



Android

Tópicos

Avançados



Persistência

Parte 1

Persistência

- ◉ Existem 3 formas de armazenar dados no dispositivo
 - ◉ Arquivos comuns (texto ou binário)
 - ◉ Classes FileInputStream e FileOutputStream
 - ◉ Armazenados em /data/data/<pacote>/files
 - ◉ Arquivos de preferências
 - ◉ Classe SharedPreferences
 - ◉ Bancos de Dados usando SQLite
 - ◉ Classes SQLiteDatabase, SQLiteOpenHelper
 - ◉ Armazenados em /data/data/<pacote>/databases

Arquivos comuns

- Métodos da classe Context

FileInputStream openFileInput(String name)	Abre um arquivo privado associado a aplicação para leitura. Argumento name não pode conter '/'.
FileOutputStream openFileOutput(String name, int mode)	Abre um arquivo privado associado a aplicação para escrita. Argumento mode pode ser Context.MODE_PRIVATE ou Context.MODE_APPEND

- **FileInputStream** contém métodos para **ler bytes**
- **FileOutputStream** contém métodos para **escrever bytes**
- **InputStreamReader** pode ser construído a partir de um **FileInputStream** e contém métodos para **ler caracteres**
- **OutputStreamWriter** pode ser construído a partir de um **FileOuputStream** e contém métodos para **escrever caracteres**

Arquivos comuns

- ◉ Subdividem em 2 tipos de armazenamento
 - ◉ Armazenamento Interno
 - ◉ Armazenamento Externo
- ◉ Interno e Externo??? Como Assim???



Armazenamento

- Geralmente o Android é dividido em 3 partições. Vamos usar de exemplo um smartphone de 32GB
- **System | ext4 | 591MB**: Partição onde o SO é armazenado
- **data | ext4 | 1.5GB**: Partição de Apps e alguns dados
- **storage/sdcard0 | vfat | 27GB**: Emula um sdcard. Usado para todo tipo de armazenamento.

Armazenamento Interno

- É possível persistir dados na memória interna do dispositivo Android.
 - Por padrão, arquivos salvos na memória interna são privados da aplicação e outras aplicações não tem acesso ao mesmo
 - Caso o usuário desinstale o app, os arquivos serão removidos
- Para criar e escrever em um arquivo interno deve-se utilizar o método **openFileOutput()** que retornará um objeto **FileOutputStream()**
 - Para escrever, usa-se o método **write()** e deve-se fechar o stream com o método **close()**

Armazenamento Interno

- Para abrir e ler um arquivo interno deve-se utilizar o método **openFileInput()** que retornará um **FileInputStream()**
- Modos de Criação
- Para ler, usa-se o método **read()** e deve-se fechar o stream com o método **close()**

Modo	Descrição
MODE_PRIVATE (Padrão)	Arquivos criados somente são acessados pela aplicação que os criou ou por outras aplicações que compartilhem o mesmo ID de usuário
MODE_APPEND	Se o arquivo já existe, escreve dados no fim do arquivo ao invés de apagá-lo para criar um novo

Exemplo - Armazenamento Interno

- Vamos partir do pressuposto que queremos salvar o objeto Contato, descrito abaixo, na memória interna:

```
public class Contato implements Serializable {  
  
    private static final String FILE_NAME = "contatos.data";  
  
    private String nome;  
    private String telefone;  
  
    public String getNome() { return nome; }  
  
    public void setNome(String nome) { this.nome = nome; }  
  
    public String getTelefone() { return telefone; }  
  
    public void setTelefone(String telefone) { this.telefone = telefone; }  
}
```

Exemplo - Armazenamento Interno

```
public static boolean saveObject(ArrayList<Contato> obj, Context context)
{
    FileOutputStream fos = null;
    ObjectOutputStream oos = null;
    boolean keep = true;

    try {

        fos = context.openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
        oos = new ObjectOutputStream(fos);
        oos.writeObject(obj);
        oos.flush();

        oos.close();
        fos.close();
    }
    catch (Exception e) {
        keep = false;
    }

    return keep;
}
```

Exemplo - Armazenamento Interno

```
public static ArrayList<Contato> getObject(Context context)
{
    ArrayList<Contato> contatos = null;
    FileInputStream fis = null;
    ObjectInputStream is = null;

    try {

        fis = context.openFileInput(FILE_NAME);
        is = new ObjectInputStream(fis);
        contatos = (ArrayList<Contato>) is.readObject();

        fis.close();
        is.close();

    } catch (Exception e) {
        contatos = null;
    }

    return contatos;
}
```

Armazenamento Externo

- Todo dispositivo Android pode ter um armazenamento externo no qual é possível salvar arquivos – p. ex. um cartão de memória
- Arquivos persistidos no armazenamento externo são acessíveis por todos os aplicativos e podem ser modificados e/ou removidos pelo usuário
- Acesso é semelhante ao armazenamento interno

Armazenamento Externo

- Para recuperar a lista de diretórios de seus arquivos externos, usa-se o método **getExternalFilesDir()**
- Além deste, sua aplicação pode também persistir arquivos em diretórios públicos, estes não serão removidos na desinstalação da App e estão na raiz cartão SD

Music/ - Media scanner classifies all media found here as user music.

Podcasts/ - Media scanner classifies all media found here as a podcast.

Ringtones/ - Media scanner classifies all media found here as a ringtone.

Alarms/ - Media scanner classifies all media found here as an alarm sound.

Notifications/ - Media scanner classifies all media found here as a notification sound.

Pictures/ - All photos (excluding those taken with the camera).

Movies/ - All movies (excluding those taken with the camcorder).

Download/ - Miscellaneous downloads.

Armazenamento Externo

- As pastas públicas podem ser descobertas usando o método **getExternalStoragePublicDirectory()**
- Para fazer cache de dados utiliza-se o método **getExternalCacheDir()** que representa o diretório para arquivos temporários da mídia externa
- Vale lembrar que para trabalhar com arquivos externos é preciso colocar permissão no Manifest

```
<uses-permission
```

```
  android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Exemplo - Armazenamento Externo

```
public static boolean saveExternalStorage(List<Contato> list, Context context) {  
  
    boolean saved = true;  
    File file;  
    ObjectOutputStream oos = null;  
  
    try {  
        file = new File(context.getExternalCacheDir(), FILE_NAME);  
        FileOutputStream fos = new FileOutputStream(file);  
        oos = new ObjectOutputStream(fos);  
        oos.writeObject(list);  
        oos.flush();  
  
        oos.close();  
        fos.close();  
    } catch (Exception e) {  
        saved = false;  
    }  
  
    return saved;  
}
```

Exemplo - Armazenamento Externo

```
public static List<Contato> getExternalStorage(Context context){  
  
    List<Contato> contatoList = null;  
    ObjectInputStream ois = null;  
  
    try {  
        File file = new File(context.getExternalCacheDir(), FILE_NAME);  
        FileInputStream fis = new FileInputStream(file);  
        ois = new ObjectInputStream(fis);  
        contatoList = (List<Contato>) ois.readObject();  
  
        ois.close();  
        fis.close();  
    } catch (Exception e) {}  
  
    return contatoList;  
}
```


Exercício

- Vamos construir o exercício de cadastro de contatos salvando os dados em um arquivo.

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent" android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView android:id="@+id/txLabel"
        android:text="Informe os Dados:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <EditText
        android:id="@+id/editNome"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txLabel"
        android:hint="Nome" />
    <EditText
        android:id="@+id/editFone"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editNome"
        android:layout_toLeftOf="@+id/btSave"
        android:inputType="phone"
        android:hint="Telefone" />
    <Button
        android:id="@+id/btSave"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editNome"
        android:layout_alignRight="@+id/editNome"
        android:onClick="salvarContato"
        android:text="Salvar" />
    <View
        android:id="@+id/divisor"
        android:layout_width="match_parent"
        android:layout_height="5dp"
        android:layout_below="@+id/btSave"
        android:background="#000" />
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/divisor" />
</RelativeLayout>

```

5:00

Informe os Dados:

Nome

Telefone

SALVAR

Item 1
Sub Item 1

Item 2
Sub Item 2

Item 3
Sub Item 3

Item 4
Sub Item 4

Item 5
Sub Item 5

Item 6
Sub Item 6

Item 7
Sub Item 7

```
public class MainActivity extends ActionBarActivity {

    ListView listView;

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

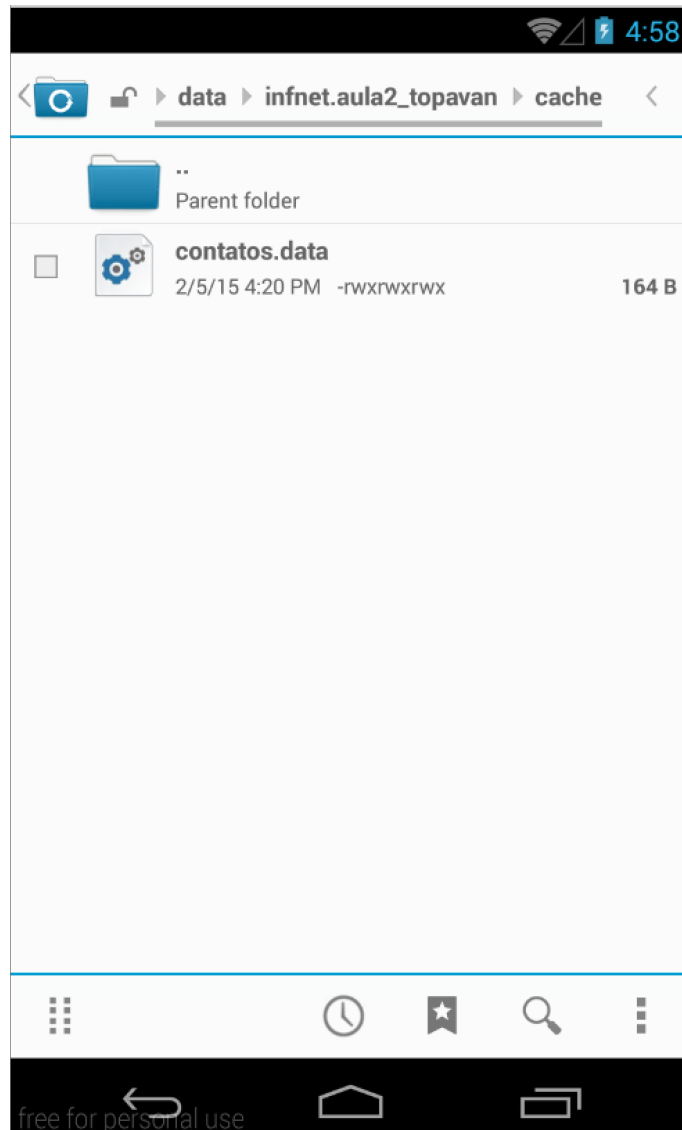
        listView = (ListView) findViewById(R.id.listView);
        List<Contato> list = Contato.getInternalStorage(this);
        if (list != null){
            ArrayAdapter<Contato> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, list);
            listView.setAdapter(adapter);
        }
    }

    public void salvarContato(View view) {
        EditText editNome = (EditText) findViewById(R.id.editNome);
        EditText editFone = (EditText) findViewById(R.id.editFone);

        Contato contato = new Contato();
        contato.setNome(editNome.getText().toString());
        contato.setTelefone(editFone.getText().toString());

        List<Contato> list = Contato.getInternalStorage(this);
        if (list == null){
            list = new ArrayList<>();
        }

        list.add(contato);
        if (Contato.saveInternalStorage(list, this)){
            ArrayAdapter<Contato> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, list);
            listView.setAdapter(adapter);
        }
    }
}
```



SharedPreferences

- A API de **SharedPreferences** oferece ao desenvolvedor uma maneira muito simples e ágil para salvar e recuperar pares chave-valor de tipos primitivos
- Serve para armazenar preferências ou configurações da aplicação.
- É possível usar quantas quiser, desde que associadas a um nome

SharedPreferences

- **DICA:** Certifique-se de utilizar um nome único para identificar a SharedPreferences, o recomendado é o nome do pacote de sua aplicação e adicionar um nome para identificar o arquivo. Por exemplo:

"com.example.myapp.PREFERENCE_KEY_NAME"

SharedPreferences

- Para obter a instância desta classe associada ao nome de arquivo “preferencias” faça

SharedPreferences pref =

Context.getSharedPreferences(“preferencias”, 0);

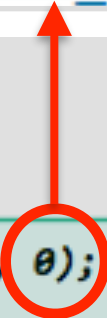
Se o arquivo ainda não existir será criado

- Para editar use a classe **SharedPreferences.Editor**

SharedPreferences.Editor editor = prefs.edit();

SharedPreferences

MODE_PRIVATE
MODE_WORLD_READABLE
MODE_WORLD_WRITEABLE



```
et = (EditText) findViewById(R.id.editText1);
cb = (CheckBox) findViewById(R.id.checkBox1);

// Obtendo objeto de preferências
SharedPreferences prefs = getSharedPreferences(ARQ_PREFS, 0);
//
et.setText(prefs.getString(PREF_STR, ""));
cb.setChecked(prefs.getBoolean(PREF_BOOL, false));
```

```
// Obtendo objeto de preferências
SharedPreferences prefs = getSharedPreferences(ARQ_PREFS, 0);
SharedPreferences.Editor editor = prefs.edit();
editor.putString(PREF_STR, et.getText().toString());
editor.putBoolean(PREF_BOOL, cb.isChecked());
editor.commit();
```


ShardPreferences

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)

String getString(String key, String)

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`.

SharedPreferences.Edit edit()

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

SharedPreferences.Editor

boolean commit()

Salva as preferências de forma síncrona, ou seja, só retorna após ter salvo em disco retornando true em caso de sucesso, ou false senão.

void apply()

Semelhante ao anterior porém o armazenamento em disco será feito de forma assíncrona, ou seja, não é possível saber se houve algum error durante o armazenamento.

clear()

Remove todos os valores de preferências do objeto.

putBoolean(String key, boolean)

putFloat(String key, float)

putInt(String key, int)

putLong(String key, long)

putString(String key, String)

Adiciona ou altera o valor de uma chave. O nome do método e o segundo parâmetro indicam o tipo do valor da chave.

remove(String key)

Remove a chave definida por key e consequentemente o seu valor.

Exercício

- Vamos construir o exercício simples com SharedPreferences um utilizando Login de Usuários

Exemplo de utilização

- Guardar username, password ou token de usuário
- Guardar o timestamp ou quantos vezes acessou um recurso
- Muito usado em booleanos para ligar/desligar configurações

Libs para SharedPreferences

- <https://github.com/Pixplicity/EasyPreferences>
- <https://github.com/fsilvestremorais/android-complex-preferences>