Практическая работа. Жизненный цикл Activity.

Активности образуют основу любого Android-приложения. Ранее вы видели, как создавать активности и как использовать активности для взаимодействия с пользователем. Но если вы еще не знакомы с жизненным циклом активностей, некоторые аспекты их поведения могут вас застать врасплох. В этой работе вы узнаете, что происходит при создании или уничтожении активностей и к каким неожиданным последствиям это может привести. Вы узнаете, как управлять их поведениями, когда они становятся видимыми или скрываются. Вы научитесь сохранять и восстанавливать состояние активности, когда это потребуется.

Рассмотрим некоторые типичные причины нарушения работоспособности ваших приложений и разберемся, как исправить их при помощи методов жизненного цикла активностей. Эти функции будут рассматриваться на примере приложения-секундомера.

Приложение Stopwatch

Приложение Stopwatch содержит четыре представления: надпись с прошедшим временем, кнопку Start для запуска секундомера, кнопку Pause для его приостановки и кнопку Reset, которая снова обнуляет показания таймера.

Создание нового проекта

Создайте новый проект Android Studio по той же схеме, которая использовалась в предыдущих главах. Выберите вариант Empty Activity, введите имя «Stopwatch» с именем пакета «com.hfad.stopwatch» и подтвердите место сохранения по умолчанию. Убедитесь в том, что выбран язык Kotlin и минимальный уровень SDK API 29, чтобы приложение работало на большинстве устройств Android.

Добавление строковых ресурсов для надписей

Макет Stopwatch включает три кнопки с надписями Start, Pause и Reset. Мы добавим эти кнопки в файл strings.xml в виде строковых ресурсов, чтобы приложение могло загружать их значения во время выполнения. Чтобы добавить строковые ресурсы, откройте файл strings.xml в папке app/src/main/res/values и обновите его (изменения выделены жирным шрифтом):

```
<resources>
 <string name="app_name">Stopwatch</string>
 <string name="start">Start</string>
 <string name="pause">Pause</string>
 <string name="reset">Reset</string>
</resources>
```

Макет включает представление Chronometer

Макет состоит из трех кнопок и представления, в котором выводятся показания секундомера в секундах. Для вывода показаний будет использоваться хронометр. Хронометр — разновидность надписи, которая выполняет функции простого таймера. Хронометр содержит встроенные методы,

которые могут использоваться для запуска и остановки отсчета времени, и выводит прошедшее время. Для включения хронометра в макет используется элемент `<Chronometer>`. Например, следующий код добавляет представление Chronometer с идентификатором «stopwatch», размер которого определяется размером содержимого:

```
<Chronometer
 android:id="@+id/stopwatch"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />
```

Полный код `activity_main.xml`

Полный код activity_main.xml содержит представление Chronometer и три кнопки. Обновите файл activity_main.xml, чтобы он содержал приведенный ниже код:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:orientation="vertical"
 android:gravity="center_horizontal"
 tools:context=".MainActivity">
 <Chronometer
 android:id="@+id/stopwatch"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="56sp" />
 <Button
 android:id="@+id/start_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/start" />
 <Button
 android:id="@+id/pause_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/pause" />
 <Button
 android:id="@+id/reset_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/reset" />
</LinearLayout>
```

Код активности управляет работой хронометра

Код активности должен определить, каким образом кнопки Start, Pause и Reset управляют работой секундомера. Кнопка Start должна запускать секундомер (если он не был запущен ранее), кнопка Pause должна останавливать секундомер, а кнопка Reset — обнулять его показания. Для управления секундомером кнопки будут обращаться к встроенным свойствам и методам Chronometer. Рассмотрим эти свойства и методы. Ключевые свойства и методы Chronometer Как говорилось ранее, представление Chronometer является разновидностью TextView. Оно наследует все свойства и методы TextView, а также определяет ряд новых свойств и методов. Ниже перечислены все свойства и методы Chronometer, которые будут использоваться в коде MainActivity:

Свойство base

Свойство base элемента Chronometer используется для определения базового времени, то есть времени, с которого начинается отсчет времени хронометром. Чтобы назначить текущее время как базовое, следует задать свойству base результат вызова SystemClock.elapsedRealtime():

```
stopwatch.base = SystemClock.elapsedRealtime()
```

`SystemClock.elapsedRealtime()` возвращает время в миллисекундах, прошедшее с момента загрузки устройства. Если присвоить свойству base это значение, выводимое время обнуляется.

Метод start() запускает отсчет от базового времени:

```
stopwatch.start()
```

Приостанавливает отсчет времени хронометром:

```
stopwatch.stop()
```

Теперь вы знаете, какие свойства и методы представления `Chronometer` будут использоваться, и мы можем перейти к рассмотрению кода активности.

MainActivity.kt

Код MainActivity требует назначения слушателя OnClickListener для каждой кнопки. Слушатель кнопки Start должен запустить секундомер, если он не работает в настоящий момент, слушатель кнопки Pause должен секундомер останавливать, а слушатель кнопки Reset — обнулять показания секундомера. В этом нам помогут три свойства: stopwatch, running и offset. Свойство stopwatch содержит ссылку на представление Chronometer. Свойство running содержит признак того, работает ли секундомер в настоящий момент. При щелчке на кнопке Start ему присваивается значение true, а при щелчке на кнопке Pause — значение false. Свойство offset используется для вывода правильного времени на секундомере, если он был приостановлен и запущен заново. Без этого на секундомере выводилось бы неправильное время. Также были включены два метода (setBaseTime и saveOffset), которые упрощают чтение кода. Ниже приведен полный код MainActivity.kt (изменения выделены жирным шрифтом):

```
package com.hfad.stopwatch
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.SystemClock
import android.widget.Button
import android.widget.Chronometer
class MainActivity : AppCompatActivity() {
 lateinit var stopwatch: Chronometer // Хронометр
 var running = false // Хронометр работает?
 var offset: Long = 0 //Базовое смещение
 override fun onCreate(savedInstanceState: Bundle?) {
 super.onCreate(savedInstanceState)
 setContentView(R.layout.activity_main)
 //Получение ссылки на секундомер
 stopwatch = findViewById<Chronometer>(R.id.stopwatch)
 //Кнопка start запускает секундомер, если он не работал
 val startButton = findViewById<Button>(R.id.start_button)
 startButton.setOnClickListener {
 if (!running) {
 setBaseTime()
 stopwatch.start()
 running = true
 }
 }
 //Кнопка pause останавливает секундомер, если он работал
 val pauseButton = findViewById<Button>(R.id.pause_button)
 pauseButton.setOnClickListener {
 if (running) {
 saveOffset()
 stopwatch.stop()
 running = false
 }
 }
 //Кнопка reset обнуляет offset и базовое время
 val resetButton = findViewById<Button>(R.id.reset_button)
 resetButton.setOnClickListener {
 offset = 0
 setBaseTime()
 }
 }
 //Обновляет время stopwatch.base
 fun setBaseTime() {
 stopwatch.base = SystemClock.elapsedRealtime() - offset
 }
 //Сохраняет offset
 fun saveOffset() {
 offset = SystemClock.elapsedRealtime() - stopwatch.base
 }
}
```

Замечание: Если повернуть устройство (с включенным режимом автоповорота), секундомер обнуляется и перестает работать.

Поворот экрана изменяет конфигурацию устройства

Когда Android запускает приложение и передает управление активности, при этом учитывается конфигурация устройства. Под этим подразумевается как конфигурация физического устройства (размер экрана, ориентация экрана, наличие подключенной клавиатуры), так и параметры конфигурации, заданные пользователем (скажем, локальный контекст). Система Android должна знать конфигурацию устройства при запуске активности, потому что конфигурация может влиять на то, какие ресурсы необходимы приложению. Например, приложение может использовать другой макет, если экран устройства находится в горизонтальном режиме вместо вертикального, и другой набор строковых значений, если выбран французский локальный контекст.

При изменении конфигурации устройства все компоненты, отображающие пользовательский интерфейс, должны быть обновлены для новой конфигурации. Например, при повороте экрана Android замечает, что ориентация изменилась, и рассматривает это как изменение конфигурации устройства. Текущая активность уничтожается и создается заново, чтобы были загружены новые ресурсы, соответствующие новой конфигурации.

Состояние выполнения

Когда Android создает и уничтожает активность, она переходит из состояния запуска в состояние выполнения, затем в состояние уничтожения. Основное состояние активности — состояние выполнения (или активное состояние). Активность выполняется, когда она находится на переднем плане экрана, обладает фокусом и пользователь может взаимодействовать с ней. Активность проводит большую часть своего жизненного цикла в этом состоянии. Активность начинает выполняться после запуска, а в конце своего жизненного цикла активность уничтожается.

Когда активность переходит в состояние запуска или в состояние уничтожения, срабатывают ключевые методы жизненного цикла активности: onCreate() и onDestroy(). Это методы жизненного цикла, наследуемые активностью, которые могут переопределяться. Метод onCreate() вызывается сразу же после запуска активности. В этом методе происходит вся нормальная настройка активности, например вызов setContentView(). Этот метод должен переопределяться всегда. Если вы не переопределите его, вы не сможете сообщить Android, какой макет должен использоваться активностью. Метод onDestroy() — последняя возможность что-то сделать перед уничтожением активности. Активность может уничтожаться в разных ситуациях: например, если она получила приказ завершиться, если активность создается заново из-за изменения конфигурации устройства или если Android решает уничтожить активность для экономии памяти.

Сохранение текущего состояния в объекте Bundle

Как было показано ранее, в приложении Stopwatch проблемы возникли при повороте устройства. Активность MainActivity была уничтожена и создана заново, в результате чего все значения свойств были потеряны и представление Chronometer было сброшено. Чтобы исправить эту ошибку, следует сохранить состояние Chronometer и значения свойств при уничтожении активности и восстановить их при ее повторном создании. Для сохранения значений будет использоваться объект Bundle. Bundle— объект, предназначенный для хранения пар «ключ–значение». Перед уничтожением активности Android позволяет сохранить пары «ключ–значение» в Bundle. Затем этот объект Bundle используется новым экземпляром активности при повторном создании. Таким образом, использование Bundle дает возможность восстановить состояние активности при повороте экрана устройства. Прежде чем

смотреть, как Bundle передается между экземплярами активностей, посмотрим, как происходит добавление и загрузка значений.

Сохранение значений методами put

Пары «ключ-значение» сохраняются в Bundle методами put. Метод putLong сохраняет значение Long в Bundle, putBoolean() сохраняет Boolean, и т. д. Например, следующий код сохраняет в объекте Bundle с именем bundle ключ «answer» и значение 42 типа Int:

```
bundle.putInt("answer", 42)
```

В Bundle можно сохранить несколько пар «ключ–значение». Каждый ключ должен относиться к типу String?, а значение определяется вызываемым методом. Например, невозможно сохранить значение Int в Bundle вызовом метода putString() — компилятор запротестует.

Чтение значений методами get

После того как пара «ключ–значение» будет сохранена в Bundle, значение можно прочитать одним из методов get объекта Bundle. Например, метод getLong() предназначен для чтения значений Long, а метод getBoolean() — для чтения значений Boolean. Каждый метод get получает один параметр: имя ключа для значения, которое вы хотите получить. Например, следующий код получает значение, сохраненное с ключом «answer» в объекте Bundle с именем bundle:

```
val answerOfLife = bundle.getInt("answer")
```

Теперь вы знаете, как сохранять и читать значения из объектов Bundle, и мы можем воспользоваться этим объектом для сохранения и восстановления состояния активности.

Сохранение состояния вызовом onSaveInstanceState()

Объекты Bundle передаются между экземплярами активности при помощи двух методов: onSaveInstanceState() и onCreate(). onSaveInstanceState() сохраняет значения в Bundle перед уничтожением активности, а onCreate() снова загружает Bundle при повторном создании активности. Каждая активность наследует метод onSaveInstanceState() от суперкласса Activity. Этот метод вызывается непосредственно перед уничтожением активности и получает один параметр: Bundle. Метод onSaveInstanceState() переопределяется в вашей активности кодом следующего вида:

```
override fun onSaveInstanceState(savedInstanceState: Bundle) {
 savedInstanceState.putInt("answer", 42)
 super.onSaveInstanceState(savedInstanceState)
}
```

Приведенный пример сохраняет в savedInstanceState (объекте Bundle, переданном onSaveInstanceState()) ключ «answer» со значением 42 типа Int. Затем он вызывает версию

onSaveInstanceState() из суперкласса, которая сохраняет состояние иерархии представлений.

Восстановление состояния в onCreate()

Метод onCreate() вызывается при повторном создании активности, и он получает один параметр Bundle?. Если активность создается с нуля, то значение Bundle? будет равно null, но при повторном создании активности это будет объект Bundle, который использовался onSaveInstanceState(). Метод onCreate() используется для чтения значений из Bundle кодом следующего вида:

```
override fun onCreate(savedInstanceState: Bundle?) {
 super.onCreate(savedInstanceState)
 setContentView(R.layout.activity_main)
 if (savedInstanceState != null) {
 var answer = savedInstanceState.getInt("answer")
 }
 ...
}
```

Обновленный код `MainActivity.kt`:

```
package com.hfad.stopwatch
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.SystemClock
import android.widget.Button
import android.widget.Chronometer
class MainActivity : AppCompatActivity() {
 lateinit var stopwatch: Chronometer //Хронометр
 var running = false // Хронометр работает?
 var offset: Long = 0 //Базовое смещение
 //Добавление строк для ключей, используемых с Bundle
 val OFFSET_KEY = "offset"
 val RUNNING_KEY = "running"
 val BASE_KEY = "base"
 override fun onCreate(savedInstanceState: Bundle?) {
 super.onCreate(savedInstanceState)
 setContentView(R.layout.activity_main)
 //Получение ссылки на секундомер
 stopwatch = findViewById<Chronometer>(R.id.stopwatch)
 //Восстановление предыдущего состояния
 if (savedInstanceState != null) {
 offset = savedInstanceState.getLong(OFFSET_KEY)
 running = savedInstanceState.getBoolean(RUNNING_KEY)
 if (running) {
 stopwatch.base = savedInstanceState.getLong(BASE_KEY)
 stopwatch.start()
 } else setBaseTime()
 }

//Кнопка start запускает секундомер, если он не работал
```

```
 val startButton = findViewById<Button>(R.id.start_button)
 startButton.setOnClickListener {
 if (!running) {
 setBaseTime()
 stopwatch.start()
 running = true
 }
 }
 //Кнопка pause останавливает секундомер, если он работал
 val pauseButton = findViewById<Button>(R.id.pause_button)
 pauseButton.setOnClickListener {
 if (running) {
 saveOffset()
 stopwatch.stop()
 running = false
 }
 }
 //Кнопка reset обнуляет offset и базовое время
 val resetButton = findViewById<Button>(R.id.reset_button)
 resetButton.setOnClickListener {
 offset = 0
 setBaseTime()
 }
 }
 override fun onSaveInstanceState(savedInstanceState: Bundle) {
 savedInstanceState.putLong(OFFSET_KEY, offset)
 savedInstanceState.putBoolean(RUNNING_KEY, running)
 savedInstanceState.putLong(BASE_KEY, stopwatch.base)
 super.onSaveInstanceState(savedInstanceState)
 }

 // Код методов saveOffset() и setBaseTime() опущен, так как их обновлять не
нужно.

}
```

Реализация остановки секундомера в onStop()

Переопределите метод onStop(), добавив в активность следующий метод:

```
override fun onStop() {
 super.onStop()
 //Код, выполняемый при остановке
активности
}
```

В строке:

```
super.onStop()
```

вызывается метод onStop() суперкласса Activity. Эта строка кода должна добавляться во всех переопределениях метода onStop(), чтобы активность выполнила все действия метода жизненного цикла суперкласса. При попытке обойти этот шаг Android сгенерирует исключение. Сказанное относится ко всем методам жизненного цикла; если вы переопределяете любые методы жизненного цикла Activity в своей активности, необходимо вызвать метод суперкласса, иначе компилятор начнет протестовать. Отсчет времени должен останавливаться при вызове метода onStop(). Для этого необходимо вызвать метод saveOffset() (чтобы позднее отсчет времени можно было продолжить с момента остановки), а затем остановить секундомер. Код выглядит так:

```
override fun onStop() {
 super.onStop()
 if (running) {
 saveOffset()
 stopwatch.stop()
 }
}
```

Итак, теперь секундомер останавливается, когда активность становится невидимой. Теперь нужно сделать следующий шаг — снова запустить отсчет времени, когда активность станет видимой.

Перезапуск отсчета времени, когда приложение становится видимым

Когда активность снова становится видимой, вызываются два ключевых метода жизненного цикла: onStart() и onRestart(). Метод OnStart() вызывается, когда активность должна стать видимой. Это может произойти как при ее создании, так и тогда, когда она снова становится видимой после потери видимости. Метод onStart() реализуется кодом следующего вида:

```
override fun onStart() {
 super.onStart()
 //Код, выполняемый при запуске активности
}
```

Как и в случае с методом onStop(), строка:

```
super.onStart()
```

вызывает метод onStart() суперкласса Activity. Метод onRestart() вызывается перед onStart(), но только когда активность становится видимой после потери видимости. Он не вызывается, когда активность становится видимой впервые.

Метод onRestart() реализуется кодом следующего вида:

```
override fun onRestart() {
 super.onRestart()
 //Код, выполняемый при перезапуске активности
}
```

В приложении Stopwatch отсчет времени должен возобновляться тогда, когда активность снова становится видимой, так что мы реализуем метод onRestart(). Код выглядит так:

```
override fun onRestart() {
 super.onRestart()
 if (running) {
 setBaseTime()
 stopwatch.start()
 offset = 0
 }
}
```

Внесите обновления кода в MainActivity.kt:

```
class MainActivity : AppCompatActivity() {
 lateinit var stopwatch: Chronometer //Хронометр
 var running = false //Хронометр работает?
 var offset: Long = 0 //Базовое смещение
 //Добавление строк для ключей, используемых с Bundle
 val OFFSET_KEY = "offset"
 val RUNNING_KEY = "running"
 val BASE_KEY = "base"
 override fun onCreate(savedInstanceState: Bundle?) {
 ...
 }
 override fun onStop() {
 super.onStop()
 if (running) {
 saveOffset()
 stopwatch.stop()
 }
 }
 override fun onRestart() {
 super.onRestart()
 if (running) {
 setBaseTime()
 stopwatch.start()
 offset = 0
 }
 }
```

```
 // Код методов saveOffset(), setBaseTime() и onSaveInstanceState() не приводится,
так как обновлять их не нужно.

}
```

Реализация методов onPause() и onResume().

Ранее вы видели, что происходит при создании и уничтожении активностей и что происходит, когда активность становится видимой или невидимой. Но необходимо рассмотреть еще одну ситуацию: когда активность видима, но не обладает фокусом. Если активность видима, но не обладает фокусом, она приостанавливается. Это может произойти, если над вашей активностью отображается другая активность, которая не занимает весь экран или прозрачна. Верхняя активность обладает фокусом, но нижняя активность остается видимой и поэтому находится в приостановленном состоянии.

Существуют два метода жизненного цикла, которые обрабатывают приостановку активности и ее повторную активизацию: onPause() и onResume(). Метод onPause() вызывается, если активность остается видимой, но фокус принадлежит другой активности. Метод onResume() вызывается непосредственно перед тем, как ваша активность будет готова к взаимодействию с пользователем. Посмотрим, как эти методы вписываются в общую картину жизненного цикла активности.

Вернемся к приложению Stopwatch. К настоящему моменту мы заставили отсчет времени приостанавливаться, если приложение Stopwatch невидимо, и перезапускаться, когда приложение снова становится видимым. Для этого были переопределены методы onStop() и onRestart(). Сделаем так, чтобы приложение точно так же вело себя при частичной видимости. Отсчет времени должен приостанавливаться при приостановке активности и снова запускаться при возобновлении ее работы. Какие же методы жизненного цикла для этого нужно реализовать? В двух словах, следует использовать методы onPause() и onResume(), но можно пойти еще дальше. Мы используем эти методы для замены уже реализованных вызовов onStop() и onRestart(). Если снова взглянуть на диаграмму жизненного цикла, методы onPause() и onResume(), помимо методов onStop(), onStart() и onRestart(), вызываются при каждой остановке и перезапуске активности. Так как мы хотим, чтобы приложение вело себя одинаково независимо от того, приостанавливается активность или останавливается, можно воспользоваться методами onPause() и onResume() в обеих ситуациях. Реализация методов onPause() и onResume() выглядит примерно так:

```
override fun onPause() {
 super.onPause()
 //Код, выполняемый при приостановке активности
}
override fun onResume() {
 super.onResume()
 //Код, выполняемый при возобновлении активности
}
```

Как и прежде, вызовы super.onPause() и super.onResume() обязательны. Они вызывают методы onPause() и onResume() суперкласса Activity, и без этих вызовов код не будет компилироваться.

Ниже приведен полный код MainActivity.kt для завершенного приложения; обновите свою версию кода:

```
package com.hfad.stopwatch
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.SystemClock
import android.widget.Button
import android.widget.Chronometer
class MainActivity : AppCompatActivity() {
 lateinit var stopwatch: Chronometer //Хронометр
 var running = false //Хронометр выполняется?
 var offset: Long = 0 //Базовое смещение
 //Добавление строк для ключей, используемых с Bundle
 val OFFSET_KEY = "offset"
 val RUNNING_KEY = "running"
 val BASE_KEY = "base"
 override fun onCreate(savedInstanceState: Bundle?) {
 super.onCreate(savedInstanceState)
 setContentView(R.layout.activity_main)
 //Получение ссылки на секундомер
 stopwatch = findViewById<Chronometer>(R.id.stopwatch)
 //Восстановление предыдущего состояния
 if (savedInstanceState != null) {
 offset = savedInstanceState.getLong(OFFSET_KEY)
 running = savedInstanceState.getBoolean(RUNNING_KEY)
 if (running) {
 stopwatch.base = savedInstanceState.getLong(BASE_KEY)
 stopwatch.start()
 } else setBaseTime()
 }

//Кнопка start запускает секундомер, если он не работал
 val startButton = findViewById<Button>(R.id.start_button)
 startButton.setOnClickListener {
 if (!running) {
 setBaseTime()
 stopwatch.start()
 running = true
 }
 }
 //Кнопка pause останавливает секундомер, если он работал
 val pauseButton = findViewById<Button>(R.id.pause_button)
 pauseButton.setOnClickListener {
 if (running) {
 saveOffset()
 stopwatch.stop()
 running = false
 }
 }
 //Кнопка reset обнуляет offset и базовое время
 val resetButton = findViewById<Button>(R.id.reset_button)
 resetButton.setOnClickListener {
 offset = 0
 setBaseTime()
```

```
    }
    }
    override fun onPause() {
    super.onStop()
    super.onPause()
    if (running) {
    saveOffset()
    stopwatch.stop()
    }
    }

    override fun onResume() {
    super.onRestart()
    super.onResume()
    if (running) {
    setBaseTime()
    stopwatch.start()
    offset = 0
    }
    }
    override fun onSaveInstanceState(savedInstanceState: Bundle) {
    savedInstanceState.putLong(OFFSET_KEY, offset)
    savedInstanceState.putBoolean(RUNNING_KEY, running)
    savedInstanceState.putLong(BASE_KEY, stopwatch.base)
    super.onSaveInstanceState(savedInstanceState)
    }
    //Обновляет время stopwatch.base
    fun setBaseTime() {
    stopwatch.base = SystemClock.elapsedRealtime() - offset
    }
    //Сохраняет offset
    fun saveOffset() {
    offset = SystemClock.elapsedRealtime() - stopwatch.base
    }
  }
```

Краткая сводка методов жизненного цикла активностей

Метод	Когда вызывается	Следующий метод
onCreate()	При первоначальном создании активности. Используется для обычной подготовки, например,	

создания представлений. Также предоставляет объект Bundle с ранее сохраненным состоянием активности.|onStart()| |onRestart()|Когда активность остановилась, но до ее повторного запуска.|onStart()| |onStart()|Когда активность становится видимой. Далее следует вызов onResume(), если активность выходит на передний план, или onStop(), если активность становится невидимой.|onResume() или onStop()| |onResume()|Когда активность выходит на передний план.|onPause()| |onPause()|Когда ваша активность уходит с переднего плана из-за того, что другая активность возобновляет работу.

Следующая активность не начнет выполняться до завершения этого метода, поэтому код метода должен завершиться быстро. Далее следует вызов onResume(), если активность возвращается на передний план, или onStop(), если она становится невидимой. |onResume() или onStop()| |onStop()|Когда активность становится невидимой. Ее может закрывать другая активность или эта активность уничтожается. Далее следует вызов onRestart(), если активность снова становится видимой, или onDestroy(), если активность уничтожается.|onRestart() или onDestroy()| |onDestroy()|Когда активность должна быть уничтожена или перед завершением активности.|Нет|

Резюме

- Представление Chronometer реализует таймер. Класс Chronometer является субклассом TextView.
- Изменение конфигурации устройства приводит к уничтожению и повторному созданию активности.
- Ваша активность наследует методы жизненного цикла от класса android.app.Activity. Если вы переопределяете любые из этих методов, в переопределении необходимо вызвать метод суперкласса.
- Метод onSaveInstanceState(Bundle) позволяет активности сохранить свое состояние перед ее уничтожением. Объект Bundle может использоваться для восстановления состояния в onCreate().
- Для добавления значений в Bundle в onSaveInstanceState() используется вызов bundle.put... ("name", value).
- Для чтения значений из Bundle в onCreate() используется вызов bundle.get...("name").
- Методы onCreate() и onDestroy() связаны с созданием и уничтожением активности.
- Методы onRestart(), onStart() и onStop() связаны с видимостью активности.
- Методы onResume() и onPause() вызываются при получении и потере фокуса активностью.

Отчет:

- в файле readme: разметка и код Activity.