Василов Артур

e·legion

GDG Kazan

Android Developer at e-Legion

Google Developers Group Kazan

vasilovartur@gmail.com

@ArturVasilov

Введение в курс

Цели курса

1) Изучить вопросы и проблемы, связанные с разработкой клиент-серверных приложений

2) Изучить наиболее действенные и популярные методы построения архитектуры приложений, а также научиться применять их на практике

Формат курса

1) Занятие – 2-2,5 часа
2) Лекция – 60-80 минут
3) Практика – 40-60 минут
4) Дополнительная лекция – 15-20 минут
5) Командная работа над финальным проектом

Рассматриваемые темы

1) Обработка смены конфигурации в Activity
2) Классические паттерны A/B/C для обеспечения клиент-серверного взаимодействия
3) Фреймворк RxJava и его применение в Android-разработке
4) Принципы Clean Architecture и их применение в построении архитектуры приложений

Рассматриваемые темы

5) Паттерн MVP

6) Unit-тестирование Android-приложений

7) UI-тестирование Android-приложений

8) Фреймворк DataBinding и его использование в паттерне MVVM

Дополнительные темы

1) Проблема Backpressure в RxJava и способы ее решения
2) Библиотека Google Agera для построения архитектуры приложений
3) Библиотека Mosby для построения архитектуры приложений
4) Методология TDD (test-driven development)

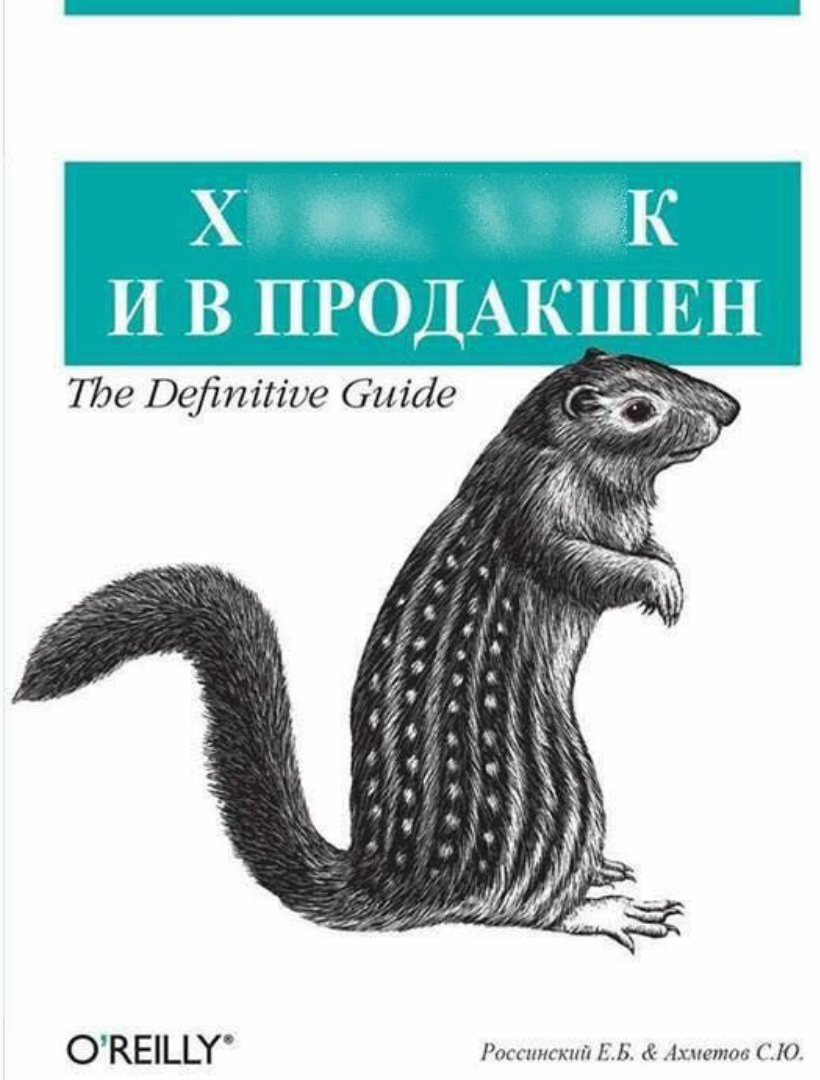
Командный проект

1) После 5-ого занятия
2) Команды по 3 человека
    a) Выбор идеи проекта
    https://github.com/toddmotto/public-apis
    b) Планирование задач по проекту

Введение в архитектуру клиент-серверных приложений

Зачем?

От
создателей
“и так сойдет”



2007


AT&T
9:42 AM
Text
Calendar
Photos
Camera
YouTube
Stocks
Maps
Weather
Clock
Calculator
Notes
Settings
Phone
Mail
Safari
iPod

2008

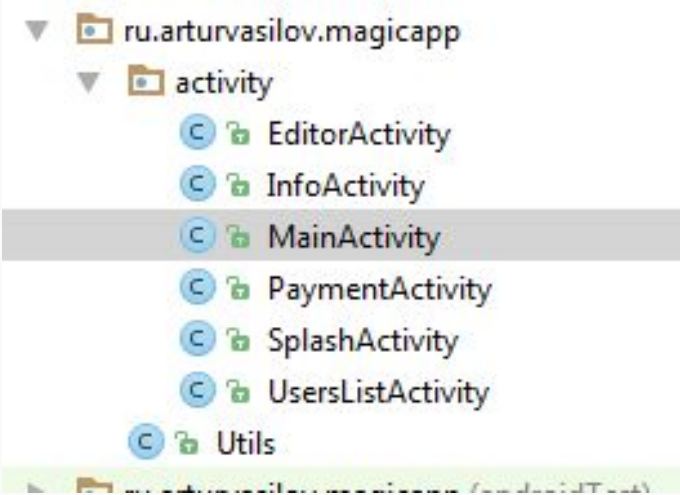

2:47 PM
myFaves
Dialer
Contacts
Browser
Maps
MENU

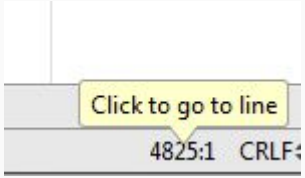
Так все же почему?

1) Крайне тяжело поддерживать

2) Невозможно писать Unit-тесты

MVC как AAA (Activity, Activity, Activity)





2014 - пора улучшить ситуацию

1) Концепция Material Design на конференции Google I/O

Fernando Cejas

2) Architecting Android...The clean way?

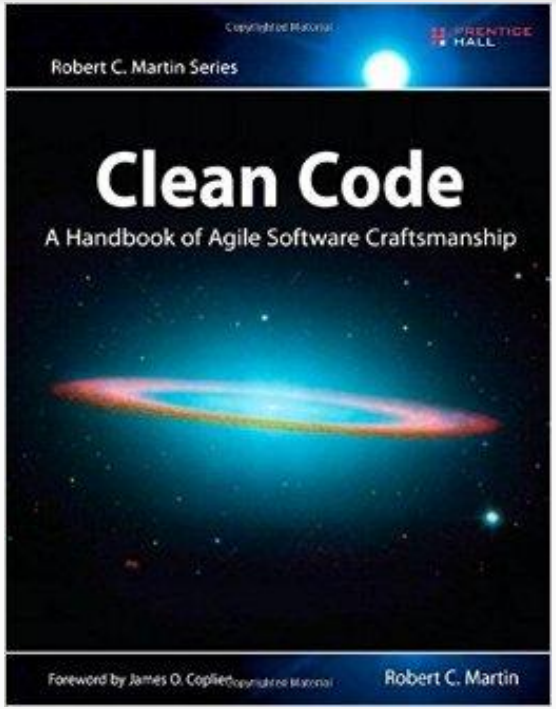
http://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/

Хорошая архитектура

1) Поддерживаемая, удобная, расширяемая и много других красивых прилагательных
2) Позволяет написание тестов

Много красивых прилагательных

1) Модульность
2) Четкое именование
3) Короткие функции и классы
4) WTF / min -> 0



Тестирование

1) Unit-тесты для бизнес-логики
2) Минимум зависимостей от Android-классов в тестируемых модулях
3) + Интеграционные тесты

Основные задачи при разработке клиент-серверных приложений

Клиент-серверные приложения

1) Организация клиент-серверного взаимодействия
2) Обеспечение возможности тестирования классов, содержащих бизнес-логику приложения

Это и есть основные задачи, которые мы будем изучать в ходе курса

Клиент-серверное взаимодействие

1) Обработка ошибок

2) Управление множеством запросов

3) Корректное управление закрытием приложения

Обеспечение возможности тестирования

1) Модульность классов

2) Минимальное число зависимостей от классов Android

3) Dependency Injection

Обработка смены конфигурации

Проблемы

1) Все поля в Activity уничтожаются при пересоздании Activity – нужно уметь восстанавливать их
2) Нельзя повторно выполнять запросы и другие “тяжелые” задачи

Запрет на смену ориентации

```
<activity
    android:name=".WeatherActivity"
    android:screenOrientation="portrait"/>
```

1) Поддержка только одной ориентации в приложении часто сказывается не лучшим образом на UX

2) Не покрывает всех ситуаций, при которых Activity пересоздается

Ручная обработка смены конфигурации

```
<activity
    android:name=".WeatherActivity"
    android:configChanges="orientation|keyboardHidden|screenSize"/>
```

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    // handle new configuration
}
```

Ручная обработка смены конфигурации

1) Система автоматически не меняет ресурсы (например, языковые ресурсы или ресурсы, зависящие от ориентации устройства)

Сохранение состояния в Bundle

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    if (mCity != null) {
        outState.putSerializable(WEATHER_KEY, mCity);
    }
}
```

Сохранение состояния в Bundle

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

    if (savedInstanceState == null || !savedInstanceState.containsKey(WEATHER_KEY)) {
        loadWeather();
    } else {
        mCity = (City) savedInstanceState.getSerializable(WEATHER_KEY);
        showWeather();
    }
}
```

Сохранение состояния в Bundle

1) Нельзя сохранять большие данные

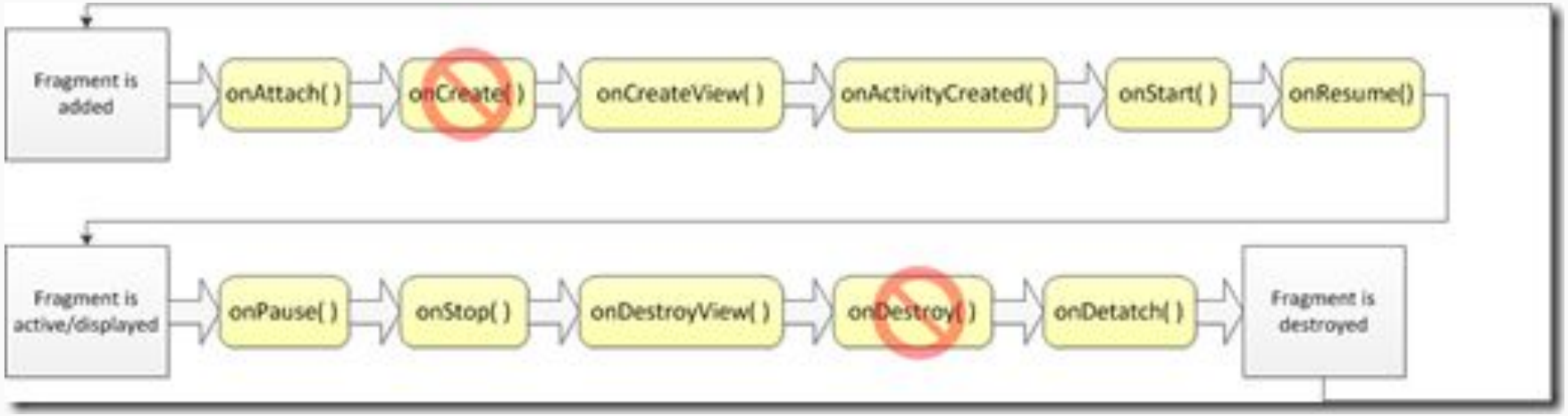
2) Вносит дополнительную непростую логику

Retain Fragment

```
public class WeatherFragment extends Fragment {

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }
}
```

Retain Fragment



Retain Fragment

```
@Override
public void onViewCreated(View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    if (mCity == null) {
        loadWeather();
    } else {
        showWeather();
    }
}
```

Retain Fragment

1) Нужна аккуратность в использовании ссылок на Activity / Context
2) При закрытии приложения уничтожается

Loader

1) Класс для загрузки данных

2) Переживает пересоздание Activity

3) Управляется специальным классом LoaderManager

Loader

```
public class StubLoader extends AsyncTaskLoader<Integer> {

    public StubLoader(Context context) {
        super(context);
    }

    @Override
    protected void onStartLoading() {
        super.onStartLoading();
        forceLoad();
    }

    @Override
    public Integer loadInBackground() {
        // emulate long-running operation
        SystemClock.sleep(2000);
        return 5;
    }
}
```

LoaderManager

```
public abstract <D> Loader<D> initLoader(int id, Bundle args,
LoaderManager.LoaderCallbacks< D> callback);
```

```
public abstract <D> Loader<D> restartLoader(int id, Bundle args,
LoaderManager.LoaderCallbacks< D> callback);
```

LoaderManager.LoaderCallbacks

```
public interface LoaderCallbacks<D> {

    public Loader<D> onCreateLoader(int id, Bundle args);

    public void onLoadFinished(Loader<D> loader, D data);

    public void onLoaderReset(Loader<D> loader);

}
```

LoaderManager.LoaderCallbacks

```
private class StubLoaderCallbacks implements LoaderManager.LoaderCallbacks<Integer> {

    @Override
    public Loader<Integer> onCreateLoader(int id, Bundle args) {
        if (id == R.id.stub_loader_id) {
            return new StubLoader(StubLoaderActivity.this);
        }
        return null;
    }

    @Override
    public void onLoadFinished(Loader<Integer> loader, Integer data) {
        if (loader.getId() == R.id.stub_loader_id) {
            Toast.makeText(StubLoaderActivity.this, R.string.load_finished, Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onLoaderReset(Loader<Integer> loader) {
        // Do nothing
    }
}
```

Запускаем работу лоадера

```
    setContentView(R.layout.activity_weather);
    getSupportLoaderManager().initLoader(R.id.stub_loader_id, Bundle.EMPTY, new StubLoaderCallbacks());
}

private class StubLoaderCallbacks implements LoaderManager.LoaderCallbacks<Integer> {

    @Override
    public Loader<Integer> onCreateLoader(int id, Bundle args) {
        if (id == R.id.stub_loader_id) {
            return new StubLoader(StubLoaderActivity.this);
        }
        return null;
    }
```

LoaderManager

```
public abstract <D> Loader<D> initLoader(int id, Bundle args,
LoaderManager.LoaderCallbacks< D> callback);
```

```
public abstract <D> Loader<D> restartLoader(int id, Bundle args,
LoaderManager.LoaderCallbacks< D> callback);
```

Загружаем данные в лоадере

```
public class WeatherLoader extends AsyncTaskLoader<City> {

    public WeatherLoader(Context context) {
        super(context);
    }

    @Override
    protected void onStartLoading() {
        super.onStartLoading();
        forceLoad();
    }

    @Override
    public City loadInBackground() {
        String city = getContext().getString(R.string.default_city);
        try {
            return ApiFactory.getWeatherService().getWeather(city).execute().body();
        } catch (IOException e) {
            return null;
        }
    }
}
```

Loader

```
private void loadWeather(boolean restart) {
    mWeatherLayout.setVisibility(View.INVISIBLE);
    mErrorLayout.setVisibility(View.GONE);
    mLoadingView.showLoadingIndicator();
    LoaderManager.LoaderCallbacks<City> callbacks = new WeatherCallbacks();
    if (restart) {
        getSupportLoaderManager().restartLoader(R.id.weather_loader_id, Bundle.EMPTY, callbacks);
    } else {
        getSupportLoaderManager().initLoader(R.id.weather_loader_id, Bundle.EMPTY, callbacks);
    }
}
```

Loader

```
protected void onStartLoading() {
}

protected void onForceLoad() {
}

protected void onStopLoading() {
}
```

Создаем свой лоадер

```
public class RetrofitWeatherLoader extends Loader<City> {

    private final Call<City> mCall;

    @Nullable
    private City mCity;

    public RetrofitWeatherLoader(Context context) {
        super(context);
        String city = context.getString(R.string.default_city);
        mCall = ApiFactory.getWeatherService().getWeather(city);
    }
```

Создаем свой лоадер

```
@Override
protected void onStartLoading() {
    super.onStartLoading();
    if (mCity != null) {
        deliverResult(mCity);
    } else {
        forceLoad();
    }
}
```

Загружаем данные

```
@Override
protected void onForceLoad() {
    super.onForceLoad();
    mCall.enqueue(new Callback<City>() {
        @Override
        public void onResponse(Call<City> call, Response<City> response) {
            mCity = response.body();
            deliverResult(mCity);
        }

        @Override
        public void onFailure(Call<City> call, Throwable t) {
            deliverResult(null);
        }
    });
}
```

Создаем свой лоадер

```
@Override
protected void onStopLoading() {
    mCall.cancel();
    super.onStopLoading();
}
```

Loader

1) При закрытии приложения уничтожается

2) Неудобно обрабатывать ошибки

3) Много кода

Больше примеров в статье

Практика

Погода в нескольких городах

1) Проект LoaderWeather. Описание задачи в файле ru.gdgkazan.simpleweather.screen.weatherlist.WeatherListActivity
2) Нужно загрузить погоду во всех городах при старте приложения
3) Сделать это наиболее быстрым способом (не каждый город последовательно)
4) Возможность обновления через SwipeRefreshLayout

Практика 2 - RxJava Loader

Реализуйте обертку в виде лоадера над RxJava (над Observable) таким образом, чтобы можно было сохранить мощь использования RxJava и при этом воспользоваться средствами лоадеров для обработки пересоздания Activity