FIAP

DESENVOLVIMENTO AVANÇADO MOBILE

GOOGLE ANDROID

DESENVOLVIMENTO ANDROID

PERSISTÊNCIA

ÍNDICE

- SHARED PREFERENCES
- FILES
- SQLITE
  - ACTIVE ANDROID

SHARED PREFERENCES

FIAP

É um formato simples de armazenamento de pares de valores contendo um nome e valor primitivo;

A classe **SharedPreferences** encapsula os dados armazenados e pode ser obtida por meio dos métodos:

getPreferences(int P1): obtém um único arquivo de preferências;

getSharedPreferences(String P1, int P2): obtém um arquivo de preferências com o nome especificado em P1;

O parâmetro P1 do **getPreferences** e P2 do **getSharedPreferences** especificam as permissões dos arquivos:

MODE_PRIVATE - somente a aplicação pode acessar;
MODE_WORLD_READABLE - público somente leitura;
MODE_WORLD_WRITEABLE - público com leitura / escrita;

SHARED PREFERENCES

Uma vez obtido um objeto **SharedPreferences** basta acionar o método **edit** para retornar a interface **Editor**;

Com o **Editor** pode-se criar pares de valores por meio de métodos:

- **putBoolean(String P1, boolean P2);**
- **putFloat(String P1, float P2);**
- **putInt(String P1, int P2);**
- **putLong(String P1, long P2);**
- **putString(String P1, StringP2);**

Onde **P1** representa o nome do valor (campo) e **P2** o valor a ser armazenado no respectivo tipo de dado;

SHARED PREFERENCES

Para confirmar a persistência dos dados chamar o método commit() da classe Editor;

```
SharedPreferences sp = getPreferences(MODE_PRIVATE);
Editor e = sp.edit();
e.putString("username", “rubim");
e.commit();
```

SHARED PREFERENCES

A leitura de uma determinada propriedade deve ser feita diretamente por meio de métodos da classe **SharedPreferences:**

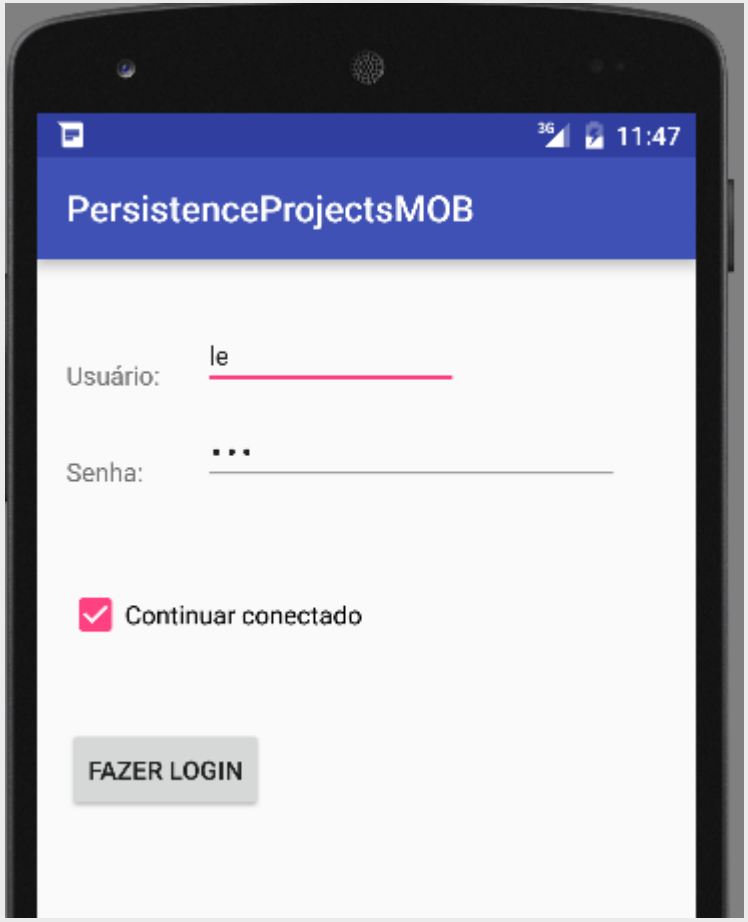
- **getBoolean(String P1, boolean P2);**
- **getFloat(String P1, float P2);**
- **getInt(String P1, int P2);**
- **getLong(String P1, long P2);**
- **getString(String P1, StringP2);**

Onde **P1** representa o nome do valor (campo) e **P2** o valor padrão (default) a ser retornado caso o nome do valor informado não tenha sido ainda persistido;

```
SharedPreferences sp = getPreferences(MODE_PRIVATE);

// Caso o valor username não exista retorna null
String username = sp.getString("username", null);
Toast.makeText(this, username, Toast.LENGTH_SHORT).show();
```

ATIVIDADE 1 – SHARED PREFERENCES



Utilizando SharedPreferences escreva uma aplicação que:

1. Usuário informa o seu usuário e senha e opta por "continuar contectado"
2. Aplicação registra o usuário e senha em uma Shared Preference
3. Usuário executa novamente a aplicação
4. Aplicação lê o usuário e senha salvos anteriormente no SharedPreference e os exibe nos respectivos campos

FILE

Também é possível gravar e ler arquivos nos diretórios do dispositivo. Para tanto, basta utilizar um **FileOutputStream** e o **FileInputStream** respectivamente;

Exemplo de gravação de um texto:

```
// Abre um stream de saída para gravação de arquivos
FileOutputStream fos = openFileOutput("teste.txt", MODE_PRIVATE);
String txt = "Boa noite!";
fos.write(txt.getBytes());
fos.close();
```

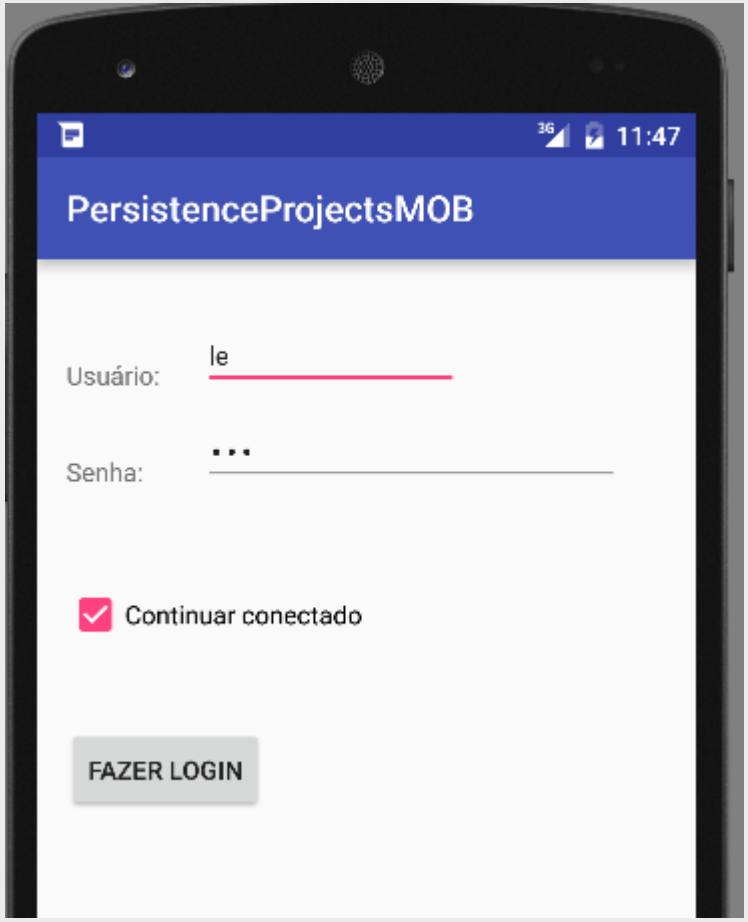
FILE

Exemplo de leitura de um texto:

```
// Abre um stream de entrada para leitura de arquivos
FileInputStream fis = openFileInput("teste.txt");
BufferedReader br = new BufferedReader(new InputStreamReader(fis));

// Lê uma linha do arquivo e a retorna uma String
String txt = br.readLine();
fis.close();
Toast.makeText(this, txt, Toast.LENGTH_SHORT).show();
```

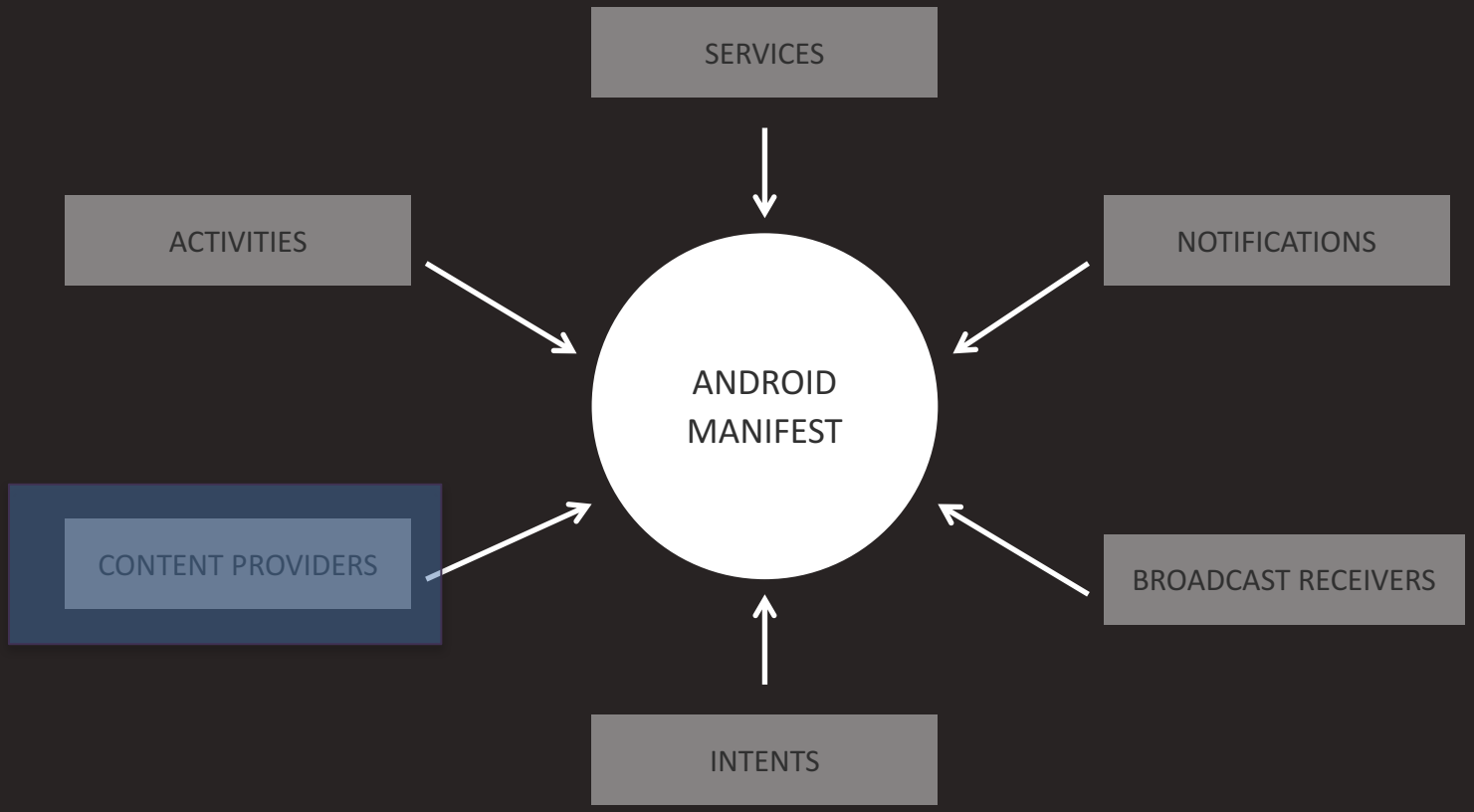
ATIVIDADE 2 – FILES



Utilizando Files escreva uma aplicação similar a escrita com SP, onde:

1. Usuário informa o seu usuário e senha e opta por "continuar contectado"
2. Aplicação registra o usuário e senha em um Arquivo
3. Usuário executa novamente a aplicação
4. Aplicação lê o usuário e senha salvos anteriormente no Arquivo e os exibe nos respectivos campos

PRINCIPAIS COMPONENTES DA APLICAÇÃO



Android Manifest é um arquivo XML (*AndroidManifest.xml*) que define e integra os componentes de uma aplicação vistos acima.

CONTENT PROVIDER

FIAP

Os Content Providers oferecem uma base de dados que pode ser compartilhada entre diversas aplicações.

Alguns exemplos: base de contatos e calendário

Todo Content Provider é identificado por meio de uma **URI**

A partir da **URI** pode-se resolver qual o Content Provider a ser acessado por meio da classe **ContentResolver**

Por exemplo, a base local de contatos e calendários são identificados por:

content://com.android.contacts/contacts, também representado pela constante ContactsContract.Contacts.CONTENT_URI

content://com.android.calendar/calendars, constante CalendarContract.Calendars.CONTENT_URI

PERMISSÕES

Para ter acesso tanto de leitura quanto de escrita na base de contatos é necessário definir uma permissão no AndroidManifest.xml

Calendário

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

Contatos

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

PERMISSÕES - MARSHMELLOW

FIAP

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M &&
    checkSelfPermission(Manifest.permission.READ_CONTACTS) !=
                                                    PackageManager.PERMISSION_GRANTED) {

    requestPermissions(new String[]{Manifest.permission.READ_CONTACTS},REQUEST_PERMISSIONS_CODE);

} else {

     consultar();
}
```

```
@Override
public void onRequestPermissionsResult(int requestCode, String permissions[], int[]
grantResults) {

    if (requestCode == REQUEST_PERMISSIONS_CODE) {
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Permissão OK
            consultar();
        } else {
            Toast.makeText(this, "Permissão necessária", Toast.LENGTH_SHORT).show();
        }
    }
}
```

BASE DE CONTATOS

A base de contatos registrados no dispositivo é constituída de tabelas onde os dados dos contatos são distribuídos

Cada tabela é representada por uma Uri:

ContactsContract.Contacts.CONTENT_URI → Contém dados básicos como nome

ContactsContract.CommonDataKinds.Phone.CONTENT_URI → dados sobre contato telefônico

ContactsContract.CommonDataKinds.Email.CONTENT_URI → dados sobre endereços de email do contato

BASE DE CONTATOS

FIAP

Cada uma das tabelas identificadas no slide anterior possuem campos onde os dados sobre o contato são armazenados.

ContactsContract.Contacts.CONTENT_URI
ContactsContract.Contacts._ID → ID do contato (chave primária)
ContactsContract.Contacts.DISPLAY_NAME → nome do contato
ContactsContract.Contacts.HAS_PHONE_NUMBER → possui telefone cadastrado

ContactsContract.CommonDataKinds.Phone.CONTENT_URI
ContactsContract.CommonDataKinds.Phone.CONTACT_ID → ID do contato (chave estrangeira)
ContactsContract.CommonDataKinds.Phone.NUMBER → número do telephone

ContactsContract.CommonDataKinds.Email.CONTENT_URI
ContactsContract.CommonDataKinds.Email.CONTACT_ID → ID do contato (chave estrangeira)
ContactsContract.CommonDataKinds.Email.DATA → endereço de e-mail

Repare que as tabelas onde os dados telefônicos e de e-mail são ligadas à tabela principal do contato por meio de chaves estrangeiras (campos de ID)

CONSULTANDO UM CONTATO

FIAP

O exemplo abaixo apresenta uma consulta à base de contatos:

```
ContentResolver resolver = getContentResolver();
Uri CONTENT_URI = ContactsContract.Contacts.CONTENT_URI;
Cursor cursor = resolver.query(CONTENT_URI, null,null, null, null);
```

O método query possui os seguintes parâmetros (opcionais):

query(Uri P1, String[] P2, String P3, String[] P4, String P5);

P1 – Uri que identifica unicamente o content provider
P2 – Campos a serem retornados (colunas)
P3 – Critério de seleção (where)
P4 – Valores para os critérios de seleção definidos em P3
P5 – Campos de ordenação do resultado

ATIVIDADE 3 – CONSULTANDO UM TELEFONE

FIAP

```
ContentResolver cr = getContentResolver();

Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI,
null, null, null, null);


if (cursor.moveToFirst()) {
    do {
        String name = cursor.getString(
cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));

        Toast.makeText(this, name, Toast.LENGTH_SHORT).show();

    } while (cursor.moveToNext());

}

cursor.close();
```

SQLITE

O Android possui um banco de dados integrado denominado SQLite;

Ele aceita comandos SQL para criação de tabelas, consultas, etc... (vide **http://www.sqlite.org/lang.html**)

Contudo, apresenta algumas limitações:

- Suporta apenas os tipos de dados TEXT, INTEGER e REAL;
- Não oferece suporte à chaves estrangeiras;
- Não mantém a integridade entre os tipos de dados (exemplo: inserir String em campo INTEGER);

Apesar disso é extremamente leve em termos de consumo de memória;

SQLITE

O SQLite separa instruções DDL de DML;

As instruções DDL para a criação da estrutura do banco devem ficar em uma classe que estende **SQLiteOpenHelper**;

Nela, os métodos **onCreate** (que cria o banco e suas tabelas) e **onUpdate** (atualiza a estrutura do banco) devem ser implementados;

Além disso, a classe **SQLiteOpenHelper** possui o seguinte construtor:

SQLiteOpenHelper(Context P1, String P2, CursorFactory P3, int P4)

Onde:

- P1 - Contexto associado ao banco (aplicação);
- P2 - Nome do banco de dados;
- P3 - Cursor padronizado para a pesquisa de dados (pode ser null);
- P4 - Versão do banco de dados (todas as vezes que a versão é atualizada, o método OnUpdate é executado).

SQLITE

FIAP

Exemplo:

```
public class MeuDb extends SQLiteOpenHelper {

    public MeuDb(Context context) {
        super(context, "MeuDb", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        final String sql = "CREATE TABLE TAB_CLIENTE (COD_CLIENTE INTEGER
        PRIMARY KEY AUTOINCREMENT, NOM_CLIENTE TEXT)";
        db.execSQL(sql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int vAntiga, int vNova) {
        db.execSQL("DROP TABLE IF EXISTS TAB_CLIENTE");
        onCreate(db);
    }
}
```

Versão do banco.

O Métodos execSQL executa instruções DDL.

Recria o banco todas as vezes que a versão é atualizada.

SQLITE

Para executar instruções DML é necessário abrir o banco de dados por meio do método **getWritableDatabase** que retorna um objeto do tipo **SQLiteDatabase**;

A classe **SQLiteDatabase** oferece os métodos **insert**, **delete** e **update**:

insert (String P1, String P2, ContentValues P3), onde:
P1 é o nome da tabela, **P2** a forma de tratamento de valores nulos (opcional) e **P3** os valores a serem inseridos. O insert retorna um long com o ID gerado para o registro inserido;

update(String P1, ContentValues P2, String P3, String[] P4), onde **P1** é o nome da tabela, **P2** os valores que serão atualizados, **P3** a cláusula where e **P4** os parâmetros da cláusula where;

delete(String P1, String P2, String[] P3), onde **P1** é o nome da tabela, **P2** a cláusula where e **P3** os parâmetros da cláusula where;

SQLITE

Os valores a serem utilizados nas instruções insert e update são encapsulados em objetos da classe **ContentValues**;

A classe ContentValues possui um método put que permite informarmos pares contendo nome da coluna e valor;

Exemplo:

```
ContentValues cv = new ContentValues();
cv.put("NOM_CLIENTE", "JOAO DA SILVA");
db.insert("TAB_CLIENTE",  null, cv);


ContentValues cv2 = new ContentValues();
cv2.put("NOM_CLIENTE", "BATISTA ANTONIO");
// Atualiza o registro onde COD_CLIENTE = 1
db.update("TAB_CLIENTE", cv2, "COD_CLIENTE = ?", new String[]{"1"});


// Apaga o registro onde COD_CLIENTE = 2
db.delete("TAB_CLIENTE", "COD_CLIENTE = ?", new String[]{"2"});
```

SQLITE

Consultas podem ser realizadas na base de dados por meio do método **query** da classe **SQLiteDatabase** que retorna um objeto **Cursor**, utilizado para manipular o conjunto de resultados obtidos;

Cursor query (String P1, String[] P2, String P3, String[] P4, String P5, String P6, String P7)

Onde:
P1 - nome da tabela;
P2 - colunas desejadas;
P3 - cláusula **where;**
P4 - parâmetros da cláusula **where**;
P5 - cláusula **group by**;
P6 - cláusula **having**;
P7 - cláusula **order by**;

```
Cursor c = db.query(
        "TAB_CLIENTE",
        new String[]{"NOM_CLIENTE"},
        "COD_CLIENTE = ?",
        new String[]{"3"},
        null,
        null,
        "NOM_CLIENTE");
```

SQLITE

A classe Cursor oferece mecanismos de manipulação dos dados por meio de métodos:

- **getCount()** → total de registros recuperados;
- **moveToFirst()** → posiciona o cursor no primeiro registro do conjunto de dados;
- **moveToNext()** → move para o próximo registro. Retorna false caso não existam mais registros no conjunto de dados;
- **getInt(int P1), getDouble(int P1), getFloat(int P1), getString(int P1)** → retorna o valor de uma coluna informada no parâmetro **P1** conforme o tipo de dado especificado no método;
- **close()** → fecha o cursor;

Exemplo:

```
c.moveToFirst();
Log.i("PersistenciaActivity", "Total: " + c.getCount());
do {
      Log.i("PersistenciaActivity",
      "COD: " + c.getInt(0) +
      " NOME: " + c.getString(1));
} while (c.moveToNext());
c.close();
```

ATIVIDADE 4 - SQLITE

FIAP

Adapte o processo de Login realizando o cadastro de uma tabela.

```
public class LoginDAO {

    private static final String DATABASE_NAME = "logindb.db";
    private static final int DATABASE_VERSION = 1;
    private static final String TABLE_NAME = "TBLOGIN";
    private Context context;
    private SQLiteDatabase db;
    private SQLiteStatement insertStmt;
    private static final String INSERT = "insert into " + TABLE_NAME + " (nome, cpf) values (?,?)";

    public LoginDAO (Context context) {
        this.context = context;
        OpenHelper openHelper = new OpenHelper(this.context);
        this.db = openHelper.getWritableDatabase();
        this.insertStmt = this.db.compileStatement(INSERT);
    }
    public long insert(Login login) {
        this.insertStmt.bindString(1, login.getUser());
        this.insertStmt.bindString(2, login.getPassword());
        return this.insertStmt.executeInsert();
    }

    public void deleteAll() {
        this.db.delete(TABLE_NAME, null, null);
    }
```

```
public class Login implements Serializable {

    private static final long serialVersionUID = 1L;

    private long id;
    private String user;
    private String password;
```

FIAP