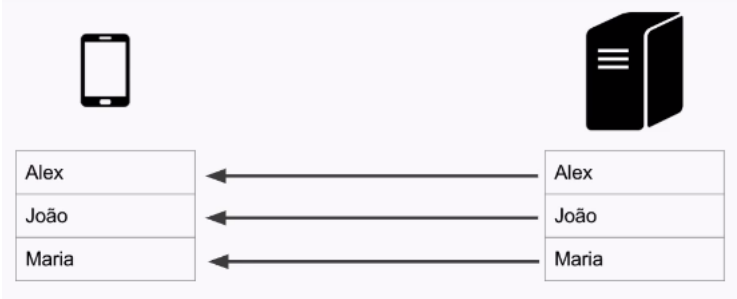
03

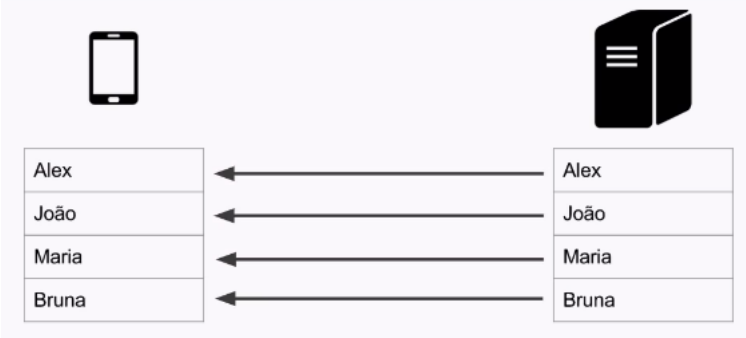
Introdução ao versionamento

Transcrição

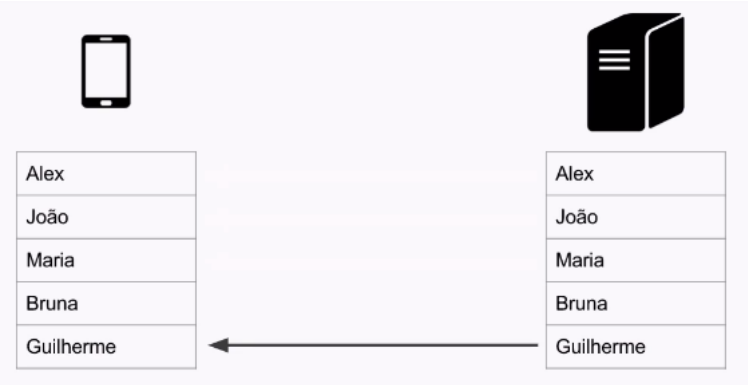
Até o momento, a nossa aplicação está funcionado muito bem. Mas tem um detalhe que é importante abordarmos. Quando a aplicação é instalada no celular, a app solicita ao servidor as informações de todos os alunos.



O que acontece se no servidor adicionarmos um novo aluno? A aplicação solicitará todos os alunos novamente.



É um processo que não faz muito sentido considerando que apenas uma única informação foi alterada no servidor. O ideal seria que o aplicativo solicitasse apenas a informação que ele não possui.

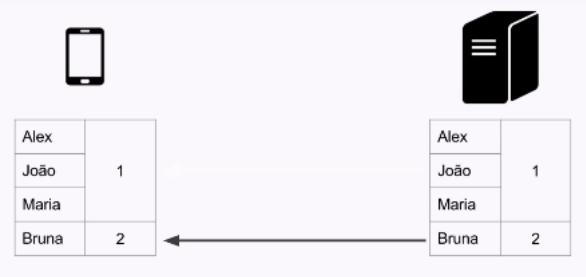


Como poderíamos implementar essa funcionalidade em nossa aplicação? Para fazermos isso, trabalharemos com um conceito chamado de Versionamento de Dados.

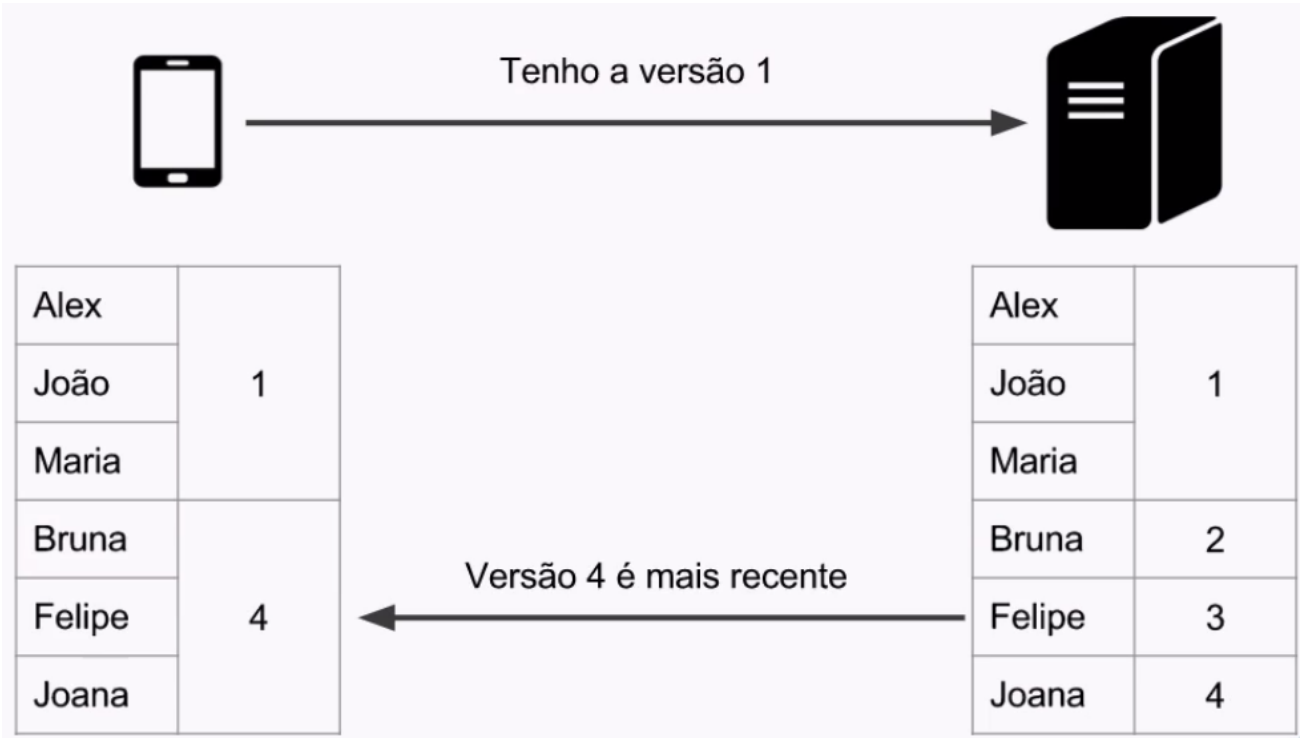
Vamos imaginar novamente o cenário onde a aplicação acabou de ser instalada e ainda não pegou nenhuma informação do servidor.

No servidor, as informações armazenadas vão possuir, por exemplo, um número de versão, que no caso seria `1`. Quando o celular solicitar os dados, o servidor vai mandar a versão **1** para o celular. Se acontecer a inserção de um novo

aluno, o servidor colocará nessa nova informação a versão **2**, assim o celular precisará apenas puxar a versão que ele ainda não possui.



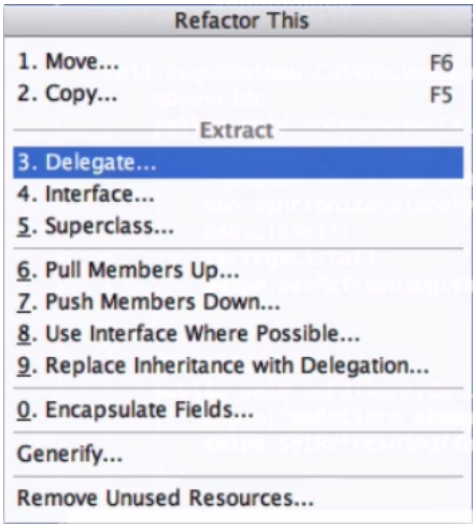
Mas vamos imaginar que no celular as informações estão na versão **1**, e o no servidor foram adicionados novos alunos, passando a ter as versões **2**, **3** e **4**, como o aplicativo vai pegar essas informações? Ele fará a solicitação informando que possui a versão **1**, e o servidor vai responder com a versão mais recente enviando todas as outras informações que ele ainda não possui.



Agora que aprendemos como funciona o versionamento de dados vamos começar a implementar de fato em nossa aplicação. Mas antes vamos fazer uma refatoração.

Na classe `ListaAlunosActivity`, acessaremos o método `buscaAlunos()`, no qual temos o código que pega os alunos do servidor a traz para a nossa aplicação. Só que esse tipo de comportamento não é responsabilidade da ***Activity***, então, vamos colocar o código em um lugar que **realmente** seja o responsável por isso.

Extrairemos todo método `buscaAlunos()`, esse tipo de processo é chamado de ***Delegate***, pois delegaremos todo o comportamento de um método para outra classe. Usando o atalho "Shift + Ctrl + Alt + T", um menu será aberto onde escolheremos a opção ***3 - Delegate***

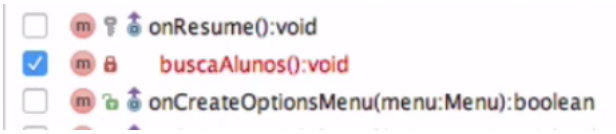


Após selecionar a opção *delegate*, cairemos em uma nova janela, onde colocamos algumas informações para a refatoração do método.

O primeiro passo é colocar no campo "Name for new class", o nome da classe para onde vamos refatorar o método, no caso, chamaremos de `AlunoSincronizador`.

No campo ***Package name***, nós colocaremos o nome do pacote onde essa nova classe será criada. Por padrão, ela vem como o diretório raiz, mas iremos colocar o pacote como `br.com.alura.agenda.sinc`.

Agora na área ***Members to Extract***, nós selecionaremos o método que desejamos refatorar. No caso, trabalharemos apenas o método `buscaAlunos():void` e depois, clicaremos em "***Refactor***".



Um campo surgirá informando que o campo ***swipe*** está sendo acessado dentro do método, ou seja, por conta da nova classe não ter o *swipe*, precisaremos lidar com isso. Mas no momento, vamos cliclar em "Continue" para finalizar a refatoração.

Veja que agora todo o comportamento do `buscaAlunos()` foi substituído por um objeto `alunoSincronizador` do tipo `AlunoSincronizador` que chama o método `buscaAlunos()`.

Vamos refatorar um pouco mais trocando o nome do objeto `alunoSincronizador`. Colocamos o *cursor* do editor em cima do objeto `alunoSincronizador` e usaremos o atalho "Shift + F6", agora é só renomear o objeto para `sincronizador`.

Nosso próximo passo será analisar o método `buscaAlunos()` na classe `AlunoSincronizador`. Veja que ele colocou todo o comportamento do método na nova classe. Mas ainda precisaremos trabalhar em alguns detalhes dessa classe.

O primeiro detalhe é o acoplamento que o *Delegate* fez no construtor da `AlunoSincronizador`, desta forma obriga que a classe `AlunoSincronizador` só funcione com a classe `ListaAlunoActivity`. Porém queremos que ela seja chamada por outras classes que necessitem de suas funcionalidades.

```
// ...

private final ListaAlunosActivity listaAlunosActivity;

public AlunoSincronizador(ListaAlunosActivity listaAlunosActivity){
    this.listaAlunosActivity = listaAlunosActivity;
}

// ...
```

Precisamos generalizar a classe que recebemos no construtor. A forma mais genérica é usar a `Context`. Vamos alterar o tipo `ListaAlunoActivity` que recebemos no construtor para o tipo `Context` e mudamos o nome da variável utilizando o atalho "Alt + F6" para `context`. Também alteramos o atributo mudando o tipo para `Context` e o nome com "Alt + F6" para `context`. Quando utilizamos o atalho "Alt + F6", nós refatoramos a variável, em todos os pontos em que ela foi utilizada na classe.

```
// ...

private final Context context;

public AlunoSincronizador(Context context){
    this.context = context;
}

// ...
```

Como o nosso atributo é um `Context`, o código quebrou em alguns métodos que eram chamados a partir do antigo objeto `listaAlunosActivity`.

Vamos começar resolvendo o `context.carregaLista()`. Para resolvermos esse problema sem criar um acoplamento com a classe `ListaAlunoActivity`, podemos fazer uso do *EventBus*. Começaremos pegando uma referência do *EventBus*.

```
// ...

private final Context context;
private EventBus bus = EventBus.getDefault();

// ...
```

Com o nosso objeto `bus`, vamos disparar um evento pedindo para atualizar a lista. Então usamos `bus.post(new AtualizaListaAlunoEvent());`. Nesse momento, `context.carregaLista()` não será mais necessário, podendo ser apagado.

```
// ...

public void buscaAlunos(){
    Call<AlunoSync> call = new RetrofitInicializador().getAlunoService().lista();

    call.enqueue(new Callback<AlunoSync>(){
```

```
        @Override
        public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response){
            AlunoSync alunoSync = response.body();
            AlunoDAO dao = new AlunoDAO(context);
            dao.sincroniza(alunoSync.getAlunos());
            dao.close();
            bus.post(new AtualizaListaAlunoEvent());
            context.getSwipe().setRefreshing(false);
        }


        @Override
        public void onFailureCall(Call<AlunoSync> call, Throwable t){
            Log.e("onFailure chamado", t.getMessage());
            context.getSwipe().setRefreshing(false);
        }

    });
}

// ...
```

O código ainda está quebrado na chamada do método `context.getSwipe().setRefreshing(false);` . Nós estavamos usando o *Swipe* para tirar o *refreshing* do aplicativo.

Mas como vamos fazer para continuar retirando o esse *refreshing*? No momento em que carregamos a lista, podemos fazer uma verificação dentro do evento que atualiza a lista.

Na classe `ListaAlunosActivity` , podemos ir no método `atualizaListaAlunoEvent()` e fazer uma verificação do tipo `if(swipe.isRefreshing()) swipe.setRefreshing(false);` .

```
// ...

@Subscribe(threadMode = ThreadMode.MAIN)
public void atualizaListaAlunoEvent(AtualizaListaAlunoEvent event){
    if(swipe.isRefreshing()) swipe.setRefreshing(false);
    carregaLista();
}

// ...
```

Como o nosso próprio evento que atualiza a lista está cuidando do *swipe*, podemos removê-lo do método `onResponse()` na classe `AlunoSincronizador` .

```
// ...

@Override
public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response){
    AlunoSync alunoSync = response.body();
    AlunoDAO dao = new AlunoDAO(context);
    dao.sincroniza(alunoSync.getAlunos());
    dao.close();
    bus.post(new AtualizaListaAlunoEvent());
```

```
    }

    // ...
```

Agora no método `onFailure()`, para corrigir o problema do `Log`, basta importá-lo. E como já não precisamos mais usar o *swipe*, podemos remover a linha `context.getSwipe().setRefreshing(false);`, e depois, colocaremos `bus.post(new AtualizaListaAlunoEvent());`.

```
    // ...

    public void buscaAlunos(){
        Call<AlunoSync> call = new RetrofitInicializador().getAlunoService().lista();

        call.enqueue(new Callback<AlunoSync>(){
            @Override
            public void onResponse(Call<AlunoSync> call, Response<AlunoSync> response){
                AlunoSync alunoSync = response.body();
                AlunoDAO dao = new AlunoDAO(context);
                dao.sincroniza(alunoSync.getAlunos());
                dao.close();
                bus.post(new AtualizaListaAlunoEvent());
            }


            @Override
            public void onFailureCall(Call<AlunoSync> call, Throwable t){
                Log.e("onFailure chamado", t.getMessage());
                bus.post(new AtualizaListaAlunoEvent());
            }

        });
    }

    // ...
```

Agora que terminamos a refatoração, vamos rodar a aplicação e ver se está tudo funcionando. Deixamos o Android Monitor aberto para monitorar e abrir o aplicativo, veremos os alunos foram chamados normalmente. Se tentarmos fazer um *swipe*, constataremos que também está funcionando.