Desenvolvimento Móvel em Android

Persistência de Dados

#Serialização #Armazenamento Interno

Sónia Luz, sonia.luz@ipleiria.pt
David Safadinho, david.safadinho@ipleiria.pt
Departamento de Engenharia Informática
Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Leiria
1º Semestre - 2021/2022

Persistência de Dados

- A persistência de dados é essencial para qualquer aplicação
  - Permitindo manipular e persistir a informação apresentada;
- O Android oferece várias opções para guardar os dados de aplicações de forma persistente;
- A solução escolhida depende das necessidades específicas:
  - Se os dados devem ser privados para a aplicação;
  - Se devem estar acessíveis para outras aplicações e/ou utilizadores;
  - Quanto espaço é necessário para os dados.

Persistência de Dados

- A opções de armazenamento de dados no Android são:
  - Preferências Partilhadas (*SharedPreferences*)
    - Armazenar dados primitivos privados em pares *chave-valor*;
  - Armazenamento Interno
    - Armazenar dados privados na memória do dispositivo;
  - Armazenamento Externo
    - Armazenar dados públicos no armazenamento externo partilhado;
  - Bases de Dados SQLite
    - Armazenar dados estruturados numa base de dados privada;
  - Ligação de Rede
    - Armazenar dados na web com o seu próprio servidor de rede.

Serialização

- Se <u>não</u> existir a classe *Singleton*
  - Para que os dados possam, de algum modo ser armazenados;
  - Mesmo que apenas para serem passados entre atividades
    - Pesquisa que deve devolver resultados;
    - Adição de um novo dado;
    - Atualização de dados;
  - Têm de ser transformados em objetos **Serializáveis** (***Serializable***)
- Ou se quisermos armazenar dados em ficheiros internos

Serialização

- **Serialização** (***Serialization***) é o processo de:
  - Traduzir o estado de um objeto para um formato que possa ser armazenado:
    - Num ficheiro;
    - Enviado através de uma ligação de rede;
    - Passado entre atividades Android;
  - E ser reconstruído (desserializado) mais tarde
    - No mesmo ou noutro contexto;

Serialização

- A linguagem Java fornece serialização automática
  - Mas requer que cada objeto seja marcado como **Serializável**
    - Implementando a interface **`java.io.Serializable`**;
  - Implementar esta interface marca a classe como “ok para serializar”;
  - Após isso o Java lida com a serialização internamente;

Serialização - Exemplo

- Considerando a nossa aplicação de contactos
  - Definir a **ListViewContactos** como atividade principal;
  - Implementar a funcionalidade “Adicionar Contacto”;
  - Mas primeiro precisamos marcar as classes **Contacto** e **GestorContactos** como serializáveis;

```
import java.io.Serializable;

public class Contacto implements Serializable{
...
```

```
import java.io.Serializable;

public class GestorContactos implements Serializable {
...
```

Serialização - Exemplo

- Vamos criar uma nova atividade **AdicionarContacto**
  - Que deve devolver o contacto criado para a **ListViewContactos**;
  - Para que seja adicionado aos seus **contactos**;
  - Para isso é necessário que:
    - A **ListViewContactos** passe a iniciar a atividade **AdicionarContacto** com o propósito de receber o resultado;
    - A atividade **AdicionarContacto** seja implementada para devolver um resultado (o novo contacto);
    - A **ListViewContactos** deve lidar com o contacto devolvido
      - Adicionando-o aos contactos;
      - E atualizando a ***listview*** apresentada;

Serialização - Exemplo

- Na **ListViewContactos** vamos adicionar uma constante

```
public class ListViewContactosActivity extends AppCompatActivity {
    public static final int REQUEST_CODE_ADICIONAR_CONTACTO = 1;
```

  - Que servirá de código de verificação para avaliar o resultado da atividade;

- Vamos iniciar a atividade **AdicionarContacto**
  - Através do FAB da **ListViewContactos**

```
public void onClickAdicionarContacto(View view) {
    Intent intent = new Intent(this, AdicionarContactoActivity.class);
    startActivityForResult(intent, REQUEST_CODE_ADICIONAR_CONTACTO);
}
```

Serialização - Exemplo

- Na atividade **AdicionarContacto**

```
public class AdicionarContactoActivity extends AppCompatActivity {
    public static final String NOVO_CONTACTO = "NOVO_CONTACTO";
    private EditText editTextID;
    private EditText editTextNome;

  ...
    public void onClickFabGuardarContacto(View view) {
        int id = Integer.parseInt(editTextID.getText().toString());
        String nome = editTextNome.getText().toString();

        Contacto novoContacto = new Contacto(id, nome);
        Intent returnIntent = new Intent();
        returnIntent.putExtra(NOVO_CONTACTO, novoContacto);
        setResult(RESULT_OK, returnIntent);
        finish();
    }
}
```

Serialização - Exemplo

- Na classe **GestorContactos**
  - Deve adicionar o método para adicionar um novo contacto à lista de contactos;

```
//adicionar um novo contacto
public void adicionarContacto(Contacto contacto){
    contactos.add(contacto);
}
```

Serialização - Exemplo

- Novamente na **ListViewContactos**
  - Vamos avaliar o resultado devolvido consoante o código do pedido

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
                                Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == REQUEST_CODE_ADICIONAR_CONTACTO){
        if(resultCode == RESULT_OK){
            Contacto contacto = (Contacto) data.getSerializableExtra(
                    AdicionarContactoActivity.NOVO_CONTACTO);
            gestorContactos.adicionarContacto(contacto);
             //notificar o adaptador de alterações nos dados
            adaptador.notifyDataSetChanged();
        }
    }
}
```

Estado de uma Atividade

- No entanto, quando o sistema destrói uma atividade
  - Para recuperar memória;
    - O objeto da atividade é destruído;
    - E o sistema não pode retomar o seu estado intacto;
  - O sistema deve recriar o objeto da atividade se o utilizador aí voltar;
- Mas é possível garantir que informação importante
  - Sobre o estado da atividade seja preservada;
  - Implementado um método adicional de *callback*: **onSaveInstanceState()**;
    - que permite guardar informação sobre o estado da atividade;

Guardar o Estado de uma Atividade

- No nosso exemplo, queremos principalmente guardar os dados dos contactos
  - Ou seja, devemos garantir que o estado do campo **GestorContactos** seja guardado;
- Reimplementando o método **onSaveInstanceState**() da **ListViewContactos**

```
public class ListViewContactosActivity extends AppCompatActivity {
    private static final String ESTADO_GESTOR_CONTACTOS = "ESTADO_GESTOR_CONTACTOS";
...
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putSerializable(ESTADO_GESTOR_CONTACTOS, gestorContactos);
    }
```

Guardar o Estado de uma Atividade

- Após isso, é necessário alterar o método **onCreate**(), para garantir que o objeto **GestorContactos** é recriado
  - No **onCreate**() passamos a verificar se há um estado de instância guardado;
  - Em caso afirmativo:
    - o campo **GestorContactos** é recuperado;
  - Senão:
    - instanciamos o objeto e adicionamos dados falsos, como fazíamos anteriormente;

Guardar o Estado de uma Atividade

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

    if(savedInstanceState == null) {
        this.gestorContactos = new GestorContactos();
        this.gestorContactos.adicionarDadosIniciais();
    }else{
        this.gestorContactos = (GestorContactos)
                savedInstanceState.getSerializable(ESTADO_GESTOR_CONTACTOS);
    }

    listViewContactos = (ListView)findViewById(R.id.listViewContactos);
    atualizarContactos();
    listViewContactos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            Intent intent = new Intent(getApplicationContext(), DetalhesContacto.class);
            intent.putExtra(GESTOR_CONTACTOS, gestorContactos);
            intent.putExtra(DetalhesContacto.POSICAO, position);
            startActivity(intent);
        }
    });
}
```

Persistência de Dados

- Mas tudo isto apenas permite manter o estado da atividade enquanto a aplicação está ativa;
- Se sairmos da aplicação explicitamente e voltarmos a iniciar
  - Perdemos os dados / alterações efetuadas;
- Para guardar de forma permanente os dados do **GestorContactos**
  - Teremos de usar uma das opções de armazenamento disponíveis, fornecidas pelo Android
    - Vamos começar pelo **Armazenamento Interno**

Armazenamento Interno

- Está sempre disponível;
- Todos os ficheiros são privados;
- Não há acesso do utilizador;
- Outras aplicações não podem ler/escrever;
- Quando o utilizador desinstala a aplicação
  - O sistema remove todos esses ficheiros do armazenamento interno;
- Espaço disponível limitado;

Armazenamento Interno

- Para criar e gravar dados num ficheiro privado:
  - Deve usar o método **`openFileOutput()`**
    - Incluindo o nome do ficheiro e o modo operacional;
    - Devolve um objeto do tipo **`FileOutputStream`**;
  - Deve criar um objeto do tipo **`ObjectOutputStream`**
    - De modo a usar o método **`writeObjet()`**
      - Para gravar os dados do objeto pretendido no ficheiro;
  - Deve usar o método **`close()`**
    - Para fechar os *streams* de acesso ao ficheiro;
- Os dados devem ser gravados no método **`onPause()`**

Armazenamento Interno

```
@Override
protected void onPause() {
    super.onPause();
    try {
        FileOutputStream fileOutputStream =
                openFileOutput("contactos.bin", Context.MODE_PRIVATE);
        ObjectOutputStream objectOutputStream =
                new ObjectOutputStream(fileOutputStream);

        objectOutputStream.writeObject(gestorContactos);

        objectOutputStream.close();
        fileOutputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
        Toast.makeText(this,
            "Não foi possível escrever contactos no armazenamento interno",
                Toast.LENGTH_SHORT).show();
    }
}
```

Armazenamento Interno

- Para ler dados de um ficheiro privado
  - Deve usar o método **`openFileInput()`**
    - Incluindo o nome do ficheiro;
    - Devolve um objeto do tipo **`FileInputStream`**;
  - Deve criar um objeto do tipo **`ObjectInputStream`**
    - De modo a usar o método **`readObjet()`**
      - Para ler os dados para o objeto pretendido;
  - Deve usar o método **`close()`**
    - Para fechar os *streams* de acesso ao ficheiro;

Armazenamento Interno

- Os dados devem ser carregados no método **`onCreate()`**
  - Primeiro devemos verificar se existe um **`savedInstanceState`** para ser recuperado;
  - Em caso negativo, devemos tentar ler os dados que persistiram no armazenamento interno;
    - Revertendo o processo descrito para gravar os dados no disco;

Armazenamento Interno

- Ao carregar os dados no método **`onCreate()`**
  - Se ocorrer algum problema é lançada uma exceção que será tratada nos blocos **`catch`**;
    - Aqui as exceções são tratadas de forma diferente:
      - Se for lançado um **`FileNotFoundException`** a mensagem a apresentar será diferente:
        - Porque pode simplesmente ser a primeira vez que a aplicação foi executada (o que não significa um erro);

Armazenamento Interno

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
     if(savedInstanceState == null) {
         try {
             FileInputStream fileInputStream =
                     openFileInput("contactos.bin");
             ObjectInputStream objectInputStream =
                     new ObjectInputStream(fileInputStream);

             this.gestorContactos =
                    (GestorContactos) objectInputStream.readObject();

             objectInputStream.close();
             fileInputStream.close();

         } catch (FileNotFoundException e) {
     ...
}
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    if(savedInstanceState == null) {
       ...
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(this, "Não foi possível ler contactos do armazenamento interno",
                    Toast.LENGTH_SHORT).show();
        } catch (IOException e) {
            e.printStackTrace();
            Toast.makeText(this, "Erro ao ler Contactos do armazenamento interno",
                    Toast.LENGTH_SHORT).show();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(this, "Erro ao ler Contactos do armazenamento interno",
                    Toast.LENGTH_SHORT).show();
        }
    }else{
        this.gestorContactos = (GestorContactos)
                savedInstanceState.getSerializable(ESTADO_GESTOR_CONTACTOS);
    }
    if(gestorContactos == null) {
        this.gestorContactos = new GestorContactos();
        //this.gestorContactos.adicionarDadosIniciais();
    }
    ...
}
```

Fontes e Mais Informação

- Opções de Armazenamento
  - https://developer.android.com/training/data-storage
- Recriar uma Atividade
  - https://developer.android.com/guide/components/activities/activity-lifecycle
- Armazenamento Interno
  - https://developer.android.com/training/data-storage#filesInternal
  - https://developer.android.com/training/data-storage/app-specific

Próximo Tema:

Padrão de Desenho: Singleton

- Classe *Singleton* em Java
  - https://www.javaworld.com/article/2073352/core-java/simply-singleton.html

Persistência de Dados: SQLite

- Opções de Armazenamento
  - https://developer.android.com/training/data-storage
- Armazenamento em Base de Dados SQLite
  - https://developer.android.com/training/data-storage/sqlite
- Armazenamento em Base de Dados Room
  - https://developer.android.com/training/data-storage/room