

# Prog Disp Móveis



## Competências:

- Desenvolvimento aplicativos mobile
- Publicar projetos em servidor

## Bases Tecnológicas, científicas e instrumentais (conteúdos):

- Arquitetura Android;
- Plataforma Android SDK e Eclipse;
- Integração usando Intents;
- Persistência com SQLite;
- Conectividade com serviços remotos e sincronização; e
- JSON
- API de arquivos

## Situação de Aprendizagem:

- Desenvolver um aplicativo em android para persistir dados locais usando Shared Preferences, Arquivo e SQLite.

# Prog Disp Móveis



Existem 3 formas de armazenar dados no dispositivo:

## 1) Arquivos comuns (texto ou binário)

Classes FileInputStream e FileOutputStream

Armazenados em **/data/data/<pacote>/files**

## 2) Arquivos de preferências

Classe SharedPreferences

## 3) Bancos de Dados usando SQLite

Classes SQLiteDatabase, SQLiteOpenHelper

Armazenados em **/data/data/<pacote>/databases**

## Shared Preferences

- Classe para armazenar preferências ou configurações da aplicação;
- É possível usar quantas quiser, desde que associadas a um nome de arquivo; e
- Guarda informações na forma de pares chave-valor.

# Prog Disp Móveis



## Shared Preferences

`boolean contains(String key)`

Retorna true caso a chave key exista.

`boolean getBoolean(String key,  
boolean)`

Retorna o valor de uma chave.

`float getFloat(String key, float)`

O segundo parâmetro indica o valor default que deve ser retornado caso a chave não exista.

`int getInt(String key, int)`

`long getLong(String key, long)`

Note que é necessário que o desenvolvedor saiba o tipo do dado armazenado na chave. Caso a chave exista mas o tipo seja diferente do especificado no método será lançado um `ClassCastException`.

`String getString(String key, String)`

`SharedPreferences.Edit edit()`

Retorna um editor para as preferências que permite editar e salvar informações.

# Prog Disp Móveis



## Shared Preferences

<code>boolean commit()</code>	Salva as preferências no objeto SharedPreferences associado e em disco. Operação síncrona, ou seja, só retorna após ter salvo em disco retornando true em caso de sucesso, ou false senão.
<code>void apply()</code>	Aplica os novos valores das preferências ao objeto em memória e retorna. O armazenamento em disco será feito de forma assíncrona, ou seja, não é possível saber se houve algum error durante o armazenamento.
<code>clear()</code>	Remove todos os valores de preferências do objeto.
<code>putBoolean(String key, boolean)</code> <code>putFloat(String key, float)</code> <code>putInt(String key, int)</code> <code>putLong(String key, long)</code> <code>putString(String key, String)</code>	Adiciona ou altera o valor de uma chave. O nome do método e o segundo parâmetro indicam o tipo do valor da chave.
<code>remove(String key)</code>	Remove a chave definida por key e consequentemente o seu valor.

# Prog Disp Móveis



**SQLite - [www.sqlite.org](http://www.sqlite.org)**



SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.

# Prog Disp Móveis



- Linguagem padrão para consulta a bancos de dados
- Construções básicas
  - SELECT Campo1, Campo2, ... FROM Tabela1, Tabela2 WHERE <condição> ORDER BY CampoX;
  - INSERT INTO Tabela(Campo1, Campo2, ...) VALUES (Valor1, Valor2, ...);
  - DELETE FROM Tabela WHERE <condição>;
  - UPDATE Tabela SET Campo1 = <expressao1>, Campo2 = <expressao2> WHERE <condição>;
- Criação e exclusão de tabelas []=opcional
  - CREATE TABLE Tabela (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    Campo1 Tipo1 [[UNIQUE] NOT NULL],  
    Campo2 Tipo2 [NOT] NULL,  
    ...);
  - DROP TABLE IF EXISTS Tabela

# Prog Disp Móveis



Auxilia abertura e criação de um banco de dados

<code>SQLiteOpenHelper(Context, String name, SQLiteDatabase.CursorFactory, int version)</code>	Cria um objeto para auxiliar no gerenciamento da base de dados.
<code>SQLiteDatabase getReadableDatabase()</code>	Cria ou abre um banco de dados apenas para leitura.
<code>SQLiteDatabase getWritableDatabase()</code>	Cria ou abre um banco de dados para leitura e escrita.
<code>void onCreate(SQLiteDatabase db)</code>	Chamado quando o banco de dados precisa ser criado, ou seja, não existe.
<code>void onOpen(SQLiteDatabase db)</code>	Chamado quando o banco de dados é aberto.
<code>void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)</code>	Chamado quando a versão do banco de dados sendo aberto é diferente da versão existente.



# Prog Disp Móveis



Representa o banco de dados e executa operações de consulta, inclusão, alteração e exclusão de registros

<code>static SQLiteDatabase openDatabase(String path,                     CursorFactory factory, flags)</code>	Abre banco de dados de acordo com os flags: OPEN_READWRITE, OPEN_READONLY, CREATE_IF_NECESSARY, NO_LOCALIZED_COLLATORS.
<code>static SQLiteDatabase openOrCreateDatabase(String path,                     CursorFactory factory) static SQLiteDatabase openOrCreateDatabase(File file,                     CursorFactory factory)</code>	Abre ou cria banco de dados.  Equivalente a usar openDatabase(...) com flags = CREATE_IF_NECESSARY
<code>boolean    isOpen()</code>	Verifica se está aberto
<code>void       close()</code>	Fecha banco de dados
<code>void       execSQL(String sql)</code>	Executa script SQL que não seja SELECT. Exemplo: CREATE TABLE, INSERT, UPDATE, etc.

# Prog Disp Móveis



```
Cursor query(String table, String[] columns,  
             String selection, String[] selectionArgs,  
             String groupBy, String having,  
             String orderBy)  
Cursor query(table, columns, selection,  
             selectionArgs, groupBy, having,  
             orderBy, limit)  
Cursor query(boolean distinct, table, columns,  
             selection, selectionArgs, groupBy,  
             having, orderBy, String limit)
```

Mostra e executa um SQL de consulta na forma:

```
SELECT <distinct> <columns>  
FROM <table>  
WHERE <selection+selectionArgs>  
GROUP BY <groupBy>  
HAVING <having>  
ORDER BY <orderBy>  
LIMIT <limit>
```

```
long insert(table, null, ContentValues values)
```

Insere um registro e retorna o id.

```
INSERT INTO <table> (values) VALUES (values)
```

```
int update(table, ContentValues values,  
           whereClause, whereArgs)
```

Altera registro(s) e retorna quantidade de linhas modificadas.

```
UPDATE <table> SET <values>  
WHERE <whereClause+whereArgs>
```

```
int delete(table, whereClause, whereArgs)
```

Remove registro(s) e retorna quantidade de linhas modificadas.

```
DELETE FROM <table>  
WHERE <whereClause+whereArgs>
```

# Prog Disp Móveis



login

senha

SHARED

ARQUIVO

SQLITE

login

id: login

senha

id: senha

SHARED

ARQUIVO

SQLITE


# Prog Disp Móveis



## ▼ Gradle Scripts

 build.gradle (Project: LoginDados)

 build.gradle (Module: app)

 gradle-wrapper.properties (Gradle Version)

 proguard-rules.pro (ProGuard Rules for app)

 gradle.properties (Project Properties)

 settings.gradle (Project Settings)

implementation 'com.fasterxml.jackson.core:jackson-databind:2.9.8'

# Prog Disp Móveis



**@Override**

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);
```

```
//SharedPreferences
```

```
this.criarSharedPreferencesAutenticacao();
```

```
//Arquivo
```

```
try {  
    this.criarArquivoAutenticacao();  
} catch (JsonProcessingException e) {  
    e.printStackTrace();  
}
```

```
//Banco
```

```
this.criarBancoAutenticacao();  
}
```

# Prog Disp Móveis



```
public void criarSharedPreferencesAutenticacao(){  
    SharedPreferences minhasPreferencias =  
    PreferenceManager.getDefaultSharedPreferences(MainActivity.this);  
    SharedPreferences.Editor myEditor = minhasPreferencias.edit();  
    myEditor.putString("LOGIN", "admin");  
    myEditor.putString("SENHA", "12345678");  
    myEditor.commit();  
}
```

# Prog Disp Móveis



```
public void autenticarSharedPreferences(View v){  
    String login = ((EditText)findViewById(R.id.login)).getText().toString();  
    String senha = ((EditText)findViewById(R.id.senha)).getText().toString();  
  
    SharedPreferences myPreferences = PreferenceManager.getDefaultSharedPreferences(MainActivity.this);  
    String loginString = myPreferences.getString("LOGIN", "unknown");  
    String senhaString = myPreferences.getString("SENHA", "unknown");  
  
    if(loginString.equalsIgnoreCase(login) && senhaString.equalsIgnoreCase(senha))  
        Toast.makeText(this, "SP OK", Toast.LENGTH_LONG).show();  
    else  
        Toast.makeText(this, "SP Error", Toast.LENGTH_LONG).show();  
}
```

# Prog Disp Móveis



```
public void criarArquivoAutenticacao() throws JsonProcessingException {
```

```
    ObjectMapper objectMapper = new ObjectMapper();
```

```
    Login login = new Login("admin", "12345678");
```

```
    File internalStorageDir = getFilesDir();
```

```
    File arquivo = new File(internalStorageDir, "login.json");
```

```
    try {
```

```
        objectMapper.writeValue(arquivo, login);
```

```
    } catch (IOException e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
}
```



# Prog Disp Móveis



```
public void autenticarArquivo(View v){
    try {
        String login = ((EditText)findViewById(R.id.login)).getText().toString();
        String senha = ((EditText)findViewById(R.id.senha)).getText().toString();

        final ObjectMapper mapper = new ObjectMapper();
        File internalStorageDir = getFilesDir();
        File jsonFile = new File(internalStorageDir, "login.json");

        Login Login = mapper.readValue(jsonFile, Login.class);

        String loginString = Login.getLogin();
        String senhaString = Login.getSenha();

        if(loginString.equalsIgnoreCase(login) && senhaString.equalsIgnoreCase(senha))
            Toast.makeText(this, "Arq OK", Toast.LENGTH_LONG).show();
        else
            Toast.makeText(this, "Arq Error", Toast.LENGTH_LONG).show();

    } catch (IOException e) {

        e.printStackTrace();
    }
}
```

# Prog Disp Móveis



```
public void criarBancoAutenticacao(){
    SQLiteDatabase myDB = openOrCreateDatabase("login.db", MODE_PRIVATE, null);
    myDB.execSQL("CREATE TABLE IF NOT EXISTS usuario (login VARCHAR(20), senha VARCHAR(20))");
    ContentValues registro = new ContentValues();
    registro.put("login", "admin");
    registro.put("senha", "12345678");

    myDB.insert("usuario", null, registro);

    myDB.close();
}
```

# Prog Disp Móveis



```
public void autenticarBanco(View v){  
    String login = ((EditText)findViewById(R.id.login)).getText().toString();  
    String senha = ((EditText)findViewById(R.id.senha)).getText().toString();  
  
    SQLiteDatabase myDB = openOrCreateDatabase("login.db", MODE_PRIVATE, null);  
    myDB.execSQL("CREATE TABLE IF NOT EXISTS usuario (login VARCHAR(20), senha VARCHAR(20))");  
  
    Cursor myCursor = myDB.rawQuery("select login, senha from usuario", null);  
  
    myCursor.moveToNext();  
    String loginString = myCursor.getString(0);  
    String senhaString = myCursor.getString(1);  
  
    if(loginString.equalsIgnoreCase(login) && senhaString.equalsIgnoreCase(senha))  
        Toast.makeText(this, "SQL OK", Toast.LENGTH_LONG).show();  
    else  
        Toast.makeText(this, "SQL Error", Toast.LENGTH_LONG).show();  
  
    myDB.close();  
  
}
```