**Использование возможностей смартфона в приложениях**

**9.1 Введение**

В этом курсе проделана уже немалая работа: установлена и настроена *среда разработки*, созданы первые приложения, хочется двигаться дальше. Разработка мобильных приложений под *Android* имеет ряд особенностей, часть из них мы уже рассмотрели, часть ожидают рассмотрения в ближайших темах. Данная тема, как видно из названия, посвящена возможностям смартфонов и их использованию в приложениях.

Особенностью большинства мобильных устройств является наличие сенсорного экрана и возможность управления пальцем (touch-interface), очевидно, что это необходимо учитывать и использовать при разработке приложений. *Смартфон*, если уж появляется у человека, становится его спутником всегда и везде, в связи с этим, довольно часто используется, как фотоаппарат или проигрыватель музыки, а также смартфоны все чаще становятся инструментами ориентирования на местности.

В данной теме предполагается рассмотрение вопросов разработки приложений, ориентированных на тач-*интерфейс*, работу со звуком, использование камеры и глобальных систем позиционирования.

**9.2 Отличительные особенности смартфонов**

Пришло время поговорить о наиболее интересных возможностях смартфонов, которые можно использовать в приложениях. Ни для кого не секрет1, что *смартфон* является "умным телефоном": предполагает обязательное наличие операционной системы и возможность установки дополнительных приложений, существенно расширяющих функционал устройства. С одной стороны, *смартфон* выполняет все привычные функции мобильного телефона и, благодаря компактным размерам, всегда под рукой. С другой стороны, благодаря наличию процессора и операционной системы, позволяет выполнять многие функции полноценного компьютера. Дополнительно ко всему, смартфоны обладают рядом интересных особенностей, не характерных для телефонов и компьютеров.

Для начала обратим внимание на экран смартфона. В современных смартфонах экран занимает практически всю *площадь* передней панели устройства, имеет высокое разрешение и является чувствительным к прикосновениям. Благодаря такой чувствительности, для взаимодействия с устройством и его приложениями можно использовать виртуальные *элементы управления*, чаще всего кнопки, отображаемые на экране. В связи с чем отпадает необходимость в физических кнопках. В смартфонах реализуется, так называемый, touch-*интерфейс* - *интерфейс*, основанный на виртуальных элементах управления, выбор которых выполняется простым касанием, а также на использовании жестов (gestures). Если точек касания несколько (т. е. используется несколько пальцев), такой *интерфейс*, уже называется multi-touch.

Еще одна особенность смартфонов состоит в том, что для большинства их владельцев не последнюю роль играет возможность использования этого "умного телефона" в качестве аудио или видеоплеера, поэтому современные устройства становятся все более и более мультимедийными. В первой лекции обсуждалось, что в состав платформы *Android* входит набор библиотек для обработки *мультимедиа* *Media* Framework, в котором реализована *поддержка* большинства общих медиа-форматов. В связи с чем, в приложения, разрабатываемые для смартфонов под управлением *Android*, можно интегрировать *запись* и воспроизведение аудио и видео, а также работу с изображениями.

Важной и часто используемой особенностью смартфонов является наличие камеры, которая позволяет снимать все самое интересное: от первых шагов ребенка до падения метеорита. Телефон всегда под рукой и готов к работе, в связи с этим количество фотографий и небольших видеороликов резко увеличилось, и любое интересное событие в жизни индивидуума может быть запечатлено и сохранено для потомков. С ростом возможностей получения фото и видео материалов увеличивается потребность в приложениях, способных работать с этими материалами. Платформа *Android* позволяет разрабатывать такие приложения, которые предоставляют пользователям возможности делать фотоснимки или записывать видео, каким-то образом обрабатывать полученные материалы и использовать их далее.

Большинство смартфонов оснащены *GPS*-модулем, а некоторые даже комбинированным модулем *GPS*/ГЛОНАСС, что позволяет использовать такое устройство в качестве инструмента для ориентирования на местности. Во многих случаях *смартфон* с установленным соответствующим программным обеспечением вполне может заменить *GPS* навигатор. В разрабатываемых приложениях иногда бывает очень полезно добавить возможность получения координат устройства и хозяина, если оба находятся в одном месте, и использовать эти *координаты* для каких-либо целей. Например, уже существуют приложения, которые позволяют отслеживать параметры человека (спортсмена) во время преодоления некоторых расстояний бегом, на велосипеде, на лыжах и т. д. Такое *приложение* работает во время тренировки (устройство должно перемещаться вместе со спортсменом), по окончанию можно получить полную статистику маршрута: *точное время* в пути, *расстояние*, подъемы/спуски, среднюю скорость, потраченные калории и т. д. Заметим, что большая часть информации опирается на данные, полученные со спутников *GPS*.

Рассмотрение особенностей смартфонов будет неполным, если оставить без внимания датчики и сенсоры, которыми оснащены большинство устройств. Эти микроустройства обеспечивают *связь* смартфона с окружающей средой и добавляют новые удивительные функции. С помощью датчика приближения, например, можно отключать подсветку экрана при приближении телефона к уху пользователя во время разговора, блокировать экран, чтобы не было возможности случайно нажать на отбой. *Акселерометр* может использоваться для смены ориентации экрана, для управления в играх, особенно симуляторах, а также в качестве шагомера. Датчик освещенности позволяет регулировать яркость экрана.*Гироскоп* может применяться для определения более точного позиционирования устройства в пространстве.

Все рассмотренные особенности в совокупности увеличивают привлекательность смартфонов, позволяют разработчикам создавать приложения с разнообразными, полезными, интересными и иногда неожиданными функциями. Далее в лекции рассмотрим перечисленные возможности смартфонов более подробно и узнаем как можно их использовать при разработке приложений.

**9.3 Сенсорное (touch) управление**

В этом разделе лекции рассмотрим возможности добавления сенсорного управления в мобильные приложения под *Android*. Сенсорное управление подразумевает использование сенсорных жестов для взаимодействия с приложением. В [таблице 9.1](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21992?page=1#table.9.1) представлен набор жестов, поддерживаемый системой *Android*.

|  |  |
| --- | --- |
| Таблица 9.1. Система жестов Android | |
| http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_a01.jpg | **Касание (touch).**  **Использование:** Запуск действия по умолчанию для выбранного элемента.  **Выполнение:** нажать, отпустить. |
| http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_a02.jpg | **Длинное касание (long touch).**  **Использование:** Выбор элемента. Не стоит использовать этот жест для вызова контекстного меню.  **Выполнение:** нажать, ждать, отпустить. |
| http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_a03.jpg | **Скольжение или перетаскивание (swipe or drag).**  **Использование:** Прокрутка содержимого или навигация между элементами интерфейса одного уровня иерархии.  **Выполнение:** нажать, переместить, отпустить. |
| http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_a04.jpg | **Скольжение после длинного касания (long press drag).**  **Использование:** Перегруппировка данных или перемещение в контейнер.  **Выполнение:** длительное касание, переместить, отпустить. |
| http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_a05.jpg | **Двойное касание (double touch).**  **Использование:** Увеличение масштаба, выделение текста.  **Выполнение:** быстрая последовательность двух касаний. |
| http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_a06.jpg | **Перетаскивание с двойным касанием (double touch drag).**  **Использование:** Изменение размеров: расширение или сужение по отношению к центру жеста.  **Выполнение:** касание, следующее за двойным касанием со смещением вверх или вниз при этом:   * смещение вверх уменьшает размер содержимого; * смещение вниз увеличивает размер содержимого. |
| http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_a07.jpg | **Сведение пальцев (pinch close).**  **Использование:** уменьшение содержимого, сворачивание.  **Выполнение:** касание экрана двумя пальцами, свести, отпустить. |
| http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_a08.jpg | **Разведение пальцев (pinch open).**  **Использование:** увеличение содержимого, разворачивание.  **Выполнение:** касание экрана двумя пальцами, развести, отпустить. |

О возможности управлять приложением с помощью сенсорных жестов можно говорить в том случае, когда *приложение* способно распознать, что под набором касаний экрана скрывается некоторый жест и выполнить соответствующее действие. Процесс распознавания жеста обычно состоит из двух этапов: сбор данных и *распознавание* жеста. Рассмотрим эти этапы подробнее.

**9.3.1 Сбор данных о сенсорных событиях**

Основные действия, которые может произвести пользователь при взаимодействии с сенсорным экраном: коснуться экрана пальцем, переместить палец по экрану и отпустить. Эти действия распознаются системой Android, как сенсорные события (touch-события).

Каждый раз при появлении сенсорного события инициируется вызов метода onTouchEvent(). Обработка события станет возможной, если этот метод реализован в классе активности или некоторого компонента, иначе событие просто игнорируется.

Жест начинается, при первом касании экрана, продолжается пока система отслеживает положение пальцев пользователя и заканчивается получением финального события, состоящего в том, что ни один палец не касается экрана. Объект MotionEvent, передаваемый в методonTouchEvent(), предоставляет детали каждого взаимодействия. Рассмотрим основные константы класса MotionEvent, определяющие сенсорные события:

* MotionEvent.ACTION\_DOWN - касание экрана пальцем, является начальной точкой для любого сенсорного события или жеста;
* MotionEvent.ACTION\_MOVE - перемещение пальца по экрану;
* MotionEvent.ACTION\_UP - поднятие пальца от экрана.

Приложение может использовать предоставленные данные для распознавания жеста.

Можно реализовать свою собственную обработку событий для распознавания жеста, таким образом можно работать с произвольными жестами в приложении. Если же в приложении необходимо использовать стандартные жесты, описанные в [таблице 9.1](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21992?page=1#table.9.1), можно воспользоваться классомGestureDetector. Этот класс позволяет распознать стандартные жесты без обработки отдельных сенсорных событий.

**9.3.2 Распознавание жестов**

Android предоставляет класс GestureDetector для распознавания стандартных жестов. Некоторые жесты, которые он поддерживает включают: onDown(), onLongPress(), onFling() и т. д. Можно использовать класс GestureDetector в связке с методомonTouchEvent(). Подробно распознавание поддерживаемых жестов рассмотрено в первой части лабораторной работы в этой теме.

Начиная с версии 1.6, Android предоставляет API для работы с жестами, который располагается в пакете android.gesture и позволяет сохранять, загружать, создавать и распознавать жесты. Виртуальное устройство Android (AVD), начиная все с той же версии 1.6, содержит предустановленное приложение, которое называется Gesture Builder и позволяет создавать жесты. После создания жесты сохраняются на SD карте виртуального устройства и могут быть добавлены в приложение в виде бинарного ресурса.

Для распознавания жестов необходимо добавить компонент GestureOverlayView, в XML файл активности. Этот компонент может быть добавлен как обычный элемент графического интерфейса пользователя и встроен в компоновку, например RelativeLayout. C другой стороны он может быть использован, как прозрачный слой поверх других компонентов, в этом случае в XML файле активности он должен быть записан, как корневой элемент.

Кроме всего вышеперечисленного, для использования собственных жестов в приложении необходимо реализовать интерфейсOnGesturePerformedListener и его метод onGesturePerformed(). Подробно создание и использование собственных жестов рассмотрено во второй части лабораторной работы в этой теме.

**9.4 Работа с мультимедиа**

*Мультимедиа* библиотека *Android* включает поддержку воспроизведения *множества* наиболее распространенных форматов, что позволяет легко использовать в приложениях аудио, видео и изображения. Можно проигрывать аудио или видео из медиа файлов сохраненных как ресурсы приложения (*raw* ресурсы), из файлов, расположенных в файловой системе или из потока данных, получаемого через сетевое соединение, для всего этого используется MediaPlayer *API*.

**Замечание:** проигрывать аудиофайлы можно только на стандартном устройстве вывода, невозможно воспроизводить аудио во время звонка.

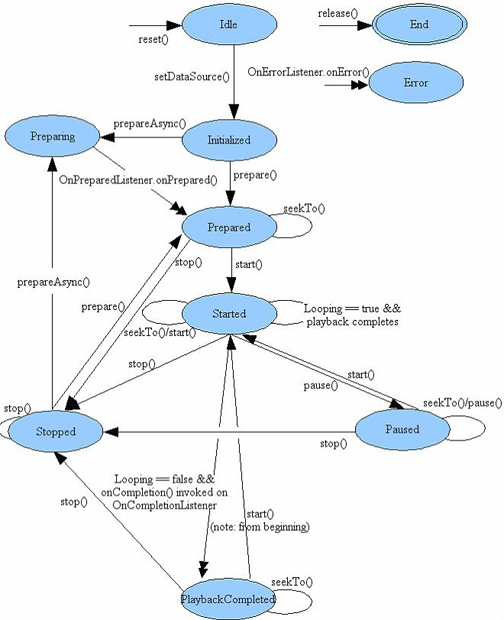
Актуальная *информация* о поддерживаемых форматах аудио и видео приводится по ссылке: <http://developer.android.com/guide/appendix/media-formats.html>.

Для воспроизведения аудио и видео *Android* предоставляет *класс* MediaPlayer. Причем при работе с аудиоконтентом этот *класс* позволяет воспроизводить необработанные данные, т. е. возможно проигрывание динамически генерируемого аудио.

*Диаграмма* жизненного *цикла* экземпляра класса MediaPlayer представлена на [рис. 9.1](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21992?page=2#image.9.1). Овалы представляют состояния объектаMediaPlayer, дуги показывают вызовы каких методов необходимо выполнить, чтобы сменить состояние объекта MediaPlayer. Дуги с одной стрелкой представляют вызовы синхронных методов, с двумя стрелками - вызовы асинхронных методов.

В ходе жизненного *цикла* *объект* MediaPlayer проходит через несколько состояний:

* **бездействие (Idle)** - создан экземпляр класса MediaPlayer для создания может использоваться оператор new или вызов методаreset() (см. [рис. 9.1](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21992?page=2#image.9.1));

[](http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_01.jpg)

[увеличить изображение](http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_01.jpg)  
**Рис. 9.1.**Жизненный цикл экземпляра класса MediaPlayer

(<http://developer.android.com/reference/android/media/MediaPlayer.html#StateDiagram>)

* **инициализирован (Initialized)** - задан источник медиа-информации, для задания источника используется метод setDataSource();
* **ошибка (Error)** - появилась какая-то ошибка, например, не поддерживаемый аудио/видео формат, слишком высокое разрешение, чтобы вывести объект из этого состояния, необходимо вызвать метод reset();
* **подготовка (Preparing)** - MediaPlayer занимается подготовкой медиаисточника к воспроизведению, подготовка инициируется методомprepareAsync();
* **готов (Prepared)** - состояние готовности к воспроизведению, может быть достигнуто двумя способами:
  + синхронный способ: вызов метода prepare(), который переводит объект в готовое состояние;
  + асинхронный способ: срабатывание метода onPrepared() интерефейса OnPreparedListener() в состоянии подготовки, как реакция на событие готовности;
* **запущен (Started)** - выполняется воспроизведение медиа-контента, в это состояние объект переходит после вызова метода start();
* **приостановлен (Paused)** - воспроизведение приостановлено, MediaPlayer переходит в это состояние после вызова метода pause();
* **остановлен (Stopped)** - воспроизведение остановлено, MediaPlayer переходит в это состояние после вызова метода stop();
* **воспроизведение завершено (Playback Completed)** - достигнут конец воспроизводимого содержания, в это состояние объект переходит после срабатывания метода onCompleted() интерфейса-слушателя OnCompitionListener, как реакции на конец воспроизводимого материала;

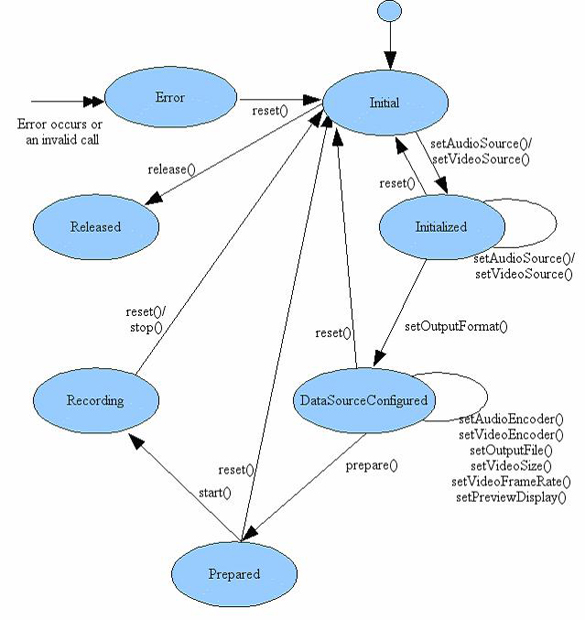
**Замечание:** из состояний Paused, Playback Completed можно вернуться к воспроизведению вызовом метода start(). Из состоянияStopped прежде, чем вернуться в состояние воспроизведения, необходимо пройти через подготовку медиа-содержимого.

Вызов метода seekTo() позволяет поменять место воспроизведения.

* **конец (End)** - конец жизненного цикла MediaPlayer объекта, в это состояние объект переходит после вызова метода release().

Для получения более детальной информации см. ссылки: <http://developer.android.com/guide/topics/media/mediaplayer.html>;<http://developer.android.com/reference/android/media/MediaPlayer.html>.

Для записи аудио и видео *Android* предоставляет *класс* MediaRecorder. *Диаграмма* жизненного *цикла* экземпляра класса MediaRecorderпредставлена на [рис. 9.2](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21992?page=2#image.9.2). Овалы представляют состояния объекта MediaPlayer, дуги показывают вызовы каких методов необходимо выполнить, чтобы сменить состояние объекта MediaPlayer. Дуги с одной стрелкой представляют вызовы синхронных методов, с двумя стрелками - вызовы асинхронных методов.

[](http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_02.jpg)

[увеличить изображение](http://www.intuit.ru/EDI/26_09_16_2/1474842043-31192/tutorial/1258/objects/9/files/09_02.jpg)  
**Рис. 9.2.**Жизненный цикл экземпляра класса MediaRecorder

(<http://developer.android.com/reference/android/media/MediaRecorder.html>)

В ходе жизненного *цикла* *объект* MediaRecorder проходит через несколько состояний:

|  |  |
| --- | --- |
| **начальное (Initial)** | - создан объект класса MediaRecover, для создания может использоваться оператор new или вызов методаreset() (см. [рис. 9.2](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21992?page=2#image.9.2)); |
| **инициализирован (Initialized)** | - объект MediaRecover готов к использованию, в данное состояние объект переходит после вызова одного из методов setAudioSource() или setVideoSource(), которые задают источники аудио или видео для записи; |
| **сконфигурирован приемник данных для записи (Data Source Configured)** | - задаются основные свойства приемника данных, состояние инициируется методом setOutputFormat(), для настройки свойств должны быть выполнены некоторые методы из списка: setAudioEncoder(),setVideoEncoder(), setOutputFile(), setVideoSize(), setVideoFrameRate(), setPreviewDisplay(); |
| **готов (Prepared)** | - состояние готовности к записи, инициируется методом prepare(); |
| **записывает (Recording)** | - идет запись, инициируется вызовом метода start(); |
| **освобожден (Released)** | - запись завершена, все ресурсы освобождены. |

Для получения более детальной информации см. ссылки: <http://developer.android.com/reference/android/media/MediaRecorder.html>;<http://developer.android.com/guide/topics/media/audio-capture.html>.

**9.5 Использование встроенной камеры**

Платформа *Android* включает поддержку камеры, доступной на устройстве, позволяющей приложениям получать фотографии и записывать видео. Для решения этих задач, существует два способа:

1. непосредственное обращение к камере;
2. использование намерений (Intent) для вызова существующего приложения.

Рассмотрим основные относящиеся к делу классы:

|  |  |
| --- | --- |
| Camera | - класс, реализующий управление камерами устройства. Этот класс используется для получения фотографий или записи видео при создании приложения, работающего с камерой. |
| SurfaceView | - класс, используемый для предоставления пользователю возможности предварительного просмотра. |
| MediaRecorder | - класс, используемый для записи видео с камеры. |
| Intent | - класс, содержащий абстрактное описание выполняемой операции, которое передается системе Android, а ОС сама находит и запускает необходимое приложение и возвращает результат его работы. Для работы с камерой используются два типа намерений:   * MediaStore.ACTION\_IMAGE\_CAPTURE - для запроса на выполнение фотоснимков; * MediaStore.ACTION\_VIDEO\_CAPTURE - для запроса на запись видео. |

Подробно процесс разработки приложения, позволяющего производить фото и видеосъемку рассмотрен в третьей части лабораторной работы к данной теме.

**9.6 Взаимодействие с системами позиционирования**

Системы позиционирования позволяют определить местоположение в некоторой системе координат, обычно определяются широта и долгота.

Так как *смартфон* является мобильным телефоном, ему доступны методы, обычно используемые мобильными телефонами для определения своего местоположения.

* Во-первых, смартфон постоянно связывается с сотовой вышкой, в зоне действия которой он находится. Каждая сотовая вышка в мире имеет уникальный идентификатор, называемый идентификатором соты (Cell ID), а также для нее точно известны широта и долгота ее расположения. В связи с этим, смартфон, зная идентификатор соты, в которой он находится, может получить географические координаты центра этой соты. Радиусы сот варьируются в зависимости от того, насколько активный сетевой трафик ожидается в конкретном районе. Разумеется, такой способ позиционирования дает очень приблизительные результаты, что называется: "плюс-минус трамвайная остановка".
* Во-вторых, чаще всего смартфон оказывается в зоне действия более, чем одной сотовой вышки. В современных мобильных технологиях, начиная с поколения 2G, сотовая вышка может определить, с какого направления приходит сигнал. В случае, когда телефон находится в зоне действия двух или трех сотовых вышек, они могут выполнять триангуляцию его местоположения. Телефон может запросить у сети информацию о том, где он находится. Такая техника определения местоположения может быть очень точной и не требует установки дополнительного оборудования.

Дополнительно к возможностям определения местоположения, доступным обычным мобильным телефонам, большинство смартфонов укомплектованы спутниковыми системами глобального позиционирования (*Global Positioning System*, *GPS*). В настоящее время наиболее распространенными в мире системами глобального спутникового позиционирования являются: *GPS*, разработанная и реализованная в США, и система ГЛОНАСС (Глобальная навигационная спутниковая система), советская, а позже российская спутниковая система навигации. Многие смартфоны могут использовать сигналы сразу от двух навигационных систем, что позволяет серьезно увеличить *надежность* и *точность*определения координат, прежде всего, в городских условиях.

В *дополнение* к вышеперечисленным методам позиционирования, добавляется возможность использования сигналов WiFi, Bluetooth и *NFC*, а также внутреннего сенсора для более точной геолокации, особенно внутри помещений.

В этом разделе нас, в первую *очередь*, будет интересовать возможность добавления в приложения способностей определять *координаты*устройства и работать с картами. При создании приложений, учитывающих текущее местоположение, под *Android* можно воспользоваться *GPS* и определением местоположения в сети (с помощью *Network* *Location* *Provider*). Несмотря на то, что *GPS* дает более точные результаты, он не очень хорошо работает в помещениях (чаще не работает), он сильно расходует заряд батареи и скорость определения координат не всегда соответствует ожиданиям пользователя. *Network* *Location* *Provider* определяет *координаты*, используя сигналы сотовых вышек и WiFi, может работать как на улице, так и внутри помещений, более экономно расходует заряд батареи и работает быстрее по сравнению с *GPS*. Для получения координат в приложении можно использовать оба способа или один из них на выбор.

*Android* предоставляет приложениям *доступ* к геолокационным возможностям мобильного устройства, через классы пакета *android*.*location*. Центральным классом этого пакета является *класс* LocationManager, который предоставляет *доступ* к системным сервисам для определения координат устройства.

В приложения можно добавлять карты, используя Google Maps *Android* *API*, которое автоматически управляет доступом к серверам Google Maps, загрузкой данных, отображением карт и сенсорными жестами на карте. Также можно использовать вызовы *API* для добавления маркеров, многоугольников и внешних прозрачных слоев, а также для изменения пользовательского представления отдельных участков карты.

Ключевым классом в Google Maps *Android* *API* является *класс* MapView, который отображает карту с данными полученными из сервиса Google Maps. Когда MapView имеет фокус, он может перехватывать нажатия клавиш и сенсорные жесты для выполнения автоматического перемещения и изменения масштаба карты, а также может управлять сетевыми запросами для получения дополнительных фрагментов карты. Этот *класс* так же предоставляет все элементы пользовательского интерфейса, необходимые для управления картой.

Google Maps *Android* *API* не является частью платформы *Android*, но доступен на любом устройстве с Google *Play* *Store*, работающем, начиная с*Android* 2.2, через Google *Play* services. Чтобы обеспечить возможность интеграции Google Maps в приложения, в *Android* *SDK* необходимо установить библиотеку Google *Play* services.

Подробнее вопросы добавления в приложения геолокационных возможностей и использование карт (Google Maps) рассмотрены в четвертой части лабораторной работы к данной теме.

**9.7 Другие сенсоры и датчики**

Большинство устройств, работающих под управлением *Android*, укомплектованы встроенными сенсорами, которые предоставляют исходные данные высокой точности. Сенсоры могут быть полезны в том случае, если необходимо регистрировать положение и перемещения, повороты устройства в трехмерном пространстве, а также изменения параметров окружающей среды.

Платформа *Android* поддерживает три категории сенсоров:

|  |  |
| --- | --- |
| **Датчики движения** | Эти сенсоры измеряют силы ускорения и вращательные силы по трем осям. Эта категория включает акселерометры, гироскопы, датчики вектора вращения и сенсоры силы тяжести. |
| **Датчики окружающей среды** | Эти сенсоры измеряют различные параметры окружающей среды, такие как температура воздуха и давление, освещенность и влажность. Эта категория включает барометры, термометры и датчики освещенности. |
| **Датчики положения** | Эти сенсоры измеряют физическое положение устройства. Эта категория включает магнитометры и датчики ориентации устройства в пространстве. |

Сенсоры могут быть реализованы аппаратно или программно. Аппаратно-реализованные датчики являются физическими элементами встроенными в *мобильное устройство*, они получают данные путем прямых измерений некоторых свойств, таких как ускорение, сила геомагнитного поля или изменение углов. Программно-реализованные датчики получают свои данные с одного или нескольких физических датчиков и вычисляют *значение*, которое от них ожидается.

Какие типы датчиков поддерживаются *Android* можно узнать по ссылке: <http://developer.android.com/guide/topics/sensors/sensors_overview.html>.

*Android* предоставляет набор классов и интерфейсов для работы с сенсорами. Эти классы и интерфейсы являются частью пакета*android*.*hardware* и позволяют выполнять следующие задачи:

* определять какие сенсоры доступны на устройстве;
* определять индивидуальные возможности сенсоров, такие как максимальное значение, производитель, требования к потребляемой энергии и разрешения;
* собирать данные с сенсоров и определять минимальную частоту, с которой выполняется сбор данных;
* подключать и отключать слушателей событий от датчиков, события состоят в изменении значений датчиков.

Для работы с датчиками *Android* предоставляет следующие классы и интерфейсы:

|  |  |
| --- | --- |
| SensorManager | Этот класс может использоваться для создания экземпляра сервиса, связанного с сенсором. Также он предоставляет различные методы для доступа и составления списка сенсоров, подключения и отключения слушателей событий от сенсоров, сбора информации. Этот класс содержит константы, которые используются для задания точности сенсора, частоты получения данных и настройки датчиков. |
| Sensor | Этот класс используется для создания экземпляра датчика, предоставляет методы, позволяющие определить свойства сенсора. |
| SensorEvent | Система использует этот класс для создания объекта, соответствующего событию датчика и предоставляющего следующую информацию: данные сенсора; тип сенсора, который породил событие, точность данных и время появления события. |
| SensorEventListener | Данный интерфейс может использоваться для реализации двух методов, получающих уведомления (события датчиков), когда меняется значение сенсора или когда меняется точность сенсора. |

Использование в приложении полученных от сенсоров данных будет рассмотрено в лабораторной работе темы 7. Подробнее об использовании сенсоров можно узнать по ссылке: <http://developer.android.com/guide/topics/sensors/sensors_overview.html>.

**Демонстрации распознавания стандартных жестов**

**Аннотация:**Разработать простейшие приложения для демонстрации распознавания стандартных жестов.

**Ключевые слова:**[android](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword1), [класс](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword2), [значение](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword5), [приложение](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword6), [распознавание](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword7), [активность](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword9), [поле](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword10), [пользователь](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword12), [информация](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword14), [интерфейс](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword16), [слово](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=1#keyword17),[поля класса](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=2#keyword37)

**Цель лабораторной работы:**

разработать простейшие приложения для демонстрации распознавания стандартных жестов.

**Задачи лабораторной работы:**

* рассмотреть распознавание всех поддерживаемых жестов;
* рассмотреть распознавание только части поддерживаемых жестов.

**10.1 Введение**

Для работы со стандартными жестами *Android* предоставляет *класс* GestureDetector. Этот *класс* содержит два вложенных интерфейса-слушателя: OnGestureListener и OnDoubleTapListener, эти интерфейсы задают методы, отслеживающие стандартные жесты. А такжеGestureDetector содержит вложенный *класс* SimpleOnGestureListener, который содержит пустые реализации, возвращающие *значение*false, где это необходимо, всех методов интерфейсов: OnGestureListener и OnDoubleTapListener.

В лабораторной работе рассмотрим две возможности распознавания жестов:

* случай распознавания всех поддерживаемых жестов, для этого реализуем в классе активности оба интерфейса;
* случай распознавания только некоторого набора поддерживаемых жестов, для этого в классе активности объявим внутренний класс-наследник класса GestureDetector.SimpleOnGestureListener.

**10.2 Распознавание всех поддерживаемых жестов**

Разработаем *приложение*, в котором продемонстрируем *распознавание* всех поддерживаемых жестов. *Приложение* содержит одну *активность*, одно информационное *поле* для вывода информации о распознанном жесте. *Приложение* работает следующим образом: *пользователь*выполняет один из поддерживаемых сенсорных жестов, в информационном *поле* отображается *информация* о распознанном жесте.

1. Создадим простое приложение и добавим на форму TextView для вывода информации.
2. Настроим логику приложения. В java класс, соответствующий активности внесем следующие дополнения.
   * Класс активности должен реализовывать интерфейсы: GestureDetector.OnGestureListener иGestureDetector.OnDoubleTapListener, для этого в объявление класса добавим конструкцию:
   * implements GestureDetector.OnGestureListener,
   * GestureDetector.OnDoubleTapListener

* + Нам понадобится экземпляр класса GestureDetectorCompat поэтому в качестве поля класса активности объявим следующую переменную:
  + GestureDetectorCompat mDetector;

В методе onCreate() класса активности, создадим экземпляр класса GestureDetectorCompat и присвоим его переменной mDetector:

mDetector = new GestureDetectorCompat(this,this);

одним из параметров конструктора является *класс*, который реализует *интерфейс* GestureDetector.OnGestureListener, в нашем случае использовано *слово* this, т. е. параметром является сам *класс* активности. Этот *интерфейс* уведомляет пользователей когда появляется определенное сенсорное событие.

В методе OnCreate() класса активности, следующая строка:

mDetector.setOnDoubleTapListener(this);

устанавливает слушатель событий, связанных с двойным касанием, это должен быть *класс*, реализующий *интерфейс*GestureDetector.OnDoubleTapListener. В нашем случае использовано *слово* this, т.е. слушателем будет опять сам *класс* активности.

1. Чтобы позволить вашему объекту GestureDetector получать события, необходимо переопределить метод onTouchEvent() для активности или элемента GUI. И передавать в экземпляр детектора все обнаруженные события.
2. public boolean onTouchEvent(MotionEvent event){
3. this.mDetector.onTouchEvent(event);
4. // Be sure to call the superclass implementation
5. return super.onTouchEvent(event);
6. }

1. После проведенной подготовки пришло время реализовать все методы, объявленные в интерфейсах, отвечающих за прослушивание сенсорных событий.

Методы интерфейса GestureDetector.OnGestureListener:

|  |  |
| --- | --- |
| onDown() | - отслеживает появление касания, т. е. палец прижат к экрану; |
| onFling() | - отслеживает появление жеста смахивания; |
| onLongPress() | - отслеживает удерживание пальца прижатым к экрану длительное время; |
| onScroll() | - отслеживает появление жеста прокрутки (пролистывания); |
| onShowPress() | - отслеживает, что произошло событие касания и больше никаких событий не происходит короткое время; |
| onSingleTapUp() | - отслеживает появление жеста одиночного нажатия (клик). |

Методы интерфейса GestureDetector.OnDoubleTapListener:

|  |  |
| --- | --- |
| onDoubleTap() | - отслеживает появление жеста двойного нажатия ("двойной клик"); |
| onDoubleTapEvent() | - отслеживает появление события во время выполнения жеста двойного нажатия, включая касание, перемещение, подъем пальца. |
| onSingleTapConfirmed() | - отслеживает появление жеста одиночного нажатия (клик). |

В [листинге 10.1](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=2#example.10.1) представлен код приложения, в котором распознаются все поддерживаемые жесты, *информация* о появившемся и распознанном жесте выдается в информационное *поле* (TextView).

В качестве практики предлагается воспроизвести данное *приложение* и проверить, как система распознает тот или иной жест. Очень полезно для понимания, как выполняются основные жесты.

**10.3 Распознавание только части поддерживаемых жестов**

Разработаем *приложение*, в котором продемонстрируем *распознавание* только некоторой части поддерживаемых жестов по выбору программиста. Мы рассмотрим *распознавание* жеста смахивания (fling). *Приложение* содержит одну *активность*, одно информационное *поле* для вывода информации о распознанном жесте. *Приложение* работает следующим образом: *пользователь* выполняет один из поддерживаемых сенсорных жестов, в информационном *поле* отображается *информация* о распознанном жесте.

1. Создадим простое приложение и добавим на форму TextView для вывода информации.
2. Настроим логику приложения. В java класс, соответствующий активности внесем следующие дополнения.

Нам понадобится экземпляр класса GestureDetectorCompat поэтому в качестве *поля класса* активности объявим следующую переменную:

GestureDetectorCompat mDetector;

В методе onCreate() класса активности, создадим экземпляр класса GestureDetectorCompat и присвоим его переменной mDetector:

mDetector=new GestureDetectorCompat(this, new MyGestListener());

в конструкторе аргументом, отвечающим за отслеживание сенсорных событий, служит экземпляр класса MyGestListener() - внутренний*класс*, который является наследником класса GestureDetector.SimpleOnGestureListener.

Имеет смысл немного рассмотреть *класс* GestureDetector.SimpleOnGestureListener. Этот *класс* реализует интерфейсыGestureDetector.OnGestureListener и GestureDetector.OnDoubleTapListener, все методы заявленные в интерфейсах в этом классе имеют пустую реализацию и те, которые должны возвращать *значение*, возвращают false. Поэтому для распознавания какого-то события или некоторого подмножества событий достаточно написать реализацию соответствующих методов, в классе наследнике.

В [листинге 10.2](http://www.intuit.ru/studies/curriculums/16235/courses/1191/lecture/21994?page=2#example.10.2) представлен код приложения, в котором распознается только жест смахивания, т. е. реализован метод onFling(),*информация* о появившемся и распознанном жесте выдается в информационное *поле* (TextView). В качестве слушателя используется экземпляр класса MyGestListener(), являющийся наследником класса GestureDetector.SimpleOnGestureListener.

Получить больше информации о распознавании жестов можно по ссылке: <http://developer.android.com/training/gestures/index.html>

package com.example.lab5\_1\_gestall;

import android.os.Bundle;

import android.app.Activity;

import android.support.v4.view.GestureDetectorCompat;

import android.view.\*;

import android.widget.\*;

public class MainActivity extends Activity

implements GestureDetector.OnGestureListener,

GestureDetector.OnDoubleTapListener

{

TextView tvOutput;

GestureDetectorCompat mDetector;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

tvOutput = (TextView)findViewById(R.id.textView1);

mDetector = new GestureDetectorCompat(this,this);

mDetector.setOnDoubleTapListener(this);

}

public boolean onTouchEvent(MotionEvent event){

this.mDetector.onTouchEvent(event);

// Be sure to call the superclass implementation

return super.onTouchEvent(event);

}

@Override

public boolean onDown(MotionEvent event) {

tvOutput.setText("onDown: " + event.toString());

return false;

}

@Override

public boolean onFling(MotionEvent event1, MotionEvent event2,

float velocityX, float velocityY) {

tvOutput.setText("onFling: " + event1.toString()+event2.toString());

return true;

}

@Override

public void onLongPress(MotionEvent event) {

tvOutput.setText("onLongPress: " + event.toString());

}

@Override

public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,

float distanceY) {

tvOutput.setText("onScroll: " + e1.toString()+e2.toString());

return true;

}

@Override

public void onShowPress(MotionEvent event) {

tvOutput.setText("onShowPress: " + event.toString());

}

@Override

public boolean onSingleTapUp(MotionEvent event) {

tvOutput.setText("onSingleTapUp: " + event.toString());

return true;

}

@Override

public boolean onDoubleTap(MotionEvent event) {

tvOutput.setText("onDoubleTap: " + event.toString());

return true;

}

@Override

public boolean onDoubleTapEvent(MotionEvent event) {

tvOutput.setText("onDoubleTapEvent: " + event.toString());

return true;

}

@Override

public boolean onSingleTapConfirmed(MotionEvent event) {

tvOutput.setText("onSingleTapConfirmed: " + event.toString());

return true;

}

}

Листинг 10.1. Распознавание поддерживаемых жестов с помощью реализации интерфейсов

package com.example.lab5\_1\_gestsubset;

import android.os.Bundle;

import android.app.Activity;

import android.support.v4.view.\*;

import android.view.\*;

import android.widget.\*;

public class SubsetGestActivity extends Activity {

private GestureDetectorCompat mDetector;

private TextView tvOut;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_subset\_gest);

mDetector = new GestureDetectorCompat(this, new MyGestListener());

tvOut = (TextView)findViewById(R.id.textView1);

}

@Override

public boolean onTouchEvent(MotionEvent event){

this.mDetector.onTouchEvent(event);

return super.onTouchEvent(event);

}

class MyGestListener extends GestureDetector.SimpleOnGestureListener {

@Override

public boolean onFling(MotionEvent event1, MotionEvent event2,

float velocityX, float velocityY) {

tvOut.setText("onFling: " + event1.toString()+event2.toString());

return true;

}

}

}

Листинг 10.2. Распознавание жестов с использованием класса GestureDetector.SimpleOnGestureListener.