INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: SALVANDO DADOS

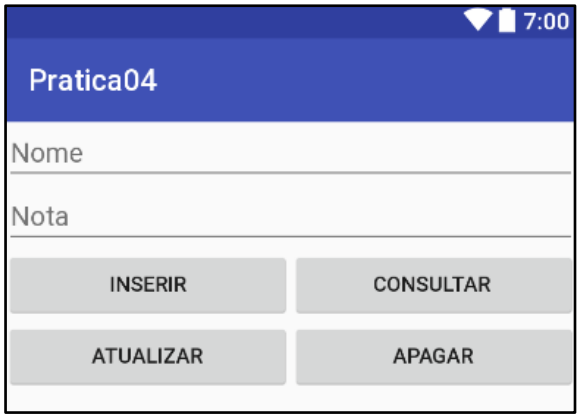
Aula Prática 04

Parte 1: Salvando dados numa arquivo de preferências simples

Passo 1: Crie uma novo projeto contendo uma atividade em branco:

Application Name:	Pratica04
Domain name:	pdm.tads.ifpe.edu.br
Template:	Empty Activity
Activity:	MainActivity

Passo 2: Construa a interface da atividade de forma a ficar como na figura abaixo.



Campo “Nome”: edit_name
Campo “Nota”: edit_grade
Botão “Inserir”: button_save
Botão “Consultar”: button_query
Botão “Atualizar”: button_update
Botão “Apagar”: button_delete

Passo 3: Associe o tratador `buttonInsertClick()` ao botão “Inserir” na atividade principal, o qual deve conter o código abaixo:

```
public void buttonInsertClick(View view) {
    String name = ((EditText) findViewById(R.id.edit_name)).getText().toString();
    String grade = ((EditText) findViewById(R.id.edit_grade)).getText().toString();

    SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putString(name, grade);
    editor.apply();

    Toast.makeText(this, "Salvo: " + name, Toast.LENGTH_SHORT).show();
}
```

O botão “Inserir” salva a combinação nome e nota como uma preferência (preferência=valor) num arquivo de preferências associado com a atividade (obtido via `getPreferences()`). O parâmetro `Context.MODE_PRIVATE` diz que o arquivo é privado à atividade, não podendo ser acessado por outras aplicações. É possível criar arquivos de preferências com nomes arbitrários (e não associados a atividades) através do método `getSharedPreferences()`, passando o nome do arquivo e modo de acesso como parâmetros.

Para editar as preferências (adiciar combinações nome=valor) é preciso criar um objeto Editor (método `edit()` do objeto de preferências). Ao final da edição usa-se o método `apply()` ou `commit()` do Editor para persistir as informações. (`commit()` salva imediatamente, enquanto `apply()` salva em background).

Apesar do nome, objetos `SharedPreferences` nem sempre são compartilhados (i.e., `Context.MODE_PRIVATE`) e não necessariamente armazenam preferências ou configurações do usuário, podendo armazenar qualquer combinação nome=valor. (SharedPreferences, porém, são usados pela API do Android para a construção de atividades de configuração (*Settings*)).

Passo 4: Associe o tratador `buttonQueryClick()` ao botão “Consultar” na atividade principal, contendo o código abaixo:

```
public void buttonQueryClick(View view) {
    String name = ((EditText)findViewById(R.id.edit_name)).getText().toString();
    SharedPreferences prefs = getPreferences(Context.MODE_PRIVATE);
    String grade = prefs.getString(name, "[Não encontrado]");
    ((EditText)findViewById(R.id.edit_grade)).setText(grade);
}
```

Para recuperar uma preferência são usados métodos como `getString()` ou `getInt()`, passando o nome da preferência e um valor default que é retornado se a preferência não existir.

Passo 5 (desafio): Implemente o tratador do botão “Atualizar”: nesse caso, verifique antes de salvar se a preferência já existe; dê um erro caso contrário.

Passo 6 (desafio): Implemente o tratador do botão “Apagar”: apague a preferência caso exista; dê um erro caso contrário.

Use o método `editor.remove()`.

Passo 7: Rode e teste a aplicação. Perceba que as preferências podem ser recuperadas depois que a aplicação é finalizada e reiniciada.

Parte 2 (Opcional/Desafio): Salvando dados no armazenamento interno

Passo 1: Modifique o layout para conter um checkbox que indica se é para salvar como preferência ou como um arquivo interno.

> Caso seja no armazenamento interno, campo “Nome” será o nome do arquivo e a nota será o valor a ser escrito.

Passo 2: Modifique o método `buttonInsertClick()` de forma que, se o checkbox estiver marcado, salve em um arquivo. Use o trecho de código abaixo para isso:

```
DataOutputStream outputStream;
try {
    outputStream = new DataOutputStream(openFileOutput(name, Context.MODE_PRIVATE));
    outputStream.writeUTF(grade);
    outputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

> Esse código estende o código anterior para suportar gravação em um arquivo no armazenamento interno identificado pelo valor no campo “Nome” (variável Name). A API usada para acessar o arquivo é `opeFileOutput()`, que retorna um `FileOutputStream`, porém é possível fazer o mesmo através de um objeto `File`.

Passo 3: Modifique o método `buttonQueryClick()` adicionando o seguinte código para o caso checkbox de salvar em arquivo estiver marcado:

```
DataInputStream inputStream;
try {
    inputStream = new DataInputStream(openFileInput(name));
    grade = inputStream.readUTF();
    inputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

> Esse trecho de código usa um `FileInputStream` para realizar a leitura do arquivo. Caso a tentativa de ler o arquivo falhe, a string “ERROR” é retornada como resultado.

Passo 4: Modifique o tratador do botão “Atualizar”: só modifique caso o arquivo já exista; dê uma mensagem de erro caso contrário.

Passo 5: Modifique o tratador do botão “Apagar”: apague o arquivo caso exista; do contrário dê uma mensagem de erro.

> Use `deleteFile("filename")`

Passo 6: Rode e teste a aplicação. Experimento apagar os dados da aplicação (nas configurações do dispositivo) e veja o que acontece.

Parte 3: Usando Banco de Dados

Alerta: essa parte da prática desfaz as partes anteriores, faça um backup.

Passo 1: Crie uma classe `SchoolContract` no pacote “db”, contendo o código abaixo:

```
public final class SchoolContract {

    public static final String DATABASE_NAME = "School.db";
    public static final int DATABASE_VERSION = 1;

    public static final String SQL_CREATE_STUDENT =
            "CREATE TABLE " + Student.TABLE_NAME + " (" +
                    Student._ID + " INTEGER PRIMARY KEY, " +
                    Student.COLUMN_NAME_STUDENT_NAME + " TEXT, " +
                    Student.COLUMN_NAME_STUDENT_GRADE + " INTEGER " + ")";

    public static final String SQL_DELETE_STUDENT = "" +
            "DROP TABLE IF EXISTS " + Student.TABLE_NAME;

    public static abstract class Student implements BaseColumns {
        public static final String TABLE_NAME = "Student";
        public static final String COLUMN_NAME_STUDENT_NAME = "Name";
        public static final String COLUMN_NAME_STUDENT_GRADE = "Grade";
    }

    private SchoolContract() {}
}
```

Essa classe contém o esquema do BD. Não é obrigatória, mas facilita o acesso a nomes de colunas e outras informações do banco. Nessa classe temos uma subclasse estática interna pra cada tabela do banco (só uma no nosso caso).

Passo 2: Crie a classe `SchoolDbHelper` no pacote “db”, contendo o código abaixo:

```
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import static br.edu.ifpe.tads.pdm.pratica04.db.SchoolContract.*;

public class SchoolDbHelper extends SQLiteOpenHelper {

    public SchoolDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_STUDENT);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL(SQL_DELETE_STUDENT);
        this.onCreate(db);
    }
}
```

Essa classe trata os eventos de criação do banco (quando não existe) e atualização (nova versão da App). Na atualização, o banco é recriado do zero no código acima (o banco local é um cache para dados da nuvem).

Passo 3: Crie o tratador `buttonInsertClick()` para o botão “Inserir” com o código:

```
public void buttonInsertClick(View view) {
        String name =
             ((EditText)findViewById(R.id.edit_name)).getText().toString();
        int grade = Integer.parseInt(
             ((EditText) findViewById(R.id.edit_grade)).getText().toString());

        SchoolDbHelper dbHelper = new SchoolDbHelper(this);
        SQLiteDatabase db = dbHelper.getWritableDatabase();

        ContentValues values = new ContentValues();

        values.put(COLUMN_NAME_STUDENT_NAME, name);
        values.put(COLUMN_NAME_STUDENT_GRADE, grade);

        long newId = db.insert(TABLE_NAME, null, values);

        Toast toast = Toast.makeText(this,
              "Registro adicionado. ID = " + newId, Toast.LENGTH_SHORT);
        toast.show();

}
```

Esse código cria um `dbHelper`, que é usado para obter acesso de escrita ao banco de dados (objeto `db`). Em seguida é criado o objeto `values` que contém os valores a serem inseridos (nome e nota) via `db.insert()`. Esse método retorna o ID do novo registro, que é exibido na tela com um `Toast`. É necessário importar as constantes da classe de contrato usando:

```
import static br.edu.ifpe.tads.pdm.pratica04.db.SchoolContract.Student.*
```

Passo 2: Crie o tratador `buttonUpdateClick()` para o botão “Atualizar” com o código:

```
public void buttonUpdateClick(View view) {
        String name =
             ((EditText)findViewById(R.id.edit_name)).getText().toString();
        int grade = Integer.parseInt(
             ((EditText) findViewById(R.id.edit_grade)).getText().toString());

        SchoolDbHelper dbHelper = new SchoolDbHelper(this);
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        ContentValues values = new ContentValues();
        values.put(COLUMN_NAME_STUDENT_GRADE, grade);

        String selection = COLUMN_NAME_STUDENT_NAME + " LIKE ?";
        String[] selectionArgs = { name + "" };

        int count = db.update(TABLE_NAME, values, selection, selectionArgs);

        Toast toast = Toast.makeText(this,
              "Registros atualizados: " + count, Toast.LENGTH_SHORT);
        toast.show();
}
```

O código de atualização é semelhante ao anterior, porém contém uma seleção (`selection`), usada para determinar os registros atualizados. As interrogações (?) em `selection` são substituídas por valores de `selectionArgs` na ordem de aparição. O objeto `values` contém os campos e valores a serem atualizados. O retorno de `db.update()` é o número de registros afetados.

Passo 3: Crie o tratador `buttonDeleteClick()` para o botão “Deletar” com o código:

```
public void buttonDeleteClick(View view) {
        String name =
               ((EditText)findViewById(R.id.edit_name)).getText().toString();

        SchoolDbHelper dbHelper = new SchoolDbHelper(this);
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        String selection = COLUMN_NAME_STUDENT_NAME + " LIKE ?";
        String[] selectionArgs = { name + "" };

        int count = db.delete(TABLE_NAME, selection, selectionArgs);

        Toast toast = Toast.makeText(this,
             "Registros deletados: " + count, Toast.LENGTH_SHORT);
        toast.show();
}
```

O código de remoção de registros trabalha de forma parecida com o de atualização, porém não há valores a serem fornecidos (i.e., não há objeto `values`).

Parte 4: Consultado os dados

Passo 1: Crie uma nova atividade em branco com as seguintes propriedades (propriedades não listadas, aceite o default do Android Studio):

Activity Name: QueryResultActivity

Parent Activity (Manifest): MainActivity

Passo 2: Altere o arquivo de layout da atividade `QueryResultActivity` para que o layout raiz seja um `LinearLayout` com as seguintes propriedades extras:

android:id=”@+id/query_result”

android:orientation=”vertical”

(Remova o `TextView` adicionado por default pelo Android Studio, se houver.)

Passo 3: Modifique o tratador `buttonQueryClick()` para o botão “Consultar” em **MainActivity.java** com o código abaixo.

Esse código realiza uma consulta por nome na tabela Student, retornando os campos ID, Nome (Name) e Nota (Grade) (o array `projection` diz os campos retornados). Assim como para atualizar e apagar, são usados `selection` e `selectionArgs`. O resultado da consulta é colocado num objeto do tipo `Cursor`, que é usado para navegar pelos resultados e gerar o `ArrayList data`, que contém strings do tipo “ID: nome = nota”. O objeto `data` é passado via `Intent` para a atividade `QueryResultActivity`.

```
public void buttonQueryClick(View view) {
        String name =
               ((EditText)findViewById(R.id.edit_name)).getText().toString();

        SchoolDbHelper dbHelper = new SchoolDbHelper(this);
        SQLiteDatabase db = dbHelper.getReadableDatabase();

        String [] projection = {_ID,
                                COLUMN_NAME_STUDENT_NAME,
                                COLUMN_NAME_STUDENT_GRADE};
        String selection = COLUMN_NAME_STUDENT_NAME + " LIKE ?";
        String[] selectionArgs = { name + "%" };
        String sortOrder = COLUMN_NAME_STUDENT_GRADE + " DESC";

        Cursor c = db.query(TABLE_NAME,
                            projection,
                            selection,
                            selectionArgs,
                            null, null,
                            sortOrder);

        ArrayList<CharSequence> data = new ArrayList<CharSequence>();

        c.moveToFirst();
        while (!c.isAfterLast()) {
            String entry = c.getInt(c.getColumnIndex(_ID)) + ": ";
            entry += c.getString(
                        c.getColumnIndex(COLUMN_NAME_STUDENT_NAME)) + " = ";
            entry += c.getInt(
                        c.getColumnIndex(COLUMN_NAME_STUDENT_GRADE));
            data.add(entry);
            c.moveToNext();
        }

        Intent intent = new Intent(this, QueryResultActivity.class);
        intent.putCharSequenceArrayListExtra("data", data);
        startActivity(intent);
}
```

Passo 4: **Complemente** o código do método `onCreate()` em `QueryResultActivity` com o trecho abaixo:

```
protected void onCreate(Bundle savedInstanceState) {

        ...

        LinearLayout layout = (LinearLayout) findViewById(R.id.query_result);

        Intent intent = getIntent();
        ArrayList<CharSequence> data =
                   intent.getCharSequenceArrayListExtra("data");

        for (CharSequence entry : data) {
            TextView text = new TextView(this);
            text.setText(entry);
            text.setLayoutParams(new LinearLayout.LayoutParams(
                    LinearLayout.LayoutParams.MATCH_PARENT,
                    LinearLayout.LayoutParams.WRAP_CONTENT));

            layout.addView(text);
        }
}
```

Esse código preenche o layout da atividade com os resultados da consulta, que foram passados à atividade via `Intent` no `ArrayList data`.

Passo 5: No manifesto, configure a `MainActivity` como sendo “SingleTop”, usando trecho de código abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.edu.ifpe.tads.pdm.pratica04">

    <application
        ...
        <activity android:name=".MainActivity" android:launchMode="singleTop">
            ...

        </activity>
        ...
    </application>

</manifest>
```

Com esse configuração, a `MainActivity` é lançada apenas uma vez e fica no topo da pilha de atividades. Caso não seja usada, ao retornar da `QueryResultActivity` para a `MainActivity`, ela seria recriada do zero, apagando os valores presentes no EditTexts.

Parte 5 (Desafio): Aprimorando a arquitetura

Passo 1: Crie uma classe Student contendo o nome (name) e nota (grade) do estudante como atributos. Coloque essa classe em um pacote adequado (model ou domain, por exemplo).

Passo 2: Utilize o padrão DAO para isolar a persistência dos dados do modelo e controle. Comece criando uma interface similar à mostrada a seguir:

```
public interface StudentDAO {
    long insert(Student student);     // retorna ID do novo registro
    int update(Student student);      // retorna numero de regs. modificados
    int delete(String name);          // retorna numero de registros apagados
    List<Student> list(String name);  // retorna lista de estudantes
}
```

Passo 3: Implemente a interface StudentDAO (por exemplo, StudentSQLiteDAO), copiando o código de manipulação do banco que está em MainActivity para os métodos apropriados na nova classe.

Passo 4: Modifique os métodos de MainActivity para utilizar as funções correspondentes da implementação de StudentDAO.

Dica: instancie o DAO apenas uma vez na criação da atividade, passando o contexto como parâmetro, e reutilize nos tratadores de evento de clique dos botões

Aviso: não misture código do DAO com tratamento da UI. Toda manipulação de elementos visuais continua em MainActivity, ficando no DAO apenas tratamento do BD.