final** Button button\_construct = (Button) root\_route.findViewById(R.id.*button\_name*);

button\_construct.setText("Construct route");

button\_collect.setText("Collect route");

button\_add.setText("Add tourist route");

AlertDialog.Builder menu\_route = **new** AlertDialog.Builder(**this**);

menu\_route.setView(root\_route);

menu\_route.show();

button\_construct.setOnClickListener(**new** OnClickListener() {

@Override

**public void** onClick(View v) {

ConstructRouteDialog();

};

});

button\_collect.setOnClickListener(**new** OnClickListener() {

@Override

**public void** onClick(View v) {

showDialog(*IDD\_COLLECT\_ROUTE*);

};

});

button\_add.setOnClickListener(**new** OnClickListener() {

@Override

**public void** onClick(View v) {

showDialog(*IDD\_ADD\_TOURIST\_ROUTE*);

};

});

item.setChecked(**true**);

**break;**

**case** *IDM\_PATH\_TEST*:

ReadCoord(path);

GeoPoint gp1 = **new** GeoPoint((**int**) (latC[0] \* 1E6),

(**int**) (lonC[0] \* 1E6));

GeoPoint gp2;

// Строим маршрут по файлу с координатами Monuments\_coord

**for** (**int** i = 0; i < sizeC; i++) {

gp2 = **new** GeoPoint((**int**) (latC[i] \* 1E6), (**int**) (lonC[i] \* 1E6));

// отрезки соединяющие точки

myMapView.getOverlays().add(**new** DirectionPathOverlay(gp1, gp2));

gp1 = gp2;

}

item.setChecked(**true**);

**break;**

**default:**

**return false;**

}

**return true;**

}

// Меню с чекбоксами для добавления групп объектов на карту

@Override

**protected** Dialog onCreateDialog(**int** id) {

**switch** (id) {

**case** *IDD\_ADD\_TOURIST\_ROUTE*:

AlertDialog.Builder builderPath = **new** lertDialog.Builder(**this**);

builderPath.setTitle("Add tourist route");

builderPath.setMultiChoiceItems(mObjects, mCheckedPathItems, **new** DialogInterface.OnMultiChoiceClickListener() {

@Override

**public void** onClick(DialogInterface dialog, **int** which, **boolean** isChecked) {

mCheckedPathItems[which] = isChecked;

}

});

builderPath.setPositiveButton("OK",

**new** DialogInterface.OnClickListener() {

**public void** onClick(DialogInterface dialog, **int** id) {

**if** (!mapOverlays.isEmpty()) {

mapOverlays.clear();

}

**if** (mCheckedPathItems[0]){

addGroup(ObjectViewProvider.*CONTENT\_URI*, theatres);

}

**if** (mCheckedPathItems[1]) {

addGroup(MuseamProvider.*CONTENT\_URI*, museams);

}

**if** (mCheckedPathItems[2]) {

addGroup(ArhitectureProvider.*CONTENT\_URI*, monuments);

ReadCoord("/mnt/sdcard/coord\_monuments.txt");

GeoPoint gp1 = **new** GeoPoint( (**int**) (latC[0] \* 1E6), (**int**) (lonC[0] \* 1E6));

GeoPoint gp2;

// Строим маршрут по файлу с координатами Monuments\_coord

**for** (**int** i = 0; i < sizeC; i++) {

gp2 = **new** GeoPoint((**int**) (latC[i] \* 1E6), (**int**) (lonC[i] \* 1E6));

// отрезки соединяющие точки

myMapView.getOverlays().add(**new** DirectionPathOverlay(gp1, gp2));

gp1 = gp2;

}

}

**if** (mCheckedPathItems[3]) {

addGroup(InterestProvider.*CONTENT\_URI*, interestObjects);

}

**if** (mCheckedPathItems[4]) {

addGroup(TemplesProvider.*CONTENT\_URI*, temples);

}

**if** (mCheckedPathItems[5]) {

addGroup(HostelProvider.*CONTENT\_URI*, hostels);

}

}

});

builderPath.setNegativeButton("Cancel",

**new** DialogInterface.OnClickListener() {

**public void** onClick(DialogInterface dialog, **int** id) {

dialog.cancel();

Toast.*makeText*(getApplicationContext(), "Dialog cancel", Toast.*LENGTH\_SHORT*).show();

}

});

builderPath.setCancelable(**false**);

**return** builderPath.create();

//Диалог выбора объектов для маршрута

**case** *IDD\_COLLECT\_ROUTE*:

AlertDialog.Builder builderCollectRoute = **new** AlertDialog.Builder(**this**);

builderCollectRoute.setTitle("Collect tourist tourist route");

builderCollectRoute.setMultiChoiceItems(mRouteObj, mCheckedRouteObjItems, **new** DialogInterface.OnMultiChoiceClickListener() {

@Override

**public** **void** onClick(DialogInterface dialog, **int** which, **boolean** isChecked) {

mCheckedRouteObjItems[which] = isChecked;

}

});

builderCollectRoute.setPositiveButton("OK",

**new** DialogInterface.OnClickListener() {

**public** **void** onClick(DialogInterface dialog, **int** id) {

checkObj = **new** **int** [11];

**if** (!mapOverlays.isEmpty()) {

mapOverlays.clear();

}

**for** (**int** i=0; i<11; i++){

checkObj[i]=0;

}

**if** (mCheckedRouteObjItems[0]) {

checkObj[0]=1;

}

**else** checkObj[0]=0;

**if** (mCheckedRouteObjItems[1]) {

checkObj[1]=1;

}

**else** checkObj[1]=0;

**if** (mCheckedRouteObjItems[2]) {

checkObj[2]=1;

}

**else** checkObj[2]=0;

**if** (mCheckedRouteObjItems[3]) {

checkObj[3]=1;

}

**else** checkObj[3]=0;

**if** (mCheckedRouteObjItems[4]) {

checkObj[4]=1;

}

**else** checkObj[4]=0;

**if** (mCheckedRouteObjItems[5]) {

checkObj[5]=1;

}

**else** checkObj[5]=0;

**if** (mCheckedRouteObjItems[6]) {

checkObj[6]=1;

}

**else** checkObj[6]=0;

**if** (mCheckedRouteObjItems[7]) {

checkObj[7]=1;

}

**else** checkObj[7]=0;

**if** (mCheckedRouteObjItems[8]) {

checkObj[8]=1;

}

**else** checkObj[8]=0;

**if** (mCheckedRouteObjItems[9]) {

checkObj[9]=1;

}

**else** checkObj[9]=0;

**if** (mCheckedRouteObjItems[10]) {

checkObj[10]=1;

}

**else** checkObj[10]=0;

temp = "";

countCheck=0;

**for** (**int** i=0; i<11; i++){

**if** (checkObj[i]==1) {

temp=temp+ i + " ";

countCheck=countCheck+1;

}

}

activeObj=**new** **int** [countCheck]; //массив номеров (от 0 до 10) выбранных объектов

Scanner scanner = **new** Scanner(temp);

**int** j=0;

**while** (scanner.hasNextInt()) {

activeObj[j]=scanner.nextInt();

j++;

}

//считывается матрица весов графа всех объектов в области

ReadInputTSP();

NewRead(activeObj); //выделяется подграф содержащий выбранные объекты

run(); //запуск коммивояжера

WriteOutputTSP(); //запись минимального пути в файл outputTSP.txt

ReadCoordRoute(TSP\_coord); //считывание координат ближайших перекрестков

ReadCoordObj(TSP\_object\_coord); //считывание координат объектов

ReadInTxt(in); //считывание графа улиц

ReadCoord(coord);

**try** {

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter( **new** FileOutputStream(TSP\_number\_ver), "UTF8"));

obr.write("");

obr.close();

} **catch** (IOException ae) {}

**try** {

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(TSP\_outcoord), "UTF8"));

obr.write("");

obr.close();

} **catch** (IOException ae) {}

itemizedOverlay = **new** HelloItemizedOverlay(getApplicationContext(), getResources().getDrawable(R.drawable.*marker*));

**int** turg4=19;

**try** {

//запись в файл TSP\_turg4.txt координат пути от Тургенева 4 до первой точки

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(TSP\_outcoord, **true**), "UTF8"));

obr.write("56.841109 60.61522");

obr.write("\r\n");

obr.close();

} **catch** (IOException ae) {}

Deikstra1(turg4,numberPerekr[iscpoint[0]]);

WriteCoordTSP(TSP\_outcoord);

**if** (!mapOverlays.isEmpty()) {

mapOverlays.clear();

}

isAddLocationPoint(56.841109, 60.61522, "");

itemizedOverlay = **new** HelloItemizedOverlay(getApplicationContext(), getResources().getDrawable(R.drawable.*marker2*));

**for** (**int** i=0; i<sizeVer-1;i++){

**try** {

//запись в файл TSP\_number\_ver.txt i-ого выбранного объкета(из построенного пути в процедуре ран)

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(TSP\_number\_ver, **true**), "UTF8"));

obr.write(iscpoint[i]);

obr.close();

} **catch** (IOException ae) {}

**try** {

//запись в файл TSP\_outcoord.txt координат i-ого выбранного объкета(из построенного пути в процедуре ран)

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(TSP\_outcoord, **true**), "UTF8"));

obr.write(latO[iscpoint[i]]+" ");

obr.write(longO[iscpoint[i]] + "\r\n");

obr.close();

} **catch** (IOException ae) {}

isAddLocationPoint(latO[iscpoint[i]], longO[iscpoint[i]], "");

Deikstra1(numberPerekr[iscpoint[i]], numberPerekr[iscpoint[i+1]]);

WriteCoordTSP(TSP\_outcoord);

isAddLocationPoint(latO[iscpoint[sizeVer-1]], longO[iscpoint[sizeVer-1]], "");

**try** {

//запись в файл TSP\_number\_ver.txt номера последнего объекта в маршруте

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(TSP\_number\_ver, **true**), "UTF8"));

obr.write(iscpoint[sizeVer-1]);

obr.close();

} **catch** (IOException ae) {}

**try** {

//запись в файл TSP\_outcoord.txt координат последнего объекта в маршруте

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter(**new** FileOutputStream(TSP\_outcoord, **true**), "UTF8"));

obr.write(latO[iscpoint[sizeVer-1]]+" ");

obr.write(longO[iscpoint[sizeVer-1]] + "\r\n");

obr.close();

} **catch** (IOException ae) {}

**try** {

BufferedReader ibr = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(TSP\_outcoord)));

countStr=0;

**while** ((ibr.readLine())!=**null**){

countStr=countStr+1;

}

ibr.close();

} **catch** (IOException ae) {}

ReadCoordDeikstraCoord(TSP\_outcoord);

GeoPoint gp1 = **new** GeoPoint((**int**) (latRD[0] \* 1E6), (**int**) (lonRD[0] \* 1E6));

GeoPoint gp2;

//Строим маршрут по файлу с координатами TSP\_outcoord

**for**(**int** i = 0; i<countStr; i++){

gp2 = **new** GeoPoint((**int**) (latRD[i] \* 1E6),(**int**) (lonRD[i] \* 1E6));

//отрезки соединяющие точки

myMapView.getOverlays().add(**new** DirectionPathOverlay(gp1, gp2));

gp1=gp2;

}

}

});

builderCollectRoute.setNegativeButton("Cancel", **new** DialogInterface.OnClickListener() {

**public** **void** onClick(DialogInterface dialog, **int** id) {

dialog.cancel();

Toast.*makeText*(getApplicationContext(), "Dialog cancel", Toast.*LENGTH\_SHORT*).show();

}

});

builderCollectRoute.setCancelable(**false**);

**return** builderCollectRoute.create();

**default**:

**return** **null**;

}

}

// Процедура Добавления группы объектов на карту

**private** **void** addGroup(Uri uri, Drawable drawable) {

Cursor cursor = managedQuery(uri, *mCoord*, "\_ID", **null**, **null**);

**int** count = cursor.getCount();

latiC = **new** **double**[count];

longC = **new** **double**[count];

nameC = **new** String[count];

**for** (**int** i = 0; i < count; i++) {

latiC[i] = -1;

longC[i] = -1;

nameC[i] = "";

}

**for** (**int** i = 0; i < count; i++) {

cursor.moveToPosition(i);

lat = **Double**.*parse****Double***(cursor.getString(2));

lng = **Double**.*parse****Double***(cursor.getString(3));

latiC[i] = lat;

longC[i] = lng;

nameC[i] = cursor.getString(1);

}

**double** nl, nlg;

itemizedOverlay = **new** HelloItemizedOverlay(**this**, drawable);

**try** {

**for** (**int** i = 0; i < count; i++) {

nl = latiC[i];

nlg = longC[i];

String name = nameC[i];

isAddLocationPoint(nl, nlg, name);

}

} **catch** (NullPointerException e) {

e.printStackTrace();

}

}

// Процедура добавления точки на карту

**public void** isAddLocationPoint(**double** netlat, **double** netlng, String name) {

GeoPoint point = **new** GeoPoint((**int**) (netlat \* 1E6),

(**int**) (netlng \* 1E6));

String coord = "Широта: " + netlat + "\nДолгота: " + netlng;

// Создаем точку и текст появляющийся при нажатии

OverlayItem overlayitem = **new** OverlayItem(point, name, coord);

// Положение карты при запуске в точке point

mapController.animateTo(point);

mapController.setZoom(13);

// Добавляем точку на карту

itemizedOverlay.addOverlay(overlayitem);

mapOverlays.add(itemizedOverlay);

}

**private** **void** ConstructRouteDialog() {

LayoutInflater inflater = LayoutInflater.*from*(**this**);

View root = inflater.inflate(R.layout.*search\_address*, **null**);

**final** EditText textStreet = (EditText)root.findViewById(R.id.*street*);

**final** EditText textHouse = (EditText)root.findViewById(R.id.*house*);

AlertDialog.Builder b = **new** AlertDialog.Builder(**this**);

b.setView(root);

b.setTitle("Construct route");

b.setPositiveButton(

"Ок", **new** DialogInterface.OnClickListener() {

**public** **void** onClick(DialogInterface dialog, **int** whichButton) {

String streetR = textStreet.getText().toString();

**int** houseR = Integer.*parseInt*(textHouse.getText().toString());

**if** (!mapOverlays.isEmpty()) {

mapOverlays.clear();

}

ReadInTxt(in);

ReadCoord(coord);

**int** ptnum = -1;

ptnum = min\_point(56.841109, 60.61522);

Toast.*makeText*(getBaseContext(), " ptnum= " + ptnum, Toast.*LENGTH\_LONG*).show();

ReadInTxtSearch();

searchadress(streetR, houseR);

**int** ptnum1 = -1;

ptnum1=min\_point(latIsc,lonIsc);

Toast.*makeText*(getBaseContext(), " ptnum1= " + ptnum1, Toast.*LENGTH\_LONG*).show();

itemizedOverlay = **new** HelloItemizedOverlay(getApplicationContext(), getResources().getDrawable(R.drawable.*marker*));

isAddLocationPoint(56.841109, 60.61522, "");

itemizedOverlay = **new** HelloItemizedOverlay(getApplicationContext(), getResources().getDrawable(R.drawable.*marker2*));

isAddLocationPoint(latIsc, lonIsc, "");

Deikstra1(ptnum,ptnum1);

**try** {

//запись в файл constr\_route.txt координат Тургенева 4

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter(

**new** FileOutputStream(constr\_route), "UTF8"));

obr.write("56.841109 60.61522");

obr.write("\r\n");

obr.close();

} **catch** (IOException ae) {}

WriteCoordTSP(constr\_route);

**try** {

//запись в файл constr\_route.txt координат конечной точки

BufferedWriter obr = **new** BufferedWriter(**new** OutputStreamWriter(

**new** FileOutputStream(constr\_route, **true**), "UTF8"));

obr.write(latIsc + " " + lonIsc);

obr.close();

} **catch** (IOException ae) {}

**try** {

BufferedReader ibr = **new** BufferedReader(**new** InputStreamReader(

**new** FileInputStream(constr\_route)));

countStr=0;

**while** ((ibr.readLine())!=**null**){

countStr=countStr+1;

}

ibr.close();

} **catch** (IOException ae) {}

ReadCoordDeikstraCoord(constr\_route);

GeoPoint gp1 = **new** GeoPoint((**int**) (latRD[0] \* 1E6),

(**int**) (lonRD[0] \* 1E6));

GeoPoint gp2;

//Строим маршрут по файлу с координатами Monuments\_coord

**for**(**int** i = 0; i<countStr; i++)

{

gp2 = **new** GeoPoint((**int**) (latRD[i] \* 1E6),(**int**) (lonRD[i] \* 1E6));

//отрезки соединяющие точки

myMapView.getOverlays().add(**new** DirectionPathOverlay(gp1, gp2));

gp1=gp2;

}

}

});

b.setNegativeButton(

"Cancel", **new** DialogInterface.OnClickListener() {

**public** **void** onClick(DialogInterface dialog, **int** whichButton) {}

});

b.show();

}

**Файл ObjectsEditorActivity.java**

**package** com.touristmap.activities;

**import** com.touristmap.R;

**import** android.os.Bundle;

**import** android.app.ListActivity;

**import** android.content.Intent;

**import** android.view.View;

**import** android.widget.ArrayAdapter;

**import** android.widget.ListView;

**public class** ObjectsEditorActivity extends ListActivity {

/\*\* Called when the activity is first created. \*/

**private** int mId = -1;

String[] mObjectsClass = { "Theaters", "Museams", "Monuments",

"Interest Objects", "Temples", "Hostels" };

@Override

**public void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.objects);

setListAdapter(new ArrayAdapter<String>(this,

android.R.layout.simple\_list\_item\_1, mObjectsClass));

}

**public void** onListItemClick(ListView parent, View v, int position, long id) {

mId = position;

**switch** (mId) {

**case** 0: {

Intent intent = new Intent();

intent.setClass(this, TheatersViewActivity.class);

startActivity(intent);

}

**break;**

**case** 1: {

Intent intent = new Intent();

intent.setClass(this, MuseamViewActivity.class);

startActivity(intent);

}

**break;**

**case** 2: {

Intent intent = new Intent();

intent.setClass(this, ArhitectureViewActivity.class);

startActivity(intent);

}

**break;**

**case** 3: {

Intent intent = new Intent();

intent.setClass(this, InterestViewActivity.class);

startActivity(intent);

}

**break;**

**case** 4: {

Intent intent = new Intent();

intent.setClass(this, TemplesViewActivity.class);

startActivity(intent);

}

**break;**

**case** 5: {

Intent intent = new Intent();

intent.setClass(this, HostelViewActivity.class);

startActivity(intent);

}

**break;**

}

}

}

**Файл ArhitectureViewActivity.java**

**package** com.touristmap.activities;

//памятники

**import** com.touristmap.ArhitectureDbHelper;

**import** com.touristmap.ObjectDbHelper;

**import** com.touristmap.R;

**import** com.touristmap.providers.ArhitectureProvider;

**import** android.os.Bundle;

**import** android.app.AlertDialog;

**import** android.app.ListActivity;

**import** android.content.Intent;

**import** android.database.Cursor;

**import** android.view.LayoutInflater;

**import** android.view.Menu;

**import** android.view.MenuItem;

**import** android.view.View;

**import** android.view.View.OnClickListener;

**import** android.widget.Button;

**import** android.widget.ListView;

**import** android.widget.TextView;

**import** android.widget.ListAdapter;

**import** android.widget.SimpleCursorAdapter;

**import** android.widget.Toast;

**public class** ArhitectureViewActivity **extends** ListActivity {

**private** Cursor mCursor;

**private** ListAdapter mAdapter;

**private** String netlat;

**private** String netlng;

**public static final** **int** *IDM\_PATH* = 101;

**private** **static** **final** String[] *mContent* = **new** String[] {

ArhitectureDbHelper.*\_ID*, ObjectDbHelper.*NAME*,

ArhitectureDbHelper.*ADDRESS* };

**private** **static** **final** String[] *displayContent* = **new** String[] {

ObjectDbHelper.*NAME*, ObjectDbHelper.*ADDRESS* };

**int**[] displayViews = **new** **int**[] { R.id.*ObjectName*, R.id.*ObjectAddress* };

**private** **int** infId;

**private** **static** **final** String[] *mCoord* = **new** String[] { ObjectDbHelper.*\_ID*,

ObjectDbHelper.*LAT*, ObjectDbHelper.*LON* };

@Override

**public void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

mCursor = managedQuery(ArhitectureProvider.*CONTENT\_URI*, *mContent*, **null**,

**null**, **null**);

mAdapter = **new** SimpleCursorAdapter(**this**, R.layout.*object\_view*, mCursor,

*displayContent*, displayViews);

setListAdapter(mAdapter);

}

**public void** onListItemClick(ListView parent, View v, **int** position, **long** id) {

infId = position;

Cursor cursor = managedQuery(ArhitectureProvider.*CONTENT\_URI*, *mCoord*,

"\_ID", **null**, **null**);

cursor.moveToPosition(position);

String text = "Широта: " + cursor.getString(1)+ "\nДолгота: " +

cursor.getString(2);

Toast.*makeText*(**this**, text, Toast.*LENGTH\_SHORT*).show();

netlat = cursor.getString(1);

netlng = cursor.getString(2);

LayoutInflater inflater = LayoutInflater.*from*(**this**);

View root = inflater.inflate(R.layout.*object\_menu*, **null**);

**final** Button button\_add = (Button) root.findViewById(R.id.*button\_add*);

**final** Button button\_info = (Button) root.findViewById(R.id.*button\_info*);

**final** Button button\_route = (Button) root.findViewById(R.id.*button\_route*);

button\_add.setText("Add on Map");

button\_info.setText("Information");

button\_route.setText("Construct a route");

AlertDialog.Builder menu = **new** AlertDialog.Builder(**this**);

menu.setView(root);

menu.show();

button\_info.setOnClickListener(**new** OnClickListener() {

@Override

**public void** onClick(View v) {

CallObjectInfoDialog();

};

});

button\_add.setOnClickListener(**new** OnClickListener() {

@Override

**public void** onClick(View v) {

CallViewOnMap();

};

});

button\_route.setOnClickListener(**new** OnClickListener() {

@Override

**public void** onClick(View v) {

};

});

}

**private** **void** CallViewOnMap(){

Intent intent = **new** Intent();

intent.setClass(**this**, ViewObjectOnMapActivity.**class**);

intent.putExtra(ViewObjectOnMapActivity.*NLAT*, netlat);

intent.putExtra(ViewObjectOnMapActivity.*NLNG*, netlng);

startActivity(intent);

finish();

}

**private** **void** CallObjectInfoDialog() {

LayoutInflater inflater = LayoutInflater.*from*(**this**);

View root = inflater.inflate(R.layout.*object\_info*, **null**);

**final** TextView textName = (TextView) root.findViewById(R.id.*name*);

**final** TextView textPhone = (TextView) root.findViewById(R.id.*address*);

mCursor.moveToPosition(infId);

textName.setText(mCursor.getString(1));

textPhone.setText(mCursor.getString(2));

AlertDialog.Builder b = **new** AlertDialog.Builder(**this**);

b.setView(root);

b.show();

}

@Override

**public boolean** onCreateOptionsMenu(Menu menu) {

menu.add(Menu.*NONE*, *IDM\_PATH*, Menu.*NONE*, "Construct a route")

.setAlphabeticShortcut('p');

**return** (**super**.onCreateOptionsMenu(menu));

}

@Override

**public boolean** onOptionsItemSelected(MenuItem item) {

**switch** (item.getItemId()) {

**case** *IDM\_PATH*:

**break;**

**default:**

**return false;**

}

**return true;**

}

}

**Файл ArhitectureProvider.java**

**package** com.touristmap.providers;

//памятники

**import** java.util.HashMap;

**import** com.touristmap.ArhitectureDbHelper;

**import** android.content.ContentProvider;

**import** android.content.ContentUris;

**import** android.content.ContentValues;

**import** android.content.UriMatcher;

**import** android.database.Cursor;

**import** android.database.SQLException;

**import** android.database.sqlite.SQLiteDatabase;

**import** android.net.Uri;

**import** android.text.TextUtils;

**public class** ArhitectureProvider extends ContentProvider {

**public static final** String DB\_ARHITECT = "arhitect.db";

**public static final** Uri CONTENT\_URI = Uri.parse("content://com.touristmap.ArhitectureProvider/arhitect");

**public static final** **int** URI\_CODE = 1;

**public static final** **int** URI\_CODE\_ID = 2;

**private static final** UriMatcher mUriMatcher;

**private static** HashMap<String, String> mObjectMap;

**private** SQLiteDatabase db;

**static** {

mUriMatcher = new UriMatcher(UriMatcher.NO\_MATCH);

mUriMatcher.addURI("com.touristmap.ArhitectureProvider",

ArhitectureDbHelper.TABLE\_NAME, URI\_CODE);

mUriMatcher.addURI("com.touristmap.ArhitectureProvider",

ArhitectureDbHelper.TABLE\_NAME + "/#", URI\_CODE\_ID);

mObjectMap = new HashMap<String, String>();

mObjectMap.put(ArhitectureDbHelper.\_ID, ArhitectureDbHelper.\_ID);

mObjectMap.put(ArhitectureDbHelper.NAME, ArhitectureDbHelper.NAME);

mObjectMap.put(ArhitectureDbHelper.ADDRESS, ArhitectureDbHelper.ADDRESS);

mObjectMap.put(ArhitectureDbHelper.LAT, ArhitectureDbHelper.LAT);

mObjectMap.put(ArhitectureDbHelper.LON, ArhitectureDbHelper.LON);

}

**public** StringgetDbName() {

**return**(DB\_ARHITECT);

}

@Override

**public boolean** onCreate() {

db = (new ArhitectureDbHelper(getContext())).getWritableDatabase();

**return** (db == null) ? false : true;

}

@Override

**public** Cursor query(Uri url, String[] projection,

String selection, String[] selectionArgs, String sort) {

String orderBy;

if (TextUtils.isEmpty(sort)) {

orderBy = ArhitectureDbHelper.NAME;

}

else {

orderBy = sort;

}

Cursor c = db.query(ArhitectureDbHelper.TABLE\_NAME, projection, selection, selectionArgs,

null, null, orderBy);

c.setNotificationUri(getContext().getContentResolver(), url);

**return** c;

}

@Override

**public** Uri insert(Uri url, ContentValues inValues) {

ContentValues values = new ContentValues(inValues);

long rowId = db.insert(ArhitectureDbHelper.TABLE\_NAME, ArhitectureDbHelper.NAME, values);

if (rowId > 0) {

Uri uri = ContentUris.withAppendedId(CONTENT\_URI, rowId);

getContext().getContentResolver().notifyChange(uri, null);

**return** uri;

}

else {

throw new SQLException("Failed to insert row into " + url);

}

}

@Override

**public int** delete(Uri url, String where, String[] whereArgs) {

int retVal = db.delete(ArhitectureDbHelper.TABLE\_NAME, where, whereArgs);

getContext().getContentResolver().notifyChange(url, null);

**return** retVal;

}

@Override

**public int** update(Uri url, ContentValues values,

String where, String[] whereArgs) {

int retVal = db.update(ArhitectureDbHelper.TABLE\_NAME, values, where, whereArgs);

getContext().getContentResolver().notifyChange(url, null);

**return** retVal;

}

@Override

**public String** getType(Uri uri) {

**return** null;

}

}

**Файл ArhitectureDbHelper.java**

**package** com.touristmap;

//памятники

**import** com.touristmap.providers.ArhitectureProvider;

**import** android.content.ContentValues;

**import** android.content.Context;

**import** android.database.sqlite.SQLiteDatabase;

**import** android.database.sqlite.SQLiteOpenHelper;

**import** android.provider.BaseColumns;

**public class** ArhitectureDbHelper **extends** SQLiteOpenHelper **implements** BaseColumns{

**public static final** String *TABLE\_NAME*="object";

**public static final** String *NAME*="name";

**public static final** String *ADDRESS*="address";

**public static final** String *LAT*="latitude";

**public static final** String *LON*="longitude";

//**private** **final** Context myContext;

**public** ArhitectureDbHelper(Context context){

**super**(context, ArhitectureProvider.*DB\_ARHITECT*, **null**, 1);

//this.myContext = context;

}

@Override

**public void** onCreate(SQLiteDatabase db){

db.execSQL("CREATE TABLE " + *TABLE\_NAME* + " (\_id INTEGER PRIMARY KEY AUTOINCREMENT, "+ *NAME* + " TEXT, " + *ADDRESS* + " TEXT, " + *LAT* + " TEXT, " + *LON* + " TEXT);");

ContentValues values = **new** ContentValues();

values.put(*NAME*, "Памятник Клавиатуре");

values.put(*ADDRESS*, "городская набережная");

values.put(*LAT*, "56.832783 ");

values.put(*LON*, "60.607516 ");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Памятник Человеку-невидимке, герою Герберта Уэллса ");

values.put(*ADDRESS*, "Белинского, 15 ");

values.put(*LAT*, "56.834242 ");

values.put(*LON*, "60.614977 ");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Памятник сантехнику Афоне ");

values.put(*ADDRESS*, "Сибирский тракт, 2");

values.put(*LAT*, "56.825639");

values.put(*LON*, "60.63957");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Площадь Советской Армии");

values.put(*ADDRESS*, "перекресток Первомайская - Луначарского");

values.put(*LAT*, "56.843394");

values.put(*LON*, "60.617973");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Памятник Первому паровозу");

values.put(*ADDRESS*, "перекресток улиц Свердлова и Челюскинцев");

values.put(*LAT*, "56.856006");

values.put(*LON*, "60.603274");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Памятник Седой Урал");

values.put(*ADDRESS*, "площадь Обороны");

values.put(*LAT*, "56.82328");

values.put(*LON*, "60.629095");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Комплекс памяти Черный тюльпан");

values.put(*ADDRESS*, "перекресток Первомайская - Луначарского ");

values.put(*LAT*, "56.843312");

values.put(*LON*, "60.617488");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Памятник “Первая конная”");

values.put(*ADDRESS*, "Ленина 71");

values.put(*LAT*, "56.841113");

values.put(*LON*, "60.624379");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Памятник Владимиру Высоцкому и Марине Влади");

values.put(*ADDRESS*, "Красноармейская, 10");

values.put(*LAT*, "56.837179");

values.put(*LON*, "60.615822");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Памятник гитарному грифу");

values.put(*ADDRESS*, " Добролюбова, 19");

values.put(*LAT*, "56.830842");

values.put(*LON*, "60.604751");

db.insert(*TABLE\_NAME*, *NAME*, values);

values.put(*NAME*, "Памятник основателям Екатеринбурга");

values.put(*ADDRESS*, "Ленина");

values.put(*LAT*, "56.838123");

values.put(*LON*, "60.605781");

db.insert(*TABLE\_NAME*, *NAME*, values);

}

@Override

**public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion){

db.execSQL("DROP TABLE IF EXISTS " + *TABLE\_NAME*);

onCreate(db);

}

}