





Tópicos Abordados



- Preferências
 - Lendo e gravando preferências do usuário
 - O framework de preferências do Android
- Arquivos na memória interna
 - Leitura e escrita de arquivos
 - Trabalhando com raw files
- Arquivos na memória externa
 - Leitura e escrita de arquivos
 - Diretórios compartilhados
- Bancos de Dados
 - Criando e atualizando bancos de dados
 - Manipulando e consultando dados

Preferências

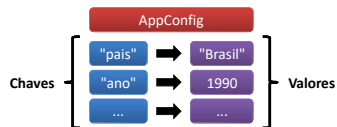


- Grande parte das aplicações têm um mecanismo para armazenar as preferências
 - Configurações feitas pelo usuário
- O objetivo das preferências é manter determinadas informações, mesmo que a aplicação seja finalizada
- As preferências no Android podem ser armazenadas facilmente usando a classe **SharedPreferences**

A Classe SharedPreferences



- Armazena as preferências em arquivos
- Por padrão, são compartilhadas entre os componentes da aplicação, mas não são visíveis para outras aplicações
- As preferências têm a forma de pares de chave e valor



Lendo Preferências



- Leitura das preferências *AppConfig*

```
SharedPreferences prefs =  
    getSharedPreferences("AppConfig", Context.MODE_PRIVATE);  
  
String nome = prefs.getString("pais", "Nenhum");  
int ano = prefs.getInt("ano", 0);
```

Acessado pelo contexto

Acesso aos dados

Salvando Preferências



- Gravação das preferências *AppConfig*

```
SharedPreferences prefs =  
    getSharedPreferences("AppConfig", Context.MODE_PRIVATE);  
  
SharedPreferences.Editor editor = prefs.edit();  
editor.putString("pais", "Brasil");  
editor.putInt("ano", 1990);  
editor.commit();
```

Confirma as alterações

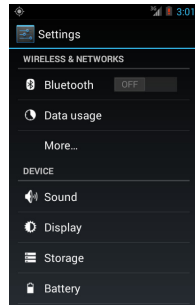
Abre o modo de edição e
coloca os dados

O método *apply()* tem o mesmo efeito,
mas executa em segundo plano

Preferences Framework



- O Android disponibiliza um framework para montar telas de preferência para aplicações
 - Layout baseado em XML
 - Integração com telas de preferências de outras aplicações do Android



Definindo o Layout da Tela



- O layout da tela é definido em XML
- É semelhante à forma de definir um layout para uma activity do Android
- O arquivo deve estar no diretório **/res/xml**

Definindo o Layout da Tela




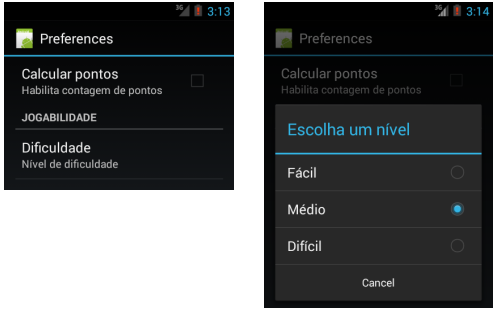
```
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android"
android:title="Configuração">

    <CheckBoxPreference android:key="enable_score"
        android:title="Calcular pontos"
        android:summary="Habilita contagem de pontos" />


    <PreferenceCategory android:title="Jogabilidade">
        <List<resources>
            and    <string-array name="mode_options">
                and    <item>Fácil</item>
                and    <item>Médio</item>
                and    <item>Difícil</item>
            and    </string-array>
            and    <string-array name="mode_options_values">
                <item>0</item>
                <item>1</item>
                <item>2</item>
            </string-array>
        </resources>
    </PreferenceCategory>
</PreferenceScreen>
```

Resultado do Uso do Layout






Controles de Preferência



- O Android possui diversos controles de preferência nativos
 - *CheckBoxPreference*
 - Checkbox que pode ser marcado ou não
 - *ListPreference*
 - Elementos em uma lista onde um pode ser escolhido
 - *EditTextPreference*
 - Caixa de texto para digitar uma informação
 - *RingtonePreference*
 - Lista dos toques do dispositivo para seleção

A Classe *PreferenceFragmentCompat*




- Para usar o framework de preferências, deve haver um fragment que mostra a tela de preferências configurada pelo XML
- Este fragment deve estender de **PreferenceFragmentCompat**

```

public class SettingsFragment extends PreferenceFragmentCompat {
    public void onCreatePreferences(Bundle bundle, String s) {
        setPreferencesFromResource(R.xml.config, s);
    }
}

```



Define o arquivo de layout

A Classe *PreferenceFragmentCompat*



- Nesta classe é possível também acessar via programação os elementos que compõem o layout da tela

```
CheckBoxPreference cp =  
    (CheckBoxPreference) findPreference("enable_score");  
boolean checked = cp.isChecked();  
  
ListPreference lp =  
    (ListPreference) findPreference("mode");  
String value = lp.getValue();
```

Obtém o elemento a partir da chave

Acessando as Preferências



- Depois que as preferências são escolhidas pelo usuário, elas podem ser recuperadas

```
SharedPreferences prefs =  
    PreferenceManager.getDefaultSharedPreferences(getActivity());  
boolean enableScore = prefs.getBoolean("enable_score", false);  
String mode = prefs.getString("mode", "0");
```

Contexto


Acessa as preferências pela chave

Monitorando Preferências



- Às vezes é necessário que sua aplicação seja notificada quando ocorre uma alteração no valor de uma preferência
- Classes que implementam a interface **OnSharedPreferenceChangeListener** podem ser notificadas quando isto ocorre

Monitorando Preferências



```

public class SettingsFragment extends PreferenceFragment
implements OnSharedPreferenceChangeListener {

    public void onSharedPreferenceChanged(
        SharedPreferences prefs, String key) {
        //...
    }
}

```

O que foi alterado


```

SharedPreferences prefs =
    PreferenceManager.getDefaultSharedPreferences(getActivity());
prefs.registerOnSharedPreferenceChangeListener(this);

```


Registra o objeto a ser notificado

Trabalhando com Arquivos



- Outra forma de salvar dados no dispositivo é através da manipulação direta de arquivos
- Os arquivos, por padrão, são visíveis apenas para as aplicações que os criam
- Métodos
 - **openFileInput()**: abre arquivo para leitura
 - **openFileOutput()**: abre arquivo para escrita
- Os métodos retornam objetos do tipo *InputStream* e *OutputStream*

Trabalhando com Arquivos



```

FileInputStream in = openFileInput("config.txt");

```

Abre o arquivo para leitura

```

FileOutputStream out =
    openFileOutput("config.txt", Context.MODE_APPEND);

```

Abre o arquivo para escrita

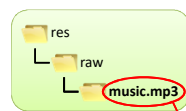
Append no arquivo

Raw Files



- Os raw files são arquivos de recursos que não são compilados pelo Android
 - São empacotados exatamente como são
- Estes arquivos ficam em **/res/raw**
- São considerados arquivos apenas de leitura
- Exemplos
 - Arquivos de vídeo e som
 - Arquivos texto e binários
 - etc.

Raw Files



```
Resources res = getResources();  
InputStream in = res.openRawResource(R.raw.music);
```


Manipulação do arquivo
como for necessário

Armazenamento Externo




- Até agora todos os mecanismos para trabalhar com dados usam a memória interna do dispositivo
- O Android possibilita que arquivos sejam salvos em uma memória externa
 - SD Card
 - Memória externa não-removível
- Arquivos da memória externa não têm restrições de segurança

Verificando a Disponibilidade



- Como a memória externa pode ser removida, é preciso checar o estado da mesma antes de usá-la


```
String status = Environment.getExternalStorageState();
```



Environment.MEDIA_MOUNTED
Environment.MEDIA_MOUNTED_READ_ONLY

Os demais status indicam que o acesso não é possível

Acessando os Diretórios




- O acesso aos diretórios para gravação é feito através do método **getExternalFilesDir()**

```
File f = getExternalFilesDir(Environment.DIRECTORY_MUSIC);
```

Define o tipo de diretório a ser retornado

Diretórios Compartilhados



- Às vezes é necessário armazenar arquivo numa área compartilhada da memória externa (não limitada à aplicação)
 - Músicas, imagens, etc.
- Chamar **getExternalStoragePublicDirectory()**

```
File f = Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_RINGTONES);
```

- É preciso permissão para acessar a área de armazenamento externo

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

Bancos de Dados



- O Android traz suporte nativo ao banco de dados SQLite
 - Página oficial: <http://www.sqlite.org>
 - Open-source
- Por padrão, os bancos de dados podem ser acessados apenas pelas aplicações que os criaram
- Embora não seja obrigatório, é recomendado que cada tabela tenha um campo de auto-incremento que representa um ID único

A Classe SQLiteOpenHelper



- Encapsula o processo de criação e atualização do banco de dados no dispositivo
 - É recomendável que a própria aplicação execute os scripts de criação e atualização das tabelas do banco de dados
- É preciso estender esta classe e implementar os métodos **onCreate()** e **onUpgrade()**
 - O SQLite usa números de versão para controlar se um banco de dados deve ser atualizado

A Classe SQLiteOpenHelper




```
public class DBHelper extends SQLiteOpenHelper {  
  
    public DBHelper(Context context, String name, int version) {  
        super(context, name, null, version);  
    }  
  
    public void onCreate(SQLiteDatabase db) {  
        //cria o banco de dados  
    }  
  
    public void onUpgrade(SQLiteDatabase db,  
        int oldVersion, int newVersion) {  
        //atualiza o banco de dados  
    }  
}
```

```
DBHelper helper = new DBHelper(this, "cursodb", 1);  
SQLiteDatabase db = helper.getWritableDatabase();  
//...  
db.close();
```

Retorna a referência
ao banco de dados

Manipulando Dados



- A inserção, atualização e exclusão de dados é feita através dos métodos **insert()**, **update()** e **delete()**, respectivamente


```
ContentValues values = new ContentValues();
values.put("nome", "José Silva");
values.put("idade", 30);

db.insert("pessoa", null, values);

db.update("pessoa", values, "_id = 1", null);

db.delete("pessoa", "_id = 1", null);
```

Lendo Dados



- A leitura é feita através do método **query()**

```
Cursor c =
db.query("pessoa", null, null, null, null, null, null);

if (c.moveToFirst()) {
do {
int id = c.getInt(0);
nome = c.getString(1);
int idade = c.getInt(2);
} while (c.moveToNext());
}

c.close();
```

Obtém os dados

Itera sobre os resultados

Fecha o cursor