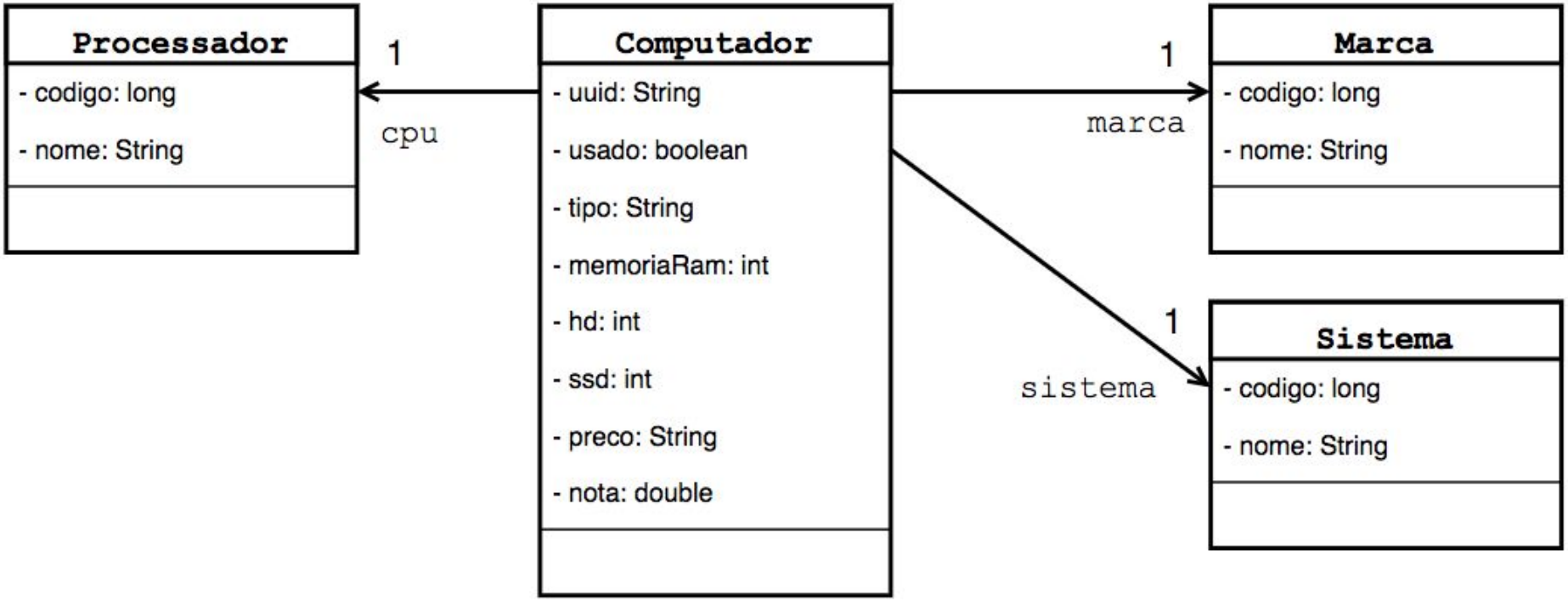
Desenvolvimento Mobile

Aula 11

Criando banco de dados SQLite

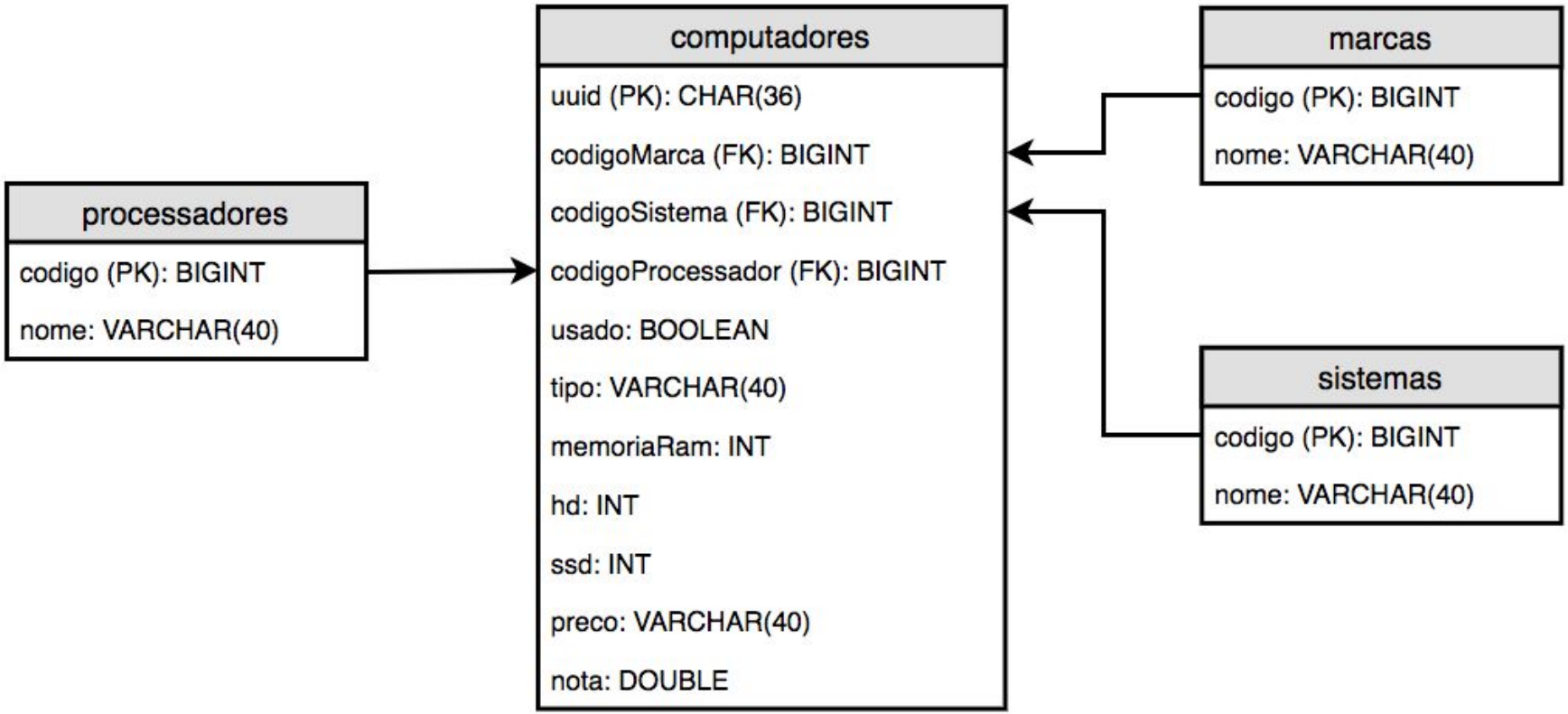
App salvando dados no *SQLite*

Modelo de Domínio (Diagrama de Classes)



Vamos converter este **Diagrama de Classes** em um **Diagrama de Estrutura de Dados!**

Diagrama de Estrutura de Dados



https://www.sqlite.org/datatype3.html

Diagrama de Estrutura de Dados

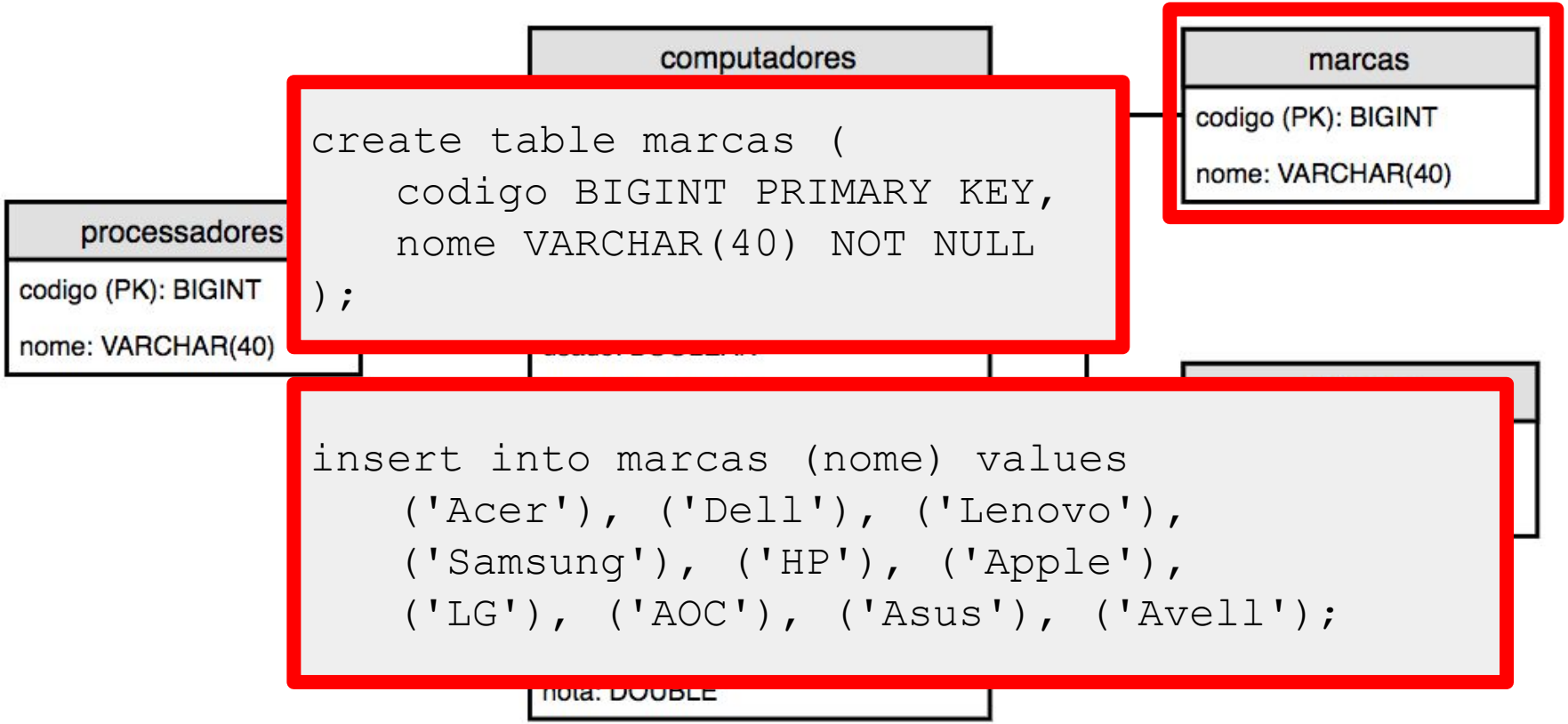


Diagrama de Estrutura de Dados

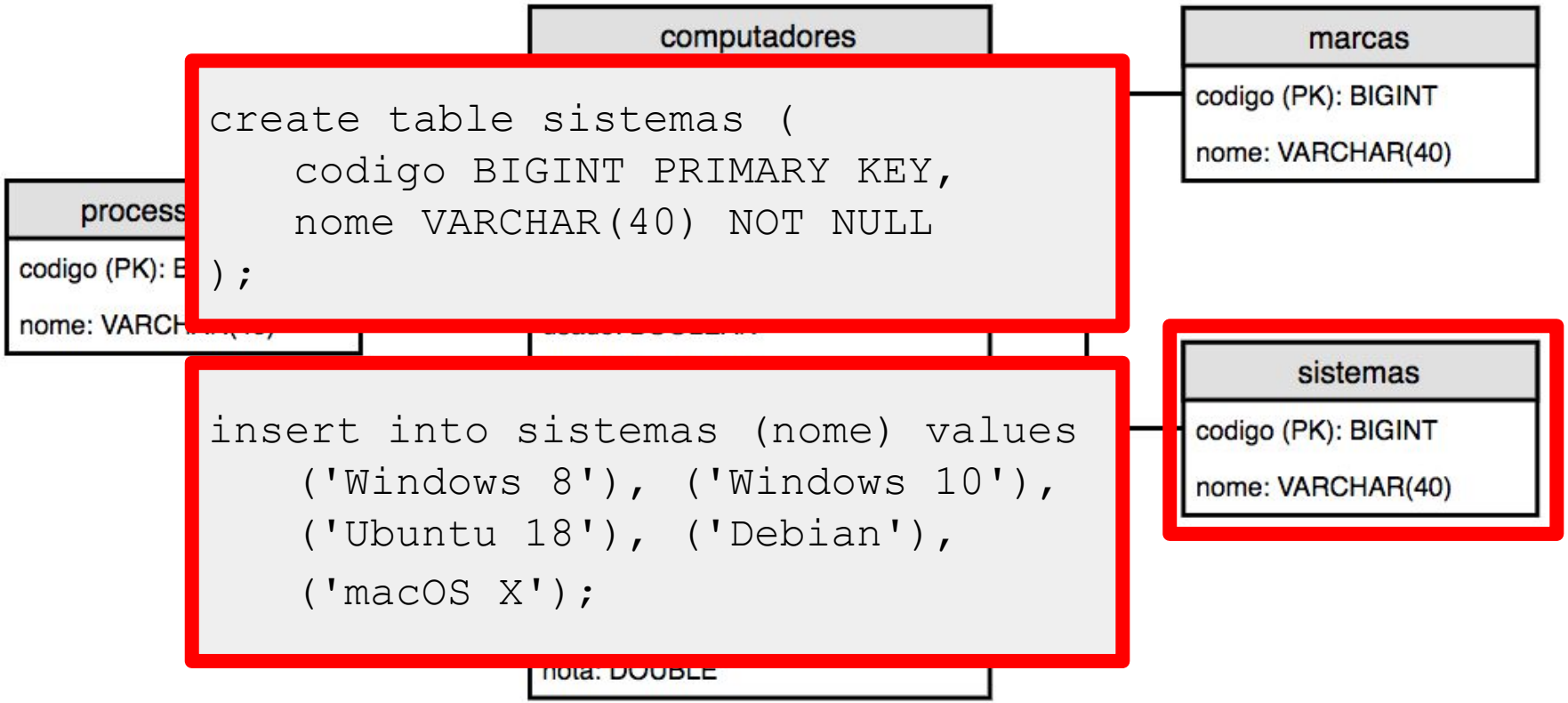


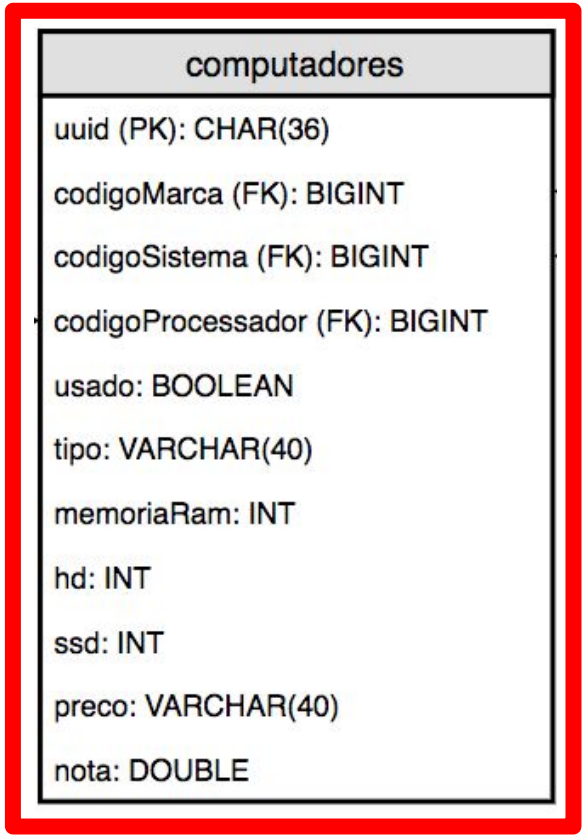
Diagrama de Estrutura de Dados

computadores

marcas

```
create table processadores (
    codigo BIGINT PRIMARY KEY,
    nome VARCHAR(40) NOT NULL
);
```

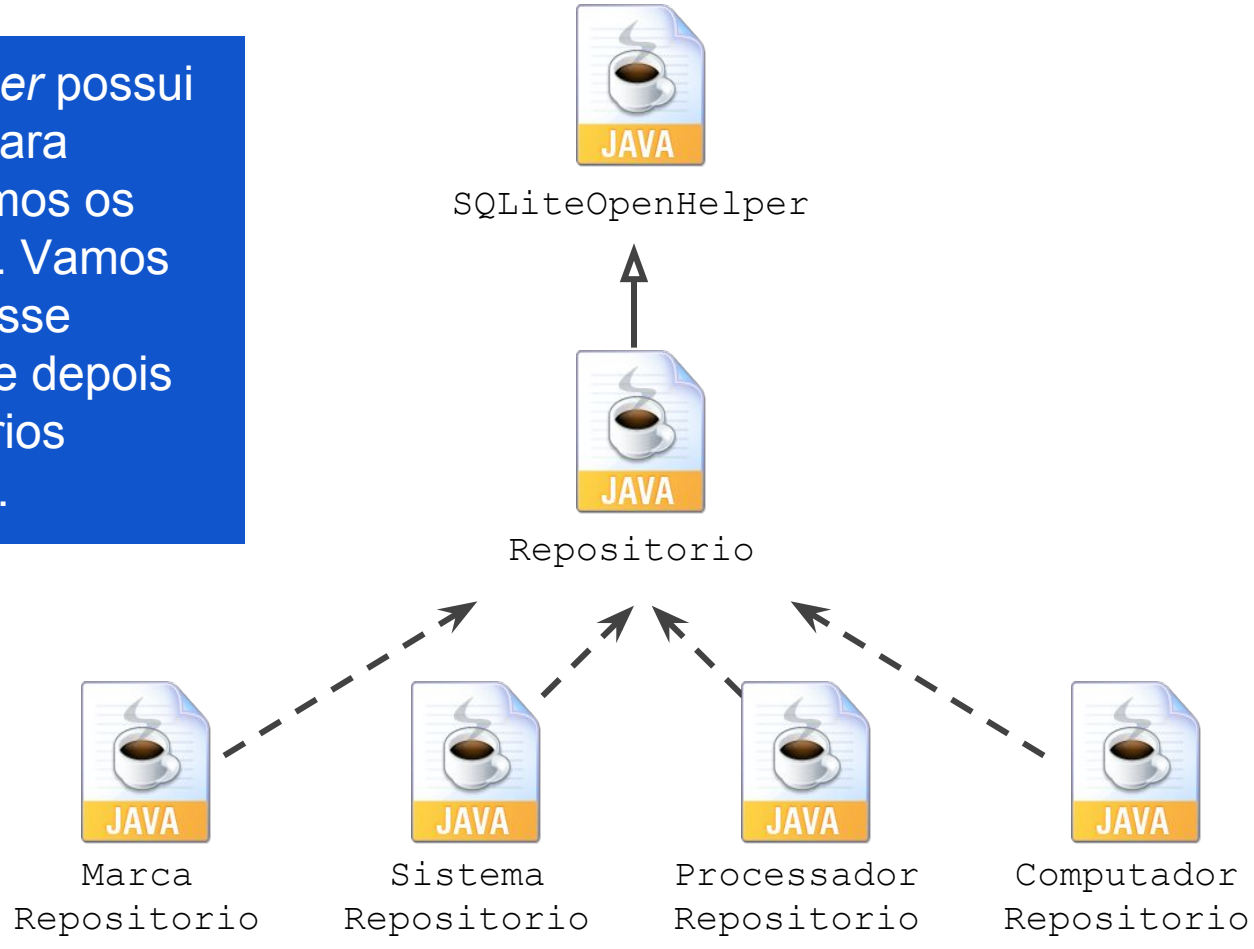
```
insert into processadores (nome) values
    ('Intel i3 2GHz'),
    ('Intel i5 2,5GHz'),
    ('Intel i5 3GHz'),
    ('Intel i7 3GHz'),
    ('Intel i9 3,5GHz');
```




```
create table computadores (
    uuid CHAR(36) PRIMARY KEY,
    codigoMarca BIGINT NOT NULL,
    codigoSistema BIGINT NOT NULL,
    codigoProcessador BIGINT NOT NULL,
    usado BOOLEAN NOT NULL,
    tipo VARCHAR(40) NOT NULL,
    memoriaRam INT NOT NULL,
    hd INT NOT NULL,
    ssd INT NOT NULL,
    preco VARCHAR(40) NOT NULL,
    nota DOUBLE NOT NULL,
    FOREIGN KEY (codigoMarca)
        REFERENCES marcas(codigo),
    FOREIGN KEY (codigoSistema)
        REFERENCES sistemas(codigo),
    FOREIGN KEY (codigoProcessador)
        REFERENCES processadores(codigo)
);
```

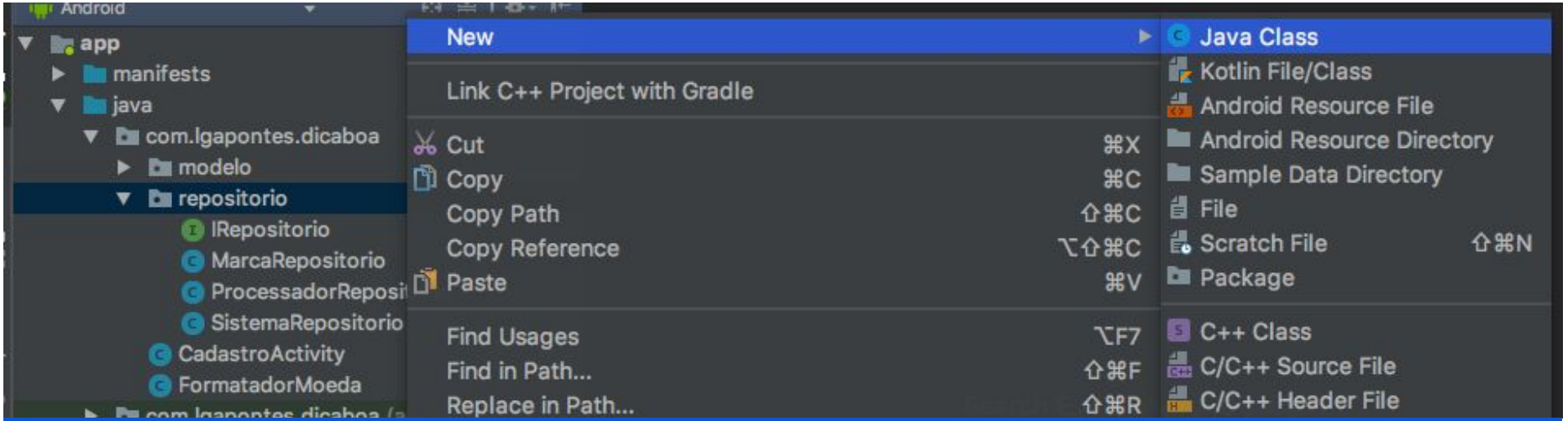
App salvando dados no *SQLite*

A classe *SQLiteOpenHelper* possui os recursos necessários para conectarmos e manipularmos os dados do banco de dados. Vamos herdá-la em uma nova classe chamada *Repositorio* - que depois invocaremos nos repositórios particulares de nossa App.



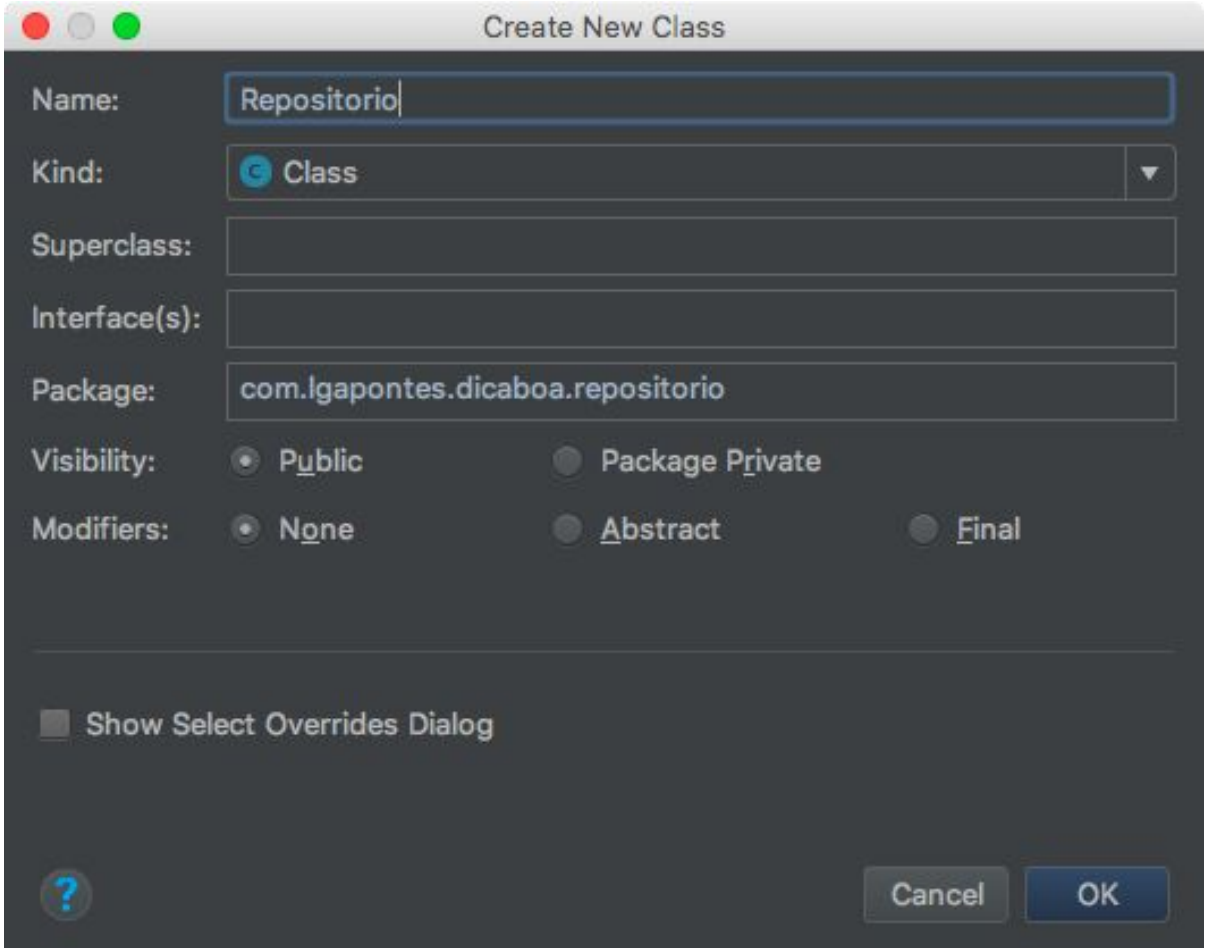
App salvando dados no *SQLite (DicaBoa_v7)*

Repositorio



Crie uma nova classe chamada *Repositorio* dentro do pacote *repositorio*.

App salvando dados no *SQLite (DicaBoa_v7)*



Repositorio

App salvando dados no *SQLite (DicaBoa_v7)*

Repositorio

PASSO 1: herde SQLiteOpenHelper, conforme o código abaixo:

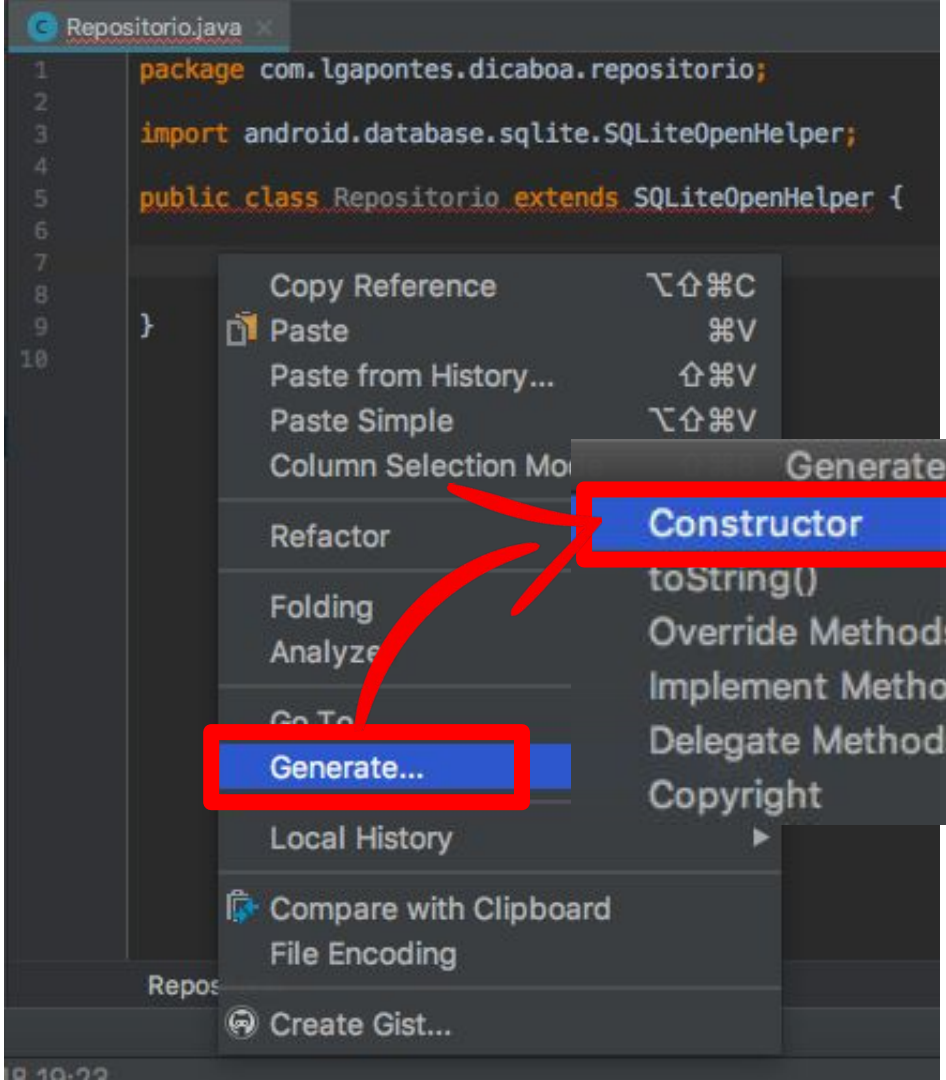
Repositorio.java

```
1   package com.lgapontes.dicaboa.repositorio;
2
3   import android.database.sqlite.SQLiteOpenHelper;
4
5   public class Repositorio extends SQLiteOpenHelper {
6
7
8
9   }
10
```

App salvando dados no *SQLite (DicaBoa_v7)*

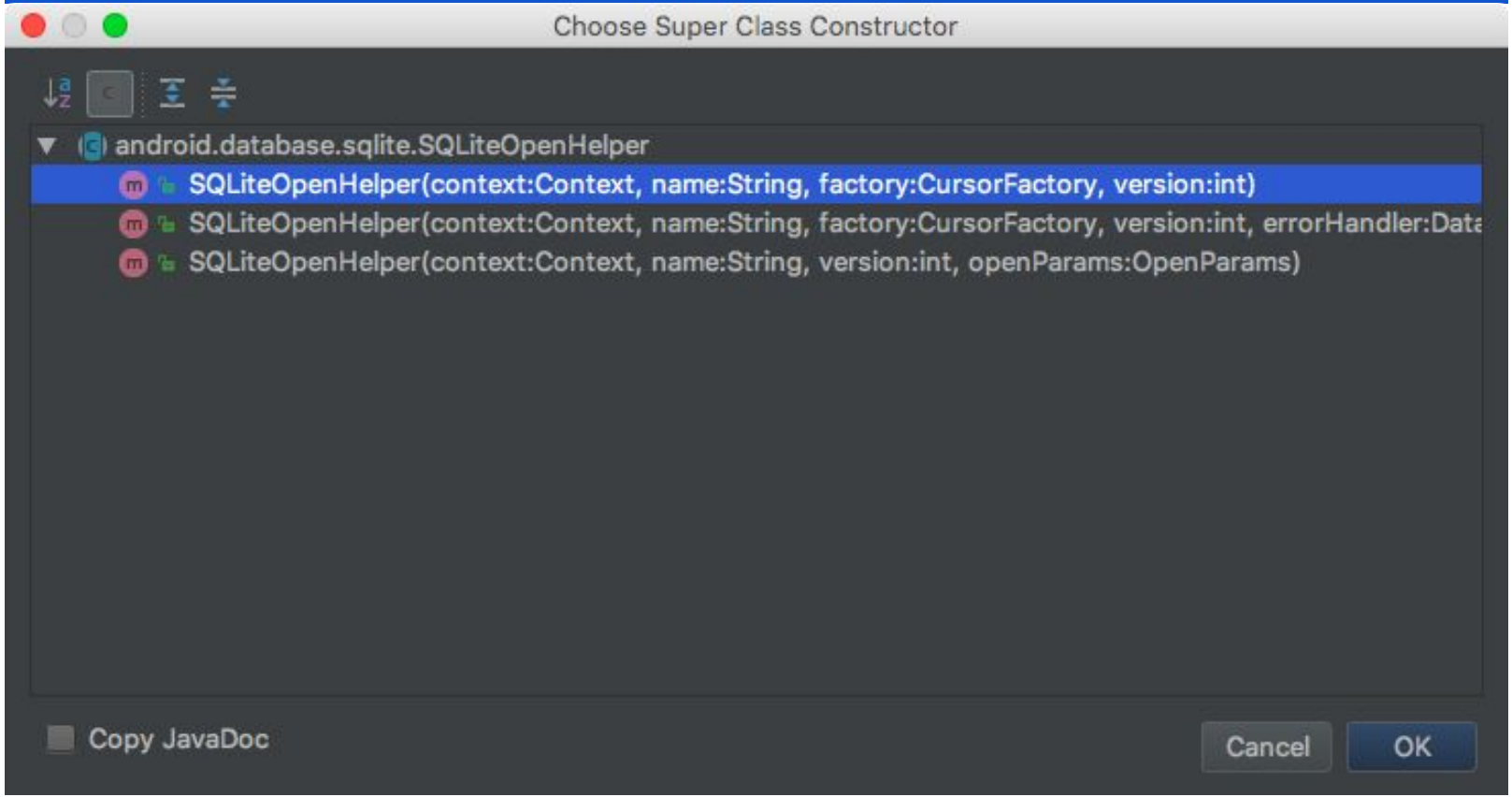
PASSO 2: após herdar SQLiteOpenHelper, seremos obrigados a implementar um *constructor* específico e os métodos *onCreate()* e *onUpgrade()*.

Vamos começar pelo constructor. Clique com o botão direito dentro da classe, opção *"Generate..."*, submenu *"Constructor"*



App salvando dados no *SQLite* *(DicaBoa_v7)*

PASSO 3: selecione a opção abaixo e clique em OK



App salvando dados no *SQLite (DicaBoa_v7)*

```
public class Repositorio extends SQLiteOpenHelper {

    public Repositorio(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, nam  factory, versio
    }
}
```

O context do App.

O SQLite organiza o banco de dados em um arquivo único. Este parâmetro é o nome deste arquivo. Este banco é restrito à esta aplicação (não pode ser acessado por outras aplicações).

Eu posso conectar neste banco?
Pode! Veja a apresentação *Slide da Aula 11 (conectar SQLite)* disponível no EAD.

Utilizado para customizar um cursor de execução das consultas. Se for nulo, será o cursor default.

Versão do banco de dados. Esse campo é utilizado para definirmos novas versões do banco em casos de evolução da App. Por enquanto vamos utilizar o número 1.

App salvando dados no *SQLite (DicaBoa_v7)*

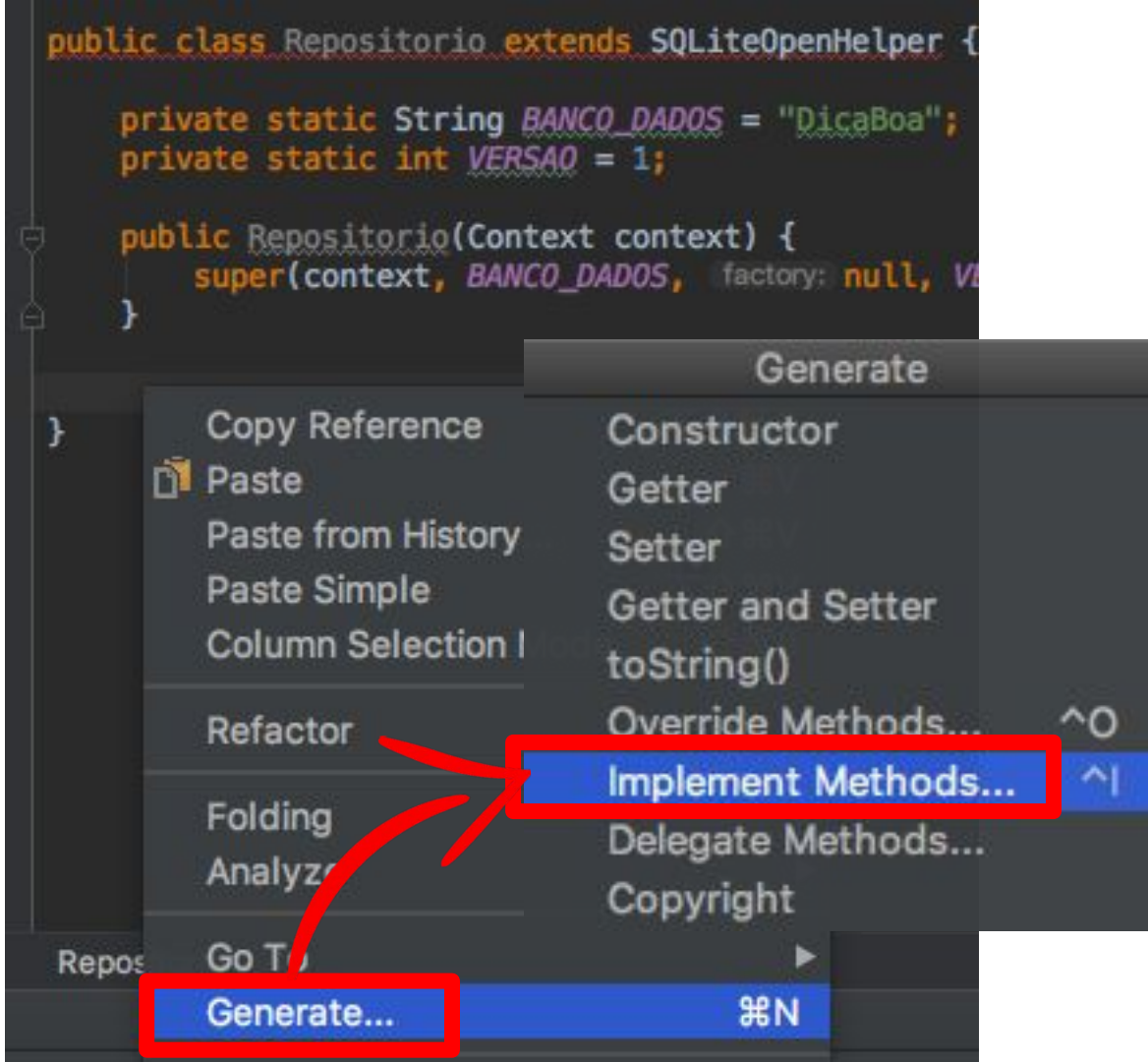
PASSO 4: já que alguns parâmetros podem ser encapsulados dentro da própria classe *Repositorio*, vamos organizar o código do constructor conforme o exemplo abaixo.

```
public class Repositorio extends SQLiteOpenHelper {

    public Repositorio(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
}
```

```
public class Repositorio extends SQLiteOpenHelper {

    private static String BANCO_DADOS = "DicaBoa.db";
    private static int VERSAO = 1;

    public Repositorio(Context context) {
        super(context, BANCO_DADOS,  factory: null, VERSAO);
    }

}
```

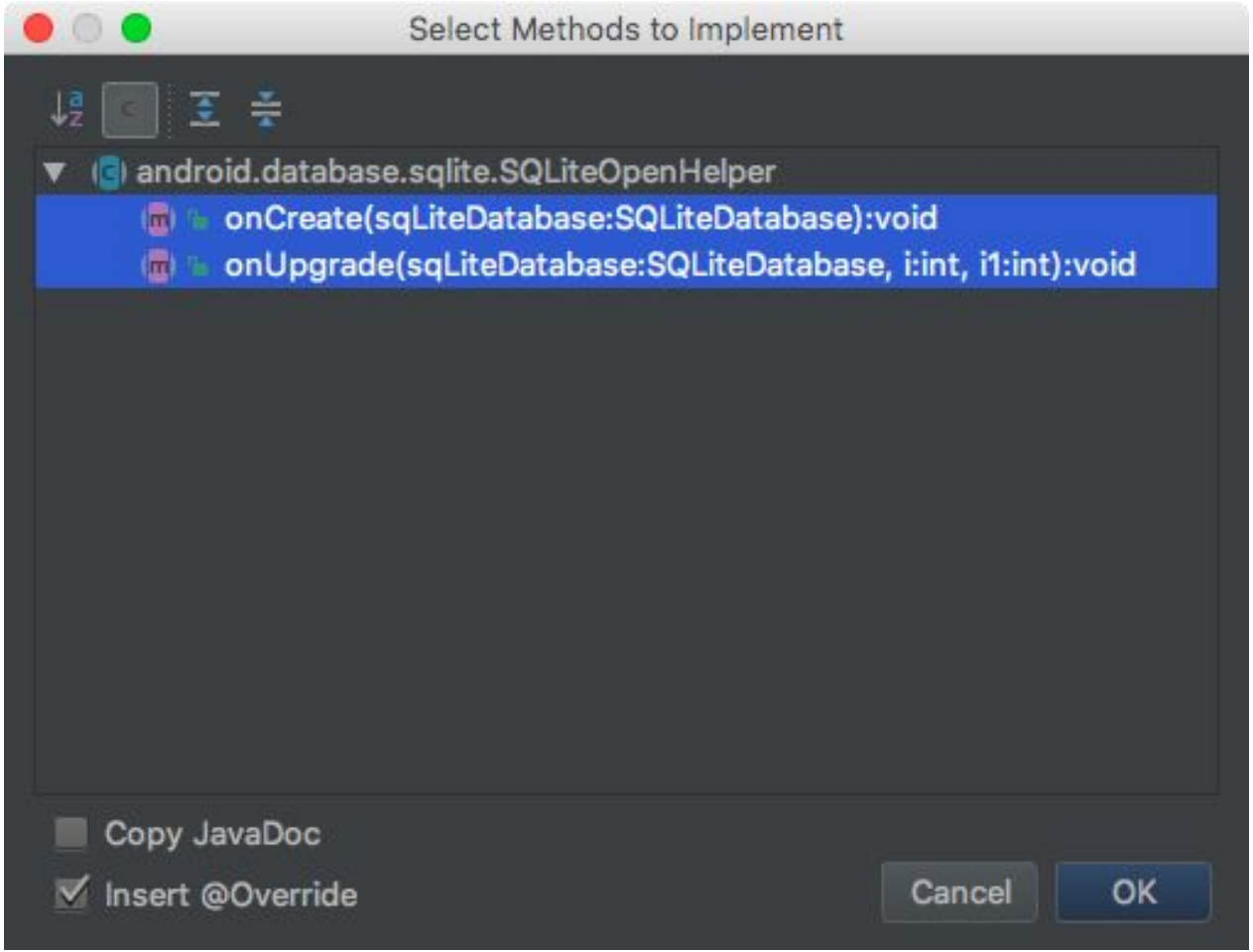
App salvando dados no *SQLite* *(DicaBoa_v7)*

PASSO 5: vamos agora implementar os métodos *onCreate()* e *onUpgrade()*. Clique com o botão direito abaixo do constructor, selecione a opção "*Generate…*", menu "*Implement Methods...*"



App salvando dados no *SQLite* (*DicaBoa_v7*)

PASSO 6: selecione ambos métodos e clique em OK.



App salvando dados no *SQLite (DicaBoa_v7)*

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("create table marcas (codigo BIGINT PRIMARY KEY, nome VARCHAR(
        db.execSQL("create table sistemas (codigo BIGINT PRIMARY KEY,
        db.execSQL("create table processadores (codigo BIGINT PRIMARY
        db.execSQL("create table computadores (uuid CHAR(36) PRIMARY
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Continua...

PASSO 7: vamos focar no método *onCreate()*. Pegue as queries DDL (Data Definition Language) apresentadas nos slides anteriores e execute-as através do método *execSQL()* disponível no objeto *SQLiteDatabase*. Como esse código pode gerar erro, coloque-o dentro de um *try/catch*.

Copie as queries formatadas em uma única linha em: https://bit.ly/2OZH3eK

App salvando dados no *SQLite* (*DicaBoa_v7*)

Além da criação das tabelas, devemos carregar os dados das *marcas, sistemas* e *processadores* nas suas respectivas tabelas. O *SQLiteDatabase* realiza isso através do método *insert()*.

O método *insert()* recebe três parâmetros:
`table`: nome da tabela em que será executado o *insert*.
`nullColumnHack`: gambiarra oficial que permite inserir valores nulos no banco. Pouco usado na prática (mais detalhes, vide https://bit.ly/2NkirNp).
`values`: valores que serão inseridos, organizados em um objeto *ContentValues*.

App salvando dados no *SQLite (DicaBoa_v7)*

PASSO 8: Como as tabelas *marcas, sistemas* e *processadores* são iguais, podemos criar um método único para inserir valores de acordo com um array de strings passado por parâmetro.

```
private void popularTabela(SQLiteDatabase db, String tabela, String[] itens) {
    for (int i=0;i<itens.length;i++) {
        ContentValues content = new ContentValues();
        int codigo = i+1;
        content.put("codigo",codigo);
        content.put("nome",itens[i]);
        db.insert(tabela, nullColumnHack: null,content);
    }
}
```

Este método poderá ser invocado a partir do *onCreate()* para podermos popular as tabelas supracitadas.

PASSO 9

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {
        db.execSQL("create table marcas (codigo BIGINT PRIMARY KEY, nome VARCHAR(40)
        db.execSQL("create table sistemas (codigo BIGINT PRIMARY KEY,nom
        db.execSQL("create table processadores (codigo BIGINT PRIMARY KE
        db.execSQL("create table computadores (uuid CHAR(36) PRIMARY KEY,codigoMarca
        popularTabela(db,  tabela: "marcas", new String[] {
                "Acer", "Dell", "Lenovo", "Samsung", "HP",
                "Apple", "LG", "AOC", "Asus", "Avell"
        });
        popularTabela(db,  tabela: "sistemas", new String[] {
                "Windows 8", "Windows 10", "Ubuntu 18",
                "Debian", "macOS X"
        });
        popularTabela(db,  tabela: "processadores", new String[] {
                "Intel i3 2GHz", "Intel i5 2,5GHz", "Intel i5 3GHz",
                "Intel i7 3GHz", "Intel i9 3,5GHz"
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Continua...

Copie os arrays de strings criados na *Aula 10* (nas classes de repositório) e coloque-os nas respectivas invocações dos métodos *popularTabela()*

PASSO 9

```
@Override
public void onCreate(SQLite
    try {
        db.execS
        db.exe
        db.ex
        db.e
        popu
        });
        popularTabela(
                "Windows 8", "Windows 10", "
                "Debian", "macOS X"
        });
        popularTabela(db,  tabela: "processadores", new
                "Intel i3 2GHz", "Intel i5 2,5GHz",
                "Intel i7 3GHz", "Intel i9 3,5GHz"
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Quando exatamente o método **onCreate()** é executado?

Continua...

Copie os arrays da *Aula 10* (nas classes de repos nas respectivas invocações dos m abela()

App salvando dados no *SQLite* (*DicaBoa_v7*)

https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper

`SQLiteOpenHelper`

onCreate(**SQLiteDatabase db**)

Called when the database is created for the first time.

A documentação não é muito clara sobre o momento exato em que os comandos de criação do banco disponíveis em *onCreate()* são executados. Na prática, a classe *SQLiteOpenHelper* assume o seguinte comportamento:

- *onCreate()* é executado somente se não houver um arquivo do SQLite na pasta interna da aplicação (`/data/data/<nome do pacote>`)
- *onCreate()* é executado somente quando houver ações de manipulação dos dados (*insert, delete, update*) ou pela invocação dos métodos *getReadableDatabase()* ou *getWritableDatabase()*.

App salvando dados no *SQLite (DicaBoa_v7)*

PASSO 10: vamos agora criar métodos para consultar os dados das tabelas.

PASSO 10.1: Obter o banco através do método *getReadableDatabase()*

```
SQLiteDatabase db = getReadableDatabase();
```

PASSO 10.2: Rodar a SQL pelo método *rawQuery()* e guardar o resultado num *Cursor*

```
Cursor cursor = db.rawQuery( sql: "select * from marcas",  selectionArgs: null);
```

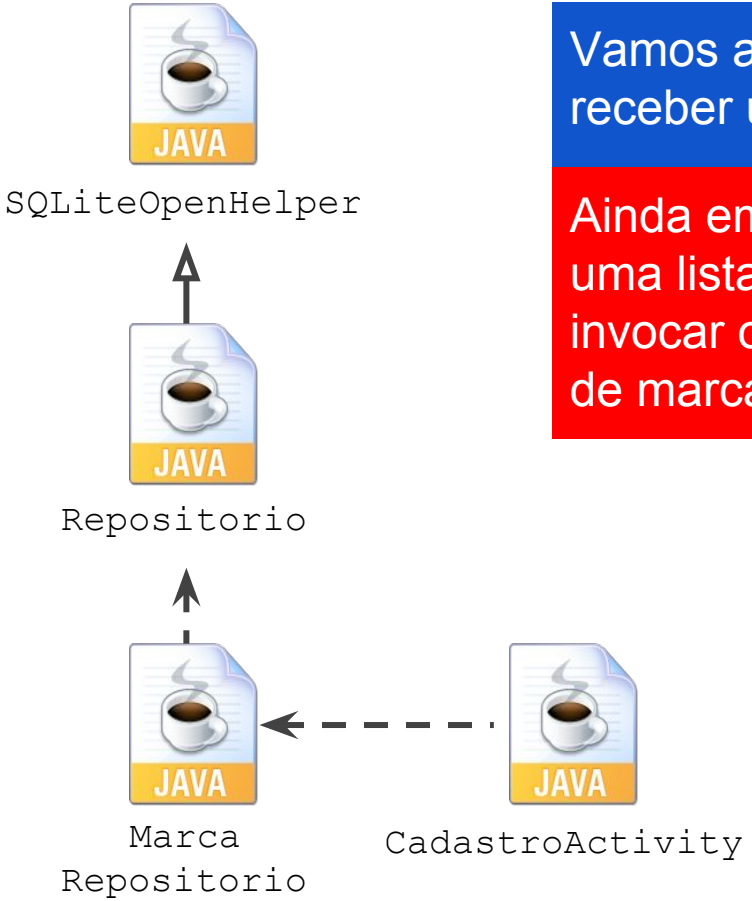
PASSO 10.3:
Iterar sobre o cursor e guardar os dados de cada valor em uma lista (que neste exemplo, é uma lista de *marcas*).

```
List<Marca> lista = new ArrayList<Marca>();
while(cursor.moveToNext()) {
    Marca marca = new Marca(
            cursor.getLong(cursor.getColumnIndex( s: "codigo")),
            cursor.getString(cursor.getColumnIndex( s: "nome"))
    );
    lista.add(marca);
}
```

PASSO 10: o código completo do método *listarMarcas()* (que deve ser implementado na classe *Repositorio*) deve conter um bloco *try/catch* para tratar erros.

```
public List<Marca> listarMarcas() {
    List<Marca> lista = new ArrayList<Marca>();
    try {
        SQLiteDatabase db = getReadableDatabase();
        Cursor cursor = db.rawQuery( sql: "select * from marcas",  selectionArgs: null);
        while(cursor.moveToNext()) {
            Marca marca = new Marca(
                    cursor.getLong(cursor.getColumnIndex( s: "codigo")),
                    cursor.getString(cursor.getColumnIndex( s: "nome"))
            );
            lista.add(marca);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return lista;
}
```

App salvando dados no *SQLite* (*DicaBoa_v7*)



Vamos agora ajustar a classe *MarcaRepositorio* para receber uma instância do *Repositorio* no *constructor*.

Ainda em *MarcaRepositorio*, ao invés de codificarmos uma lista *hard-coded* no método *listar()*, vamos invocar o método *listarMarcas()* (que obtém os dados de marcas do SQLite).

Por fim, vamos ajustar *CadastroActivity* para instanciar um novo *Repositorio* e repassá-lo na construção de *MarcaRepositorio*.

PASSO 11: implemente um constructor em *MarcaRepositorio* que receba um objeto *Repositorio* e guarde-o em um atributo de mesmo nome. Altere também o código do método *listar()* para obter a listagem das marcas a partir do banco de dados.

MarcaRepositorio.java

```
1   package com.lgapontes.dicaboa.repositorio;
2
3   import ...
7
8   public class MarcaRepositorio implements IRepositorio<Marca> {
9
10      private Repositorio repositorio;
11
12      public MarcaRepositorio(Repositorio repositorio) {
13          this.repositorio = repositorio;
14      }
15
16      @Override
17      public List<Marca> listar() {
18          return this.repositorio.listarMarcas();
19      }
20  }
21
```

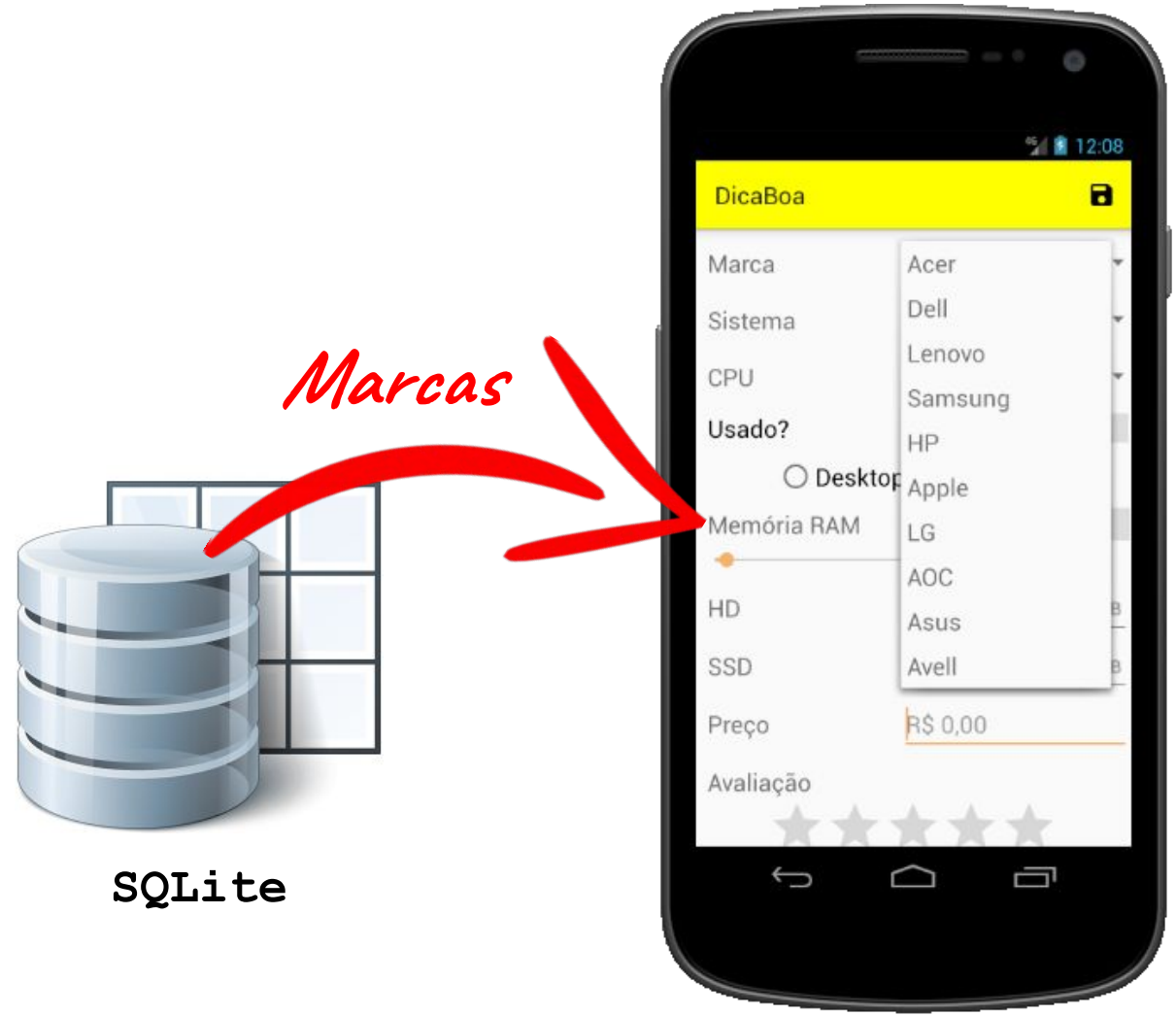
App salvando dados no *SQLite (DicaBoa_v7)*

PASSO 12: em *CadastroActivity*, crie um atributo para uma instância de *Repositorio*

```
private Repositorio repositorio;
```

PASSO 13: no método *onCreate(),* instancie o *Repositorio* passando o *context* e repasse-o na criação de *MarcaRepositorio()*.

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

    this.repositorio = new Repositorio( context: this);
    spinnerMarcas = (Spinner) findViewById(R.id.form_spinner_marcas);
    popularSpinner(spinnerMarcas, new MarcaRepositorio(this.repositorio));

    // Resto do código ...
```



Execute o App. Agora a listagem de marcas é obtida do banco de dados.

```
sqlite> insert into marcas (codigo,nome) values (11,'IBM');
```

Para comprovar que os dados estão vindo do SQLite, podemos realizar um insert diretamente no banco (vide SQL acima) e depois executar novamente a aplicação.

Como podemos acessar o banco? Vide o material *Slide da Aula 11 (conectar SQLite)* disponível no EAD.

Exercício em Sala

Altere a aplicação *DicaBoa* para buscar os dados de *sistemas* e *processadores* do banco de dados SQLite.

Resolvendo o Exercício

```
public List<Sistema> listarSistemas() {
    List<Sistema> lista = new ArrayList<Sistema>();
    try {
        SQLiteDatabase db = getReadableDatabase();
        Cursor cursor = db.rawQuery( sql: "select * from sistemas",  selectionArgs: null);
        while(cursor.moveToNext()) {
            Sistema sistema = new Sistema(
                    cursor.getLong(cursor.getColumnIndex( s: "codigo")),
                    cursor.getString(cursor.getColumnIndex( s: "nome"))
            );
            lista.add(sistema);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return lista;
}
```

Resolvendo o Exercício

```
public List<Processador> listarProcessadores() {
    List<Processador> lista = new ArrayList<Processador>();
    try {
        SQLiteDatabase db = getReadableDatabase();
        Cursor cursor = db.rawQuery( sql: "select * from processadores",  selectionArgs: null);
        while(cursor.moveToNext()) {
            Processador processador = new Processador(
                    cursor.getLong(cursor.getColumnIndex( s: "codigo")),
                    cursor.getString(cursor.getColumnIndex( s: "nome"))
            );
            lista.add(processador);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return lista;
}
```

Resolvendo o Exercício

SistemaRepositorio.java

```
1   package com.lgapontes.dicaboa.repositorio;
2
3   import ...
7
8   public class SistemaRepositorio implements IRepositorio<Sistema> {
9
10      private Repositorio repositorio;
11
12      public SistemaRepositorio(Repositorio repositorio) {
13          this.repositorio = repositorio;
14      }
15
16      @Override
17      public List<Sistema> listar() {
18          return this.repositorio.listarSistemas();
19      }
20  }
21
```

Resolvendo o Exercício

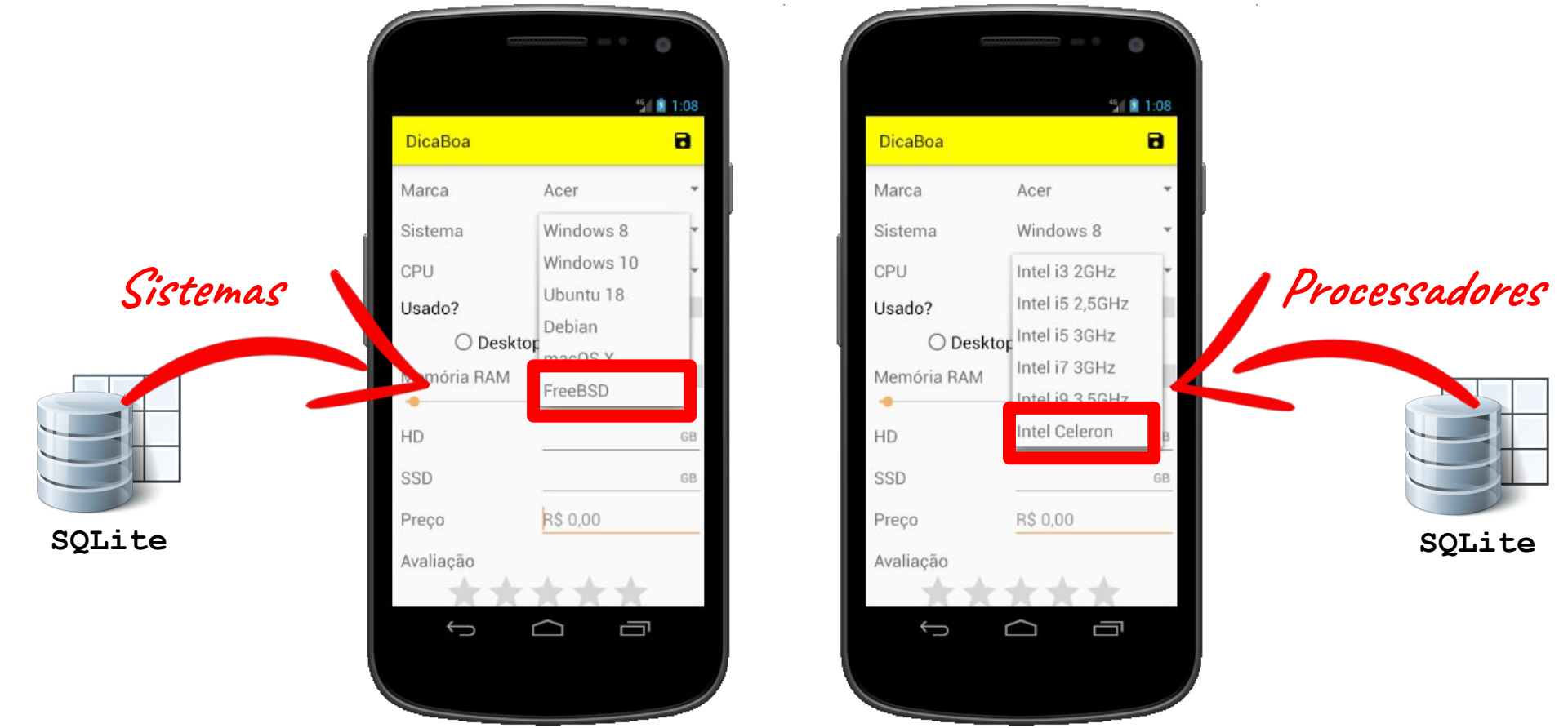
ProcessadorRepositorio.java

```java
package com.lgapontes.dicaboa.repositorio;

import ...

public class ProcessadorRepositorio implements IRepositorio<Processador> {

    private Repositorio repositorio;

    public ProcessadorRepositorio(Repositorio repositorio) {
        this.repositorio = repositorio;
    }

    @Override
    public List<Processador> listar() {
        return this.repositorio.listarProcessadores();
    }
}
```

Resolvendo o Exercício

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

    this.repositorio = new Repositorio( context: this);
    spinnerMarcas = (Spinner) findViewById(R.id.form_spinner_marcas);
    popularSpinner(spinnerMarcas, new MarcaRepositorio(this.repositorio));

    spinnerSistemas = (Spinner) findViewById(R.id.form_spinner_sistemas);
    popularSpinner(spinnerSistemas, new SistemaRepositorio(this.repositorio));

    spinnerProcessadores = (Spinner) findViewById(R.id.form_spinner_processadores);
    popularSpinner(spinnerProcessadores, new ProcessadorRepositorio(this.repositorio));

    // Resto do código
```

```
sqlite> insert into sistemas (codigo,nome) values (6,'FreeBSD');
sqlite> insert into processadores (codigo,nome) values (6,'Intel Celeron');
```



Exercício Extra (Opcional)

Como vocês devem ter percebido, os métodos *listarMarcas()*, *listarSistemas()* e *listarProcessadores()* são quase idênticos. Podemos nos beneficiar dos recursos das *interfaces* Java para evitarmos essa duplicidade.

Continua no próximo slide...

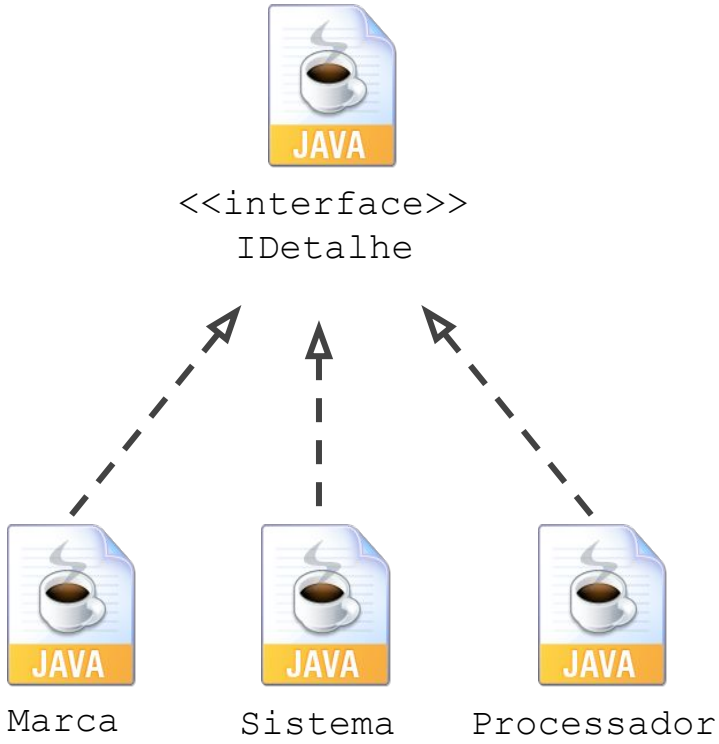
Exercício Extra (Opcional)

Crie uma *interface* chamada *IDetalhe* com os métodos:

```
+  setCodigo(long codigo)
+  setNome(String nome)
```

Depois implemente-a nas classes *Marca*, *Sistema* e *Processador*. Para cada uma dessas classes, crie também um constructor sem parâmetros.

Continua no próximo slide...



Neste caso o constructor sem parâmetros servirá para instanciar um novo objeto através do método de reflexão *newInstance()*.

Exercício Extra (Opcional)

Por fim, crie um método *listarDetalhes()* com a assinatura:

```
public List<IDetalhe> listarDetalhes(Class<? extends IDetalhe> clazz, String tabela)
```

`clazz`: uma classe que obrigatoriamente implementa a interface *IDetalhe* (o *extends* para classificar *generics* serve para herança ou implementação). A partir deste parâmetro podemos utilizar o método de reflexão *newInstance()* para criar uma nova instância e guardá-la em uma *IDetalhe* (que depois será adicionado à lista de retorno).
`tabela`: uma *String* que vai guardar o nome da tabela que será realizada a consulta.

Resolvendo o Exercício

IDetalhe.java

```
1  package com.lgapontes.dicaboa.modelo;
2
3  public interface IDetalhe {
4
5      public void setCodigo(long codigo);
6      public void setNome(String nome);
7
8  }
9
```

Marca.java

```java
package com.lgapontes.dicaboa.modelo;

public class Marca implements IDetalhe {

    private long codigo;
    private String nome;

    public Marca() {}

    public Marca(long codigo, String nome) {
        this.codigo = codigo;
        this.nome = nome;
    }

    public long getCodigo() { return codigo; }

    @Override
    public void setCodigo(long codigo) { this.codigo = codigo; }

    public String getNome() { return nome; }

    @Override
    public void setNome(String nome) { this.nome = nome; }

    @Override
    public String toString() { return nome; }
}
```

Resolvendo o Exercício

Lembre-se de criar um constructor sem parâmetros!

Sistema.java

```java
package com.lgapontes.dicaboa.modelo;

public class Sistema implements IDetalhe {

    private long codigo;
    private String nome;

    public Sistema() {}

    public Sistema(long codigo, String nome) {
        this.codigo = codigo;
        this.nome = nome;
    }

    public long getCodigo() { return codigo; }

    @Override
    public void setCodigo(long codigo) { this.codigo = codigo; }

    public String getNome() { return nome; }

    @Override
    public void setNome(String nome) { this.nome = nome; }

    @Override
    public String toString() { return nome; }
}
```

Resolvendo o Exercício

Lembre-se de criar um constructor sem parâmetros!

Processador.java

```java
package com.lgapontes.dicaboa.modelo;

public class Processador implements IDetalhe {

    private long codigo;
    private String nome;

    public Processador() {}

    public Processador(long codigo, String nome) {
        this.codigo = codigo;
        this.nome = nome;
    }

    public long getCodigo() { return codigo; }

    @Override
    public void setCodigo(long codigo) { this.codigo = codigo; }

    public String getNome() { return nome; }

    @Override
    public void setNome(String nome) { this.nome = nome; }

    @Override
    public String toString() { return nome; }
}
```

Resolvendo o Exercício

Lembre-se de criar um constructor sem parâmetros!

Crie na classe *Repositorio* o método *listarDetalhes()* que deve receber uma classe classificada com qualquer objeto que implemente a interface *IDetalhe* e uma String para o nome da tabela do banco. Através do método *newInstance()* disponível na classe, podemos instanciar os objetos adequados.

Apague os métodos *listarMarcas(), listarSistemas()* e *listarProcessadores()*

```
public List<IDetalhe> listarDetalhes(Class<? extends IDetalhe> clazz, String tabela) {
    List<IDetalhe> lista = new ArrayList<IDetalhe>();
    try {
        SQLiteDatabase db = getReadableDatabase();
        Cursor cursor = db.rawQuery( sql: "select * from " + tabela,  selectionArgs: null);
        while(cursor.moveToNext()) {
            IDetalhe novo = clazz.newInstance();
            novo.setCodigo(cursor.getLong(cursor.getColumnIndex( s: "codigo")));
            novo.setNome(cursor.getString(cursor.getColumnIndex( s: "nome")));
            lista.add(novo);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return lista;
}
```

Resolvendo o Exercício

MarcaRepositorio.java

```
1   package com.lgapontes.dicaboa.repositorio;
2
3   import ...
8
9   public class MarcaRepositorio implements IRepositorio<IDetalhe> {
10
11      private Repositorio repositorio;
12
13      public MarcaRepositorio(Repositorio repositorio) {
14          this.repositorio = repositorio;
15      }
16
17      @Override
18      public List<IDetalhe> listar() {
19          return this.repositorio.listarDetalhes(Marca.class,  tabela: "marcas");
20      }
21  }
22
```

Resolvendo o Exercício

SistemaRepositorio.java

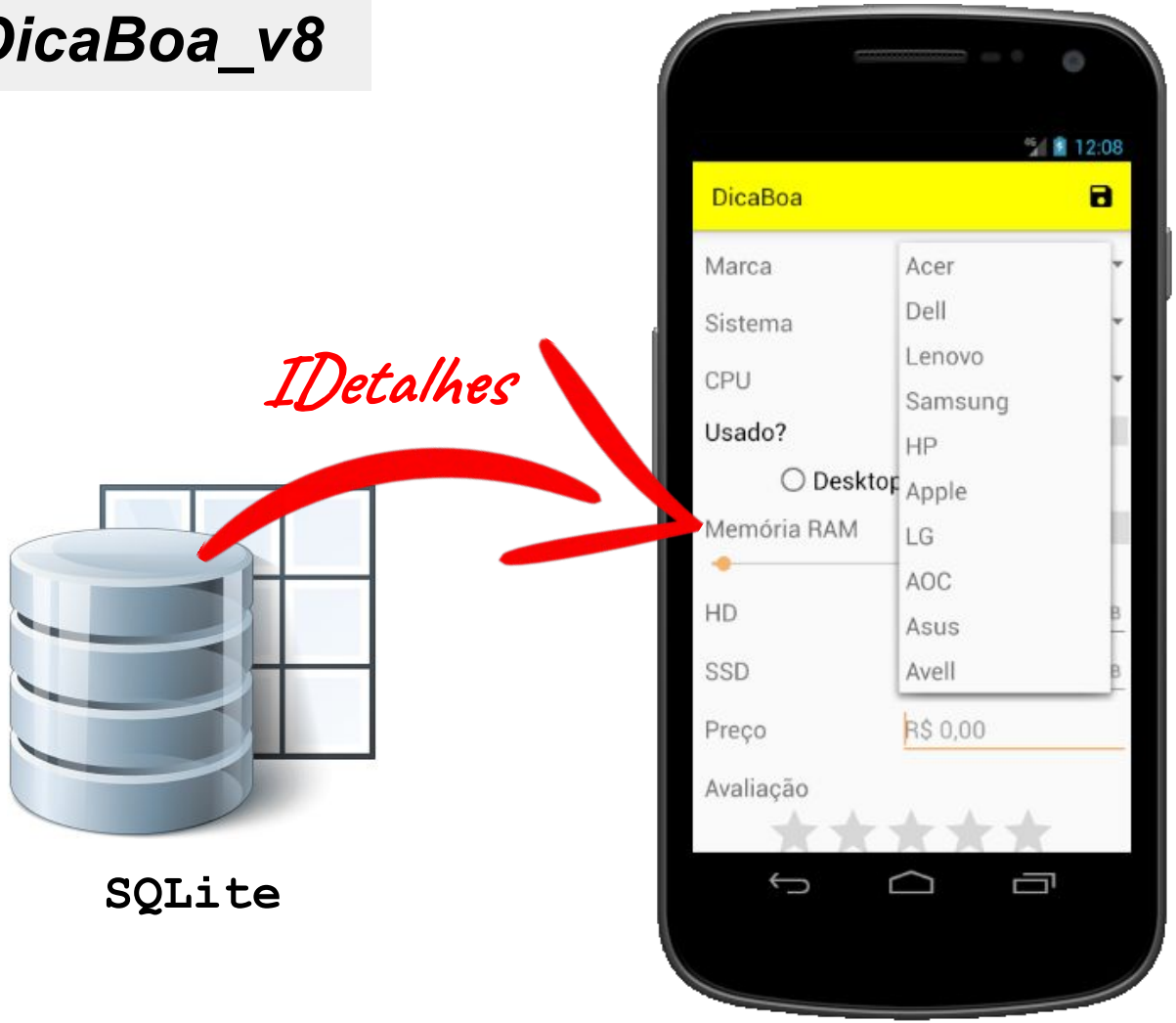
```
1   package com.lgapontes.dicaboa.repositorio;
2
3   import ...
8
9   public class SistemaRepositorio implements IRepositorio<IDetalhe> {
10
11      private Repositorio repositorio;
12
13      public SistemaRepositorio(Repositorio repositorio) {
14          this.repositorio = repositorio;
15      }
16
17      @Override
18      public List<IDetalhe> listar() {
19          return this.repositorio.listarDetalhes(Sistema.class, tabela: "sistemas");
20      }
21  }
22
```

Resolvendo o Exercício

ProcessadorRepositorio.java

```java
package com.lgapontes.dicaboa.repositorio;

import ...

public class ProcessadorRepositorio implements IRepositorio<IDetalhe> {

    private Repositorio repositorio;

    public ProcessadorRepositorio(Repositorio repositorio) {
        this.repositorio = repositorio;
    }

    @Override
    public List<IDetalhe> listar() {
        return this.repositorio.listarDetalhes(Processador.class,  tabela: "processadores");
    }
}
```

DicaBoa_v8



Resolvendo o Exercício

Agora o App trabalha com a interface *IDetalhe* para obter o valor das listas.

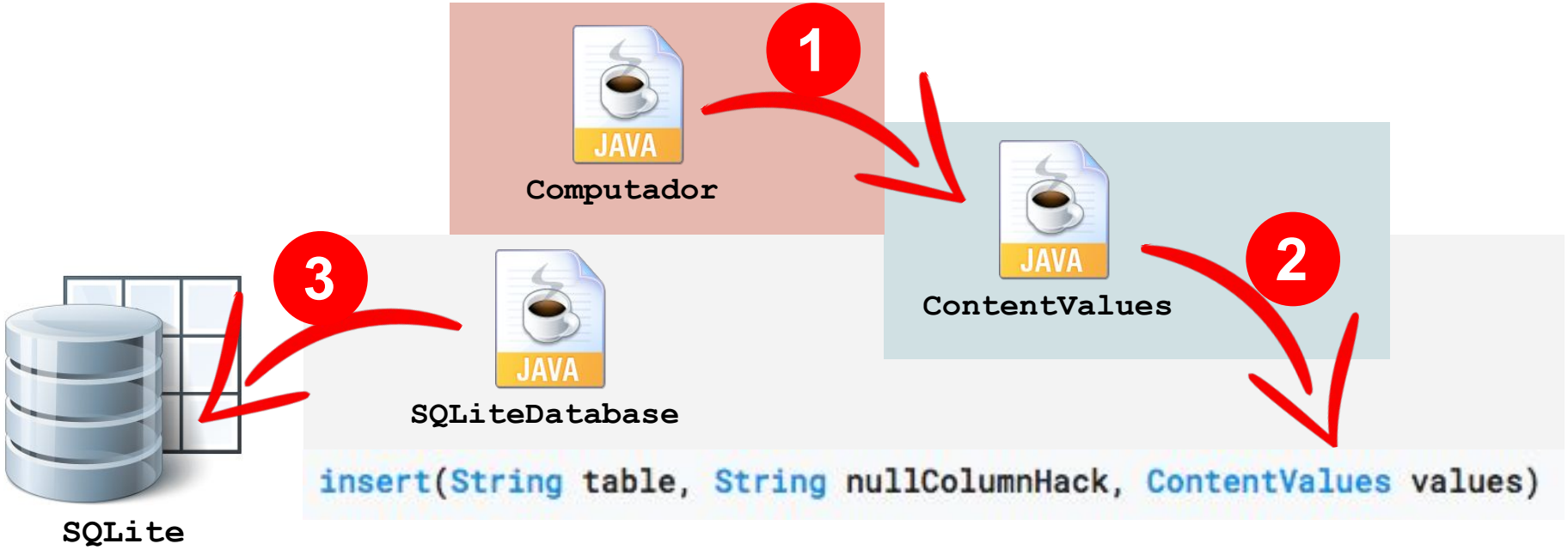
Salvando um novo *computador* (*DicaBoa_v9*)

Vamos agora salvar um novo computador no banco SQLite quando o usuário tocar sobre o ícone de menu do ActionBar.

Para isso, podemos criar um novo método ***salvarComputador()*** na classe ***Repositorio***.

Salvando um novo *computador* (*DicaBoa_v9*)

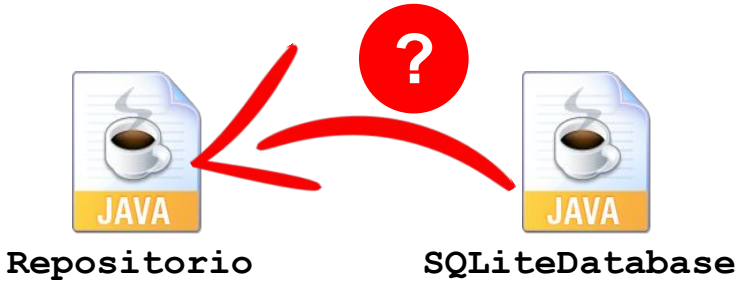
Relembrando, o (1) ícone salvar do *ActionBar* já cria um objeto *Computador*. Este objeto pode (2) ter seus dados salvos em um *ContentValues* que, ao ser passado como parâmetro para o método *insert()* de *SQLiteDatabase*, (3) salvará os dados no banco de dados na tabela informada no parâmetro *table*.



Salvando um novo *computador* (*DicaBoa_v9*)

No método *popularTabela()* que criamos para inserir os dados das *marcas, sistemas* e *processadores*, nós já tínhamos uma instância de *SQLiteDatabase* repassada automaticamente no método *onCreate()* do *Repositorio*.

```
@Override
public void onCreate(SQLiteDatabase db) {...}

private void popularTabela(SQLiteDatabase db, String tabela, String[] itens) {...}
```



Porém, como vamos obter uma instância de *SQLiteDatabase* no método *salvarComputador()?*

Salvando um novo *computador* (*DicaBoa_v9*)

`getReadableDatabase()`	VS	`getWritableDatabase()`
Obtém um *SQLiteDatabase* para **leitura** de dados.		Obtém um *SQLiteDatabase* para **escrita** de dados.

Assim como fizemos no método *listarDetalhes()*, podemos obter uma instância do banco através dos métodos *getReadableDatabase()* ou *getWritableDatabase()* de acordo com a necessidade do uso. Ambos são obtidos da superclasse *SQLiteOpenHelper*.

Computador

ContentValues

SQLite

Computador
- uuid: String
- marca: Marca
- sistema: Sistema
- cpu: Processador
- usado: boolean
- tipo: String
- memoriaRam: int
- hd: int
- ssd: int
- preco: String
- nota: double

```
ContentValues c = new ContentValues();
c.put("uuid",computador.getUUID());
c.put("codigoMarca",
        computador.getMarca().getCodigo());
c.put("codigoSistema",
        computador.getSistema().getCodigo());
c.put("codigoProcessador",
        computador.getCpu().getCodigo());
c.put("usado",computador.isUsado());
c.put("tipo",computador.getTipo());
c.put("memoriaRam",
        computador.getMemoriaRam());
c.put("hd",computador.getHd());
c.put("ssd",computador.getSsd());
c.put("preco",computador.getPreco());
c.put("nota",computador.getNota());
```

computadores
uuid (PK): CHAR(36)
codigoMarca (FK): BIGINT
codigoSistema (FK): BIGINT
codigoProcessador (FK): BIGINT
usado: BOOLEAN
tipo: VARCHAR(40)
memoriaRam: INT
hd: INT
ssd: INT
preco: VARCHAR(40)
nota: DOUBLE

```java
public void salvarComputador(Computador computador) {
    try {

        SQLiteDatabase db = getWritableDatabase();

        ContentValues c = new ContentValues();
        c.put("uuid",computador.getUUID());
        c.put("codigoMarca",computador.getMarca().getCodigo());
        c.put("codigoSistema",computador.getSistema().getCodigo());
        c.put("codigoProcessador",computador.getCpu().getCodigo());
        c.put("usado",computador.isUsado());
        c.put("tipo",computador.getTipo());
        c.put("memoriaRam",computador.getMemoriaRam());
        c.put("hd",computador.getHd());
        c.put("ssd",computador.getSsd());
        c.put("preco",computador.getPreco());
        c.put("nota",computador.getNota());

        db.insert( table: "computadores",  nullColumnHack: null, c);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

O código do método *salvarComputador()* pode ser visualizado ao lado.

Veja que obtemos o *SQLiteDatabase* a partir do método *getWritableDatabase()*

Devemos colocar seu conteúdo em um bloco *try/catch* para evitar possíveis erros relacionados ao salvamento.

Salvando um novo *computador* (*DicaBoa_v9*)

Vamos criar uma classe *ComputadorRepositorio* para organizar a ações de persistência relacionadas ao objeto *Computador*.

ComputadorRepositorio.java

```java
package com.lgapontes.dicaboa.repositorio;

import com.lgapontes.dicaboa.modelo.Computador;

public class ComputadorRepositorio {

    private Repositorio repositorio;

    public ComputadorRepositorio(Repositorio repositorio) {
        this.repositorio = repositorio;
    }

    public void salvar(Computador computador) {
        this.repositorio.salvarComputador(computador);
    }

}
```

Ele deve conter um *constructor* para receber uma instância do *Repositorio.*

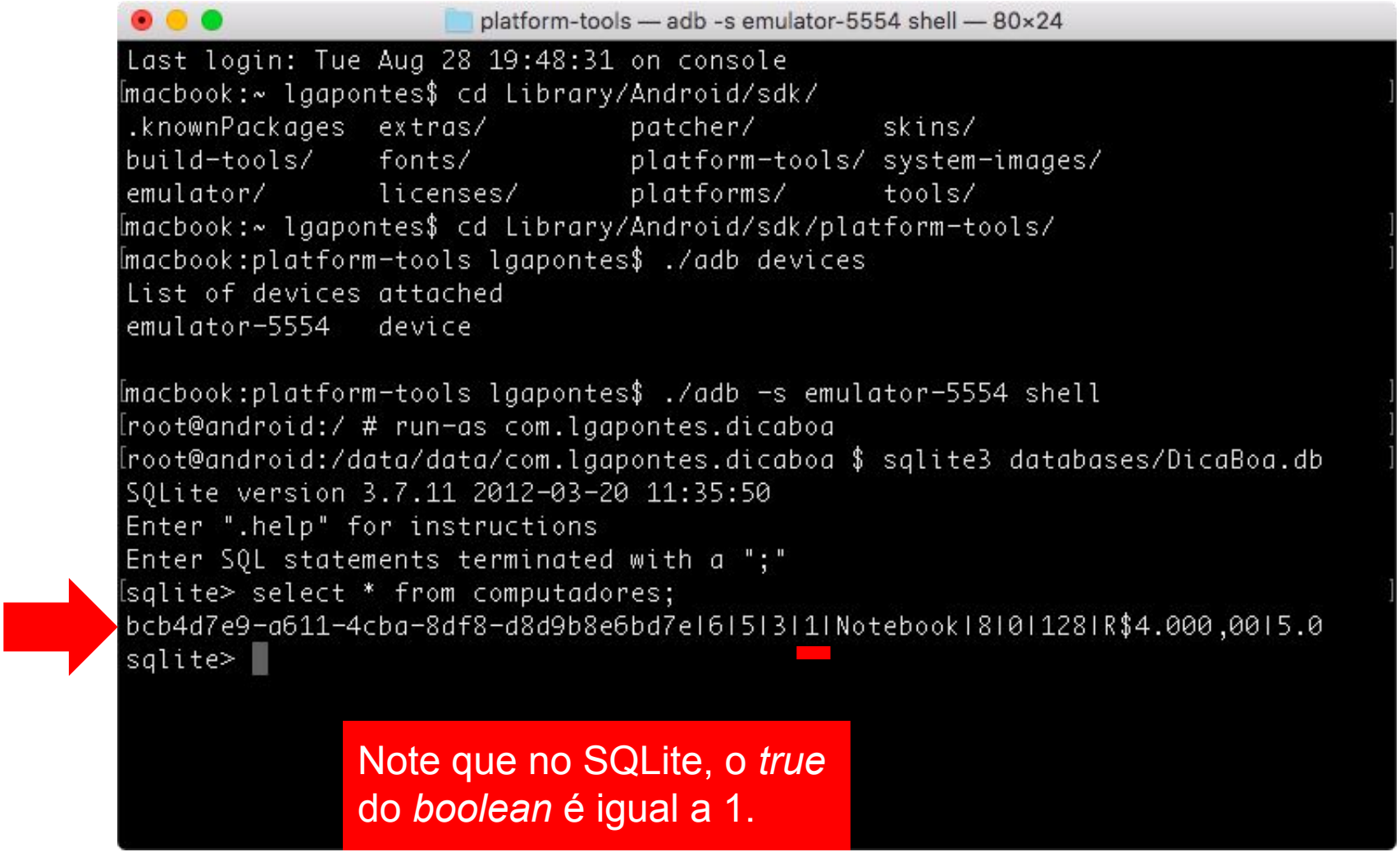
Deve conter também um método *salvar()* que simplesmente invoca o método *salvarComputador()* de *Repositorio*.

Salvando um novo *computador* *(DicaBoa_v9)*

Por fim, devemos alterar o método *criarComputador()* de *CadastroActivity* para criar um novo *ComputadorRepositorio* e salvar o computador. Veja que devemos repassar a instância de *Repositorio* como parâmetro do constructor.

```
private void criarComputador() {

    Computador computador = new Computador();

    // Restante do código...

    ComputadorRepositorio db = new ComputadorRepositorio(this.repositorio);
    db.salvar(computador);

    Toast.makeText( context: this, text: "Dados salvos com sucesso!",
            Toast.LENGTH_SHORT).show();
    Log.i( tag: "COMPUTADOR", computador.toString());
}
```

Ao rodar o App, veremos um comportamento igual ao já visualizado na última aula. A diferença, porém, é que os dados foram salvos no SQLite (veja no próximo slide).


platform-tools — adb -s emulator-5554 shell — 80×24
Last login: Tue Aug 28 19:48:31 on console
macbook:~ lgapontes$ cd Library/Android/sdk/
.knownPackages extras/   patcher/   skins/
build-tools/   fonts/   platform-tools/ system-images/
emulator/   licenses/   platforms/   tools/
macbook:~ lgapontes$ cd Library/Android/sdk/platform-tools/
macbook:platform-tools lgapontes$ ./adb devices
List of devices attached
emulator-5554   device
macbook:platform-tools lgapontes$ ./adb -s emulator-5554 shell
root@android:/ # run-as com.lgapontes.dicaboa
root@android:/data/data/com.lgapontes.dicaboa $ sqlite3 databases/DicaBoa.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from computadores;
bcb4d7e9-a611-4cba-8df8-d8d9b8e6bd7e|6|5|3|1|Notebook|8|0|128|R$4.000,00|5.0
sqlite>
Note que no SQLite, o true
do boolean é igual a 1.

Obrigado!

Na próxima aula vamos construir uma Activity para exibir os dados dos computadores cadastrados.