09

Enviando aluno para o servidor

Agora que criamos a `Call` executá-la. O nosso primeiro passo é ir até o `onOptionsItemSelected()` da `FormularioActivity`. Em seguida comente o código da `InsereAlunoTask` pois não precisaremos mais utilizá-la! Agora faça a chamada para a classe `RetrofitInicializador`:

```
// new InsereAlunoTask(aluno).execute();
new RetrofitInicializador();
```

Pegando o service do aluno

Veja que atualmente estamos apenas realizando a configuração do Retrofit, ou seja, precisamos pegar o service do aluno! Para isso adicione o método `getAlunoService()` no mesmo código que realizou a instância da classe `RetrofitInicializador`, e então, utilize o atalho **Alt + Enter** e crie o método que devolve um `AlunoService`:

```
public AlunoService getAlunoService() {
    return null;
}
```

Agora que temos o método que devolve o `AlunoService`, basta apenas pedir para o atributo `retrofit` criar esse service por meio do método `create()` passando como parâmetro o valor `AlunoService.class`. Por fim, ao invés de retornar `null`, retorne o método `create()`.

Executando a Call

Agora que temos acesso ao service, basta apenas chamar o método `insere()` enviando o aluno como parâmetro. Em seguida, retorne a `Call` que foi configurada neste método:

```
Call<Void> call = new RetrofitInicializador().getAlunoService().insere(aluno);
```

Com a `Call` em mãos podemos executá-la! Para isso podemos fazer de duas formas, ou seja, chamar os seguintes métodos:

- **execute():** Realiza requisição síncroniza, ou seja, prende a thread principal.
- **enqueue():** Realiza requisição assíncrona, ou seja, cria uma thread separada da thread principal e executa em background.

Conforme vimos em aula, mesmo que o método `execute()` funcione, se o utilizarmos, o Android lançará uma exception, pois existe a possibilidade de travar a thread da Activity já que uma requisição via rede pode demorar! Ou seja, teríamos que utilizar algo como uma AsyncTask para que ele enviasse o aluno! Porém, como foi mencionado, o Retrofit resolve esse problema utilizando o `enqueue` que já executa em uma thread separada sem a necessidade de criar uma!

Implementando o Callback

Portanto, chame o método `enqueue()` e implemente a interface `Callback` via classe anônima:

```
Call<Void> call = new RetrofitInicializador().getAlunoService().insere(aluno);
call.enqueue(new Callback<Void>() {
 @Override
 public void onResponse(Call<Void> call, Response<Void> response) {

 }
 @Override
 public void onFailure(Call<Void> call, Throwable t) {

 }});
```

O callback é justamente o retorno que o Retrofit tem após realizar a requisição! Em outras palavras quando implementamos essa interface somos obrigados a implementar 2 métodos que possuem os seguintes significados:

- **onResponse:** A requisição foi realizada com "sucesso", ou seja, a App conseguiu conectar com o servidor.
- **onFailure:** Caso a requisição for realizada, porém, houve alguma falha na rede e por isso não foi possível conectar com o servidor.

Em outras palavras, para indicar que tudo foi como esperado adicione o seguinte Log informativo: `Log.i("onResponse", "requisicao com sucesso");` dentro do `onResponde()`. Então adicione também o Log de erro dentro do `onFailure()`: `Log.e("onFailure", "requisicao falhou");`

Corrigindo problema de duplicidade de arquivos

Caso tentemos rodar a nossa app atualmente, receberemos um problema de build do gradle indicando duplicidade de arquivos a partir do arquivo **META-INF/LICENSE**. Esse detalhe acontece por causa do plugin do Jackson que adicionamos! Para resolver esse problema basta apenas adicionar o código abaixo no arquivo **build.gradle** de nível app:

```
packagingOptions {
    exclude 'META-INF/LICENSE'
}
```

Testando a App

Agora que que foi implementado o `enqueue()` e o `Callback` e rode a aplicação e adicione um novo aluno novamente! Em seguida, verifique se o aluno foi adicionado no servidor.