



PROGRAMAÇÃO PARA
DISPOSITIVOS MÓVEIS

COMUNICAÇÃO WEB II

RETROFIT

O que é o retrofit?

- Tradicionalmente, “retrofit” é um termo comum utilizado em engenharia para especificar a modernização de algum equipamento já considerado ultrapassado ou fora de norma.

O que é o retrofit?

- Retrofit também é o nome de um framework Java (e Android) para consumo de APIs desenvolvidas em REST
- Através deste framework, é possível estabelecer comunicação e consumir serviços implementados, por exemplo, com Jersey

Vantagens do retrofit

- O retrofit fornece uma plataforma muito mais robusta e simples de utilizar em relação a chamadas REST com apenas elementos nativos do Android (Network Calls, AsyncTasks e URL/HttpClient/Connection), vistos na aula passada
- Daí o nome “retrofit”

Como o retrofit funciona?

- O retrofit precisa de alguns elementos básicos para sua execução:
 - Classe de modelo que simboliza os elementos a serem recebidos/enviados do/ao serviço remoto
 - Interface definindo os serviços remotos disponíveis para o cliente
 - Callback de tratamento do retorno do serviço

Como o retrofit funciona?

- Através destes elementos, o retrofit poderá ser utilizado seguindo os passos a seguir:
 1. Cria-se uma instância do retrofit que aponte para a URL onde encontram-se os serviços. Neste momento, também é necessário especificar qual será o “conversor” de dados (normalmente Json ou XML)

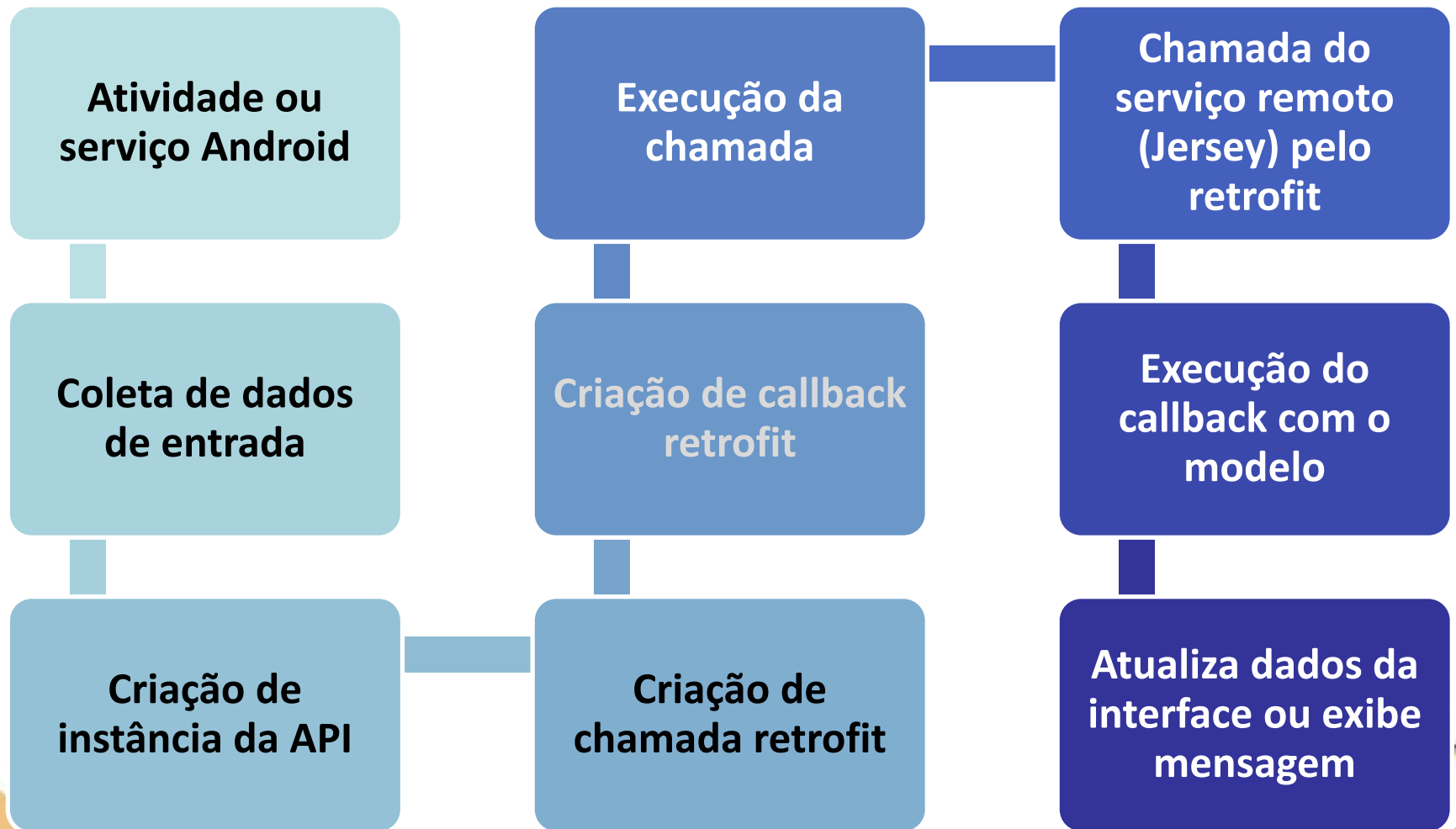
Como o retrofit funciona?

2. Com a instância do retrofit configurada, cria-se uma “instância” da API para chamada remota do serviço com base na definição da interface de API
3. Tendo a instância da API, podemos criar uma chamada para a mesma. Esta chamada é baseada no tipo de dados do modelo e espera os parâmetros de entrada especificados na interface da API

Como o retrofit funciona?

4. Define-se um callback, a ser executado quando o serviço finalizar (com sucesso ou erro). O callback receberá o elemento de modelo indicado e precisará tratar a resposta (por exemplo, atualizar a interface com os dados recebidos)
5. O serviço retrofit é executado. Quando concluir, o callback indicado será chamado

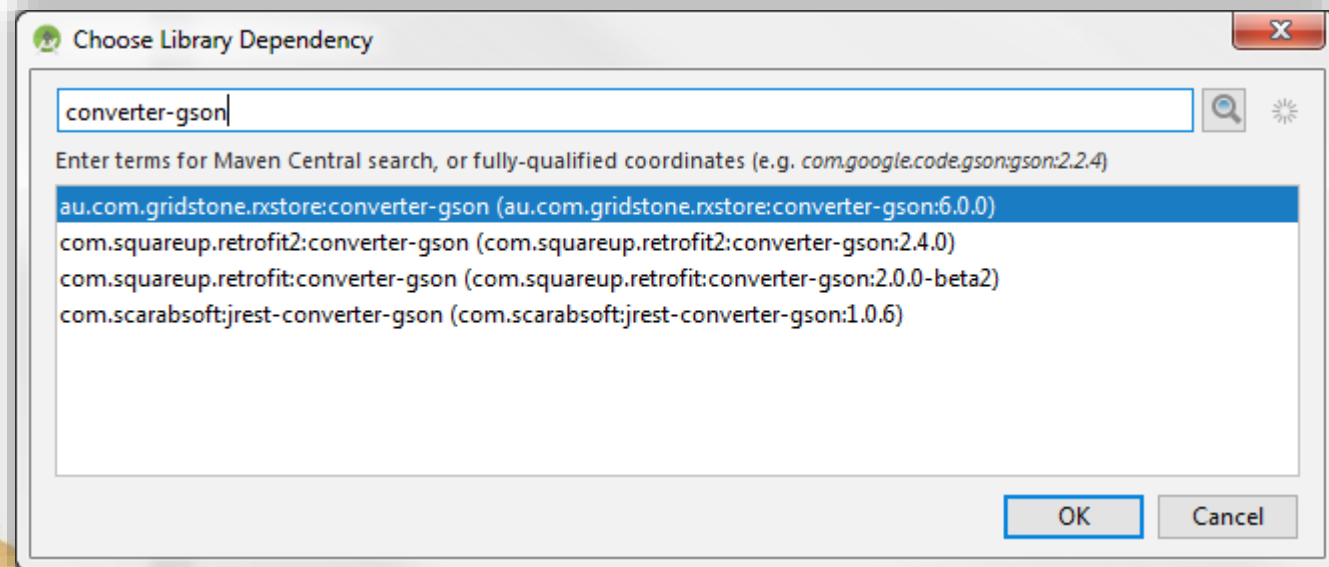
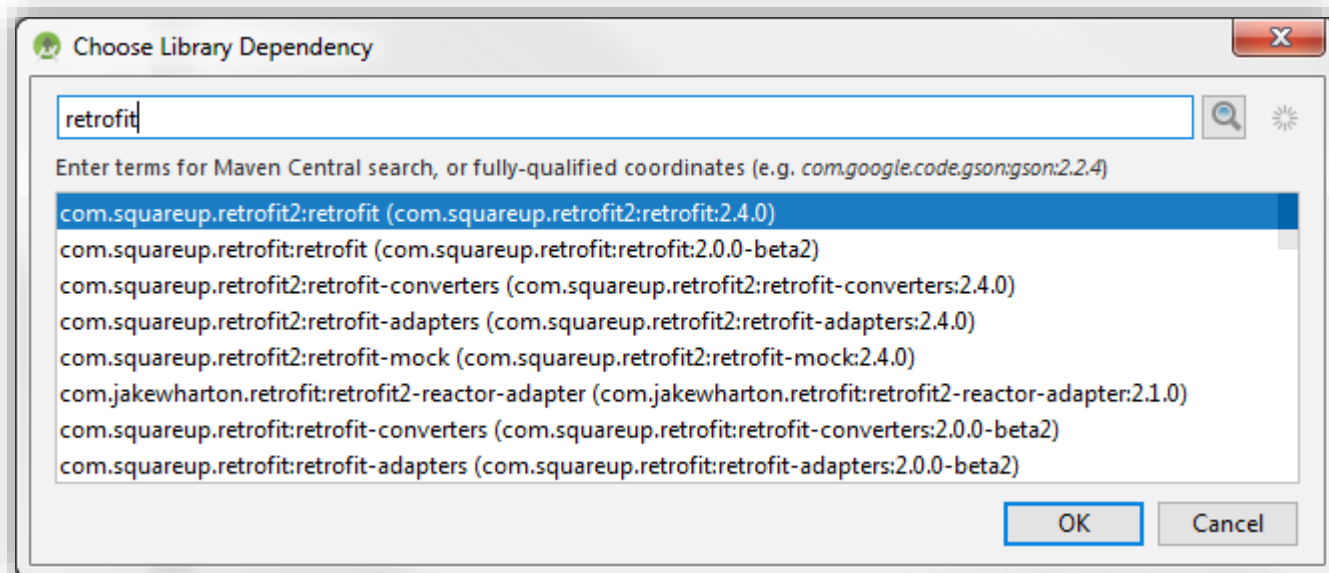
Como o retrofit funciona?



Como adicionar o retrofit ao projeto?

- O retrofit não está disponível por padrão no Android. Para utiliza-lo, é necessário incluir a biblioteca do framework nas dependências de módulo (assim como fizemos com o material design)
- É preciso adicionar suporte ao “retrofit” e ao conversor de dados Json, “converter-gson”

Como adicionar o retrofit ao projeto?



Elementos do retrofit

- Como vimos, precisamos de alguns elementos para utilizar o retrofit:
 - Modelo
 - Interface da API
- Além disso, na utilização, será preciso
 - Criar uma instância da chamada
 - Criar um callback de chamada e associá-lo a instância
 - Executar a chamada

Classes de modelo

- O primeiro requisito para utilização do retrofit são as classes de modelo
- Classes de modelo (também conhecidas como classes de negócio ou POJOs), são classes que representam entidades do negócio da aplicação

Classes de modelo

- Classes de modelo tem relação direta com a finalidade do negócio cuja aplicação participa
- Por exemplo, se estamos falando de uma aplicação que permite cadastrar e realizar manutenção em dados de clientes, é muito provável que exista um modelo “Cliente”

Classes de modelo

- Uma classe de modelo tem estado (atributos) e métodos de acesso (sets e gets)
- Os atributos variam de acordo com cada entidade, tanto em nomenclatura quanto em tipo e permissões de acesso (leitura – apenas get, escrita – apenas set ou leitura e escrita – set e get)

Exemplo de classe de modelo

```
public class Usuario {  
    private String nomeUsuario;  
    private String senha;  
    private Date dataCadastro;  
  
    public String getNomeUsuario() {  
        return nomeUsuario;  
    }  
  
    public void setNomeUsuario(String nomeUsuario) {  
        this.nomeUsuario = nomeUsuario;  
    }  
  
    public String getSenha() {  
        return senha;  
    }  
  
    public void setSenha(String senha) {  
        this.senha = senha;  
    }  
  
    public Date getDataCadastro() {  
        return dataCadastro;  
    }  
  
    public void setDataCadastro(Date dataCadastro) {  
        this.dataCadastro = dataCadastro;  
    }  
}
```

Definição da interface da API

- Além do modelo, será necessário criar uma interface que especifique os métodos disponíveis no serviço remoto
- A interface poderá especificar quais métodos HTTP serão utilizados para chamada dos serviços remotos, bem como dados de entrada (parâmetros) e saída (retorno)

Definição da interface da API

- O elemento de API utilizado pelo retrofit normalmente não precisa ser uma classe Java, apenas uma interface. Desta forma, não possuirá implementações específicas, apenas as definições de métodos e quais pontos da API remota serão acionados quando estes forem chamados

Exemplo de interface de API

```
public interface ApiUsuario {  
  
    @GET("/usuario/{usuario}")  
    Call<Usuario> getUsuario(@Path("usuario") Integer id);  
  
    @POST("/usuario")  
    Call<Usuario> inserirUsuario(@Body Usuario usuario);  
  
}
```

Criando e executando uma chamada

- Para realizar a chamada ao serviço, será necessário
 1. Criar uma instância do retrofit, apontando para o endereço da API e configurando um conversor
 2. Criar uma instância da API
 3. Criar uma chamada usando a instância da API
 4. Criar um callback
 5. Executar o serviço, associando um callback a chamada

Criando uma instância do retrofit

- É possível obter uma instância do retrofit através da chamada abaixo:

```
Retrofit instanciaRetrofit = new Retrofit.Builder()  
    .baseUrl("URL_DO_SERVICO_REMOTO")  
    .addConverterFactory(CONVERSOR_DE_DADOS.create())  
    .build();
```

Criando uma instância do retrofit

- Para, por exemplo, criar uma instância do retrofit para uma API localizada no servidor “minhaapi”, que devolve dados em formato Json, é necessário executar:

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("http://minhaapi.com.br/")  
    .addConverterFactory(GsonConverterFactory.create())  
    .build();
```

Criando uma instância da API

- Para criar uma instância da API, utilizamos a instância do retrofit para obter uma instância de API com base na interface definida anteriormente

```
InterfaceApi api =  
    retrofit.create(InterfaceApi.class);
```


Criando uma instância da API

- Para criar uma instância da API para a interface de serviços de usuário que criamos anteriormente, podemos realizar a seguinte chamada:

```
ApiUsuario apiUsuario =  
    retrofit.create(ApiUsuario.class) ;
```

Criando uma chamada

- Para criar uma chamada, basta utilizar a instância da API, conforme comandos abaixo:

```
Call<ClasseDeModelo> chamada =  
    api.metodoApi(parametrosApi);
```

Criando uma chamada

- Em nosso cenário da API de usuários, seria necessário solicitar a instância de API de usuários o método “getUsuario” e armazená-lo num Call do tipo “Usuario” (modelo)

```
Call<Usuario> chamadaUsuario =  
    apiUsuario.getUsuario(idUsuario);
```

Criando um callback

- Para criar um callback, é necessário definir uma classe que implemente “Callback” e criar uma instância deste elemento
- Podemos utilizar classes anônimas para tanto (semelhante ao que é feito com listeners)

Criando um callback

```
Callback<ClasseDeModelo> Callback =  
    new Callback<ClasseDeModelo>() {  
  
        @Override  
        public void onResponse(Call<ClasseDeModelo> call,  
            Response<ClasseDeModelo> response) {  
  
        }  
  
        @Override  
        public void onFailure(Call<ClasseDeModelo> call,  
            Throwable t) {  
  
        }  
  
    };
```

Criando um callback

- No cenário do serviço de usuários, teríamos um callback semelhante ao elemento a seguir

Criando um callback de usuário

```
Callback<Usuario> usuarioCallback =  
    new Callback<Usuario>() {  
        @Override  
        public void onResponse(Call<Usuario> call,  
            Response<Usuario> response) {  
            //Tratar o retorno do serviço de usuário  
            //Por exemplo, exibir uma mensagem  
        }  
  
        @Override  
        public void onFailure(Call<Usuario> call,  
            Throwable t) {  
            //Tratar erros do serviço de usuário  
            //Por exemplo, exibir uma mensagem  
        }  
    };
```

Executando a chamada com callback

- Por fim, basta solicitar a execução da chamada, fornecendo o callback a ser utilizado
- As chamadas são assíncronas e enfileiradas, portanto, é necessário utilizar o método “enqueue” (enfileirar)

```
chamada.enqueue(callback);
```


Chamada com callback em usuários

- Em nosso cenário de usuários, seria necessário efetuar a chamada a call com o callback adequado, conforme comando abaixo:

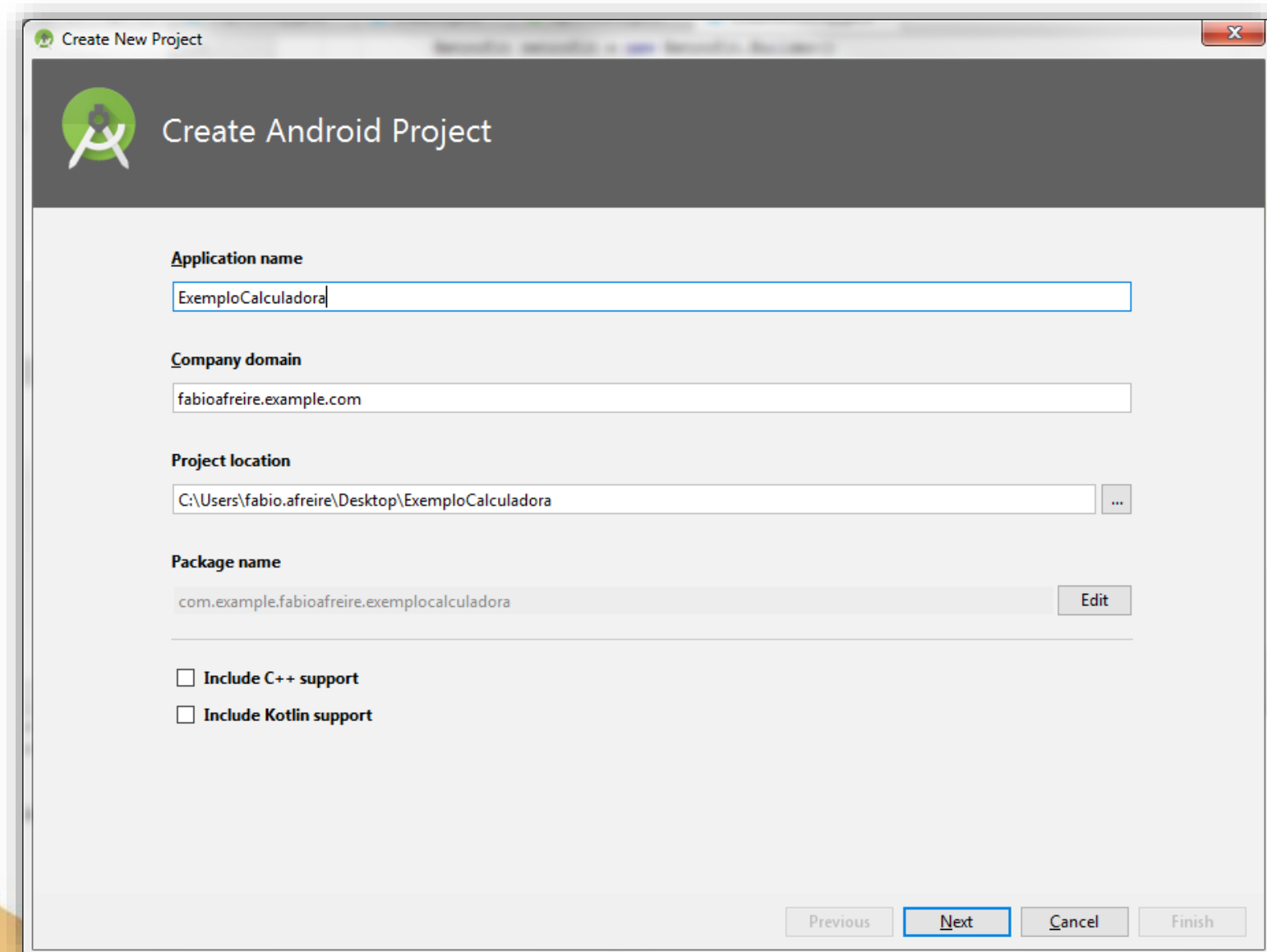
```
chamadaUsuario.enqueue(usuarioCallback);
```

EXEMPLO PRÁTICO


Exemplo da calculadora

- Vamos utilizar o serviço que vimos na aula passada (calculadora), mas desta vez com a ajuda do retrofit
- Para tanto, crie um novo projeto chamado “CalculadoraRetrofit” seguindo os padrões de criação que temos utilizado
- Defina o pacote padrão da aplicação como “com.example.fabio.retrofittest” (substitua “fabio” pelo seu nome)

Criando o projeto



Create New Project

 **Create Android Project**

Application name
ExemploCalculadora

Company domain
fabioafreire.example.com


Project location
C:\Users\fabio.afreire\Desktop\ExemploCalculadora ...


Package name
com.example.fabioafreire.exemplocalculadora Edit

☐ Include C++ support
☐ Include Kotlin support

Previous **Next** Cancel Finish

Criando o projeto

 Create New Project

 Target Android Devices

Select the form factors and minimum SDK

Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

☒ **Phone and Tablet**

API 23: Android 6.0 (Marshmallow)

By targeting **API 23 and later**, your app will run on approximately 39,3% of devices. [Help me choose](#)

☐ Include Android Instant App support

☐ **Wear**

API 21: Android 5.0 (Lollipop)

☐ **TV**

API 21: Android 5.0 (Lollipop)

☐ **Android Auto**

☐ **Android Things**

API 24: Android 7.0 (Nougat)

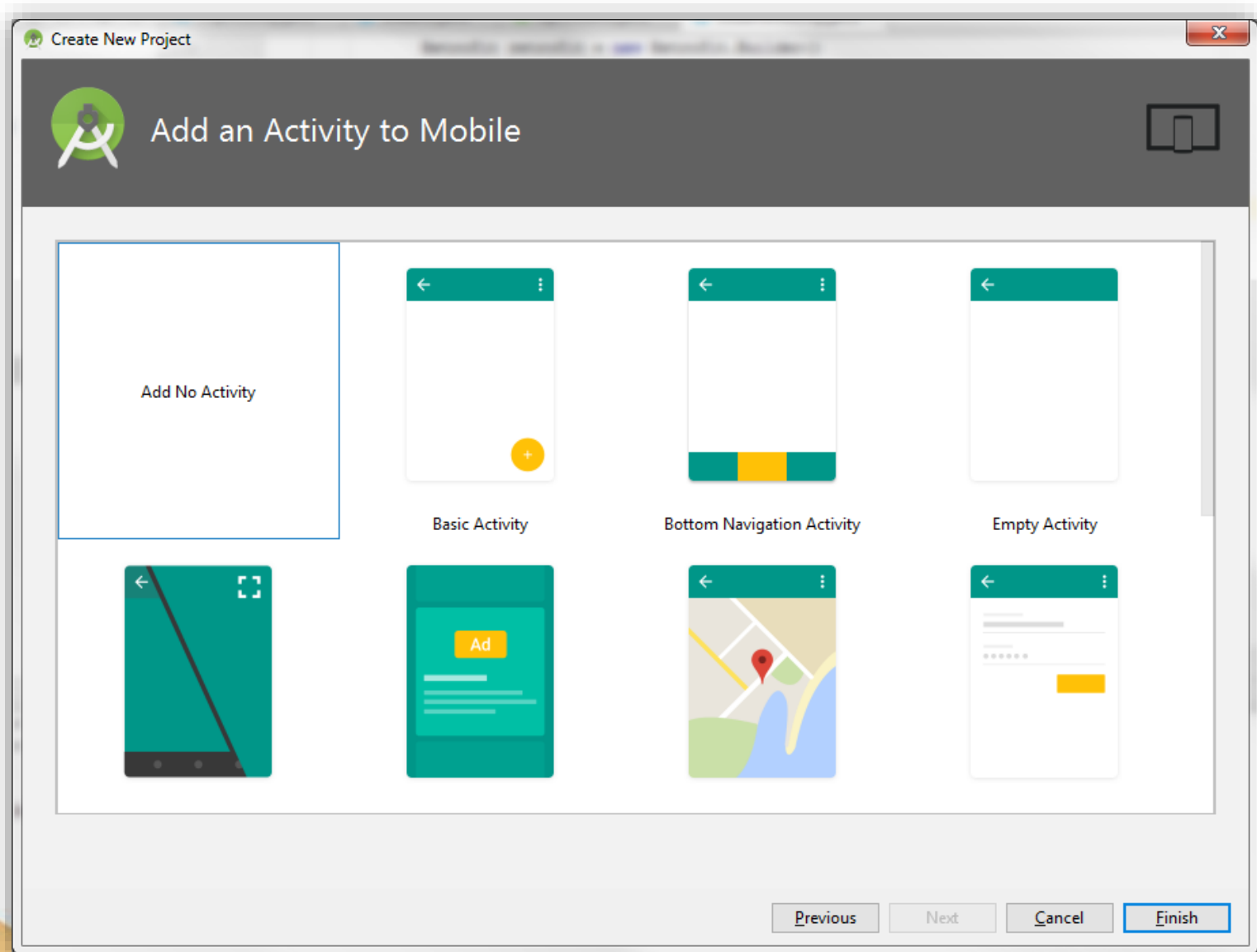
Previous

Next

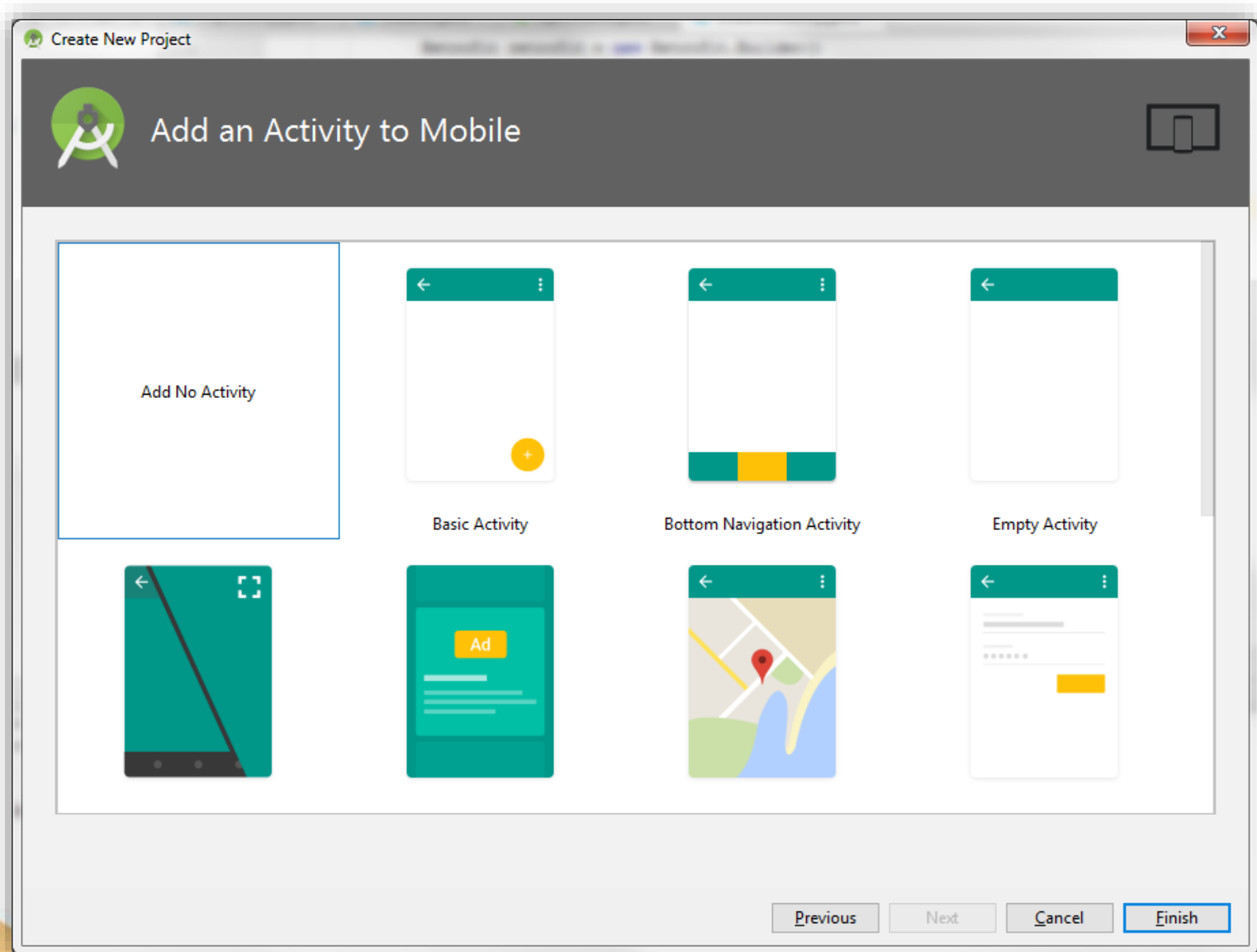
Cancel

Finish

Criando o projeto



Criando o projeto



Item de modelo

- O serviço que utilizamos

(<http://fabiohenriqueaf.esy.es/getObject.php>)

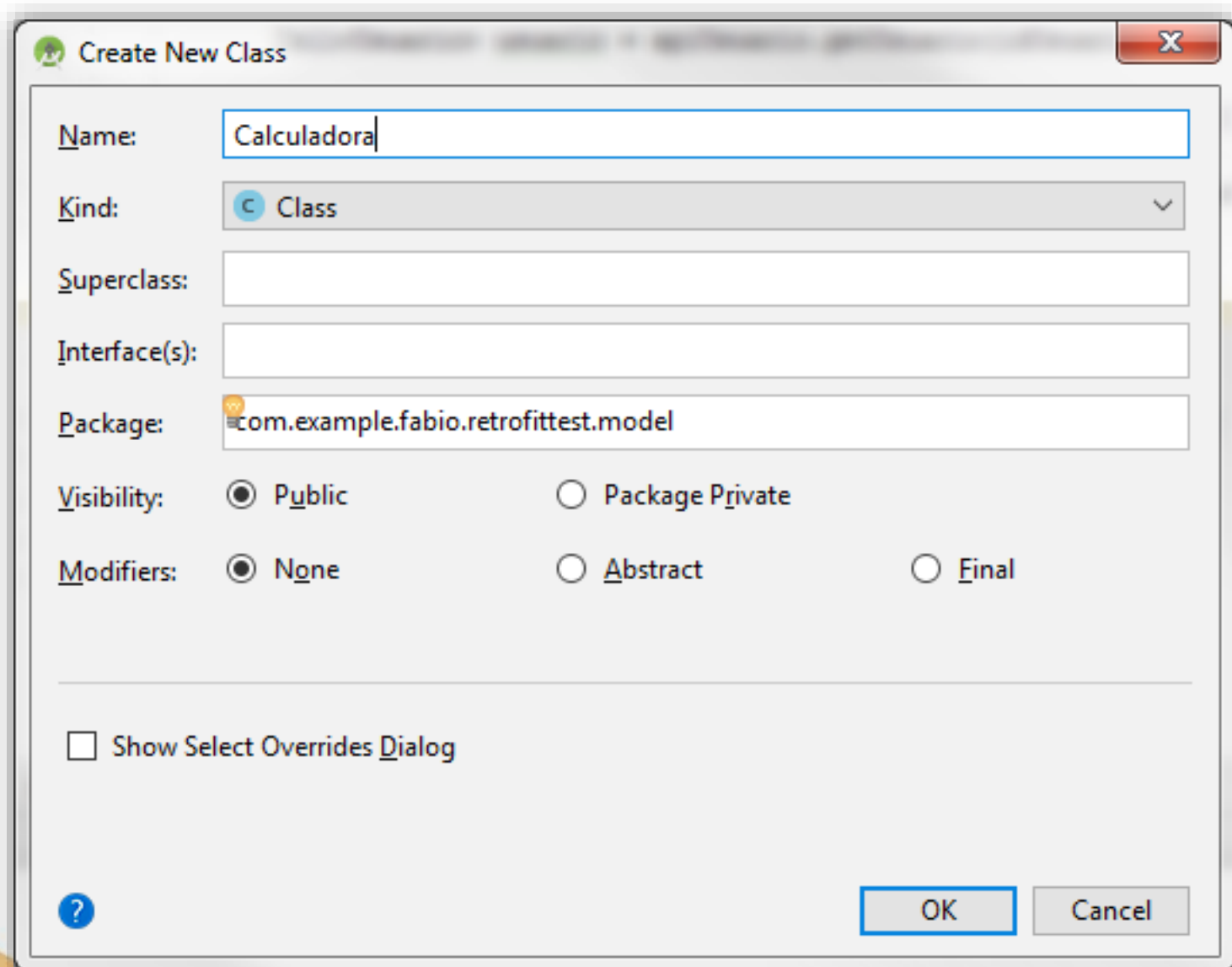
devolve dados no seguinte formato:

```
{  
    "soma": 3,  
    "sub": -1,  
    "mult": 2,  
    "div": 0.5  
}
```


Classe de modelo

- Portanto, precisamos criar um elemento Java semelhante a esta estrutura
- Crie uma nova Classe Java (botão direito no projeto, “New” > “Java Class”) chamada “Calculadora” no pacote “com.example.fabio.retrofittest.model”

Classe de modelo



Create New Class

Name:

Kind: Class

Superclass:

Interface(s):

Package:

Visibility: ☒ Public ☐ Package Private

Modifiers: ☒ None ☐ Abstract ☐ Final

☐ Show Select Overrides Dialog

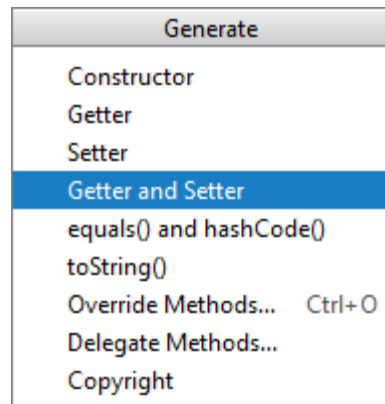
Classe de modelo

- Defina a classe com os parâmetros semelhantes ao retorno da API a ser utilizada
- Os parâmetros da classe devem ser privados e a mesma deve ter métodos de acesso de leitura e escrita para todos os atributos

Classe de modelo

```
public class Calculadora {  
    private Double soma;  
    private Double sub;  
    private Double mult;  
    private Double div;
```

ALT + INSERT

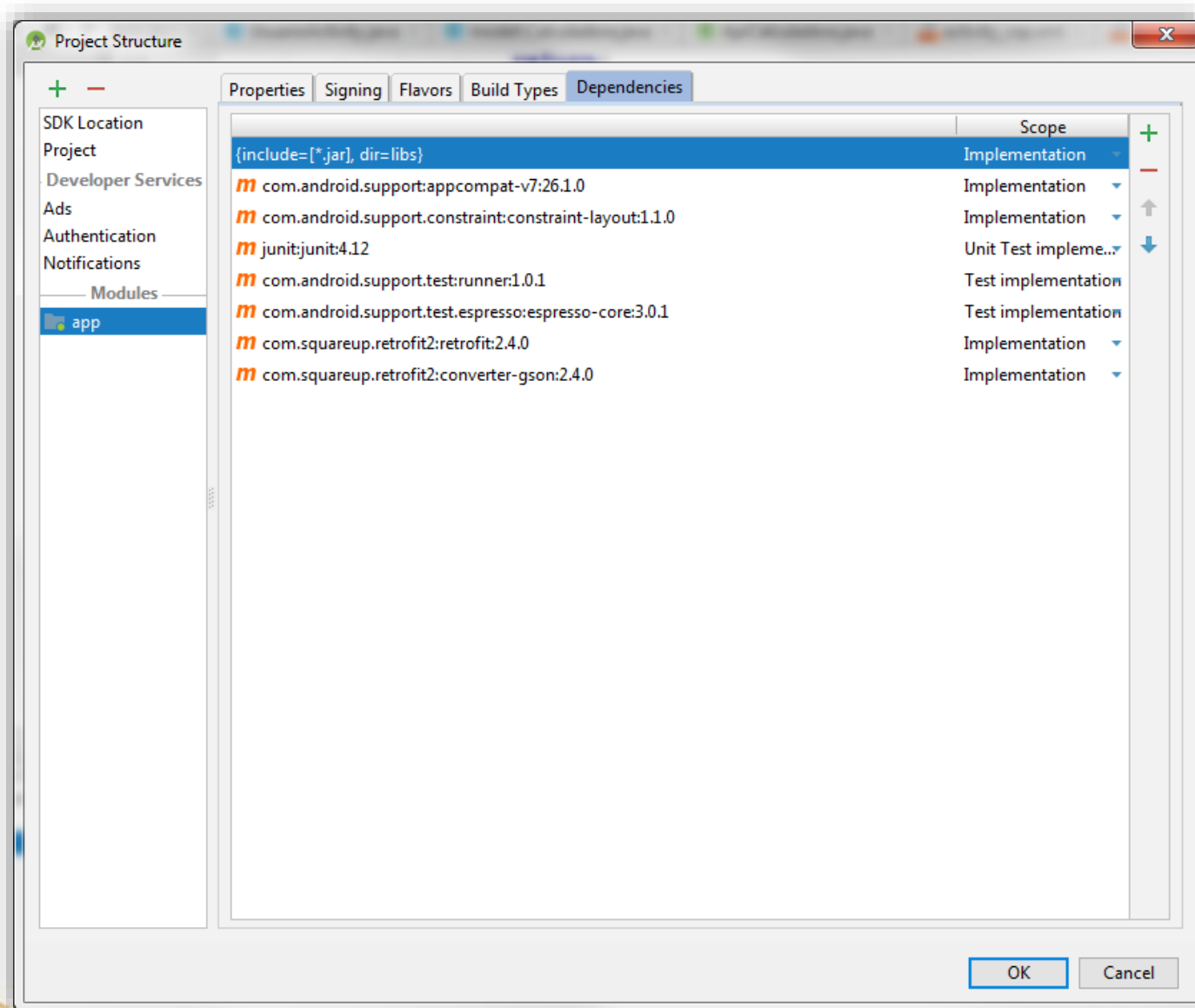


```
public class Calculadora {  
    private Double soma;  
    private Double sub;  
    private Double mult;  
    private Double div;  
  
    public Double getSoma() {  
        return soma;  
    }  
  
    public void setSoma(Double soma) {  
        this.soma = soma;  
    }  
  
    public Double getSub() {  
        return sub;  
    }  
  
    public void setSub(Double sub) {  
        this.sub = sub;  
    }  
  
    public Double getMult() {  
        return mult;  
    }  
  
    public void setMult(Double mult) {  
        this.mult = mult;  
    }  
  
    public Double getDiv() {  
        return div;  
    }  
  
    public void setDiv(Double div) {  
        this.div = div;  
    }  
}
```

Inclusão do retrofit

- Para construção da API e utilização do retrofit, é necessário adicionar o framework como uma dependência de sua aplicação Android
- Para isso, clique com o botão direito no projeto e escolha a opção “Open Module Settings”
- Em seguida, clique na aba “Dependencies”

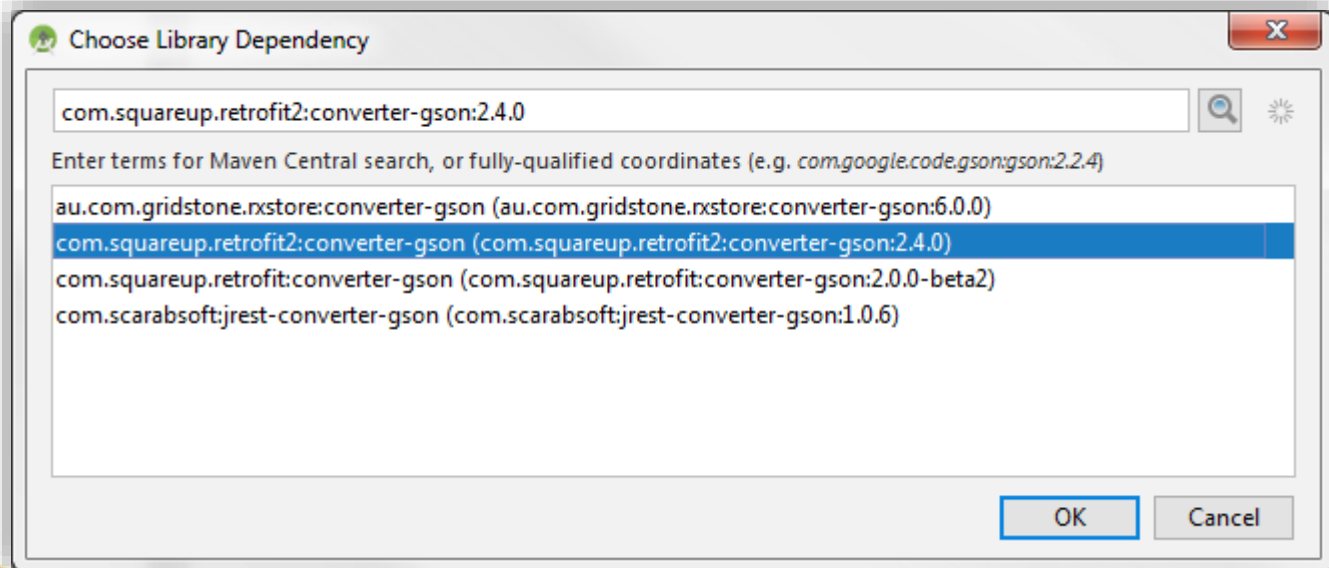
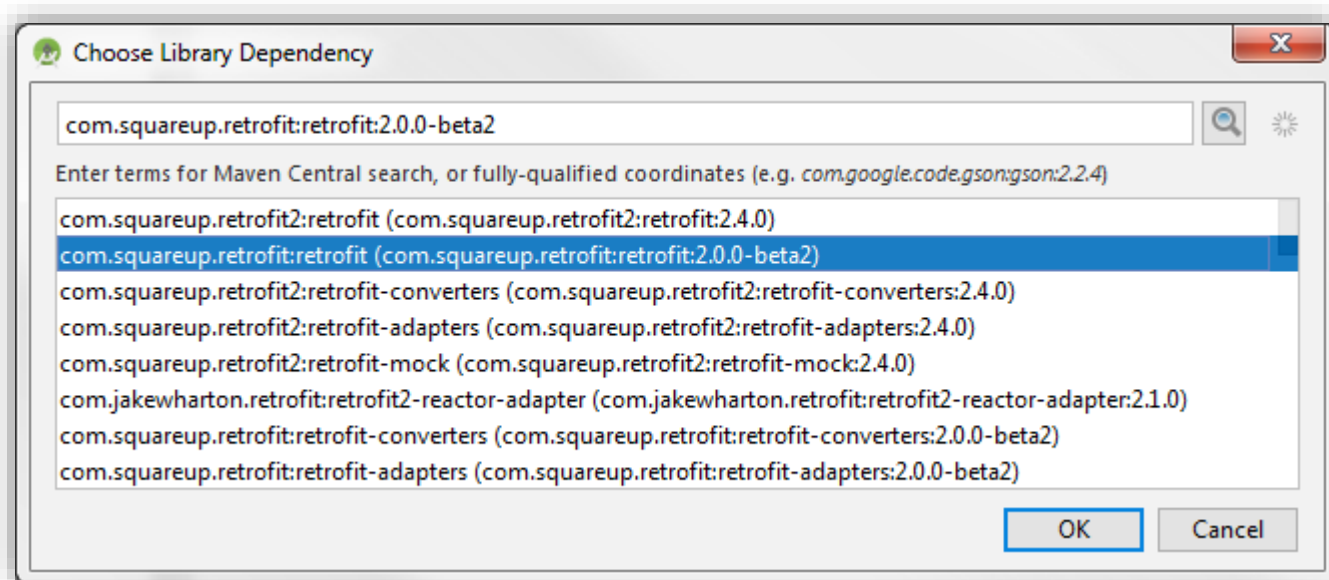
Inclusão do retrofit



Inclusão do retrofit

- Na janela de dependências, clique no ícone de “+” e selecione “Library dependency”
- Procure por “retrofit” e escolha a dependência “com.squareup.retrofit2:retrofit:2.4.0”
- Faça o mesmo com “converter-gson” (escolha “com.squareup.retrofit2:converter-gson:2.4.0”)

Inclusão do retrofit



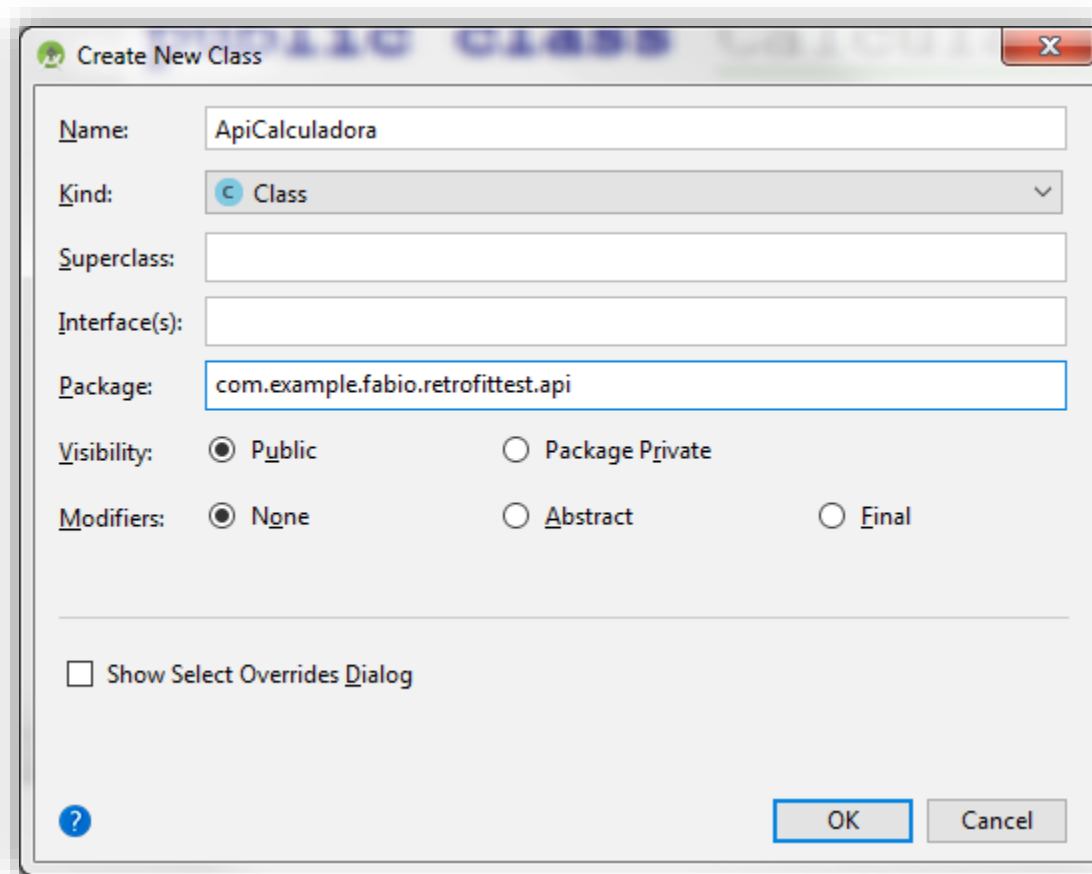
Definindo a interface de API

- Agora, precisamos criar uma interface de API que especifique a chamada desejada
- Vamos utilizar o método de API “getObject”, portanto, precisamos especificá-lo
- Este método aceita dois parâmetros (num1 e num2) e devolve o objeto “Calculadora”

Definindo a interface de API

- Crie uma nova Interface Java (botão direito no projeto > “New” > “Java Class” e selecione “Interface” no combo “Kind”)
- Chame-a “ApiCalculadora” e crie-a no pacote “com.example.fabio.retrofittest.api”

Definindo a interface de API



Create New Class

Name:

Kind: ☒ Class

Superclass:

Interface(s):

Package:

Visibility: ☒ Public ☐ Package Private

Modifiers: ☒ None ☐ Abstract ☐ Final

☐ Show Select Overrides Dialog

OK Cancel

Interface de API

- Defina a interface de acordo com a especificação da API, conforme exemplo a seguir

Interface de API

```
public interface ApiCalculadora {  
    @GET("/getObject.php")  
    Call<Calculadora> getObject(  
        @Query("num1") Double num1,  
        @Query("num2") Double num2);  
}
```

Atividade de calculadora

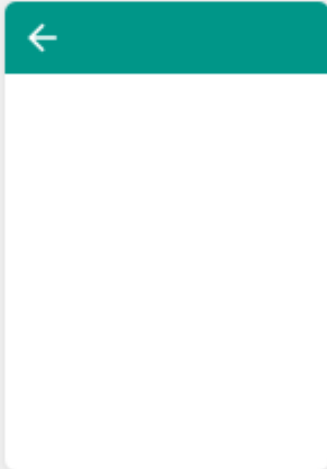
- Já temos os elementos necessários para utilização do retrofit
- Agora, vamos criar a atividade que será responsável por colher os dados do usuário, efetuar a chamada ao serviço retrofit e tratar o resultado

Atividade de calculadora

- Crie uma nova atividade (botão direito no projeto > “New” > “Activity” > “Empty Activity”) e chame-a “CalculadoraActivity”. Crie a atividade no pacote “com.example.fabio.retrofittest.ui”

Atividade de calculadora

Creates a new empty activity



Activity Name:

☒ Generate Layout File

Layout Name:

☐ Launcher Activity

☒ Backwards Compatibility (AppCompat)

Package name: ▾

Source Language: ▾

Target Source Set: ▾

Atividade de calculadora

- O layout da atividade deve ser exatamente como o do exemplo anterior
- Você pode copiar o layout do exemplo ou montá-lo como a seguir

Atividade de calculadora

Valor 01

Valor 02

☐ Soma

☐ Subtração

☐ Multiplicação

☐ Divisão

CALCULAR

Resultado

Ab **textVal1** (EditText) (Plain Text)

Ab **textVal2** (EditText) (Plain Text)

▼ ☒ **radioOp** (RadioGroup) (horizontal)

☒ **radioSoma** (RadioButton) - "Soma"

☒ **radioSub** (RadioButton) - "Subtração"

☒ **radioMult** (RadioButton) - "Multiplicação"

☒ **radioDiv** (RadioButton) - "Divisão"

☒ **buttonCalc** - "Calcular"

Ab **textResult** (EditText) (Plain Text)

Declaração de elementos e binding

- Na classe da atividade, realize a declaração e o binding dos elementos, também da mesma forma que o exemplo anterior

Declaração de elementos e binding

```
public class Calculadora extends AppCompatActivity {  
    EditText textVal1;  
    EditText textVal2;  
    RadioButton radioSoma;  
    RadioButton radioSub;  
    RadioButton radioMult;  
    RadioButton radioDiv;  
    TextView textResult;  
    Button buttonCalc;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_calculadora);  
  
        textVal1 = (EditText) findViewById(R.id.textVal1);  
        textVal2 = (EditText) findViewById(R.id.textVal2);  
        radioSoma = (RadioButton) findViewById(R.id.radioSoma);  
        radioSub = (RadioButton) findViewById(R.id.radioSub);  
        radioMult = (RadioButton) findViewById(R.id.radioMult);  
        radioDiv = (RadioButton) findViewById(R.id.radioDiv);  
        textResult = (TextView) findViewById(R.id.textResult);  
        buttonCalc = (Button) findViewById(R.id.buttonCalc);  
    }  
}
```

Método de diálogo

- Declare o método de exibição do diálogo conforme exemplo

Método de diálogo

```
private void showDialog(String val, String title) {  
    AlertDialog.Builder builder = new  
        AlertDialog.Builder(Calculadora.this);  
    builder.setMessage(val);  
    builder.setTitle(title);  
    builder.setCancelable(false);  
    builder.setPositiveButton("OK", null);  
    AlertDialog dialog = builder.create();  
    dialog.show();  
}
```

Declaração de listener

- Declare o listener do botão e associe-o ao componente.

Declaração de listener

```
View.OnClickListener listener = new View.OnClickListener()
{
    @Override
    public void onClick(View v) {

    }
};

buttonCalc.setOnClickListener(listener);
```


Definição do listener

- No listener, declare, converta e obtenha os parâmetros numéricos da tela, conforme exemplo anterior

Definição do listener

```
@Override
public void onClick(View v) {
    Double val1 = null;
    Double val2 = null;

    try {
        val1 = Double.parseDouble(textVal1.getText().toString());
    }
    catch(Exception e) {
        e.printStackTrace();
        showDialog("Valor 1 inválido", "Erro");
        return;
    }

    try {
        val2 = Double.parseDouble(textVal2.getText().toString());
    }
    catch(Exception e) {
        e.printStackTrace();
        showDialog("Valor 2 inválido", "Erro");
        return;
    }
}
```

Validação dos radios

- Por fim, valide também os valores dos rádios (logo abaixo do código anterior)

Definição do listener

```
if (!radioSoma.isChecked() && !radioSub.isChecked()  
    && !radioMult.isChecked() && !radioDiv.isChecked()) {  
    showDialog("Selecione uma operação", "Erro");  
    return;  
}
```

Obtenção de instância do retrofit

- Após os elementos de obtenção de valores, declarações e validações, vamos, finalmente, chamar o serviço através do retrofit
- Para isso, logo após a validação dos radios, declare e obtenha uma instância do retrofit para o endereço da API da calculadora

Obtenção de instância do retrofit

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("http://fabiohenriqueaf.esy.es")  
    .addConverterFactory(GsonConverterFactory.create())  
    .build();
```

Obtenção de instância da API

- Em seguida, solicite a criação de uma instância para a API, conforme o comando a seguir

```
ApiCalculadora apiCalculadora =  
    retrofit.create(ApiCalculadora.class);
```

Obtenção da instância da chamada

- Com a API, será possível solicitar a criação de uma chamada ao serviço
- Utilize a o método da API desejado (getObject) para criar a chamada, passando os parâmetros adequados, conforme exemplo a seguir

```
Call<Calculadora> callCalculadora =  
    apiCalculadora.getObject(val1, val2);
```


Criação do callback

- Antes de efetuar a chamada ao serviço, é necessário definir seu callback
- O callback deverá tratar a resposta do servidor. Caso tenha sucesso, o método “onResponse” será executado. Caso aconteçam erros, o método “onFailure” será executado

Criação do callback

- Declare a classe de callback logo abaixo do código de criação da chamada que criamos anteriormente, conforme exemplo a seguir

Criação do callback

```
Callback<Calculadora> callbackCalculadora =  
    new Callback<Calculadora>() {  
  
        @Override  
        public void onResponse(Call<Calculadora> call,  
            Response<Calculadora> response) {  
  
        }  
  
        @Override  
        public void onFailure(Call<Calculadora> call,  
            Throwable t) {  
  
        }  
  
    };
```

Tratamento do callback de sucesso

- Quando o callback for executado com sucesso, devera obter o valor desejado do modelo e alterar a interface de acordo
- Acrescente o conteúdo do método “onResponse” conforme exemplo a seguir

Tratamento do callback de sucesso

```
@Override
public void onResponse(Call<Calculadora> call,
    Response<Calculadora> response) {

    Calculadora calc = response.body();

    if (response.isSuccessful() && calc != null) {

        if (radioSoma.isChecked()) {
            textResult.setText(String.valueOf(calc.getSoma()));
        }
        else if (radioSub.isChecked()) {
            textResult.setText(String.valueOf(calc.getSub()));
        }
        else if (radioMult.isChecked()) {
            textResult.setText(String.valueOf(calc.getMult()));
        }
        else if (radioDiv.isChecked()) {
            textResult.setText(String.valueOf(calc.getDiv()));
        }
    }
}
```

Tratamento do callback de erro

- Quando o callback for executado com erro, vamos exibir uma mensagem de erro ao usuário
- Acrescente o conteúdo do método “onFailure” conforme exemplo a seguir

Tratamento do callback de erro

```
@Override  
public void onFailure(Call<Calculadora> call, Throwable t) {  
    t.printStackTrace();  
    showDialog( val: "Falha ao obter o resultado", title: "erro");  
}
```

Tratamento do callback de erro

- Por fim, vamos executar a chamada, “enfileirando-a” junto ao callback

```
callCalculadora.enqueue(callbackCalculadora);
```


Adicionando a permissão de internet

- Ainda é necessário adicionar a permissão de internet no Android Manifest (fora de “application”)

```
<uses-permission android:name="android.permission.INTERNET" />
```

Resultado da execução da aplicação

RetrofitTest

1

1

☐ Soma

☐ Subtração

☐ Multiplicação

☒ Divisão

CALCULAR

1.0

EXERCÍCIOS

Exercício 01

- Construa a aplicação de CEP com AsyncTask E utilizando retrofit (dois exercícios diferentes):
 - *<http://viacep.com.br/ws/XXXXXX-XXX/json/>*
 - Onde **XXXXXX-XXX** é o CEP desejado

Exercício 02

- Consoma um serviço criado na disciplina “Desenvolvimento de Componentes e Serviços” com retrofit, criando uma interface gráfica que possibilite acionar o serviço corretamente.
- Entregue o projeto Android junto ao projeto da outra disciplina que contém o serviço utilizado.

"That's all Folks!"