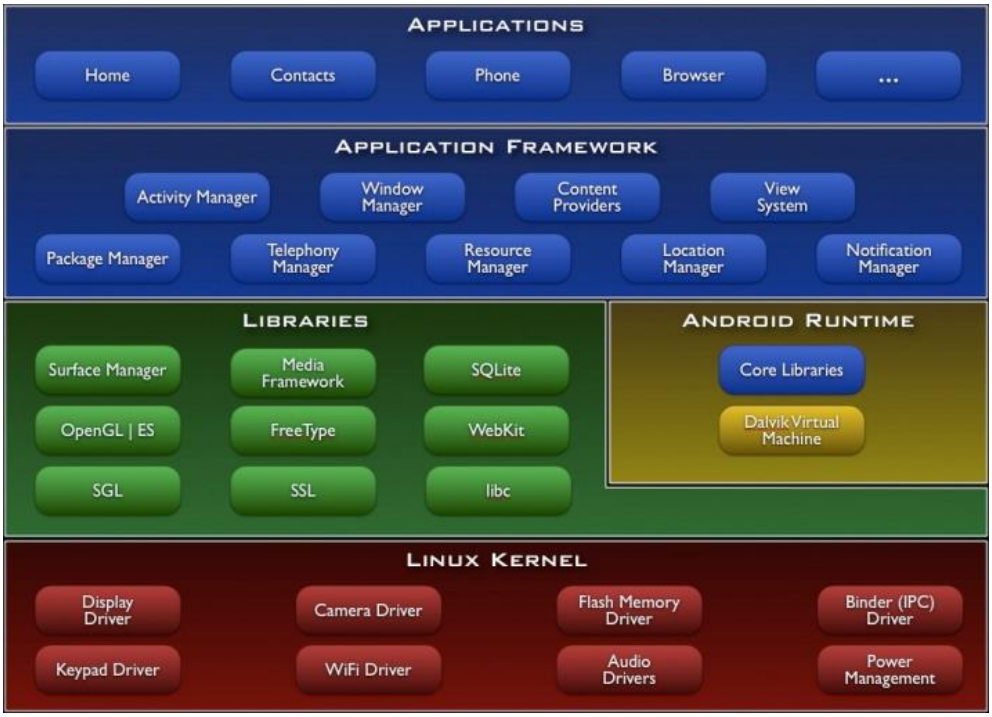
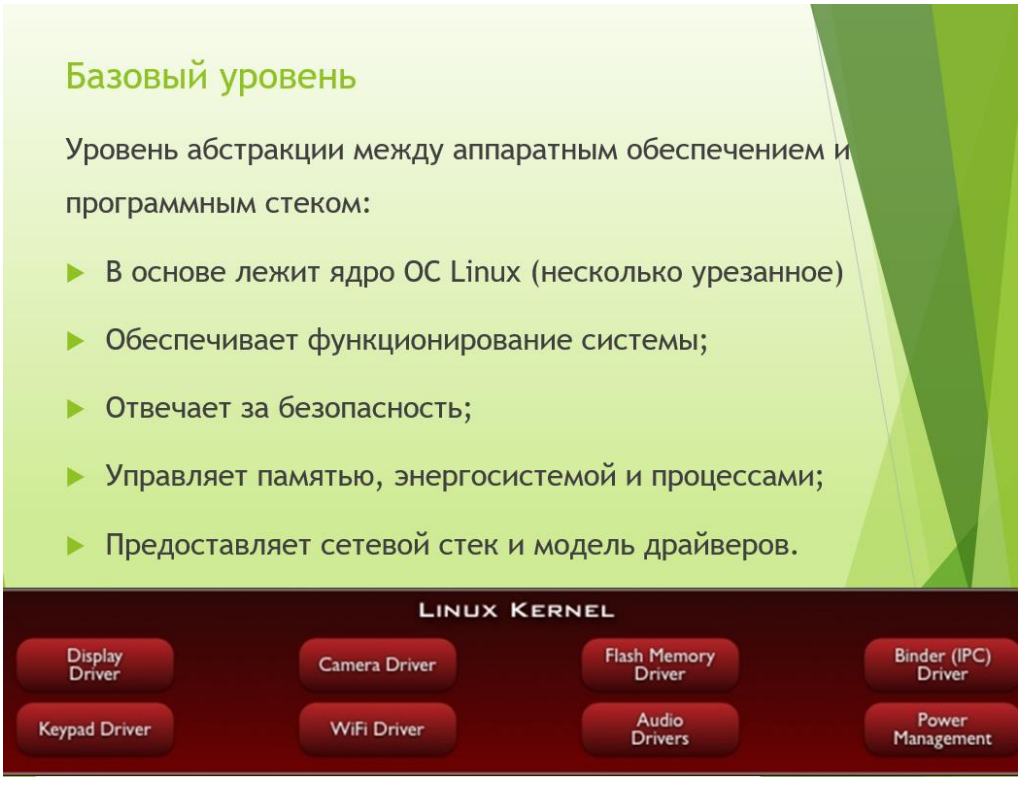
Оглавление

1. Архитектура Android

Архитектура Android приложений основана на идее многократного использования компонентов, которые являются основными строительными блоками

Каждый компонент является отдельной сущностью и помогает определить общее поведение приложения





Набор библиотек

Обеспечивает важнейший базовый функционал для приложений:

- Алгоритмы для вышележащих уровней
- Поддержка файловых форматов
- Кодирование и декодирование информации
- Отрисовка графики и т. д.

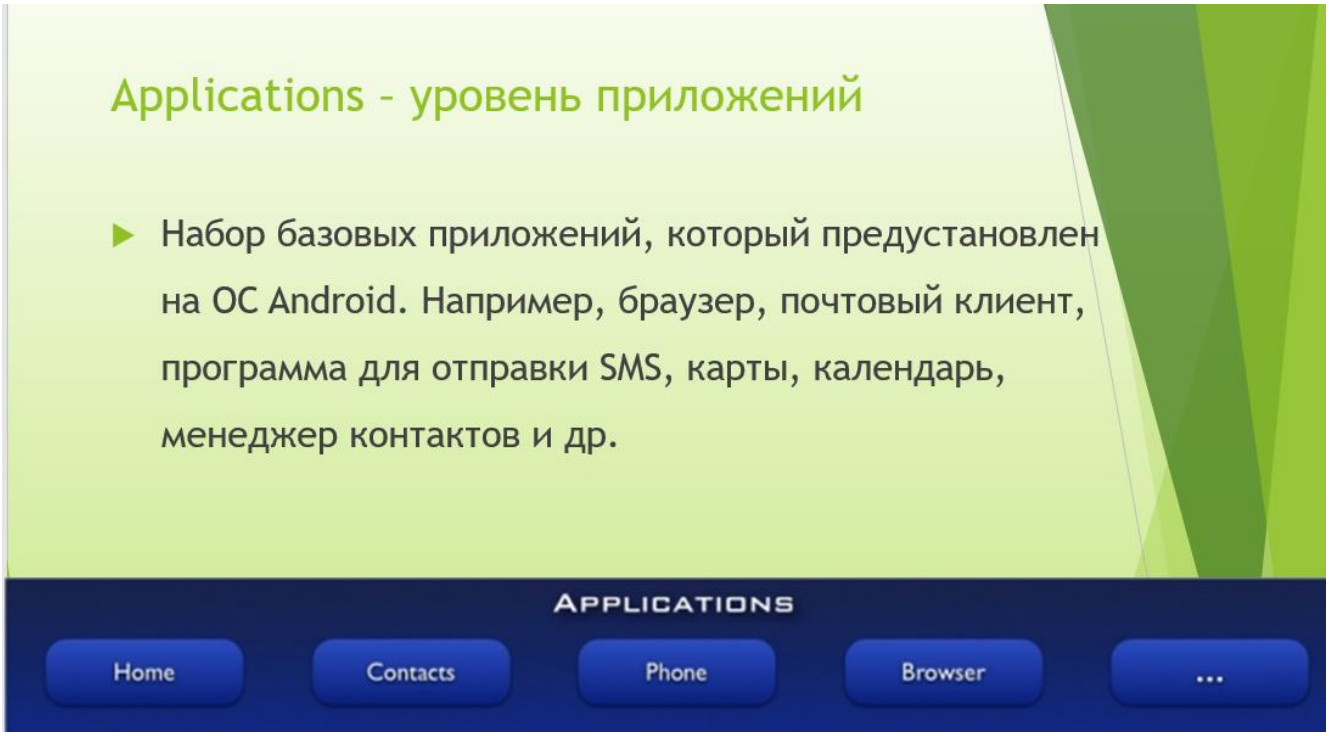
Application Framework - уровень каркаса приложений

- Обеспечивает разработчикам доступ к API, предоставляемым компонентами системы уровня библиотек
- Любому приложению предоставляются уже реализованные возможности других приложений, к которым разрешено получать доступ

2. Среда выполнения Android Runtime

Android Runtime — среда выполнения Android-приложений, разработанная компанией Google как замена Dalvik. ART впервые появился в Android 4.4 как тестовая функция, а в Android 5.0 полностью заменил Dalvik. В отличие от Dalvik, который использует JIT-компиляцию, ART компилирует приложение во время его установки.

3. Состав приложения Android



4. Activity жизненный цикл

Activity – основная единица графического интерфейса (аналог окна или экранной формы)

Активность – это видимая часть приложения (экран, окно, форма), отвечает за отображение графического интерфейса пользователя

Например: Приложение для работы с будильником

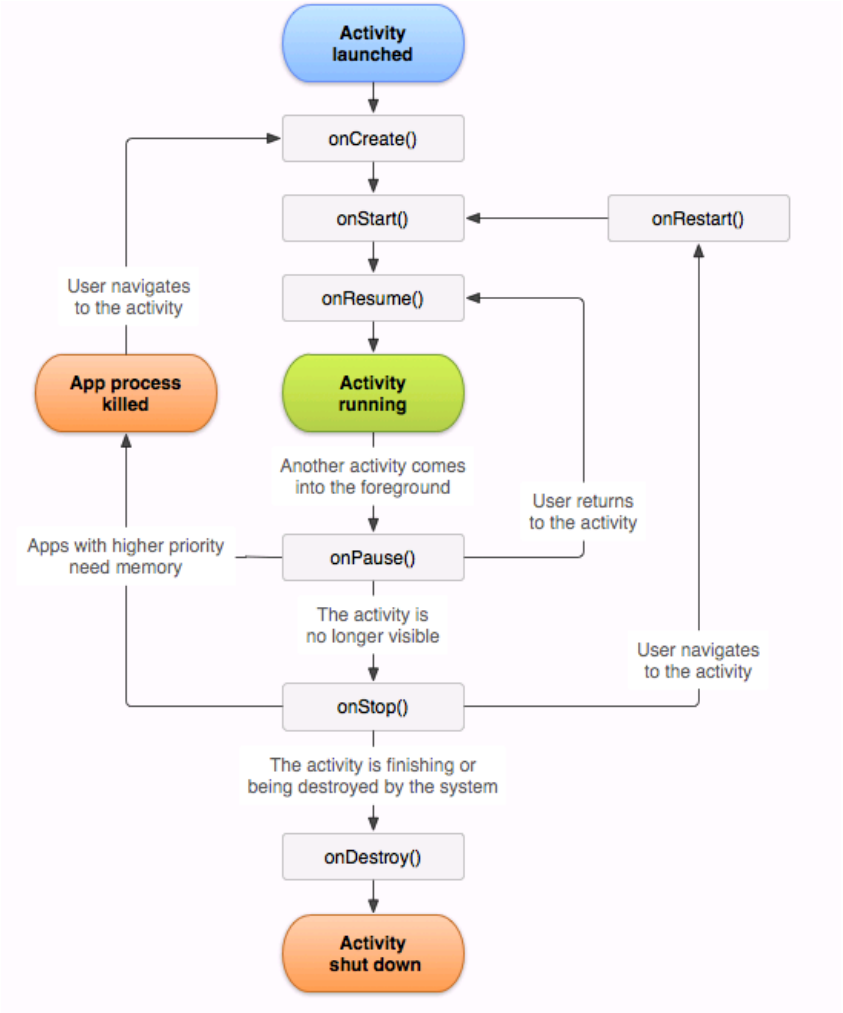
При этом Приложение может иметь несколько активностей. Активности приложения не зависят друг от друга

Активность может находиться в состояниях:

- Active
- Paused
- Stopped
- Dead

Принципы и особенности работы с состояниями Activity:

- На переднем плане: *запущено* или *работает*
- Не в фокусе, но всё ещё показывается пользователю: *видимо*. Это возможно, например, если поверх полноэкранной activity отображается другая, не полноэкранная. Такая activity полностью живая (помнит своё состояние и остаётся присоединена к оконному менеджеру)
- Полностью перекрыта другой activity: *остановлена* или *скрыта*
- Система выбросила activity из памяти (когда остановлено пользователем или просто убив процесс): *сброшено*. Состояние такой активности сбрасывается до начальных значений, когда вновь показывается пользователю



5. Content Provider

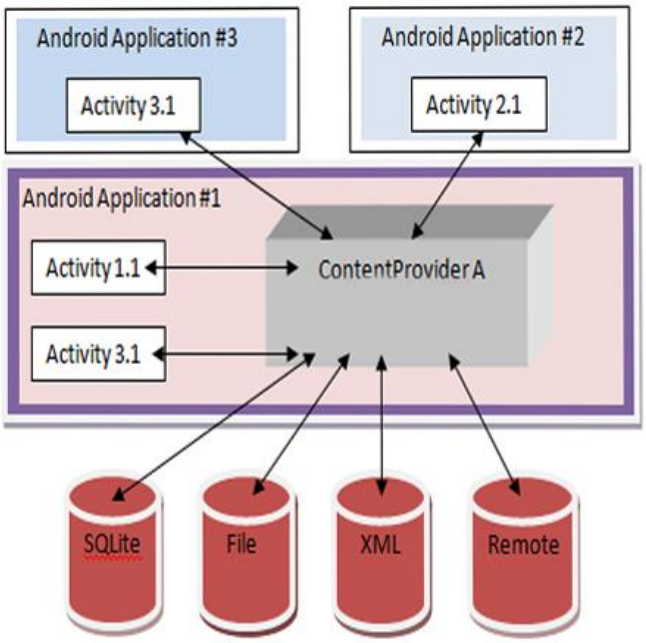
Content Provider – “прослойка” между приложением и хранилищами данных

Контент-провадер занимается управлением распределенным множеством данных приложения

Например: Контент-провайдер в системе Android, управляющий информацией о контактах пользователя

Контент-провайдера необходимы в следующих случаях:

- приложение предоставляет сложные данные или файлы другим приложениям
- приложение позволяет пользователям копировать сложные данные в другие приложения
- приложение предоставляет специальные варианты поиска, используя поисковую платформу (framework)



6. Services в Android, жизненный цикл

Services - это приложения, не имеющие GUI и выполняющиеся в фоновом режиме.

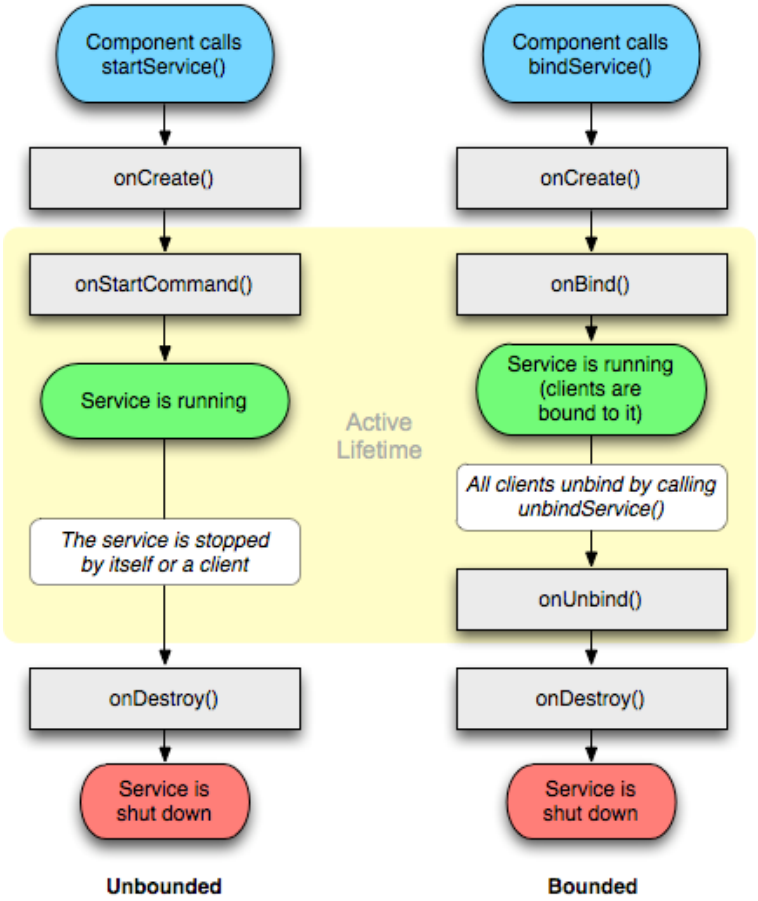
Сервис (Service) - компонент, который работает в фоновом режиме, выполняет длительные по времени операции или работу для удаленных процессов

Примеры сервисов (проверка электронной почты, получение гео-информации, Проигрывание музыки в фоновом режиме).

Сервисы представляют собой особую организацию приложения. В отличие от activity они не требуют наличия визуального интерфейса. Сервисы позволяют выполнять долговременные задачи без вмешательства пользователя.

Все сервисы наследуются от класса **Service** и проходят следующие этапы жизненного цикла:

- Метод onCreate(): вызывается при создании сервиса
- Метод onStartCommand(): вызывается при получении сервисом команды, отправленной с помощью метода startService()
- Метод onBind(): вызывается при закреплении клиента за сервисом с помощью метода bindService()
- Метод onDestroy(): вызывается при завершении работы сервиса



7. Жизнь приложения в ОС Android

Жизненный цикл приложения в операционной системе Android описывает различные состояния, через которые проходит приложение с момента запуска до его завершения. Этот цикл позволяет системе управлять ресурсами и обеспечивать эффективную работу устройства. Вот основные этапы жизненного цикла Android-приложения:

1. Создание (onCreate):

- Этот метод вызывается при первом создании приложения.

- Инициализация основных компонентов приложения, таких как интерфейс пользователя, базовые переменные и структуры данных.

2. Запуск (onStart):

- После onCreate вызывается метод onStart.

- Приложение становится видимым для пользователя, но еще не в фокусе.

3. Возобновление (onResume):

- Приложение становится активным и в фокусе.

- Здесь можно начинать выполнение задач, которые должны продолжаться в активном состоянии.

4. Приостановка (onPause):

- Вызывается, когда приложение теряет фокус, но остается видимым.

- Например, при входящем звонке или открытии другого приложения.

5. Остановка (onStop):

- Вызывается, когда приложение становится невидимым для пользователя.

- Может использоваться для сохранения состояния приложения перед его завершением.

6. Уничтожение (onDestroy):

- Вызывается перед тем, как приложение полностью завершится.

- Здесь можно освобождать ресурсы и завершать необходимые процессы.

7. Перезапуск (onRestart):

- Вызывается, если приложение было остановлено, но теперь снова запускается.

Важно отметить, что Android-система может уничтожить приложение, если она считает, что оно занимает слишком много ресурсов или если пользователь явно завершает его. Поэтому разработчики должны обращаться с ресурсами бережно и правильно обрабатывать жизненный цикл приложения для обеспечения стабильной и эффективной работы.

8. Класс Intent

Intent – это объект, который представляет собой намерение или запрос на выполнение определенного действия. Он используется для связи между компонентами приложения, такими как активности, службы или приемники широковещательных сообщений.

Intent может быть явным (Explicit) или неявным (Implicit). Явный Intent указывает на конкретный компонент приложения, с которым нужно взаимодействовать, указывая его имя или класс. Неявный Intent не указывает конкретный компонент, а описывает действие, которое нужно выполнить, и позволяет системе найти подходящий компонент для его обработки.

Примеры использования Intent:

- ▶ поступление телефонного звонка
- ▶ приход sms-сообщения
- ▶ вставлена SD-карта
- ▶ запущена новая активность

Intent это объект класса Android.Content.Intent. Через этот объект достигается обмен сообщениями между активностями и службами, или их вызов.

Intent можно использовать для:

1. Стартовать Activity
2. Стартовать дочерний Activity
3. Стартовать сервис (Service).

Чтобы создать объект Intent и запустить с помощью него Activity,

нужно в параметрах передать объект-Activity класса, который вызывает и имя

Activity, которое нужно запустить:

```
Intent intent = new Intent(this, ActivityTwo.class);
intent.putExtra("message",message);
startActivity(intent);
```

Из под второй дочерней активности можно получить информацию о записанном главной активностью сообщении:

```
mTextView.setText((String)getIntent().getSerializableExtra("message"));
```

9. Манифест приложения

- Корневой каталог каждого приложения под Android должен содержать файл **AndroidManifest.xml**
- Содержит всю необходимую информацию, используемую системой для запуска и выполнения приложения

Основная информация в манифесте:

- Имя Java пакета приложения
- Описание компонентов приложения
- Определение процессов
- Объявление полномочий, которыми должно обладать приложение для доступа к защищенным частям API и взаимодействия с другими приложениями
- Объявление полномочий, которыми должны обладать другие приложения для взаимодействия с компонентами данного
- Список вспомогательных классов
- Определение минимального уровня Android API для приложения
- Список библиотек связанных с приложением

10. Структура проекта в Android Studio

Структура проекта в Android Studio организована в соответствии с принятыми стандартами для проектов на платформе Android. Вот основные компоненты структуры проекта:

1. app/:

- src/: Каталог, в котором находятся исходные коды вашего приложения. Внутри этого каталога есть подкаталоги для разных версий и типов кода (например, `main/` для основного кода, `androidTest/` для тестов на уровне Android).

- java/: Исходные коды на языке Java или Kotlin. Внутри этого каталога вы обычно организуете свой код в соответствии с пакетами.

- res/: Ресурсы приложения, такие как макеты, изображения, значения строк, цветов и т.д.

- manifest/: Файл манифеста приложения (`AndroidManifest.xml`), в котором указываются основные настройки приложения, разрешения и информация о компонентах приложения.

2. gradle/:

- Файлы конфигурации Gradle для проекта и модулей. Например, `build.gradle` в корневом каталоге проекта и `build.gradle` в каталоге модуля (обычно `app/`).

3. build/:

- Каталог, в который Gradle помещает результаты сборки, такие как APK-файлы.

4. app/build/:

- Каталог с результатами сборки и временными файлами для модуля приложения.

5. .idea/:

- Каталог с настройками проекта для IntelliJ IDEA (базовая IDE, на которой основан Android Studio).

6. gradle/wrapper/:

- Каталог с файлами оболочки Gradle, необходимыми для автоматической установки правильной версии Gradle для вашего проекта.

7. .gitignore/:

- Файл, указывающий системе контроля версий Git, какие файлы и каталоги должны быть проигнорированы при фиксации изменений.

11. Виды диалоговых окон

Диалоговое окно

- Позволяет пользователю получить или ввести информацию
- Работает в модальном режиме

Виды диалоговых окон:

- Класс Dialog и его производные
- Уведомления (notifications)
- Всплывающие подсказки (toasts)

Класс Dialog

- AlertDialog. Может содержать заголовок, до трех кнопок, список выбираемых значений или настраиваемое содержимое
- DatePickerDialog или TimePickerDialog. Позволяет выбрать дату или время
- ProgressDialog. Содержит линейку процесса выполнения какого-то действия

Класс DialogFragment используется в качестве контейнера

Уведомления (notifications)

- Показываются в верхней части экрана
- Могут быть свернуты или развернуты

Всплывающие подсказки (toasts)

- Помогают отобразить обратную связь с действиями пользователя
- Занимают минимум места
- Быстро исчезают
- Могут появляться в любом месте экрана

12. Классы объектов по работе с камерой

Разберемся, какие основные объекты нам понадобятся для вывода изображения с камеры на экран.

Их три: Camera, SurfaceView, SurfaceHolder.

Camera используется, чтобы получить изображение с камеры.

А чтобы это изображение в приложении отобразить, будем использовать SurfaceView. Surface будет означать некий компонент, который отображает изображение с камеры.

Работа с surface ведется не напрямую, а через посредника – SurfaceHolder (далее holder). Именно с этим объектом умеет работать Camera. Также, holder будет сообщать нам о том, что surface готов к работе, изменен или более недоступен.

А если подытожить, то: Camera берет holder и с его помощью выводит изображение на surface.

Основные методы класса Activity:

- onCreate
- onResume
- onPause (release)

Основные методы класса Camera:

- open(int cameraId)
- release()
- startPreview()
- stopPreview()
- setPreviewDisplay(SurfaceHolder holder)
- takePicture(ShutterCallback shutter, PictureCallback raw, PictureCallback jpeg)

13. Сенсорное управление

Большинство устройств на базе Android имеют встроенные датчики, которые измеряют движение, ориентацию и различные условия окружающей среды. Эти датчики способны предоставлять необработанные данные с высокой точностью, и они полезны, если вы хотите отслеживать трехмерное перемещение или позиционирование устройства, или изменения в окружающей среде рядом с устройством.

За работу с сенсорами отвечает класс SensorManager, содержащий несколько констант, которые характеризуют различные аспекты системы датчиков Android

Примеры датчиков:

- TYPE_ACCELEROMETER Измеряет ускорение в пространстве по осям X, Y, Z

- TYPE_GRAVITY Трёхосевой датчик силы тяжести. Как правило, это виртуальный датчик и представляет собой низкочастотный фильтр для показаний, возвращаемых акселерометром

- TYPE_LIGHT Измеряет степень освещённости. Датчик окружающей освещённости, который описывает внешнюю освещённость в люксах

- TYPE_MAGNETIC_FIELD Датчик магнитного поля, определяющий текущие показатели магнитного поля в микротеслах по трём осям.

- TYPE_HEART_BEAT пульс

14. База данных SQLite

SQLite доступен на любом Android-устройстве, его не нужно устанавливать отдельно.

SQLite поддерживает типы TEXT (аналог String в Java), INTEGER (аналог long в Java) и REAL (аналог double в Java). Остальные типы следует конвертировать, прежде чем сохранять в базе данных. SQLite сама по себе не проверяет типы данных, поэтому вы можете записать целое число в колонку, предназначенную для строк, и наоборот.

Работа с базой данных сводится к следующим задачам:

- Создание и открытие базы данных
- Создание таблицы
- Создание интерфейса для вставки данных (insert)
- Создание интерфейса для выполнения запросов (выборка данных)
- Закрытие базы данных

Класс ContentValues Класс ContentValues используется для добавления новых строк в таблицу.

Курсоры В Android запросы к базе данных возвращают объекты класса Cursor. Вместо того чтобы извлекать данные и возвращать копию значений, курсоры ссылаются на результирующий набор исходных данных.

Класс SQLiteOpenHelper: Создание базы данных Библиотека Android содержит абстрактный класс SQLiteOpenHelper, с помощью которого можно создавать, открывать и обновлять базы данных. Это основной класс, с которым вам придётся работать в своих проектах

Класс SQLiteOpenHelper содержит два обязательных абстрактных метода:

- onCreate() — вызывается при первом создании базы данных
- onUpgrade() — вызывается при модификации базы данных

Также остальные методы этого класса:

- onDowngrade(SQLiteDatabase, int, int)
- onOpen(SQLiteDatabase)
- getReadableDatabase()
- getWritableDatabase()

Для управления базой данных SQLite существует класс SQLiteDatabase. В классе SQLiteDatabase определены методы query(), insert(), delete() и update() для чтения, добавления, удаления, изменения данных.

15. Категории сенсоров в Android

Платформа Android поддерживает три широкие категории датчиков:

• Датчики движения
Эти датчики измеряют силы ускорения и вращения по трем осям. К этой категории относятся акселерометры, датчики силы тяжести, гироскопы и датчики вектора вращения.

• Датчики окружающей среды
Эти датчики измеряют различные параметры окружающей среды, такие как температура и давление окружающего воздуха, освещенность и влажность. К этой категории относятся барометры, фотометры и термометры.

• Датчики положения
Эти датчики измеряют физическое положение устройства. К этой категории относятся датчики ориентации и магнитометры.
Можно получить доступ к датчикам, доступным на устройстве, и получать необработанные данные с них с помощью Android sensor framework. Платформа sensor framework предоставляет несколько классов и интерфейсов, которые помогают выполнять широкий спектр задач, связанных с датчиками.

Примеры сенсоров доступны в ответе на вопрос 14.

16. Push-уведомления

Push-уведомления (или уведомления в реальном времени) - это сообщения, которые посылаются приложению на мобильном устройстве, даже если оно не запущено. Они могут содержать текст, изображения, звуки и другие данные, и они служат способом уведомить пользователя о каких-то событиях или действиях, связанных с вашим приложением.

Firebase Cloud Messaging (FCM) - это служба, предоставляемая Google, которая упрощает отправку сообщений на устройства Android, iOS и веб-приложения. Она является частью Firebase, который представляет собой платформу для разработки мобильных и веб-приложений.

Для добавления push-уведомлений в Android-проект, необходимо интегрировать FCM в свое приложение (зарегистрировать его в манифесте)

Чтобы принимать и отправлять сообщения (уведомления), необходимо создать и наследовать класс сервиса FirebaseMessagingService и реализовать его метод onMessageReceived.

Имя пакета: android:name="com.google.firebase.MESSAGING_EVENT"

17. Среда выполнения приложений (виртуальная машина) Dalvik и ART

Dalvik и ART (Android Runtime) - это две различные среды выполнения, которые используются в операционной системе Android для выполнения приложений.

1. Dalvik:

- Архитектура: Dalvik был первоначальной средой выполнения, используемой в Android. Он был разработан Дэном Борхом на основе виртуальной машины Apache Harmony и предназначен для работы в ограниченных ресурсах мобильных устройств.

- Just-In-Time (JIT) компиляция: Dalvik использует JIT-компиляцию, что означает, что байт-код DEX (Dalvik Executable) переводится в машинный код во время выполнения приложения. Это позволяет улучшить производительность, так как код адаптируется под конкретное устройство в процессе работы приложения.

- Ограничения: Несмотря на преимущества JIT-компиляции, Dalvik имеет свои ограничения, такие как высокое потребление энергии и медленная загрузка приложений.

2. ART (Android Runtime):

- Архитектура: ART был представлен в Android 4.4 KitKat и заменяет Dalvik как среду выполнения по умолчанию в новых версиях Android. Он представляет собой движок выполнения приложений, который использует AOT (Ahead-Of-Time) компиляцию.

- Ahead-Of-Time (AOT) компиляция: ART переводит байт-код DEX в машинный код во время установки приложения на устройство, а не во время выполнения. Это снижает накладные расходы на компиляцию во время работы и улучшает производительность приложений в целом.

- Преимущества: ART был разработан с целью улучшения эффективности и производительности приложений. Он также сокращает задержку при запуске приложений и снижает потребление энергии.

- Изменения в структуре файлов: ART использует файлы с расширением `.oat` (сокращение от "optimized ART") вместо `.dex`, которые используются Dalvik.

В целом, ART считается более эффективной и производительной средой выполнения по сравнению с Dalvik. С момента внедрения ART стал стандартной средой выполнения в Android, и большинство современных устройств используют его для выполнения приложений.

Dalvik vs. ART

Dalvik (JIT)	ART (AOT)
Используется меньше постоянной памяти, однако дольше запускаются приложения	Задействуется больше постоянной памяти, однако быстрее открываются программы и меньше используется процессор
Загрузка устройства происходит быстрее, так как кэш приложений создается постепенно	Загрузка устройства происходит медленнее, так как кэш всех приложений создается при включении аппарата
Больше подходит для устройств с маленьким объемом постоянной памяти	Занимает больше постоянной памяти, так как хранит уже скомпилированное приложение вместе с APK
Стабильно и проверено временем - основная виртуальная машина для разработчиков	Эксперимент и новшество - пока сообщество разработчиков не поддерживает технологию в полной мере

18. Интерфейс SensorListener

SensorEventListener` - это интерфейс в Android, предназначенный для прослушивания событий сенсоров (например, акселерометра, гироскопа, магнитометра) на устройстве. Этот интерфейс предоставляет методы, которые вызываются при изменении значений сенсоров.

Вот пример использования `SensorEventListener`:

```
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;


public class SensorActivity extends AppCompatActivity implements
SensorEventListener {


    private SensorManager sensorManager;
    private Sensor accelerometer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sensor);
        // Получение менеджера сенсоров
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        // Получение конкретного сенсора (например, акселерометра)
        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        if (accelerometer != null) {
            // Регистрация слушателя сенсора
            sensorManager.registerListener(this, accelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
        }
```

```
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        // Обработка изменений значений сенсора
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            float x = event.values[0];
            float y = event.values[1];
            float z = event.values[2];

            // Делайте что-то с полученными данными
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Метод вызывается, когда точность сенсора изменяется
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Отмена регистрации слушателя при приостановке активности
        sensorManager.unregisterListener(this);
    }
    @Override
    protected void onResume() {
        super.onResume();
        // Регистрация слушателя при возобновлении активности
        if (accelerometer != null) {
            sensorManager.registerListener(this, accelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
        }
    }
}
```

В данном примере `SensorActivity` реализует `SensorEventListener`. В методе `onCreate`, мы получаем `SensorManager` и регистрируем слушателя (`this`) для конкретного сенсора (в данном случае, акселерометра).

В методах `onSensorChanged` и `onAccuracyChanged` происходит обработка изменений значений сенсора и изменения точности, соответственно. Также, мы регистрируем и отменяем регистрацию слушателя при активации и приостановке активности.

19. Достоинства и недостатки кроссплатформенной разработки

Кроссплатформенная разработка имеет свои преимущества и недостатки. Важно учитывать конкретные требования проекта, цели разработки и потребности в конкретных характеристиках. Вот некоторые основные достоинства и недостатки кроссплатформенной разработки:

Достоинства:

1. Переносимость кода: Одним из главных преимуществ кроссплатформенной разработки является возможность использования большей части кода для разработки приложений под разные платформы. Это сокращает время и усилия, затрачиваемые на разработку и поддержку.

2. Экономия времени и ресурсов: Кроссплатформенные фреймворки (например, React Native, Xamarin, Flutter) позволяют создавать приложения для разных платформ, используя общий код. Это может существенно сэкономить время и ресурсы разработки.

3. Быстрый релиз и обновления: Разработка для нескольких платформ с использованием единого кодовой базы облегчает выпуск обновлений и новых функций одновременно на разных устройствах.

4. Снижение затрат: Кроссплатформенная разработка может быть более экономически эффективной, поскольку разработчики могут использовать свои навыки для создания приложений для разных платформ, без необходимости в глубоких знаниях каждой из них.

5. Единый дизайн: Возможность создавать приложения с единым дизайном и пользовательским интерфейсом для разных платформ, что может способствовать брендированию и узнаваемости.

Недостатки:

1. Ограниченные возможности доступа к аппаратному обеспечению: В некоторых случаях кроссплатформенные фреймворки могут предоставлять ограниченный доступ к аппаратным возможностям устройств, что может быть проблемой для приложений, требующих высокой производительности или специфичных функций.

2. Производительность: На кроссплатформенных решениях может наблюдаться небольшое снижение производительности по сравнению с нативной разработкой, особенно в случаях, когда требуется максимальная оптимизация под конкретную платформу.

3. Зависимость от сторонних инструментов: Разработчики могут столкнуться с ограничениями, связанными с функциональностью и обновлениями фреймворков кроссплатформенной разработки.

4. Ограниченный доступ к новым функциям платформ: Некоторые новые функции, появляющиеся в операционных системах, могут быть доступны только при использовании нативных сред разработки.

20. Достоинства и недостатки нативной разработки

Нативная разработка означает создание приложений с использованием нативных инструментов и языков программирования, предоставляемых платформой для конкретной операционной системы (например, Swift и Objective-C для iOS, Java и Kotlin для Android). Вот несколько достоинств и недостатков нативной разработки:

Достоинства:

1. Полный доступ к функциональности устройства: Нативные приложения имеют полный доступ к аппаратному обеспечению устройства и платформенным API. Это позволяет максимально использовать возможности устройства и создавать более производительные приложения.

2. Высокая производительность: Нативные приложения обычно обеспечивают более высокую производительность по сравнению с кроссплатформенными аналогами, так как они оптимизированы под конкретную платформу.

3. Лучшая интеграция с экосистемой платформы: Нативные приложения лучше интегрированы в экосистему платформы, что может включать в себя более удобные инструменты разработки, стандарты дизайна и лучшую поддержку для новых функций операционной системы.

4. Поддержка последних технологических новшеств: Разработчики нативных приложений обычно получают более быстрый доступ к последним технологическим новшествам и обновлениям платформы.

5. Лучший пользовательский опыт: Нативные приложения обеспечивают лучший пользовательский опыт благодаря более тщательной оптимизации и интеграции с платформой.

Недостатки:

1. Большие затраты на разработку и поддержку: Разработка нативных приложений для разных платформ может потребовать больше времени и ресурсов, так как требуется создание и поддержка отдельного кода для каждой платформы.

2. Ограниченная переносимость кода: Код, написанный для одной платформы, в большинстве случаев не может быть использован без изменений на другой платформе.

3. Сложности в многозадачности: Нативные приложения могут столкнуться с трудностями при реализации многозадачности на некоторых платформах, особенно когда речь идет о многозадачности на разных устройствах.

4. Длительные сроки выпуска: Из-за необходимости разработки отдельных версий для разных платформ, выпуск нативных приложений может занять больше времени по сравнению с кроссплатформенной разработкой.

5. Ограниченные навыки разработчика: Разработчики нативных приложений должны обладать навыками в конкретных языках программирования и средах разработки для каждой платформы.

Выбор между нативной и кроссплатформенной разработкой зависит от конкретных требований проекта, бюджета, приоритетов по производительности и стратегии разработки.

21. Каково одно из главных преимуществ платформы Android?

Одним из главных преимуществ платформы Android является ее открытость и широкая распространенность. Вот несколько ключевых аспектов, которые отражают это преимущество:

1. Открытость и свобода разработки:

- Android является открытой платформой с открытым исходным кодом (основан на проекте AOSP - Android Open Source Project). Это означает, что разработчики могут свободно изучать и изменять исходный код операционной системы, что обеспечивает большую гибкость в создании настраиваемых решений и адаптаций.

2. Широкая распространенность и разнообразие устройств:

- Android работает на огромном разнообразии устройств от различных производителей. Это включает смартфоны, планшеты, смарт-телевизоры, умные часы, автомобильные системы и другие устройства.

3. Гибкость в распространении приложений:

- Разработчики Android могут распространять свои приложения не только через официальный магазин приложений Google Play, но и через альтернативные каналы и даже непосредственно с веб-сайтов.

4. Операционная система для разных целевых аудиторий:

- Android предназначен для широкого спектра потребителей, от обычных пользователей до разработчиков и корпоративных клиентов.

5. Обширные возможности настройки пользовательского интерфейса:

- Android предоставляет разработчикам широкие возможности настройки пользовательского интерфейса. Это включает в себя создание виджетов, кастомизацию элементов управления и даже замену стандартного домашнего экрана.

22. Какие инструменты входят в состав Android SDK?

Android SDK (Software Development Kit) представляет собой набор инструментов и библиотек для разработки приложений под операционную систему Android. В состав Android SDK входят различные компоненты, обеспечивающие функциональность разработки, тестирования и отладки приложений.

Некоторые из основных инструментов в составе Android SDK включают:

1. Android Studio:

- Официальная интегрированная среда разработки (IDE) для создания приложений под Android. Android Studio обеспечивает мощные инструменты для написания кода, дизайна пользовательского интерфейса, отладки и профилирования приложений.

2. SDK Manager:

- Инструмент для управления установкой и обновлением компонентов Android SDK. С его помощью разработчики могут загружать различные версии Android API, системные образы, и другие инструменты для разработки под разные версии Android.

3. AVD Manager (Android Virtual Device):

- Инструмент для создания, настройки и управления виртуальными устройствами Android, которые используются для тестирования приложений в эмулированной среде.

4. Android Debug Bridge (ADB):

- Командная утилита для взаимодействия с устройствами Android. ADB предоставляет возможность установки и удаления приложений, копирования файлов, выполнения команд в оболочке устройства и многого другого.

5. Android Emulator:

- Встроенный эмулятор Android, который позволяет разработчикам запускать и тестировать свои приложения на различных версиях Android, разрешениях экрана и устройствах.

6. Build Tools:

- Набор инструментов для сборки, компиляции и упаковки приложений. К ним относятся, например, `aapt` (Android Asset Packaging Tool) для упаковки ресурсов, `dx` для преобразования Java-байткода в формат Dalvik Executable (DEX) и другие.

7. Android Support Library (теперь часть AndroidX):

- Набор библиотек и инструментов для обеспечения совместимости приложений с более старыми версиями Android, а также предоставления новых возможностей на старых платформах.

23. Какие наиболее распространенные виды датчиков используются в мобильных устройствах

• Датчики движения

Эти датчики измеряют силы ускорения и вращения по трем осям. К этой категории относятся акселерометры, датчики силы тяжести, гироскопы и датчики вектора вращения.

• Датчики окружающей среды

Эти датчики измеряют различные параметры окружающей среды, такие как температура и давление окружающего воздуха, освещенность и влажность. К этой категории относятся барометры, фотометры и термометры.

• Датчики положения

Эти датчики измеряют физическое положение устройства. К этой категории относятся датчики ориентации и магнитометры.

Можно получить доступ к датчикам, доступным на устройстве, и получать необработанные данные с них с помощью Android Sensor Framework.

Платформа Android Sensor Framework предоставляет несколько классов и интерфейсов, которые

помогают выполнять широкий спектр задач, связанных с датчиками.

Примеры сенсоров доступны в ответе на вопрос 14 и в следующем вопросе.

24. Приведите хотя бы по одному примеру использования для каждого датчика

Примеры использования различных видов датчиков (сенсоров):

Датчики движения:

Акселерометр: Автоматическая смена ориентации экрана в приложениях, например, когда пользователь поворачивает устройство.

Гироскоп: Игры, в которых управление осуществляется вращением устройства, например, гонки или летательные симуляторы.

Датчики окружающей среды:

Датчик освещенности: Автоматическое регулирование яркости экрана в зависимости от окружающего уровня освещенности.

Датчик температуры: Приложения метеослужбы, предоставляющие текущую температуру в местоположении пользователя.

Датчики положения:

Магнитометр (компас): Приложения навигации, которые определяют направление движения пользователя на карте.

Датчик близости: Автоматическое отключение экрана во время телефонного разговора при прикладывании устройства к уху.

Датчик шагов: Приложения для фитнеса, которые отслеживают количество пройденных шагов и рассчитывают расстояние.

25. В каких случаях вызываются методы onSensorChanged и onAccuracyChanged?

Методы onSensorChanged и onAccuracyChanged являются частью интерфейса SensorEventListener, который используется для обработки событий от датчиков в Android.

Оба эти метода обязательны для реализации, когда вы регистрируете слушатель датчика (registerListener) и используете SensorEventListener. Они обеспечивают обратную связь от датчиков и позволяют вашему приложению реагировать на изменения в данных и точности датчиков.

Методы интерфейса SensorEventListener:

- onSensorChanged:

Вызывается, когда данные от датчика изменяются. Метод передает объект SensorEvent, который содержит новые данные от датчика. Этот метод обычно используется для обновления интерфейса пользователя или выполнения определенных действий на основе данных от датчика. Например, при изменении позиции устройства с использованием акселерометра, в этот метод можно включить код для обновления интерфейса или изменения состояния приложения.

- onAccuracyChanged:

Вызывается, когда точность данных от датчика изменяется. Метод передает объект Sensor, представляющий конкретный датчик, и значение accuracy, которое представляет текущую точность данных. Этот метод может быть использован для управления поведением приложения в зависимости от точности данных датчика. Например, если точность данных ухудшается, приложение может принять решение о приостановке определенных операций или предостеречь пользователя.

26. Опишите три возможных случая поведения приложения, при работе с базой данных SQLite

При работе с базой данных SQLite в Android приложения могут столкнуться с различными сценариями и поведениями. Ниже описаны три возможных случая:

1. Успешная операция записи в базу данных:

- *Сценарий:* Приложение выполняет операцию вставки (insert), обновления (update) или удаления (delete) данных в базе данных SQLite.

- *Поведение:* Операция выполняется успешно, и база данных обновляется соответствующим образом. Например, если это вставка новой записи, она добавляется в таблицу. Если это обновление, существующие данные обновляются.

2. Ошибка при выполнении SQL-запроса:

- *Сценарий:* Приложение выполняет SQL-запрос (например, через метод `execSQL` или `rawQuery`), и в запросе содержится ошибка синтаксиса или другая проблема.

- *Поведение:* В случае ошибки выполнения SQL-запроса, будет сгенерировано исключение, такое как `SQLiteException`. Приложение должно обрабатывать исключение, предоставлять пользователю информацию об ошибке или производить другие действия в зависимости от контекста. Важно логгировать подобные ошибки для последующего анализа и улучшения приложения.

3. Конфликт транзакций при многопоточной работе:

- *Сценарий:* В приложении используется многопоточность, и несколько потоков одновременно пытаются изменить данные в базе данных без использования механизма транзакций.

- *Поведение:* В многопоточной среде возможны конфликты транзакций, что может привести к некорректному состоянию базы данных. Для предотвращения таких конфликтов рекомендуется использовать транзакции с методами `beginTransaction()`, `setTransactionSuccessful()` и `endTransaction()`. Это позволяет атомарно выполнить несколько операций и обеспечивает целостность данных. В случае некорректного использования транзакций, приложение может столкнуться с ситуациями, такими как потеря данных или блокировки базы данных.

27. Назовите по крайней мере три метода класса SQLiteDatabase, которые используются для работы с БД

1. `execSQL(String sql)`:

- Вызывается для выполнения SQL-запросов, таких как создание таблицы, вставка данных или изменение схемы базы данных.

2. `insert(String table, String nullColumnHack, ContentValues values)`:

- Используется для вставки новых записей в таблицу базы данных. Принимает имя таблицы, столбец для вставки (обычно `null`), и объект `ContentValues`, содержащий значения для вставки.

3. `query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)`:

- Предоставляет гибкий механизм для выполнения SQL-запросов с возвращением результатов. Принимает параметры, такие как имена столбцов, условия выборки, аргументы выборки, группировка, сортировка и ограничение.

4. `update(String table, ContentValues values, String whereClause, String[] whereArgs)`:

- Используется для обновления записей в таблице базы данных. Принимает имя таблицы, объект `ContentValues` с новыми значениями, условия обновления и аргументы условий.

5. `delete(String table, String whereClause, String[] whereArgs)`:

- Удаляет записи из таблицы базы данных. Принимает имя таблицы, условия удаления и аргументы условий.

7. `rawQuery(String sql, String[] selectionArgs)`:

- Выполняет произвольный SQL-запрос и возвращает результат в виде объекта `Cursor`. Этот метод предоставляет большую гибкость в написании собственных SQL-запросов.

28. Назовите основные два способа получения доступа к Canvas

Для рисования используется объект Canvas. Canvas является лишь инструментом для рисования. А весь результат сохраняется на Bitmap. Мы не можем напрямую попросить Bitmap нарисовать на себе линию или круг, поэтому Canvas выступает посредником и помогает нам нарисовать то, что нужно.

Основные способы получения доступа к объекту Canvas:

Первый способ – через наследника View класса. Нам нужно просто переопределить его метод onDraw и он даст нам доступ к Canvas. Кода тут минимум и все предельно просто. Но есть недостаток – все рисование выполняется в основном потоке. Этот вариант подойдет, если у вас статичная картинка или не слишком динамичная анимация.

Второй способ – через SurfaceView. Этот способ подойдет, если планируете рисовать что-то тяжелое и динамичное. Под рисование здесь будет выделен отдельный поток. Это уже немного посложнее в реализации, чем первый способ.

Для рисования используются методы draw*. Если посмотреть их в официальной документации можно обратить внимание, что одним из их параметров является объект Paint. В этом объекте задаются графические характеристики рисования. Т.е. можно считать, что это кисть, которой будут рисоваться ваши фигуры. Через него вы сообщаете канве цвет и толщину линии для рисования.

29. Что такое индекс и ID при обработке касаний экрана? Для чего они применяются?

Система умеет обрабатывать до 10 касаний включительно. При этом учитывайте, что далеко не все устройства поддерживают 10 касаний.

Теперь надо понять, как отличить - для какого именно пальца сработали события ACTION_POINTER_DOWN и ACTION_ POINTER_UP. Для этого используются две системы нумерации – индекс и ID. Индекс – порядковый номер пальца. Не привязан к пальцу – один палец может иметь разные индексы в течение одного касания. ID - привязан к пальцу от начала до конца касания.

Если совершить несколько касаний по экрану – то можно считать, что все текущие касания хранятся в неком массиве. И ID пальцев - это их идентификаторы (значения элементов массива), а индексы пальцев - индексы этого массива касаний.

30. Опишите основные типы событий при работе с касаниями

Для View-компонентов можно использовать OnClickListener для обработки коротких нажатий. В данной лабораторной работе попробуем ловить касания и перемещения пальца по компоненту. Они состоят из трех типов событий:

- нажатие (палец прикоснулся к экрану)

- движение (палец движется по экрану)

- отпускание (палец оторвался от экрана)

Все эти события мы сможем ловить в обработчике OnTouchListener, который присвоим для View-компонента. Этот обработчик дает нам объект MotionEvent, из которого мы извлекаем тип события и координаты

Рассмотрим систему событий для мультитача. К событиям ACTION_DOWN, ACTION_MOVE и ACTION_UP добавляются ACTION_POINTER_DOWN и ACTION_POINTER_UP.

ACTION_DOWN – срабатывает при касании первого пальца

ACTION_POINTER_DOWN – срабатывает при касании каждого последующего пальца

ACTION_MOVE - срабатывает при любом движении

ACTION_ POINTER_UP – срабатывает при отпускании каждого пальца кроме последнего

ACTION_ UP – срабатывает при отпускании последнего пальца

31. Назовите основные компоненты для работы с камерой устройства Android

Базовые объекты для вывода изображения с камеры Android на экран устройства:

Их три: Camera, SurfaceView, SurfaceHolder.

Camera используется, чтобы получить изображение с камеры.

А чтобы это изображение в приложении отобразить, будем использовать SurfaceView. Surface будет означать некий компонент, который отображает изображение с камеры.

Работа с surface ведется не напрямую, а через посредника – SurfaceHolder (далее holder). Именно с этим объектом умеет работать Camera. Также, holder

будет сообщать нам о том, что surface готов к работе, изменен или более недоступен.

А если подытожить, то: Camera берет holder и с его помощью выводит изображение на surface.

Различные компоненты для работы с камерой устройства Android:

Для работы с камерой устройства Android используются различные компоненты и API. Вот основные компоненты, которые часто используются при работе с камерой:

1. Camera API (старый подход):

- `Camera`: Этот класс представляет собой аппаратную камеру устройства и обеспечивает методы для управления камерой, такие как открытие, получение изображений и т.д.

2. Camera2 API (новый подход):

- `CameraManager`: Предоставляет доступ к камерам устройства, работающим с новым Camera2 API. Позволяет получать информацию о камерах и их характеристиках.

- `CameraCaptureSession`: Используется для создания сеансов захвата, где вы можете отправлять запросы на захват изображений камерой.

- `CameraDevice`: Представляет собой конкретное устройство камеры, с которым вы взаимодействуете при использовании Camera2 API.

- `CameraCaptureRequest`: Используется для создания запросов на захват камерой. Определяет параметры захвата, такие как разрешение, ориентация и др.

3. Intent для захвата изображений:

- `Intent(MediaStore.ACTION_IMAGE_CAPTURE)`: Используется для запроса системного приложения камеры для захвата изображения. Результат можно получить в `onActivityResult()`.

4. SurfaceView или TextureView:

- `SurfaceView` или `TextureView` используются для предоставления видеопотока или изображения с камеры на пользовательский интерфейс.

5. Обработчики (Callbacks):

- `CameraCaptureSession.CaptureCallback`: Обратные вызовы, позволяющие отслеживать состояние захвата, такие как успешное завершение, ошибки и т.д.

6. MediaRecorder (для видеозаписи):

- `MediaRecorder`: Используется для записи видео с камеры. Предоставляет методы для установки параметров записи, начала и завершения записи.

Эти компоненты обеспечивают широкие возможности для работы с камерой устройства Android, включая захват изображений, запись видео и другие сценарии использования.

32. Приведите названия событий, используемых для работы с камерой Android

Использование событий камеры в Android приложении

В onCreate настраиваем Activity так, чтобы оно было без заголовка и в полный экран. Затем мы определяем surface, получаем его holder и устанавливаем его тип =SURFACE_TYPE_PUSH_BUFFERS (настройка типа нужна только в Android версии ниже 3.0).

Далее для holder создаем callback объект HolderCallback через который holder будет сообщать нам о состояниях surface.

В onResume получаем доступ к камере, используя метод open. На вход передаем id камеры, если их несколько (задняя и передняя). Этот метод доступен с API level 9. Также существует метод open без требования id на вход. Он даст доступ к задней камере. Он доступен и в более ранних версиях.

После этого вызываем метод setPreviewSize, в котором настраиваем размер surface. Его подробно обсудим ниже.

В onPause освобождаем камеру методом release, чтобы другие приложения могли ее использовать.

Класс HolderCallback, реализует интерфейс SurfaceHolder.Callback. Через него holder сообщает нам о состоянии surface.

В нем три метода:

- surfaceCreated – surface создан. Мы можем дать камере объект holder с помощью метода setPreviewDisplay и начать транслировать изображение методом startPreview.

- surfaceChanged – был изменен формат или размер surface. В этом случае мы останавливаем просмотр (stopPreview), настраиваем камеру с учетом поворота устройства (setCameraDisplayOrientation, подробности ниже), и снова запускаем просмотр.

- surfaceDestroyed – surface более недоступен. Не используем этот метод.

33. Как определить, есть ли камера на устройстве

Как определить, есть ли камера в устройстве? Об этом сообщит конструкция context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)

Получить id камеры, можно используя метод getNumberOfCameras (доступен с API Level 9). Он вернет нам некое кол-во камер N, которые доступны на устройстве. Соответственно их ID будут 0, 1, …, N-1. По этому id уже получаете CameraInfo и определяете, что это за камера. Метод open может вернуть Exception при запуске если по каким-то причинам не удалось получить доступ к камере. Имеет смысл это обрабатывать и выдавать сообщение пользователю, а не вылетать с ошибкой.

34. Понятие Activity, Fragment, View, Intent

Класс Activity

Activity – основная единица графического интерфейса (аналог окна или экранной формы)

Активность – это видимая часть приложения (экран, окно, форма), отвечает за отображение графического интерфейса пользователя

При этом приложение может иметь несколько активностей. Активности приложения не зависят друг от друга

Класс Fragment

В Android Fragment представляет собой модуль поведения или часть интерфейса пользователя внутри Activity. Они были введены в Android для облегчения создания адаптивных пользовательских интерфейсов, которые могут адекватно реагировать на различные экраны и ориентации устройства.

Жизненный цикл Fragment:

Fragment имеет свой жизненный цикл, аналогичный жизненному циклу Activity. Он включает методы, такие как onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy(), и другие. Управление жизненным циклом позволяет фрагментам адекватно реагировать на изменения состояния.

Класс View

- Основной строительный блок для компонентов пользовательского интерфейса (UI)
- Определяет прямоугольную область экрана и отвечает за прорисовку и обработку событий

Класс Intent

Intent – это объект, который представляет собой намерение или запрос на выполнение определенного действия. Он используется для связи между компонентами приложения, такими как активности, службы или приемники широковещательных сообщений.

35. Понятие Service, Content Provider

Класс Service

Services - это приложения, не имеющие GUI и выполняющиеся в фоновом режиме.

Сервис (Service) - компонент, который работает в фоновом режиме, выполняет длительные по времени операции или работу для удаленных процессов

Примеры сервисов (проверка электронной почты, получение гео-информации, Проигрывание музыки в фоновом режиме).

Класс Content Provider

Content Provider – “прослойка” между приложением и хранилищами данных

Контент-провадер занимается управлением распределенным множеством данных приложения

Например: Контент-провайдер в системе Android, управляющий информацией о контактах пользователя

Контент-провайдера необходимы в следующих случаях:

- приложение предоставляет сложные данные или файлы другим приложениям
- приложение позволяет пользователям копировать сложные данные в другие приложения
- приложение предоставляет специальные варианты поиска, используя поисковую платформу (framework)

36. Архитектуры Android приложений: MVC. Преимущества и недостатки

MVC (Model-View-Controller) - это популярная архитектурная модель, которая применяется для построения программных приложений. Давайте рассмотрим преимущества и недостатки применения архитектуры MVC в контексте Android-приложений:

Преимущества MVC в Android:

1. Разделение ответственности: MVC обеспечивает разделение приложения на три основные компоненты: модель (Model), представление (View) и контроллер (Controller). Это позволяет легко разделять ответственность между разными частями приложения.

2. Легкость тестирования: Компоненты архитектуры MVC могут быть тестированы независимо друг от друга. Модели и контроллеры могут быть протестированы отдельно от пользовательского интерфейса (View), что облегчает юнит-тестирование.

3. Гибкость и расширяемость: За счет разделения компонентов, MVC обеспечивает гибкость и улучшает расширяемость приложения. Модификации в одной части системы редко затрагивают другие части.

4. Поддержка множественных представлений: Модель может поддерживать несколько представлений, что позволяет одной модели использоваться различными способами в зависимости от необходимых пользовательских интерфейсов.

Недостатки MVC в Android:

1. Сложность управления состоянием: Управление состоянием может стать сложным, особенно в случае больших приложений. Контроллеры могут становиться трудными для поддержки, когда приложение разрастается.

2. Возможность появления "раздутых" контроллеров: В процессе развития приложения контроллеры могут становиться "толстыми", накапливая логику, которая лучше была бы разделена на более мелкие компоненты.

3. Сложности синхронизации представления и модели: В некоторых случаях может возникнуть сложность с синхронизацией данных между моделью и

представлением. Например, изменения в модели должны быть отражены в соответствующем представлении.

4. Избыточное количество связей: MVC может привести к большому количеству связей между компонентами, особенно при больших проектах, что усложняет понимание структуры приложения.

В целом, архитектура MVC может быть полезной в разработке Android-приложений, но важно знать, что существуют и другие архитектурные подходы (например, MVVM, MVP), которые также могут быть рассмотрены в зависимости от конкретных потребностей и особенностей проекта.

37. Что такое гибридные мобильные приложения?

Гибридные мобильные приложения представляют собой приложения, созданные с использованием комбинации веб-технологий (HTML, CSS, JavaScript) и нативных элементов. Эти приложения обычно упаковываются в контейнер, который может быть установлен и запущен на мобильных устройствах, а затем взаимодействовать с нативными API для доступа к устройственным функциям.

Основные черты гибридных мобильных приложений включают:

1. Использование веб-технологий: Гибридные приложения используют веб-технологии для построения пользовательского интерфейса и бизнес-логики. Обычно это HTML для разметки, CSS для стилей и JavaScript для взаимодействия с пользователями и обработки логики приложения.

2. Использование некоторых нативных элементов: Гибридные приложения могут использовать фреймворки, такие как Apache Cordova (также известный как PhoneGap), Ionic, Xamarin и другие, чтобы упаковать веб-код в нативный контейнер. Этот контейнер предоставляет мост между веб-кодом и нативными API устройства, что позволяет приложению использовать функции, такие как камера, геолокация, контакты и другие.

3. Кросс-платформенность: Гибридные приложения обычно написаны один раз, а затем могут быть развернуты на различных мобильных платформах, таких как Android, iOS и других. Это обеспечивает кросс-платформенность и упрощает процесс разработки для разных операционных систем.

4. Доступ к функциям устройства: Через нативные элементы, гибридные приложения получают доступ к устройственным функциям и API, что позволяет использовать возможности устройства, такие как камера, геолокация, уведомления и т.д.

5. Относительная простота разработки: Разработка гибридных приложений может быть более простой, чем полностью нативная разработка, так как они используют знакомые веб-технологии и могут быть созданы одним кодом для нескольких платформ.

38. Преимущества и недостатки гибридных мобильных приложений?

Преимущества гибридных мобильных приложений:

1. Кроссплатформенность: Одним из основных преимуществ гибридных приложений является их кросс-платформенность. Однажды написанный код может быть использован на нескольких платформах, таких как Android и iOS, что упрощает процесс разработки и снижает затраты.

2. Быстрое развертывание: Гибридные приложения могут быть быстро развернуты на многих платформах, так как они используют общий код и не требуют полной перекомпиляции для каждой платформы.

3. Использование веб-технологий: Разработчики могут использовать знакомые веб-технологии, такие как HTML, CSS и JavaScript, что облегчает создание интерфейса и бизнес-логики.

4. Применение изменений без обновления приложения: Внесение изменений в гибридные приложения может осуществляться без необходимости обновления через магазин приложений. Это может быть полезно для быстрой коррекции ошибок или внесения небольших изменений.

5. Доступ к функциям устройства: Гибридные приложения могут получить доступ к устройственным функциям через нативные мосты, что позволяет использовать камеру, геолокацию, контакты и другие возможности.

Недостатки гибридных мобильных приложений:

1. Производительность: Гибридные приложения могут иметь более низкую производительность по сравнению с полностью нативными, особенно при выполнении сложных операций или взаимодействии с графикой.

2. Ограниченный доступ к нативным API: В некоторых случаях доступ к нативным API может быть ограничен, что может вызвать сложности при реализации некоторых функций или использовании последних возможностей платформы. Зависимость от обновлений фреймворка:

- Гибридные приложения могут зависеть от фреймворков, таких как Cordova или Ionic, и обновление этих фреймворков может требовать изменений в коде приложения.

4. Ограниченные возможности адаптации интерфейса: Некоторые гибридные фреймворки могут иметь ограниченные возможности адаптации интерфейса под стандарты каждой платформы, что может создавать ощущение "нативного" приложения.

5. Зависимость от сторонних библиотек: Возможно использование сторонних библиотек и фреймворков, что может привести к зависимости от их обновлений и поддержки.

39. Сравнение REST API, и GraphQL

REST API (Representational State Transfer) и GraphQL представляют два различных подхода к построению и взаимодействию с веб-сервисами. Вот сравнение между ними:

RESTful API — это интерфейс,используемые двумя компьютерными системами для безопасного обмена информацией через Интернет

Особенности:

1. Структура данных: REST API обычно возвращает предопределенные структуры данных (например, JSON или XML), и клиенты получают только те данные, которые были запрошены.

2. Множество конечных точек (Endpoints): REST API предоставляет различные конечные точки для разных ресурсов и операций. Каждая конечная точка представляет собой определенный путь к ресурсу.

3. Избыточные запросы: Клиенты часто сталкиваются с проблемой получения избыточных данных, поскольку REST API возвращает все поля ресурса, даже если клиенту нужны только некоторые из них.

4. Количество запросов: Для получения необходимых данных, клиентам может потребоваться сделать несколько последовательных запросов к различным конечным точкам.

5. Обновление данных: Обновление данных осуществляется через операции HTTP, такие как PUT, POST, PATCH, DELETE.

GraphQL — язык запросов данных и язык манипулирования данными с открытым исходным кодом для построения веб ориентированных программных интерфейсов. GraphQL был разработан как внутренний проект компании Facebook в 2012 году, а позднее в 2015 году был выпущен публично.

Особенности:

1. Гибкость запросов: GraphQL позволяет клиентам запрашивать только те данные, которые им нужны, и получать их в едином запросе. Клиент определяет формат ответа.

2. Единая конечная точка: GraphQL использует единую конечную точку для всех запросов, что делает его более простым в использовании и поддержке.

3. Минимизация избыточных данных: Клиент может точно указать, какие поля ему нужны, и избежать получения избыточных данных.

4. Графовая модель данных: GraphQL представляет данные в виде графа, где клиенты могут определять связи между ресурсами и получать данные, связанные с определенным контекстом.

5. Реальное время (Real-time): GraphQL обеспечивает возможность работы в режиме реального времени через подписки (subscriptions), что позволяет клиентам получать обновления данных при изменении на сервере.

6. Мутации: В GraphQL операции мутаций используются для изменения данных, аналогично HTTP-методам в REST, но с более гибкими возможностями.

Общие соображения по использованию обоих подходов:

- Сложность реализации является проблемой языка GraphQL:

  - GraphQL может быть сложнее в реализации на сервере из-за необходимости обрабатывать динамические запросы. REST более прямолинеен и проще в этом отношении.

- Переиспользование кода является проблемой использования подхода, предлагаемого REST API:

  - REST API часто обеспечивает повторное использование кода через использование стандартизированных методов HTTP, в то время как GraphQL может потребовать более сложной логики запроса на сервере.

Оба подхода имеют свои преимущества и недостатки, и выбор между ними зависит от конкретных потребностей проекта, требований к клиентам и предпочтений разработчиков.