android

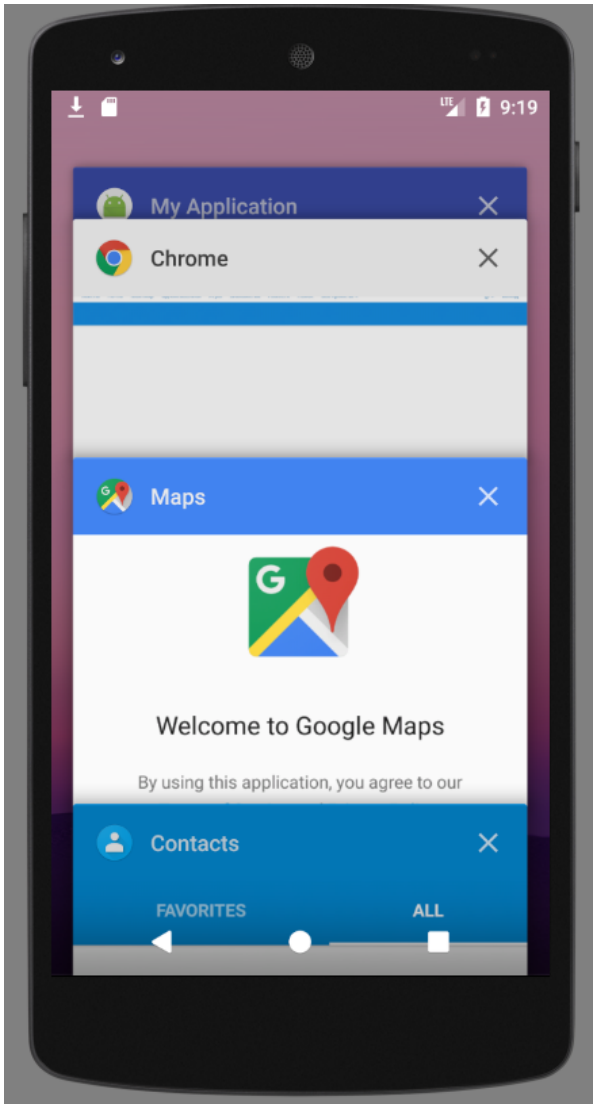
Разработка мобильных приложений

Android

5

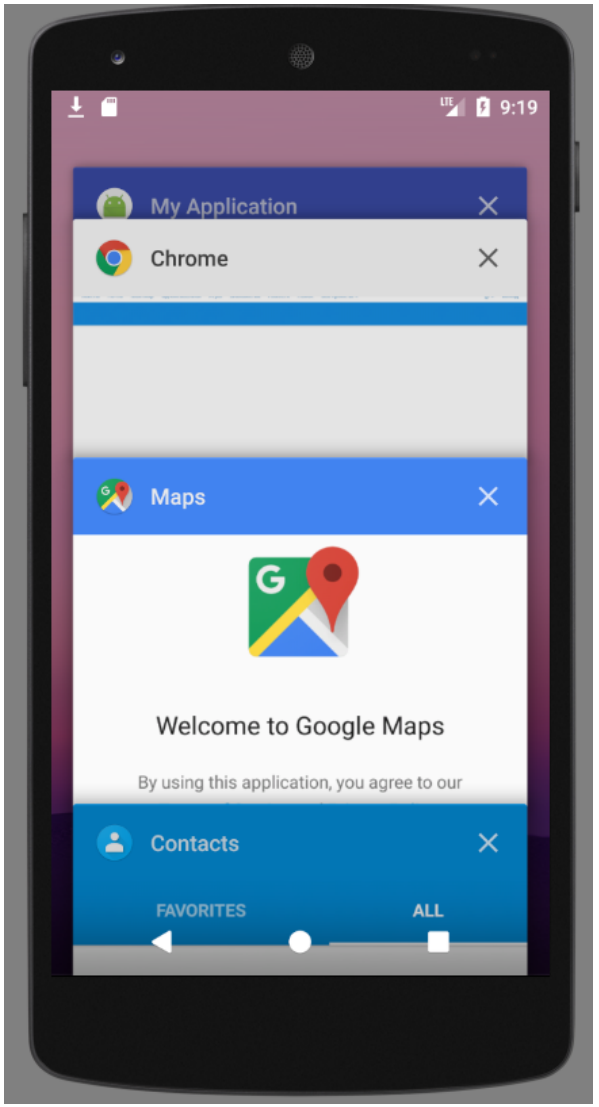
Как обычно пользователи используют телефон?

- Пример:
  - Открыли приложение MyApplication
  - Приложение MyApplication запустило браузер
  - Нажали кнопку Home
  - Открыли список последних приложений
  - Выбрали карты
  - Решили позвонить: открыли контакты
  - ...



Как обычно пользователи используют телефон?

- Результат:
  - Приложение MyApplication оказывается в фоновом режиме
  - Стек приложений (back stack) переполняется



Для поддержки работоспособности система **в любой момент может закрыть MyApplication**

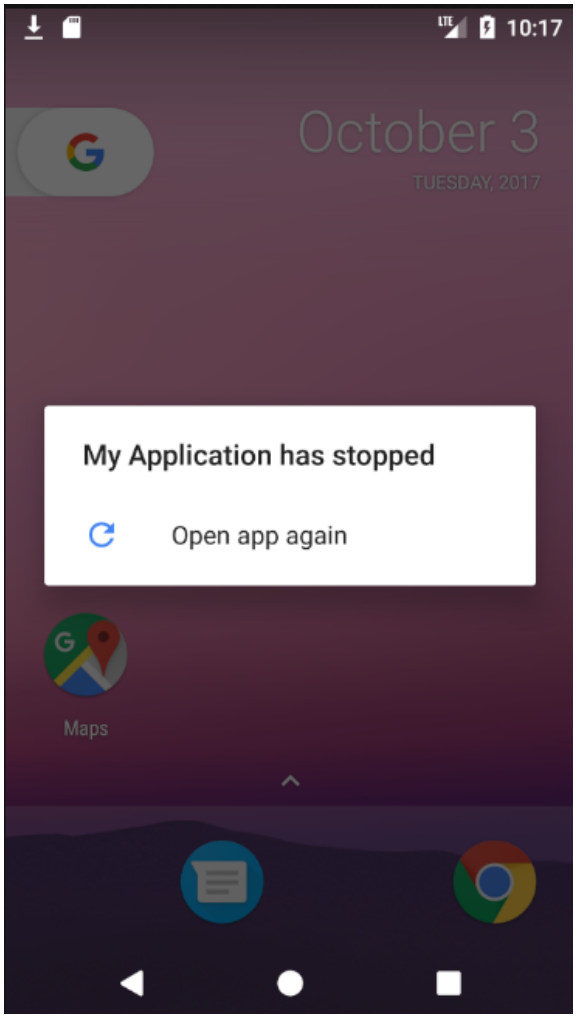
https://developer.android.com/guide/components/activities/tasks-and-back-stack.html

Задача

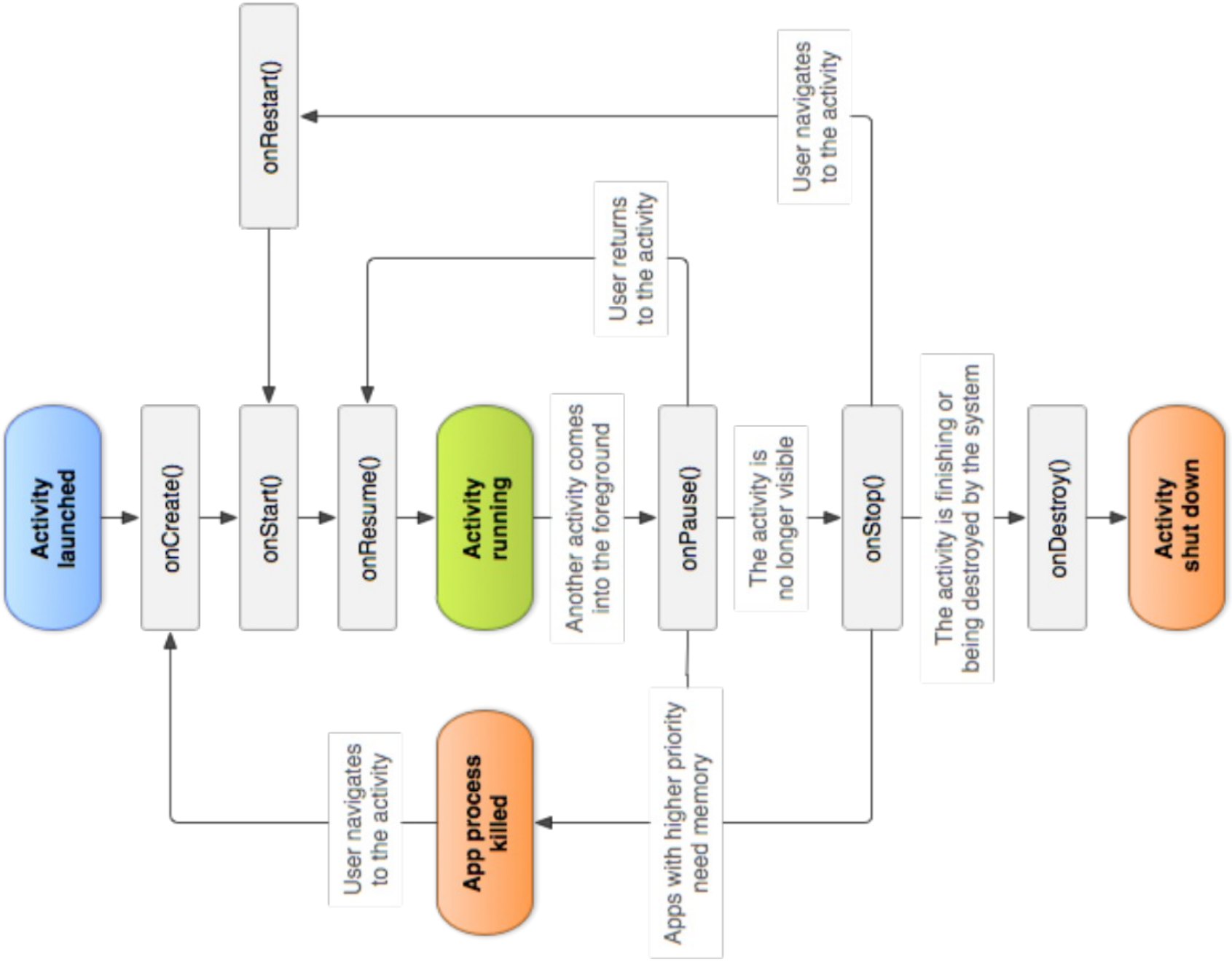
понимать, из чего состоит жизненный цикл приложения.

Цель — **избежать** в своем приложении следующего:

- проблем в случаях, когда пользователю необходимо принять звонок или переключиться на другое приложение
- использования большого количества ресурсов, когда пользователь не пользуется активно приложением
- потери текущего прогресса выполнения некоторой операции из-за того, что пользователь свернул приложение
- потери текущего прогресса выполнения некоторой операции из-за поворота экрана в ландшафтный или портретный режим и обратно



Жизненный цикл Activity (Lifecycle)



https://developer.android.com/guide/components/activities/activity-lifecycle.html

Когда приложение оказывается в разных состояниях

- Active
  - в фокусе, активно (видно пользователю)
- Visible
  - частично скрыто (например, появилось диалоговое окно поверх основной Activity)
- Background
  - приложение в фоновом режиме (пользователю видно другое приложение)

https://developer.android.com/guide/components/activities/state-changes.html_

Методы, вызываемые при повороте экрана

onPause

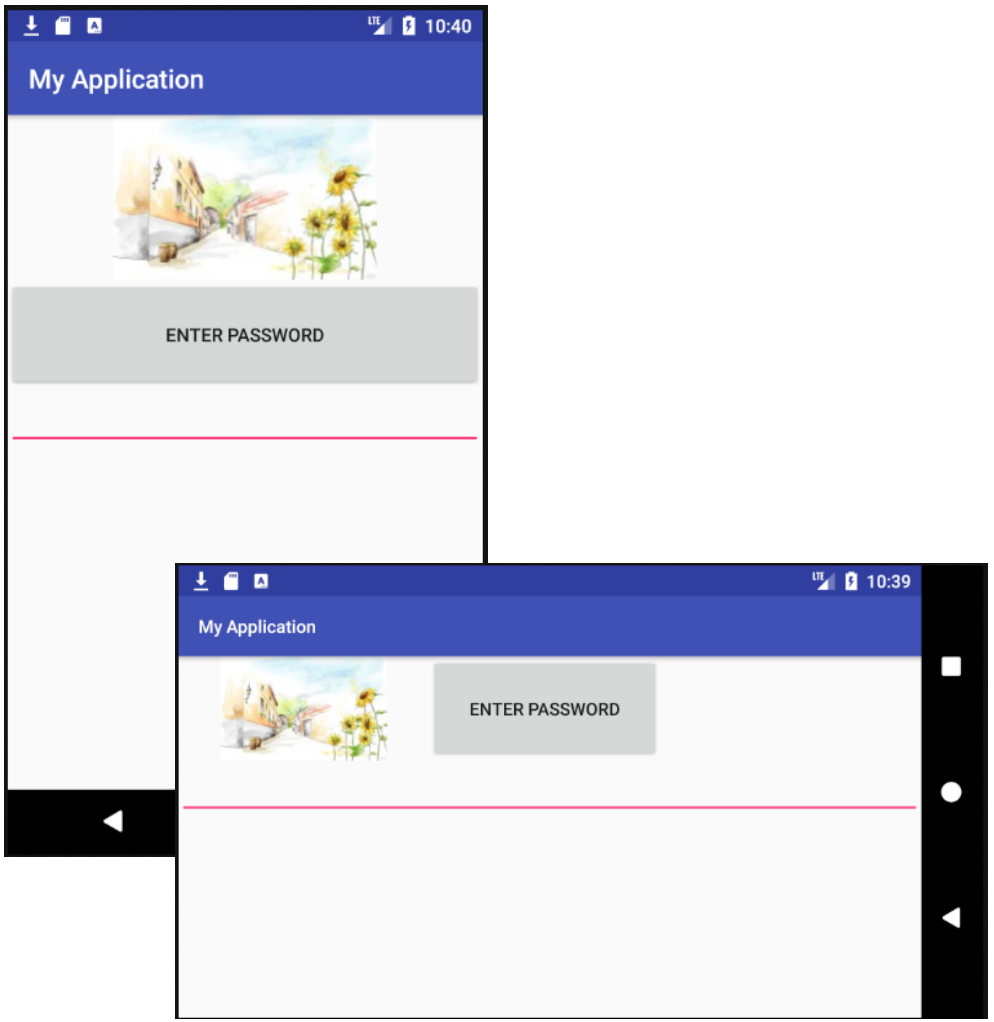
onStop

on Destroy

onCreate

onStart

onResume



onSavedInstanceState

- Использовать объект типа Bundle для сохранения значений нужных типов в методе onSavedInstanceState

```
private String SAVE_TO_BUNDLE__KEY = "bundle_key";

@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);

    outState.putString(SAVE_TO_BUNDLE__KEY,
            "will not loose this string when screen rotates");
}
```

- Считать сохраненные значения из объекта типа Bundle в методе onCreate

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

    if (savedInstanceState != null) {
        System.out.println("savedstate not null");
        if (savedInstanceState.containsKey(SAVE_TO_BUNDLE__KEY)) {
            System.out.println("savedstate contains key");
            String text = savedInstanceState.getString(SAVE_TO_BUNDLE__KEY).toString();
            //use data (for ex. to show it in TextView)
        }
    }
```

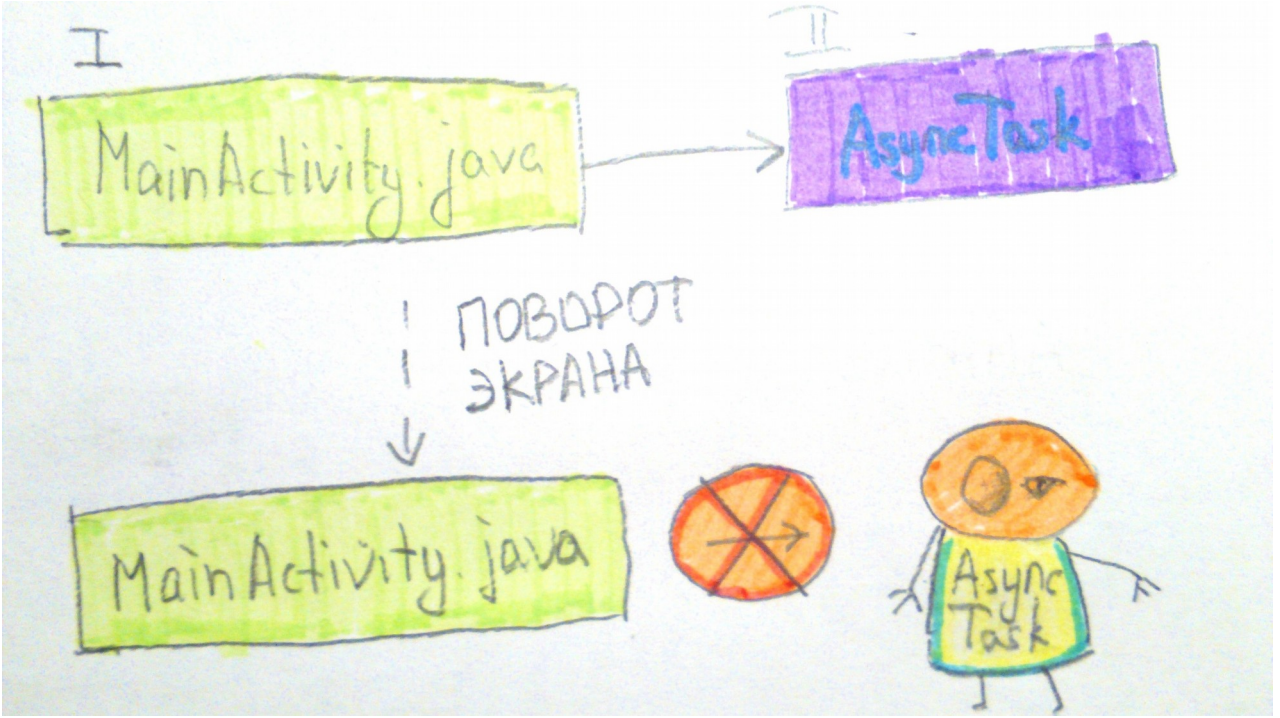
Сохранение в статической переменной

- OnSavedInstanceState вызывается
  - после onPause
  - до onStop, onDestroy
- Чтобы увидеть изменения, внесенные в методах onStop, onDestroy:
  - В классе главной Activity создается статическая переменная
  - В методах onStop, onDestroy в созданную переменную сохраняются необходимые данные
  - В методе onCreate сохраненные данные достаются из статической переменной

Зомби-процессы

Процессы, которые запущены нашим приложением, но из приложения у нас больше нет к ним доступа

Возникают, когда например, запустили AsyncTask, а потом повернули экран и приложение перезапустилось



Зомби-процессы

Проблемы после повторного создания процесса-дубля:

- больше времени уйдет на загрузку данных из интернета
- больше времени уйдет у пользователя на ожидание результата
- большее давление на память

Класс Loader

- framework для выполнения асинхронной загрузки данных
- предотвращают существование дублирующихся процессов

- Пример: AsyncTaskLoader
  - функциональность та же, что у AsyncTask, но с другим жизненным циклом

AsyncTaskLoader

- Создать id

```
private static final int UNIQUE_LOADER_ID = 734;
```

- implement LoaderManager.LoaderCallbacks

```
public class MainActivity extends AppCompatActivity
        implements LoaderManager.LoaderCallbacks<String> {

@Override
public Loader<String> onCreateLoader(int id, Bundle args) {
    return null;
}

@Override
public void onLoadFinished(Loader<String> loader, String data) {

}

@Override
public void onLoaderReset(Loader<String> loader) {

}
```

AsyncTaskLoader

- Реализация callback-функций

```
@Override
public Loader<String> onCreateLoader(int id, final Bundle args) {
    return new AsyncTaskLoader<String>(this) {

        @Override
        protected void onStartLoading() {
            //similar to onPreExecute of AsyncTask
            if (args == null) return;
            //show loading indicator
            forceLoad();
        }

        @Override
        public String loadInBackground() {
            //similar to doInBackground of AsyncTask
            return "";
        }
    };
}

@Override
public void onLoadFinished(Loader<String> loader, String data) {
    //similar to onPostExecute of AsyncTask
}

@Override
public void onLoaderReset(Loader<String> loader) {

}
```

AsyncTaskLoader

- Передача параметров — через Bundle,

  запуск AsyncTaskLoader

```
//instead of:
//      new someAsyncTaskClass().execute(stringParam);

Bundle asyncTaskLoaderParams = new Bundle();
asyncTaskLoaderParams.putString(BUNDLE_PARAMS_KEY,
        "test input data for AsyncTask Loader");

LoaderManager loaderManager = getSupportLoaderManager();
Loader<String> loader = loaderManager.getLoader(UNIQUE_LOADER_ID);
if (loader == null) {
    loaderManager.initLoader(UNIQUE_LOADER_ID, asyncTaskLoaderParams, this);
} else {
    loaderManager.restartLoader(UNIQUE_LOADER_ID,asyncTaskLoaderParams,this);
}
```

- Инициализация AsyncTaskLoader с помощью LoaderManager

```
LoaderManager loaderManager = getSupportLoaderManager();
Loader<String> loader = loaderManager.getLoader(UNIQUE_LOADER_ID);
loaderManager.initLoader(UNIQUE_LOADER_ID, null, this);
```

Лабораторная работа 5.

- Заменить AsyncTask на AsyncTaskLoader (из лабораторной работы 2 или 3)
- Реализовать запись в лог и в TextView в рассмотренных методах жизненного цикла Activity; посмотреть, чем отличается вывод, если для поворота экрана и сворачивания окна:
  - Не использовать сохранение содержимого TextView
  - Использовать для сохранения onSavedInstanceState
  - Использовать для сохранения статическую переменную