

# ANDROID KOMUNIKACJA HTTP

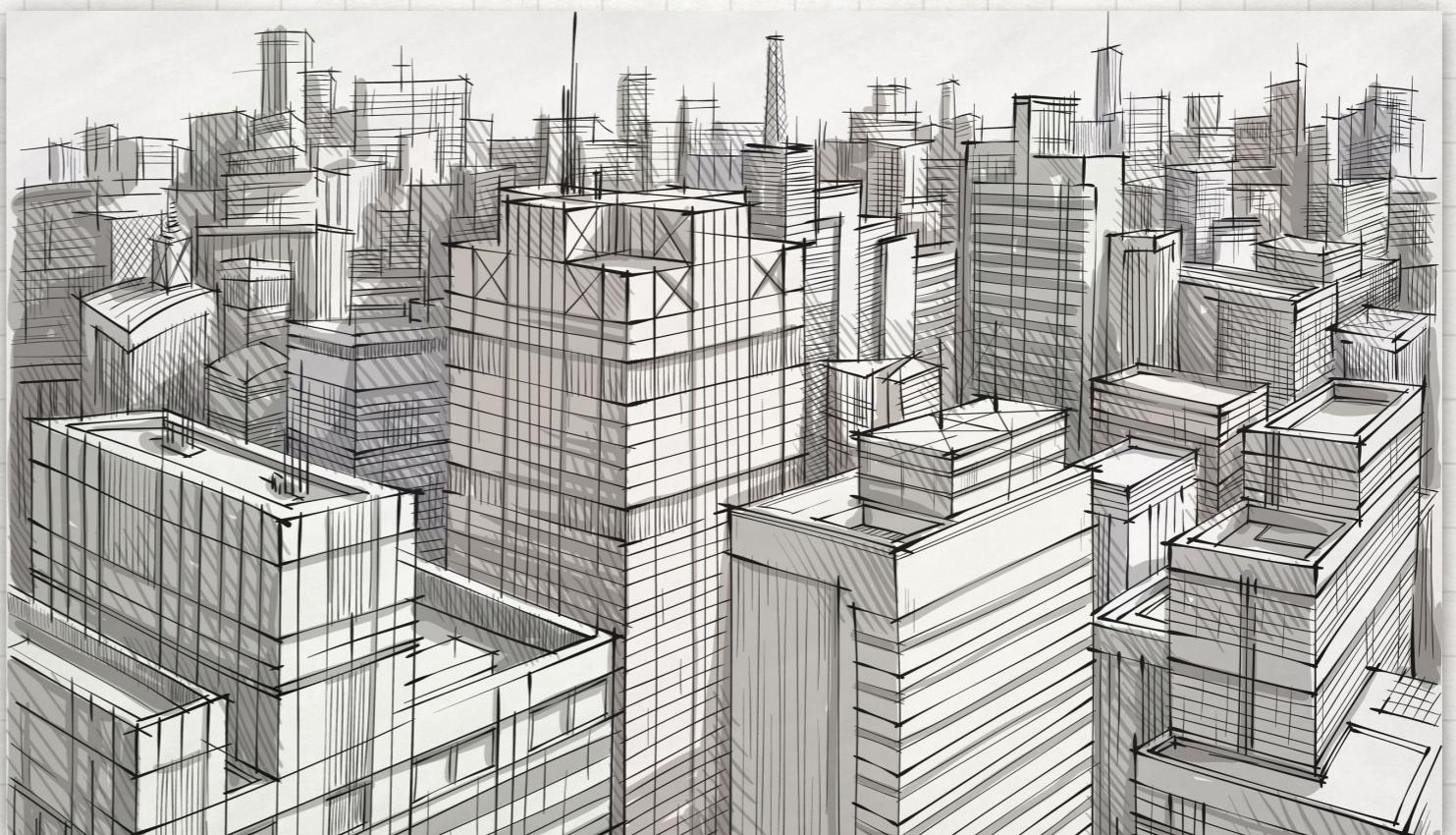
Software Development Academy

Max Jaglak

komunikacja sieciowa na androidzie - trochę teorii (wątki, uprawnienia, problemy)

Async task + HttpURLConnection

Inne (proste) opcje wątków (intent service / loader)



# HTTP + JSON

- Standardowe HTTP - biblioteki, łatwo obsłużyć
- HTTPS - szyfrowanie połączenia
- JSON - wygodny format danych, łatwo się mapuje na obiekty
- Wszystko musi działać w wątku w tle -  
`NetworkOnMainThreadException`
- Wymagane uprawnienia - `android.permission.INTERNET`

# WĄTKI

- Wszystko co dostępne w Javie
- Problem blokowania wątku głównego (ANR),  
NetworkOnMainThreadException
  - Operacje wykonywane w tle, ale wynik wyświetlany w wątku głównym!
  - AsyncTask - dobrodziejstwo androida :)
  - Inne możliwości - Loaders / Intent Service

# ĆWICZENIE - ASYNCTASK

1. Stworzyć AsyncTask który zmienia tekst po zadanym czasie
2. Przekazanie parametru
3. Wyświetlanie “progresu”
  - Klasa generyczna - do czego te typy danych?
  - doInBackground
  - pre i postExecute - wątek główny :)
  - progress
  - co jeśli operacja się nie uda?
  - Szeregowanie zadań - async taski wykonują się po kolej!

# HTTP URL CONNECTION

- Wywołanie zapytania HTTP w najprostszy sposób.
- <https://developer.android.com/reference/java/net/HttpURLConnection.html> - dokumentacja

# ĆWICZENIE - ZAPYTANIE HTTP Z UŻYCIEM HTTPURLCONNECTION

1. Wywołać podaną metodę GET przy użyciu AsyncTask'a
2. Wyświetlić wynik na ekranie
3. Wyłączyć internet i zobaczyć co będzie - trzeba złapać wyjątek



```
URL url = new URL("http://www.android.com/");
HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
try {
    InputStream in = new BufferedInputStream(urlConnection.getInputStream());
    readStream(in);
} finally {
    urlConnection.disconnect();
}
```

## Ćwiczenie - Async task + HttpURLConnection

1. Dodać Gson'a do projektu
2. Pobrać dane w formacie JSON z adresu poniżej
3. Sparsować listę obiektów (GSON)
4. Wyświetlić listę nazw (pole "name") na TextView, oddzielone nową linią

<http://91.134.143.223:9000/asset/test.json?X-BAASBO>

compile group: 'com.google.code.gson', name: 'gson', version: '2.8.0'

```
Gson gson = new Gson();
List<Book> books = gson.fromJson(responseBody, new TypeToken<List<Book>>(){}.getType());
```

```
public class Book {
    private final String name;
    public Book(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

podstawy retrofita

Api Client Factory

@GET/POST/PUT/DELETE

- Retrofit - popularna biblioteka do obsługi zapytań HTTP
- Nie tylko android (java też)
- <https://square.github.io/retrofit/>
- Używamy GSON'a (kolejna biblioteka) do mapowania JSON'a na obiekty

## Przykład / ćwiczenie - stworzenie obiektu retrofit'a + proste zapytanie

1. Deklaracja interface'u
2. Stworzenie klienta
3. Wywołanie zapytania
4. Obsługa odpowiedzi serwera
5. Różnica między błędnią odpowiedzią a wyjątkiem

```
compile 'com.squareup.retrofit2:retrofit:(insert latest version)'

public interface GitHubService {
    @GET("users/{user}/repos")
    Call<List<Repo>> listRepos(@Path("user") String user);
}

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com")
    .addConverterFactory(GsonConverterFactory.create())
    .build();

GitHubService service = retrofit.create(GitHubService.class);

Call<List<Repo>> repos = service.listRepos("octocat");

@POST("users/new")
Call<User> createUser(@Body User user);
```

## Przykład - Jak logować komunikacje z serwerem?

```
public OkHttpClient buildOkHttpClient() {
    OkHttpClient.Builder httpClientBuilder = new OkHttpClient.Builder()
        .connectTimeout(30, TimeUnit.SECONDS)
        .readTimeout(30, TimeUnit.SECONDS)
        .writeTimeout(30, TimeUnit.SECONDS);

    httpClientBuilder.addInterceptor(getHttpLoggingInterceptor());

    return httpClientBuilder.build();
}

@NotNull
private HttpLoggingInterceptor getHttpLoggingInterceptor() {
    HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
    logging.setLevel(HttpLoggingInterceptor.Level.BODY);
    return logging;
}

public <T> T buildService(String apiUrl, Class<T> clazz) {
    OkHttpClient client = customOkHttpClientFactory.buildOkHttpClient();

    Retrofit retrofit = new Retrofit.Builder()
        .baseUrl(apiUrl)
        .addConverterFactory(GsonConverterFactory.create(new Gson()))
        .client(client)
        .build();

    return retrofit.create(clazz);
}
```

# RETROFIT - KONFIGURACJA

```
@GET("/plugin/book.list")
Call<BookListResponse> getBooksList();

@GET("/plugin/book.details")
Call<BookResponse> getBookDetails(@Query("id") String bookId);

@POST("/plugin/book.create")
Call<ResponseBody> createBook(@Body Book book);

@GET("/plugin/book.id/{id}")
Call<Book> getBookById(@Path("id") String id);
```

Metody: @GET, @POST, @PUT, @DELETE + inne (w zależności czego serwer wymaga)

Parametry: @Query (bez dodatku w ścieżce!), @Path (z wskazaniem w ścieżce {nazwa})

@Body (wysyłanie jako ciało requestu)

@Header (pojedyńczy nad parametrem), @Headers (jeden wiele, nad metodą)

# RETROFIT - FACTORY

```
public BookApiClient create() {  
  
    //blok odpowiedzialny za dodanie logowania requestów  
    HttpLoggingInterceptor httpLoggingInterceptor = new HttpLoggingInterceptor();  
    httpLoggingInterceptor.setLevel(HttpLoggingInterceptor.Level.BODY);  
  
    OkHttpClient.Builder builder = new OkHttpClient.Builder();  
    builder.addInterceptor(httpLoggingInterceptor);  
  
    Retrofit retrofit = new Retrofit.Builder()  
        .baseUrl("http://91.134.143.223:9000/") //adres serwera, można wyjąć do stałej,  
                                                // albo pola w konfiguracji  
        .addConverterFactory(GsonConverterFactory.create())  
        .client(builder.build())  
        .build();  
  
    return retrofit.create(BookApiClient.class); // BookApiClient – klasa z retrofita  
}
```

# RETROFIR - UŻYCIE W ASYNC TASK

```
public class GetBookDetailsAsyncTask extends AsyncTask<String, Void, Book>{

    private final OnBookDownloadedListener onBookDownloadedListener;
    private final BookApiClient bookApiClient;

    public GetBookDetailsAsyncTask(OnBookDownloadedListener onBookDownloadedListener) {
        this.onBookDownloadedListener = onBookDownloadedListener;
        bookApiClient = new RetrofitApiClientFactory().create(); // Użycie factory - patrz poprzednie slajdy
    }

    @Override
    protected Book doInBackground(String... strings) {
        try {
            //metoda wykonywana w wątku w tle - nie blokuje wątku UI!
            if(strings == null || strings.length == 0) {
                return null;
            }
            String bookId = strings[0];
            //tworzymy obiekt call przez wywołanie metody z interfejsu BookApiClient
            Call<BookResponse> bookDetails = bookApiClient.getBookDetails(bookId);
            //Uruchamiamy zapytanie do serwera przy pomocy execute
            Response<BookResponse> response = bookDetails.execute();
            //po otrzymaniu odpowiedzi, sprawdzamy, czy jest serwer zwócił poprawną odpowiedź.
            //Jeśli nie udało się nawiązać połączenia, to metoda execute wyrzuci wyjątek
            if(response.isSuccessful()) {
                //pobieramy dane z zapytania
                return response.body().getData();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(Book book) {
        //ta metoda wywołuje się ponownie w wątku UI
        onBookDownloadedListener.bookDownloaded(book);
    }

    public interface OnBookDownloadedListener { // interfejs będziemy implementować w Activity albo Fragmentie
        void bookDownloaded(Book book);
    }
}
```

```
private TextView name;
private TextView author;
private ImageView image;

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

    String bookId = getIntent().getStringExtra("bookId");

    if(bookId == null) {
        finish();
        return;
    }

    name = (TextView) findViewById(R.id.name);
    author = (TextView) findViewById(R.id.author);
    image = (ImageView) findViewById(R.id.image);

    //uruchomienie async taska
    new GetBookDetailsAsyncTask(this).execute(bookId);
}
```

## Wywołanie w activity

```
@Override
//metoda z interfejsu OnBookDownloadedListener (activity implementuje ten interfejs)
public void bookDownloaded(Book book) {
    //sprawdzamy, czy nie dostaliśmy null'a
    if(book == null) {
        //wyświetlenie błędu - w prawdziwej aplikacji do rozbudowy
        Toast.makeText(this, "failed to download details!", Toast.LENGTH_SHORT).show();
        //finish - zamyka activity
        finish();
        return;
    }

    //jeśli book != null to wyświetlamy wynik
    name.setText(book.getName());
    author.setText(book.getAuthor());

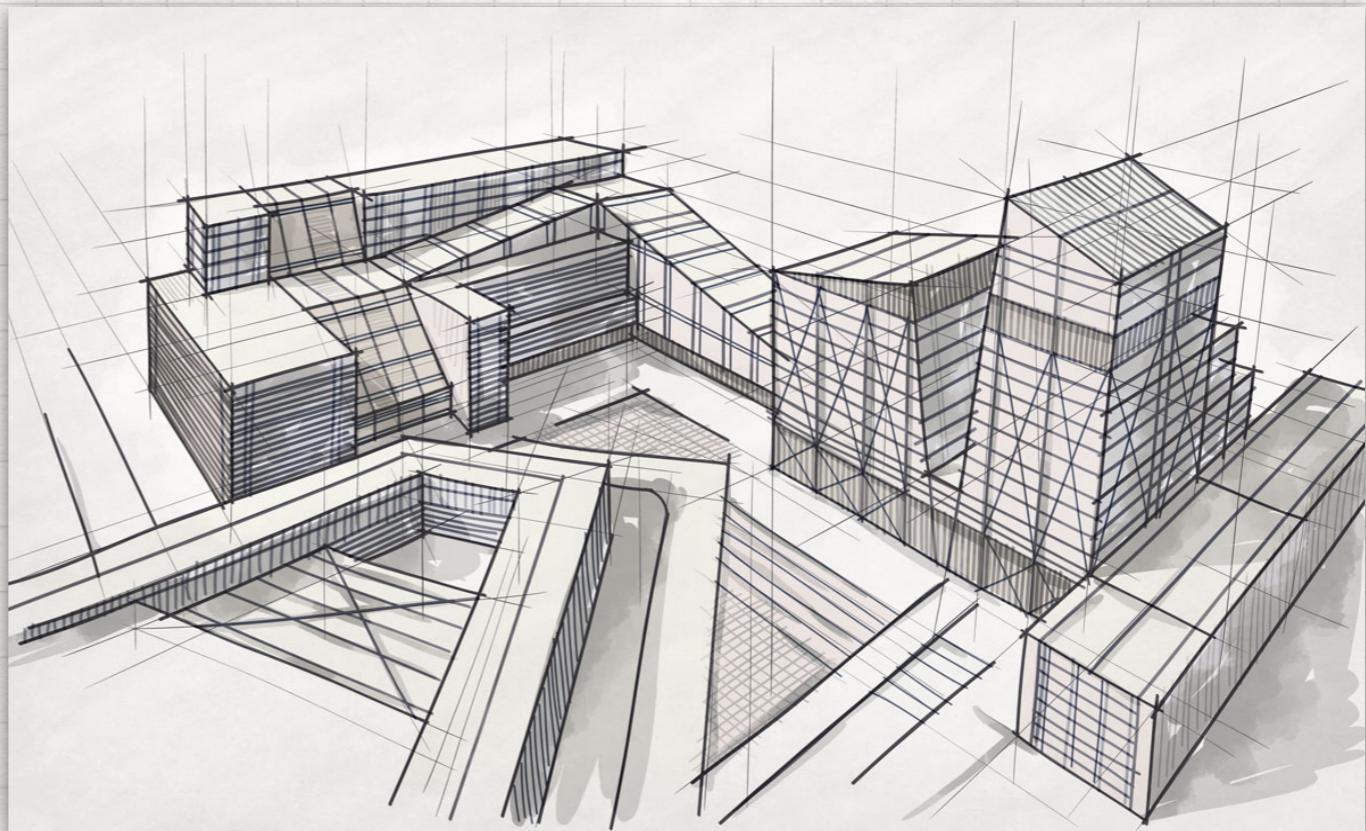
    //wyświetlenie obrazka przy użyciu Picasso
    Picasso.with(this)
        .load(book.getImageUrl())
        .fit()
        .centerCrop()
        .into(image);
}
```

## Obsługa wyniku operacji

pobieranie danych + ListView i Adapter

ListView - prosty ArrayAdapter i złożona implementacja BaseAdapter'a + problemy z ponownym wykorzystaniem widoków

RecyclerView - dlaczego lepszy + przykłady



- List View - koncepcja "Adapter View"
- Sposób na wyświetlenie dużej ilości danych w oszczędny sposób (pamięć, wydajność rysowania layoutów)
- Adapter i ponowne użycie tych samych widoków
  - Array adapter do najprostszych zastosowań
  - Base adapter - własna implementacja
  - View holder pattern
- Aktualnie w większości nowych implementacji ListView zastąpiona przez RecyclerView ( o tym za chwilę )

- RecyclerView - lepsze i nowsze rozwiązanie
- Automatyczne zarządzanie ponownym wykorzystaniem widoków
- wymuszony ViewHolder pattern
- Łatwa zmiana sposobu wyświetlania elementów (LayoutManager)
- Łatwa implementacja klasy adaptera

# RECYCLER VIEW - PRZYKŁAD

```
public class BookListAdapter extends RecyclerView.Adapter<BookListAdapter.BookListViewHolder> {

    private final Context context;
    private final List<BookListItem> bookListItems;
    private final OnBookItemClicked onBookItemClicked;
    private final LayoutInflater layoutInflater;

    public BookListAdapter(Context context, List<BookListItem> bookListItems, OnBookItemClicked onBookItemClicked) {
        this.context = context;
        this.bookListItems = bookListItems;
        layoutInflater = LayoutInflater.from(context);

        //on book clicked - callback przekazany i implementowany przez activity
        this.onBookItemClicked = onBookItemClicked;
    }

    @Override
    public BookListViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        //tworzymy widok i view holdera
        View view = layoutInflater.inflate(R.layout.view_book_item, parent, false);
        //do każdego view holdera przekazujemy callbacka
        return new BookListViewHolder(view, onBookItemClicked);
    }

    @Override
    public void onBindViewHolder(BookListViewHolder holder, int position) {
        BookListItem bookListItem = bookListItems.get(position);

        holder.bookName.setText(bookListItem.getName());

        //ustawiamy book list item, żeby móc przekazać go do callbacka kiedy użytkownik kliknie element
        holder.bookListItem = bookListItem;
    }

    public interface OnBookItemClicked {
        void bookClicked(BookListItem bookListItem);
    }

    @Override
    public int getItemCount() {
        return bookListItems.size(); // koniecznie zwracamy ilość elementów!
    }

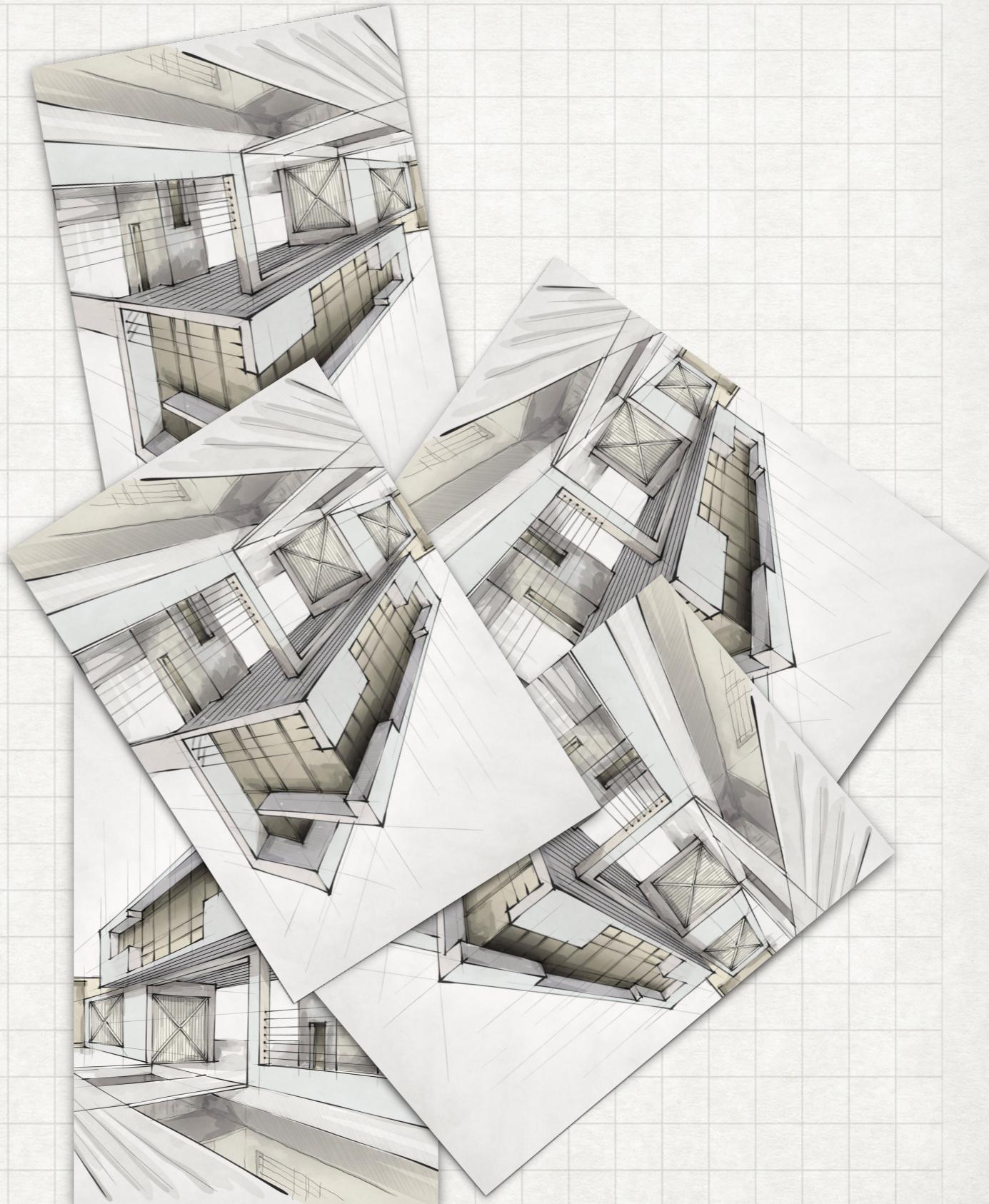
    public static class BookListViewHolder extends RecyclerView.ViewHolder {
        //view holder, trzyma widoki i reference do aktualnie wyświetlanego elementu
        private final OnBookItemClicked onBookItemClicked;
        private TextView bookName;
        private BookListItem bookListItem;

        public BookListViewHolder(View itemView, final OnBookItemClicked onBookItemClicked) {
            super(itemView);
            this.onBookItemClicked = onBookItemClicked;
            this.bookName = (TextView) itemView.findViewById(R.id.name);

            // ustawiamy click listenera na pojedyńczym elemencie, albo całym itemView
            itemView.setOnClickListener((view) -> {
                onBookItemClicked.bookClicked(bookListItem);
            });
        }
    }
}
```

## AsyncTask - obracanie ekranu

- Problem z restartowaniem tej samej operacji
- Problem z wykorzystaniem tego samego wywołania
- Czego NIE robić?
  - Własne zarządzanie zmianami konfiguracji
- Co robić?
  - Retainable Fragment
  - Przykłady



# RETROFIT - CALLBACKS

```
BookApiClient bookApiClient = new RetrofitApiClientFactory().create();

Call<BookListResponse> booksListCall = bookApiClient.getBooksList();

booksListCall.enqueue(new Callback<BookListResponse>() {
    @Override
    public void onResponse(Call<BookListResponse> call, Response<BookListResponse> response) {
        if(response.isSuccessful()) {
            bookListDownloaded(response.body().getData());
        } else {
            //todo show error
        }
    }

    @Override
    public void onFailure(Call<BookListResponse> call, Throwable t) {
        //todo show error
    }
});
```

# RETROFIT - REQUEST INTERCEPTOR

```
OkHttpClient.Builder builder = new OkHttpClient.Builder();
//uwaga, może być wiele interceptorów!
builder.addInterceptor(httpLoggingInterceptor);
builder.addInterceptor(createRequestInterceptor());

Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("http://91.134.143.223:9000/") //adres serwera, można wyjąć do stałej,
                                                // albo pola w konfiguracji
    .addConverterFactory(GsonConverterFactory.create())
    .client(builder.build())
    .build();

return retrofit.create(BookApiClient.class); // BookApiClient – klasa z retrofita
```

```
private Interceptor createRequestInterceptor() {
    return new Interceptor() {
        @Override
        public Response intercept(Chain chain) throws IOException {
            Request request = chain.request();

            // tutaj możemy manipulować requestem, dodawać nowe nagłówki itp.
            request = request.newBuilder()
                .addHeader("Authentication", "Basic 0930923")
                .addHeader("Checksum", calculateChecksum())
                .build();

            Response response = chain.proceed(request); // w tym miejscu faktycznie wywołujemy request

            // tutaj możemy podejrzeć response

            return response; // zwracamy request dalej
        }
    };
}
```