

**Docentes** Sérgio Lopes, sergio.lopes@ipleiria.pt  
David Safadinho, david.safadinho@ipleiria.pt

# Ficha de Exercícios N/06

*Conversão da aplicação “Livros” para “Books” – Aplicação de estilos e FAB.*

## Aplicação “Books”

A ficha de exercícios anterior (N/05) terminou com uma aplicação *Android* que exemplifica a personalização da interface gráfica recorrendo aos ficheiros de recursos `strings.xml`, `colors.xml`, `dimens.xml`, etc. Além de uma lista de livros (com dados exemplo em código) e de uma pequena galeria de imagens (capas dos livros) a aplicação não oferecia outras funcionalidades nem uma interface gráfica atrativa.

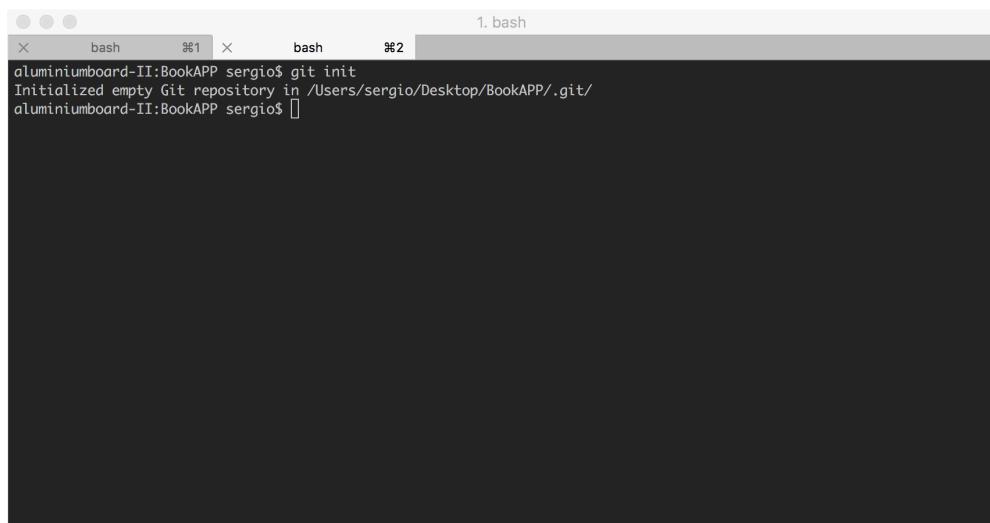
Nesta ficha iremos pegar nessa aplicação (numa versão parecida) e converter o código, interface e estilos e adicionar novas funcionalidades. Juntamente com as alterações, vamos também usar o sistema de controlo de versões *Git* e o alojamento GitHub para fazer o controlo de versões do nosso projeto. Para apoio deve consultar o código, e o PDF, da ficha anterior.

## Preparar o projeto

Comece por descompactar o projeto *Android* e abrir o *Android Studio* com este projeto. Verifique se o projeto compila adequadamente e se é possível executá-lo. Neste momento apenas as cores estarão diferentes e os botões não terão eventos associados.

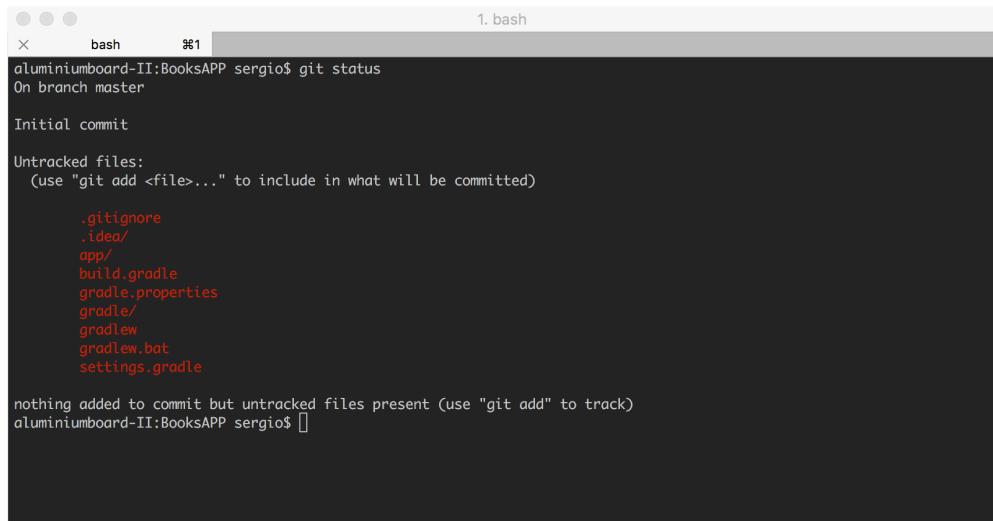
Abra um terminal/linha de comandos e inicie o repositório *Git* que vai ser usado. Execute o comando dentro da pasta do projeto (nota: o \$> não deve ser usado, representa a *prompt* da linha de comandos). **IMPORTANTE:** Não copie os comandos da ficha, os caracteres podem ser interpretados erradamente no terminal e causar erros.

```
$> git init
```



```
aluminumboard-II:BookAPP sergio$ git init
Initialized empty Git repository in /Users/sergio/Desktop/BookAPP/.git/
aluminumboard-II:BookAPP sergio$
```

Depois de executar o comando acima, se executar um status (`git status`) deverá ter uma lista de ficheiros identificados como na imagem seguinte.



```
aluminiumboard-II:BooksAPP sergio$ git status
On branch master

Initial commit

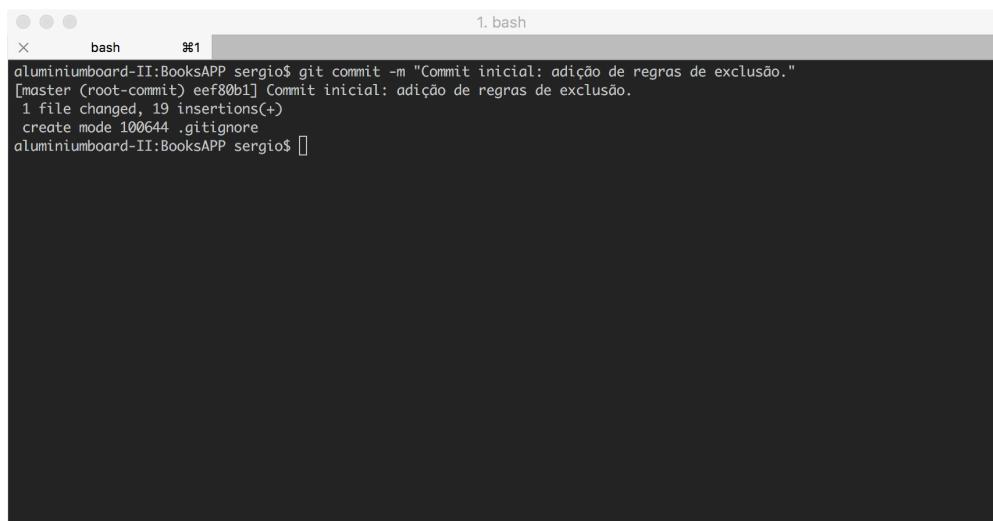
Untracked files:
  (use "git add <file>..." to include in what will be committed)

  .gitignore
  .idea/
  app/
  build.gradle
  gradle.properties
  gradle/
  gradlew
  gradlew.bat
  settings.gradle

nothing added to commit but untracked files present (use "git add" to track)
aluminiumboard-II:BooksAPP sergio$
```

Adicione apenas o ficheiro `.gitignore` e execute o primeiro **commit**.

```
$> git add .gitignore
$> git commit -m "Commit inicial: Execução de regras de exclusão."
```



```
aluminiumboard-II:BooksAPP sergio$ git commit -m "Commit inicial: adição de regras de exclusão."
[master (root-commit) eef80b1] Commit inicial: adição de regras de exclusão.
 1 file changed, 19 insertions(+)
 create mode 100644 .gitignore
aluminiumboard-II:BooksAPP sergio$
```

Adicione os restantes ficheiros do projeto e faça **commit**.

```
$> git add -all
$> git commit -m "Adiciona ficheiros base para projecto BooksAPP"
```

De modo a separar desenvolvimento de produção/lançamento da aplicação vamos usar dois **branches**: o **master** para desenvolvimento (instável); e o **deployment** para registar código estável e pronto a usar para distribuir a aplicação. Não execute este passo sem antes confirmar que o seu repositório tem todas as alterações registadas (`git status`).

```
$> git checkout -b deployment
```

Até ao momento o nosso repositório é local e não é fácil partilhá-lo com outros programadores. Para facilitar a interação entre programadores do mesmo projeto vamos usar o sistema *Github* ([github.com](https://github.com)) e alojar lá uma cópia do nosso repositório.

Se ainda não tem uma conta *Github* pode ignorar este passo, devendo ignorar todos os comandos que sincronizem o repositório local com o repositório remoto.

The left screenshot shows the 'Create a new repository' wizard. It asks for the repository name ('booksapp') and sets it to 'Public'. It also includes optional fields for a description and a README file. The right screenshot shows the GitHub repository page for 'Knitter / booksapp'. It provides quick setup instructions for cloning via HTTPS or SSH, pushing from the command line, or importing code from another repository.

Com o repositório criado no [github.com](https://github.com) é necessário submeter o código, criando uma referência entre o repositório local e o repositório remoto (que passará a ser designado **origin**). Os dois **branches** serão, também, submetidos.

```
$> git remote add origin git@github.com:Knitter/booksapp.git
$> git push -u origin master
$> git push -u origin deployment
```

A terminal window titled '1. bash' shows the execution of three git commands. The first command adds the GitHub repository as a remote named 'origin'. The second command pushes the 'master' branch to 'origin'. The third command pushes the 'deployment' branch to 'origin'. The output indicates that the 'deployment' branch is a new branch being pushed to 'origin'.

Volte ao **branch** de desenvolvimento (master) e volte ao *Android Studio* para iniciar a alteração do código para a nova versão.

```
$> git checkout master
```

## Dados de modelo

A nossa aplicação, designada *Books*, deverá permitir a gestão de livros de uma biblioteca pessoal. Quais são os requisitos de uma aplicação deste tipo?

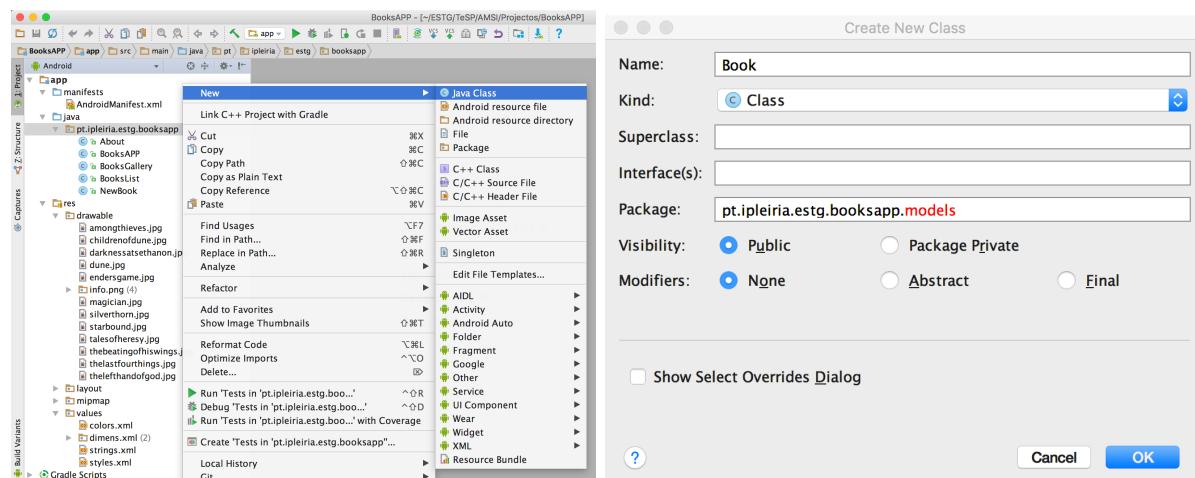
Podemos considerar como requisitos base/iniciais, os seguintes:

R.01	Permitir a inserção de livros, onde cada livro é composto por: <ul style="list-style-type: none"> <li>• Título</li> <li>• Série/coleção a que pertence, podendo não pertencer a uma coleção</li> <li>• Nome do autor</li> <li>• Ano de publicação</li> <li>• Código ISBN13</li> <li>• Sinopse/resumo</li> <li>• Classificação do leitor</li> </ul>
R.02	Apresentar uma lista de livros, permitindo a inserção de novos, pesquisa de livros registados e oferecendo acesso aos detalhes dos livros
R.03	Apresentar um ecrã com todos os detalhes do livro selecionado
R.04	Permitir ver uma galeria só com as capas dos livros.

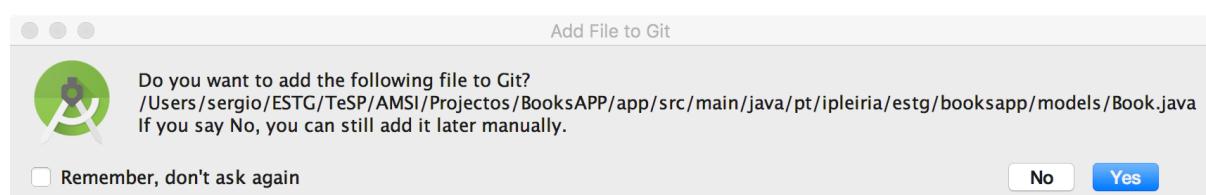
Com base nos requisitos anteriores crie a classe que representa o nosso modelo de dados, **Book**, e crie uma classe de apoio que forneça uma lista de livros de teste.

Será importante, para uma boa organização do projeto, organizar as nossas classes com packages de nomes sugestivos. Para isso tanto podemos criar o package manualmente ou criar o package no mesmo momento em que criamos uma classe. Usaremos esta última opção.

Clique com o botão direito em cima do package existente, escolha a opção **New > Java Class**. Na janela de detalhes da nova classe acrescente o package “**models**” (identificado a vermelho na imagem abaixo). Este package será criado e a classe será colocada na pasta correta automaticamente.

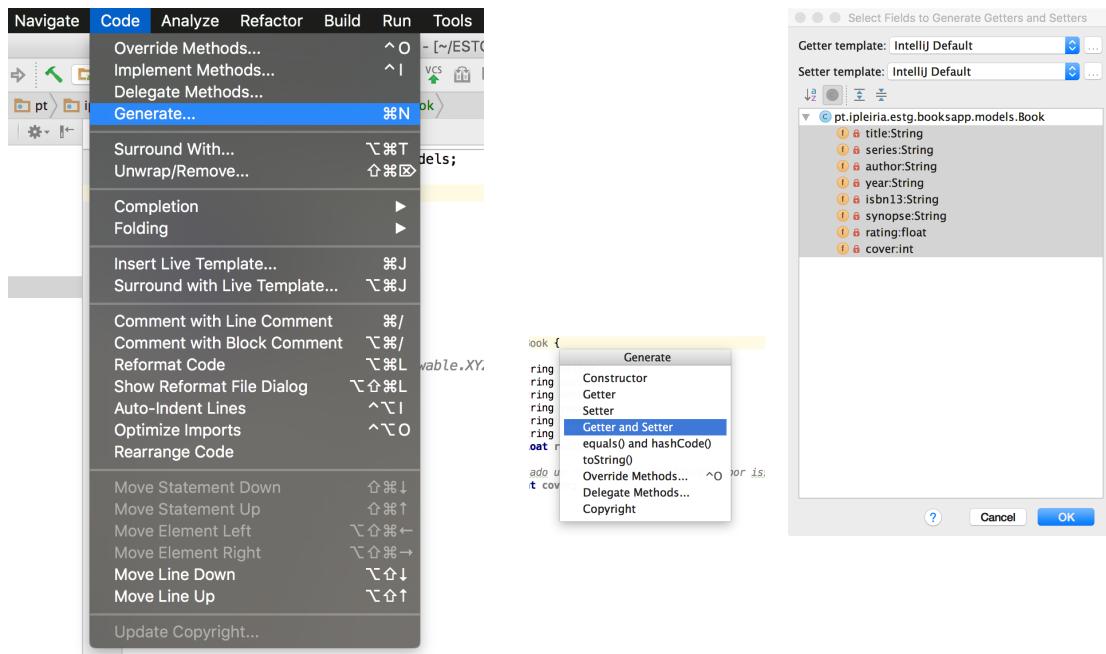


Como o projeto está a ser controlado por um sistema de controlo de versões, e como o *Android Studio* deteta o repositório, sempre que forem adicionados novos ficheiros o IDE irá perguntar se os queremos adicionar também ao repositório (adicionar ao **index/stage**). Podemos marcar a opção “**Remember, don't ask again**” para que seja o comportamento por omissão para que não esteja sempre a ser colocada a pergunta.



Criado o ficheiro da classe é necessário adicionar as propriedades correspondentes aos detalhes do livro, todas privadas e do tipo *String* exceto a capa. Esta será do tipo *int* por estarmos a usar referências para a pasta *drawable*.

Depois de adicionadas as propriedades 8 propriedades (**title**, **series**, **author**, **year**, **isbn13**, **synopse**, **rating** e **cover**) podemos usar o gerador de código para criar os **Getter/Setter** necessários.



No package onde criou a classe **Book** adicione uma segunda classe, **ExampleData**, e adiciona a essa a variável e os dois métodos definidos no código abaixo:

```
private static List<Book> books;

public static List<Book> getBooks() {
    if (books == null) {
        books = new ArrayList<>();

        books.add(new Book("Among Thieves", "Tales of the Kin, #1", "Douglas Hulick", "2011", "9780330536202", "Drothe is a Nose, an informant who finds and takes care of trouble inside the criminal organization he's a part of. He also smuggles imperial relics on the side. When his boss sends him to Ten Ways to track down who's been leaning on his organization's people, Drothe discovers hints of a much bigger mystery.", 0.0f, R.drawable.amongthieves));
        books.add(new Book("Children of Dune", "Dune, #3", "Frank Herbert", "1987", "9780441104024", "The desert planet of Arrakis has begun to grow green and lush. The life-giving spice is abundant. The nine-year-old royal twins, possessing their father's supernatural powers, are being groomed as Messiahs.", 0.0f, R.drawable.childrenofdune));
        books.add(new Book("A Darkness At Sethanon", "The Riftwar Saga, #4", "Raymond E. Feist", "2007", "9780007229437", "An evil wind blows through Midkemia. Dark legions have risen up to crush the Kingdom of the Isles and enslave it to dire magics. The final battle between Order and Chaos is about to begin in the ruins of the city called Sethanon.", 0.0f, R.drawable.darknessatsethanon));
        books.add(new Book("Dune", "Dune, #1", "Frank Herbert", "2005", "9780441013593", "Set on the desert planet Arrakis, Dune is the story of the boy Paul Atreides, who would become the mysterious man known as Muad'Dib. He would avenge the traitorous plot against his noble family--and would bring to fruition humankind's most ancient and unattainable dream. A stunning blend of adventure and mysticism, environmentalism and politics.", 0.0f, R.drawable.dune));
        books.add(new Book("Ender's Game", "The Ender Quintet, #1", "Orson Scott Card", "2013", "9780765370624", "Andrew \"Ender\" Wiggin thinks he is playing computer simulated war games; he is, in fact, engaged in something far more desperate. Ender may be the military genius Earth desperately needs in a war against an alien enemy seeking to destroy all human life. The only way to find out is to throw Ender into ever harsher training, to chip away and find the diamond inside, or destroy him utterly. Ender Wiggin is six years old when it begins. He will grow up fast.", 0.0f, R.drawable.endersgame));
        books.add(new Book("Magician", "The Riftwar Saga, #1 e #2", "Raymond E. Feist", "2009", "9780586217832", "At Crydee, a frontier outpost in the tranquil Kingdom of the Isles, an orphan boy, Pug, is apprenticed to a master magician - and the destinies of two worlds are changed forever.", 0.0f, R.drawable.magician));
        books.add(new Book("Silverthorn", "The Riftwar Saga, #3", "Raymond E. Feist", "2008", "9780007229420", "For nearly a year peace has reigned in the Kingdom of the Isles, but mischief is stirring again in the city of Krondor and new challenges await Prince Arutha when Jimmy the Hand - the youngest thief in the Guild of Mockers - stumbles upon a sinister Nighthawk poised to assassinate him.", 0.0f, R.drawable.silverthorn));
        books.add(new Book("Starbound", "Lightship Chronicles, #2", "Dave Bara", "2016", "9780091956424", "The Lightship H.M.S. Impulse is gone, sacrificed in a battle against First Empire ships. And though the fragile galactic alliance has survived the
    }
}
```

```

unexpected invasion, the Union forces might not prove victorious against a full onslaught by this legendary enemy", 0.0f,
R.drawable.starbound));
        books.add(new Book("Tales of Heresy", "The Horus Heresy, #10", "Dan Abnett", "2009", "9781844166824", "When Horus the
Warmaster rebelled against the Emperor, the ensuing civil war nearly destroyed the Imperium. War raged across galaxy, pitting
Astartes against their battle-brothers in a struggle where death was the only victor.", 0.0f, R.drawable.talesofheresy));
        books.add(new Book("The Beating of His Wings", "The Left Hand of God #3", "Paul Hoffman", "2013", "9780718155223", "Since
discovering that his brutal military training has been for one purpose-to destroy God's greatest mistake, mankind itself-Cale has
been hunted by the very man who made him into the Angel of Death: Pope Redeemer Bosco.", 0.0f, R.drawable.thebeatingofhiswings));
        books.add(new Book("The Last Four Things", "The Left Hand of God, #2", "Paul Hoffman", "2011", "9780718155216", "To the
warrior-monks known as the Redeemers, who rule over massive armies of child slaves, \"the last four things\" represent the
culmination of a faithful life. Death. Judgement. Heaven.", 0.0f, R.drawable.thelastfourthings));
        books.add(new Book("The Left Hand of God", "The Left Hand of God #1", "Paul Hoffman", "2010", "9780718155186", "The
Sanctuary of the Redeemers is a vast and desolate place-a place without joy or hope. Most of its occupants were taken there as boys
and for years have endured the brutal regime of the Lord Redeemers whose cruelty and violence have one singular purpose-to serve in
the name of the One True Faith.", 0.0f, R.drawable.thelighthandofgod));
    }

    return books;
}

public static Integer[] getAllCovers() {
    Integer covers[] = new Integer[getBooks().size()];

    int i = 0;
    for (Book book : getBooks()) {
        covers[i++] = book.getCover();
    }

    return covers;
}

```

Volte ao terminal/linha de comandos e registe todas as alterações, adicionando os ficheiros modificados ao **index/stage**, fazendo **commit**, **merge** do **branch** master no **deployment** e **push** de todas as mudanças para sincronização com o repositório alojado no github.com

```

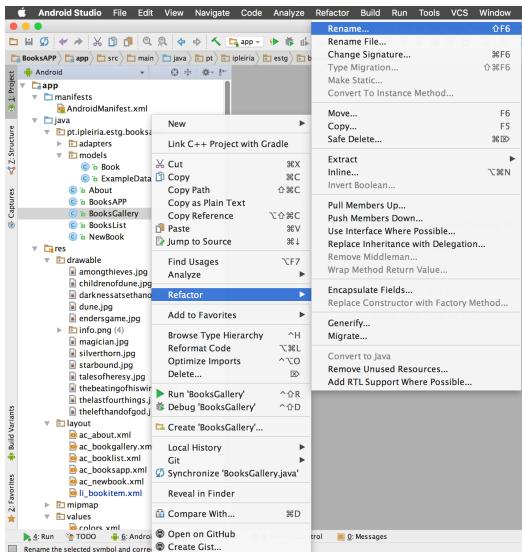
$> git add -all
$> git commit -m "Adiciona classe para modelo e dados exemplo"
$> git checkout deployment
$> git merge master
$> git checkout master
$> git push --all

```

The image consists of two side-by-side terminal windows. Both windows have a title bar '1. bash' and a status bar at the bottom. The left terminal shows the command 'git merge master' being run, with several file names listed below it followed by progress bars. The right terminal shows the command 'git push --all' being run, with a message indicating the branch is ahead of 'origin/master' by 1 commit.

## Atualização do Código

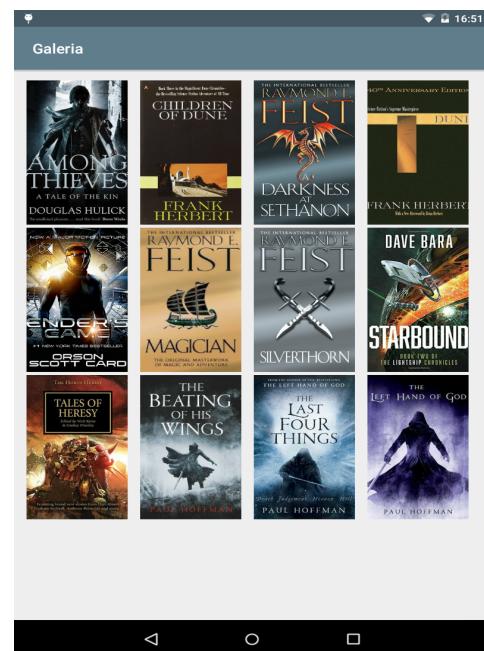
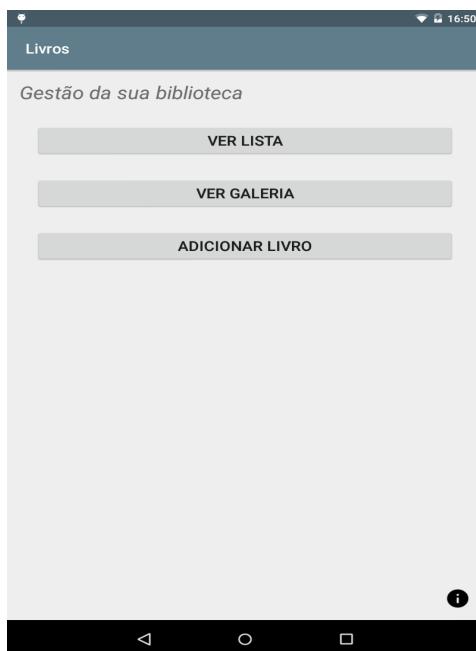
Atualize o código do projeto corrigindo o nome de classes, layouts e as propriedades que foram encontradas em português de modo a que passem a estar todas em inglês. Para alterar o nome de ficheiros use a opção de **refactoring** de modo a evitar erros no projeto.



Implemente a grelha de imagens com as seguintes alterações:

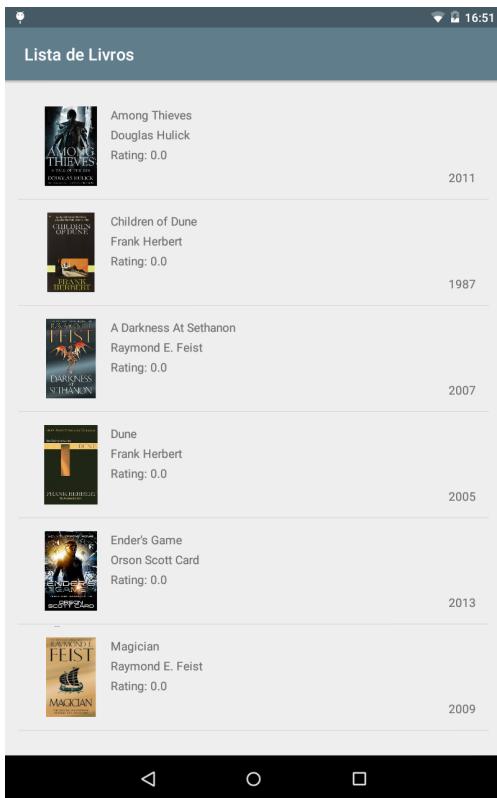
- Renomear a classe da atividade da galeria para **BookGallery**
- Adicionar a classe **ImageCellAdapter**, no package **adapters** (necessário criar) para fazer a adaptação entre as imagens e os dados
- Remover a propriedade que define o número de colunas no XML do layout
- Definir o número de colunas no método **onCreate** da atividade (**setNumColuns**), usar 3 colunas para *smartphones* e 4 para *tablets*.
- Criar dimensões (em *dimens.xml*) para:
  - Espaçamento horizontal entre células, **5dp**, aplicar, no XML da atividade BookGallery (*ac\_bookgallery.xml*), na GridView, com **android:horizontalSpacing**
  - Espaçamento vertical entre células, **5dp**, aplicar, com no ponto anterior, com **android:verticalSpacing**
  - Altura das imagens (das *ImageView*), **214dp**, aplicar no método **getView** da classe **ImageCellAdapter**
  - Largura das imagens (das *ImageView*), **128dp**, aplicar no método **getView** da classe **ImageCellAdapter**
  - Como é que convertemos o valor da altura/largura (**214dp/128dp**) para o valor correto em pixel que precisamos de usar no método **setLayoutParams** da *ImageView*?

Implemente o código necessário para abrir a galeria de imagens com o novo formato, o resultado final deverá ser semelhante à imagem seguinte:



## Implementar lista de Livros

À semelhança do que foi feito acima é necessário corrigir os nomes das classes das atividades e dos layouts de modo a respeitarem a nomenclatura adotada ao longo do projeto.



- Corrigir nome da atividade de lista de livros para **BookList**
- Implementar o adaptador para conversão de dados do livro num item da lista
- No método **getView** do adaptador é necessário carregar o XML que representa um item da lista:  
**inflater = (LayoutInflater)**  
**context.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);**
- Com o **inflater** criado acima é possível carregar a view base do layout:  
**reusedView = inflater.inflate(R.layout.li\_bookitem, null);**
- O resto do método é semelhante ao que foi feito para o adaptador da grelha, mas em vez de aceder apenas a uma *ImageView* acede a todos os componentes do layout que representa um item (**title, author, rating, year, cover**).

## Adicionar menu da aplicação

O menu atual da aplicação não é visualmente agradável, a interação normal não faz uso de uma atividade apenas para ter uma lista de botões. Assim, vamos substituir a atividade do menu pela **BookList**, fazendo desta a atividade principal (atenção: não esquecer de atualizar o **AndroidManifest.xml**) e convertendo o menu num menu de aplicação disponível na barra de topo.

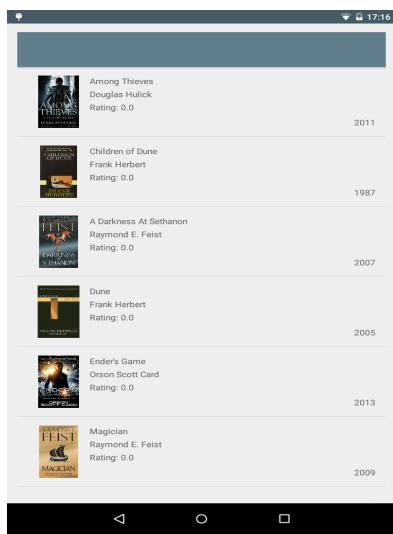
Remova a atividade do menu e o layout correspondente

Atualize o **AndroidManifest.xml** para que a atividade principal seja a **BookList**, retire o **intent-filter** que permitia iniciar a atividade manualmente

Corrija algum ID ou texto que já não seja usado ou que tenha sido alterado (ex.: nome da aplicação, *labels* das atividades, *Strings* para menu principal)

Nem todas as versões *Android* possuem uma barra de ferramentas no topo (**ActionBar**). Em alguns casos este menu é apresentado no fundo do ecrã. Para garantir que a nossa aplicação funciona num maior número de dispositivos vamos desativar a **ActionBar** padrão e implementar uma **ActionBar** de compatibilidade.

- Altere o ficheiro **style.xml** para que o tema base seja **Theme.AppCompat.Light.NoActionBar**, com esta alteração estamos a configurar a nossa aplicação para que não tenha qualquer **ActionBar**
- Altere o layout da atividade **BookList** para **RelativeLayout**
- Adicione (ou confirme se existe) na tag XML inicial o parâmetro para usar o namespace “app”, o XML deve conter a seguinte propriedade (além da que já se encontram):  
**xmlns:app="http://schemas.android.com/apk/res-auto"**
- Coloque no topo uma Toolbar com a seguinte configuração:
  - ```
<android.support.v7.widget.Toolbar  
    android:id="@+id/tbAcBookListToolbar"  
    android:layout_width="match_parent"  
    android:layout_height="?attr/actionBarSize"  
    android:background="?attr/colorPrimary"  
    android:elevation="4dp"  
    android:titleTextColor="@color/textColor"  
    app:titleTextColor="@color/textColor"  
    app:subtitleTextColor="@color/secundaryTextColor"/>
```
- Adicione à atividade, no método **onCreate**, o código necessário para registar a nossa **ActionBar**:
  - ```
Toolbar toolbar = (Toolbar) findViewById(R.id.tbAcBookListToolbar);  
setSupportActionBar(toolbar);
```



**Q:** Se já experimentou executar a aplicação, o que é que acontece às margens?

**Q:** Porque é que a nossa **ActionBar** não está no local certo?

É necessário remover as margens extra que o **RelativeLayout** aplica. As margens devem ser feitas pelos componentes filhos do **RelativeLayout** para que o pai posso ocupar o ecrã todo, possibilitando assim que a **ActionBar** tenha as dimensões corretas.

Ao **RelativeLayout** adicione a propriedade **android:fitsSystemWindows="true"** e remove todas as definições de margens ou *padding*.

**Q:** Porque é que o layout da nossa **ActionBar** usa **?attr** em vez de **@color**?

O que é que nos falta? A aplicação já possui uma **ActionBar** mas não temos nenhum botão para responder à interação do utilizador. Para adicionarmos opções à **ActionBar** (na verdade uma **Toolbar**) precisamos de um ficheiro XML para definir o menu, de implementar o método *onCreateOptionsMenu()* que vai devolver o menu de opções da atividade e de implementar o método *onOptionsItemSelected()* que será responsável por tratar os pedidos do utilizador (escolhas dos itens do menu).

Crie uma pasta menu na raiz da pasta res:

1. Botão direito do rato em cima da pasta res;
2. Opção **New > Android resource directory**;
3. *Directory name: menu; Directory type: menu;*

Dentro dela crie um novo ficheiro XML com o nome **toolbar.xml** (o nome pode ser qualquer outro desde que o código seja ajustado):

1. Botão direito do rato em cima da nova pasta menu;
2. Opção **New > Menu resource file**;
3. File name: **toolbar.xml**;

Neste XML inclua a definição das duas opções que estavam na antiga atividade principal (Galeria e Sobre):

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"  
      xmlns:app="http://schemas.android.com/apk/res-auto">  
    <item  
        android:id="@+id/mi_booklist_gallery"  
        android:title="@string/mi_booklist_toolbar_gallery"  
        app:showAsAction="never"/>  
  
    <item  
        android:id="@+id/mi_booklist_about"  
        android:title="@string/mi_booklist_toolbar_about"  
        app:showAsAction="never"/>  
</menu>
```

Voltando ao código da atividade (*BookList*) adicione os dois métodos necessários para apresentar o menu e tratar os pedidos do utilizador:

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.toolbar, menu);  
    return true;  
}  
  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
  
    int id = item.getItemId();  
    if (id == R.id.mi_booklist_gallery) {  
        Intent i = new Intent("pt.ipleiria.estg.booksapp.GALLERY");  
        startActivity(i);  
  
        return true;  
    }  
  
    if (id == R.id.mi_booklist_about) {  
        Intent i = new Intent("pt.ipleiria.estg.booksapp.ABOUT");  
        startActivity(i);  
  
        return true;  
    }  
  
    return super.onOptionsItemSelected(item);  
}
```

**Q:** O que fazem estes dois métodos? Consegue perceber o objetivo de cada um deles?

E embora o nosso menu não tenha muitas opções, podemos já usar uma APP com uma interação mais atual, e evoluir a nossa aplicação para uma versão melhor.

### Detalhes e criação/editação de livros

A próxima evolução da nossa aplicação implica a criação de uma atividade com um formulário de criação de novos livros. Não nos iremos preocupar com as imagens, focando apenas a edição de detalhes de novos livros e de livros existentes.

Crie uma nova atividade chamada **BookDetails** (não se esqueça de adicionar o *intent-filter* necessário à abertura da atividade no *AndroidManifest.xml*) e implemente um ecrã para apresentação dos detalhes de um livro. A atividade recebe, por parâmetros extra do *Intent* (ver passagem de parâmetros para atividades), o número do livro (posição da lista de livros) e apresenta os dados desse livro usando *TextView* e *ImageView*.

Na atividade *BookList* adicione o evento necessário para que, quando o utilizador carregar num dos livros da lista, seja apresentada a atividade de detalhes (método *setOnItemClickListener*). Passe por parâmetro extra a posição do item selecionado (que no nosso caso permite identificar o livro) de forma a que seja possível à atividade *BookDetails* saber que livro deve apresentar.

Nesta atividade, adicione uma nova *Toolbar* (implementar o método *onCreateOptionsMenu*, o *onOptionsItemSelected*, criar um novo ficheiro XML com o item do menu, etc.), com uma opção sempre visível para editar o livro que está a ser apresentado. Esta opção deve apresentar a atividade de edição de dados que se chamará **BookForm** – Altere o nome da atividade **NewBook** (que já está criada) para **BookForm** (não se esqueça de alterar também o *AndroidManifest.xml*).

Como a atividade *BookForm* não tem qualquer elemento, implemente nesta um formulário para edição dos dados de um livro ou, se não for passado nenhum à atividade (se o *Intent* não tem parâmetros extra), para criação de um novo livro.

Adicione a esta atividade uma nova *Toolbar* com um botão para guardar as alterações (o código para guardar novos livros ou editar dados será criado na secção seguinte).

### Criar/editar livros no modelo de dados exemplo

Para podermos manipular os livros na nossa pequena biblioteca, e como ainda usamos a classe *ExampleData* para fornecer os dados da aplicação, iremos adicionar métodos a esta classe. Assim, adicione à classe *ExampleData* os métodos **addBook**, **replaceBook** e **deleteBook**.

Os três métodos devem ter o modificador **static** para que possam manipular a lista de livros (por esta ser também **static**) e o método **addBook** recebe o livro a adicionar à lista e não devolve valor; o método **replaceBook** recebe a posição do livro a substituir e o novo livro, devolvendo verdadeiro se a substituição funcionar, falso caso contrário; e o método **deleteBook** recebe a posição do livro a remover, devolvendo verdadeiro caso o livro seja removido ou falso caso contrário.

Atualize a atividade **BookForm** para que o evento do botão guardar, na *ActionBar*, invoque o método **addBook** ou **replaceBook** caso seja a introdução de um novo livro ou a edição de dados, respetivamente.

(código disponível abaixo)

## Ponto de Situação

Neste momento a aplicação móvel **Books** deve ter:

- Atividade principal chamada **BookList** com uma *ListView* que apresenta objetos do tipo *Book*;
- Atividade secundária chamada **About**, acessível na *ActionBar* da atividade principal;
- Atividade secundária chamada **BookGallery**, acessível na *ActionBar* da atividade principal e com uma *GridView* que apresenta várias imagens estáticas;
- Atividade **BookDetails**, acessível através da seleção de um dos elementos da lista de livros (atividade **BookList**) que apresenta os detalhes do livro selecionado;
- Atividade **BookForm**, acessível através da *ActionBar* da atividade **BookDetails**, que permite editar os dados do livro selecionado ou criar um novo livro se não for selecionado nenhum (ainda não está implementado o código para gravar o novo livro);
- Os dados de um livro são editáveis e é possível adicionar novos livros.

**Q:** Porque é que, ao editar um livro, se voltarmos à atividade de detalhes esta não apresenta os novos detalhes do livro? Como é que podemos resolver este problema?

## Sugestão de Código para classe ExampleData

```
public static void addBook(Book book) {
    if (books == null) {
        criarLista();
    }

    books.add(book);
}

public static boolean replaceBook(int position, Book book) {
    if (books != null && position >= 0 && position < books.size()) {
        books.set(position, book);
        return true;
    }

    return false;
}

public static boolean removeBook(int position) {
    if (books != null && position >= 0 && position < books.size()) {
        books.remove(position);
        return true;
    }

    return false;
}
```

## Base de Dados Local numa APP

A adição de uma base de dados local que use *SQLite* passa por criar uma classe de apoio, que seja subclasse de **SQLiteOpenHelper**. Esta é uma classe que nos ajuda a organizar o acesso à base de dados e esconde alguns dos pormenores de criação da base de dados, verificação da sua existência, abertura para manipulação, etc.

Assim, vamos começar por criar uma nova classe, de nome **DataAccessHelper** no *package models*. Como indicado acima, esta classe deve estender de **SQLiteOpenHelper** deve implementar dois métodos obrigatórios e um construtor: o método **onCreate** e o método **onUpdate**, e um construtor que receba o contexto de execução.

Este construtor necessita apenas invocar o construtor pai e passar os parâmetros de configuração da nossa base de dados:

```
private static final int DB_VERSION = 1;
private static final String DB_NAME = "books";

public DataAccessHelper(Context context) {
    super(context, DB_NAME, null, DB_VERSION);
}

//. . .
```

No método **onCreate** adicione, numa variável do tipo *String*, o código SQL necessário para criar a tabela que guarda os dados dos livros. A tabela terá o mesmo nome que a classe de modelo que a representa e os mesmos campos e tipos de dados.

**Q:** Que alterações são necessárias fazer à nossa classe de modelo (*Book*) para que possa ser guardada numa base de dados? E nas restantes classes para que seja usada a base de dados?

Tal como foi necessário fazer na classe *ExampleData*, e como a classe *DataAccessHelper* vai gerir os dados dos livros, é necessário adicionar métodos para inserir, listar, atualizar e remover livros. Nesse sentido, crie os métodos **boolean addBook(Book book)**, e **List<Book> listBooks()**.

A fim de ter alguns dados de teste crie o método **void populateDatabase()** e nele insira os dados dos livros de exemplo (tal como é feito no método *criarLista* da classe *ExampleData*) e invoque este método no fim do método *onCreate* da classe *DataAccessHelper*. Ao executar a aplicação pela primeira vez e ao ser criada a base de dados, estes dados exemplo serão injetados e poderão ser usados para teste até que a aplicação esteja desenvolvida.

Altere a atividade **BookList** para que sejam listados os livros existentes na base de dados em vez de ser usada a classe *ExampleData*. Faça o mesmo para a atividade **BookDetails** e **BookForm** e finalmente remova a classe *ExampleData* do projeto de modo a garantir que só é usado o acesso à base de dados.

## Botão FloatingAction e mensagens ao utilizador

Continuando com as melhorias à nossa aplicação, vamos remover as duas atividades extra, as **ActionBar** que não são práticas e converter a aplicação para usar um **FloatingActionButton**, possibilitar a remoção de livros e apresentar mensagens ao utilizador recorrendo à classe *Toast*.

Comece por remover as imagens da pasta “res”, referentes às capas dos livros exemplo e ao ícone *info.png*. Remova também as atividades **BookGallery** e **About**.

Adicione um FAB à lista de livros e configure o evento de clique para que seja apresentada a atividade de criação de livros.

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fabNewNook"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/info"
    android:layout_alignParentBottom="true"
    android:layout_alignParentEnd="true"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="16dp"
    android:clickable="true"
    app:elevation="4dp"/>
```

Remova o código que permite criar o menu da **Toolbar** da atividade de lista de livros (removendo o método `onOptionsItemSelected`). Altere as *toolbars* existentes para que façam uso dos ícones fornecidos e não apresentem o texto associado.

No fim, a sua aplicação deverá ter o aspeto apresentado nas imagens seguintes.

