Notepad – Exercício 03

Nesse exercício, vamos usar callbacks de eventos do ciclo de vida para guardar e recuperar dados de estado da aplicação. Esse exercício demonstrará:

- Eventos do ciclo de vida e como sua aplicação pode usá-los.
- Técnicas para manter o estado da aplicação.
- Se você não passou pelo exercício 01 e 02, recomendo que o faça. Lá estarão as instruções iniciais de como chegar nessa etapa do projeto Notepad.

Passo 1

Abra o projeto Notepadv3. O ponto inicial desse exercício é exatamente o ponto onde finalizamos o exercício 02.

A aplicação atual tem alguns problemas – se você tocar o botão de voltar quando estiver editando uma nota causará travamento da aplicação e qualquer outra coisa que você faça durante a edição será perdida.

Para corrigir tal problema, nós vamos mover a maioria da funcionalidade para criação e edição de notas para dentro da classe NoteEdit e introduzir o ciclo de vida completo para edição de notas.

1. Remova o código em NoteEdit que faz o parse do title e body dos Bundles extras. Ao invés disso, vamos usar a classe DBHelper para acessar as notas do banco de dados diretamente. Tudo que precisaremos é passar para a atividade NoteEdit o mRowId (mas apenas enquanto editando. Se estivermos inserindo, nós passamos um valor nulo). Portanto, remova essas linhas:

   ```
   String title = extras.getString(NotesDbAdapter.KEY_TITLE);
   String body = extras.getString(NotesDbAdapter.KEY_BODY);
   ```

2. Teremos também de nos livrar das propriedades que foram passadas nos Bundles extras que nós estávamos usando para setar os valores title e body dentro da interface. Então, faça a exclusão também de:

```
if (title != null) {
        mTitleText.setText(title);
}

if (body != null) {
        mBodyText.setText(body);
}
```

Passo 2

Crie um campo, dentro da classe, NotesDbAdapter.

```
private NotesDbAdapter mDbHelper;
```

Também adicione uma instância de NotesDbAdapter no método onCreate() abaixo da chamada para super.onCreate():

```
mDbHelper = new NotesDbAdapter(this);
mDbHelper.open();
```

Passo 3

Dentro de NoteEdit, nós precisamos checar o savedInstanceState com o mRowId em caso da nota que se está editando conter um estado salvo dentro do Bundle, o qual nós recuperaremos.

1. Substitua o código que inicializa o mRowId:

```
mRowId = null;

Bundle extras = getIntent().getExtras();

if (extras != null) {
        mRowId = extras.getLong(NotesDbAdapter.KEY_ROWID);
}
```

Por esse:

```
mRowId = (savedInstanceState == null) ? null :
      (Long) savedInstanceState.getSerializable(NotesDbAdapter.KEY_ROWID);

if (mRowId == null) {
   Bundle extras = getIntent().getExtras();
   mRowId = extras != null ? extras.getLong(NotesDbAdapter.KEY_ROWID): null;
}
```

2. Note a checagem por valores ”null” para savedInstanceState e que nós ainda precisamos carregar mRowId a partir do Bundle extras e ele não estiver sendo provido pelo savedInstanceState.

3. Note o uso de Bundle.getSerializable() em vez de Bundle.getLong(). Esse último método retorna um primitivo “long” e não poderá ser usado para representar no caso do mRowId ser nulo.

Passo 4

Agora, nós precisamos popular os campos baseados na mRowId que passamos:

```
populateFields();
```

O código acima deverá ser inserido antes de confirmButton.setOnClickListener(). Vamos criar esse método daqui a pouco.

Passo 5

Se livre da criação do Bundle e configuração de valores de Bundle que se encontra no método onClick(). A atividade não precisa mais retornar quaisquer informações extras para quem o chama. E já que não precisa mais ter um Intent para retornar, nós usaremos uma versão mais curta de setResult():

```
public void onClick(View view) {
        setResult(RESULT_OK);
        finish();
}
```

Agora tomaremos conta do processo de guardar as atualizações e novas notas no banco de dados nós mesmos, usando os métodos de ciclo de vida.

O método completo de onCreate() deverá se parecer com o código abaixo:

```
super.onCreate(savedInstanceState);

mDbHelper = new NotesDbAdapter(this);
mDbHelper.open();

setContentView(R.layout.note_edit);
setTitle(R.string.edit_note);

mTitleText = findViewById(R.id.title);
mBodyText = findViewById(R.id.body);
Button confirmButton = findViewById(R.id.confirm);

mRowId = (savedInstanceState == null) ? null :
      (Long) savedInstanceState.getSerializable(NotesDbAdapter.KEY_ROWID);
if (mRowId == null) {
   Bundle extras = getIntent().getExtras();
   mRowId = extras != null ? extras.getLong(NotesDbAdapter.KEY_ROWID): null;
}
populateFields();
confirmButton.setOnClickListener(new View.OnClickListener() {
   public void onClick(View view) {
      setResult(RESULT_OK);
      finish();
   }
});
```

Passo 6

Defina o método populateFields():

```
private void populateFields() {
   if (mRowId != null) {
      Cursor note = mDbHelper.fetchNote(mRowId);
      startManagingCursor(note);
      mTitleText.setText(note.getString(
            note.getColumnIndexOrThrow(NotesDbAdapter.KEY_TITLE)));
      mBodyText.setText(note.getString(
            note.getColumnIndexOrThrow(NotesDbAdapter.KEY_BODY)));
   }
}
```

Esse método usa o método NotesDbAdapter.fetchNote() para encontrar a nota correta a ser editada e então chama startManagingCursor() a partir da classe Activity,

que é um método conveniente do Android provido para tomar conta do ciclo de vida de um Cursor. Ele vai liberar e recriar recursos como ditado pelo ciclo de vida da Atividade, para não precisarmos nos preocupar em fazer isso nós mesmos. Após isso, nós apenas procuramos o title e body a partir do Cursor e populamos os elementos da View com eles.

Passo 7

Ainda dentro da classe NoteEdit, nós fazemos o override dos métodos onSaveInstanceState(), onPause() e onResume(). Esses são métodos de ciclo de vida (em parceria com o onCreate() que nós já temos).

O onSaveInstanceState() é chamado pelo Android se uma atividade está sendo parada e deve ser morta antes de ser resumida. Isso significa que devemos guardar quaisquer estados necessários para a sua reinicialização para a mesma condição de quando a atividade é reiniciada. Ela é a contraparte de onCreate() e, de fato, o Bundle savedInstanceState passado para o onCreate() é o mesmo Bundle que você construiu como onState no método onSaveInstanceState().

O onPause() e onResume() são também métodos complementares. onPause() é sempre chamado quando a atividade termina, mesmo que ela seja instigada (com um finish(), por exemplo). Nós usaremos isso para salvar a nota atual para o banco de dados. A boa prática é liberar quaisquer recursos que possam ser liberados durando o onPause() também, para que menos recursos sejam usados quando estivermos em estado passivo. onResume() vai chamar o nosso método populateFields() para ler a nota do banco de dados novamente e popular os campos.

Então, adicione algum espaço após o método populateFields() e adicione o seguinte método de ciclo de vida:

a) onSaveInstanceState():

```
@Override
protected void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    saveState();
    outState.putSerializable(NotesDbAdapter.KEY_ROWID, mRowId);
}
```

b) onPause():

```
@Override
protected void onPause() {
```

```
    super.onPause();
    saveState();
}
```

c) onResume():

```
@Override
protected void onResume() {
    super.onResume();
    populateFields();
}
```

Note que saveState() é chamado em onSaveInstanceState() e onPause() para certificarmos que os dados são salvos. Isso acontece já que não há garantias que onSaveInstanceState() será chamado porque quando é chamado ele é chamado antes de onPause().

Passo 8

Defina o método saveState():

```
private void saveState() {
    String title = mTitleText.getText().toString();
    String body = mBodyText.getText().toString();

    if (mRowId == null) {
        long id = mDbHelper.createNote(title, body);
        if (id > 0) {
            mRowId = id;
        }
    } else {
        mDbHelper.updateNote(mRowId, title, body);
    }
}
```

Note que capturamos o valor de retorno de createNote() e se um ID válido de linha é retornado, nós guardamos esse valor em mRowId para que nós possamos fazer o update dessa nota no futuro.

Passo 9

Agora insira o seguinte código para onActivityResult():

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    fillData();
}
```

Passo 10

Remova as linhas que setam o title e body de onListItemClick().

```
Cursor c = mNotesCursor;

c.moveToPosition(position);
c.getColumnIndexOrThrow(NotesDbAdapter.KEY_BODY)));
```

e também remova:

```
i.putExtra(NotesDbAdapter.KEY_TITLE, c.getString(
        c.getColumnIndexOrThrow(NotesDbAdapter.KEY_TITLE)));
i.putExtra(NotesDbAdapter.KEY_BODY, c.getString(
```

para que o que deve sobrar no método é apenas o código abaixo:

```
super.onListItemClick(l, v, position, id);
Intent i = new Intent(this, NoteEdit.class);
i.putExtra(NotesDbAdapter.KEY_ROWID, id);

startActivityForResult(i, ACTIVITY_EDIT);
```

Você também pode remover o campo mNotesCursor da classe e voltar a usar uma variável local no método fillData().

```
Cursor notesCursor = mDbHelper.fetchAllNotes();
```

Pronto, basta rodar o aplicativo para ver como ficou.