Simplificando a manipulação de eventos com EventBus

Existem várias implementações diferentes para o **Eventbus**, por exemplo o **Otto**, pertencente à **Squareup**, a mesma que nos fornece o **Retrofit**. Usaremos em nosso projeto uma outra implementação, fornecida pela **Greenrobot**, que é a mais utilizada no mercado de trabalho. Para isso, vamos adicioná-la como dependência no arquivo **build.gradle** (Módulo):

dependencies {  
 ...  
 compile 'org.greenrobot:eventbus:3.0.0'  
 ...  
}

A primeira coisa que temos que fazer é registrar nossa Activity como ouvinte do evento e da mesma forma, temos que parar de ouvir no pouco antes da Activitymorrer. Temos uma classe especialista para fazer esse procedimento para nós, a própria classe Eventbus, para pegarmos um objeto desse tipo, há o método getDefault(), que retorna uma referência que será única durante toda a aplicação, que é um *Singleton*. Agora vamos ver como começar a ouvir e também parar de ouvir os eventos que o **Eventbus** irá disparar:

// MainActivity.java  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.chat\_activity);  
  
 ButterKnife.bind(this);  
  
 picasso.with(this).load("https://api.adorable.io/avatars/285/" + idDoCliente + ".png").into(avatar);  
  
 ChatApplication app = (ChatApplication) getApplication();  
 component = app.getComponent();  
 component.inject(this);  
  
 mensagens = new ArrayList<>();  
  
 MensagemAdapter adapter = new MensagemAdapter(idDoCliente, mensagens, this);  
 listaDeMensagens.setAdapter(adapter);  
  
 ouvirMensagem();  
  
 // vamos começar a ouvir o evento, por isso passamos o this  
 EventBus.getDefault().register(this);  
  
}  
  
@Override  
protected void onStop() {  
 super.onStop();  
  
 // deixando claro que não queremos mais ouvir o evento  
 EventBus.getDefault().unregister(this);  
}

Agora precisamos dar um jeito de enviar a mensagem que recebemos, para que o próprio **EventBus** consiga disparar isso para quem o está ouvindo, e é bem simples fazer isso. Precisamos chamar o comportamento que fará com que a mensagem seja disparada, para isso faremos uso do método **post()**, que precisará receber por parâmetro a mensagem, para isso precisamos vamos criar no pacote br.com.caelum.ichat.event uma classe que possua os dados:

public class MensagemEvent {  
  
 public Mensagem mensagem;  
  
 public MensagemEvent(Mensagem mensagem) {  
 this.mensagem = mensagem;  
 }  
  
}

E agora, na classe **OuvirMensagemCallback**:

// OuvirMensagemCallback.java  
  
@Override  
public void onResponse(Call<Mensagem> call, Response<Mensagem> response) {  
 if(response.isSuccessful()) {  
 Mensagem mensagem = response.body();  
  
 EventBus.getDefault().post(new MensagemEvent(mensagem));   
 }  
  
}

Remova também o atributo do tipo MainActivity do callback.

Falta apenas conseguir capturar esse *post* que estamos realizando, seria bem interessante se conseguirmos reaproveitar o comportamento que deixamos com o método do delegate:

// MainActivity.java  
  
public void colocaNaLista(Mensagem mensagem) {  
 mensagens.add(mensagem);  
  
 MensagemAdapter adapter = new MensagemAdapter(idDoCliente, mensagens, this);  
 listaDeMensagens.setAdapter(adapter);  
  
 ouvirMensagem();  
}

E de fato, a única coisa que precisamos fazer é adicionar ao método a anotação @Subscribe do *EventBus* e receber um **MensagemEvent** por parâmetro:

@Subscribe  
public void colocaNaLista(MensagemEvent mensagemEvent) {  
 mensagens.add(mensagemEvent.mensagem);  
  
 MensagemAdapter adapter = new MensagemAdapter(idDoCliente, mensagens, this);  
 listaDeMensagens.setAdapter(adapter);  
  
 ouvirMensagem();  
}

Vamos anotar o método ouvirMensagens também com @Subscribe e fazer ele receber também o evento de mensagem:

@Subscribe  
public void ouvirMensagem(MensagemEvent mensagemEvent) {  
 Call<Mensagem> call = chatService.ouvirMensagens()  
 call.enqueue(new OuvirMensagemCallback(this));  
}

Nesse momento, o código da MainActivity irá quebrar, já que ele tenta chamar **ouvirMensagem()**. Nesse caso, vamos passar um **MensagemEvent** vazio:

// MainActivity.java  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.chat\_activity);  
  
 ButterKnife.bind(this);  
  
 picasso.with(this).load("https://api.adorable.io/avatars/285/" + idDoCliente + ".png").into(avatar);  
  
 ChatApplication app = (ChatApplication) getApplication();  
 component = app.getComponent();  
 component.inject(this);  
  
 mensagens = new ArrayList<>();  
  
 MensagemAdapter adapter = new MensagemAdapter(idDoCliente, mensagens, this);  
 listaDeMensagens.setAdapter(adapter);  
  
 ouvirMensagem(new MensagemEvent());  
  
 EventBus.getDefault().register(this);  
  
}

Rode a aplicação e veja que tudo continua funcionando com sucesso, agora sem acoplamento :)