Практическая работа 2

Методические указания

Поддержка различных устройств. Жизненный цикл явлений.

Устройства, работающие под Android выпускаются во всем мире и имеют разные формы и размеры. При таком огромном ассортименте вы можете выпускать ваше приложение для огромной аудитории. Для того, чтобы добиться большого успеха среди пользователей Android, ваше приложение должно быть адаптировано для различных конфигураций. Важными параметрами, которые вы должны учитывать, являются различные языки, размеры экранов и установленные версии Android.

В данном материле мы рассмотрим как использовать базовые возможности платформы для управления ресурсами для оптимизации вашего приложения под различные Android-совместимые устройства, и разместить его в единственном APK-файле.

Поддержка различных языков

Размещение строковых констант в отдельном файле считается хорошим тоном. В каждом проекте Andoid для строковых констант существует специальная директория.

Если вы создаете проекты используя инструменты Android SDK (смотрите раздел Создание проекта), то директория res/ создается автоматически в корневой директории проекта. Также для каждого из типов ресурсов создаются собственные поддиректории, а так же несколько стандартных файлов. Например файл res/values/strings.xml, в котором хранятся строковые константы.

Создание директорий для поддержки различных языков

Для поддержки нескольких языков, необходимо создать в директории res несколько дополнительных директорий values, содержащих в имени ISO коды языков. К примеру, директория values-ru будет включать в себя ресурсы для поддержки русского языка.

Android загружает нужные ресурсы в соответствии с настройками локализации устройства, на котором было запущено приложение. Подробную информацию смотрите в разделе Использование ресурсов.

Если вы решили включить поддержку нескольких языков, создайте необходимые поддиректории и файлы строковых ресурсов. К примеру таким образом:

```
MyProject/
    res/
       values/
           strings.xml
       values-es/
           strings.xml
       values-fr/
           strings.xml
```

Добавьте строковые константы в каждый из файлов.

При запуске Android самостоятельно выберет подходящий файл в соответствии с настройками локализации устройства.

Посмотрите пример содержания файлов строковых ресурсов для различных языков:

Английский (по умолчанию), /values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">My Application</string>
    <string name="hello_world">Hello World!</string>
</resources>
```

Русский, /values-es/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title"><span lang="ru-RU">Мое приложение</span></string>
    <string name="hello_world"><span lang="ru-RU">Привет мир</span>!</string>
</resources>
```

Французский, /values-fr/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
    <string name="hello_world">Bonjour le monde !</string>
</resources>
```

Примечание: вы можете использовать спецификатор локализации (как и любой другой спецификатор настроек) для любого типа ресурса. Например вы можете создать различные значки приложения для различных языков.

Использование строковых ресурсов

Вы можете ссылаться на строковые ресурсы из программного кода и других XML файлов, используя название ресурса, которое задается в свойстве name элемента <string>.

В программном коде вы можете ссылаться на строковый ресурс, используя синтаксис R.string.<string_name>. Есть множество способов использования строковых ресурсов.

Например:

```
// Получение строки из ресурса приложения
String hello = <code class="western"><a href="developer.android.com/reference/android/content/Con

// <span lang="ru-RU">Установка строки из ресурса в качестве значения текстового элемента</span>
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

Вы можете ссылаться на строковый ресурс из других XML файлов, используя синтаксис @string/<string_name>, всякий раз, когда требуется строковое значение для свойства.

Например:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

Поддержка устройств с различными экранами

Экраны Android устройств различаются по двум основным параметрам: размер и разрешение. Вы должны быть готовы к тому, что ваше приложение может быть установлено на устройствах с различными размерами и разрешениями экрана. Чтобы оптимизировать приложение под различные размеры и разрешения, оно должно содержать разные ресурсы под каждый вид экрана.

- Существует четыре обобщенных размера: маленький (small), нормальный(normal), большой(large), очень большой(x-large).
- Также существует четыре вида разрешений: низкое (low), среднее (mdpi), высокое (hdpi), очень высокое (xhdpi).

Чтобы использовать разную разметку и изображения для разных экранов, необходимо размещать данные ресурсы в различные директории, точно также, как мы это делали со строками для различных языков.

Также следует помнить об ориентации экрана (альбомная (landscape) или портретная (portrait)), она так же считается отдельным размером и для многих приложений необходимо отдельно оптимизировать разметку для двух ориентаций.

Создание различной разметки

Чтобы оптимизировать пользовательский интерфейс под различные размеры экрана, необходимо создать собственные файлы разметки для каждого из размеров, которые вы хотите поддерживать. Каждый файл разметки должен быть сохранен в соответствующей директории, название которой заканчивается строкой -<screen_size>. Например, файл разметки для больших экранов должен быть сохранен в директории res/layout-large/.

Примечание: Android автоматически рассчитывает размеры вашего макета для нужного экрана, поэтому вам не нужно заботиться об абсолютных размерах элементов. Вместо этого сосредоточьтесь на структуре разметки, так как она очень влияет на удобство использования вашего приложения. Обратите особое внимание на то, как элементы располагаются относительно друг друга.

Например, если проект включает в себя стандартную разметку и разметку для больших экранов, структура директорий будет выглядеть так:

```
MyProject/
    res/
        layout/
            main.xml
        layout-large/
            main.xml
```

Имена файлов должны быть абсолютно одинаковыми, а их содержимое должно быть оптимизировано для соответствующих размеров экрана.

В программном коде обращение к файлу разметки остается прежним:

```
@Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
}
```

Система автоматически загрузит нужный файл разметки из подходящей директории, основываясь на размере экрана устройства, на котором запущено приложение.

Рассмотрим еще один пример проекта с поддержкой различной ориентации экрана:

```
MyProject/
    res/
        layout/
            main.xml
        layout-land/
            main.xml
```

По умолчанию файл layout/main.xml используется для потретной ориентации.

Если же вы хотите создать отдельную разметку для альбомной ориентации на больших экранах, используйте одновременно два спецификатора large и land:

```
MyProject/
    res/
        layout/             # <span lang="ru-RU">по умолчанию </span>(<span lang="ru-RU">потрет
            main.xml
        layout-land/        # <span lang="ru-RU">альбомная</span>
            main.xml
        layout-large/       # <span lang="ru-RU">для большого экрана</span> (<span lang="ru-RU">
            main.xml
        layout-large-land/  # <span lang="ru-RU">для большого экрана альбомная</span>
            main.xml
```

Примечание: Android 3.2 и выше поддерживает расширенные методы указания размеров экрана. Это позволяет создавать разметку основываясь на минимальной ширине и высоте экрана с использованием точек (псевдо-пикселей, на зависящих от разрешения экрана). В данном материале не рассматривается данная технология.

Использование различных изображений

Всегда создавайте растровые изображения для каждого из основных разрешений: низкого, среднего, высокого и очень высокого. Это позволит добиться отличного качества и производительности на устройствах с любым разрешением.

Чтобы создать такие изображения, создайте базовый вариант в векторном формате, а затем экспортируйте его в растровый формат для каждого разрешения, используя следующую шкалу размеров:

- xhdpi: 2.0
- hdpi: 1.5
- mdpi: 1.0 (базовый размер)
- ldpi: 0.75

Это означает, что если вы создали картинку размером 200×200 пикселей для xhdpi устройств, то для остальных устройств будут такие размеры: 150x150 для hdpi, 100×100 для mdpi и 75x75 для ldpi устройств.

После этого разместите файлы в соответствующих директориях:

```
MyProject/
    res/
        drawable-xhdpi/
            awesomeimage.png
        drawable-hdpi/
            awesomeimage.png
        drawable-mdpi/
            awesomeimage.png
        drawable-ldpi/
            awesomeimage.png
```

Каждый раз, когда вы будете обращаться к ресурсу @drawable/awesomeimage, система будет автоматически выбирать подходящий файл в зависимости от разрешения экрана.

Примечание: Не всегда существует необходимость создавать ldpi ресурсы. Когда вы создаете hdpi ресурс, система может автоматически пересчитать размеры и использовать его для ldpi экранов.

Поддержка различных версий Android

В то время, как последние версии Andoid часто добавляют в API великолепные возможности, вы должны продолжать поддерживать старые версии Android до тех пор, пока большинство устройств не получат обновление. В этом уроке мы рассмотрим как используя широкие возможности последних API также продолжать поддержку старых версий Android.

Диаграмма используемых версий регулярно обновляется и показывает соотношение установленных версий Android. Диаграмма строится на основе статистики из Google Play Store. В целом, хорошей практикой является поддержка 90% активных устройств при нацеливании на последние версии.

Совет: Для того, чтобы использовать лучшие возможности и создать лучшую функциональность в таком разнообразии версий, вы должны использовать библиотеку поддержки в вашем приложении до тех пор, пока решите больше не поддерживать старые устройства.

Указание минимальной и целевой версии API

В файле AndroidManifest.xml описывается какие версии Android поддерживает приложение. Конкретно указываются атрибуты minSdkVersion и targetSdkVersion для элемента <uses-sdk>, которые указывают на минимальную версию с которым ваше приложение совместимо и максимальную версию, на которой вы разрабатывали и тестировали приложение.

Например:

```
<manifest xmlns:android="schemas.android.com/apk/res/android" ... >
    <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15" />
    ...
</manifest>
```

После выхода новой версии некоторые стили и поведение системы может измениться. Чтобы разрешить приложению использовать эти новшества, а также гарантировать, что приложение использует нужные стили на каждом из устройств, необходимо устанавливать значение targetSdkVersion равным последней доступной версии Android.

Получение версии Android во время выполнения приложения

Android хранит специальный код для каждой из версий в виде константы в классе Build.

Приведенный ниже код задает условие, чтобы функции, доступные в более высоких версиях API выполнялись, только если данный API доступен на устройстве.

```
private void setUpActionBar() {
    // <span lang="ru-RU">Проверяем, что на устройстве установлена </span><span lang="en-US">Honey
    <span lang="ru-RU">//спокойно использовать </span><span lang="en-US">API </span><span lang="ru
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
}
```

Примечание:При обработке XML ресурсов, Android игнорирует атрибуты, которые не поддерживаются на текущем устройстве. Вы можете безопасно использовать XML атрибуты, доступные только в новых версиях не беспокоясь о том, что на старых устройствах это вызовет ошибку. Например, если вы установите targetSdkVersion=”11”, то панель инструментов автоматически добавится на устройствах с Android 3.0 и выше. Для добавления кнопок на панель вам необходимо указать атрибут android:showAsAction=”ifRoom” для пункта меню. Вы можете добавить данный атрибут в XML файл для всех версий устройств, поскольку в старых

версиях атрибут showAsAction будет просто проигнорирован. Другими словами, вам не нужно создавать в этом случае отдельную версию в директории res/menu-v11.

Использование встроенных тем и стилей

Android содержит стандартные темы интерфейса, которые позволяют приложению выглядеть как одно целое с системой. Эти темы легко использовать в приложении, указав их в файле манифеста. При использовании встроенных тем, приложение будет «следить за модой» и выглядеть по новому с каждым новым выпуском Android.

Чтобы сделать явление похожим на диалоговое окно:

```
1 <activity android:theme="@android:style/Theme.Dialog">
```

Чтобы применить вашу тему, описанную в файле /res/values/styles.xml:

```
1 <activity android:theme="@style/CustomTheme">
```

Чтобы применить тему ко всему приложению целиком (ко всем явлениям), добавьте атрибут android:theme к элементу <application>

```
1 <application android:theme="@style/CustomTheme">
```

Жизненный цикл явлений

Когда пользователь просматривает приложение, выходит из него и снова открывает, экземпляры Activity вашего приложения переключаются между различными состояниями их жизненного цикла. К примеру, когда вы впервые запускаете приложение, оно занимает экран устройства и получает фокус пользователя. В это время система вызывает некоторые методы управления жизненным циклом явлений, отрисовывает интерфейс пользователя и другие компонунты. Если вы открыли другое явление или другое приложение, система выполняет другие методы управления жизненным циклом, теперь первое явление уйдет в режим ожидания (в котором оно невидимо, но экземпляр класса Activity и его состояние остаются неизменными).

С помощью функций обратного вызова жизненного цикла, вы можете задать поведение явления при различных действиях пользователя, таких как закрытие и повторное открытие. Например ваш видео-плеер может останавливать воспроизведение и отключать интернет-соединение, если пользователь переключился на другое приложение. Когда пользователь вновь откроет приложение, можно начать воспроизведение с того же места.

В материале рассматриваются важные функции управления жизненным циклом, которые есть у каждого экземпляра Activity. Вы узнаете как использовать эти функции, чтобы явление не потребляло ресурсы, когда оно не используется.

Запуск явлений

В отличие от других парадигм программирования, в которых приложения запускаются при помощи метода main(), Android использует функции обратного вызова экземпляра Activity, соответствующие этапам его жизненного цикла. Методы вызываются в различной последовательности при запуске явления и при его завершении.

В данном уроке мы рассмотрим наиболее важные из этих методов и покажем как создается экземпляр явления.

Функции обратного вызова жизненного цикла

Во время существования явления, система вызывает методы один за другим, подобно движению к вершине пирамиды. Каждая стадия жизненного цикла соответствует отдельному шагу. Вершина пирамиды – это точка, в которой явление видимо и пользователь может с ним взаимодействовать.

Когда пользователь покидает явление, система вызывает другие методы, которые спускают явление к основанию пирамиды. В некоторых случаях явление может спустить только на один шаг и перейти в режим ожидания (например если пользователь переключился на другое приложение), а затем вернуться обратно в вершину (если пользователь заново открыл явление) и начать выполнение с места прерывания.

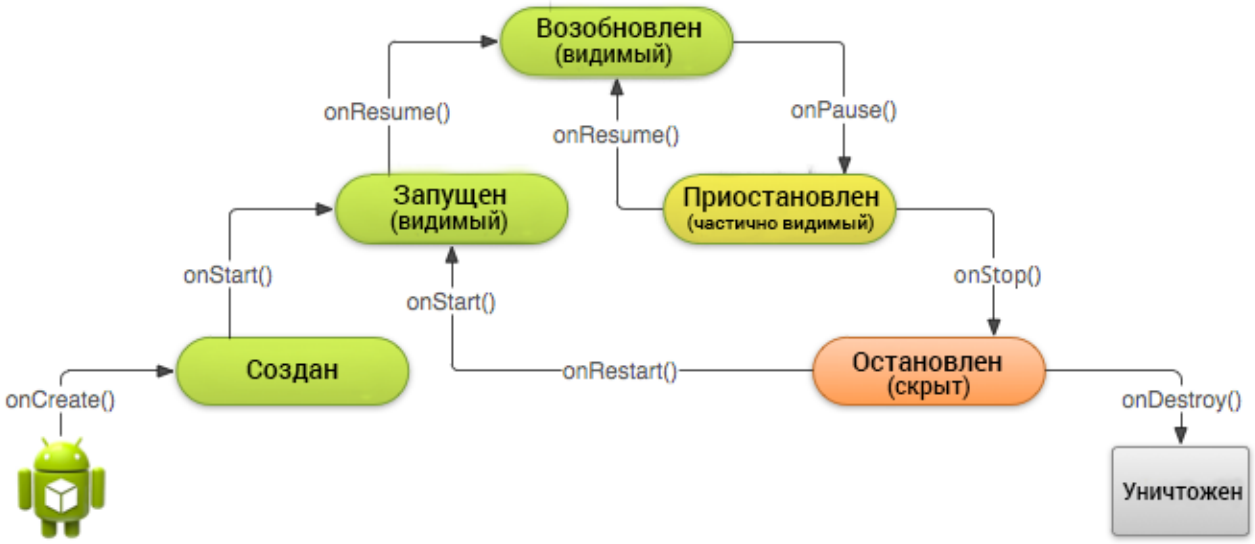

Рисунок 1.Упрощенная иллюстрация жизненного цикла в виде пирамиды.

Необязательно объявлять все методы жизненного цикла – это зависит от сложности вашего явления, но очень важно понимать когда вызывается каждый из них. Их грамотное использование гарантируют правильную работу приложения в разных случаях, включая:

- Предотвращение аварийного завершения приложения при поступлении звонка или переключении на другое приложение.
- Предотвращение расхода ресурсов системы, когда приложение неактивно.
- Продолжение работы с того же места, при повторном открытии приложения.
- Продолжение работы при повороте устройства в альбомную или потретную ориентацию.

Далее мы рассмотрим различные ситуации, в которых явления переключается между состояниями, показанными на рисунке 1. Однако, только три состояния могут быть статическими, в которых явление может находиться длительное время:

Возобновлено
: В этом состоянии явление находится на переднем плане и позволяет пользователю с ним взаимодействовать. (Его еще называют “запущенное состояние”)

Приостановлено
: В данном состоянии явление частично скрыто другим явлением, которое имеет полупрозрачный фон или не закрывает весь экран

целиком. В этом состоянии явление не может обрабатывать запросы пользователя и выполнять какой-либо программный код.

Остановлено

В данном состоянии явление полностью скрыто и не видимо для пользователя. Все данные при этом сохраняются, однако явление не может выполнять какой-либо код.

Остальные состояния (создано и запущено) промежуточные и система быстро перемещается на следующие. Так после вызова onCreate() быстрое вызывается onStart(), который также быстро переходит на onResume().

Всё это базовые элементы жизненного цикла явления. Далее мы подробнее рассмотрим некоторые особенности его поведения.

Объявление главного явления

Когда пользователь щелкает по иконке вашего приложения, система вызывает метод onCreate() явления, которое вы объявили главным. Это явление, которое является точкой входа в пользовательский интерфейс приложения.

Вы можете указать какое из явлений будет главным в файле манифеста AndroidManifest.xml, который находится в корневой директории проекта.

В файле манифеста главное явление должно содержать раздел <intent-filter>, включающий действие MAIN и категорию LAUNCHER. Например:

```
<activity android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Примечание:Если вы используете Android Studio, данный фильтр будет автоматически прописан в файле манифеста.

Если один из элементов MAIN в качестве действия или LAUNCHER в качестве категории не будет указан, иконка вашего приложения не появится в списке программ.

Создание экземпляра явления

Часто приложения состоят из нескольких явлений, для выполнения различных действий. Система всегда создает экземпляр любого явления, вызывая метод onCreate().

В методе onCreate() описываются начальные действия явления, которые выполняются однократно во время всего жизненного цикла, к примеру создание пользовательского интерфейса и инициализация переменных класса.

Ниже приведен пример реализации метода onCreate(), в котором подключается и настраивается XML разметка интерфейса пользователя и определяются некоторые переменные:

```
TextView mTextView; // Переменная для хранения текстового поля разметки

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Установка разметки интерфейса
    // Разметка должна находиться в файле res/layout/main_activity.xml
    setContentView(R.layout.main_activity);

    // Инициализация textView
    mTextView = (TextView) findViewById(R.id.text_message);

    // Проверяем, что приложение запущено на версии Honeycomb или выше,
    //чтобы использовать API панели инструментов
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        //Указываем не использовать иконку приложения в качестве кнопки "назад"
        ActionBar actionBar = getActionBar();
        actionBar.setHomeButtonEnabled(false);
    }
}
```

Внимание:Используйте SDK_INT только на Android 2.0 (API 5) и выше. Использование в более старых версиях могут вызывать исключение.

После выполнения onCreate() система вызывает методы onStart() и onResume().Помните, что явление никогда не задерживается в состоянии **Создано** или **Запущено.** Технически явление становится видимым уже при вызове onStart(), однако состояние быстро меняется на Возобновлено и явление будет находиться в этом состоянии до тех пор, пока какое-либо событие (например телефонный вызов или отключение экрана устройства) не заставит его измениться.

В следующих материалах мы рассмотрим чем могут быть полезны методы onStart() и onResume().

Примечание: метод onCreate() содержит параметр savedInstanceState, который обсуждается в материале Пересоздание явления.

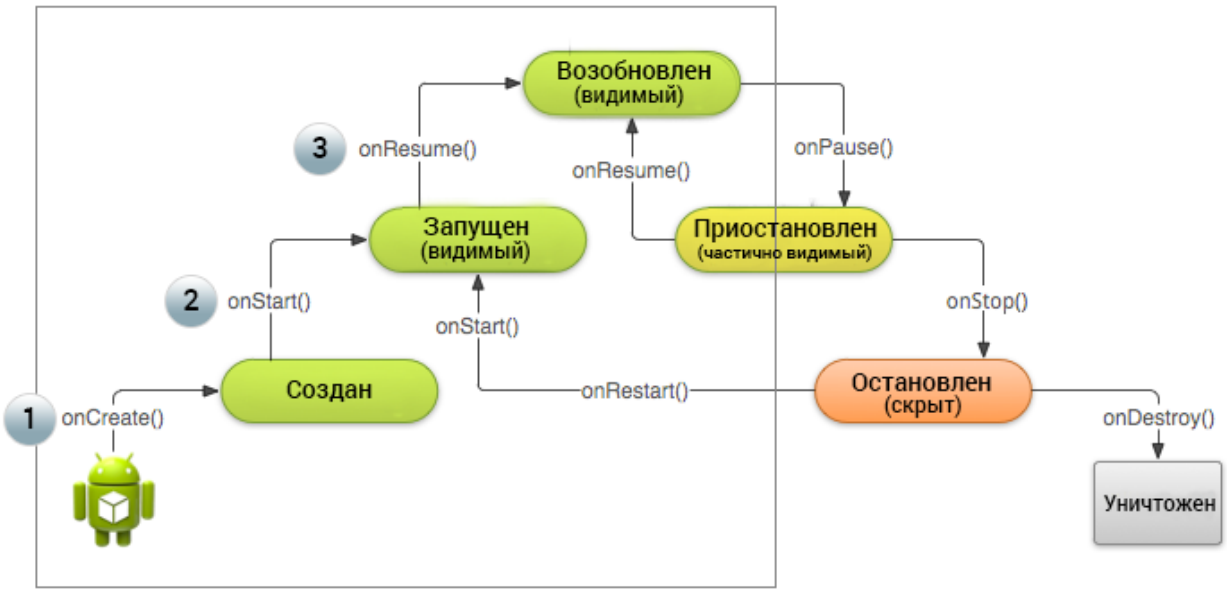

Рисунок 2.Схема вызова методов при создании экземпляра явления. После выполнения последовательности вызовов, явление будет находиться в состоянии **Возобновлено** и может взаимодействовать с пользователем.

Уничтожение явления

Жизнь явления начинается с вызова метода onCreate(), а уничтожение явления сопровождается вызовом метода onDestroy(). Система вызывает этот метод последним перед удалением явления из оперативной памяти.

В большинстве приложений не обязательно реализовывать метод onDestroy(), поскольку все объекты уничтожаемого класса также будут уничтожены. Однако, если ваше явление создает в методе onCreate() дополнительные процессы, потоки или ресурсы, использующие память, вы должны обязательно закрыть их в методе onDestroy().

```
@Override
public void onDestroy() {
    super.onDestroy();  // Всегда вызывайте метод базового класса

    // Остановка трассировки, которая была запущена в onCreate()
    android.os.Debug.stopMethodTracing();
}
```

Примечание:перед вызовом onDestroy(), система всегда вызывает onPause() и onStop(), за исключением случая, когда был вызван finish() внутри метода onCreate(). Такой подход применяется например в том случае, когда явление является временным и используется для запуска другого явления.

Приостановка и возобновление явлений

Во время использования приложения, явление может перекрываться другими визуальными компонентами, из-за чего происходит его **приостановка**. К примеру, при открытии диалога явление приостанавливается. Явление может оставаться частично видимым, однако будет оставаться приостановленным до тех пор, пока не получит фокус.

Если же явление скрыто полностью и невидимо, оно переходит в состояние **остановлено**.

Когда явление переходит в приостановленное состояние, система вызывает метод onPause(), который позволяет вам остановить действия, которые не должны выполняться в этом состоянии (например воспроизведение видео) или сохранить информацию о текущем состоянии явления, чтобы при возобновлении пользователь мог продолжить работу с того же места. Когда явление вновь становится активным, система вызывает метод onResume().

Примечание:Вызов метода onPause() может означать, что явление приостановлено на мгновение и пользователь сейчас вернет ему фокус. Однако это может также служить первым сигналом закрытия явления.

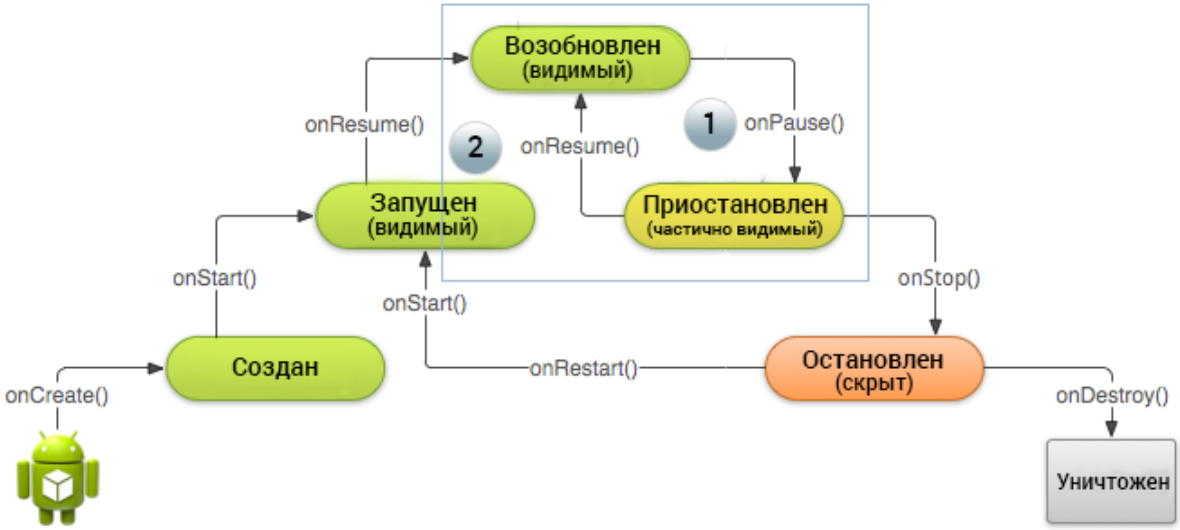

Рисунок 1.Если полупрозрачное явление перекроет ваше, система вызовет onPause() и ваше явление будет приостановлено (1). При возвращении фокуса, система вызовет onResume() и явление станет возобновленным (2).

Приостановка вашего явления

Технически вызов onPause() может означать, что явление все еще частично видимо, но чаще всего это указывает на то, что пользователь закрыл явление

и скоро оно перейдет в состояние **Остановлено**. Поэтому обычно onPause() используется в следующих случаях:

- Остановка анимации или действий, активно использующих процессор.
- Сохранение данных, которые пользователь ожидает увидеть при повторном открытии явления (например незаконченное письмо).
- Освобождение системных ресурсов, таких как получатели широковещательных сообщений, дескрипторы датчиков (например GPS) и других ресурсов, которые могут расходовать заряд батареи и в которых приостановленное явление не нуждается.

Пример применения onPause() в приложении, использующем камеру:

```
@Override
public void onPause() {
    super.onPause();  // Всегда сначала вызывайте метод базового класса

    // Освобождаем камеру, поскольку она может понадобиться другому приложению
    if (mCamera != null) {
        mCamera.release()
        mCamera = null;
    }
}
```

Если говорить в целом, вы **не должны** использовать метод onPause() для помещения данных в постоянное хранилище. Делайте это только в том случае, когда уверены, что пользователи ждут от явления автоматического сохранения данных (например черновиков писем). Однако избегайте выполнение в onPause() сложных операций, таких как запись в базу данных, поскольку это может замедлить открытие следующего явления(выполняйте затратные операции внутри onStop()).

Примечание:Экземпляр класса Activity продолжает храниться в память во время приостановки. Вам не нужно заново создавать и инициализировать используемые компоненты.

Возобновление вашего явления

При возобновлении явления система вызывает метод onResume().

Важно помнить, что этот метод система вызывает каждый раз при возвращении фокуса явлению, в том числе при первом запуске приложения. Поэтому в методе onResume() вы должны инициализировать те компоненты, которые освободили в методе onPause(), и которые могли измениться за время пока явление не было возобновлено.

Следующий пример применения onResume() сочетается с примером onPause(), который мы рассмотрели выше. Здесь мы заново инициализируем камеру, которая была освобождена во время паузы явления:

```java
@Override
public void onResume() {
    super.onResume();  // Всегда сначала вызывайте метод базового класса

    // Занимаем камеру как только явление возобновлено
    if (mCamera == null) {
        initializeCamera(); // Инициализация камеры
    }
}
```

Остановка и перезапуск явлений

Остановка и **перезапуск** явлений, это важные процессы жизненного цикла, которые гарантируют пользователю, что их данные не потеряются. Вот небольшие примеры остановки и перезапуска явлений:

- Пользователь открыл окно “последние приложения” и переключился на другое приложение. Явление, которое на этот момент было активным **остановится**. Когда пользователь снова запустит приложение (неважно из окна последних приложений или с рабочего стола), явление **перезапустится**.
- Пользователь запустил другое явление вашего приложения. Текущее явление **остановится.** Когда пользователь нажмет кнопку назад, явление **перезапустится.**
- Телефонный вызов заставит явление **остановиться.**

Класс Activity предоставляет два метода onStop() и onRestart(), которые позволяют описать действия при **остановке** и **перезапуске** явления. В отличие от **приостановленного** состояния, в котором интерфейс явления может быть частично видим, **остановка** гарантирует, что интерфейс больше не видим и активно другое приложение (или другое явление текущего приложения).

Примечание:поскольку после остановки система хранит экземпляр Activity в памяти, возможно вам не придется в каждом проект использовать методыonStop(), onRestart() и onStart(). Большинство явлений довольно простые и прекрасно останавливаются и перезапускаются без вмешательства с вашей стороны. Чаще вам будет необходим метод onPause().

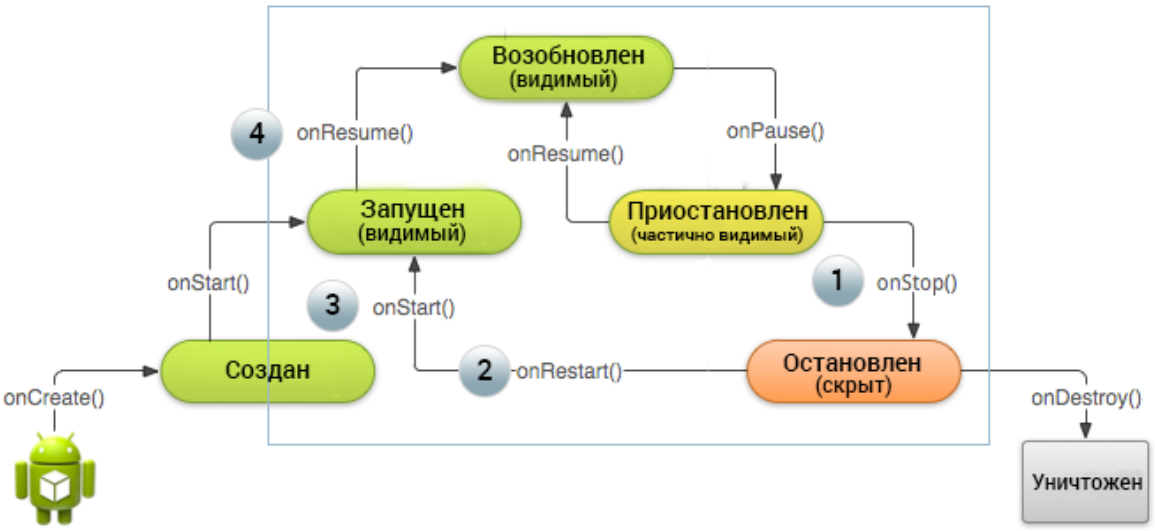

Рисунок 1.Когда пользователь покидает явление, система вызывает `onStop()` для его остановки (1). При повторном открытии явления, система вызывает `onRestart()` (2), `onStart()` (3) и `onResume()` (4). Помните, что назависимо от причин остановки явления, система всегда вызывает `onPause()` перед вызовом `onStop()`.

Остановка вашего Явления

Когда явление получает вызов onStop(), оно уже невидимо пользователю и вы должны освободить все неиспользуемые ресурсы. Система может уничтожить экземпляр остановленного явления, чтобы освободить память. В критичных случаях система может убить процесс вашего приложения без вызова onDestroy(), поэтому очень важно использовать onStop() для освобождения ресурсов, пожирающих память.

Хотя перед вызовом onStop() вызывается метод onPause(), для выполнения сложных операций, требующих интенсивной работы процессора (таких как запись в базу данных), вы должны использовать onStop().

Ниже показан пример метода onStop(), который сохраняет черновик записи в постоянное хранилище:

```
@Override
protected void onStop() {
    super.onStop();  // Всегда вызывайте сначала метод базового класса

    // Сохраняем текущие данные, поскольку явление останавливается
    ContentValues values = new ContentValues();
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());

    getContentResolver().update(
            mUri,    // URI для обновления записи
            values,  // Карта колонок и значений
            null,    // Не использовать критерии отбора.
            null     // Не использовать условия отбора.
            );
}
```

Объект явления типа Activity хранится в памяти после его остановки и вызывается при возобновлении. Вам не нужно заново инициализировать компоненты, которые были созданы во время методов, ведущих к возобновленному состоянию. Система также сохраняет состояние каждого элемента разметки. Если пользователь ввел строку в текстовое поле, его содержание сохранится и вам не надо об этом заботиться.

Примечание: Даже если система уничтожила ваше явление, пока оно было остановлено, состояние элементов разметки (объектов типа View, такие как тестовое поле EditText) сохраняется в объекте типа Bundle(пары ключ-значение) и восстанавливаются, когда пользователь возвращается к экземпляру явления. В следующем материале мы рассмотрим подробнее использование объектов Bundle для сохранения данных в случае уничтожения и пересоздания явлений.

Запуск и перезапуск явления

Когда вы заново активируете явление после его остановки, вызывается метод onRestart(). Система также вызывает метод onStart(), который срабатывает каждый раз, когда приложение становится видимым(неважно было оно запущено впервые или перезапущено). В то же время метод onRestart() вызывается только когда приложение возвращается из остановленного состояния. В нем вы можете выполнять действия, для явлений, которые были **остановлены**, но **не уничтожены**.

Очень редко приходится использовать метод onRestart() для восстановления состояния явлений, поэтому нет общепринятых правил по использованию этого метода. Однако, поскольку onStop() полностью освобождает все ресурсы явления, вам необходимо заново их инициализировать при перезапуске. Но вы делаете это так же при первоначальном запуске явления, поэтому в подавляющем большинстве случаев вы будете использовать для этого метод onStart(), тем более, что он вызывается как при первоначальном запуске приложения, так и при перезапуске.

Поскольку пользователь может надолго закрыть ваше приложение, метод onStart() подходит лучше всего для проверки существования всех необходимых обектов:

```java
@Override
protected void onStart() {
    super.onStart();  // Всегда сначала вызывайте метод базового класса

    // Явление было запущено или перезапущено, поэтому здесь мы убеждаемся, что GPS включен
    LocationManager locationManager =
            (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    boolean gpsEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

    if (!gpsEnabled) {
        //Если GPS отключен, создать в этом месте диалог и использовать
        //Intent с действием android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS
        //для открытия окна настроек GPS
    }
}

@Override
protected void onRestart() {
    super.onRestart();  // Всегда вызывайте сначала метод базового класса

    //А этот код выполнится только если явление было перезапущено
}
```

Если система уничтожила экземпляр явления, будет вызван метод onDestroy(). Поскольку в большинстве случаев ресурсы освобождаются в методе onStop(), большинству приложений не придется использовать onDestroy(). Данный метод это последний шанс очистить ресурсы, пожирающие память. Убедитесь, что вы уничтожили все дополнительные процессы и потоки

Пересоздание явлений

Сейчас мы рассмотрим случай уничтожения явлений в ходе обычной эксплуатации приложения, например при нажатии пользователем кнопки **назад** или при вызове метода finish(). Система может уничтожить явление в случае, если оно остановлено и не используется долгое время или если другому приложению не хватает памяти.

Если явление уничтожено из-за нажатия кнопки **назад** или явление само себя завершило, система удалит экземпляр Activity навсегда. Однако, если система уничтожит явление из-за системных ограничений (в обход нормальному функционированию приложения), при его создании данные, описывающее состояние явления будут восстановлены. Данные, которые система использует для восстановления предыдущего состояния называются “состояние экземпляра” и являются коллекцией элементов ключ-значение, хранящихся в объекте типа Bundle.

Внимание: ваше явление уничтожается и создается заново при каждом повороте экрана. Система делает это, поскольку конфигурация экрана меняется и ваше явление может потребовать загрузки альтернативных ресурсов(например разметки).

По умолчанию, система использует экземпляр Bundle для сохранения информации о каждом объекте типа View в разметке явления. В общем, если ваше явление было уничтожено и пересоздано, состояние разметки автоматически восстановится, не требуя вашего вмешательства. Однако, ваше явление может иметь информацию, которую вы не хотите сохранять.

Примечание:Для того, чтобы система восстановила состояние View элементов явления, каждый визуальный компонент типаView должен иметь уникальный идентификатор, заданный с помощью атрибута android:id.

Чтобы хранить дополнительные данные о состоянии явления, необходимо переопределить метод обратного вызова onSaveInstanceState(). Система вызывает данный метод когда пользователь покидает явление и передает ему объект типа Bundle, который будет сохранен в том случае, если явление будет неожиданно уничтожено. Если позже системе понадобится пересоздать экземпляр явления, она передаст данный Bundle объект в методы onRestoreInstanceState() и onCreate().

Рисунок 1.Перед остановкой явления система вызывает `onSaveInstanceState()` (1), в котором вы можете указать дополнительные данные для восстановления состояния. Если явление уничтожено и должно быть пересоздано, система передает данные состояния экземпляра, объявленные в точке (1), в методы `onCreate()` (2) и `onRestoreInstanceState()` (3).

Сохранение состояния экземпляра явлений

Перед завершением явления система вызывает метод onSaveInstanceState(), в котором явление сохраняет состояние в виде пар ключ-значение. По умолчанию данный метод сохраняет информацию о состоянии элементов разметки, таких как текстовые поля или позиция скролла у списка.

Для сохранения дополнительной информации необходимо переопределить метод onSaveInstanceState() и добавить нужные пары ключ-значение в объект типа Bundle. Например:

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Сохраняем текущее состояние пользователя в игре
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Всегда вызываем метод базового класса, чтобы сохранить информацию об элементах разметки
    super.onSaveInstanceState(savedInstanceState);
}
```

Внимание:Всегда вызывайте метод onSaveInstanceState() базового класса, поскольку он содержит код для сохранения состояния иерархии разметки.

Восстановление состояния экземпляра явлений

Если явление пересоздано после уничтожения, вы можете восстановить его предыдущее состояние из объекта типа Bundle, который система передает в методы onCreate() и onRestoreInstanceState().

Поскольку метод onCreate() вызывается и при первом запуске явления и после его пересоздания, необходимо проверять объект Bundle на null, прежде чем попытаться его прочитать. Если он равен null, система создаст новый экземпляр класса Activity, иначе восстановит его в предыдущее состояние.

Пример восстановления данных состояния в методе onCreate():

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Всегда сначала вызывайте метод базового класса

    // Проверяем, существует ли предыдущее состояние
    if (savedInstanceState != null) {
        // Восстанавливаем данные предыдущего состояния
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Инициализируем значениями по умолчанию
    }
}
```

Вместо восстановления данных в методе onCreate() вы можете переопределить метод onRestoreInstanceState(), который система вызывает после метода onStart(). Метод onRestoreInstance() система вызывает только в том случае, если есть данные о предыдущем состоянии явления и нет необходимости проверять их наличие внутри метода:

```java
public void onRestoreInstanceState(Bundle savedInstanceState) {
    //Всегда вызывайте метод базового класса, чтобы восстановить состояние элементов разметки
    super.onRestoreInstanceState(savedInstanceState);

    // Восстановление сохраненных пользовательских данных
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```

Внимание:Всегда вызывайте метод onRestoreInstanceState() базового класса для того, чтобы восстановить состояние элементов разметки.

Задание

1. Поддержка различных языков. Создать директории для поддержки различных языков. Обеспечить использование строковых ресурсов.
2. Поддержка устройств с различными экранами. Обеспечить создание различной разметки. Обеспечить использование различных изображений.
3. Поддержка различных версий Android. Указать минимальную и целевую версии API. Обеспечить получение версии Android во время выполнения приложения.
4. Обеспечить использование встроенных тем и стилей
5. Жизненный цикл явлений. Обеспечить запуск явлений. Обеспечить функции обратного вызова жизненного цикла. Обеспечить объявление главного явления. Обеспечить создание экземпляра явления. Обеспечить приостановку и возобновление явлений. Обеспечить остановку, запуск и перезапуск явлений. Обеспечить пересоздание явлений. Сохранить состояния экземпляра явлений. Восстановить состояния экземпляра явлений.