[Lifecycle](https://startandroid.ru/ru/courses/architecture-components/27-course/architecture-components/524-urok-1.html" \o "Урок 1. Lifecycle)

В этом уроке рассмотрим возможности Lifecycle. Как подписаться на его события. Как узнать текущее состояние Activity.

Полный список уроков курса:

* [Урок 1. Lifecycle](https://startandroid.ru/ru/27-course/architecture-components/524-urok-1.html)
* [Урок 2. LiveData](https://startandroid.ru/ru/27-course/architecture-components/525-urok-2-livedata.html)
* [Урок 3. LiveData. Дополнительные возможности](https://startandroid.ru/ru/27-course/architecture-components/526-urok-3-livedata.html)
* [Урок 4. ViewModel](https://startandroid.ru/ru/27-course/architecture-components/527-urok-4-viewmodel.html)
* [Урок 5. Room. Основы](https://startandroid.ru/ru/27-course/architecture-components/529-urok-5-room-osnovy.html)
* [Урок 6. Room. Entity](https://startandroid.ru/ru/27-course/architecture-components/530-urok-6-room-entity.html)
* [Урок 7. Room. Insert, Update, Delete, Transaction](https://startandroid.ru/ru/27-course/architecture-components/531-urok-7-room-insert-update-delete-transaction.html)
* [Урок 8. Room. Query](https://startandroid.ru/ru/27-course/architecture-components/532-urok-8-room-query.html)
* [Урок 9. Room. RxJava](https://startandroid.ru/ru/27-course/architecture-components/533-urok-9-room-rxjava.html)
* [Урок 10. Room. Запрос из нескольких таблиц. Relation](https://startandroid.ru/ru/27-course/architecture-components/534-urok-10-room-zapros-iz-neskolkih-tablic-relation.html)
* [Урок 11. Room. Type converter](https://startandroid.ru/ru/27-course/architecture-components/539-urok-11-room-type-converter.html)
* [Урок 12. Room. Миграция версий базы данных](https://startandroid.ru/ru/27-course/architecture-components/540-urok-12-migracija-versij-bazy-dannyh.html)
* [Урок 13. Room. Тестирование](https://startandroid.ru/ru/27-course/architecture-components/541-urok-13-room-testirovanie.html)
* [Урок 14. Paging Library. Основы](https://startandroid.ru/ru/27-course/architecture-components/542-urok-14-paging-library-chast-1.html)
* [Урок 15. Paging Library. PagedList и DataSource. Placeholders.](https://startandroid.ru/ru/27-course/architecture-components/543-urok-15-paging-library-chast-2.html)
* [Урок 16. Paging Library. LivePagedListBuilder. BoundaryCallback.](https://startandroid.ru/ru/27-course/architecture-components/544-urok-15-paging-library-chast-2-pagedlist-i-datasource.html)
* [Урок 17. Paging Library. Виды DataSource](https://startandroid.ru/ru/27-course/architecture-components/545-urok-17-paging-library-vidy-datasource.html)
* [Урок 18. Android Data Binding. Основы](https://startandroid.ru/ru/27-course/architecture-components/551-urok-18-data-binding-osnovy.html)
* [Урок 19. Android Data Binding. Код в layout. Доступ к View](https://startandroid.ru/ru/27-course/architecture-components/552-urok-19-android-data-binding-vozmozhnosti.html)
* [Урок 20. Android Data Binding. Обработка событий](https://startandroid.ru/ru/27-course/architecture-components/553-urok-20-android-data-binding-obrabotka-sobytiy.html)
* [Урок 21. Android Data Binding. Observable поля. Двусторонний биндинг.](https://startandroid.ru/ru/27-course/architecture-components/554-urok-21-android-data-binding-observable.html)
* [Урок 22. Android Data Binding. Adapter. Conversion.](https://startandroid.ru/ru/27-course/architecture-components/555-urok-22-android-data-binding-include-viewstub-holders.html)
* [Урок 23. Android Data Binding. Использование с include, ViewStub и RecyclerView.](https://startandroid.ru/ru/27-course/architecture-components/556-urok-23-android-data-binding.html)
* [Урок 24. Navigation Architecture Component. Введение](https://startandroid.ru/ru/27-course/architecture-components/557-urok-24-android-navigation-component-vvedenie.html)
* [Урок 25. Navigation. Передача данных. Type-safe аргументы.](https://startandroid.ru/ru/27-course/architecture-components/558-urok-25-navigation-peredacha-dannyh-typesafe-argumenty.html)
* [Урок 26. Navigation. Параметры навигации](https://startandroid.ru/ru/27-course/architecture-components/559-urok-26-navigation-parametry-navigacii.html)
* [Урок 27. Navigation. NavigationUI.](https://startandroid.ru/ru/27-course/architecture-components/560-urok-27-navigation-navigationui.html)
* [Урок 28. Navigation. Вложенный граф. Global Action. Deep Link.](https://startandroid.ru/ru/27-course/architecture-components/561-urok-28-navigation-vlozhennyj-graf.html)
* [Урок 29. WorkManager. Введение](https://startandroid.ru/ru/27-course/architecture-components/562-urok-29-workmanager-vvedenie.html)
* [Урок 30. WorkManager. Критерии запуска задачи.](https://startandroid.ru/ru/27-course/architecture-components/563-urok-30-workmanager-constraints.html)
* [Урок 31. WorkManager. Последовательность выполнения задач.](https://startandroid.ru/ru/27-course/architecture-components/564-urok-31-workmanager-posledovatelnosti-vypolnenija-zadach.html)
* [Урок 32. WorkManager. Передача и получение данных](https://startandroid.ru/ru/27-course/architecture-components/565-urok-32-workmanager-peredacha-i-poluchenie-dannyh.html)
* [Урок 33. Практика. О чем это будет.](https://startandroid.ru/ru/27-course/architecture-components/566-urok-33-praktika-o-chem-eto-budet.html)
* [Урок 34. Практика. TodoApp. Список задач.](https://startandroid.ru/ru/27-course/architecture-components/567-urok-34-praktika-todoapp-tasks.html)
* [Урок 35. Практика. TodoApp. Просмотр задачи](https://startandroid.ru/ru/27-course/architecture-components/569-urok-35-praktika-todoapp-taskdetail.html)

Довольно часто часть логики приложения завязана на жизненный цикл Activity. Мы включаем что-либо в методах onStart или onResume и выключаем в onPause или onStop.

Рассмотрим пример.

У нас есть какой-то класс для работы с сервером. Он должен взаимодействовать с сервером, пока Activity открыто. Соответственно, мы будем подключать его к серверу при показе Activity и отключать при скрытии Activity.

|  |
| --- |
| public class MyServer {       public void connect() {         // ...     }       public void disconnect() {         // ...     }    } |

Метод connect используется для подключения к серверу, disconnect - для отключения.

Вызываем эти методы в onStart и onStop в Activity.

|  |
| --- |
| @Override  protected void onStart() {     super.onStart();     myServer.connect();  }      @Override  protected void onStop() {     super.onStop();     myServer.disconnect();  } |

Теперь MyServer будет подключен пока Activity видимо на экране.

Это вполне классическая, часто используемая схема. И в простом примере все выглядит неплохо. Но в сложных приложениях содержание методов onStart, onStop и пр. может состоять из нескольких десятков строк и быть достаточно запутанным. Гугл рекомендует выносить эту логику из Activity. Давайте посмотрим, как это можно сделать.

Lifecycle

У Activity есть метод getLifecycle, который возвращает объект Lifecycle. На этот объект можно подписать слушателей, которые будут получать уведомления при смене lifecycle-состояния Activity.

*Activity и фрагменты в Support Library, начиная с версии 26.1.0 реализуют интерфейс LifecycleOwner. Именно этот интерфейс и добавляет им метод getLifecycle.*

*Т.е. у вас должна быть такая строка в build.gradle файле модуля, в секции dependencies*

|  |  |
| --- | --- |
| 1 | implementation 'com.android.support:appcompat-v7:26.1.0' |

*Либо используйте более свежую версию.*

В нашем примере слушателем будет MyServer. Чтобы иметь возможность подписаться на Lifecycle, он должен наследовать интерфейс [LifecycleObserver](https://developer.android.com/reference/android/arch/lifecycle/LifecycleObserver.html" \t "_blank).

|  |
| --- |
| public class MyServer implements LifecycleObserver {       @OnLifecycleEvent(Lifecycle.Event.ON\_START)     public void connect() {         // ...     }       @OnLifecycleEvent(Lifecycle.Event.ON\_STOP)     public void disconnect() {         // ...     }    } |

Обратите внимание, что интерфейс LifecycleObserver пустой. В нем нет кучи методов типа onStart, onStop и т.п. Мы просто помечаем в классе MyServer его же собственные методы аннотацией OnLifecycleEvent и указываем, при каком lifecycle-событии метод должен быть вызван.

В нашем примере, мы указываем, что метод connect должен вызываться в момент onStart, а метод disconnect - в момент onStop.

Осталось подписать экземпляр MyServer на Lifecycle.

|  |
| --- |
| @Override  protected void onCreate(Bundle savedInstanceState) {     super.onCreate(savedInstanceState);     setContentView(R.layout.activity\_main);     // ...     getLifecycle().addObserver(myServer);  } |

В Activity методом getLifecycle получаем Lifecycle, и методом [addObserver](https://developer.android.com/reference/android/arch/lifecycle/Lifecycle.html" \l "addObserver(android.arch.lifecycle.LifecycleObserver)" \t "_blank) подписываем myServer.

А методы onStart и onStop в Activity нам больше не нужны, их можно удалить.

Теперь, при переходе Activity из состояния CREATED в состояние STARTED, его объект Lifecycle вызовет метод myServer.connect. А при переходе из STARTED в CREATED - Lifecycle вызовет myServer disconnect.

При этом в Acivity это потребовало от нас минимум кода - только подписать myServer на Lifecycle. Все остальное решает сам MyServer.

На схеме ниже вы можете увидеть какие состояние проходит Activity и какие события при этом вызываются.

Ничего нового тут для нас нет, все это азы Android. Тут вы можете видеть состояния и события. Они связаны очень просто - при переходе между состояниями происходят события.

Эти события мы указывали в аннотациях OnLifecycleEvent к методам объекта MyServer.

Полный список событий можно посмотреть в [документации](https://developer.android.com/reference/android/arch/lifecycle/Lifecycle.Event.html).

Отписаться от Lifecycle можно методом [removeObserver](https://developer.android.com/reference/android/arch/lifecycle/Lifecycle.html" \l "removeObserver(android.arch.lifecycle.LifecycleObserver)" \t "_blank).

 ny

Вы можете использовать событие ON\_ANY для получения всех событий в одном методе

|  |  |
| --- | --- |
| 1  2  3  4 | @OnLifecycleEvent(ON\_ANY)  void onAny(LifecycleOwner source, Lifecycle.Event event) {      // ...  } |

В этом случае все события будут вызывать этот метод.

Используйте входящий параметр event, чтобы определить, какое именно событие произошло

Состояние

Если вы хотите узнать текущее состояние Activity, то у его объекта Lifecycle есть метод getCurrentState:

|  |
| --- |
| if (getLifecycle().getCurrentState() == Lifecycle.State.RESUMED) {     // ...  } |

Полный список состояний можно посмотреть в [документации](https://developer.android.com/reference/android/arch/lifecycle/Lifecycle.State.html).

Также, вы можете проверить, что текущее состояние Activity не ниже определенного состояния.

|  |
| --- |
| if (getLifecycle().getCurrentState().isAtLeast(Lifecycle.State.STARTED)) {     // ...  } |

Метод isAtLeast здесь проверяет, что состояние Activity не ниже, чем STARTED. Т.е. либо STARTED, либо RESUMED.

Присоединяйтесь к нам в **Telegram**:

- в канале [StartAndroid](https://t.me/startandroid" \t "_blank) публикуются ссылки на новые статьи с сайта startandroid.ru и интересные материалы с хабра, medium.com и т.п.

- в чатах решаем возникающие вопросы и проблемы по различным темам: [Android](https://t.me/startandroidchat" \t "_blank), [Kotlin](https://t.me/kotlinchat" \t "_blank), [RxJava](https://t.me/rxjavachat" \t "_blank), [Dagger](https://t.me/daggerchat" \t "_blank), [Тестирование](https://t.me/androidtestingchat)

- ну и если просто хочется поговорить с коллегами по разработке, то есть чат [Флудильня](https://t.me/startandroidflood" \t "_blank)

- новый чат [Performance](https://t.me/android_performance_chat" \t "_blank) для обсуждения проблем производительности и для ваших пожеланий по содержанию курса по этой теме