



DESENVOLVIMENTO DE APLICAÇÕES MOBILE E DISTRIBUÍDAS

ILUMNO

DESENVOLVIMENTO DE APLICAÇÕES MOBILE E DISTRIBUÍDAS

ILUMNO

Copyright © UVA 2019

Nenhuma parte desta publicação pode ser reproduzida por qualquer meio sem a prévia autorização desta instituição.

Texto de acordo com as normas do Novo Acordo Ortográfico da Língua Portuguesa.

AUTORIA DO CONTEÚDO

Carlos Augusto Sicsú Ayres do Nascimento

PROJETO GRÁFICO

UVA

REVISÃO

REVISÃO

Theo Cavalcanti

Isis Batista

Lydianna Lima

DIAGRAMAÇÃO

UVA

SUMÁRIO

| | |
|---------------------|----------|
| Apresentação | 6 |
| Autor | 8 |

UNIDADE 1

Conceitos básicos e fundamentos da programação mobile 9

- O histórico do desenvolvimento de aplicações móveis
- Características do sistema operacional Android®
- O ambiente de desenvolvimento de aplicativos para Android®

UNIDADE 2

O modelo MVC e a criação de atividades no Android® 39

- O modelo MVC
- O ciclo de vida de uma atividade
- Construção de um aplicativo

SUMÁRIO

UNIDADE 3

Trabalhando com componentes do sistema e múltiplas interfaces 85

- A *intent* e o acesso a componentes do sistema
- Envio e recebimento de mensagens no Android
- Navegação entre telas e envio de parâmetros

UNIDADE 4

Persistência de dados no Android 145

- Trabalhando com a persistência de dados em arquivos de preferências
- Trabalhando com a persistência de dados em arquivos locais
- Trabalhando com a persistência de dados em banco de dados local (*SQLite*)

APRESENTAÇÃO

Com a crescente popularização dos aplicativos para dispositivos móveis, o desenvolvimento de sistemas tem se expandido muito mais ao longo dos últimos anos. A criação de aplicações para auxiliar o dia a dia das pessoas já não se restringe aos computadores e tem se expandido para os dispositivos móveis. Cada vez mais pessoas passam a utilizar smartphones e tablets para suas tarefas diárias e para o entretenimento. O uso de redes sociais e de aplicativos de mobilidade urbana, compra e venda, aluguel de imóveis por temporada, compartilhamento de fotos e vídeos, entre vários outros, tem tido um crescimento enorme. As pessoas atualmente pouco utilizam seus aparelhos para conversas telefônicas, mas principalmente para trocas de mensagens, postagens, compartilhamento e uso de aplicativos que facilitam seu cotidiano.

Dentro desse contexto, a elaboração de aplicativos para atender à enorme demanda de inovação e disponibilização de novas tecnologias tem tido um grande efeito no desenvolvimento de sistemas voltados para os aparelhos móveis. Simples aplicações de trocas de mensagens não mais são suficientes para a demanda existente, e aplicações antes disponibilizadas na web passaram a ter versões específicas para os dispositivos móveis. Empresas de vendas pela internet, por exemplo, passaram a disponibilizar versões específicas em seus aplicativos para facilitar a compra, o pagamento e o controle do envio por meio de ferramentas de rastreamento. As redes sociais passaram a desenvolver versões específicas de forma a atender melhor seus usuários. Empresas que trabalham com transporte e aluguel por temporada também disponibilizam aplicativos específicos para atender ao número cada vez maior de clientes, assim como companhias aéreas e várias outras empresas dos mais variados tipos de nichos de mercado passaram a disponibilizar versões *mobile* para satisfazer as demandas de seus clientes.

O desenvolvimento de aplicativos para dispositivos móveis demanda todas as técnicas e metodologias de desenvolvimento criadas até o momento, de forma que a arquitetura cliente × servidor, que inicialmente foi elaborada para atender à demanda de sistemas

empresariais, com um servidor de bases de dados e usuários clientes com computadores pessoais, deu lugar ao desenvolvimento para a web. Hoje em dia, toda essa tecnologia está à disposição para o desenvolvimento de aplicações móveis, permitindo que a arquitetura cliente × servidor evolua cada vez mais e atenda a uma maior demanda do mercado. Desenvolver aplicativos é um mercado em forte expansão e que exigirá ao longo dos próximos anos muita mão de obra.

AUTOR

PROF. ENG. CARLOS AUGUSTO SICSÚ AYRES DO NASCIMENTO, DSC

Possui curso técnico em Edificações pelo Centro Federal de Educação Tecnológica Celso Suckow da Fonseca – Cefet/RJ e é formado em Matemática Aplicada à Informática e em Engenharia da Computação. É especialista em Engenharia da Computação pela Universidade do Estado do Rio de Janeiro – Uerj, mestre em Engenharia da Computação também pela Uerj e doutor em Computação de Alto Desempenho pelo Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia – Coppe/UFRJ.

Possui mais de 20 anos de experiência no magistério em cursos presenciais e EAD, trabalhou na área de TI da indústria de embalagens durante dois anos e, atualmente, dedica-se ao desenvolvimento de projetos de pesquisa na área de aplicativos para dispositivos móveis.

UNIDADE 1

Conceitos básicos e fundamentos
da programação *mobile*

INTRODUÇÃO

A programação *mobile*, ou programação para dispositivos móveis, vem se tornando uma das principais áreas no desenvolvimento de sistemas. Redes sociais e empresas de transporte, aluguel, vendas, serviços, voltadas ao atendimento ao cliente (agendamentos), da saúde, de educação e de entretenimento ou já possuem seus aplicativos ou estão desenvolvendo suas versões para dispositivos móveis. O principal foco está em atender à enorme demanda gerada por seus usuários e clientes por aplicativos simples, fáceis de usar e, principalmente, úteis.



OBJETIVO

Nesta unidade, você será capaz de:

- Conhecer o que são os sistemas operacionais – SOs, além dos dois principais SOs para dispositivos móveis, assim como aprenderá a instalar e a configurar o Android® Studio. Você ainda conhecerá o ambiente de desenvolvimento mais usado para criar aplicativos para Android®.

O histórico do desenvolvimento de aplicações móveis

Histórico da computação móvel

A computação móvel se iniciou a partir de equipamentos de computação pessoais, chamados PDAs (personal digital assistants), ainda na década de 1980. Esses aparelhos permitiam o uso de agendas e aplicativos simples, mas ainda não eram usados para comunicação por voz e possuíam transferência de dados muito limitadas, sendo na verdade superagendas pessoais.



Aparelho PDA

Com o aumento da capacidade e da qualidade da comunicação de voz e dados pelas redes sem fio, ocorreu convergência entre os PDAs e os aparelhos celulares, chegando aos hoje bastante conhecidos smartphones, que são aparelhos de comunicação por voz e dados que permitem a execução de aplicativos mais complexos e avançados. Esses equipamentos se utilizam da comunicação de dados e voz via diversas formas de troca wireless, tais como: redes 2G/3G/4G/5G, Bluetooth®, Wi-Fi®, NFC, entre outras.

Uma das primeiras empresas a disponibilizar aplicações em seus aparelhos foi a BlackBerry®, empresa norte-americana que comercializou durante muito tempo um dos primeiros smartphones de sucesso. Esse aparelho possuía um miniteclado completo e um *trackball* (equipamento semelhante a um mouse no qual o usuário movimenta uma esfera que orienta o deslocamento do cursor), e também permitia fácil acesso à internet e a utilização de aplicações, o que fez com que muitas empresas e usuários desejassesem ter isso ao alcance das mãos. Os usuários desses equipamentos passaram a ter acesso

à comunicação de voz e dados em praticamente todo lugar e tinham acesso rápido e preciso a informações necessárias para o fechamento de negócios.



Aparelho BlackBerry.

Os smartphones muito avançaram e ganharam mais versatilidade e desempenho a partir do uso das telas de toque.



Aparelhos do tipo smartphone.

Atualmente, compramos aparelhos smartphones como compramos um computador. Algumas de suas principais características na hora da compra são o processador, a quantidade de memória e a resolução, tanto da tela como das câmeras, uma vez que aparelhos com apenas uma câmera não traduzem o desejo dos usuários. As novas gerações de aparelhos vêm exigindo cada vez mais desempenho computacional para atender às

necessidades de aplicativos cada vez mais complexos, o que faz a indústria do ramo investir muito no desenvolvimento de novos modelos de processadores voltados para o setor de computação móvel pessoal. Mesmo aparelhos de baixo custo precisam atender a essa demanda crescente por desempenho computacional.



Jovens aproveitando seus smartphones.

O que temos hoje é uma grande revolução, na qual o smartphone passou a fazer parte do nosso dia a dia e os aplicativos são os grandes protagonistas.



MIDIA TECA

Acesse a midiateca da Unidade 1 e veja o conteúdo complementar indicado pelo professor a respeito da introdução aos sistemas operacionais.



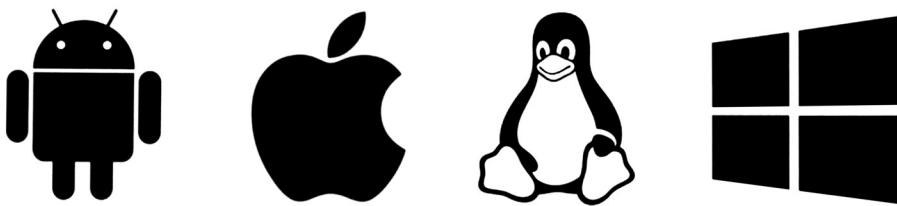
Saiba mais

Introdução ao Android

DEITEL, P. J.; DEITEL, H. M.; DEITEL, A. **Android:** como programar. Porto Alegre: Bookman, 2015. Minha Biblioteca. cap. 1, seções 1.1 a 1.3, p. 1-6.

Características do sistema operacional Android®

Dentro da área dos sistemas de informação, nós temos diferentes tipos de sistemas, mas um dos tipos mais importantes é o software básico. Como o próprio nome já diz, o software básico é responsável pelo controle do sistema de computação como um todo, controlando todos os seus componentes. O SO é um grande exemplo de software básico, pois é ele quem controla o computador ou dispositivo de computação desde o momento em que o ligamos até a hora em que ele é desligado. O SO controla a inicialização do sistema (*boot*), o(s) processador(es), as memórias (principal e secundárias), os dispositivos de entrada e de saída, as câmeras etc. O SO ainda é responsável por inicializar as aplicações, controlá-las durante o tempo de execução e encerrá-las. Toda a comunicação entre as aplicações e os componentes é realizada pelo SO, tornando-o assim o principal software de um sistema de computação.



Rose Carson / Shutterstock.com
Exemplos de SOs.

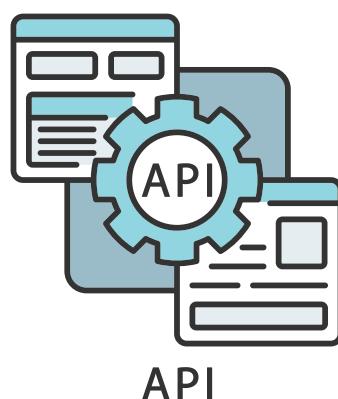
Os aplicativos, ou aplicações, não fazem acesso direto a nenhum componente do sistema — toda a troca de informações e o ligar ou desligar de um componente são realizados pelo SO. Isso permite que as aplicações tenham total transparência quanto aos componentes e, dessa forma, possamos ter diferentes fabricantes e tecnologias empregadas nesses componentes. A aplicação, então, não precisa saber o tipo de memória, tela ou câmera, por exemplo, ela apenas fará uma requisição ao SO, que se encarregará de realizar a ação solicitada com total transparência para a aplicação. O desenvolvedor não precisa se ater, assim, às questões de hardware, basta realizar as solicitações por meio das APIs (*application programming interfaces*) de acesso aos componentes. APIs são um conjunto de funções ou rotinas preestabelecidas para uso por meio de software. As APIs dos sistemas operacionais funcionam como uma ponte entre os componentes e as aplicações, proporcionando a integração entre diferentes sistemas ou componentes.

Os componentes e as aplicações são instalados no SO, sendo controlados por ele. Toda forma de comunicação entre eles deve passar pelo SO, que torna transparentes para ambos (aplicações e componentes) as informações de instalação e configuração, fazendo apenas com que as funções sejam usadas independentemente de características específicas. A API permite, então, que o software da aplicação não precise se preocupar com as características específicas de cada componente, ficando restrita às funções de uso disponibilizadas pelo SO.



Aplicações executadas simultaneamente.

Para ilustrar melhor, uma aplicação pode inicializar a câmera, ligar a luz e tirar uma foto apenas utilizando as APIs, mas as características da câmera e como ligá-la, acender a luz e tirar a foto são ações informadas ao SO pela aplicação, e o sistema fará todas as execuções necessárias, inclusive o armazenamento da foto, sem que a aplicação tenha informações específicas quanto ao modelo e ao fabricante da câmera; posteriormente, a aplicação poderá acessar a foto. Podemos, então, inferir que não precisamos conhecer detalhadamente nenhum componente do sistema de computação, apenas as APIs de controle dos componentes.



API com interface (ponte de comunicação).

Os principais sistemas operacionais para smartphones atualmente são o Android® (Google®) e o iOS® (Apple®). Esses SOs juntos detêm quase todo o mercado para dispositivos móveis (smartphones e tablets).



rvlsoft / Shutterstock.com

Principais sistemas operacionais para smartphones.

Uma pesquisa mostrava há alguns anos que a plataforma mais usada para smartphones era o Android®, com 72% do mercado, com o iOS® em segundo lugar, com 18% dos usuários de smartphones, seguido do BlackBerry®, com apenas 4% do total, e o Windows® Phone, perto de 3% do mercado. Dados mais recentes apontam que o Android® segue na frente com 86,1% do mercado e o iOS® vem em segundo lugar, com 13,7%. Juntos, esses dois sistemas operacionais para smartphones possuem 99,8% do mercado. Os números demonstram que o Android® é o principal sistema operacional para smartphones no mundo.

O desenvolvimento para as duas plataformas segue os mesmos conceitos, diferenciando-se apenas nas ferramentas de desenvolvimento e nas linguagens de programação, mas ambas seguem o padrão MVC para desenvolvimento de sistemas. Veremos os conceitos do modelo MVC, suas características e técnicas de desenvolvimento. Devido ao alto custo necessário para as licenças e a preparação do ambiente de desenvolvimento para aplicativos para iOS®, nosso foco será o desenvolvimento para Android®, que não necessita de investimentos em licenças para a preparação do ambiente. Outro ponto importante para a definição dessa escolha é o mercado, no qual o Android® possui enorme vantagem sobre o iOS®.

O iOS®

O iOS® é o sistema operacional para dispositivos móveis desenvolvido pela Apple®, sendo um ambiente de execução para sistemas de aplicações móveis. Foi desenvolvido especificamente para uso em dispositivos móveis da Apple®, tais como iPhones®, iPads®, iPods® e AppleTVs®, e é uma versão do sistema operacional baseada no Linux®. O desenvolvimento de aplicações para o iOS® não será discutido nesta disciplina.

O Android®

O Android® é o sistema operacional para dispositivos móveis disponibilizado pela Google® e por diversas outras grandes empresas por meio de um consórcio, e é um ambiente para a execução para sistemas de aplicação. É responsável por controlar todos os dispositivos e componentes do sistema. Foi concebido para uso em dispositivos móveis, tais como smartphones e tablets, e é uma versão do sistema operacional Linux® voltada especificamente para uso em dispositivos móveis.

• Histórico do Android®:

- A primeira versão foi lançada em 2008.
- Em 2010, houve um aumento de 707% nas ativações de smartphones com Android® em um único semestre.
- Em 2011, o Android® era responsável por 37% do mercado de smartphones, seguido de 27% do iOS®, da Apple®, e de 22% do BlackBerry®.
- Em agosto de 2010, foram ativados perto de 200.000 smartphones com Android® por dia; já em junho de 2011 foram mais de 500.000, também por dia.

Atualmente, o Android® é desenvolvido tecnologicamente por um consórcio com 81 empresas unidas para impulsionar a inovação na tecnologia móvel, com forte liderança da Google®.

• Vantagens e desvantagens:

O sistema operacional é de código aberto e gratuito, de modo que se pode obter e conhecer seu código-fonte, mas, por outro lado, o código aberto acaba por ser de domínio público, permitindo um alto conhecimento a seu respeito por muitas pessoas e impactando a segurança.

- **Desenvolvimento Android®:**

Os aplicativos Android® são desenvolvidos na linguagem Java®, que é a mais usada no mundo. A linguagem Java® também é aberta e de código livre, sendo usada tanto para criação de aplicações empresariais, serviços web e controles de robôs (Nasa com uso em Marte) como para diversos dispositivos eletrodomésticos e móveis.

Por meio do Java®, podemos usar APIs (interfaces de programação de aplicativos) disponibilizadas pelo Android® diretamente, facilitando o desenvolvimento de aplicativos que usem dispositivos específicos, tais como: câmeras, GPS, vídeos, fotos, áudio (músicas), interfaces gráficas com uso de toques etc. A linguagem Java® permite o acesso a um conjunto de bibliotecas com uma infinidade de códigos prontos para serem usados.

Os dispositivos Android® disponibilizam aplicativos de fábrica que incluem telefone, contatos, correio, navegador, entre vários outros.

- **Principais versões do Android®:**

| Versão | Nome |
|---------------------|------------------------|
| Android 1.0 | Alpha |
| Android 1.1 | Beta |
| Android 1.5 | Cupcake |
| Android 1.6 | Donut |
| Android 2.0/2.1 | Eclair |
| Android 2.2 | Froyo |
| Android 2.3 | Gingerbread |
| Android 3.0/3.1/3.2 | Honeycomb |
| Android 4.0 | Ice Cream Sandwich |
| Android 4.1/4.2/4.3 | Jelly Bean |
| Android 4.4 | KitKat |
| Android 5.0/5.1 | Lollipop |
| Android 6.0: | Marshmallow |
| Android 7.0/7.1 | Nougat |
| Android 8.0/8.1 | Oreo |
| Android 9.0 | Pie |
| Android 10.0 | Q (ainda não definido) |

Você deve ter notado que, a partir da terceira versão, todas possuem um doce como nome.

- **O Google® Android Market (Play Store):**

É uma loja virtual para a aquisição de aplicativos que nada mais é do que um aplicativo próprio para aquisição, instalação e manutenção (atualização) das demais aplicações. Existem aplicativos para diversas áreas e funções, desde financeira, educacional, jogos, entretenimento, científicas, engenharias etc. Você precisará efetuar um cadastro e pagar a licença uma única vez, com um baixo custo, e poderá disponibilizar seus aplicativos na Play Store, que podem ser pagos ou gratuitos, ainda havendo a possibilidade de você incluir cobranças dentro do aplicativo.



Saiba mais

Características para o desenvolvimento de aplicações para o Android

DEITEL, P. J.; DEITEL, H. M.; WALD, A. **Android 6 para programadores:** uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2016. Minha Biblioteca. cap. 1, seções 1.4 a 1.8, p. 6-10.



**MIDIA
TECA**

Acesse a midiateca da Unidade 1 e veja o conteúdo complementar indicado pelo professor sobre os sistemas operacionais Android® (Google®) e iOS® (Apple®) para dispositivos móveis.

O ambiente de desenvolvimento de aplicativos para Android®

O Android® Studio – AS

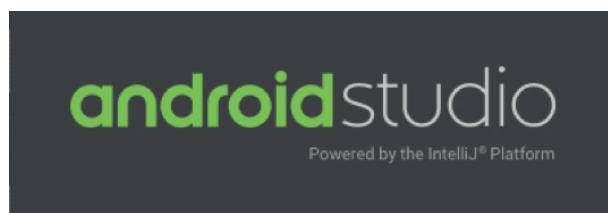
Existem várias ferramentas para o desenvolvimento de aplicativos para a plataforma Android®, entre os quais o Eclipse (com o *plug-in* ADT – Android Development Tools) dominou durante muito tempo, mas o Android® Studio, no momento, é a principal ferramenta para o desenvolvimento de aplicativos para essa plataforma. A ferramenta é disponibilizada gratuitamente (licença Apache 2.0) pela comunidade de desenvolvimento Android®, sendo uma IDE (*integrated development environment*) de desenvolvimento nativo para Android® da Google® com versões para Windows®, Mac® e Linux® (incluindo o Chrome OS®).

Antes da instalação do Android® Studio é importante que você possua uma versão do Java® JDK instalado em seu computador. Lembre-se de que o desenvolvimento nativo para o Android® é realizado em Java®. Você ainda poderá usar a linguagem Kotlin®, derivada do Java®.

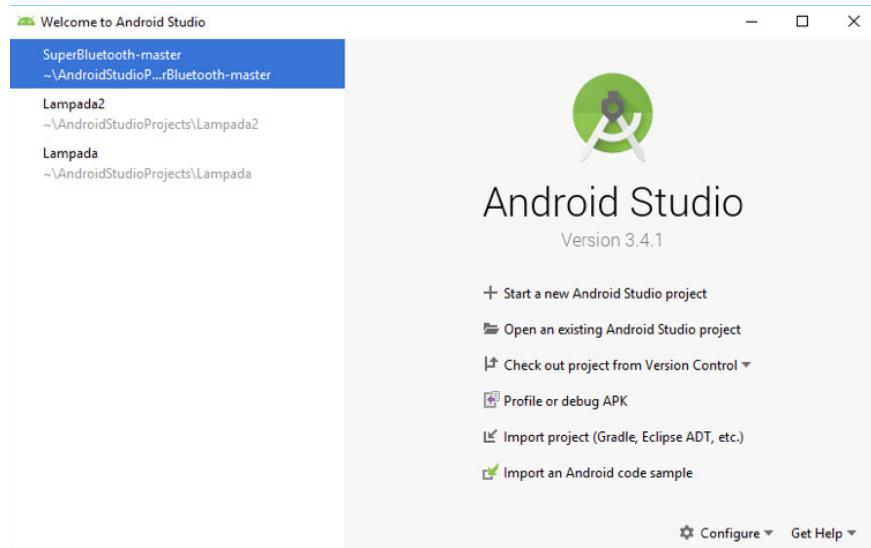
Você pode realizar o download do Java® JDK a partir do endereço: <https://www.oracle.com/technetwork/java/javase/downloads/index.html>, basta você identificar a última versão do JDK para download. Siga as instruções e reinicie o computador após a instalação para assegurar que as variáveis de ambiente sejam atualizadas e que o Android® Studio tenha acesso ao JDK durante sua instalação.

Você pode realizar o download do Android® Studio em: <https://developer.android.com/studio>. Após realizar o download e ter se certificado de que você já possui o JDK instalado, siga as instruções de instalação do Android® Studio.

Após a instalação, você já poderá iniciar a ferramenta:



Tela de carregamento do Android® Studio.

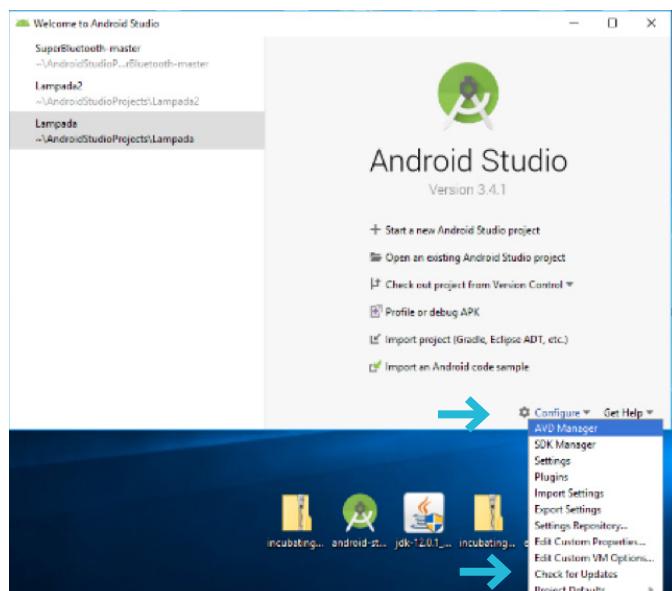


Tela inicial do Android® Studio caso você tenha fechado o último projeto trabalhado.

Se você não tiver fechado o último projeto, o Android® Studio abrirá diretamente o ambiente de desenvolvimento.

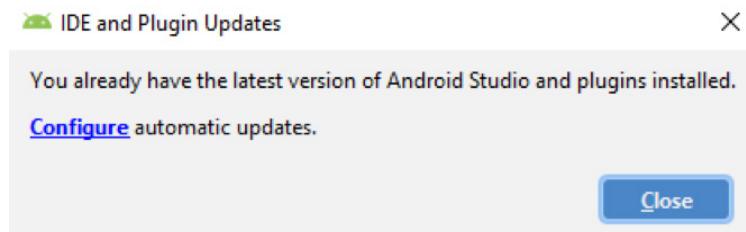
É muito importante manter o Android® Studio atualizado, pois quase sempre existem atualizações a serem realizadas. O AS verifica a necessidade de atualizações periodicamente, mas nem sempre essa periodicidade é suficiente, por isso devemos, sempre que possível, verificar se não existem novas atualizações.

Clique sobre “Configure” e escolha “Check for Updates”.



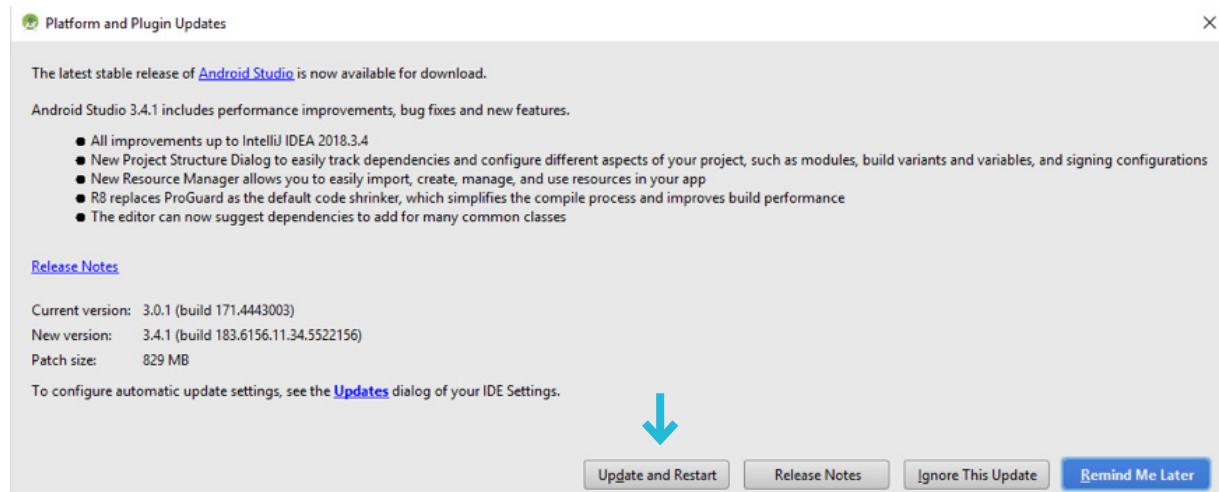
Para realizar a atualização manual do Android® Studio.

Será feita uma verificação via internet, e, caso seja necessária alguma atualização, ela será apresentada a você; do contrário, você receberá uma mensagem de que seu AS está atualizado.



Mensagem informando que o Android® Studio está atualizado.

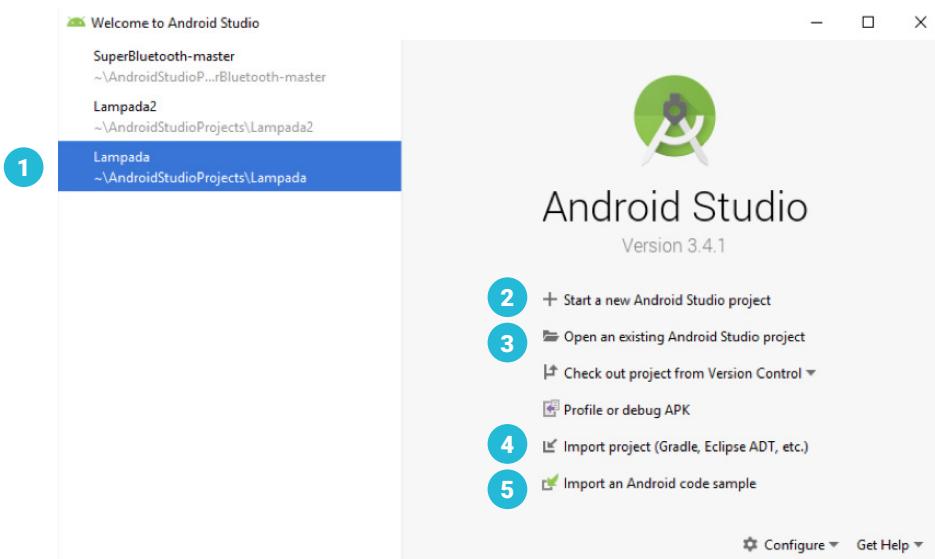
Sendo necessária a atualização do Android® Studio, clique sobre “Update and Restart” e siga as instruções de atualização, assim o Android® Studio irá reiniciar após a atualização. Esse processo poderá ser repetido algumas vezes quando tivermos várias atualizações a serem realizadas.



Mensagem informando que o Android® Studio precisa ser atualizado.

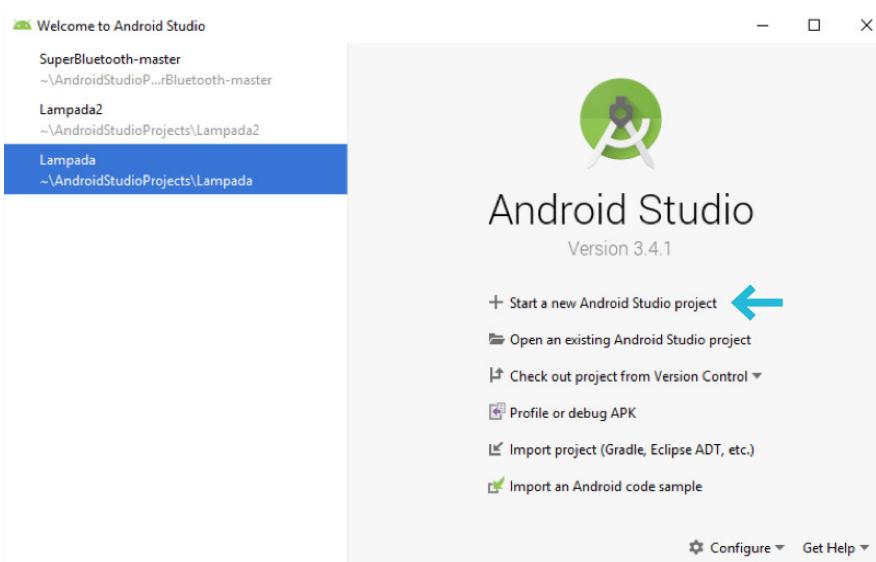
Nesta tela inicial, você pode realizar diversas ações, tais como:

1. Abrir rapidamente algum dos últimos projetos desenvolvidos, bastando clicar sobre ele.
2. Criar um novo projeto.
3. Abrir um projeto diretamente do local onde ele está armazenado.
4. Importar um projeto desenvolvido em outra ferramenta.
5. Importar um exemplo prático.



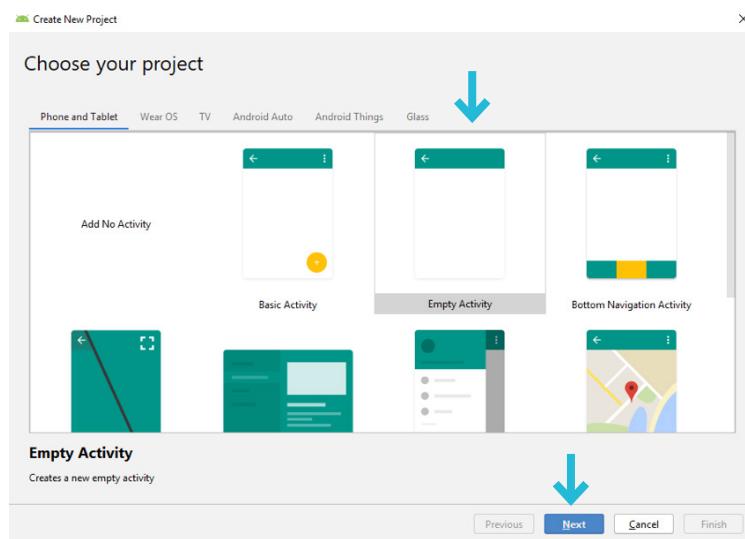
As principais opções disponíveis do AS.

Vamos criar um novo projeto e conhecer o ambiente de desenvolvimento do AS. Para isso, escolha a opção “+ Start a new Android Studio project”.



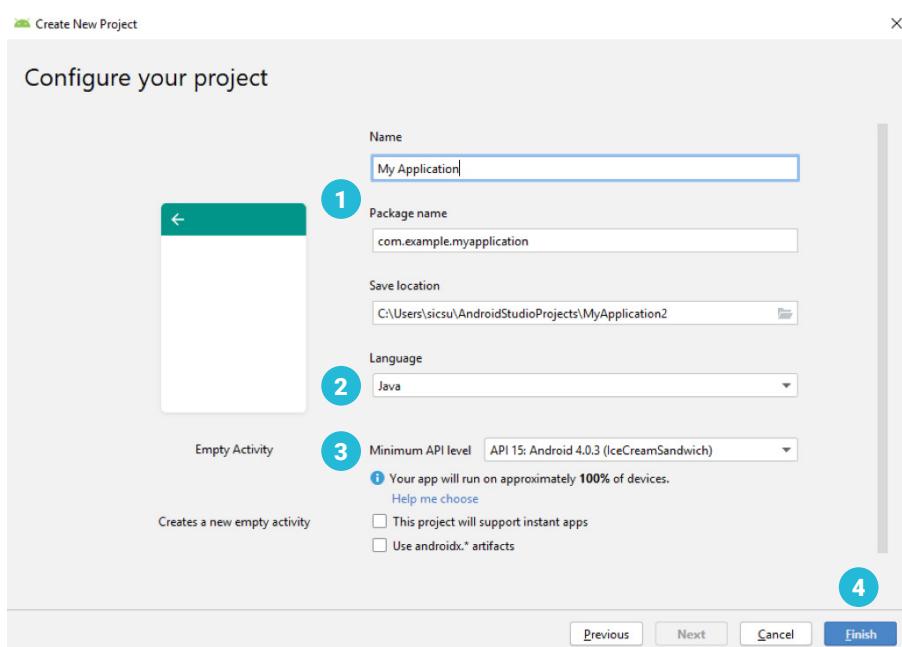
Criação de um novo projeto no AS.

Vamos iniciar um projeto simples escolhendo “Empty Activity” e “Next” para continuar.



Para criação de um projeto simples: “Empty Activity”.

Configuração do novo projeto:

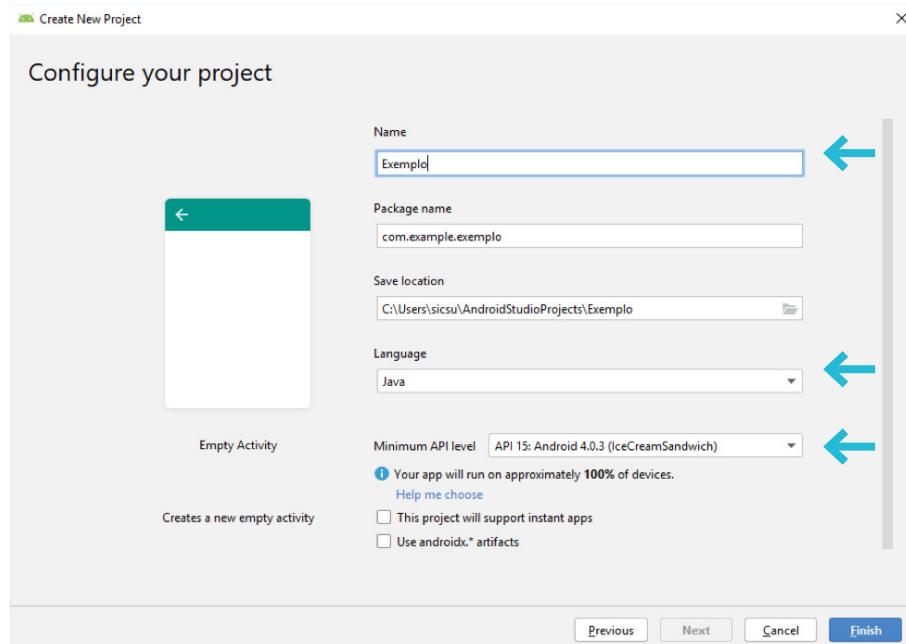


1. Nome do projeto: podem ser usados espaços e símbolos, mas estes serão eliminados no nome do pacote e no nome do diretório do projeto.
2. Você poderá escolher a linguagem de programação entre Java® e Kotlin®, mas nós manteremos nosso projeto com a linguagem Java®.
3. Minimum API level: determina a compatibilidade do aplicativo com as versões do Android®. É importante compreender que usar uma versão muito nova impedirá a instalação de seu aplicativo em versões mais antigas, diminuindo a quantidade de possíveis usuários. Somente use versões mais recentes como mínimas para os

casos em que sejam necessários componentes ou características específicas de novas versões e leve em consideração a possível redução de usuários aptos a usar o aplicativo.

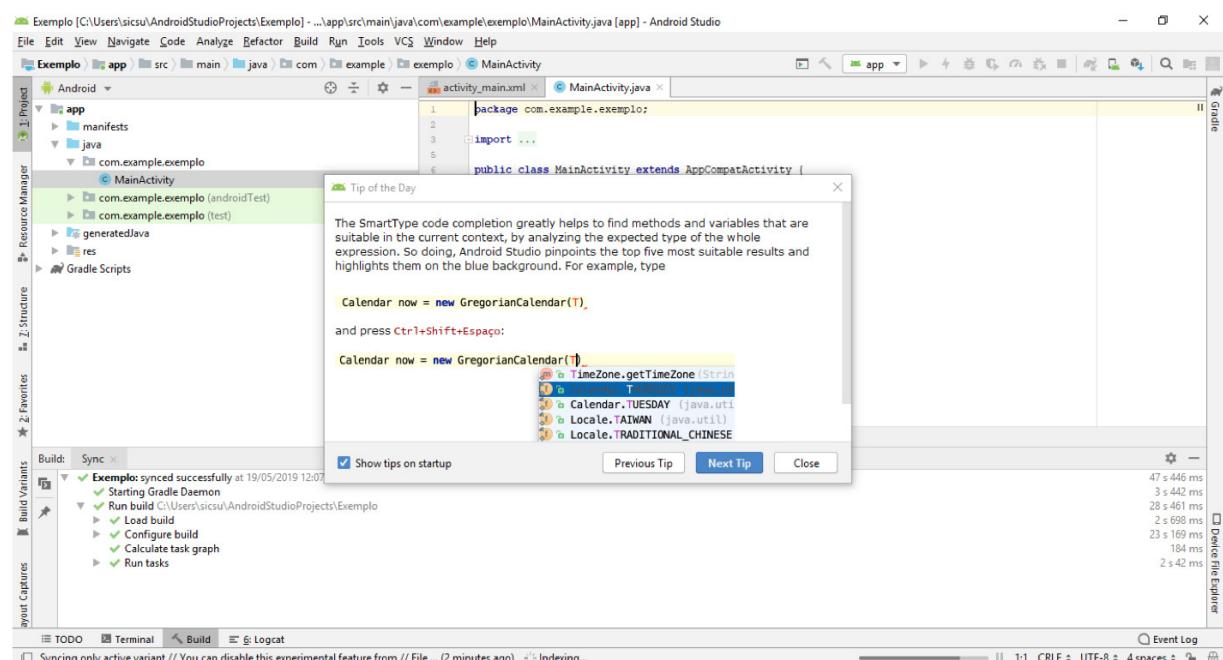
4. Clique em “Finish” para criar o projeto.

Criaremos o projeto **Exemplo** utilizando a linguagem Java®, e ele será compatível com a API 15 (Ice Cream Sandwich):



Criação do projeto **Exemplo**.

O Android® Studio pode lhe dar dicas sempre que você o iniciar:

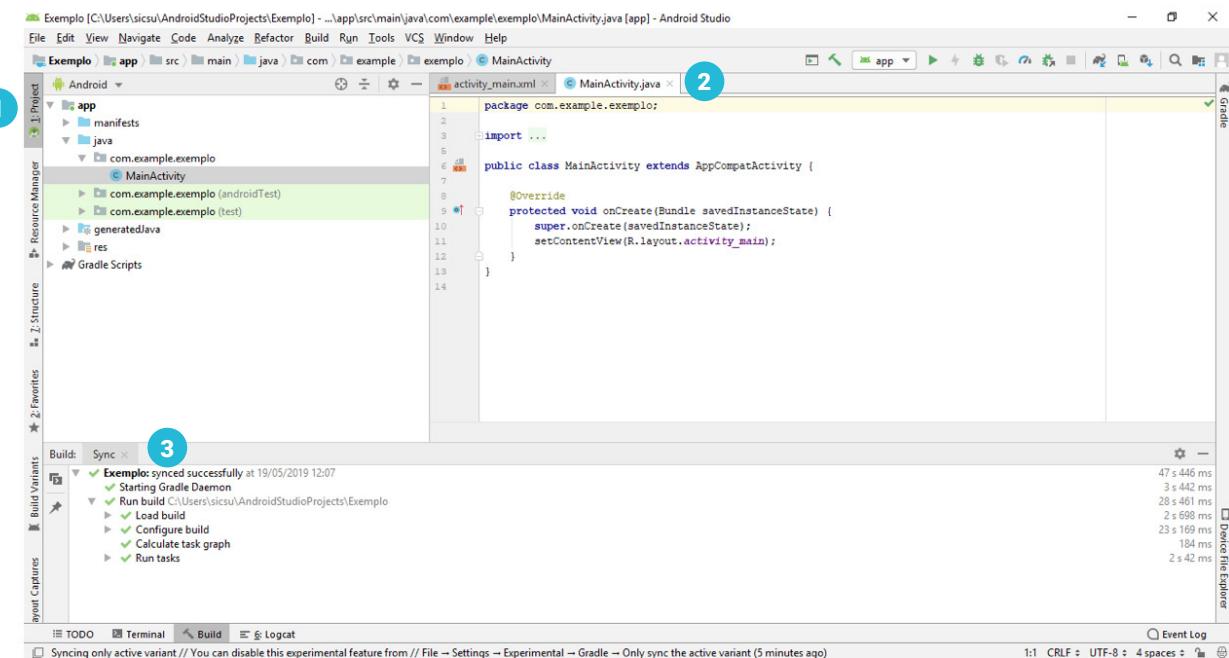


Tela inicial com dicas de uso do Android® Studio.

Conhecendo o ambiente de desenvolvimento

- Programação:

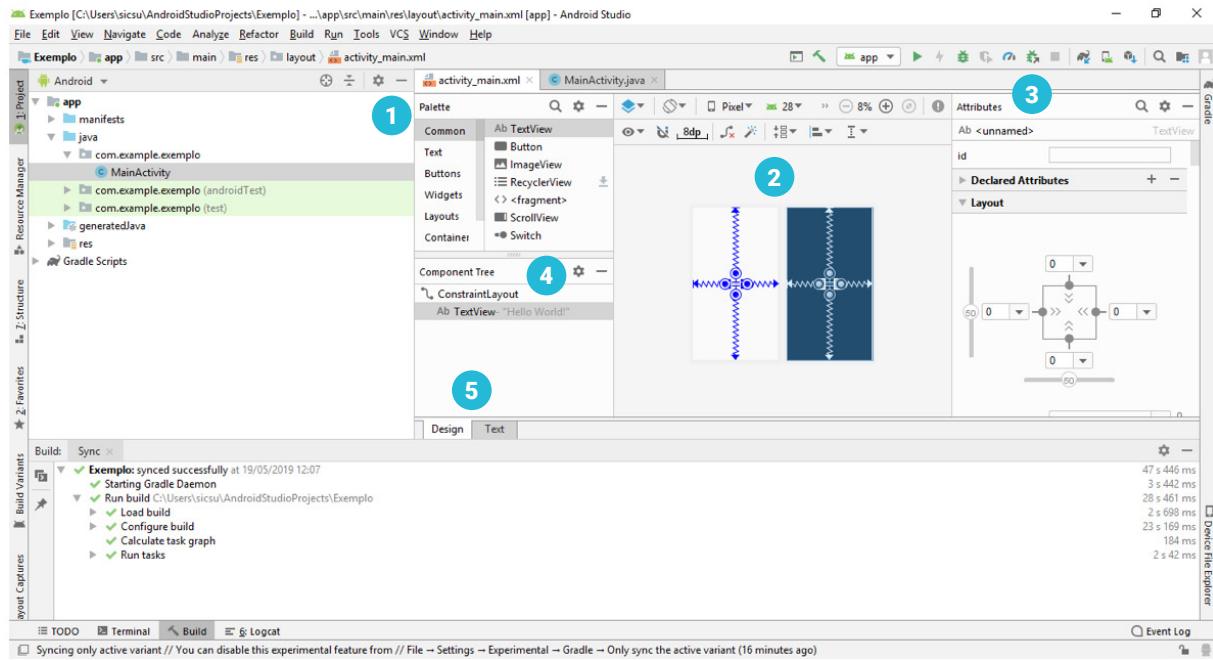
1. Arquivos do projeto.
2. Editor de código.
3. Acompanhamento do projeto.



Ambiente de codificação.

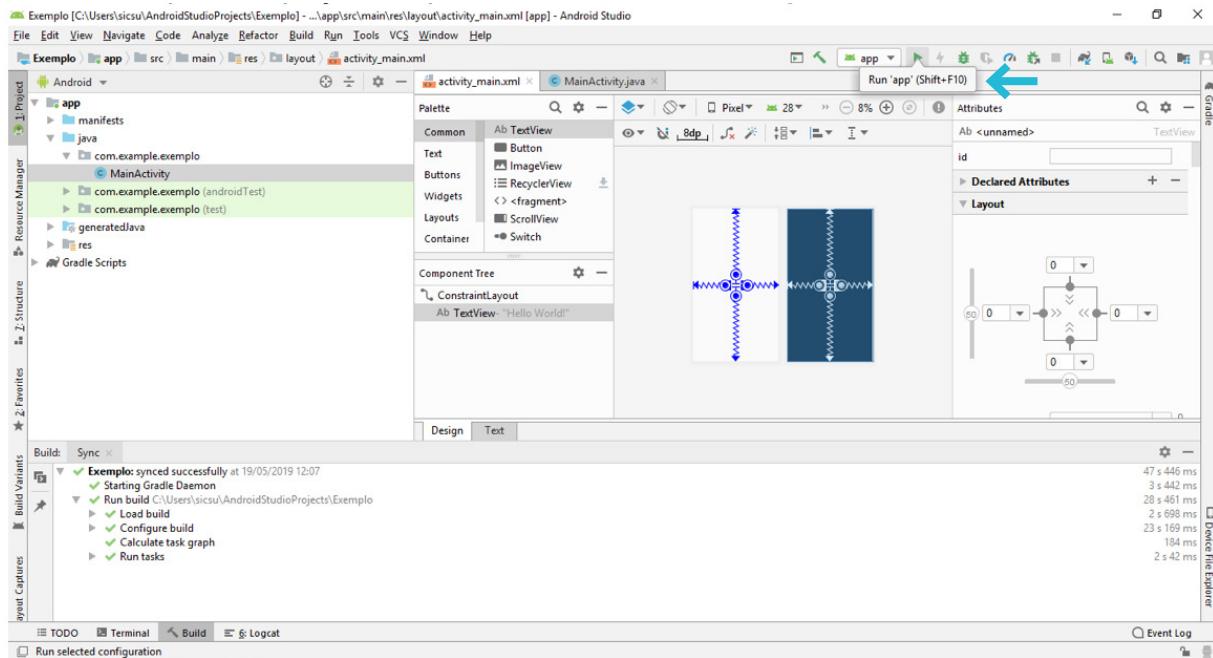
- Criação da tela (view): você deve trocar a aba da Atividade Java® (codificação) para a XML®, que configura a interface gráfica da atividade (view):

1. Palette: contém os componentes disponíveis para a criação de telas, separadas por grupos.
2. Editor gráfico.
3. Atributos do componente selecionado.
4. Component Tree: apresenta hierarquicamente os componentes da tela. É muito útil para posicionar os componentes em determinados tipos de layout.
5. Design × Text: alterna a apresentação da tela entre os formatos gráficos e de código.



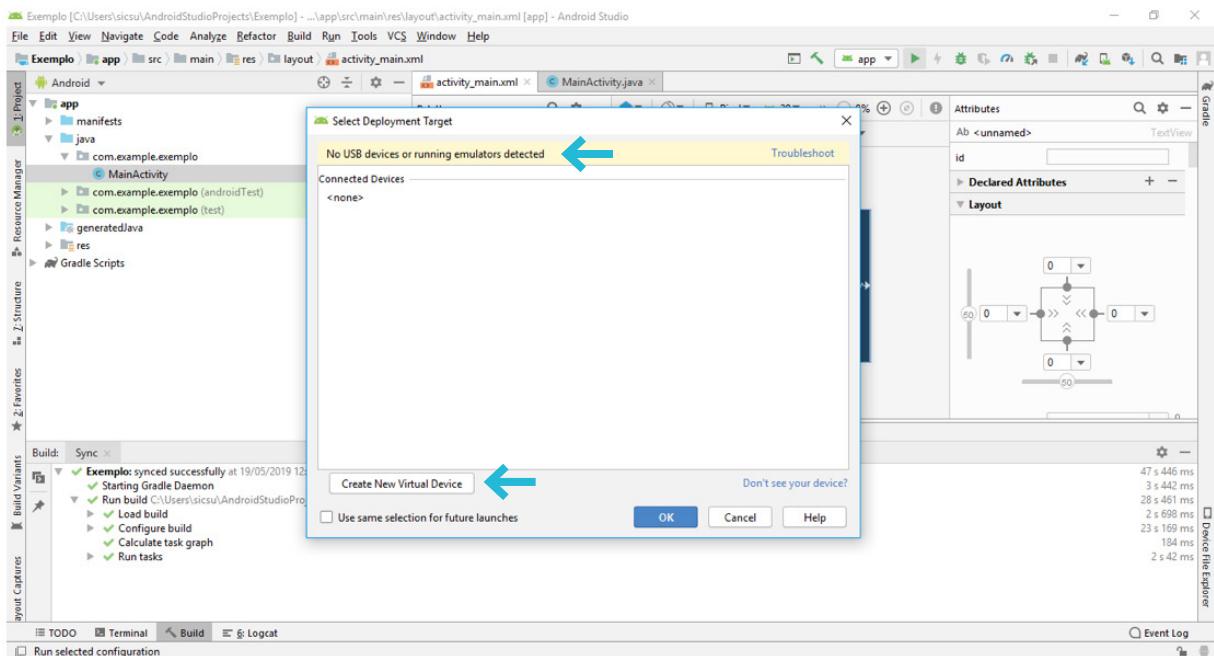
Ambiente de configuração da tela.

Para testar seu primeiro projeto, clique sobre o ícone de execução:



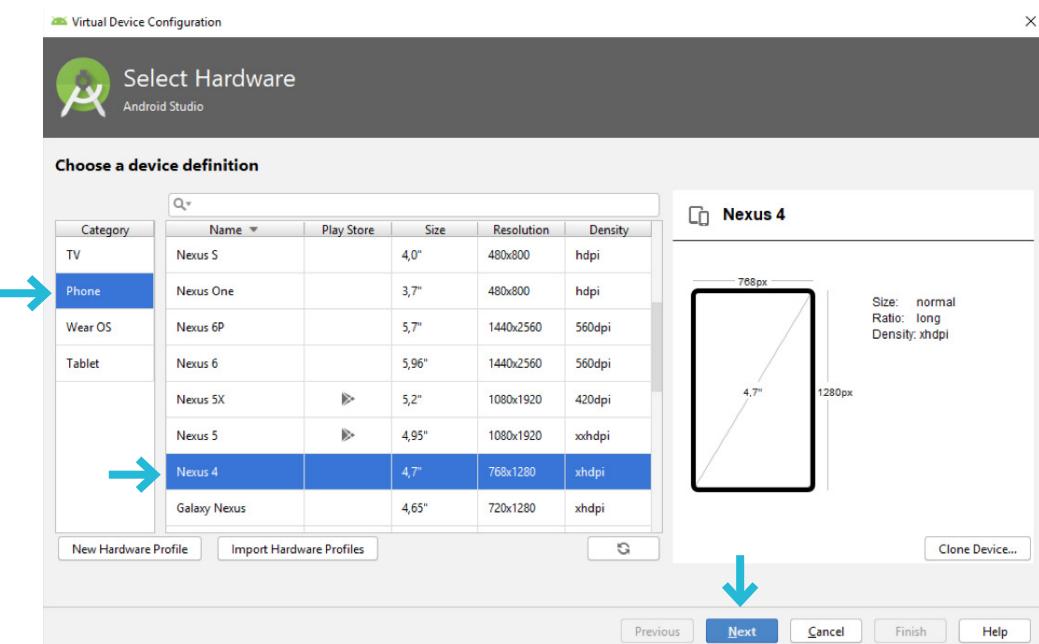
Execução do projeto ("Run 'app'").

Para a execução, é necessário que você tenha um emulador (aparelho virtual) ou um smartphone liberado para o modo desenvolvedor. A tela a seguir permite que você escolha o aparelho no qual será realizado o teste. Como não há aparelho real ou emulador disponível, você deverá criar um aparelho virtual. Um aparelho real ou virtual será exibido aqui.



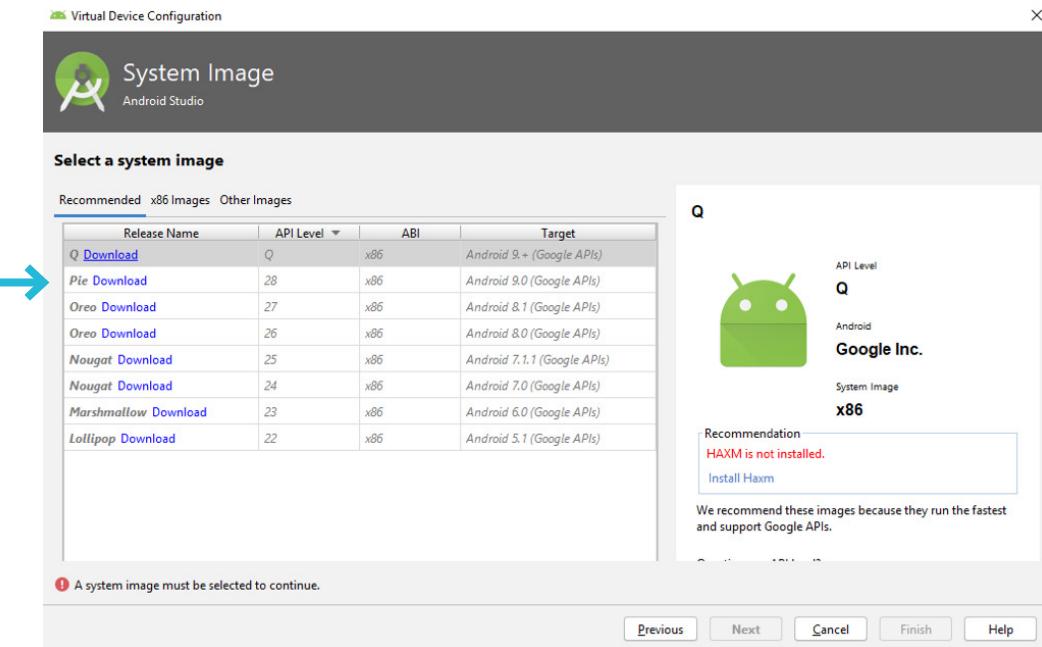
Definição do aparelho de teste, virtual ou real.

Para criar um aparelho virtual (*virtual device*), você poderá escolher entre “TV”, “Phone”, “Wear OS” ou “Tablet”. Vamos manter selecionado “Phone” e escolher um Nexus 4, clicando em “Next” para criar:



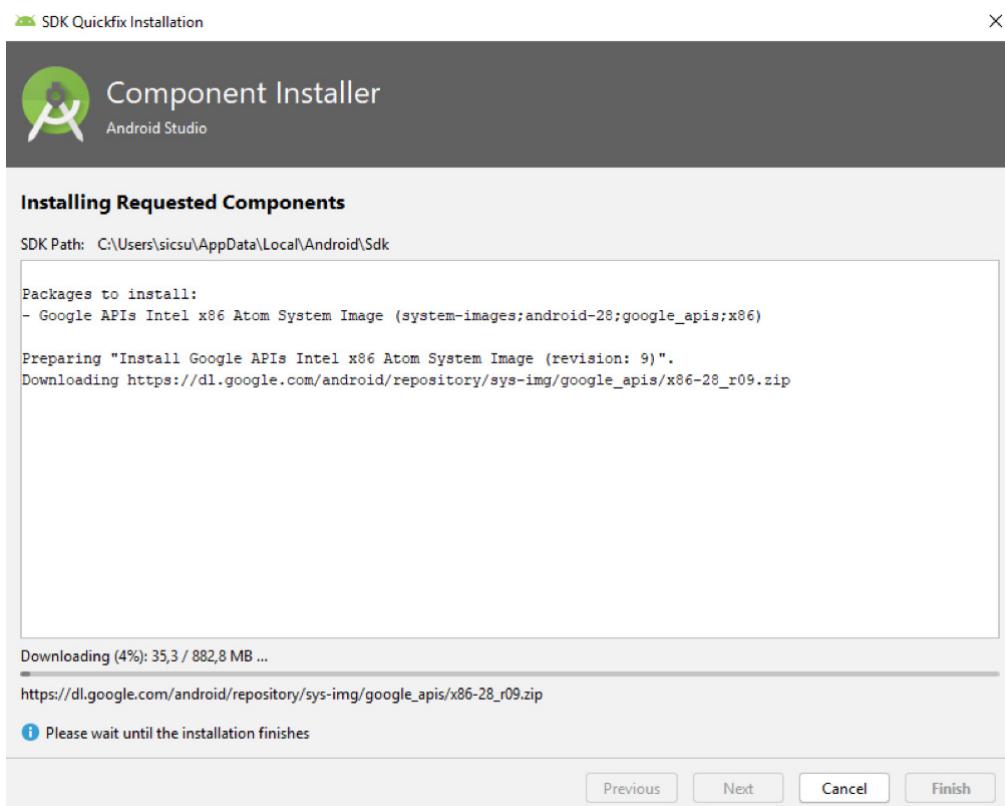
Escolha “Phone” para usarmos o Nexus 4 em nossos primeiros testes.

Agora, você deve escolher a versão (API Android) que irá usar. Como a Q está em final de desenvolvimento, você poderá escolher a Pie (API 28) para realizar o download, uma vez que não estão disponíveis outras versões já instaladas.



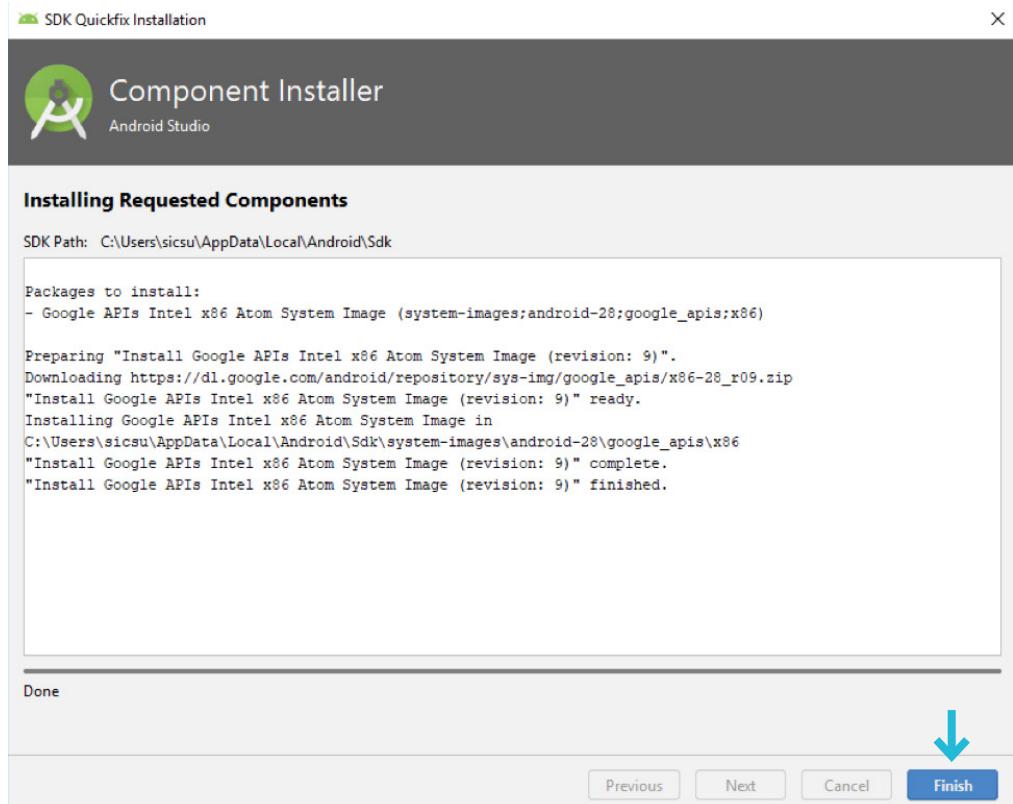
Escolha da versão Android® que será usada no emulador.

Aguarde o download e a instalação (o acesso à internet deverá estar liberado).



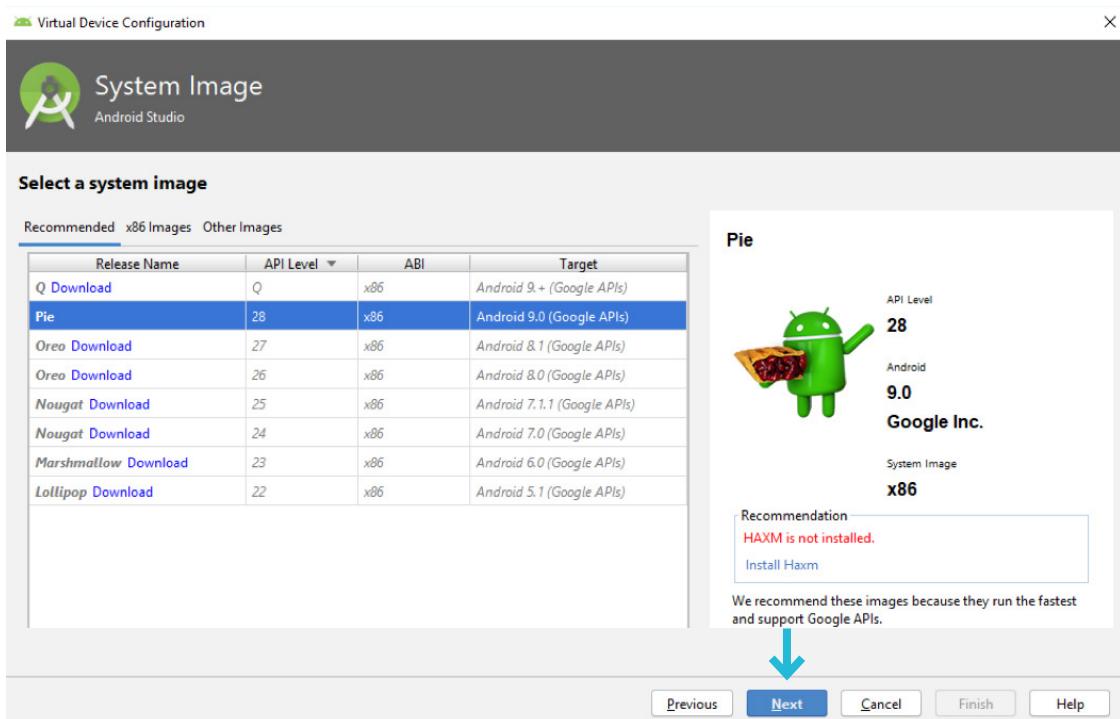
Download e instalação do Android 9 (API 28) para uso no emulador.

Após a realização do download, você poderá encerrar o processo de criação do emulador clicando no botão "Finish".



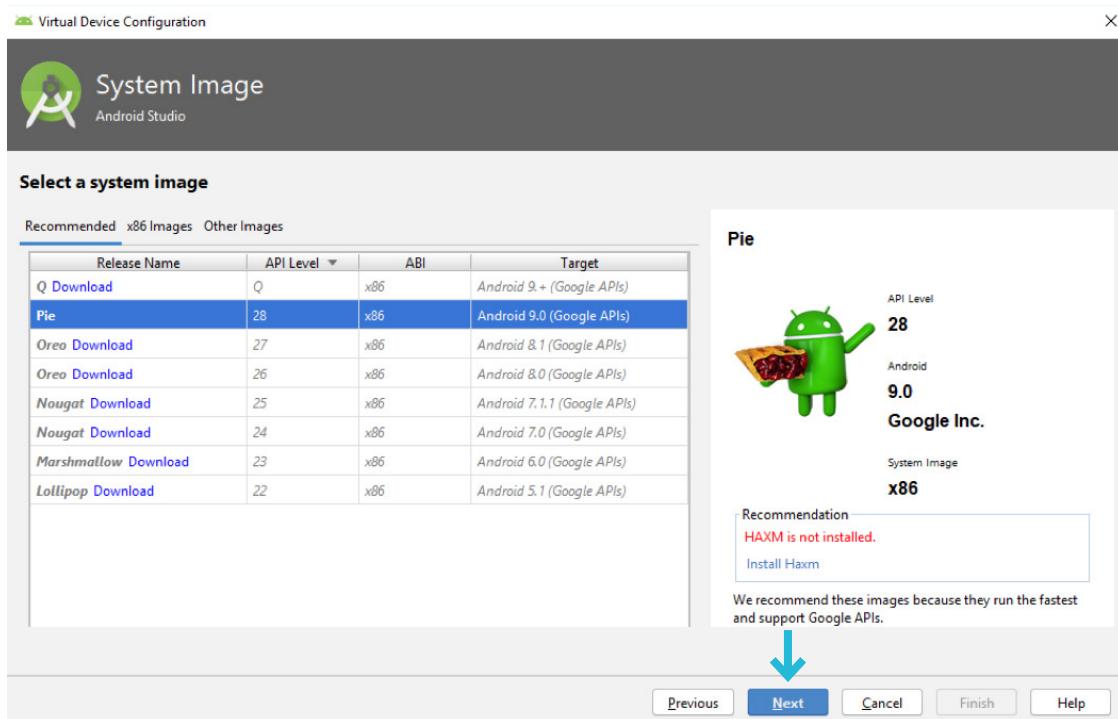
Complete a instalação da versão do Android® que será usada no emulador.

Selecione a versão que você acabou de instalar e clique em “Next”.



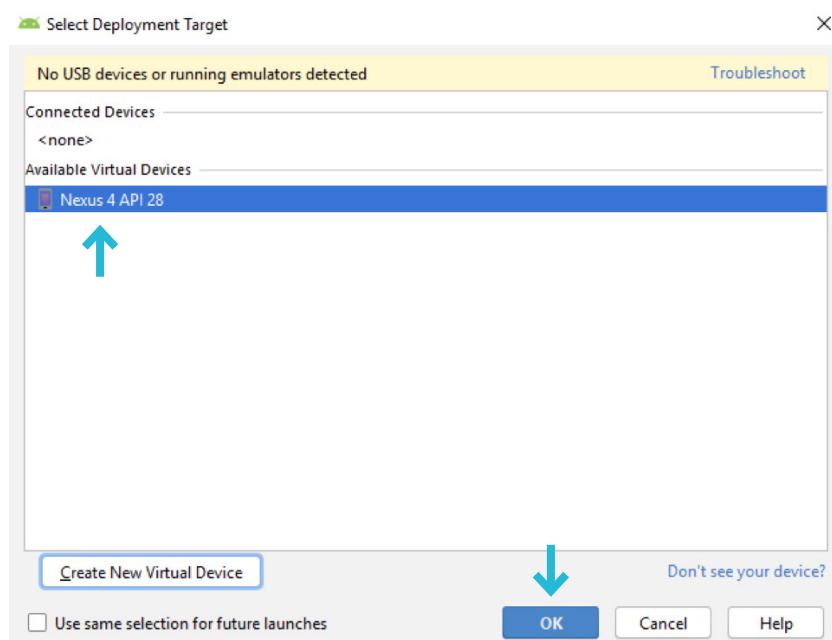
Defina a versão do Android® a ser usada em seu emulador.

Para finalizar, o AS irá sugerir um nome para o dispositivo virtual que lhe informe o aparelho e a versão do Android® instalada. Você poderá finalizar a criação do emulador clicando no botão “Finish”.



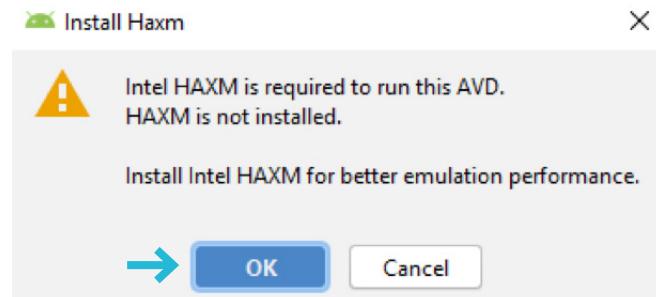
Tela final da criação do emulador.

O emulador criado se encontrará disponível para uso. Selecione o emulador e clique em “OK”.



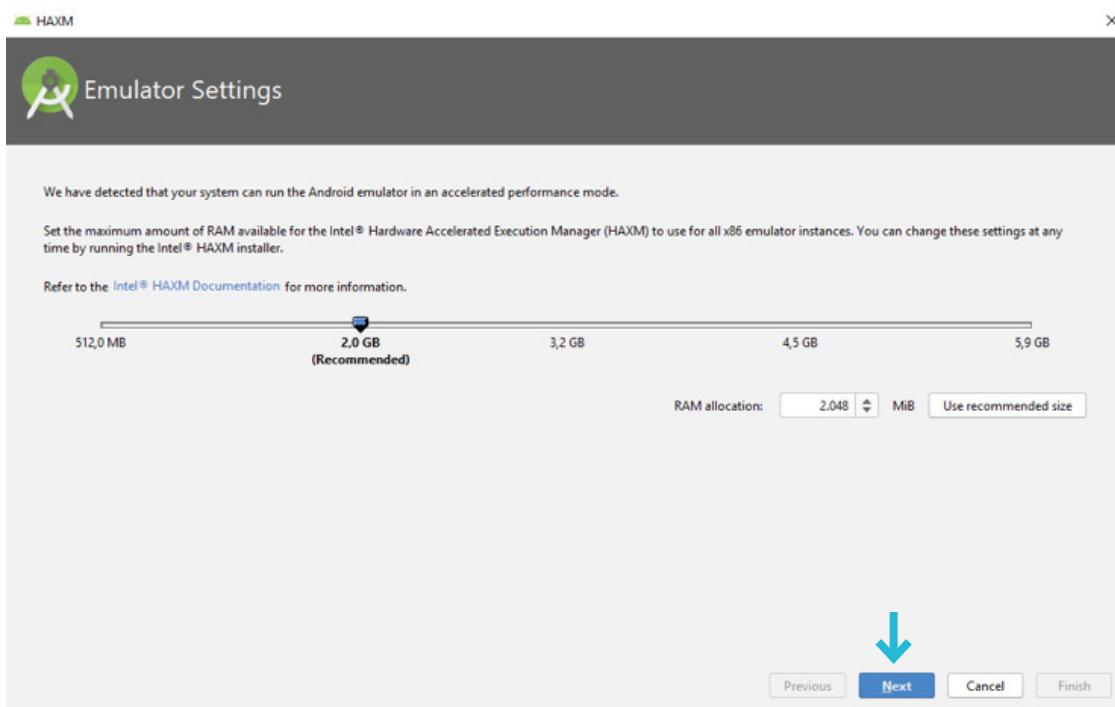
Escolha do emulador para a realização do teste.

A utilização do Intel® HAXM pode melhorar o desempenho, mas em alguns casos pode inviabilizar o uso do emulador (implica alterações, inclusive, na BIOS do computador em algumas circunstâncias). Faça a instalação apenas se você tiver certeza de que não terá problemas. Você poderá clicar em “OK” para instalar caso tenha certeza.



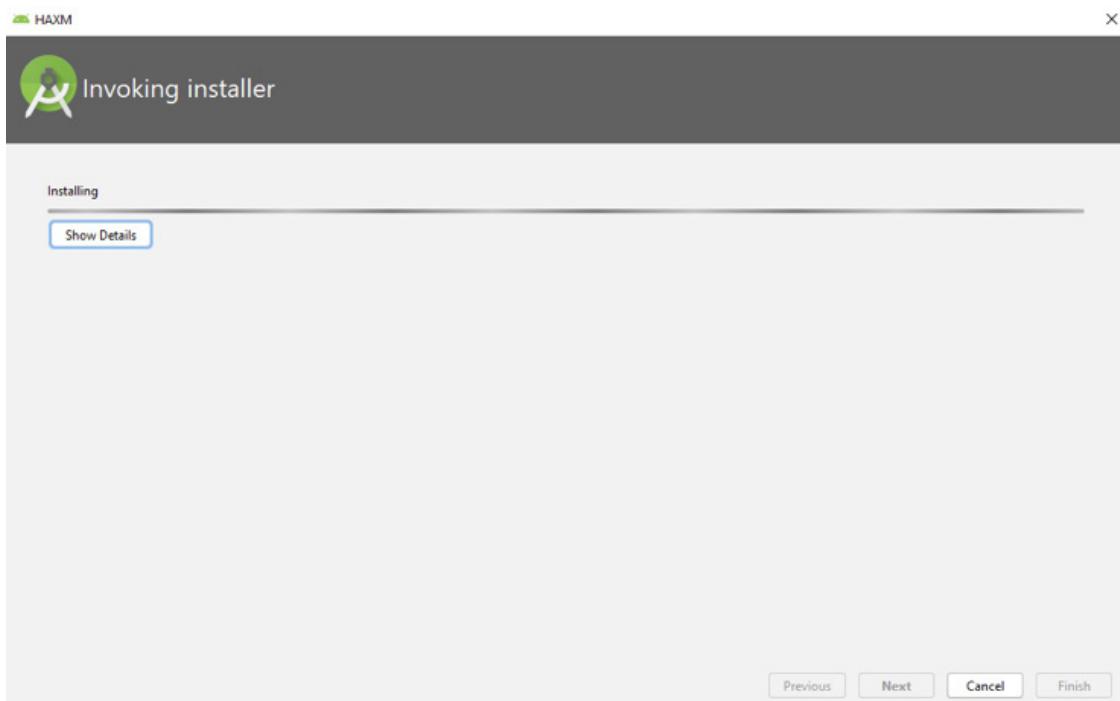
Tela de instalação do Intel® HAXM.

Defina a configuração de memória de seu emulador e continue a execução do projeto clicando em “Next”. Você poderá manter a versão padrão.



Configuração da memória do emulador.

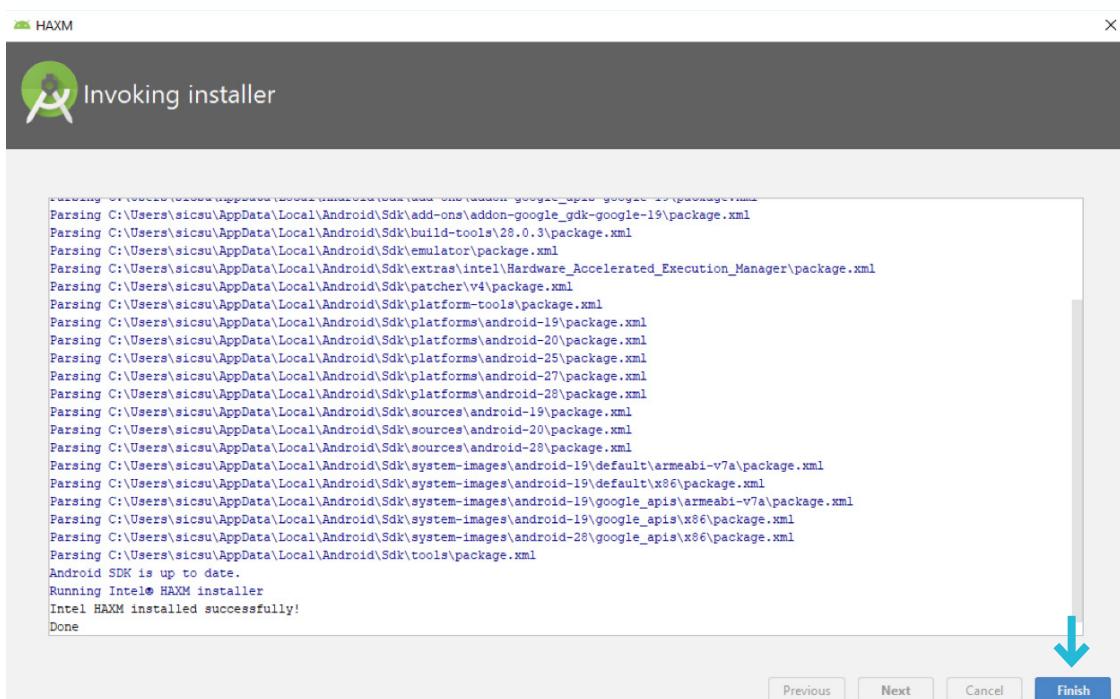
Aguarde a instalação.



Tela de instalação do Intel® HAXM.

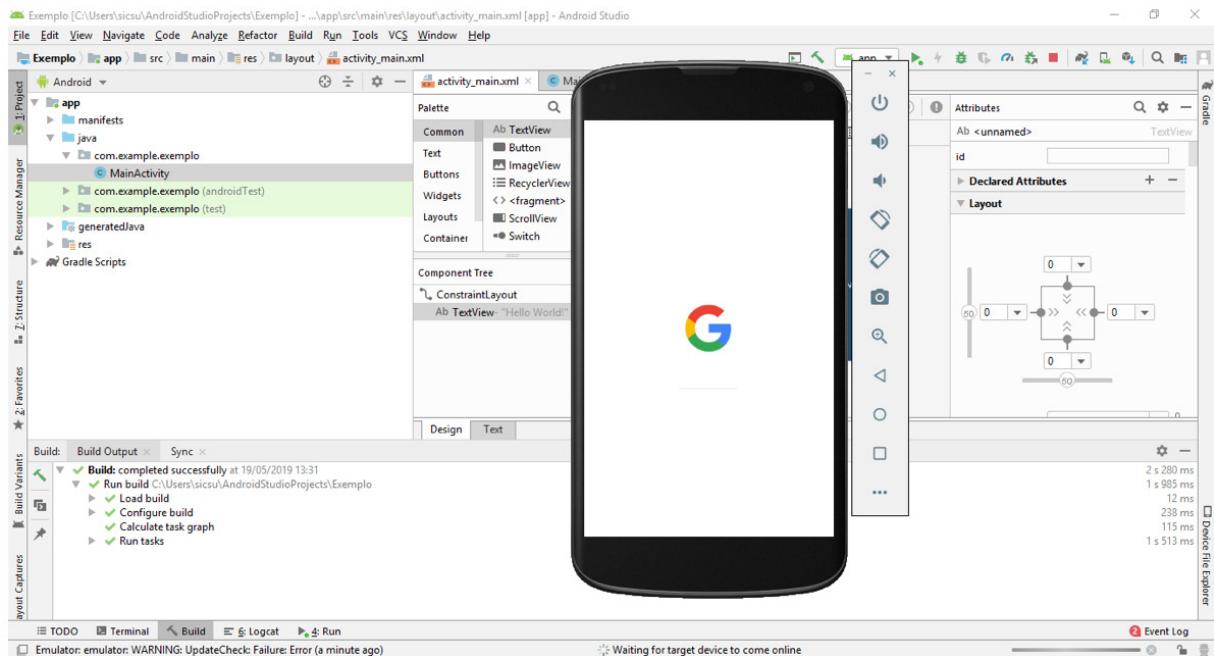
Pode ser que o Windows® exija permissão para essa instalação.

Após a instalação, clique em “Finish” para completar o processo.



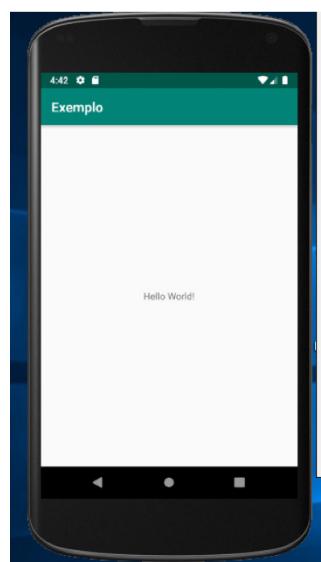
Complete a instalação do Intel® HAXM.

O AS irá preparar o emulador e executá-lo. Esse processo poderá demorar um pouco; não encerre a execução do emulador até terminar seu trabalho. Caso precise realizar novos testes, você não precisará iniciar esse processo (iniciar o emulador) demorado novamente – o AS e o emulador são executados paralelamente, mas em sincronismo.



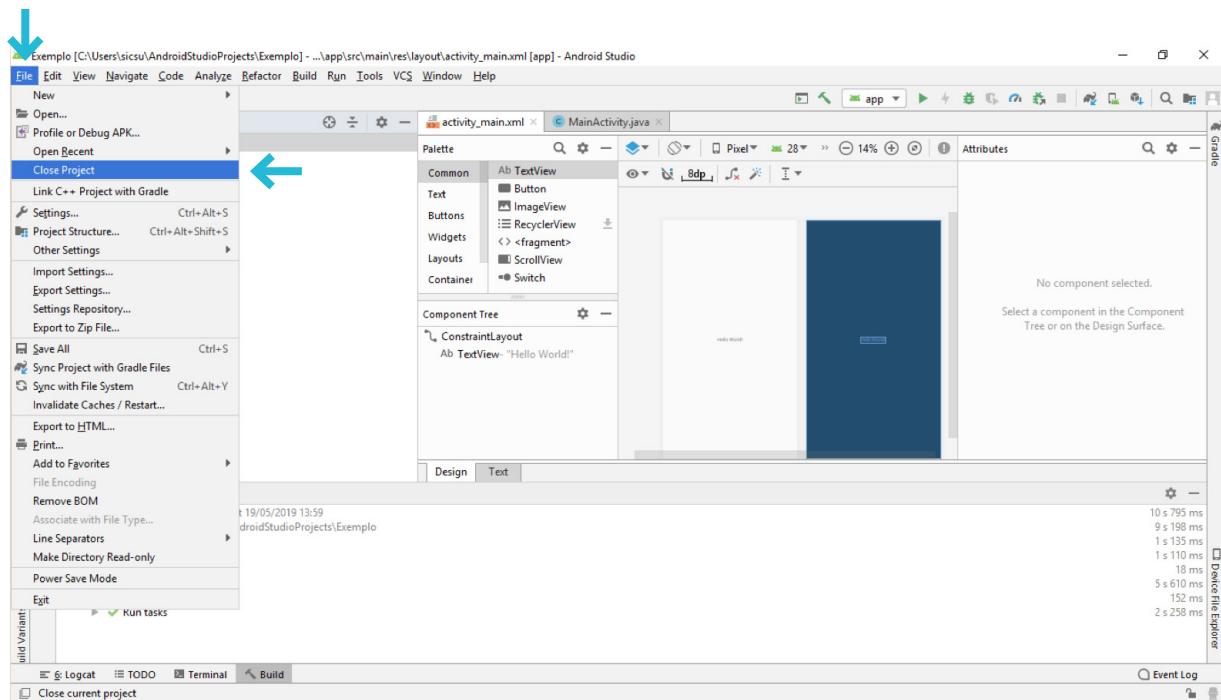
Execução simultânea do AS e do emulador.

Pronto, você já poderá ver seu aplicativo em funcionamento!



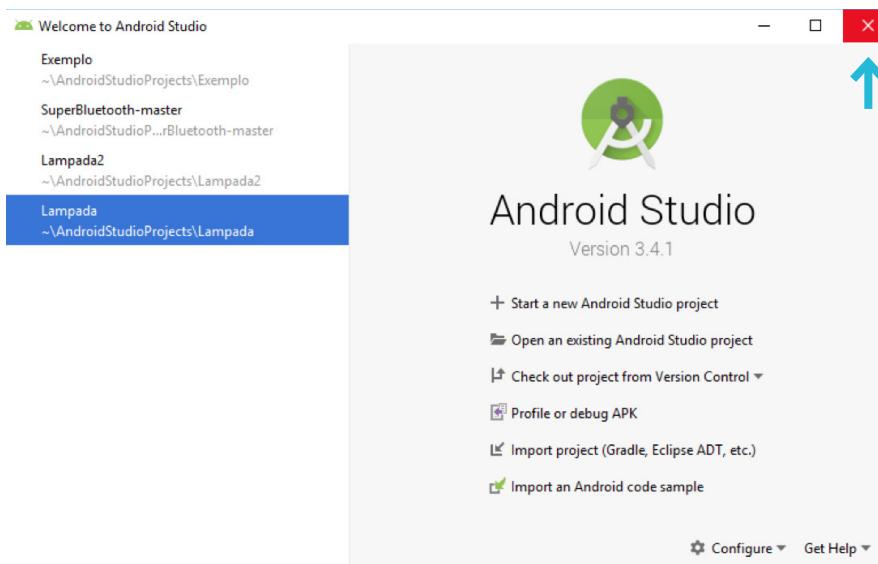
Emulador com o projeto em execução para testes.

Você poderá fechar o projeto clicando em “File” e, em seguida, “Close Project”.



Fechamento do projeto.

Depois, é só fechar o Android® Studio.



Encerramento do Android® Studio.



Saiba mais

Trabalhando com um projeto existente (exemplo prático)

DEITEL, P. J.; DEITEL, H. M.; WALD, A. **Android 6 para programadores**: uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2016. Minha Biblioteca. cap. 1, seção 1.9, p. 21-30.



Dica

O Android Developer é um site dedicado ao desenvolvimento de aplicativos para Android® e é uma grande fonte de informações, últimas notícias, tutoriais e documentação. Faça uma visita em: <https://developer.android.com/>.



MEDIATECA

Acesse a midiateca da Unidade 1 e veja o conteúdo complementar indicado pelo professor para você acompanhar o processo de instalação.



NA PRÁTICA

Você compreendeu o que são sistemas operacionais e suas características, preparou seu ambiente de desenvolvimento instalando e configurando o Android® Studio e aprendeu a usar esse programa para criação e testes de um projeto de exemplo.

Resumo da Unidade 1

Nesta unidade, você conheceu a história dos dispositivos móveis ao longo da evolução dos aparelhos pessoais e dos celulares. Compreendeu, também, o que é um sistema operacional e suas características e conheceu os dois principais sistemas operacionais para dispositivos móveis. Por fim, você aprendeu a instalar e a configurar o Android® Studio, assim como a criar um projeto e a testá-lo em um emulador criado e configurado por você.



CONCEITO

Histórico do desenvolvimento de aplicações para sistemas móveis; o que são sistemas operacionais e suas características; e o ambiente de desenvolvimento para Android®.

Referências

DEITEL, P. J.; DEITEL, H. M.; WALD, A. **Android 6 para programadores**: uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2016. Minha Biblioteca.

_____ ; _____; DEITEL, A. **Android**: como programar. Porto Alegre: Bookman, 2015. Minha Biblioteca.

UNIDADE 2

O modelo MVC e a criação
de atividades no Android®

INTRODUÇÃO

Atualmente, o MVC tem se destacado como modelo em camadas para o desenvolvimento de sistemas. Ele visa separar as camadas de interface com o usuário (*view*) da camada de dados e lógica do sistema (*model*) da camada de fluxo de controle (*control*) da aplicação. Também é importante a compreensão sobre o ciclo de vida de uma atividade, que interage diretamente com o sistema operacional para controlar as diversas aplicações que estão sendo usadas simultaneamente. Dessa forma, nesta unidade teremos a oportunidade de criar uma aplicação simples com a intenção de dominar a ferramenta de desenvolvimento, o Android® Studio.



OBJETIVO

Nesta unidade, você será capaz de:

- Compreender e aplicar o modelo de desenvolvimento de sistemas MVC.
- Conhecer e aplicar os métodos relacionados ao ciclo de vida de uma atividade.
- Criar um projeto de aplicação prática para uso em dispositivos móveis.

O modelo MVC

O modelo MVC (*model-view-control*) é usado para facilitar o desenvolvimento de sistema com alta complexidade e grande número de requisitos. Ele foi criado por Trygve Reenskaug e concebido para o desenvolvimento de aplicações em camadas com a separação de suas diferentes funcionalidades. Esse modelo é definido em três camadas e tem grande afinidade com o desenvolvimento de aplicações para a web. A ideia era separar a interface, normalmente desenvolvida em HTML, da camada de controle de fluxo, normalmente em JavaScript, da camada de negócios com seus respectivos modelos no servidor.

Model

Essa camada determina todo o processamento do sistema, sendo responsável pela persistência dos dados, separando-os, e pela lógica de negócios das demais camadas. Permite que os dados e a lógica de negócios possam ser acessados por meio de diferentes interfaces. Essa camada também é comumente chamada de “modelo” e representa o modelo da aplicação.

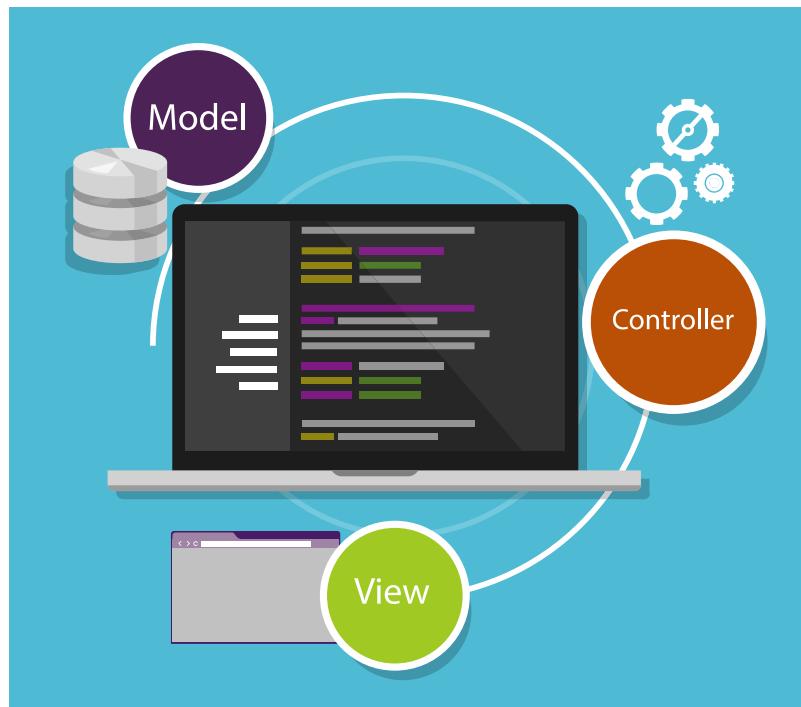
View

Essa camada determina a interação com o usuário, sendo responsável pelas interfaces com o usuário dentro do sistema. Deve ser capaz de receber os dados e as requisições do usuário e repassá-los ao sistema, que posteriormente retornará o resultado do processamento ao usuário. Não deve interferir na lógica de negócios, mas deve realizar críticas tanto em relação ao tipo de dados como sua formatação e a validade para o sistema (verificar se o dado informado está dentro dos valores esperados). Todos os envios e os retornos de dados devem ser realizados pelo controlador (control).

Control (controller)

Essa camada determina como ocorrerão as transferências de dados entre a interface (view) e o modelo (model), sendo responsável pelas trocas de mensagens entre o usuário e o sistema. Realiza o controle sobre o conjunto de processos do sistema. Também pode ser chamado de “objetos de negócios”. Recebe os dados de entrada da interface com o usuário (view), envia-os para o sistema (model) e retorna ao usuário as informações geradas pelo sistema.

Podemos observar na figura a seguir que a camada de controle (*controller*) realiza a troca de mensagens (comunicação de dados) entre as camadas de interface com o usuário (*view*) e de modelo (*model*).

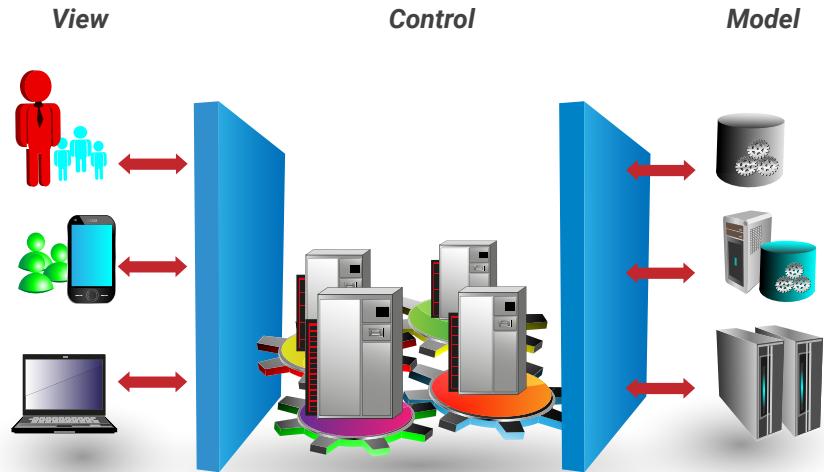


Camadas do modelo MVC.

Na figura a seguir, podemos observar com clareza a separação entre cada camada, mas é possível termos mais de uma camada em um mesmo local físico.

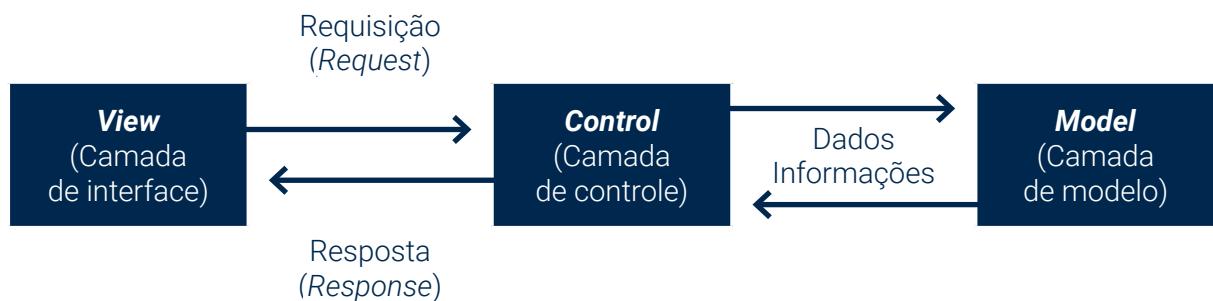
Uma aplicação pode ter:

- As três camadas locais, com interface (*view*), controle (*control*) e modelo (*model*) com armazenamento (persistência) de dados local (banco de dados ou arquivos de configuração). Temos, ainda, aplicações mais simples que podem não necessitar de armazenamento de dados.
- Duas camadas locais, como as camadas de interface (*view*) e controle (*control*) locais, mas com acesso remoto à camada de modelo (*model*). Essa é a forma mais comum de desenvolvimento de aplicações para dispositivos móveis.
- Uma camada local com a interface (*view*) e acesso remoto a um mesmo servidor, responsável pelas camadas de controle (*control*) e de modelo (*model*).



Separação das camadas do modelo MVC.

Também é importante entender como ocorre o fluxo de troca de mensagens (dados) realizada em sistemas baseados no modelo MVC. A figura a seguir representa como esse fluxo de dados acontece no MVC.



Fluxo de mensagens no modelo MVC.

1. Usuário faz uma requisição (request) por meio da camada de interface (view).
2. A requisição é tratada pela camada de controle (control) e enviada à camada de modelo (model).
3. A camada de modelo (model) trata os dados da requisição e prepara a informação de resposta ao usuário.
4. A informação gerada pela camada de modelo (model) é recebida pela camada de controle (control), que prepara a resposta (response) e a encaminha para a camada de interface (view).
5. A interface recebe a resposta (response) e exibe ao usuário a informação.

- **Vantagens:**

- Com a independência das camadas, a camada de modelo pode ser reutilizada quando temos a necessidade de criar aplicações capazes de serem utilizadas em diferentes ambientes de execução. Podemos criar uma aplicação para web e para dispositivos móveis, mantendo um único modelo, capaz de atender às duas aplicações, web e *mobile*.
- O desenvolvimento e a manutenção do sistema podem ficar a cargo de diferentes equipes, otimizando e especializando o desenvolvimento.
- Determinadas mudanças em uma camada podem não afetar as demais.
- Vários *frameworks* de desenvolvimento implementam o modelo MVC, tais como: J2EE, Struts, .Net, entre outros.
- Está em conformidade com a arquitetura cliente × servidor.
- Pode realizar transações distribuídas.
- O sistema é escalável e portável.
- Aumenta a segurança no acesso e na manipulação da base de dados.

- **Desvantagens:**

- Dificulta o desenvolvimento de aplicações de pequeno porte.
- Necessita de treinamento adequado para as equipes de desenvolvimento.
- Pode aumentar o tempo de análise e desenvolvimento.
- É necessária grande atenção para que a equipe implemente corretamente os conceitos do MVC.

O modelo MVC atualmente é utilizado pela maioria das ferramentas de desenvolvimento, seja para o desenvolvimento de aplicações para desktop, web ou dispositivos móveis.



MEDIATECA

Acesse a midiateca da Unidade 2 e veja o conteúdo complementar indicado pelo professor a respeito dos conceitos do modelo MVC.



Saiba mais

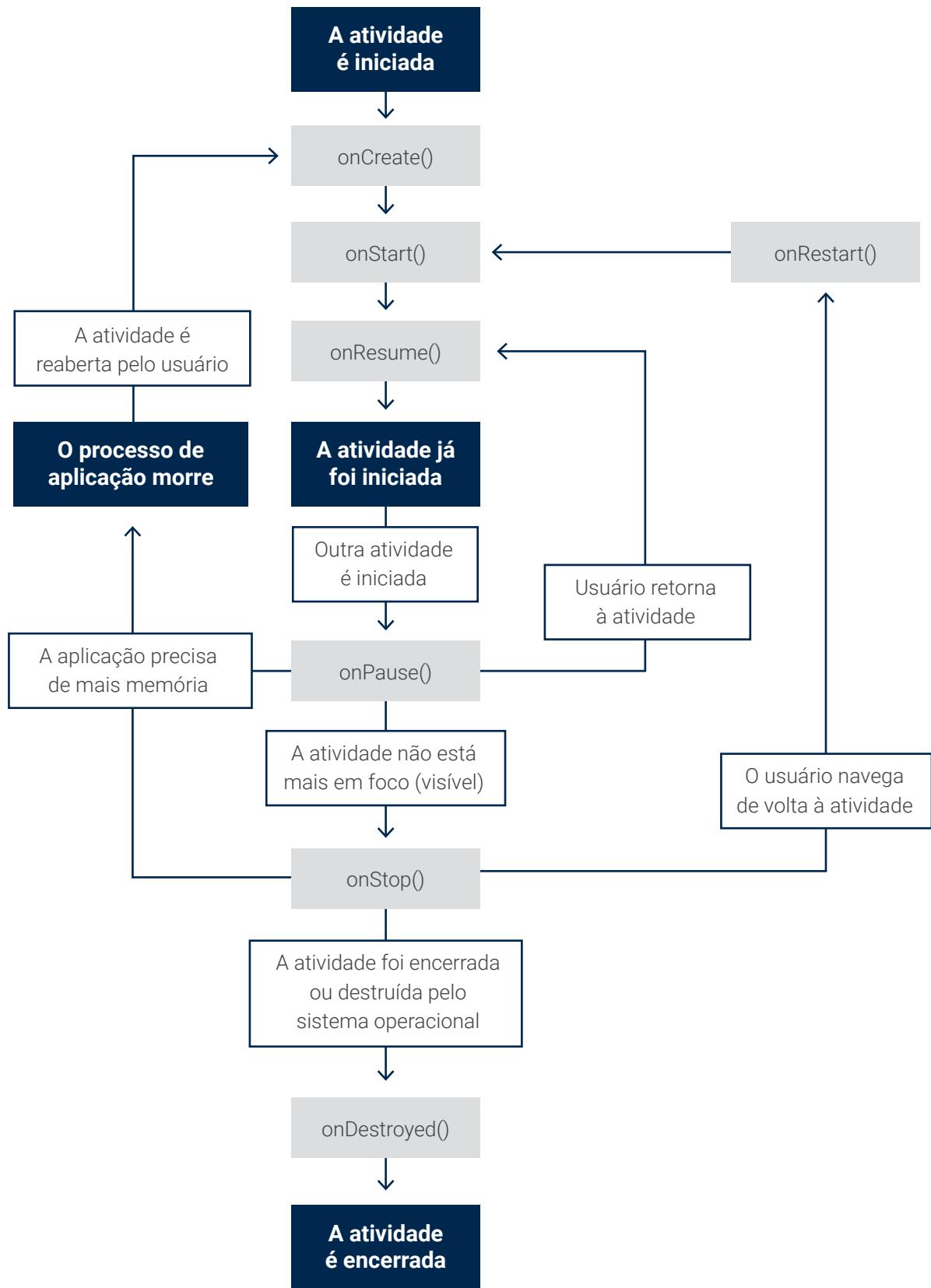
Modelos de desenvolvimento em camadas

BOND, M. et al. **Aprenda J2EE em 21 dias**. São Paulo: Pearson Education do Brasil, 2003. Biblioteca Virtual Pearson. cap. 1, p. 3-10.

O ciclo de vida de uma atividade

Uma atividade no Android® (activity) fornece uma tela (view) para a construção de interface com o usuário. Ao iniciarmos um novo projeto, ou se incluirmos uma atividade nova em um projeto Android® existente, são incluídos dois arquivos: o primeiro é um arquivo XML para a configuração da view, e o segundo é um arquivo .java para o controle da atividade (controller). É possível criarmos várias views em um projeto com apenas um controller, mas isso não é prático e dificulta muito a manutenção.

Toda atividade possui um ciclo de vida, no qual um conjunto de sete métodos controlam a execução da aplicação em sincronia com o sistema operacional. Essa característica permite que possamos controlar a aplicação durante todo o tempo em que ela está em execução. Por exemplo, você pode salvar o estado da aplicação caso o usuário interrompa sua aplicação e troque para outra atividade. Você também pode salvar dados e estado no caso de o usuário resolver encerrar a aplicação sem salvar anteriormente alguma informação importante. Esses métodos permitem, então, que possamos controlar as interações do usuário não somente com as nossas aplicações, mas também da nossa aplicação com o sistema. Dessa forma, é importante conhecer esses métodos e, principalmente, entender quando cada um deles ocorre. Para isso, analise a figura a seguir para ter a possibilidade de entender a dinâmica da execução desses métodos de controle da atividade.



Esquema do fluxo de controle do ciclo de vida de uma atividade no Android®.

| Método | Descrição | Próximo |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|
| onCreate() | É a primeira função a ser executada em uma atividade (<i>activity</i>). Geralmente, é a responsável por carregar os layouts XML e outras operações de inicialização, além de definir todas as configurações da atividade. Esse método recebe um <i>bundle</i> (pacote com os dados do estado anterior). É executada apenas uma vez e é a atividade mais importante, porque é responsável pela criação da atividade. Muitas aplicações exigem apenas esse método para executar. | onStart() |
| onStart() | É chamada imediatamente após a <i>onCreate()</i> e imediatamente antes de a atividade se tornar visível. Pode ser executada, também, quando uma atividade (<i>activity</i>) que estava em segundo plano (<i>background</i>) volta a ter foco. | onResume() ou onStop() |
| onResume() | Assim como o <i>onStart()</i> , esse método é chamado na inicialização da atividade (<i>activity</i>) e também quando uma atividade volta a ter foco. Qual é a diferença? O método <i>onStart()</i> só é chamado quando a atividade não está visível e volta a ter o foco, já o método <i>onResume()</i> é chamado sempre que o foco é retomado. | onPause() |
| onPause() | É o primeiro método a ser chamado quando o sistema está mudando de atividade, sendo invocado quando a atividade perde o foco (isso ocorre quando uma nova atividade é iniciada ou quando da mudança para outra aplicação). | onResume() ou onStop() |
| onStop() | Esse método é chamado sempre que a atividade não está mais visível, quando uma atividade fica completamente encoberta por outra. | onRestart() ou onDestroy() |
| onDestroy() | É o último método a ser executado. Depois dele, a atividade é considerada “morta”, ou seja, não poderá mais ser retornada. Se o usuário voltar à execução da aplicação, uma nova atividade será criada. | Nenhum |
| onRestart() | Esse método é chamado imediatamente antes do <i>onStart()</i> e logo após a atividade ter sido reiniciada. Isso ocorre somente após a atividade ter sido interrompida anteriormente e quando a atividade volta a ter o foco depois de estar em <i>background</i> . | onStart() |

Tempos de vida (conjuntos de métodos a serem executados):

- **Tempo de vida da atividade (entire lifetime):** é o tempo decorrido entre a chamada do método `onCreate()` e a chamada do método `onDestroy()`. Dessa forma, temos o tempo em que a atividade ficou em execução no sistema.
- **O tempo de vida visível da atividade (visible lifetime):** é o tempo decorrido entre a chamada do método `onStart()` e a chamada do método `onStop()`. Dessa forma, podemos obter o tempo em que o usuário tem visibilidade sobre a atividade e interage com ela.
- **O tempo de vida em primeiro plano da atividade (foreground lifetime):** é o tempo decorrido entre a chamada do método `onResume()` e a chamada do método `onPause()`. Dessa forma, podemos obter o tempo em que a atividade ficou à frente das demais atividades no sistema.

Para você compreender melhor, crie uma aplicação Android® no Android® Studio tal como você aprendeu na nossa última unidade.

Na classe **Controller** (.java), inclua a seguinte biblioteca:

```
import android.widget.Toast;
```

Ainda nessa classe, substitua o método `onCreate()` pelos métodos a seguir:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toast toast = Toast.makeText(getApplicationContext(),
        "A atividade foi CRIADA!!!", Toast.LENGTH_LONG);
    toast.show();
}
protected void onStart() {
    Toast toast = Toast.makeText(getApplicationContext(),
        "A atividade INICIOU !!!", Toast.LENGTH_LONG);
    toast.show();
    super.onStart();
}
protected void onRestart() {
    Toast toast = Toast.makeText(getApplicationContext(),
        "A atividade REINICIOU !!!", Toast.LENGTH_LONG);
    toast.show();
    super.onRestart();
}
protected void onResume() {
```

```

Toast toast = Toast.makeText(getApplicationContext(),
        "A atividade RETORNOU !!!", Toast.LENGTH_LONG);
toast.show();
super.onResume();
}

protected void onPause() {
    Toast toast = Toast.makeText(getApplicationContext(),
        "A atividade entrou em PAUSA !!!", Toast.LENGTH_LONG);
toast.show();
super.onPause();
}

protected void onStop() {
    Toast toast = Toast.makeText(getApplicationContext(),
        "A atividade PAROU !!!", Toast.LENGTH_LONG);
toast.show();
super.onStop();
}

protected void onDestroy() {
    Toast toast = Toast.makeText(getApplicationContext(),
        "A atividade ENCERROU !!!", Toast.LENGTH_LONG);
toast.show();
super.onDestroy();
}

```

Notas:

1. A biblioteca *Toast* é responsável pela criação de mensagens flutuantes sobre a tela e será utilizada para a apresentação das mensagens para o usuário durante a execução de cada método.
2. Algumas mensagens poderão não ser apresentadas quando do encerramento da aplicação, pois esta poderá não ter tempo suficiente para exibi-las.
3. Como os métodos de controle estão sendo recriados na subclasse, é necessário realizar a chamada ao método original da superclasse, que já contém a codificação necessária para a interação com o sistema operacional.
4. Utilize o emulador já criado no tutorial da unidade anterior para realizar os testes e não se esqueça de alternar entre as aplicações no Android® para identificar as mensagens que estão sendo exibidas e associá-las à execução do método relacionado.



MIDIA TECA

Acesse a midiateca da Unidade 2 e veja o conteúdo complementar indicado pelo professor sobre o ciclo de vida das atividades no Android®.



Saiba mais

O ciclo de vida das atividades

DEITEL, P. J.; DEITEL, H. M.; WALD, A. **Android 6 para programadores:** uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2016. Minha Biblioteca. cap. 2, seções 3.3.1 a 3.3.3, p. 76-78.

Construção de um aplicativo

A criação de uma aplicação prática requer conhecimentos básicos da ferramenta de desenvolvimento e das camadas de interface (view) e controle (controller). Vamos iniciar com a criação de uma aplicação simples, para começarmos a dominar o Android® Studio, em que criaremos uma interface simples com alguns dos componentes mais comuns em uma interface e compreenderemos como realizar o relacionamento dos componentes da nossa interface com o código de controle (controller) em Java®. Iremos construir, agora, uma aplicação para realizar o cálculo do índice de massa corporal – IMC de uma pessoa e apresentar o valor do índice e a situação do indivíduo em relação a seu peso e sua altura. O IMC é calculado de acordo com a fórmula:

$$\text{IMC} = \frac{\text{peso}}{\text{altura}^2}$$

Já a situação da pessoa é definida de acordo com as tabelas a seguir, baseadas no sexo do indivíduo, sabendo-se que pessoas com menos de 15 anos não devem ter a situação definida por elas.

Tabela: Situação para o sexo feminino.

| IMC | Situação |
|-------------------|-----------------------------|
| IMC < 19,1 | Abaixo do peso |
| 19,1 < IMC < 25,8 | No peso normal |
| 25,8 < IMC < 27,3 | Marginalmente acima do peso |
| 27,3 < IMC < 32,3 | Acima do peso ideal |
| 32,3 < IMC | Obesa |

Tabela: Situação para o sexo masculino.

| IMC | Situação |
|-------------------|-----------------------------|
| IMC < 20,7 | Abaixo do peso |
| 20,7 < IMC < 26,4 | No peso normal |
| 26,4 < IMC < 27,8 | Marginalmente acima do peso |
| 27,8 < IMC < 31,1 | Acima do peso ideal |
| 31,1 < IMC | Obeso |

Para calcularmos o IMC de uma pessoa, devemos obter o peso e a altura dela. O IMC é calculado pela razão do peso pela altura ao quadrado, conforme a fórmula apresentada anteriormente. Após se obter o valor do IMC, para sabermos a situação da pessoa, precisamos saber o sexo e a idade dela. Pela idade, determinaremos se a situação deve ou não ser calculada, uma vez que não devemos determinar a situação de uma pessoa com idade inferior a 15 anos. O sexo do indivíduo é importante, uma vez que existem duas diferentes tabelas para a análise da situação, uma para o sexo feminino e outra para o sexo masculino. Identificado o sexo, devemos aplicar a tabela correta para termos a situação da pessoa em relação ao peso ideal.

Podemos preparar um algoritmo baseado na estrutura do VisualG® para facilitar o entendimento e a conversão para uma aplicação Android®:

```

algoritmo "IMC"

// Declaração de variáveis:
var
    peso, altura, imc : real
    idade : inteiro
    sexo : inteiro // (1- Feminino, 2- Masculino)
    situacao : caractere
Inicio

    // Primeiro realizar todas as entradas de dados:
    escreval ("Peso:")
    leia (peso)
    escreval ("Altura:")
    leia (altura)
    escreval ("Idade menor de 15 anos (1-Sim, 2-Não)?")
    leia (idade)
    escreval ("Sexo (1- Feminino, 2- Masculino)")
    leia (sexo)

    // Segundo efetuar o processamento:

```

```

imc <- peso / (altura ^ 2)
se (idade = 1) entao // idade 1, Sim.
    situacao <- "Idade fora da faixa. Situação não determinada")
senao // idade 2, Não.
    se (sexo = 1) entao      // sexo = 1 - Feminino
        se(imc < 19.1) entao
            situacao <- "Abaixo do peso."
        senao
            se (imc < 25.8) entao
                situacao <- "No peso normal."
            senao
                se (imc < 27.3) entao
                    situacao <- "Marginalmente acima do peso."
                senao
                    se (imc < 32.3) entao
                        situacao <- "Acima do peso."
                    senao
                        situacao <- "Obesa."
                    fimse
                fimse
            fimse
        fimse
    fimse

senao //     sexo = 2 - Masculino

    se(imc < 20.7) entao
        situacao <- "Abaixo do peso."
    senao
        se (imc < 26.4) entao
            situacao <- "No peso normal."
        senao
            se (imc < 27.8) entao
                situacao <- "Marginalmente acima do peso."
            senao
                se (imc < 31.1) entao
                    situacao <- "Acima do peso."
                senao
                    situacao <- "Obeso."
                fimse
            fimse
        fimse
    fimse
fimse

fimse // fim das tabelas
fimse

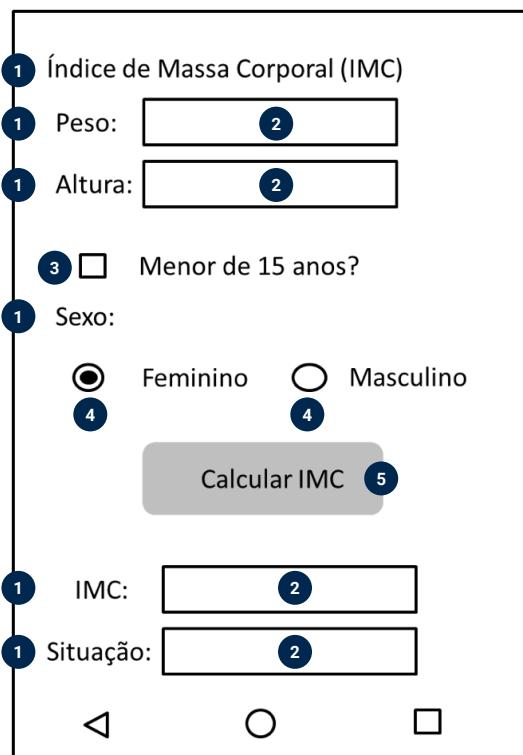
// Terceiro apresentar os resultados:
escreval ("IMC = ", imc:5:2) // saída formatada para duas casas
escreval ("Situação: ", situacao)

```

fimalgoritmo

A primeira coisa a se fazer é determinar um layout de como a tela de interface com o usuário deverá ficar. Segue um exemplo:

Componentes de tela utilizados na interface:



Tela de layout da aplicação (view).

1 TextView – Exibição de mensagens.

2 EditText – Entrada de dados.

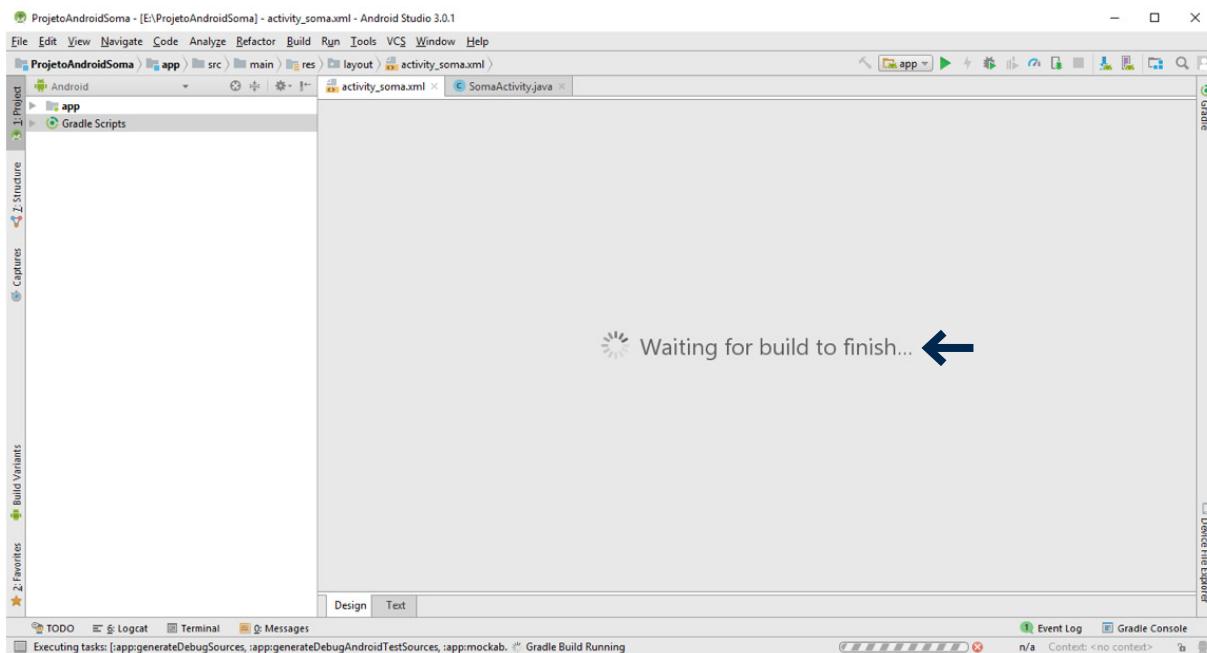
3 CheckBox – Caixa de seleção.

4 RadioButton – Escolha de opções.

5 Button – Botão.

1º passo: criar um projeto no Android® Studio chamado IMC, baseado em uma *empty activity*, com desenvolvimento na linguagem Java® e compatível com a API 15: Android 4.0.3 (Ice Cream Sandwich).

- O Android® Studio demora um pouquinho para carregar o projeto e preparar o ambiente de desenvolvimento.



2º passo: criar a interface com o usuário seguindo de acordo com o layout predeterminado:

Principais componentes de tela utilizados na interface:

| Componente | Descrição | Localização |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| TextView | Esse componente serve para exibir mensagens para o usuário. | Palette : Common |
| EditText | Esse componente serve para realizar a entrada de dados digitados pelo usuário. Neste exemplo, será utilizado também para a exibição dos resultados do aplicativo. | Palette : Text |
| CheckBox | Esse componente serve para a entrada de dados do tipo confirmação (sim/não) e está normalmente associado a uma pergunta direta ao usuário. | Palette : Button |

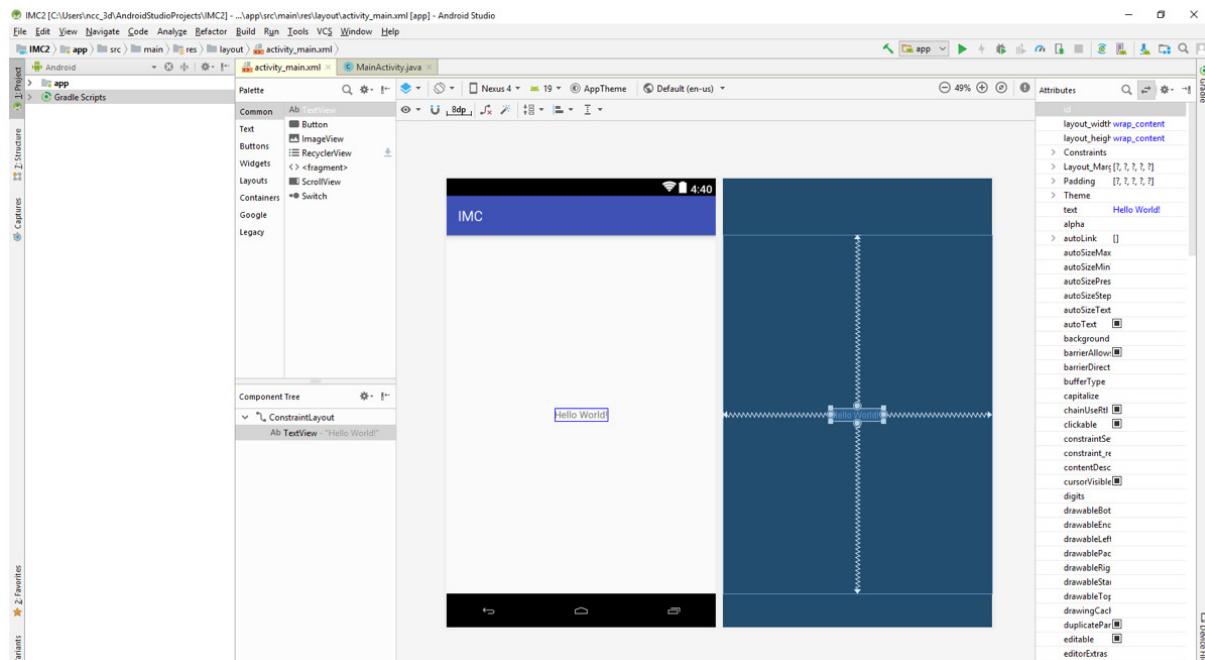
| | | |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| RadioButton | <p>Esse componente serve para permitir que o usuário faça uma escolha entre algumas opções apresentadas a ele e que ele possa escolher apenas uma delas.</p> | Palette : Button |
| Button | <p>Esse componente serve para que o usuário possa realizar a ação após efetuar a entrada de dados nos demais componentes.</p> | Palette : Button |



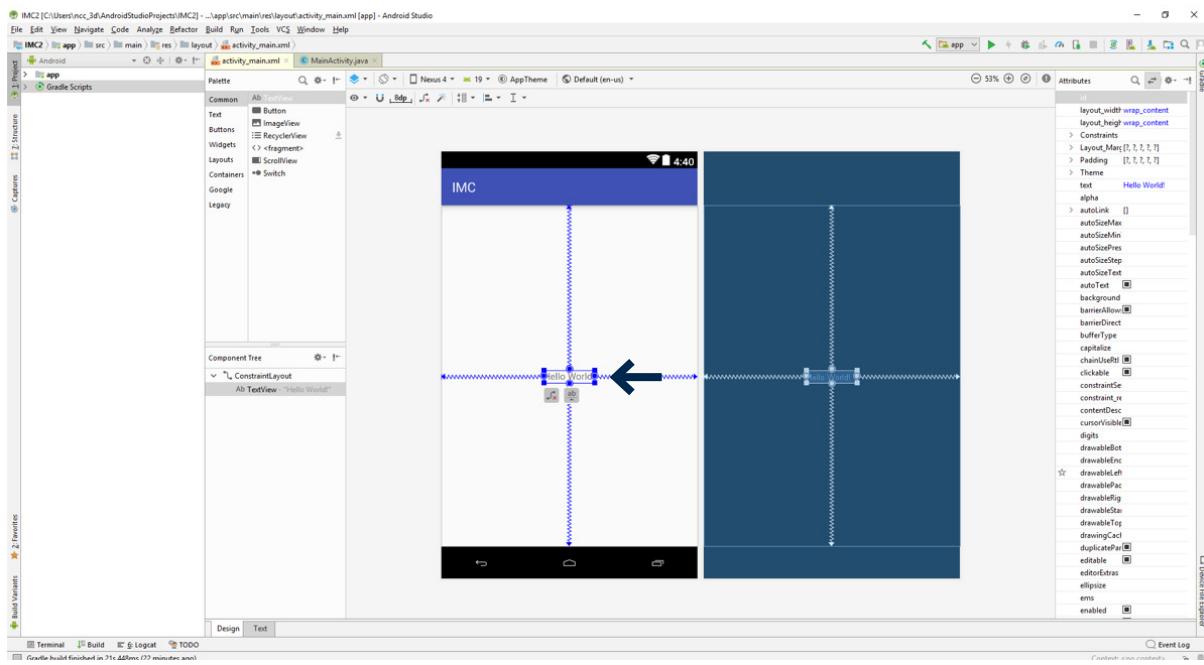
Observação

O componente **EditText** é único, mas possui diferentes modos de configuração de acordo com o tipo de teclado virtual. Para o nosso exemplo, todos os **EditText usados devem ser do tipo Number (Decimal) ou 42.0**, o que estiver disponível em sua configuração.

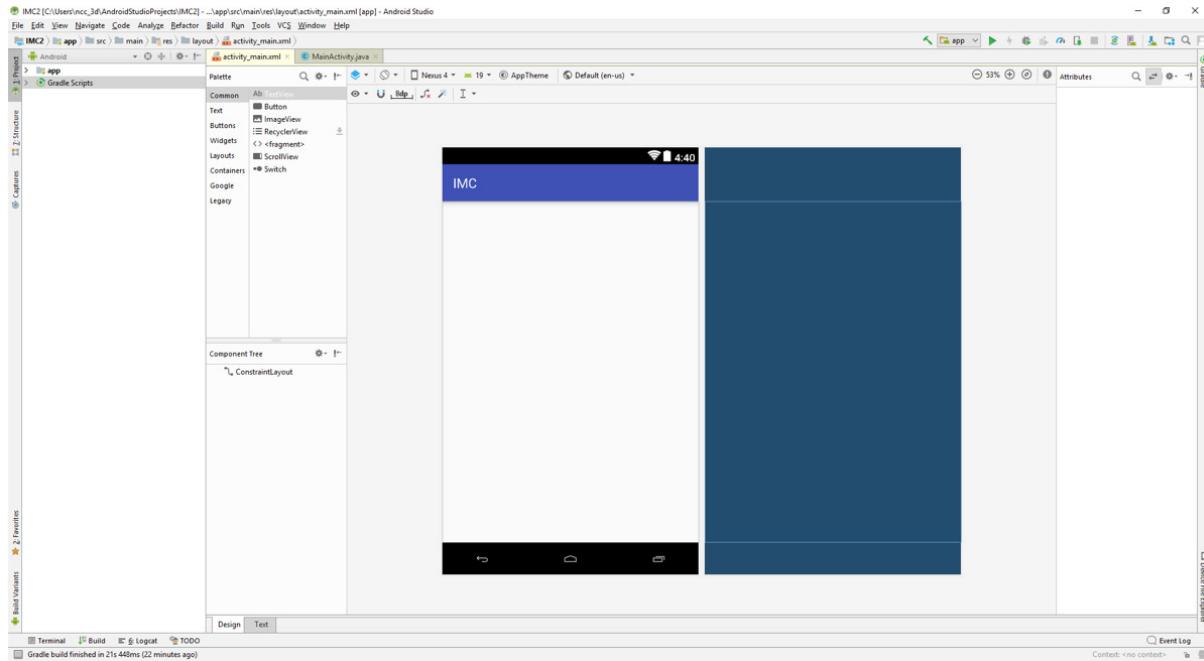
- Após a carga do projeto, o Android® Studio estará pronto para o desenvolvimento.



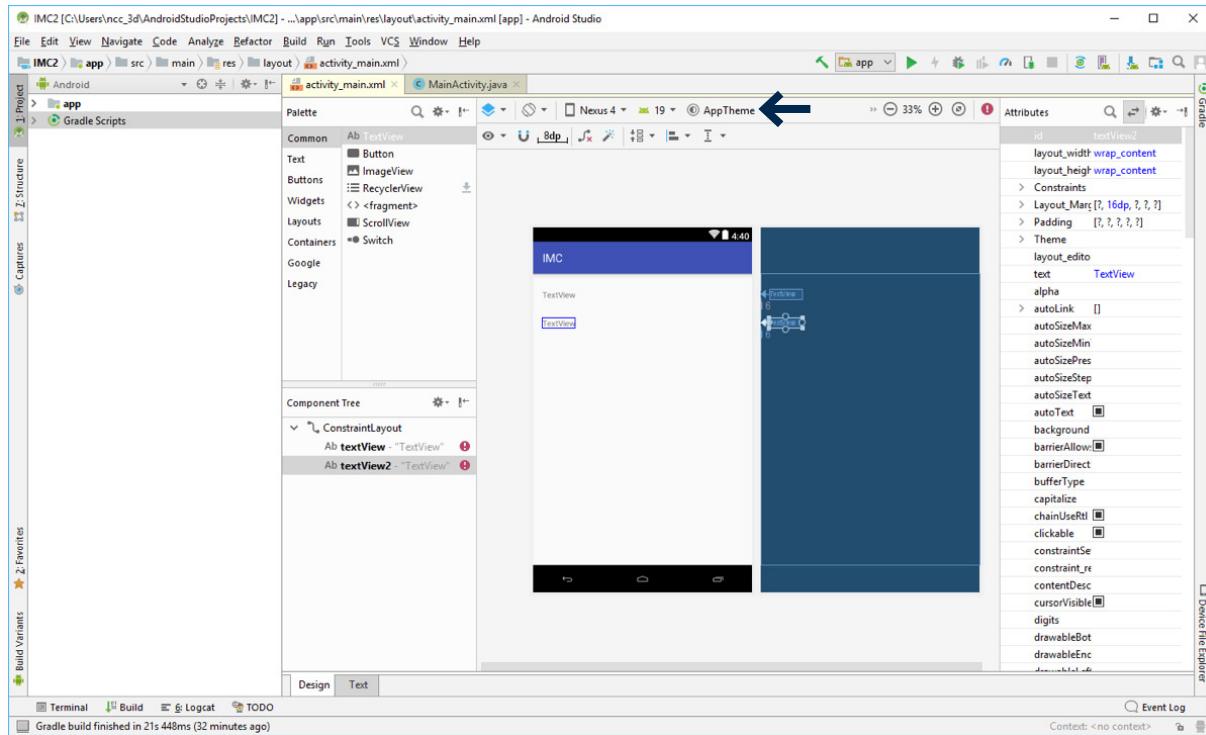
- O Android® Studio sempre cria uma tela de interface com a mensagem "Hello World".



- Podemos deletar esse componente para criarmos a tela de acordo com o nosso layout. Após selecionar o componente, basta usar a tecla "Delete", e o componente será eliminado da interface.

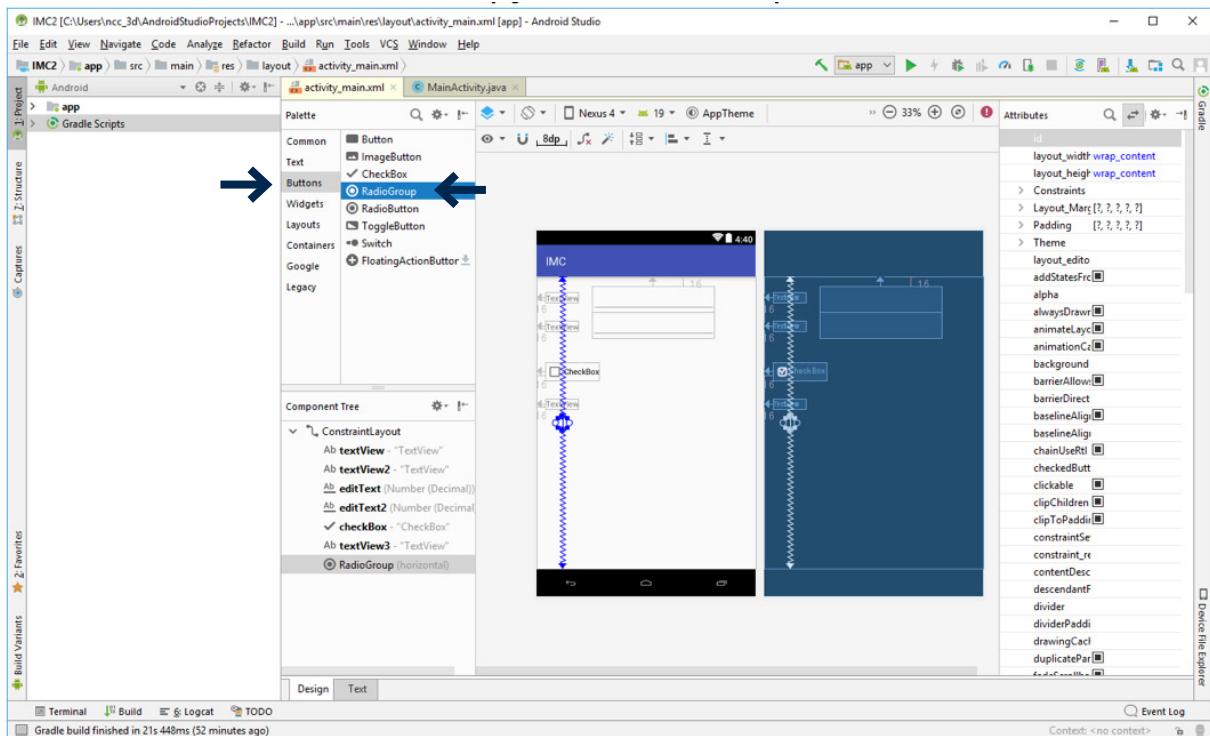


- Com a interface limpa, podemos começar a incluir os componentes necessários à nossa interface de acordo com o layout predeterminado.

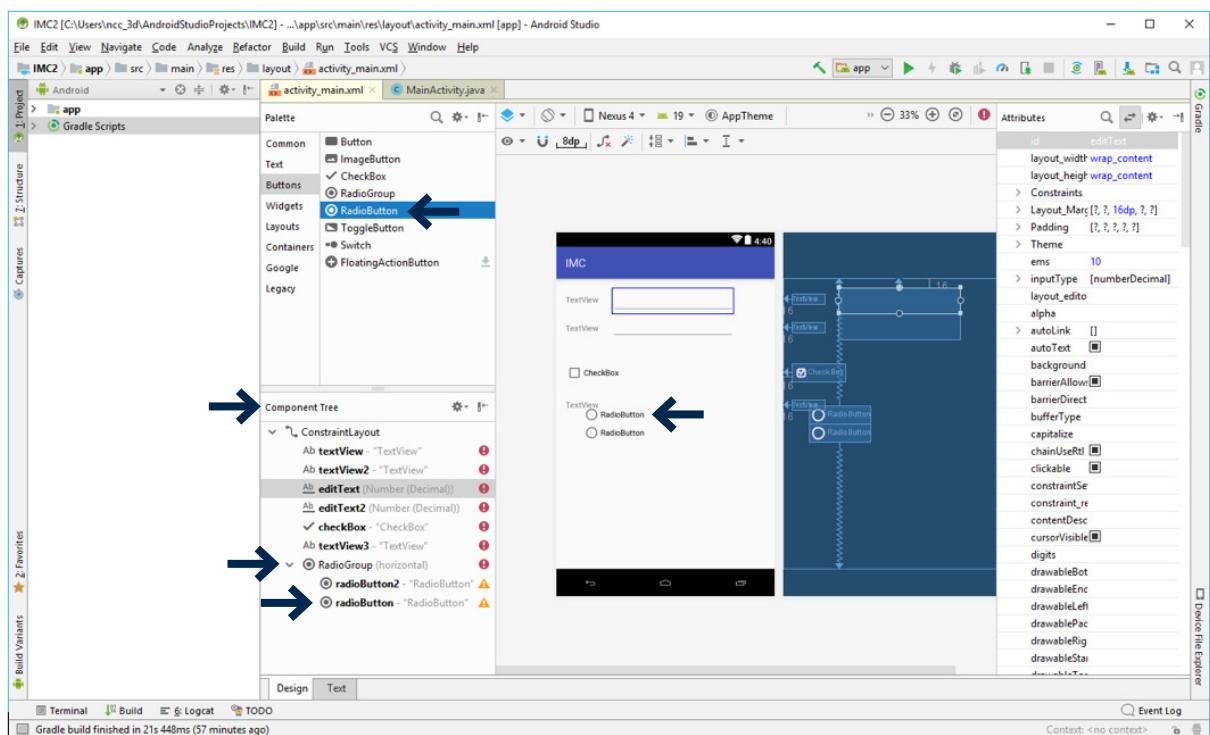


Caso você inclua algum componente na tela e ele não apareça, mude o tema da aplicação para Light/Light.

- Vá à área de “Palette” e selecione e inclua os componentes necessários à sua interface.
- Para incluir os botões de rádio (*RadioButton*), é necessário criar antes um *RadioGroup*, caso contrário as opções não estarão sincronizadas e o usuário poderá marcar mais de uma opção ao mesmo tempo.

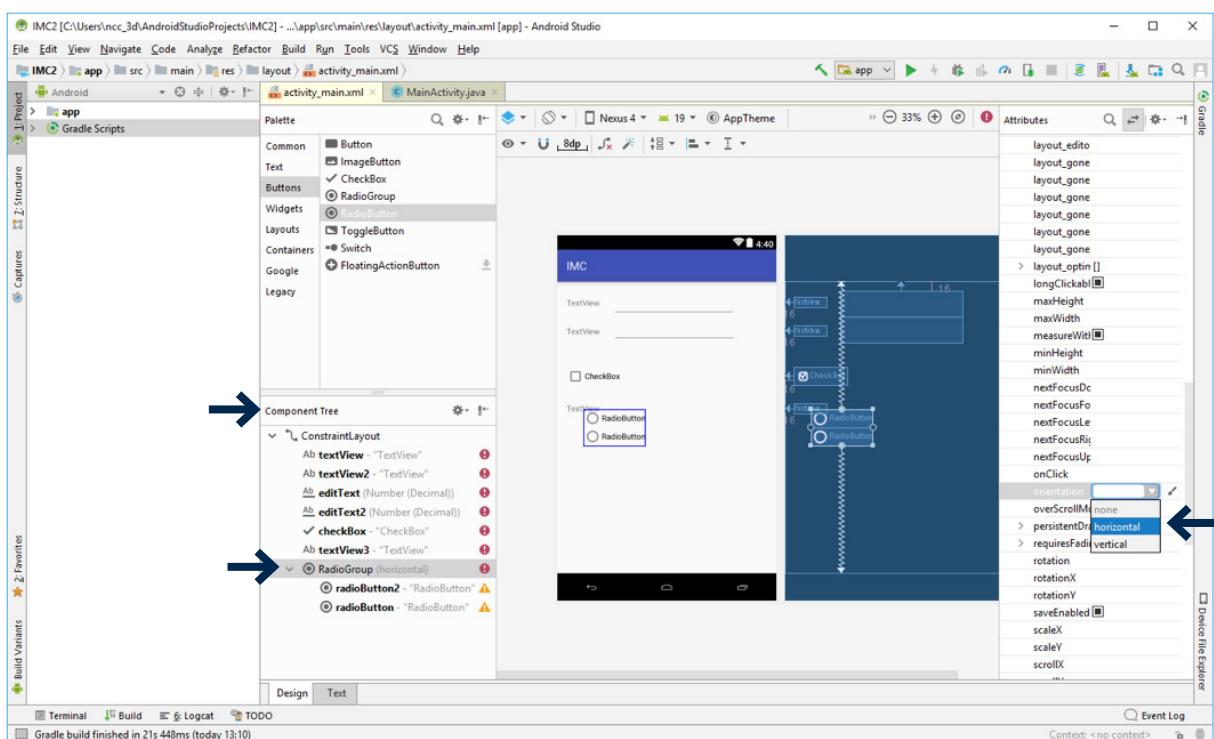


- Devemos, então, incluir os botões de rádio (*RadioButton*) dentro do *RadioGroup* por meio da tela. Esse processo se torna mais complicado, para isso devemos usar a área de “Component Tree” e incluir os dois botões de rádio sobre (dentro) o *RadioGroup*.

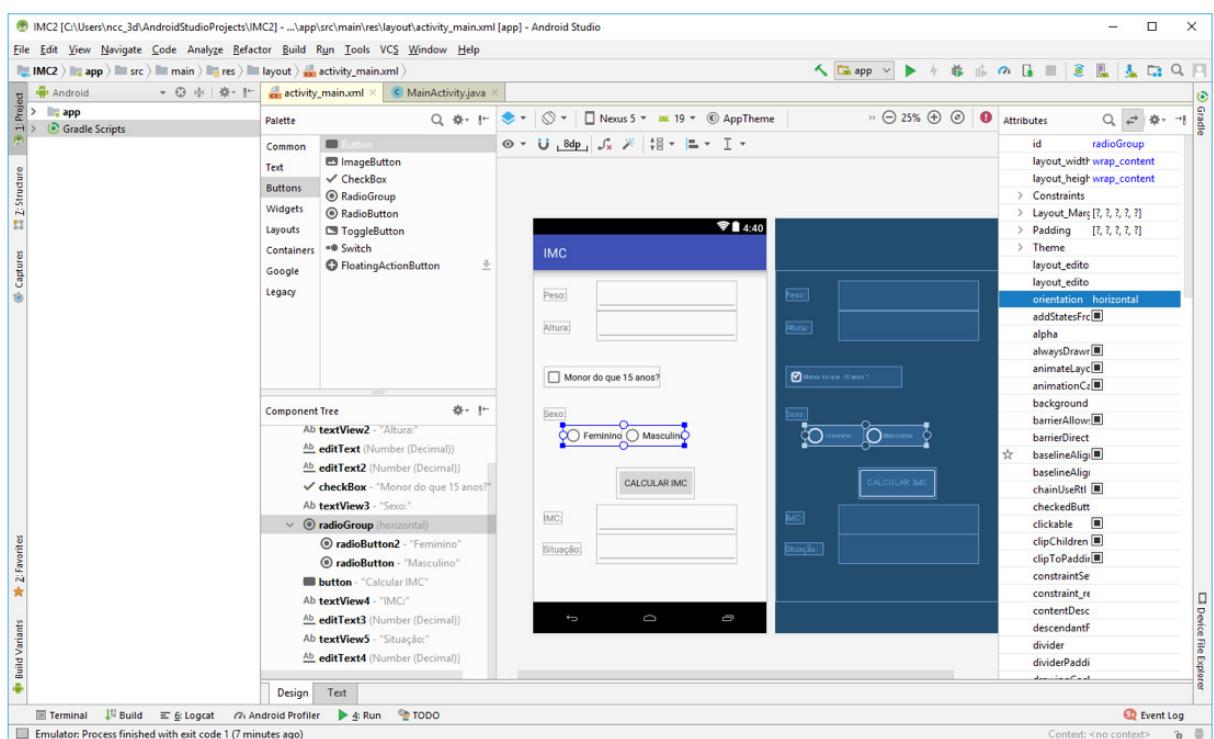


Os botões de rádio ficam mais à direita porque estão “dentro” do *RadioGroup*.

- Selecione o RadioGroup (será mais fácil selecioná-lo por meio do “Component Tree”).
- Para colocarmos os botões um ao lado do outro, conforme nosso layout, é necessário trocar a orientação do RadioGroup. Podemos selecionar o RadioGroup e alterar a propriedade “orientação” (orientation) para horizontal. Existem outras formas de realizarmos essa configuração, mas no momento manteremos essa.

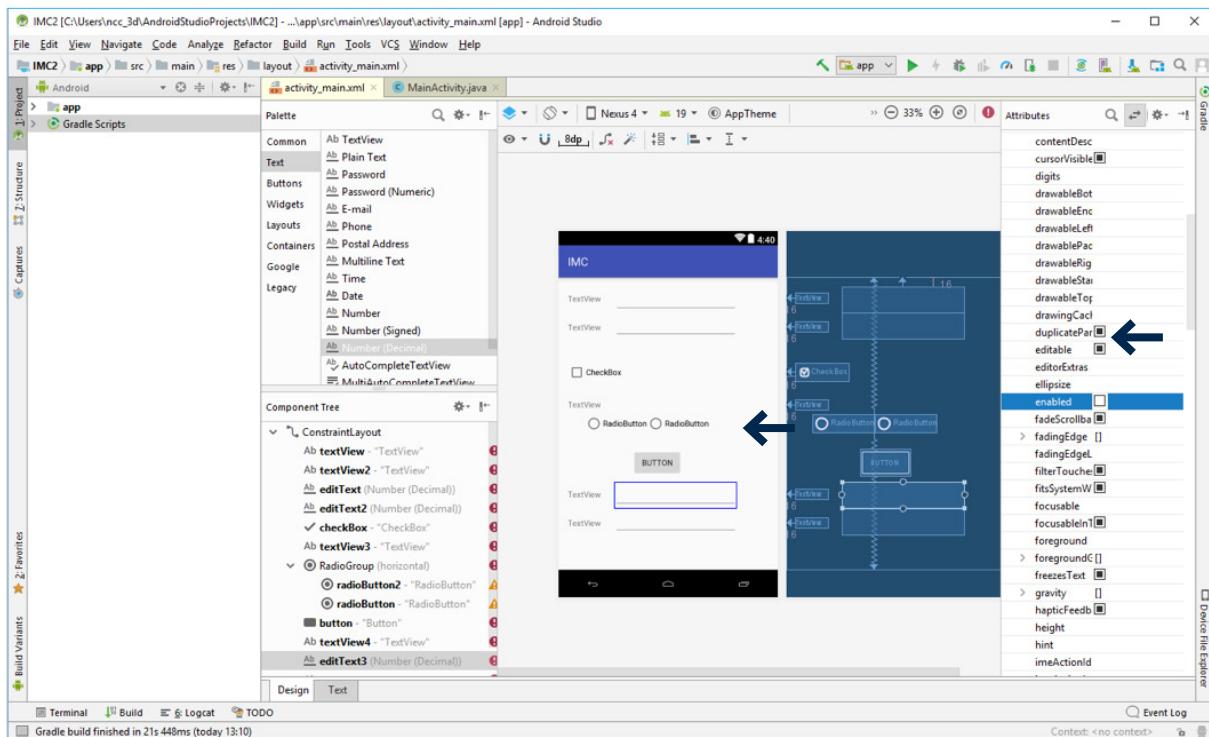


- Complete sua interface.

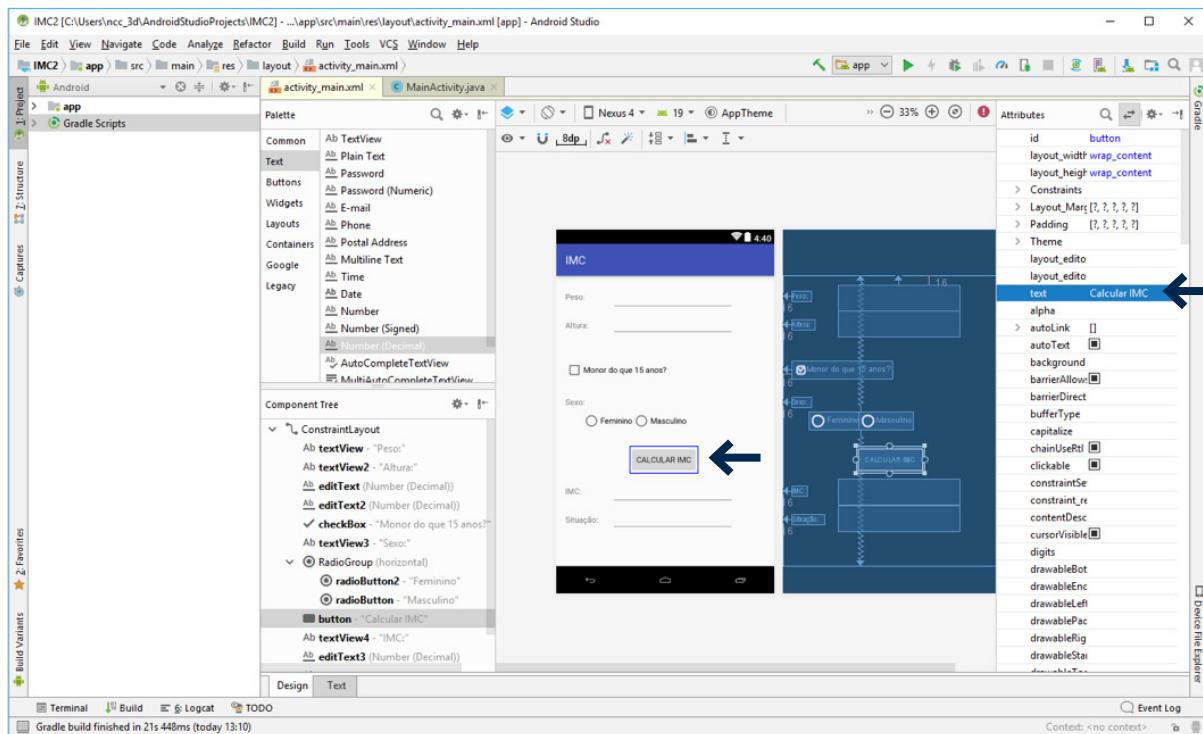


- Os *EditText* de saída podem ter a propriedade “enable” alterada para “false”, de forma que os usuários não possam editar seu valor.

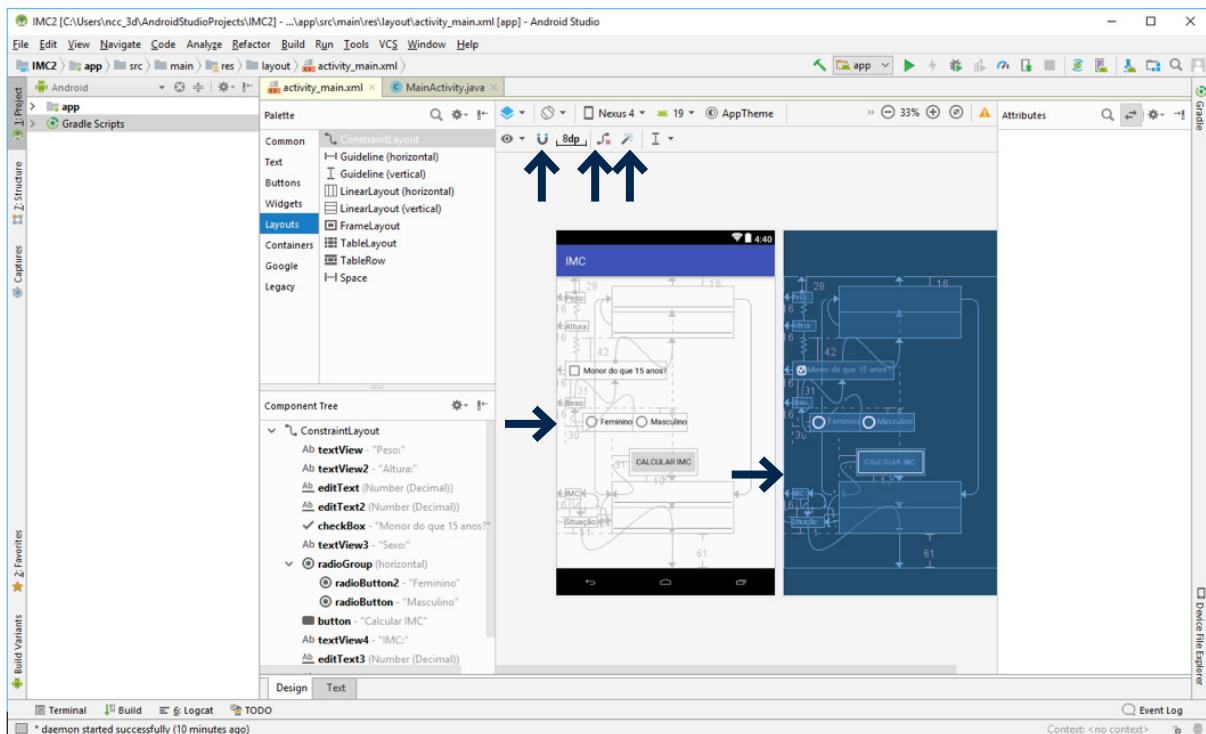
- Selecione cada um dos *EditText* de saída e troque a propriedade “enable” para “false” (desmarcada).



- A tela de interface com o usuário estará quase pronta, mas ainda precisaremos realizar mais uma ação. Seleccionaremos cada um dos componentes, com exceção dos *EditText*, e alteraremos a propriedade *Text* desses componentes. Os textos deverão ser os mesmos apresentados no nosso layout.

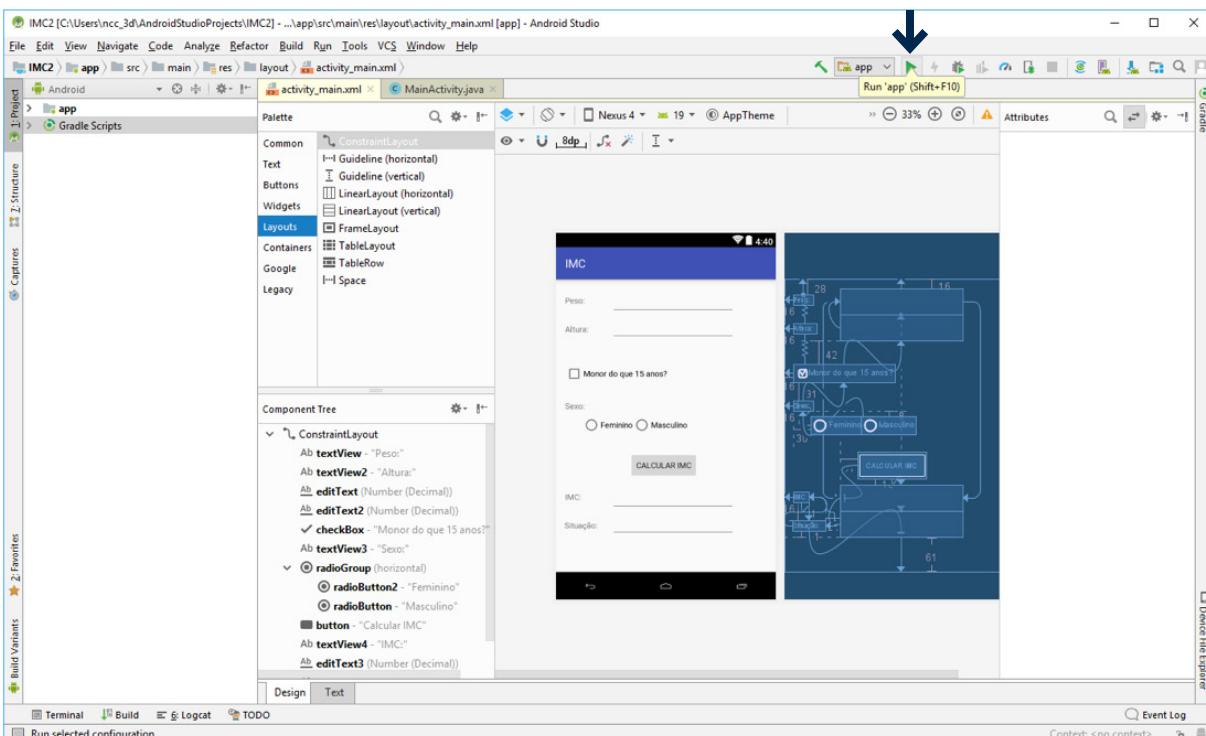


- Para terminar a nossa tela (view), devemos gerar as constraints para o nosso *ConstraintLayout*. Em breve, discutiremos os diferentes tipos de layout. Para que não tenhamos uma infeliz surpresa ao testar a tela, devemos gerar as constraints para os componentes, mas antes devemos permitir o *autoConnect* para facilitar a geração automática das *constraints* clicando sobre o ícone: para que ele fique liberado: .
- Clique sobre o ícone de limpeza das *constraints*: ; depois, clique sobre o ícone para gerar automaticamente as *constraints*: , caso contrário os componentes ficarão amontoados na tela do emulador.
- Já podemos realizar um teste para averiguar se a tela está de acordo com o nosso layout, ou se será necessário realizar alguma alteração, mas, para isso, é importante que você sempre limpe as *constraints* antes de alterar e as recrie antes de testar.

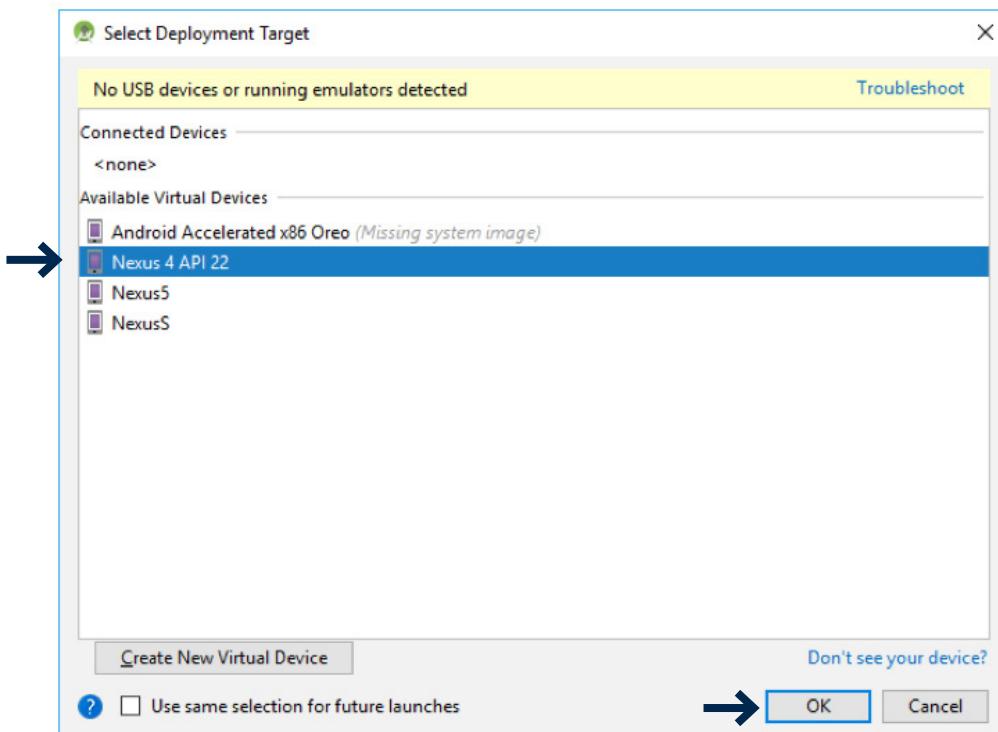


As constraints foram geradas.

- Vamos realizar um teste para verificar o resultado final da nossa interface. Clique sobre o botão de "Play" para iniciar o teste.



- Escolha um emulador já criado e clique em "OK" para realizar o teste da interface, mas lembre-se de que, até agora, apenas preparamos a interface e ainda não realizamos a programação, por isso o botão não funcionará.



- A tela (View) ficará da seguinte forma:

The image shows a comparison between a mobile application and a desktop application for calculating the Body Mass Index (IMC).

Mobile Application (Left):

- Header: 'Android Emulator - Nexus4:5554'
- Time: 9:44
- Form Fields:
 - Peso: [Text Input]
 - Altura: [Text Input]
 - Menor do que 15 anos?
 - Sexo:
 - Feminino
 - Masculino
 - CALCULAR IMC** button
 - IMC: [Text Input]
 - Situação: [Text Input]
- Navigation: Back, Home, Stop

Desktop Application (Right):

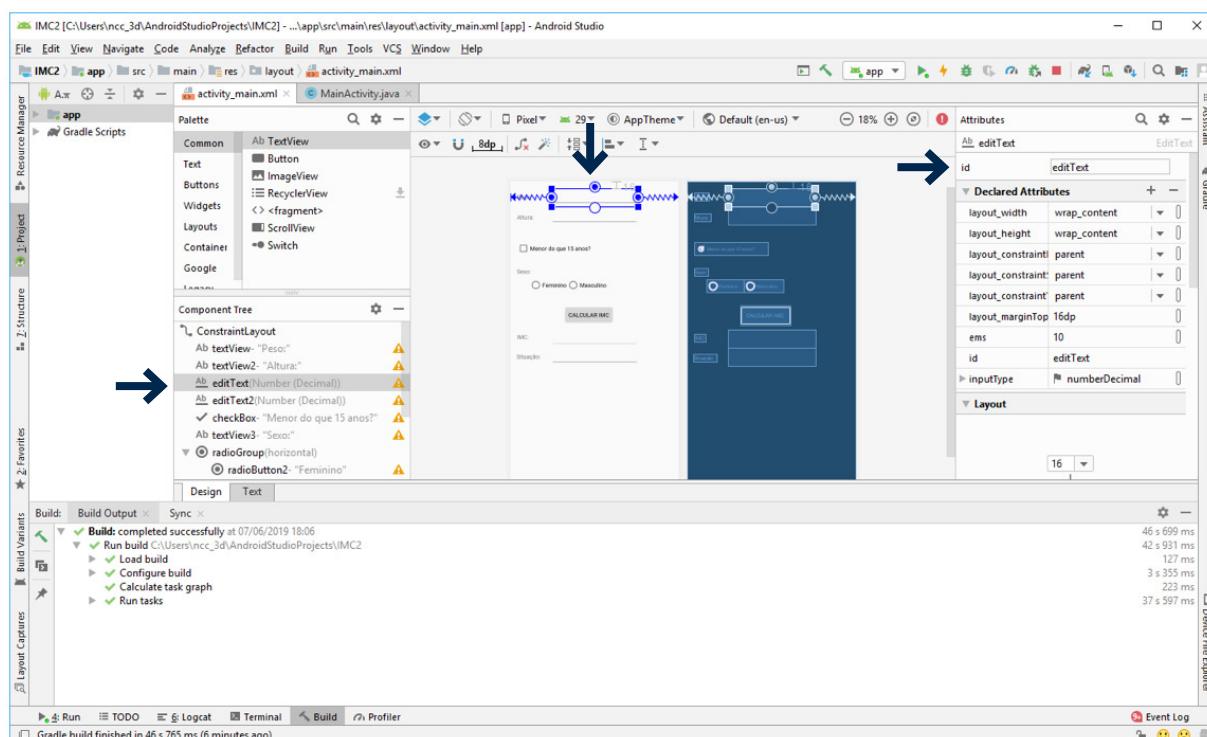
- Header: 'Índice de Massa Corporal (IMC)'
- Form Fields:
 - Peso: [Text Input]
 - Altura: [Text Input]
 - Menor de 15 anos?
 - Sexo:
 - Feminino
 - Masculino
 - Calcular IMC** button
 - IMC: [Text Input]
 - Situação: [Text Input]
- Navigation: Back, Home, Stop

A tela de interface criada (view) está de acordo com o layout idealizado.

Você pode melhorar a tela já definindo uma escolha inicial para o sexo. Selecione o *RadioButton* do sexo feminino e altere a propriedade “checked” para “true” (*RadioButton*).

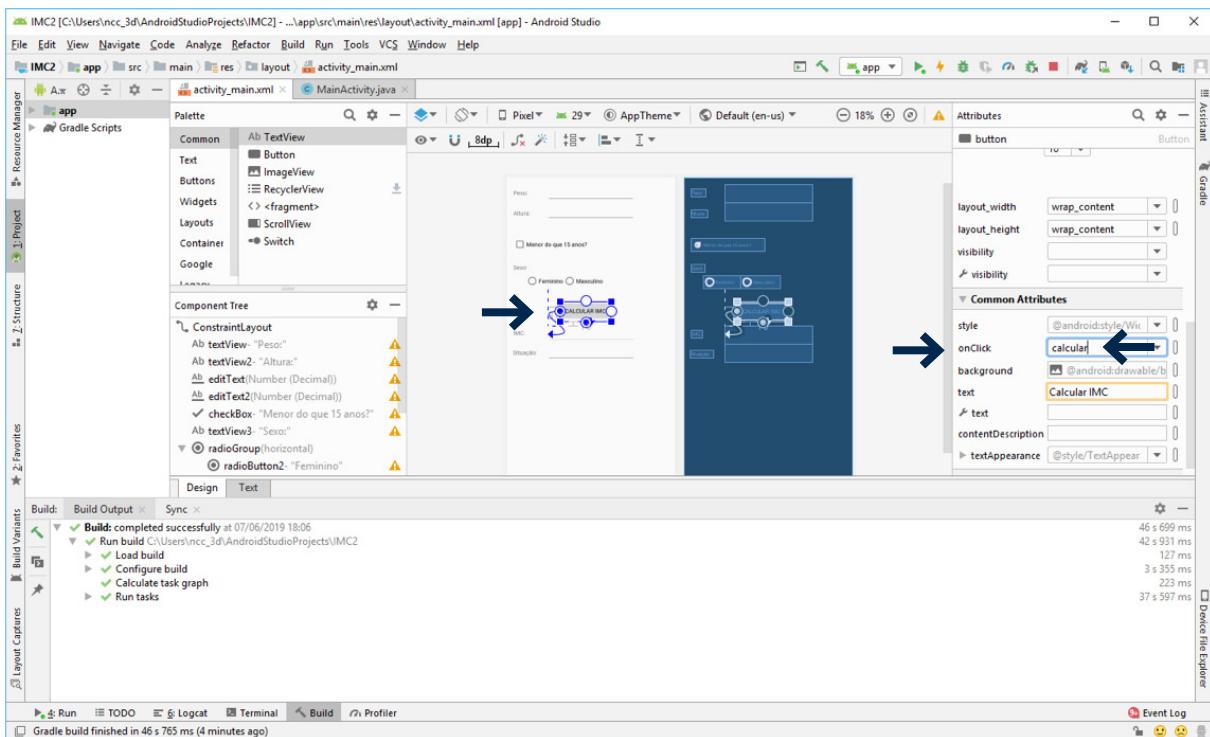
3º passo: programar a aplicação na camada de controle.

- Faça uma lista de todos os componentes da tela que serão usados para entrada e saída de dados. Não são necessários os componentes *TextView*, *Button* e *RadioGroup*, apenas os *EditText*, *CheckBox* e *RadioButton*.
- A lista deve conter o tipo e o id de cada componente. O *id* pode ser identificado ao selecionar o componente na propriedade (*id*). Cuidado com a escrita – as diferenças entre maiúsculas e minúsculas devem ser mantidas. Esses identificadores também podem ser obtidos na área de “Component Tree”. Os identificadores poderão ser alterados se você quiser.

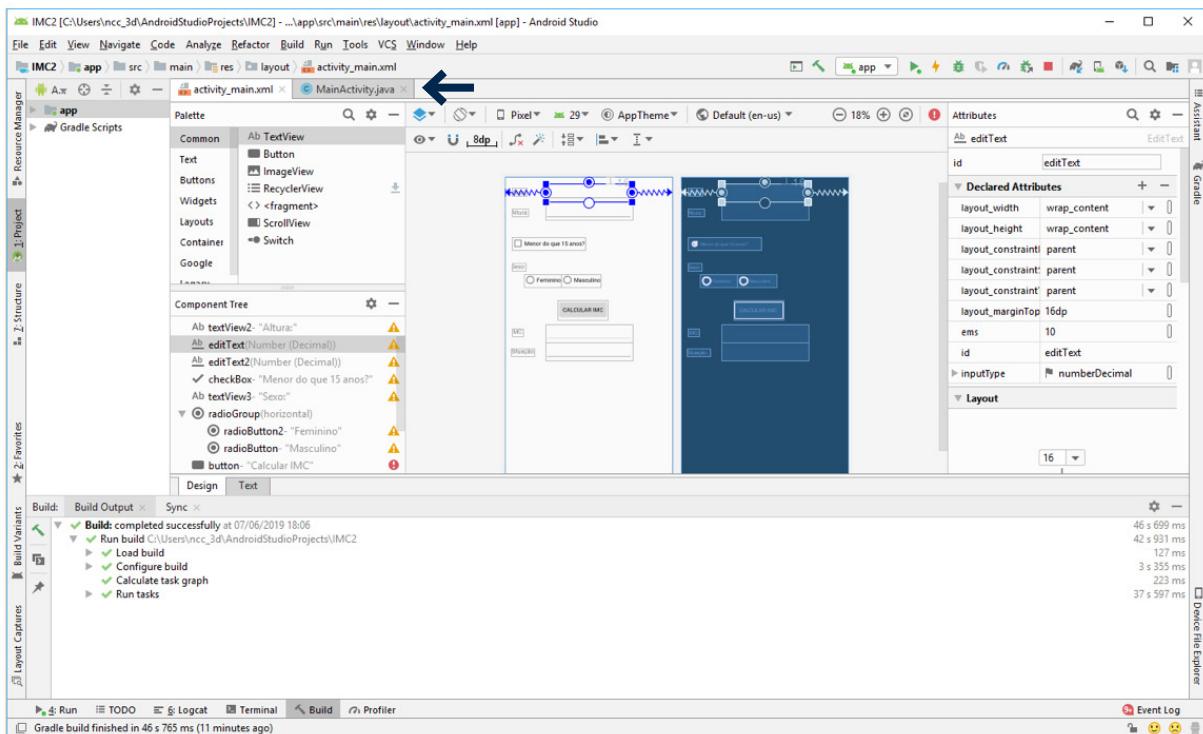


| Tipo do Componente | <i>id</i> (identificação do componente na view) | Entrada/Saída |
|--------------------|-------------------------------------------------|----------------|
| EditText | editText | Peso |
| EditText | editText2 | Altura |
| EditText | editText3 | IMC |
| EditText | editText4 | Situação |
| CheckBox | checkBox | Sim / Não |
| RadioButton | radioButton | Sexo Feminino |
| RadioButton | radioButton2 | Sexo Masculino |

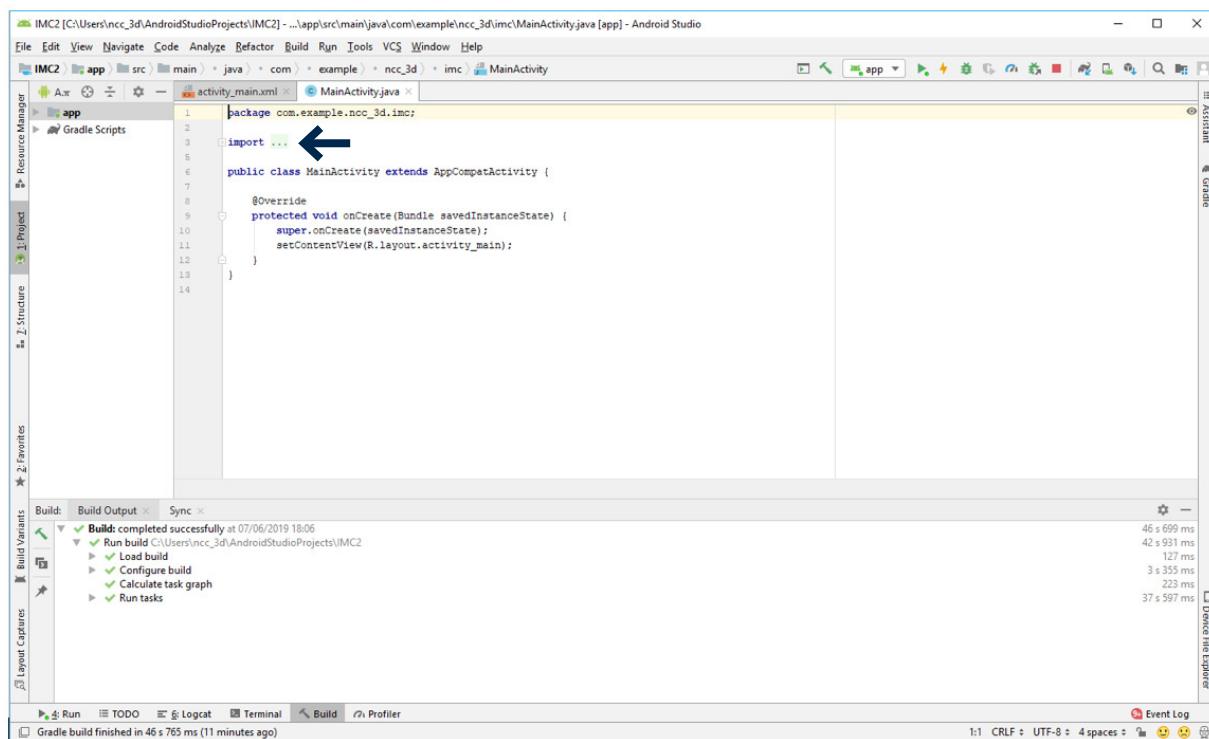
- Selecione o botão e mude a propriedade “onClick” para o nome que será usado para a execução do código – eu usarei: calcular (todas as letras minúsculas).



- Isso permitirá que o método chamado “calcular” no código seja executado quando o usuário clicar no botão.
- Agora, devemos mudar para a parte de codificação do projeto selecionando o arquivo da programação Java®, basta clicar sobre a aba com o nome da atividade com a extensão “.java”.



- A tela de programação mudará para:



Clique no pequeno botão de “+” ao lado da palavra “import”.

1. A primeira coisa a ser feita é a declaração dos componentes usados na tela que serão utilizados para entrada e saída de dados. Isso deve ser feito após a instrução de declaração da classe (public class). Não se esqueça do ponto e vírgula ao final das instruções, pois isso é obrigatório em Java®. Os identificadores (nomes) poderão ser iguais ou não aos dos componentes da tela.

```
package com.example.imc;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class IMCActivity extends Activity {

    // declaração dos componentes:
    EditText    edt1;
    EditText    edt2;
    EditText    edt3;
    EditText    edt4;
    CheckBox    cb1;
    RadioButton rad1;
    RadioButton rad2;

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

2. O Android® Studio inicialmente não os identificará, e será necessário incluir as bibliotecas desses componentes na região com os demais *import*:

```
package com.example.imc;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.CheckBox;
import android.widget.RadioButton;

public class IMCActivity extends Activity {
```

```

    // declaração dos componentes:
EditText    edt1;
EditText    edt2;
EditText    edt3;
EditText    edt4;
CheckBox    cb1;
RadioButton rad1;
RadioButton rad2;

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

```

3. Para trabalhar com os componentes da tela, devemos relacionar cada componente declarado no Java® com os componentes da tela. Isso ocorre após a última instrução dentro do método `onCreate`:

```

package com.example.imc;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.CheckBox;
import android.widget.RadioButton;

public class IMCActivity extends Activity {

    // declaração dos componentes:
EditText    edt1;
EditText    edt2;
EditText    edt3;
EditText    edt4;
CheckBox    cb1;
RadioButton rad1;
RadioButton rad2;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
}
}

```

```

        setContentView(R.layout.activity_imc);
        edt1 = (EditText) findViewById(R.id.editText);
        edt2 = (EditText) findViewById(R.id.editText2);
        edt3 = (EditText) findViewById(R.id.editText3);
        edt4 = (EditText) findViewById(R.id.editText4);
        cb1 = (CheckBox) findViewById(R.id.checkBox);
        rad1 = (RadioButton) findViewById(R.id.radioButton);
        rad2 = (RadioButton) findViewById(R.id.radioButton2);
    }

}

```

- A preparação para a inclusão do código está pronta.
 - A programação referente a seu algoritmo será realizada dentro do método com o mesmo nome que usamos na propriedade “onClick” do botão na tela. No meu caso, o nome foi: calcular.
4. Preparação do método “calcular” para a inclusão do algoritmo e a conversão para Java®. Sua inclusão deve ser realizada antes do último fechar chaves }. Também será necessário importar a biblioteca da view, junto dos demais *imports*, no início do código:

```

import android.view.View;

public void calcular (View v) {
}

```

- O código final de preparação do projeto ficará:

```

package com.example.imc;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.CheckBox;
import android.widget.RadioButton;

public class IMCActivity extends Activity {
    // declaração dos componentes:
    EditText edt1;

```

```

EditText    edt2;
EditText    edt3;
EditText    edt4;
CheckBox    cb1;
RadioButton rad1;
RadioButton rad2;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_imc);
    edt1 = (EditText) findViewById(R.id.editText);
    edt2 = (EditText) findViewById(R.id.editText2);
    edt3 = (EditText) findViewById(R.id.editText3);
    edt4 = (EditText) findViewById(R.id.editText4);
    cb1 = (CheckBox) findViewById(R.id.checkBox);
    rad1 = (RadioButton) findViewById(R.id.radioButton);
    rad2 = (RadioButton) findViewById(R.id.radioButton2);
}
public void calcular (View v) {
}
}

```

5. Agora, vamos trabalhar apenas dentro do método “calcular”:

```

public void calcular (View v) {
}

```

O Android® Studio deverá estar da seguinte forma com a inclusão dos códigos gerados até o momento:

```

4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.CheckBox;
7 import android.widget.EditText;
8 import android.widget.RadioButton;
9
10 public class MainActivty extends AppCompatActivity {
11     EditText edt1;
12     EditText edt2;
13     EditText edt3;
14     EditText edt4;
15     CheckBox cb1;
16     RadioButton rad1;
17     RadioButton rad2;
18
19     @Override
20     protected void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         setContentView(R.layout.activity_main);
23         edt1 = (EditText) findViewById(R.id.editText);
24         edt2 = (EditText) findViewById(R.id.editText2);
25         edt3 = (EditText) findViewById(R.id.editText3);
26         edt4 = (EditText) findViewById(R.id.editText4);
27         cb1 = (CheckBox) findViewById(R.id.checkBox);
28         rad1 = (RadioButton) findViewById(R.id.radioButton);
29         rad2 = (RadioButton) findViewById(R.id.radioButton2);
30     }
31     public void calcular (View v) {
32
33     }
34 }

```

MainActivity

Build: Build Output Sync x

- Build completed successfully at 07/06/2019 18:06
- Run build C:\Users\ncc_3d\AndroidStudioProjects\IMC2
- Load build
- Configure build
- Calculate task graph

46 s 699 ms
42 s 931 ms
127 ms
3 s 355 ms
223 ms

Event Log

Gradle build finished in 46 s 765 ms (22 minutes ago)

6. Declaração de variáveis no Java® de acordo com o algoritmo.

Algoritmo:

```

var
    peso, altura, imc : real
    idade : inteiro
    sexo : inteiro // (1- Feminino, 2- Masculino)
    situacao : caractere

```

Em Java®:

```

double peso, altura, imc; // o real pode ser declarado como double
int idade; // inteiro pode ser declarado como int
int sexo; // inteiro para: (1- Feminino, 2- Masculino)
String situacao; // texto pode ser declarado como String

```

Atualizando o método “calcular”:

```
public void calcular (View v) {  
    // declaração de variáveis:  
    double peso=0, altura=0, imc=0;  
    int idade=0;  
    int sexo=0;  
    String situacao;  
}
```

7. Entrada de dados em Java®.

```
// Entradas de dados.  
escreval ("Peso:")  
leia (peso)  
escreval ("Altura:")  
leia (altura)  
escreval ("Idade menor de 15 anos (1-Sim, 2-Não)?")  
leia (idade)  
escreval ("Sexo (1- Feminino, 2- Masculino)")  
leia (sexo)
```

Em Java®, não são necessários os escritos com as mensagens, pois a tela já contém as mensagens:

A) Para ler valores reais:

```
peso = Double.parseDouble(editText1.getText().toString());  
altura = Doubl [REDACTED] le(editText2.getText().toString());  
[REDACTED]
```

B) Para ler valores inteiros:

```
idade = Integer.parseInt(editText1.getText().toString());  
[REDACTED]
```

Você deve identificar corretamente o componente para não pegar o valor errado.

C) Para saber se um *CheckBox* está marcado:

```
if(cb1.isChecked()) { // se estiver marcado idade valerá 1, caso  
contrário valerá 2
```

```

        idade = 1;
    }
else {
    idade = 2;
}

```

D) Para saber qual *RadioButton* está marcado:

```

if(rad1.isChecked()) { // se o rad1 estiver marcado sexo valerá 1.
    sexo = 1;
} else { // senão valerá 2.
    sexo = 2;
}

```

Atualizando o método “calcular”:

```

public void calcular (View v) {
    // declaração de variáveis:
    double peso=0, altura=0, imc=0;
    int idade=0;
    int sexo=0;
    String situacao;
    // Entradas de dados.
    peso = Double.parseDouble(editText1.getText().toString()); // Lê o
real armazenado em editText
    altura = Double.parseDouble(editText2.getText().toString());// Lê o
real armazenado em editText2

    if(cb1.isChecked()) { // se estiver marcado idade valerá 1, caso
contrário valerá 2
        idade = 1;
    }
    else {
        idade = 2;
    }
    if(rad1.isChecked()) { // se o rad1 estiver marcado sexo valerá 1.
        sexo = 1;
    } else { // senão valerá 2.
        sexo = 2;
    }
}

```

8. Processamento no algoritmo.

```
// Processamento
imc <- peso / (altura ^ 2)
se (idade = 1) entao // idade 1, Sim.
    situacao <- "Idade fora da faixa. Situação não determinada"
senao // idade 2, Não.
    se (sexo = 1) entao      // sexo = 1 - Feminino
        se(imc < 19.1) entao
            situacao <- "Abaixo do peso."
        senao
            se (imc < 25.8) entao
                situacao <- "No peso normal."
            senao
                se (imc < 27.3) entao
                    situacao <- "Marginalmente acima do peso."
                senao
                    se (imc < 32.3) entao
                        situacao <- "Acima do peso."
                    senao
                        situacao <- "Obesa."
                fimse
            fimse
        fimse
    fimse

senao //     sexo = 2 - Masculino

    se(imc < 20.7) entao
        situacao <- "Abaixo do peso."
    senao
        se (imc < 26.4) entao
            situacao <- "No peso normal."
        senao
            se (imc < 27.8) entao
                situacao <- "Marginalmente acima do peso."
            senao
                se (imc < 31.1) entao
                    situacao <- "Acima do peso."
                senao
                    situacao <- "Obeso."
            fimse
        fimse
    fimse
fimse // fim das tabelas
fimse
```

Convertendo todo o processamento:

```
// Processamento
imc = peso / Math.pow(altura, 2); // Math.pow realiza o cálculo de
potência
if (idade == 1) { // idade 1, Sim.
    situacao = "Idade fora da faixa. Situação não determinada";
}
else { // idade 2, Não.
    if (sexo == 1) { // sexo = 1 - Feminino
        if(imc < 19.1) {
            situacao = "Abaixo do peso.";
        }

        else {
            if (imc < 25.8) {
                situacao = "No peso normal.";
            }
            else {
                if (imc < 27.3) {
                    situacao = "Marginalmente acima do peso.";
                }
                else {
                    if (imc < 32.3) {
                        situacao = "Acima do peso.";
                    }
                    else {
                        situacao = "Obesa.";
                    }
                }
            }
        }
    }
}
else { // sexo = 2 - Masculino
    if(imc < 20.7) {
        situacao = "Abaixo do peso.";
    }
    else {
        if (imc < 26.4) {
            situacao = "No peso normal.";
        }
        else {
            if (imc < 27.8) {
                situacao = "Marginalmente acima do peso.";
            }
            else {
```

```

        if (imc < 31.1) {
            situacao = "Acima do peso.";
        }
        else {
            situacao = "Obeso.";
        }
    }
}
} // fim das tabelas
}

```

9. Saída de dados no algoritmo.

```

// Saída de dados.
escreval ("IMC = ", imc:5:2) // saída formatada para duas casas
escreval ("Situação: ", situacao)

```

Em Java®, precisamos apenas preencher os componentes de saída (*EditText*) com os valores obtidos após o processamento:

A) Para apresentar valores (reais ou inteiros):

```
editText3.setText(String.format("%.2f",imc));// o valor real será exibido
com duas casas decimais
```

B) Para apresentar variáveis do tipo texto (*String*):

```
editText4.setText(situacao);
```

Código final do projeto IMC:

```

package com.example.imc;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.RadioButton;

public class MainActivity extends AppCompatActivity {
    EditText edt1;

```

```

EditText    edt2;
EditText    edt3;
EditText    edt4;
CheckBox   cb1;
RadioButton rad1;
RadioButton   rad2;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    edt1 = (EditText) findViewById(R.id.editText);
    edt2 = (EditText) findViewById(R.id.editText2);
    edt3 = (EditText) findViewById(R.id.editText3);
    edt4 = (EditText) findViewById(R.id.editText4);
    cb1 = (CheckBox) findViewById(R.id.checkBox);
    rad1 = (RadioButton) findViewById(R.id.radioButton);
    rad2 = (RadioButton) findViewById(R.id.radioButton2);
}
public void calcular (View v) {
    // declaração de variáveis:
    double peso=0, altura=0, imc=0;
    int idade=0;
    int sexo=0;
    String situacao;

    // Entradas de dados.
    peso = Double.parseDouble(edt1.getText().toString());
    altura = Double.parseDouble(edt2.getText().toString());
    if(cb1.isChecked()) { // se estiver marcado idade valerá 1, caso
contrário valerá 2
        idade = 1;
    }
    else {
        idade = 2;
    }
    if(rad1.isChecked()) { // se o rad1 estiver marcado sexo valerá 1.
        sexo = 1;
    } else { // senão valerá 2.
        sexo = 2;
    }
    // Processamento
    imc = peso / Math.pow(altura, 2); // Math.pow realiza o cálculo
de potência
    if (idade == 1) { // idade 1, Sim.
        situacao = "Idade fora da faixa. Situação não determinada";
    }
}

```

```

else { // idade 2, Não.
    if (sexo == 1) { // sexo = 1 - Feminino
        if(imc < 19.1) {
            situacao = "Abaixo do peso.";
        }
        else {
            if (imc < 25.8) {
                situacao = "No peso normal.";
            }
            else {
                if (imc < 27.3) {
                    situacao = "Marginalmente acima do peso.";
                }
                else {
                    if (imc < 32.3) {
                        situacao = "Acima do peso.";
                    }
                    else {
                        situacao = "Obesa.";
                    }
                }
            }
        }
    }
}

else { // sexo = 2 - Masculino
    if(imc < 20.7) {
        situacao = "Abaixo do peso.";
    }
    else {
        if (imc < 26.4) {
            situacao = "No peso normal.";
        }

        else {
            if (imc < 27.8) {
                situacao = "Marginalmente acima do peso.";
            }
            else {
                if (imc < 31.1) {
                    situacao = "Acima do peso.";
                }
                else {
                    situacao = "Obeso.";
                }
            }
        }
    }
}
}

```

```

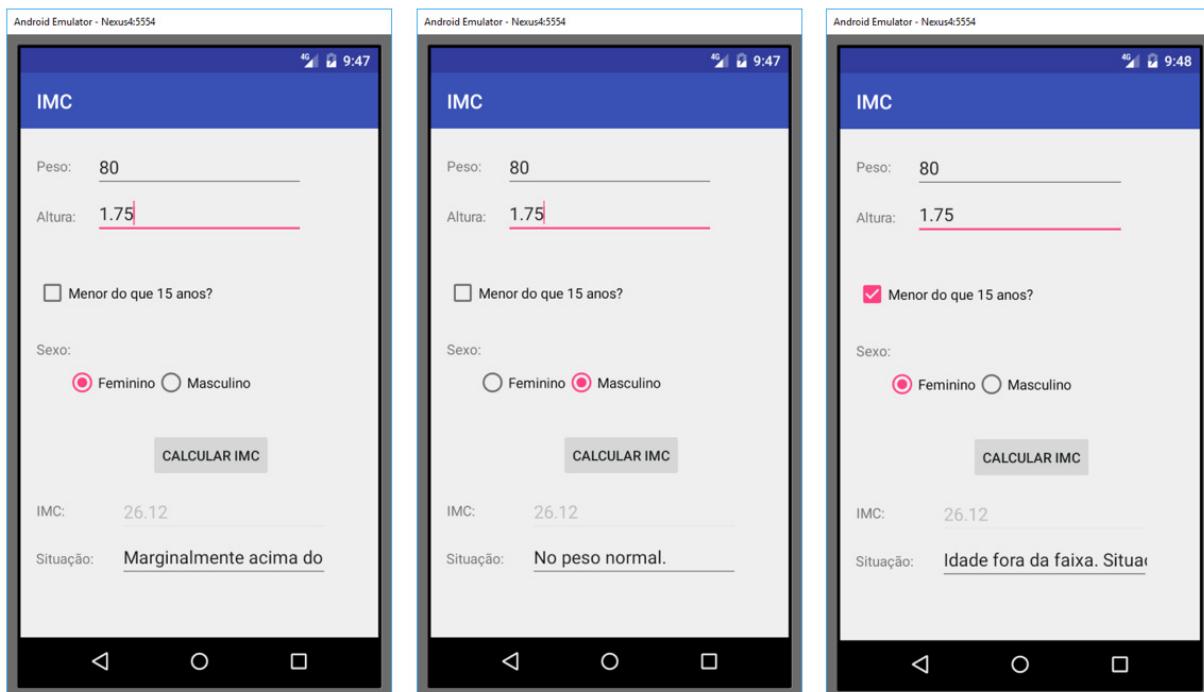
        } // fim das tabelas
    }

    // Saída de dados
    edt3.setText(String.format("%.2f",imc));
    edt4.setText(situacao);
}

}

```

Testes no emulador do aplicativo:



Observação

Se você estiver com seu smartphone conectado ao computador e este estiver liberado para o modo programador, você poderá testar suas aplicações instalando-as diretamente em seu aparelho.



MEDIATECA

Acesse a midiateca da Unidade 2 e veja o conteúdo complementar indicado pelo professor sobre o desenvolvimento de aplicações simples no Android®.



Saiba mais

Desenvolvimento de aplicações no Android® utilizando o Android® Studio

DEITEL, P. J.; DEITEL, H. M.; DEITEL, A. **Android**: como programar. Porto Alegre: Bookman, 2015. Minha Biblioteca. cap. 2, seções: 2.4 a 2.9, p. 46-66.



NA PRÁTICA

Utilize seu smartphone para analisar alguns aplicativos quanto à mudança de foco (primeiro e segundo planos) no sistema operacional e como eles se comportam entre essas mudanças.

Resumo da Unidade 2

Nesta unidade, você conheceu o modelo MVC – o desenvolvimento nas camadas de interface (*view*), controle (*control*) e modelo (*model*) – e como implementá-lo no desenvolvimento de suas aplicações. Outro ponto muito importante foi compreender o ciclo de vida das aplicações, que é definido por um conjunto de sete métodos que são executados em conjunto com o sistema operacional, permitindo que sua aplicação possa ser controlada mesmo que ela não esteja em primeiro plano no sistema operacional. Esse conceito permite que você realize operações mesmo que a aplicação não esteja em foco (primeiro plano). Por fim, você teve uma maior interação com o Android® Studio e pôde criar sua primeira aplicação. Isso permitiu que você conhecesse melhor os passos para o desenvolvimento de aplicações utilizando a ferramenta e que pudesse testar sua aplicação.



CONCEITO

Exploramos o modelo MVC, suas características e o desenvolvimento em camadas e entendemos como aplicar os métodos de controle do ciclo de vida da atividade para manter o controle sobre sua aplicação mesmo quando esta deixa de estar em primeiro plano (foco) pelo sistema operacional e, ainda, como desenvolver um projeto prático de uma aplicação para dispositivos móveis.

Referências

BOND, M. et al. **Aprenda J2EE em 21 dias**. São Paulo: Pearson Education do Brasil, 2003. Biblioteca Virtual Pearson.

DEITEL, P. J.; DEITEL, H. M.; DEITEL, A. **Android**: como programar. Porto Alegre: Bookman, 2015. Minha Biblioteca.

_____ ; _____ ; WALD, A. **Android 6 para programadores**: uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2016. Minha Biblioteca.

UNIDADE 3

Trabalhando com componentes
do sistema e múltiplas interfaces

INTRODUÇÃO

Na criação de nossos aplicativos, podemos aproveitar outros aplicativos já existentes no Android do aparelho, aproveitando o aplicativo de acordo com o seu tipo, uma vez que nem sempre precisamos identificar a aplicação diretamente. O uso de filtros permite identificar um aplicativo por meio do uso, e não de seu nome. É comum chamar para execução uma aplicação a partir de um recurso, sem que necessitemos identificá-lo explicitamente. A troca de mensagens entre nossos aplicativos e o sistema operacional permite que as aplicações reproveitem recursos existentes em outros aplicativos, reduzindo nossa necessidade de desenvolvimento. Por exemplo, se precisamos iniciar uma chamada telefônica dentro de nossa aplicação, podemos realizar uma chamada ao sistema, que executará a chamada por meio do aplicativo do telefone. Outra troca de mensagens importante é entre as atividades de nossa aplicação. Diferentes atividades podem repassar dados e permitir que distintas janelas tenham acesso a dados oriundos de outras janelas.

Todos os testes dos projetos desta unidade serão realizados com um aparelho Nexus 4 com a API 22 (Android 5.1 - Lollipop), por meio de um emulador no Android Studio.



OBJETIVO

Nesta unidade, você será capaz de:

- Compreender como utilizar componentes do sistema nos aplicativos e como trabalhar com múltiplas interfaces gráficas com o usuário.

A *intent* e o acesso a componentes do sistema

A *intent* é a forma em que aplicações Android fazem acesso a outros recursos existentes no sistema, incluindo outras aplicações. Muitas vezes não notamos, mas nossa aplicação acaba por chamar outras aplicações ou funções implicitamente. Já em outros casos, podemos realizar de forma explícita essa chamada.

Quando definimos que será usado um recurso, mas não identificamos a aplicação e não definimos o que será usado, essa *intent* é dita implícita. Uma *intent* implícita determina a chamada a uma página da internet ou a um número para uma chamada telefônica. O Android então identifica um aplicativo capaz de realizar a operação e o chama para execução com os dados passados. Se for uma página, a partir de um endereço, será chamado para execução um aplicativo de browser (navegador); se for um número de telefone, será chamado para execução o aplicativo do telefone. Em ambos os casos, os parâmetros passados serão usados pelo aplicativo chamado para atender à necessidade do usuário. Nas *intents* explícitas, devemos chamar para execução um aplicativo específico e, caso este não exista, será gerada uma exceção.

A *intent* é um objeto para troca de mensagens em aplicações móveis. Por meio da *intent*, é possível solicitar a execução de um outro componente do aplicativo ou de outra aplicação.

Formas de uso

Normalmente, a *intent* é utilizada de três diferentes maneiras em aplicações móveis:

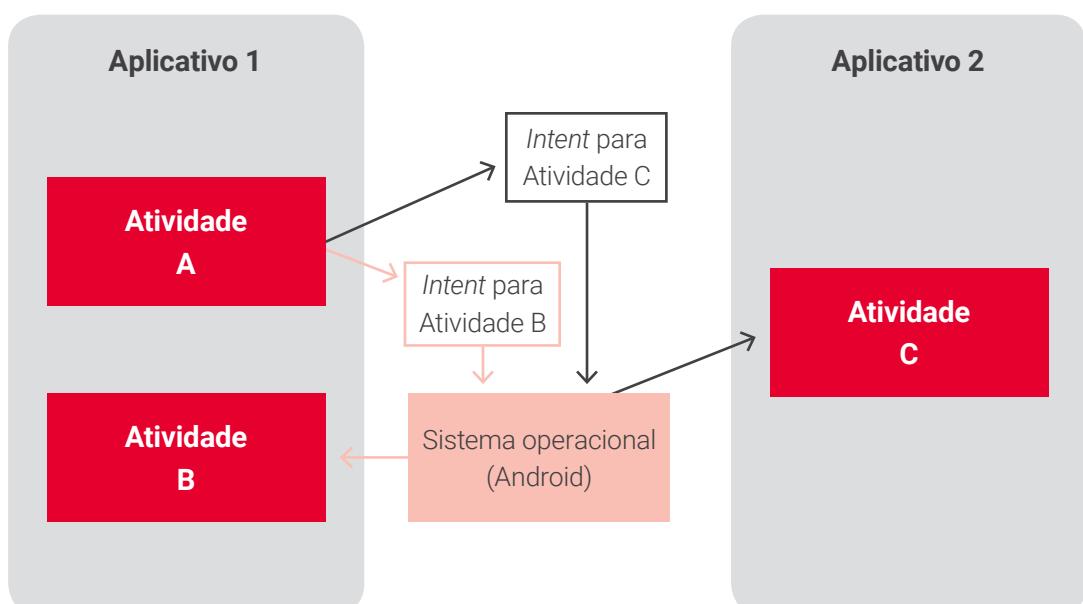
1. Iniciar uma atividade – Ao trabalhar com diversas telas (*views*) em uma mesma aplicação, a *intent* pode ser responsável por instanciar uma nova *activity*, fazendo com esta passe à execução e, consequentemente, para o primeiro plano da aplicação, com sua *view* em foco.
2. Iniciar um serviço – Ao trabalhar com serviços do Android, a *intent* pode iniciar a execução de um serviço em segundo plano, normalmente sem que uma *view* seja apresentada.
3. Iniciar uma transmissão – Ao trabalhar com troca de mensagens, a *intent* pode iniciar um outro aplicativo disponível no Android. Essas transmissões permitem que

outro aplicativo seja chamado para execução, mas com informações passadas pela aplicação que chamou. Dessa forma, um aplicativo existente no sistema pode ser chamado para execução com determinados parâmetros predefinidos, fazendo com que esse aplicativo entre em execução com informações prévias enviadas pela aplicação chamadora.

Tipos de *intents*

- Explícitas – Uma *intent* é dita explícita quando especifica o componente pelo nome. É o caso de uma chamada por *intent* a uma *activity* do próprio aplicativo. Uma *intent* explícita também é comumente utilizada para a chamada para execução de serviços.
- Implícitas – O componente não é especificado por nome, mas por uma ação relacionada, ou seja, é expressa uma necessidade e o Android identifica um aplicativo relacionado àquela ação e o chama para execução com os parâmetros indicados pela aplicação que a chamou. Pode ser usado para realizar uma chamada telefônica, utilizar um mapa, tocar uma música, entre outros. O arquivo de manifesto (configurações de acesso do aplicativo) pode conter definições de filtros sobre ações e arquivos. O Android recorre a esse arquivo para identificar os possíveis aplicativos para atender a uma chamada de *intent* implícita.

A figura a seguir apresenta o processo de execução de uma *intent* realizada pelo Android.



Processamento de chamadas de *intent* pelo Android.

Uso de *intents*

Como visto anteriormente, uma *intent* é um objeto. Como um objeto, tem atributos e métodos que podem definir parâmetros de configuração para a execução direcionada do aplicativo chamado. O nome do componente, como visto, pode ser definido de forma implícita ou explícita. A ação deve ser definida por meio de uma *string* contendo os parâmetros necessários para a execução da ação.

Formas de chamada

Existem várias formas de se iniciar uma ação a partir de uma *intent*, mas as mais comuns são:

- ACTION_VIEW – Usada quando alguma informação poderá ser exibida ao usuário, como uma foto ou uma view da atividade chamada.
- ACTION_SEND – Usada quando dados serão compartilhados entre os aplicativos, podendo ser um aplicativo de e-mail ou no uso de uma rede social.

Categorias

A *string* com as informações de execução da *intent* podem determinar como o componente será processado. Na maioria das vezes, uma *intent* não precisa determinar uma categoria, mas as categorias mais comuns para uso de *intents* são:

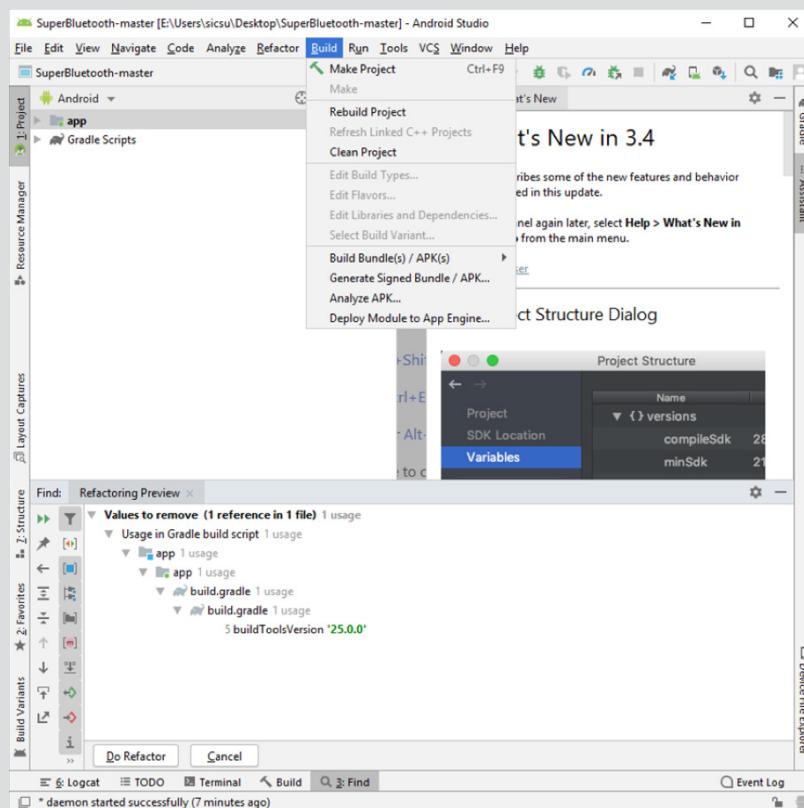
- CATEGORY_BROWSABLE – A atividade será executada por um navegador (browser), permitindo a exibição de fotos ou mensagens de e-mail.
- CATEGORY_LAUNCHER – A atividade relacionada à tarefa será listada no inicializador do aplicativo do sistema.



Dica

Não se esqueça de manter o seu Android Studio sempre atualizado e atualizar seus aplicativos caso seja necessário. Às vezes, nossos projetos podem necessitar de algumas atualizações ao realizarmos a atualização do Android Studio ou no caso de abrirmos o projeto em outros computadores. Nesses casos, você pode realizar um processo de correção, utilizando a opção *Build* do menu principal e executando as opções *Clean Project* e *Rebuild Project*.

Reorganizar projetos após atualizações ou mudança de computador.



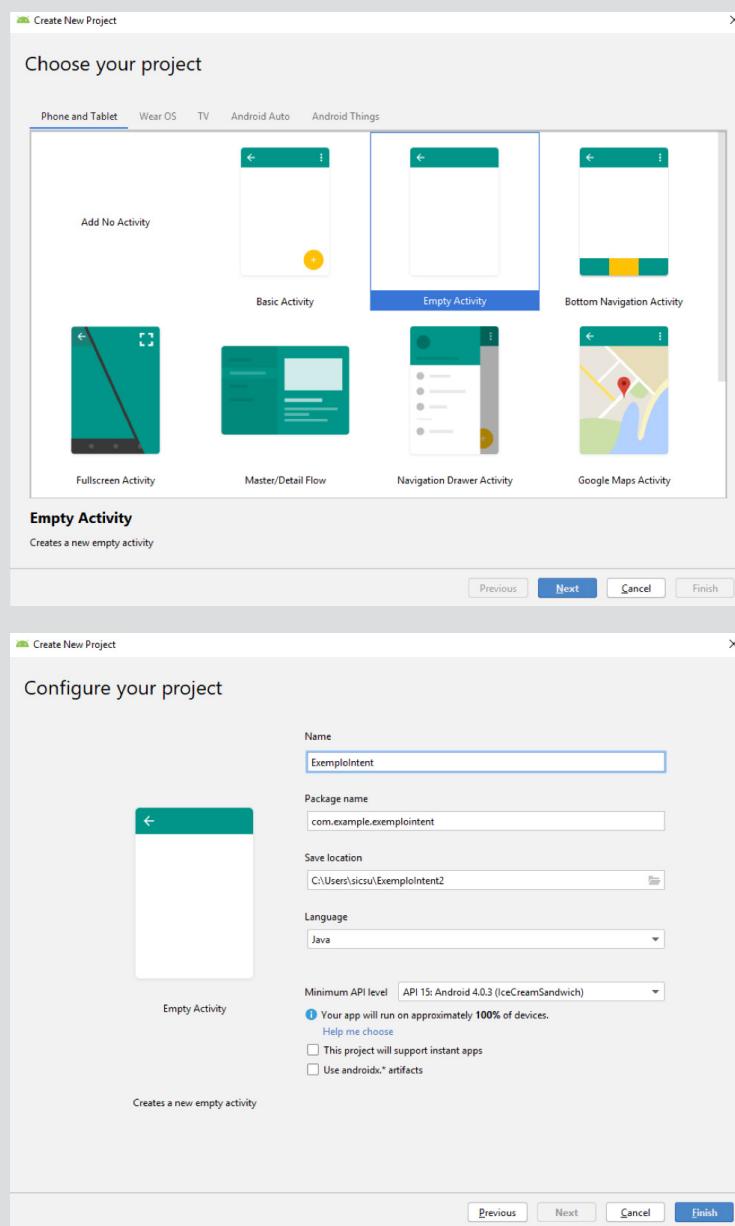
A partir do exemplo prático a seguir, será criado um aplicativo para a chamada de diversas outras ações, que estão relacionadas a outras aplicações dentro de um sistema Android.



Dica

Crie um novo projeto compatível com a API 15 e codificação em Java. O projeto terá o nome de **ExemploIntent**. Caso você altere os nomes da sua atividade, certifique-se de que fará o mesmo com a classe da atividade e com a chamada da view.

Criação do projeto **ExemploIntent**.



Criação da tela (*view*)

Layout e codificação da view do projeto:



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginEnd="180dp"
        android:layout_marginRight="180dp"
        android:layout_marginBottom="8dp"
        android:text="Uso de Intents:"/>
```

```
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="1.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.102" />

<Button
    android:id="@+id/button"
    android:layout_width="120dp"
    android:layout_height="70dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:onClick="chamaWeb"
    android:text="Página\nda Web."
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.189"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.286" />

<Button
    android:id="@+id/button2"
    android:layout_width="120dp"
    android:layout_height="70dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:onClick="chamaTelefone"
    android:text="Chamada \ntelefônica."
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.92"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.285" />
```

```

<Button
    android:id="@+id/button3"
    android:layout_width="120dp"
    android:layout_height="70dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:onClick="chamaToast"
    android:text="Mensagem\nflutuante."
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.541"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.739" />

<Button
    android:id="@+id/button4"
    android:layout_width="120dp"
    android:layout_height="70dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:onClick="chamaMapa"
    android:text="Google\nMaps."
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.189"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.499" />

<Button
    android:id="@+id/button5"
    android:layout_width="120dp"
    android:layout_height="70dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="28dp"
    android:layout_marginRight="28dp"
    android:layout_marginBottom="8dp"
    android:onClick="chamaSMS"

```

```

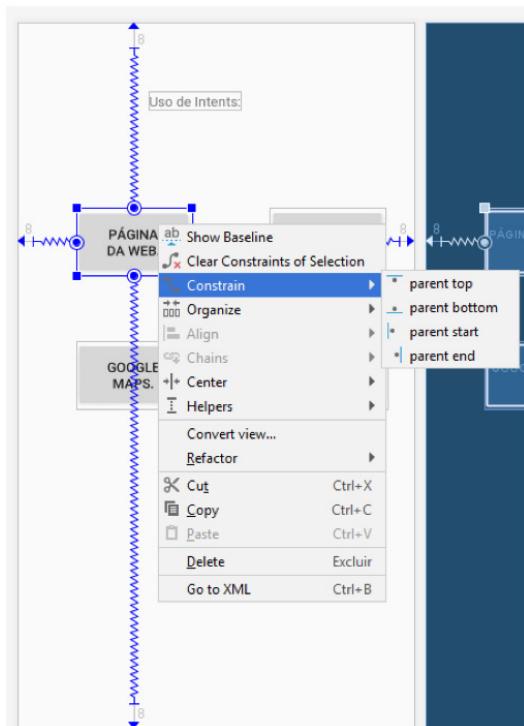
        android:text="SMS."
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.499" />

    </android.support.constraint.ConstraintLayout>

```

Notas:

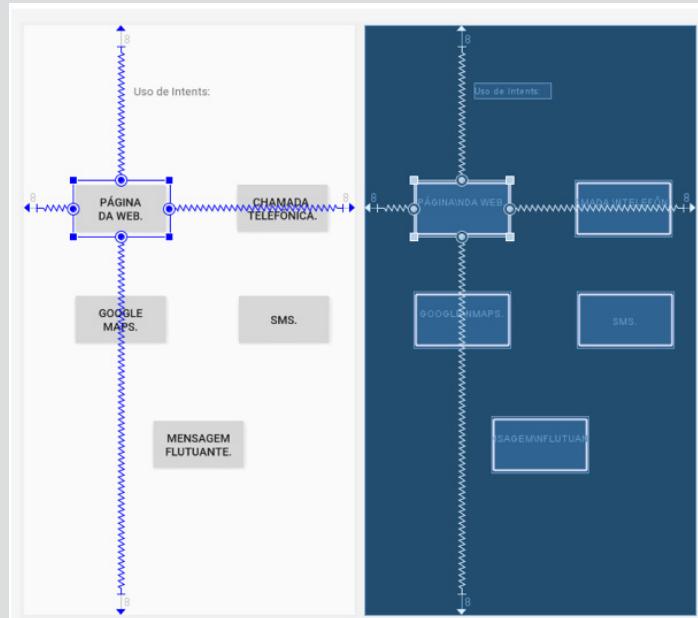
1. Para colocar os textos em duas ou mais linhas nos botões, use um `\n` para realizar a quebra da linha do texto.
2. Em todos os componentes, as *constraints* foram determinadas em função das margens-parente: *top*, *bottom*, *start* e *end*.





Dica

Se você preferir criar a sua própria view, não se esqueça de definir as *constraints* de cada componente da tela.



Determinação das *constraints*.

Codificação

Código de programação da aplicação (*controller*):

```
package com.example.exemplointent;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    Uri uri = null; // Objeto Uri para passagem dos parâmetros da chamada
    Intent intent = null; // Intent para chamada externa
    @Override
```

```

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

public void chamaWeb(View v){
    //Preparação dos parâmetros
    uri = Uri.parse("https://ilumno.com/pt");
    // Definição da Intent
    intent = new Intent(Intent.ACTION_VIEW, uri);
    // Chamada para execução pelo Intent
    startActivity(intent);
}

public void chamaTelefone(View v){
    //Preparação dos parâmetros
    uri = Uri.parse("tel:+5521900000001");
    // Definição da Intent
    intent = new Intent(Intent.ACTION_DIAL, uri);
    // Chamada para execução pelo Intent
    startActivity(intent);
}

public void chamaMapa(View v){
    //Preparação dos parâmetros
    uri = Uri.parse("geo:0,0?q=rua+ibiruna,rio de janeiro");
    // Definição da Intent
    intent = new Intent(Intent.ACTION_VIEW, uri);
    // Chamada para execução pelo Intent
    startActivity(intent);
}

public void chamaSMS(View v){    q

    //Preparação dos parâmetros
    uri = Uri.parse("sms: +5521900000001");
    // Definição da Intent
    intent = new Intent(Intent.ACTION_VIEW, uri);
    // Passagem de mensagem (dados) pelo Intent
    intent.putExtra("sms_body", "Corpo de conteúdo do SMS");
    // Chamada para execução pelo Intent
    startActivity(intent);
}

public void chamaToast(View v){
    // Chamada a Toast do Android
    Toast.makeText(getApplicationContext(), "Chamada Toast!", Toast.LENGTH_LONG).show();
}
}

```

Notas:

1. Para cada componente, foi definido o método de exceção na propriedade *OnClick* de cada botão na view.
2. O método *chamaWeb* utiliza como identificador uma chamada para “*https:*”, o que fará com que um navegador seja chamado para execução pelo sistema operacional (Android), com a página já determinada como parâmetro da chamada da *intent*.
3. O método *chamaTelefone* utiliza como identificador uma chamada para “*tel:*”, o que fará com que o aplicativo do telefone seja chamado, já com o número do telefone definido pelo parâmetro da chamada da *intent*.
4. O método *chamaMapa* utiliza como identificador uma chamada para “*geo:*”, o que fará com que seja chamado pelo Android um aplicativo de mapas, já com o endereço predeterminado pelo parâmetro da chamada.
5. O método *chamaSMS* utiliza como identificador uma chamada para “*sms:*”, o que fará com que o aplicativo de SMS seja chamado pelo Android. No parâmetro da chamada, foi determinado o número do telefone para o envio do SMS. Ainda foi definido um parâmetro extra para a chamada pela *intent*, definindo o corpo da mensagem.
6. Por final, o método *chamaToast* faz uma chamada a um componente do Android, que é responsável por apresentar uma mensagem sobre a tela, sendo o tempo padrão de três segundos.

Realize os seus testes e compare com os resultados obtidos.

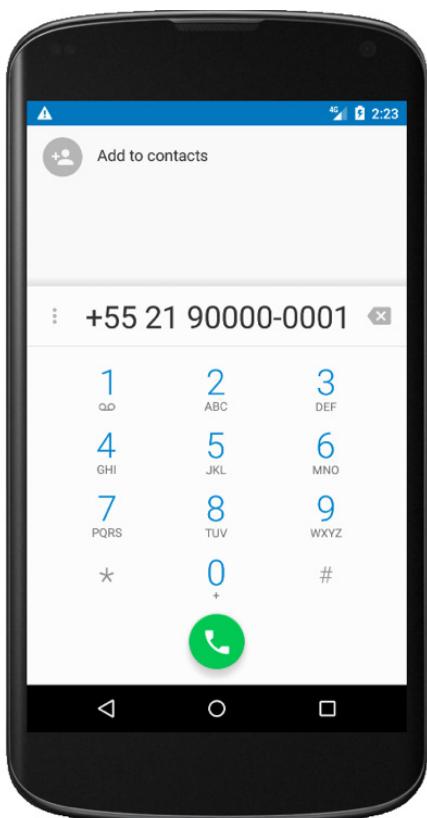
Testes realizados



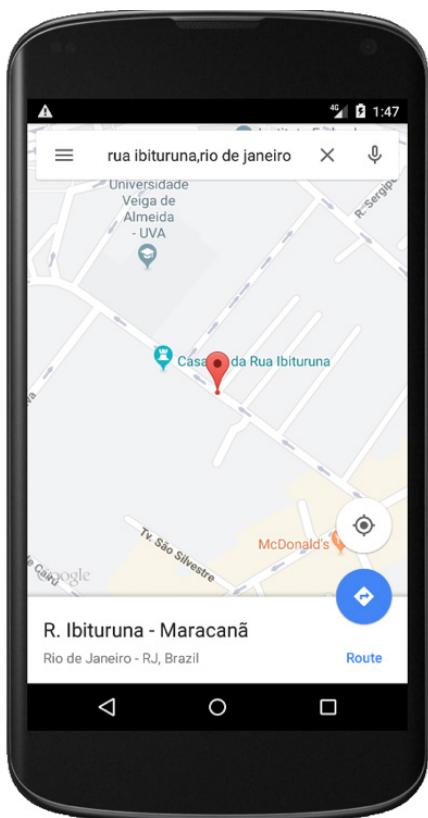
Tela principal.



Chamada para navegador.



Chamada para telefone.



Chamada para mapa.



Chamada para SMS, com texto.



Chamada para mensagem *toast*.

O exemplo prático apresentado permitirá que você possa aproveitar outros aplicativos do Android em seus aplicativos.



MEDIATECA

Acesse a midiateca da Unidade 3 e assista ao vídeo com conteúdo complementar indicado pelo professor a respeito do uso de *intents* implícitas para acesso a um navegador com parâmetros passados por outra aplicação.



Saiba mais

Ativação de outra atividade por meio de um objeto *intent* explícito

DEITEL, P. J.; DEITEL, H. M.; WALD, A. **Android 6 para programadores**: uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2016. Minha Biblioteca. cap. 4, seção 4.3.17, p. 119-120.

Envio e recebimento de mensagens no Android

Como vimos, por meio da *intent* podemos realizar a troca de mensagens entre atividades de uma mesma aplicação, assim como entre atividades de diferentes aplicações. Essa característica é muito importante, pois não é necessário desenvolver aplicativos “completos” e “fechados”, ou seja, totalmente desenvolvidos por você e sem qualquer tipo de comunicação. A comunicação entre atividades e aplicativos torna o desenvolvimento de aplicativos móveis menos complexo, uma vez que você não precisará desenvolver toda uma aplicação de mapas para simplesmente obter a posição geográfica do aparelho, bastando determinar uma *intent* com uma chamada ao aplicativo de mapas, que retornará a posição atual.

O uso da *intent*, então, abre uma quantidade ilimitada de possibilidades com a troca de mensagens entre atividades e aplicativos existentes. O desenvolvimento de um aplicativo passa então pelo reaproveitamento de todo e qualquer aplicativo existente no sistema, fazendo com que câmeras, geoposicionamento, telefone, SMS, e-mail, entre os tantos outros aplicativos, possam ser realizados por outras aplicações, sem a necessidade de desenvolvimento específico, trabalhoso e complexo.

A troca de mensagens entre atividades e aplicativos se torna indispensável para o desenvolvimento de aplicativos robustos, estabelecendo comunicações com troca de mensagens.

Podemos, então, desenvolver aplicações que irão utilizar a troca de mensagens para completar determinadas ações a partir de diferentes aplicativos.

Por meio do exemplo prático a seguir, será criado um aplicativo para a entrada de dados e cálculo do IMC de uma pessoa, mas os dados e os resultados serão enviados por e-mail, mediante uma chamada de *intent* ao aplicativo de e-mail do Android:



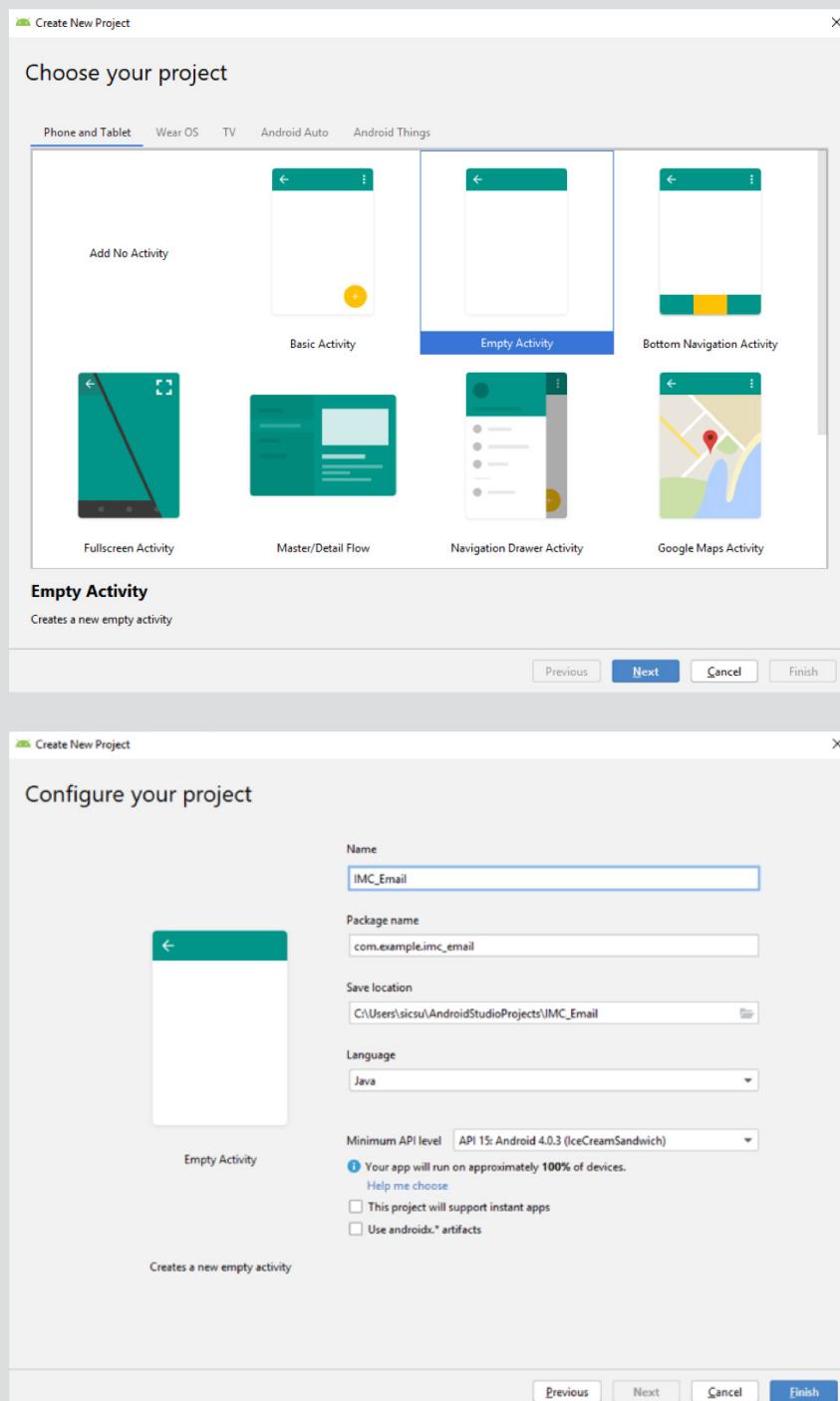
Importante

É necessário que você configure um e-mail válido no Gmail do emulador (um emulador simula um aparelho original; neste exemplo, o e-mail será realmente enviado pelo seu emulador). O e-mail configurado deve ser o mesmo que você irá usar no envio através do seu aplicativo. Não se esqueça de colocá-lo como remetente no aplicativo.



Dica

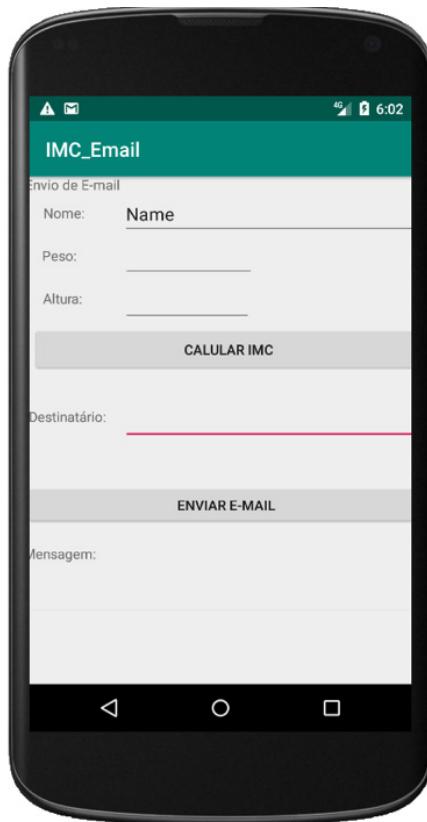
Crie um novo projeto compatível com a API 15 e codificação em Java. Esse projeto terá o nome de **IMC_Email**. Caso você altere os nomes da sua atividade, certifique-se de que fará o mesmo com a classe da atividade e com a chamada da view.



Criação do projeto **IMC_Email**.

Criação da tela (view)

Layout e codificação da view do projeto:



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.imc_email"
    tools:layout_editor_absoluteX="0dp"
    tools:layout_editor_absoluteY="81dp">

    <TextView
        android:id="@+id/textView4"
        android:layout_width="230dp"
        android:layout_height="20dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="190dp"
        android:layout_marginEnd="173dp"
        ...>
```

```
    android:layout_marginRight="173dp"
    android:layout_marginBottom="686dp"
    android:layout_weight="1"
    android:text="Envio de E-mail"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText"
    android:layout_width="302dp"
    android:layout_height="50dp"
    android:layout_marginStart="107dp"
    android:layout_marginLeft="107dp"
    android:layout_marginTop="300dp"
    android:layout_marginEnd="2dp"
    android:layout_marginRight="2dp"
    android:layout_marginBottom="327dp"
    android:ems="10"
    android:inputType="textEmailAddress"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="12dp"
    android:layout_marginLeft="12dp"
    android:layout_marginTop="315dp"
    android:layout_marginEnd="322dp"
    android:layout_marginRight="322dp"
    android:layout_marginBottom="343dp"
    android:text="Destinatário:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button1"
    android:layout_width="403dp"
    android:layout_height="43dp"
    android:layout_marginStart="9dp"
```

```
    android:layout_marginLeft="9dp"
    android:layout_marginTop="330dp"
    android:layout_marginBottom="180dp"
    android:onClick="enviarEmail"
    android:text="Enviar E-mail"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="12dp"
    android:layout_marginLeft="12dp"
    android:layout_marginTop="480dp"
    android:layout_marginEnd="337dp"
    android:layout_marginRight="337dp"
    android:layout_marginBottom="232dp"
    android:text="Mensagem:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="395dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="400dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="67dp"
    android:ems="10"
    android:enabled="false"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button"
    android:layout_width="395dp"
```

```
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="270dp"
    android:layout_marginBottom="433dp"
    android:onClick="calcularIMC"
    android:text="Calcular IMC"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="20dp"
    android:layout_marginStart="27dp"
    android:layout_marginLeft="27dp"
    android:layout_marginTop="210dp"
    android:layout_marginEnd="342dp"
    android:layout_marginRight="342dp"
    android:layout_marginBottom="647dp"
    android:text="Nome:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView6"
    android:layout_width="wrap_content"
    android:layout_height="20dp"
    android:layout_marginStart="26dp"
    android:layout_marginLeft="26dp"
    android:layout_marginTop="240dp"
    android:layout_marginEnd="350dp"
    android:layout_marginRight="350dp"
    android:layout_marginBottom="591dp"
    android:text="Peso:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView7"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_marginStart="26dp"
    android:layout_marginLeft="26dp"
    android:layout_marginTop="250dp"
    android:layout_marginEnd="344dp"
    android:layout_marginRight="344dp"
    android:layout_marginBottom="516dp"
    android:text="Altura:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<EditText
    android:id="@+id/editText4"
    android:layout_width="305dp"
    android:layout_height="43dp"
    android:layout_marginStart="107dp"
    android:layout_marginLeft="107dp"
    android:layout_marginTop="200dp"
    android:layout_marginBottom="633dp"
    android:ems="10"
    android:inputType="textPersonName"
    android:text="Name"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<EditText
    android:id="@+id/editText5"
    android:layout_width="133dp"
    android:layout_height="49dp"
    android:layout_marginStart="107dp"
    android:layout_marginLeft="107dp"
    android:layout_marginTop="220dp"
    android:layout_marginEnd="171dp"
    android:layout_marginRight="171dp"
    android:layout_marginBottom="573dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<EditText
    android:id="@+id/editText6"
```

```

        android:layout_width="130dp"
        android:layout_height="50dp"
        android:layout_marginStart="107dp"
        android:layout_marginLeft="107dp"
        android:layout_marginTop="240dp"
        android:layout_marginEnd="174dp"
        android:layout_marginRight="174dp"
        android:layout_marginBottom="505dp"
        android:ems="10"

        android:inputType="numberDecimal"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

```

Notas:

1. Não se esqueça de definir o evento *OnClick* dos botões (*calcularIMC* e *enviarEmail*).
2. A determinação do remetente será feita diretamente na codificação, não sendo necessário um *EditText* para recebê-lo.



Observação

Não se esqueça de determinar as *constraints* dos componentes.

Codificação

Código da programação da aplicação (*controller*):

```

package com.example.imc_email;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

```

```

import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    EditText edtNome, edtPeso, edtAltura, edtDest, edtMensagem;
    // Declaração como atributo do imc para que possa ser acessado
    // por todos os métodos
    double imc =0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        edtNome = (EditText) findViewById(R.id.editText4);
        edtPeso = (EditText) findViewById(R.id.editText5);
        edtAltura = (EditText) findViewById(R.id.editText6);
        edtDest = (EditText) findViewById(R.id.editText);
        edtMensagem = (EditText) findViewById(R.id.editText3);
    }

    public void calcularIMC(View v) {
        double peso, altura;
        // Faz a entrada de dados e o cálculo
        // Foi incluído o tratamento de exceções para proteger o código
        try {
            // Leitura dos dados nos EditTexts
            altura = Double.parseDouble(edtAltura.getText().toString());
            peso = Double.parseDouble(edtPeso.getText().toString());
            // Cálculo do IMC
            this.imc = peso / Math.pow(altura, 2);
            // Apresentação do resultado pelo Toast
            Toast toast = Toast.makeText(getApplicationContext(), "IMC = " +
String.format("%.2f", this.imc), Toast.LENGTH_LONG);
            toast.show();
        } catch (Exception e) {
            // Tratamento da exceção com aviso através do Toast
            Toast toast = Toast.makeText(getApplicationContext(), "Ocorreu al-
gum problema nos dados da Pessoa!", Toast.LENGTH_LONG);
            toast.show();
        }
    }

    public void enviarEmail(View v) {

        Intent emailIntent = new Intent(Intent.ACTION_SENDTO);
        //Defina aqui o endereço de e-mail do remetente:
    }
}

```

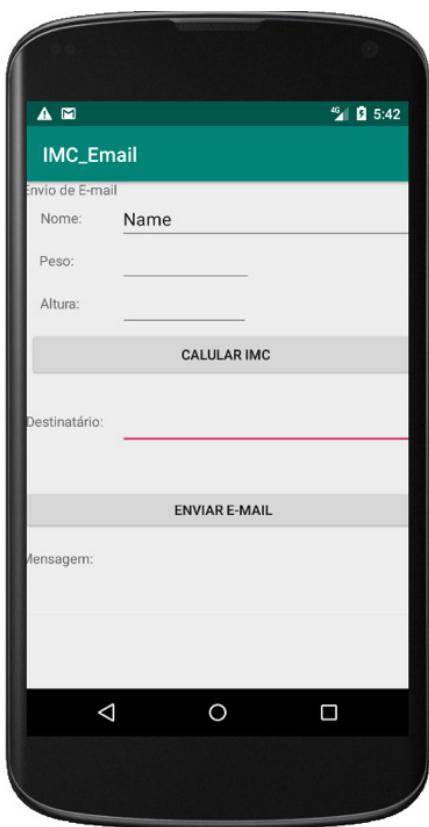
```
emailIntent.setData(Uri.parse("mailto:" + "email.remetente@hotmail.com"));
//Busca o endereço do destinatário no editText:
emailIntent.setData(Uri.parse("mailto:" + edtDest.getText()));
//Define o assunto do e-mail:
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Avaliação do IMC.");
//Confirma que o imc foi calculado:
calcularIMC(v);
//Prepara os dados do conteúdo do e-mail:
String conteudo;
conteudo = "Nome: " + edtNome.getText();
conteudo += "\n Peso: " + edtPeso.getText();
conteudo += "\nAltura: " + edtAltura.getText();
conteudo += "\nImc = " + String.format("%.2f", this.imc);
//Apresenta o conteúdo na tela do aplicativo:
edtMensagem.setText(conteudo);
//Define o conteúdo do e-mail:
emailIntent.putExtra(Intent.EXTRA_TEXT, conteudo);
//Chama para execução o aplicativo de e-mail:
try {
startActivity(Intent.createChooser(emailIntent, "Enviando E-mail..."));
} catch (android.content.ActivityNotFoundException ex) {
Toast.makeText(MainActivity.this, "Não existem clientes configurados!", Toast.LENGTH_SHORT).show();
}
}
```

Notas:

1. Foram criados dois botões, um apenas para calcular o IMC e outro para o envio do e-mail.
 2. Caso algum botão não funcione, verifique se o evento *OnCLick* foi configurado na view.
 3. Certifique-se de ter colocado um e-mail de remetente válido no código do método *EnviarEmail*.
 4. Realize os testes.

Realize os seus testes e compare com os resultados obtidos.

Testes realizados



Tela principal.



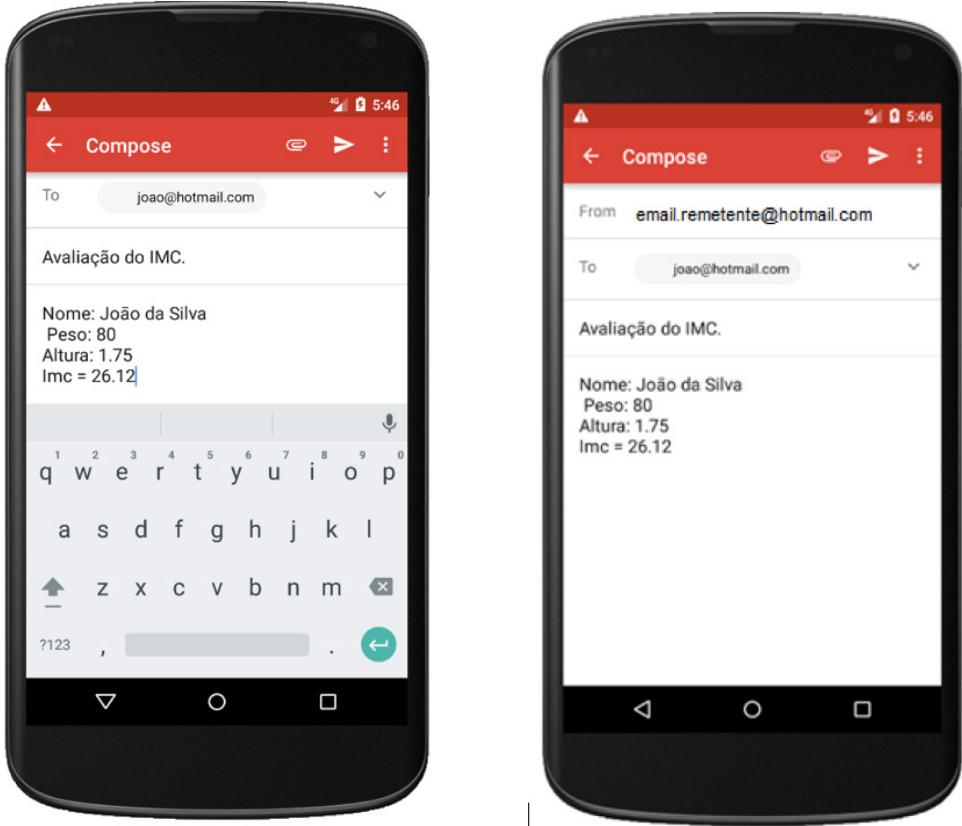
Preenchimento dos dados.



Cálculo do IMC.

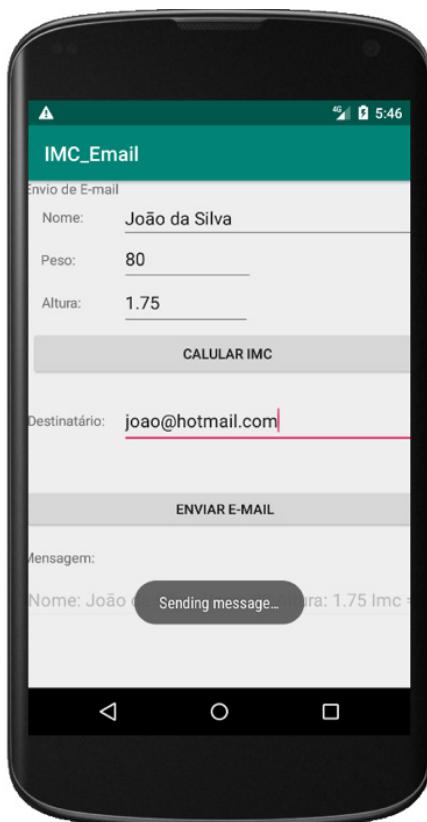


Carregando o aplicativo de e-mail.



O e-mail pode ser editado, mas já recebeu os dados de destinatário, assunto e conteúdo.

Com o e-mail pronto e o remetente definido na aplicação que chamou, basta enviar.



A mensagem está sendo enviada!

O aplicativo recebe os dados, calcula o IMC e envia os dados e o valor calculado do IMC por e-mail para a pessoa. Preparar aplicativos com o uso de e-mail não é tão complicado quando utilizamos uma *intent* para a troca de mensagens entre atividades e aplicativos no Android. É possível realizar a tarefa de forma mais transparente, mas a ideia do exemplo é que você perceba o que realmente acontece. Configure um e-mail de remetente válido no Android, assim como um e-mail de destinatário também válido, e você poderá conferir que o seu aplicativo irá realmente enviar o e-mail para o destinatário.

Podemos observar claramente que o desenvolvimento foi muito mais simples, uma vez que utilizamos uma *intent* para o envio do e-mail. A aplicação se tornou simples e fácil de desenvolver com o reaproveitamento de aplicativos já existentes.



MEDIATECA

Acesse a midiateca da Unidade 3 e veja o guia com conteúdo complementar indicado pelo professor sobre o uso de *intents* para enviar mensagens de uma atividade para outra e como trabalhar com os *intent filters*.



Saiba mais

Tratamento de exceções na linguagem Java

HORSTMANN, C. S. **Conceitos de computação com Java**: compatível com Java 5 & 6. Porto Alegre: Bookman, 2009. Minha Biblioteca. cap. 11, seções 11.2 a 11.6, p. 455-466.

Navegação entre telas e envio de parâmetros

Aplicativos, normalmente, têm mais de uma tela (view). Essas telas, muitas vezes, precisam trocar dados entre si. Até o momento trabalhamos com apenas uma tela, e essa tarefa não foi necessária. Porém, a partir deste momento, nossos aplicativos passarão a trabalhar com mais de uma tela. Dessa forma, a troca de mensagens entre diferentes telas (*views*) em nossos aplicativos se torna muito importante. Vamos ver agora como realizar a troca de mensagens entre as diferentes atividades. A troca de mensagens entre atividades ainda permite o reaproveitamento de código, uma vez que uma atividade desenvolvida para um projeto pode ser reaproveitada em outro, sem a necessidade de um novo desenvolvimento. No Android é possível criar atividades como componentes reutilizáveis e, assim, reaproveitá-las em diferentes projetos. Assim, uma atividade pode determinar um conjunto de dados que deverão ser repassados a outras atividades, e essas atividades então deverão apenas estar preparadas para receber esses dados. Assim, a troca de mensagens se torna mais importante no desenvolvimento de nossos projetos de aplicativos.

Como as *intents* chamarão atividades por meio dos nomes das classes, você verá como trabalhar com *intents* explícitas.

A troca de mensagens depende do objeto *intent*, pois é por meio dele que dados são passados para as outras atividades a partir do uso de métodos *put*, para enviar, e de métodos *get*, para receber. Existem diferentes métodos *put* e *get*, cada um voltado para uma forma de uso ou tipo de dado. É importante você identificar claramente o tipo de dado, mas não se esqueça de que você ainda pode passar os dados convertendo para texto, no envio, e convertendo para o tipo adequado na recepção. No exemplo a seguir, iremos trabalhar com os diferentes tipos para envio e recebimento para conhecer melhor esses métodos e suas variações.

O mais comum é o envio de dados de uma página para outra, mas sem que ocorra o retorno à primeira. Para termos uma noção melhor, o exemplo a seguir fará o envio de dados da primeira atividade para a segunda, mas também receberá de volta dados da segunda para a primeira.

Chamadas de *intent* para troca de mensagens

- Chamadas somente para envio são mais simples. Nesse tipo de chamada, é possível enviar os dados a partir de métodos *put*, como no exemplo a seguir, usado em nosso primeiro projeto da unidade. A *intent* é criada, os dados são transmitidos por métodos *put* e a *intent* é executada.

```
//Preparação dos parâmetros
uri = Uri.parse("sms: +552190000001");
// Definição da Intent
intent = new Intent(Intent.ACTION_VIEW, uri);
// Passagem de mensagem (dados) pelo Intent
intent.putExtra("sms_body", "Corpo de conteúdo do SMS");
// Chamada para execução pelo Intent
startActivity(intent);
```

- Chamadas para envio e recebimento são mais complexas. Nesse tipo de chamada, é possível enviar os dados por meio de métodos *put* e receber de volta dados da atividade chamada. A única diferença é que devemos identificar a atividade por meio de um número, pois podemos ter mais de uma atividade sendo chamada por uma mesma atividade principal. A identificação então é necessária para que se possa diferenciar os dados de retorno de cada atividade. O método *onActivityResult* é o responsável por tratar os dados retornados por uma atividade, sendo necessário, primeiro, identificar a atividade, já que uma mesma atividade pode chamar mais de uma atividade e receber retorno de todas.

```
// definição do número da Intent para retorno
static final int ACTIVITY_REQUEST_IMC = 1;

// Preparação da Intent com previsão de retorno de dados
intent1 = new Intent(getApplicationContext(), CalculaIMC.class);
// Envia o nome para a segunda atividade
intent1.putExtra("nome", ed1.getText().toString());
// chama a atividade aguardando valores de retorno
startActivityForResult(intent1, ACTIVITY_REQUEST_IMC);

protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
```

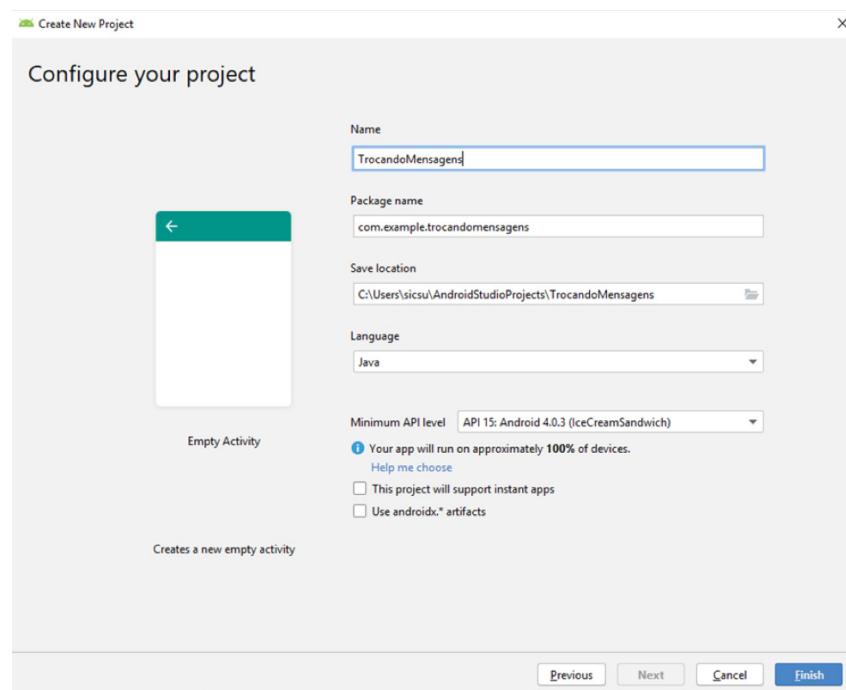
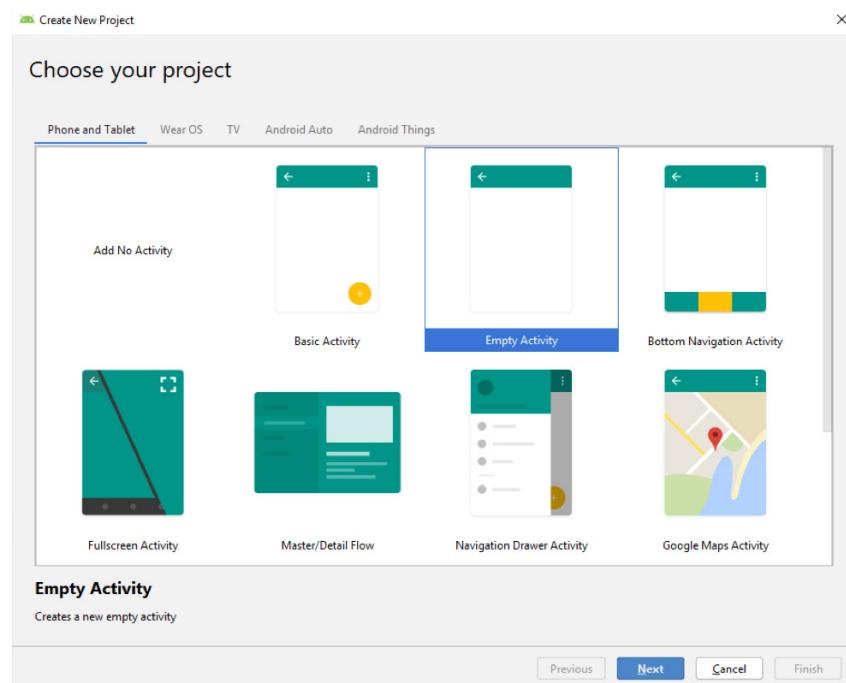
```

// Este método é responsável por tratar as requisições de re-
torno
// cada requisição de retorno é diferenciada pelo seu número
// em ACTIVITY _ REQUEST
if (requestCode == ACTIVITY _ REQUEST _ IMC) {
    if(resultCode == RESULT _ OK){
        double imc = data.getDoubleExtra("imc",0);
        //Coloque no EditText
        ed2.setText(String.format("%.2f", imc));
    }
}
if (requestCode == ACTIVITY _ REQUEST _ SITUACAO) {
    if(resultCode == RESULT _ OK){
        String situacao = data.getStringExtra("situacao");
        //Coloque no EditText
        ed3.setText(situacao);
    }
}

```

Vamos dividir nosso projeto do cálculo do IMC em diferentes atividades para podermos trabalhar com o envio, recebimento e retorno de dados entre diferentes atividades. A primeira atividade (*main/principal*) receberá apenas o nome da pessoa e passará esse nome para a segunda atividade (cálculo do IMC), que receberá o nome, solicitará o peso e a altura, fará o cálculo do IMC e retornará à primeira atividade o valor do IMC calculado. De volta à atividade principal (*main*), o usuário poderá chamar a terceira atividade (verificar situação), que enviará o nome e o valor do IMC. A terceira atividade receberá o nome da pessoa e o valor do IMC e solicitará a idade e o sexo. De posse do valor do IMC, sexo e idade, será verificada a situação da pessoa e a atividade enviará a situação à atividade principal. De volta à atividade principal, a situação será recebida e exibida na tela principal.

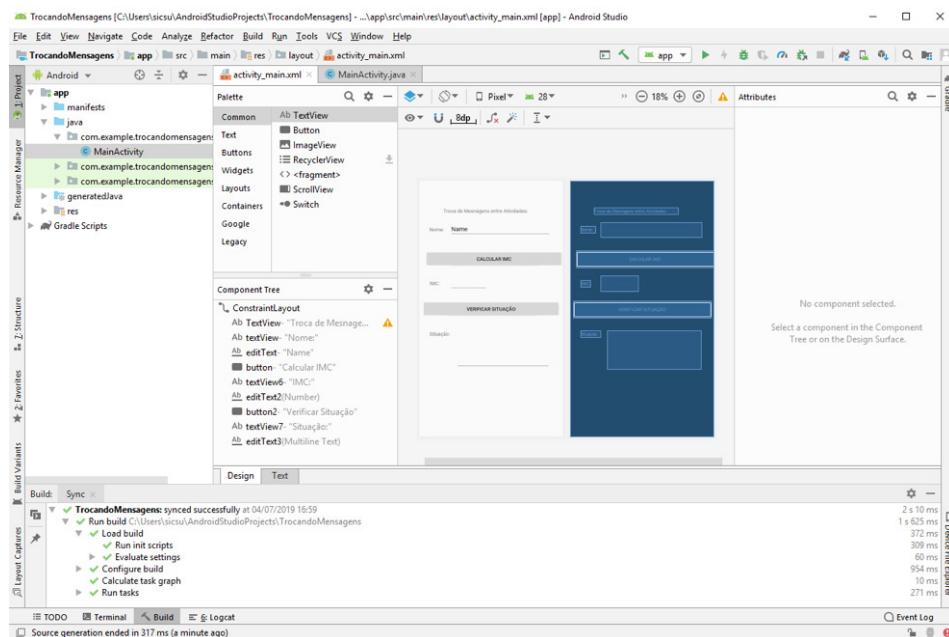
Exemplo prático: projeto **TrocandoMensagens**.



Criação do projeto **TrocandoMensagens**.

Criação da tela principal, primeira atividade (activity_main.xml)

Layout e codificação da view principal do projeto:



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```

    android:layout_height="match_parent"
    tools:context=".MainActivity">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="149dp"
    android:layout_marginLeft="149dp"
    android:layout_marginTop="100dp"
    android:layout_marginEnd="185dp"
    android:layout_marginRight="185dp"
    android:layout_marginBottom="677dp"
    android:text="Troca de Mensagens entre Atividades:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.49"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.369" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="28dp"
    android:layout_marginLeft="28dp"
    android:layout_marginTop="129dp"
    android:layout_marginEnd="341dp"
    android:layout_marginRight="341dp"
    android:layout_marginBottom="583dp"
    android:text="Nome:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText"
    android:layout_width="289dp"
    android:layout_height="42dp"
    android:layout_marginStart="87dp"
    android:layout_marginLeft="87dp"
    android:layout_marginTop="118dp"
    android:layout_marginEnd="35dp"
    android:layout_marginRight="35dp"
    android:layout_marginBottom="571dp"
    android:ems="10"

```

```
    android:inputType="textPersonName"
    android:text="Name"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button"
    android:layout_width="395dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="199dp"
    android:layout_marginBottom="484dp"
    android:onClick="chamarCalcularIMC"
    android:text="Calcular IMC"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView6"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="29dp"
    android:layout_marginLeft="29dp"
    android:layout_marginTop="283dp"
    android:layout_marginEnd="354dp"
    android:layout_marginRight="354dp"
    android:layout_marginBottom="429dp"
    android:text="IMC:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="107dp"
    android:layout_height="44dp"
    android:layout_marginStart="87dp"
    android:layout_marginLeft="87dp"
    android:layout_marginTop="271dp"
    android:layout_marginEnd="217dp"
    android:layout_marginRight="217dp"
```

```
    android:layout_marginBottom="416dp"
    android:editable="false"
    android:ems="10"
    android:inputType="number"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button2"
    android:layout_width="395dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="342dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="341dp"
    android:onClick="chamarVerificarSituacao"
    android:text="Verificar Situação"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView7"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="29dp"
    android:layout_marginLeft="29dp"
    android:layout_marginTop="428dp"
    android:layout_marginEnd="324dp"
    android:layout_marginRight="324dp"
    android:layout_marginBottom="284dp"
    android:text="Situação:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="268dp"
    android:layout_height="111dp"
    android:layout_marginStart="106dp"
```

```

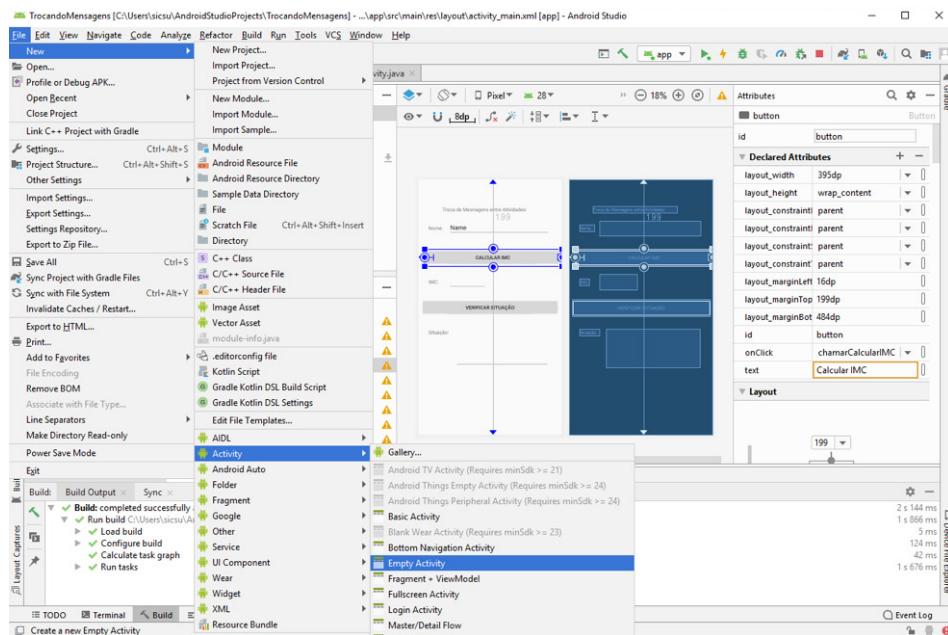
        android:layout_marginLeft="106dp"
        android:layout_marginTop="427dp"
        android:layout_marginEnd="37dp"
        android:layout_marginRight="37dp"
        android:layout_marginBottom="193dp"
        android:ems="10"
        android:gravity="start|top"
        android:inputType="textMultiLine"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

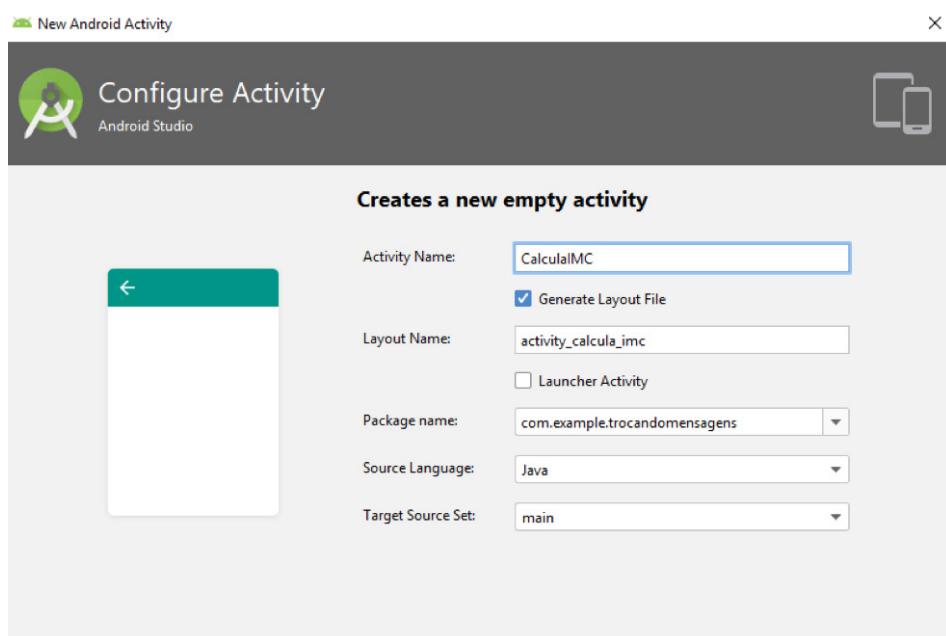
```

Criação da tela da segunda atividade, para o cálculo do IMC (activity_calcula_imc.xml)

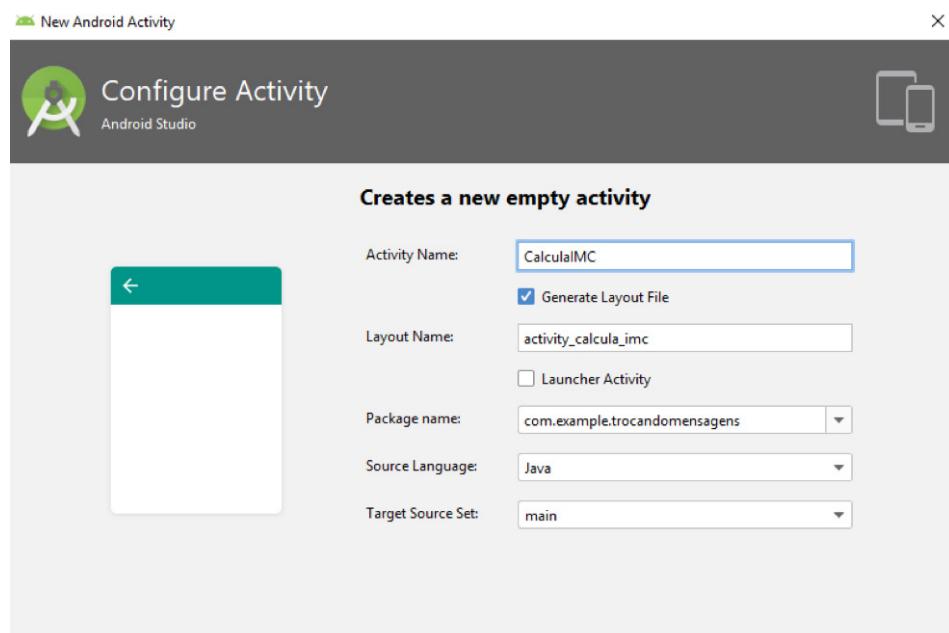
1. Selecionar o projeto app e clicar com o botão direito do mouse em *File/New/Activity/Empty Activity*:



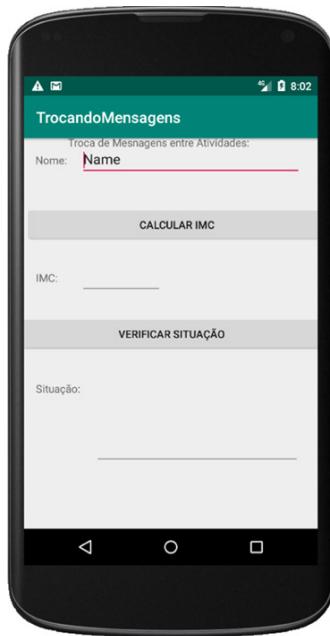
2. Definir o nome da atividade: CalculaIMC.



3. A nova ativiade é incorporada ao projeto:



Layout e codificação da view de cálculo do IMC do projeto:



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".CalculaIMC">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="28dp"
        android:layout_marginLeft="28dp"
        android:layout_marginTop="129dp"
        android:layout_marginEnd="341dp"
        android:layout_marginRight="341dp"
        android:layout_marginBottom="583dp"
        android:text="Nome:"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView2"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="28dp"
    android:layout_marginLeft="28dp"
    android:layout_marginTop="180dp"
    android:layout_marginEnd="349dp"
    android:layout_marginRight="349dp"
    android:layout_marginBottom="532dp"
    android:text="Peso:"/>
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="145dp"
    android:layout_marginLeft="145dp"
    android:layout_marginTop="180dp"
    android:layout_marginEnd="225dp"
    android:layout_marginRight="225dp"
    android:layout_marginBottom="532dp"
    android:text="Altura:"/>
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText"
    android:layout_width="289dp"
    android:layout_height="42dp"
    android:layout_marginStart="87dp"
    android:layout_marginLeft="87dp"
    android:layout_marginTop="118dp"
    android:layout_marginEnd="35dp"
    android:layout_marginRight="35dp"
    android:layout_marginBottom="571dp"
    android:ems="10"
    android:inputType="textPersonName"
    android:text="Name"/>
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```

<EditText
    android:id="@+id/editText2"
    android:layout_width="64dp"
    android:layout_height="39dp"
    android:layout_marginStart="71dp"
    android:layout_marginLeft="71dp"
    android:layout_marginTop="171dp"
    android:layout_marginEnd="276dp"
    android:layout_marginRight="276dp"
    android:layout_marginBottom="521dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="67dp"
    android:layout_height="41dp"
    android:layout_marginStart="187dp"
    android:layout_marginLeft="187dp"
    android:layout_marginTop="171dp"
    android:layout_marginEnd="157dp"
    android:layout_marginRight="157dp"
    android:layout_marginBottom="519dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button"
    android:layout_width="395dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="268dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="415dp"
    android:onClick="calcularVoltar"
    android:text="Calcular e Voltar"
    app:layout_constraintBottom_toBottomOf="parent"

```

```

    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

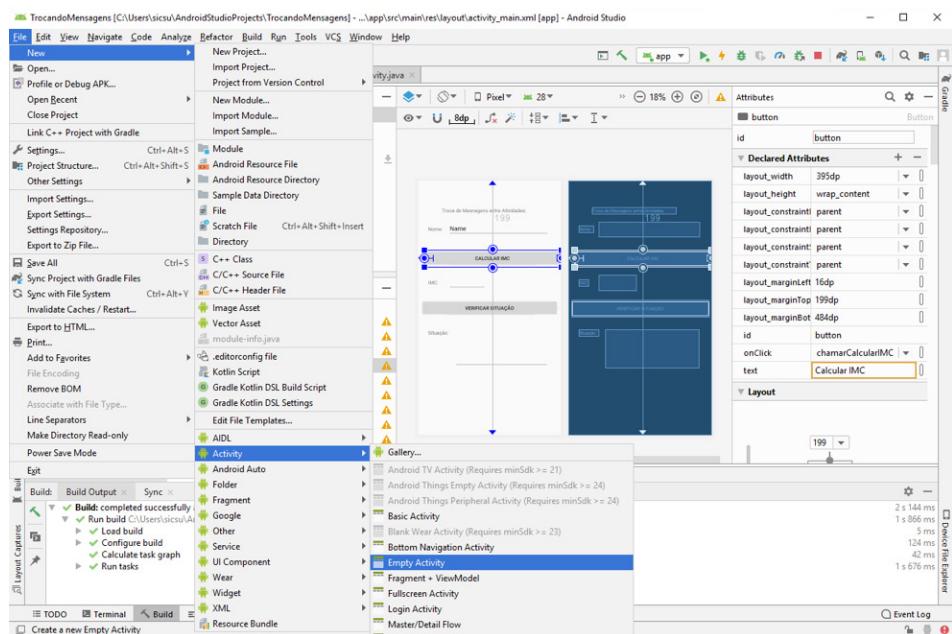
</android.support.constraint.ConstraintLayout>

```

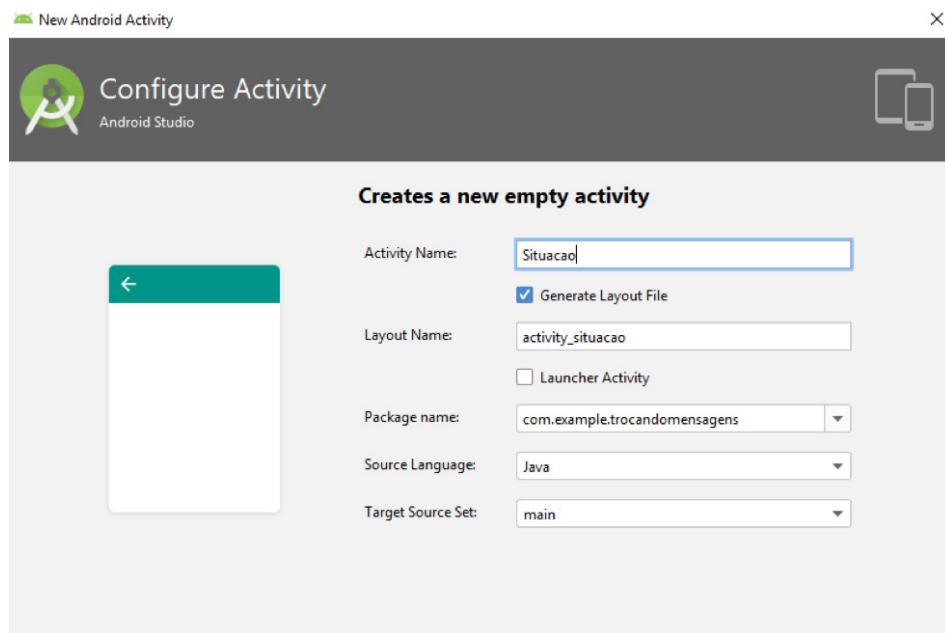
Criação da tela da terceira atividade, para verificação da situação da pessoa (*activity_situacao.xml*)

Processo de criação da nova atividade para a verificação da situação da pessoa no Android Studio:

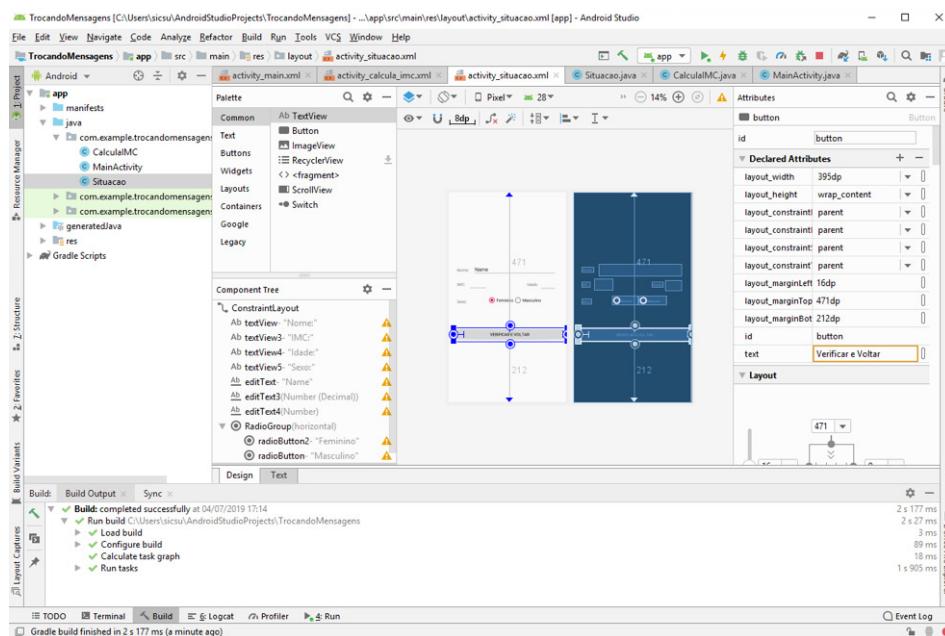
1. Selecionar o projeto app e clicar com o botão direito do mouse em File/New/Activity/Empty Activity:



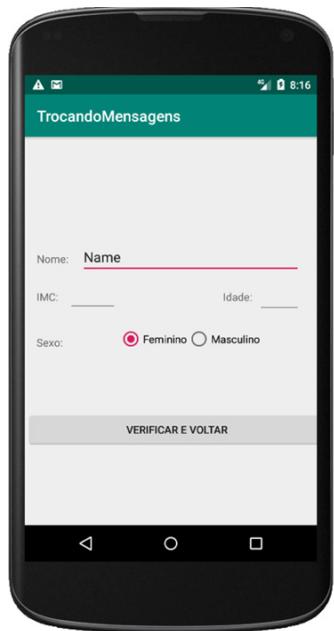
2. Definir o nome da atividade: Situacao.



3. A nova atividade é incorporada ao projeto:



Layout e codificação da view de verificação da situação da pessoa do projeto:



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Situacao">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="29dp"
        android:layout_marginLeft="29dp"
        android:layout_marginTop="261dp"
        android:layout_marginEnd="340dp"
        android:layout_marginRight="340dp"
        android:layout_marginBottom="451dp"
        android:text="Nome:"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView3"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="29dp"
    android:layout_marginLeft="29dp"
    android:layout_marginTop="311dp"
    android:layout_marginEnd="354dp"
    android:layout_marginRight="354dp"
    android:layout_marginBottom="401dp"
    android:text="IMC:"/>
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="274dp"
    android:layout_marginLeft="274dp"
    android:layout_marginTop="311dp"
    android:layout_marginEnd="99dp"
    android:layout_marginRight="99dp"
    android:layout_marginBottom="401dp"
    android:text="Idade:"/>
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="29dp"
    android:layout_marginLeft="29dp"
    android:layout_marginTop="371dp"
    android:layout_marginEnd="348dp"
    android:layout_marginRight="348dp"
    android:layout_marginBottom="341dp"
    android:text="Sexo:"/>
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```
<EditText
    android:id="@+id/editText"
    android:layout_width="289dp"
    android:layout_height="42dp"
    android:layout_marginStart="87dp"
    android:layout_marginLeft="87dp"
    android:layout_marginTop="249dp"
    android:layout_marginEnd="35dp"
    android:layout_marginRight="35dp"
    android:layout_marginBottom="440dp"
    android:ems="10"
    android:inputType="textPersonName"
    android:text="Name"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="64dp"
    android:layout_height="39dp"
    android:layout_marginStart="71dp"
    android:layout_marginLeft="71dp"
    android:layout_marginTop="302dp"
    android:layout_marginEnd="276dp"
    android:layout_marginRight="276dp"
    android:layout_marginBottom="390dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="56dp"
    android:layout_height="36dp"
    android:layout_marginStart="320dp"
    android:layout_marginLeft="320dp"
    android:layout_marginTop="307dp"
    android:layout_marginEnd="35dp"
    android:layout_marginRight="35dp"
    android:layout_marginBottom="388dp"
    android:ems="10"
```

```

        android:inputType="number"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="135dp"
    android:layout_marginLeft="135dp"
    android:layout_marginTop="360dp"
    android:layout_marginEnd="87dp"
    android:layout_marginRight="87dp"
    android:layout_marginBottom="339dp"
    android:orientation="horizontal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">
    <RadioButton
        android:id="@+id radioButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:checked="true"
        android:text="Feminino" />

    <RadioButton
        android:id="@+id radioButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Masculino" />
</RadioGroup>

<Button
    android:id="@+id/button"
    android:layout_width="395dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="471dp"
    android:layout_marginBottom="212dp"
    android:onClick="verificarVoltar"
    android:text="Verificar e Voltar"
    app:layout_constraintBottom_toBottomOf="parent"

```

```

    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

```



Observação

Não se esqueça de determinar as *constraints* dos componentes.

Codificação

Código da programação da primeira atividade (principal): *MainActivity*.

```

package com.example.trocandomensagens;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    Intent intent1, intent2;
    EditText ed1, ed2, ed3;
    // define os valores para o request de cada atividade
    static final int ACTIVITY_REQUEST_IMC = 1;
    static final int ACTIVITY_REQUEST_SITUACAO = 2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ed1 = (EditText) findViewById(R.id.editText);
        ed2 = (EditText) findViewById(R.id.editText2);
        ed3 = (EditText) findViewById(R.id.editText3);
    }

    protected void onActivityResult(int requestCode, int resultCode,
Intent data) {

```

```

    if (requestCode == ACTIVITY_REQUEST_IMC) {
        if(resultCode == RESULT_OK){
            double imc = data.getDoubleExtra("imc",0);
            //Coloca o valor do imc no EditText
            ed2.setText(String.format("%.2f", imc));
        }
    }
    if (requestCode == ACTIVITY_REQUEST_SITUACAO) {
        if(resultCode == RESULT_OK){
            String situacao = data.getStringExtra("situacao");
            //Coloca a situação da pessoa no EditText
            ed3.setText(situacao);
        }
    }
}

public void chamarCalcularIMC(View v){

    intent1 = new Intent(getApplicationContext(), CalculaIMC.class);
    // Envia o nome para a segunda atividade
    intent1.putExtra("nome", ed1.getText().toString());
    // chama a segunda atividade aguardando valores de retorno
    // no request = 1
    startActivityForResult(intent1, ACTIVITY_REQUEST_IMC);
}

public void chamarVerificarSituacao(View v){
    intent2 = new Intent(getApplicationContext(), Situacao.class);
    // Envia o nome e o imc para a segunda atividade
    intent2.putExtra("nome", ed1.getText().toString());
    intent2.putExtra("imc", Double.parseDouble(ed2.getText().toString()));
    // chama a segunda atividade aguardando valores de retorno
    // no request = 2
    startActivityForResult(intent2, ACTIVITY_REQUEST_SITUACAO);
}
}

```

Notas:

1. Não se esqueça de definir os nomes dos métodos dos botões no evento *OnClick* dos componentes da view.
2. Foram criadas duas *intents*, uma para cada tarefa, e foram definidos valores de *request* diferentes a fim de distinguir entre as ações para a recuperação dos dados de retorno.

3. O primeiro método passa um dado (nome) para a segunda atividade, e o segundo passa dois (nome e IMC).

4. Deve-se, primeiro, acessar o cálculo do IMC e, depois, a verificação da situação. Se for verificada a situação antes do cálculo do IMC, haverá uma exceção, já que o valor do IMC não foi preenchido no *EditText*. Isso pode ser resolvido pela inclusão de um tratamento de exceções ou, de forma mais simples, colocando o valor 0.00 como texto inicial do *EditText* do IMC.

Código da programação da segunda atividade: *CalculaIMC*.

```
package com.example.trocandomensagens;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class CalculaIMC extends AppCompatActivity {
    EditText ed1, ed2, ed3;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_calcula_imc);
        ed1 = (EditText) findViewById(R.id.editText);
        ed2 = (EditText) findViewById(R.id.editText2);
        ed3 = (EditText) findViewById(R.id.editText3);
        // recebe o nome da pessoa e coloca no Edittext do nome
        String nome = getIntent().getExtras().getString("nome");
        ed1.setText(nome);

    }
    public void calcularVoltar(View v){
        double peso, altura, imc;
        // Pega os valores do peso e altura nos EditTexts
        peso = Double.parseDouble(ed2.getText().toString());
        altura = Double.parseDouble(ed3.getText().toString());
        // Calcula o imc
        imc = peso / Math.pow(altura,2);
        // Apresenta o imc através de um Toast - pode ser retirado
        Toast.makeText(getApplicationContext(), "IMC=" + String.valueOf(imc), Toast.LENGTH_LONG).show();
        // Prepara a Intent de retorno para o valor do imc
    }
}
```

```

        Intent returnIntent = new Intent();
        // Define o parâmetro e o valor
        returnIntent.putExtra("imc",imc);
        setResult(RESULT_OK,returnIntent);
        // Encerra a atividade
        finish();
    }
}

```

Notas:

1. Recebe o parâmetro do nome no `onCreate` e coloca no `EditText`.
2. Solicita o peso e a altura, calcula o IMC e retorna o valor do IMC para a atividade principal por meio de uma *intent* de retorno.
3. Encerra a atividade e retorna à atividade principal.

Código da programação da terceira atividade: *Situacao*.

```

package com.example.trocandomensagens;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.RadioButton;

public class Situacao extends AppCompatActivity {
    EditText ed1, ed2, ed3;
    RadioButton rbFeminino, rbMasculino;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_situacao);
        ed1 = (EditText) findViewById(R.id.editText);
        ed2 = (EditText) findViewById(R.id.editText2);
        ed3 = (EditText) findViewById(R.id.editText3);
        rbFeminino = (RadioButton) findViewById(R.id.radioButton);
        rbMasculino = (RadioButton) findViewById(R.id.radioButton2);
        // Recebe os parâmetros do nome e do imc
        // Coloca nos EditTexts da View
        ed1.setText(getIntent().getExtras().getString("nome"));
    }
}

```

```

    ed2.setText(String.format("%.2f", getIntent().getExtras().getDouble("imc")));
}

public void verificarVoltar(View v) {
    String situacao = "";
    // Pega os valores e dados do imc, idade e sexo dos componentes
    da View
    double imc = Double.parseDouble(ed2.getText().toString());
    int idade = Integer.parseInt(ed3.getText().toString());
    // Determina o sexo
    int sexo = 0;
    if (rbFeminino.isChecked()) {
        sexo = 1;
    }
    if (rbMasculino.isChecked()) {
        sexo = 2;
    }
    // Verifica a situação
    if (idade == 1) { // idade 1, Sim.
        situacao = "Idade fora da faixa. Situação não determinada";
    } else { // idade 2, Não.
        if (sexo == 1) { // sexo = 1 - Feminino
            if (imc < 19.1) {
                situacao = "Abaixo do peso.";
            } else {
                if (imc < 25.8) {
                    situacao = "No peso normal.";
                } else {
                    if (imc < 27.3) {
                        situacao = "Marginalmente acima do peso.";
                    } else {
                        if (imc < 32.3) {
                            situacao = "Acima do peso.";
                        } else {
                            situacao = "Obesa.";
                        }
                    }
                }
            }
        }
    } else { // sexo = 2 - Masculino
        if (imc < 20.7) {
            situacao = "Abaixo do peso.";
        } else {
            if (imc < 26.4) {
                situacao = "No peso normal.";
            }
        }
    }
}

```

```

        } else {
            if (imc < 27.8) {
                situacao = "Marginalmente acima do peso.";
            } else {
                if (imc < 31.1) {
                    situacao = "Acima do peso.";
                } else {
                    situacao = "Obeso.";
                }
            }
        }
    }

    // Prepara a Intent de retorno para a situação
    Intent returnIntent = new Intent();
    // Define o parâmetro e o valor
    returnIntent.putExtra("situacao",situacao);
    setResult(RESULT_OK,returnIntent);
    // Encerra a atividade
    finish();
}
)

```

Notas:

1. Recebe como parâmetros o nome e o IMC e coloca nos respectivos *EditTexts*.
2. Pega os dados necessários para determinar a situação da pessoa nos componentes da view.
3. Executa a verificação e prepara o retorno da situação da pessoa para a atividade principal.
4. Encerra a atividade.

Realize os seus testes e compare com os resultados obtidos.

Testes realizados



Tela principal.



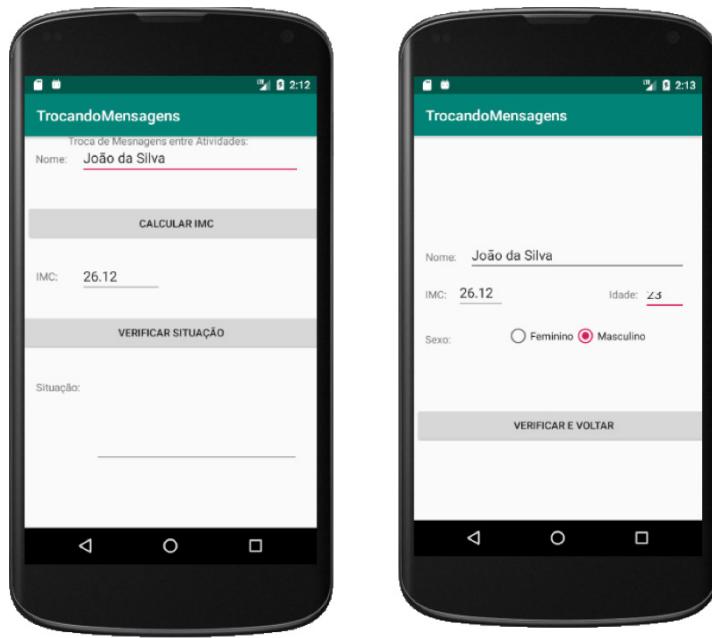
Preenchimento do nome da pessoa.



Chamada para o cálculo do IMC.



Calculando e retornando à tela principal.



Retorno à tela principal com recebimento do valor do IMC e chamada para verificar a situação.

Preenchimento da idade, definição do sexo, verificação da situação e retorno para a atividade principal.



Tela principal com os dados preenchidos.

Com esse exemplo prático, você pode trabalhar as trocas de mensagens com envio e retorno entre diferentes atividades em um mesmo projeto.



MIDIA TECA

Acesse a midiateca da Unidade 3 e assista ao vídeo com conteúdo complementar indicado pelo professor sobre como desenvolver projetos com múltiplas atividades no Android.



Saiba mais

Criação do aplicativo Twitter Searchs, com uso de *intent* e abordagens sobre o uso de novos componentes de tela que poderão enriquecer os seus projetos:

DEITEL, P. J.; DEITEL, H. M.; DEITEL, A. **Android**: como programar. Porto Alegre: Bookman, 2015. Minha Biblioteca. cap. 4, p. 99-133.

A biblioteca do Java com as APIs do Android permite que criemos aplicações móveis muito robustas e complexas, mas sem um alto grau de dificuldade. Existem muitas maneiras diferentes de realizar determinadas tarefas no desenvolvimento para o Android e não podemos ver tudo aqui, mas, com o conhecimento obtido nesta unidade, você poderá desenvolver muitos projetos.



NA PRÁTICA

Analise os aplicativos existentes em seu aparelho móvel. Você perceberá que vários deles solicitam permissões para acessar outros componentes ou funcionalidades. Como exemplo, observe que o WhatsApp solicita acesso aos seus contatos durante a instalação. Isso ocorre em razão de um arquivo chamado de manifesto do aplicativo. Dessa forma, você não precisa ter dois diferentes conjuntos de usuários, e os contatos do telefone são compartilhados com o WhatsApp. Agora é a sua vez. Procure outros aplicativos e verifique quais deles solicitam acesso a recursos de outros aplicativos.

Resumo da Unidade 3

Nesta unidade, compreendemos como ocorre a troca de mensagens entre aplicativos e atividades no Android. Vimos que podemos reaproveitar tanto outros aplicativos como atividades já desenvolvidas em novos projetos. O uso da *intent* se torna importantíssimo para aumentar nossos limites e realizar a troca de mensagens, seja entre atividades de um mesmo aplicativo, seja entre diferentes aplicativos.



CONCEITO

Uso da *intent* para realizar a troca de mensagens entre diferentes aplicativos e atividades de uma mesma aplicação.

Referências

DEITEL, P. J.; DEITEL, H. M.; DEITEL, A. **Android**: como programar. Porto Alegre: Bookman, 2015.

_____ ; _____; WALD, A. **Android 6 para programadores**: uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2016.

INTENTS e filtros de intents. **Android Developers**, [s. l.], 2018. Disponível em: <https://developer.android.com/guide/components/intents-filters?hl=pt-br>. Acesso em: 15 jun. 2019.

MONK, S. **Projetos com Arduino e Android**: use seu smartphone ou tablet para controlar o Arduino. Porto Alegre: Bookman, 2014.

UNIDADE 4

Persistência de dados no Android

INTRODUÇÃO

O desenvolvimento de sistemas de computação requer, na maioria das situações, que dados sejam armazenados e recuperados. Essa ação de armazenamento e recuperação de dados é chamada de **persistência de dados**. A persistência de dados pode ocorrer com o armazenamento de poucos dados para provimento de configurações do sistema ou com o armazenamento de grandes quantidades de dados, quando trabalhamos com tabelas em bancos de dados. Conhecer formas diferentes de armazenamento e recuperação de dados permite que tenhamos sistemas mais robustos e confiáveis, inclusive sendo possível o uso de mais de uma forma de persistência no desenvolvimento de um mesmo projeto.



OBJETIVO

Nesta unidade, você será capaz de:

- Construir aplicativos para o Android com uso de diferentes recursos de persistência de dados.

Trabalhando com a persistência de dados em arquivos de preferências

O armazenamento de informações de status ou configurações de sistemas não requer grande armazenamento de dados. Normalmente esses dados são armazenados em pequenos arquivos locais e gravados pela aplicação. Ao se iniciar a aplicação, os arquivos são carregados.

É comum armazenarmos dados da empresa e caminhos para recursos de uso do sistema.

No Android, temos um recurso muito útil para o armazenamento de dados de status e configuração dos aplicativos: o *SharedPreferences*.

SharedPreferences — em português, “**preferências compartilhadas**” — é uma forma bastante comum de armazenamento de configurações simples. Podemos armazenar dados do usuário e de configurações do aplicativo, bem como outros dados que sejam informações importantes para uma boa execução da aplicação. As informações armazenadas por meio das preferências compartilhadas estão disponíveis para uso de uma aplicação de forma exclusiva ou podem ser compartilhadas para uso de outras aplicações.



Exemplo

Você pode identificar o local de armazenamento de arquivos ou fotos de um outro aplicativo consultando suas preferências compartilhadas.

O arquivo de preferências compartilhadas utiliza uma forma de armazenamento muito comum e simples do tipo **identificador/valor**. Cada dado armazenado no arquivo deve ter um identificador para permitir o armazenamento e a recuperação individual de cada dado, além do valor a ser armazenado.

Por meio do par identificador/valor, podemos armazenar diferentes informações do aplicativo, com diferentes tipos de dados em função do método de armazenamento utilizado.

O armazenamento do par identificador/valor varia de acordo com o tipo de dado e, por isso, existem diferentes tipos de métodos para a realização do armazenamento.

SharedPreferences.Editor

Essa classe é responsável por trabalhar os métodos de **organização**, de **armazenamento** e de **recuperação de dados** por meio das preferências compartilhadas.

Os modos de acesso podem ser:

a. PRIVATE

Não disponibiliza os dados armazenados para outros aplicativos.

b. MODE_APPEND

Para uso com criação de arquivos utilizando as APIs Java:

- Caso o arquivo exista, o dado será incluído ao final.
- Caso contrário, o arquivo original será apagado.

c. MODE_ENABLE_WRITE_AHEAD_LOGGING

Para uso com gerenciadores de bancos de dados.



Observação

Existem outros modos de acesso, mas estes se encontram em desuso.

Métodos

a. Principais métodos de controle

| Método | Uso |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| apply() | <ul style="list-style-type: none">• Efetua o armazenamento das preferências no arquivo.• É a forma mais nova e não usa a <i>thread</i> principal para efetivar as alterações.• É do tipo <i>void</i>, não retornando a informação de que os dados foram realmente alterados. |

| | |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| clear() | <ul style="list-style-type: none"> Remove todas as preferências, limpando o arquivo de preferências. |
| commit() | <ul style="list-style-type: none"> Efetua o armazenamento das preferências no arquivo. É a forma mais antiga e usa a <i>thread</i> principal para efetivar as alterações. É do tipo <i>boolean</i>, retornando verdadeiro (<i>true</i>) se os dados foram realmente alterados, ou falso (<i>false</i>), caso contrário. |
| contains (String key) | <ul style="list-style-type: none"> Verifica se um identificador (<i>String Key</i>) existe no arquivo de preferências. |
| Remove (String key) | <ul style="list-style-type: none"> Remove uma preferência identificada pelo identificador (<i>String Key</i>). |

b. Como visto com a passagem de dados entre atividades, os métodos de armazenamento são do tipo *put*:

| Método | Uso |
|------------------------------------------|-------------------------------------------------------------------------|
| putBoolean(String key, boolean value) | Determina um valor lógico (<i>boolean</i>) para um identificador. |
| putFloat(String key, float value) | Determina um valor real (<i>float</i>) para um identificador. |
| putInt(String key, int value) | Determina um valor inteiro (<i>int</i>) para um identificador. |
| putLong(String key, long value) | Determina um valor inteiro longo (<i>long</i>) para um identificador. |
| putString(String key, String value) | Determina um valor de texto (<i>String</i>) para um identificador. |

c. Como visto com a passagem de dados entre atividades, os métodos de recuperação são do tipo get:

| Método | Uso |
|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| getBoolean(String key, boolean defVal) | <ul style="list-style-type: none">Retorna um valor lógico (<i>boolean</i>) por meio do identificador (<i>key</i>).Usará o valor padrão (<i>default / defVal</i>), caso não encontre o identificador. |
| getFloat(String key, float defVal) | <ul style="list-style-type: none">Retorna um valor real (<i>float</i>) por meio do identificador (<i>key</i>).Usará o valor padrão (<i>default / defVal</i>), caso não encontre o identificador. |
| getInt(String key, int defVal) | <ul style="list-style-type: none">Retorna um valor inteiro (<i>int</i>) por meio do identificador (<i>key</i>).Usará o valor padrão (<i>default / defVal</i>), caso não encontre o identificador. |
| getLong(String key, long defVal) | <ul style="list-style-type: none">Retorna um valor inteiro longo (<i>long</i>) por meio do identificador (<i>key</i>).Usará o valor padrão (<i>default / defVal</i>), caso não encontre o identificador. |
| getString(String key, String defVal) | <ul style="list-style-type: none">Retorna um valor textual (<i>String</i>) por meio do identificador (<i>key</i>).Usará o valor padrão (<i>default / defVal</i>), caso não encontre o identificador. |

Vamos trabalhar com o armazenamento e a recuperação dos dados de um usuário por meio de um exemplo prático?

O aplicativo irá solicitar alguns dados de uma pessoa e, ao clicar no botão **SALVAR**, esses dados serão armazenados em um arquivo de configuração do aplicativo, chamado **dadosUsuario**.

O aplicativo, ao iniciar por meio do método **onCreate**, irá verificar a existência dos dados armazenados:

- Caso existam os dados, o aplicativo preencherá os dados na tela do aplicativo.
- Caso não existam, aguardará que o usuário preencha os dados e salve-os.

A aplicação sempre tentará preencher os dados do usuário.

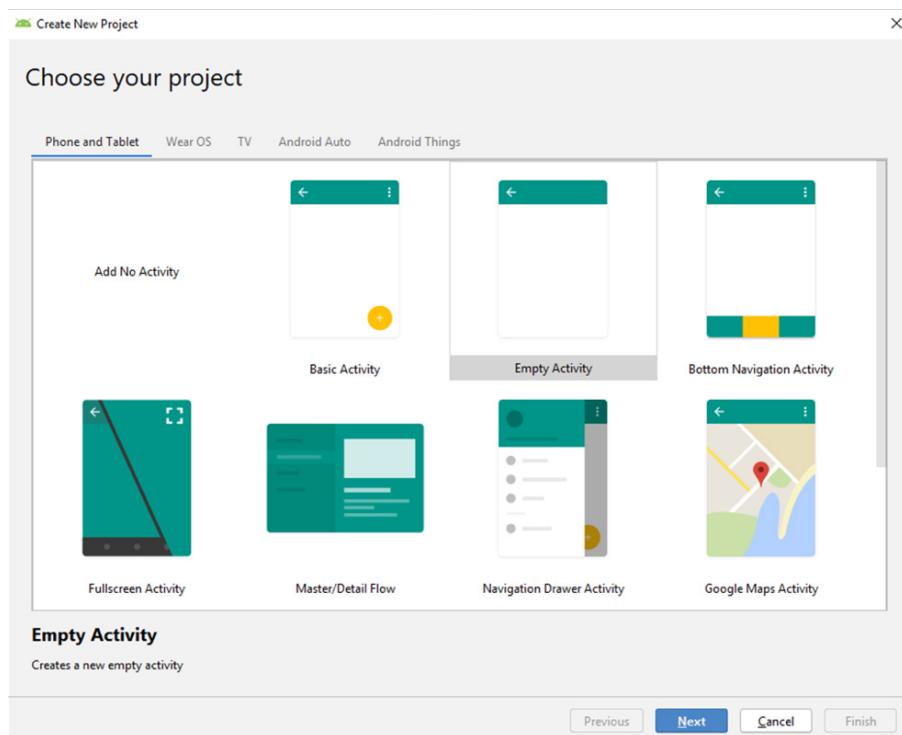
Para facilitar o entendimento, o aplicativo terá um **botão de limpeza** para limpar os dados do usuário – tanto na tela quanto no arquivo de preferências.

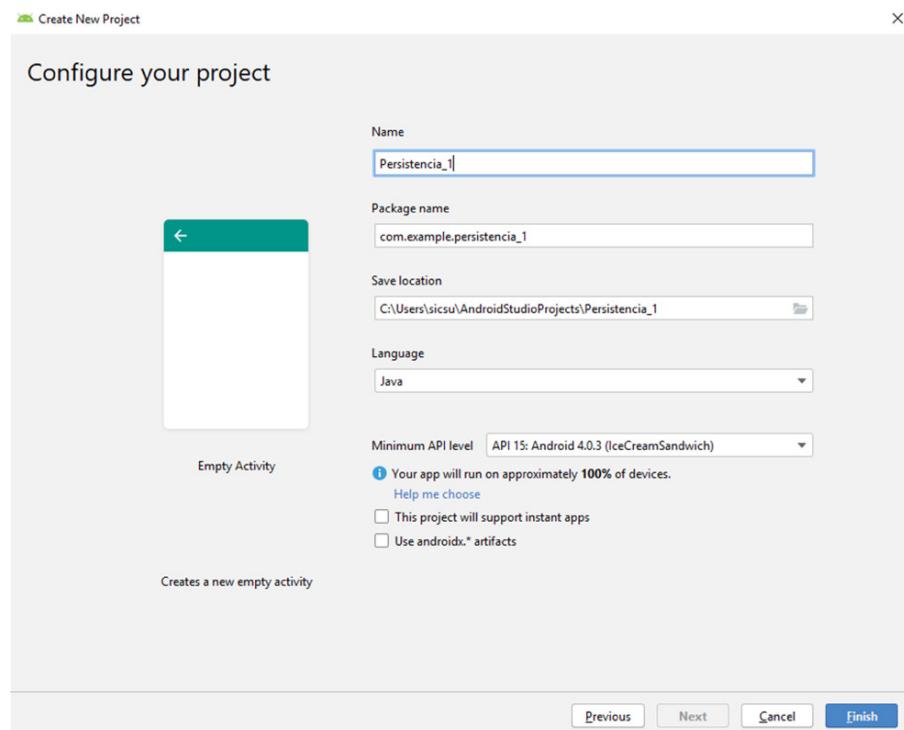
Por meio das diretrizes a seguir, será criado um aplicativo para a chamada de diversas outras ações que estão relacionadas a outras aplicações dentro de um sistema Android.

Vamos praticar?

- I. Crie um novo projeto compatível com a API 15 e codificação em Java.
- II. O projeto terá o nome de **Persistencia_1**.
- III. Caso você altere os nomes da sua atividade, certifique-se de que fará o mesmo com a classe da atividade e com a chamada da view.

Criação do projeto **Persistência_1**

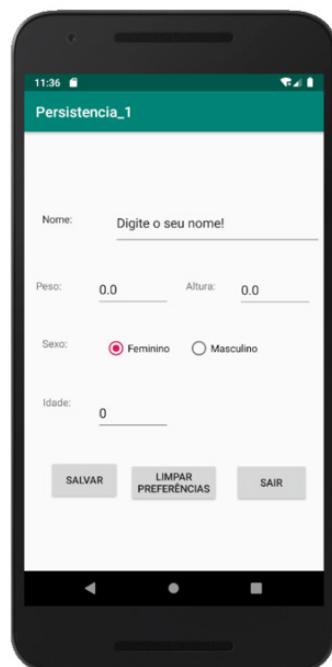




Android Studio

a. Criação da tela – View

Layout e codificação da view do projeto:



```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <android.support.constraint.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        tools:context=".MainActivity">

        <EditText
            android:id="@+id/editText4"
            android:layout_width="101dp"
            android:layout_height="49dp"
            android:layout_marginStart="99dp"
            android:layout_marginLeft="99dp"
            android:layout_marginTop="426dp"
            android:layout_marginEnd="211dp"
            android:layout_marginRight="211dp"
            android:layout_marginBottom="256dp"
            android:ems="10"
            android:inputType="number"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/textView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="16dp"
            android:layout_marginLeft="16dp"
            android:layout_marginTop="174dp"
            android:layout_marginEnd="336dp"
            android:layout_marginRight="336dp"
            android:layout_marginBottom="538dp"
            android:text="Nome:"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

```

```

<EditText
    android:id="@+id/editText2"
    android:layout_width="101dp"
    android:layout_height="49dp"
    android:layout_marginStart="99dp"
    android:layout_marginLeft="99dp"
    android:layout_marginTop="256dp"
    android:layout_marginEnd="211dp"
    android:layout_marginRight="211dp"
    android:layout_marginBottom="426dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<RadioGroup
    android:layout_width="235dp"
    android:layout_height="48dp"
    android:layout_marginStart="110dp"
    android:layout_marginLeft="110dp"
    android:layout_marginTop="345dp"
    android:layout_marginEnd="66dp"
    android:layout_marginRight="66dp"
    android:layout_marginBottom="338dp"
    android:orientation="horizontal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <RadioButton
        android:id="@+id radioButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:checked="true"
        android:text="Feminino" />

    <RadioButton
        android:id="@+id radioButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Masculino" />
</RadioGroup>
```

```

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="266dp"
    android:layout_marginEnd="360dp"
    android:layout_marginRight="360dp"
    android:layout_marginBottom="446dp"
    android:text="Peso:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="222dp"
    android:layout_marginLeft="222dp"
    android:layout_marginTop="266dp"
    android:layout_marginEnd="148dp"
    android:layout_marginRight="148dp"
    android:layout_marginBottom="446dp"
    android:text="Altura:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="25dp"
    android:layout_marginLeft="25dp"
    android:layout_marginTop="345dp"
    android:layout_marginEnd="352dp"
    android:layout_marginRight="352dp"
    android:layout_marginBottom="367dp"
    android:text="Sexo:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<TextView
    android:id="@+id/textView6"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="26dp"
    android:layout_marginLeft="26dp"
    android:layout_marginTop="426dp"
    android:layout_marginEnd="347dp"
    android:layout_marginRight="347dp"
    android:layout_marginBottom="286dp"
    android:text="Idade:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button"
    android:layout_width="96dp"
    android:layout_height="55dp"
    android:layout_marginStart="39dp"
    android:layout_marginLeft="39dp"
    android:layout_marginTop="518dp"
    android:layout_marginEnd="284dp"
    android:layout_marginRight="284dp"
    android:layout_marginBottom="165dp"
    android:onClick="salvar"
    android:text="Salvar"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button2"
    android:layout_width="101dp"
    android:layout_height="55dp"
    android:layout_marginStart="289dp"
    android:layout_marginLeft="289dp"
    android:layout_marginTop="518dp"
    android:layout_marginEnd="21dp"
    android:layout_marginRight="21dp"
    android:layout_marginBottom="158dp"
    android:onClick="sair"
    android:text="Sair"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"

```

```

        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="144dp"
    android:layout_marginLeft="144dp"
    android:layout_marginTop="518dp"
    android:layout_marginEnd="144dp"
    android:layout_marginRight="144dp"
    android:layout_marginBottom="158dp"
    android:onClick="limpar"
    android:text="Limpar\nPreferências"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginLeft="16dp"
    android:layout_marginTop="174dp"
    android:layout_marginEnd="336dp"
    android:layout_marginRight="336dp"
    android:layout_marginBottom="538dp"
    android:text="Nome:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText"
    android:layout_width="285dp"
    android:layout_height="62dp"
    android:layout_marginStart="110dp"
    android:layout_marginLeft="110dp"
    android:layout_marginTop="158dp"
    android:layout_marginEnd="1dp"
    android:layout_marginRight="1dp"

```

```

        android:layout_marginBottom="511dp"
        android:ems="10"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.873"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText3"
    android:layout_width="101dp"
    android:layout_height="49dp"
    android:layout_marginStart="294dp"
    android:layout_marginLeft="294dp"
    android:layout_marginTop="257dp"
    android:layout_marginEnd="16dp"
    android:layout_marginRight="16dp"
    android:layout_marginBottom="425dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

```

Notas:

a.1. Foram criados três botões:

- Um botão para a gravação do arquivo de preferências (**SALVAR**).
- Um botão para limpar a tela e o arquivo de preferências (**LIMPAR PREFERÊNCIAS**).
- Um botão para encerrar o aplicativo (**SAIR**).

a.2. Foi utilizado o exemplo com os dados do IMC por eles serem heterogêneos, permitindo o armazenamento e a recuperação de diferentes tipos de dados ao mesmo tempo.

b. Codificação

Código de programação da aplicação (*controller*):

```
package com.example.persistencia_1;

import android.content.SharedPreferences;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    // Objetos relacionados aos componentes da tela (view)
    EditText edNome, edPeso, edAltura, edIdade;
    RadioButton rdFeminino, rdMasculino;
    // Declaração do objeto para o armazenamento das preferências
    SharedPreferences prefs;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        edNome = (EditText) findViewById(R.id.editText);
        edPeso = (EditText) findViewById(R.id.editText2);
        edAltura = (EditText) findViewById(R.id.editText3);
        edIdade = (EditText) findViewById(R.id.editText4);
        rdFeminino = (RadioButton) findViewById(R.id.radioButton);
        rdMasculino = (RadioButton) findViewById(R.id.radioButton2);
        // Definição do arquivo de preferências com o nome do arquivo
        // e o modo de compartilhamento de dados privado
        prefs = getSharedPreferences("dadosUsuario", MODE_PRIVATE);
        // Método para preencher a tela com os dados do arquivo de
        // preferências ou com valores-padrão
        iniciarDadosUsuario();
    }

    private void iniciarDadosUsuario(){
        // Verifica se o dado com o nome está disponível no arquivo
        // de preferências e preenche o componente; caso contrário,
        // preenche o componente com um texto padrão
        if(prefs.contains("nome")) {
            edNome.setText(prefs.getString("nome", ""));
        }
    }
}
```

```

else{
    edNome.setText("Digite o seu nome!");
}

// Verifica se o dado com a idade está disponível no arquivo
// de preferências e preenche o componente; caso contrário,
// preenche o componente com valor 0 (zero)
if(prefs.contains("idade")) {
    edIdade.setText(String.valueOf(prefs.getInt("idade", 0)));
}
else{
    edIdade.setText("0");
}

// Verifica se o dado com o peso está disponível no arquivo
// de preferências e preenche o componente; caso contrário,
// preenche o componente com valor 0.0 (zero.zero)
if(prefs.contains("peso")) {
    edPeso.setText(String.valueOf(prefs.getFloat("peso", 0.0f)));
}
else{
    edPeso.setText("0.0");
}

// Verifica se o dado com a altura está disponível no arquivo
// de preferências e preenche o componente; caso contrário,
// preenche o componente com valor 0.0 (zero.zero)
if(prefs.contains("altura")){
    edAltura.setText(String.valueOf(prefs.getFloat("altura",
0.0f)));
}
else{
    edAltura.setText("0.0");
}

// Verifica se o dado com o sexo está disponível no arquivo
// de preferências e seleciona o componente radioButton adequado;
// caso contrário, seleciona o componente padrão (feminino)
if(prefs.contains("sexo")) {
    if (prefs.getBoolean("sexo", true)) {
        // Caso verdadeiro (true), seleciona o radioButton do sexo
        feminino
        rdFeminino.setChecked(true);
    } else {
        // Caso false (falso), seleciona o radioButton do sexo
        masculino
        rdMasculino.setChecked(true);
    }
}

```

```

    else {
        rdFeminino.setChecked(true);
    }
}

public void salvar(View v){
    // Abre o arquivo de preferências para edição
    Sharedpreferences.Editor prefUsuario = prefs.edit();
    // Determina os pares identificador/valor
    // de cada dado do arquivo de preferências
    // para texto (String)
    prefUsuario.putString("nome", edNome.getText().toString());
    // Para inteiro (int)
    prefUsuario.putInt("idade", Integer.parseInt(edIdade.getText().
    toString()));
    // Para real (float)
    prefUsuario.putFloat("peso", Float.parseFloat(edPeso.getText().
    toString()));
    prefUsuario.putFloat("altura", Float.parseFloat(edAltura.get-
    Text().toString()));
    // Para booleano (boolean)
    if(rdFeminino.isChecked()){
        prefUsuario.putBoolean("sexo", true);
    }
    else{
        prefUsuario.putBoolean("sexo", false);
    }
    // Confirma a alteração
    prefUsuario.commit();
    // Mensagem de aviso ao usuário
    Toast.makeText(getApplicationContext(),
        "Suas preferências foram salvas!\nAguarde alguns instantes para sair.",
        Toast.LENGTH_LONG).show();
}

public void limpar(View v){
    // Abre o arquivo de preferências para edição
    Sharedpreferences.Editor prefUsuario = prefs.edit();
    // Limpa o arquivo de preferências
    prefUsuario.clear();
    // Confirma a alteração
    prefUsuario.apply();
    // Prepara a tela novamente, sem os dados de preferência
    iniciarDadosUsuario();
    // Mensagem de aviso ao usuário
    Toast.makeText(getApplicationContext(),

```

```

    "Suas preferências foram apagadas!\nAguarde alguns instantes para sair.",
    Toast.LENGTH_LONG).show();
}

private void sair(View v){
    // Encerra o aplicativo
    finish();
}

}

```

Notas:

- Não se esqueça de confirmar a programação dos métodos **OnClick** dos três botões.
- O aplicativo, ao iniciar, verifica se existe o arquivo de preferências (**dadosUsuario**); caso os dados existam, são carregados para a tela do aplicativo; caso contrário, a tela é preenchida com valores-padrão.
- Qualquer alteração em algum dos dados do usuário pode ser feita com o botão de **SALVAR**.
- O botão de **LIMPAR** volta a tela ao padrão original e apaga os dados de preferência do arquivo.
- Ao sair do aplicativo (**SAIR**), é possível que as alterações ainda não tenham sido realizadas, e uma mensagem do Android sobre o aplicativo poderá ser exibida, mas os dados não serão perdidos.

Realize os seus testes e compare com os resultados obtidos!

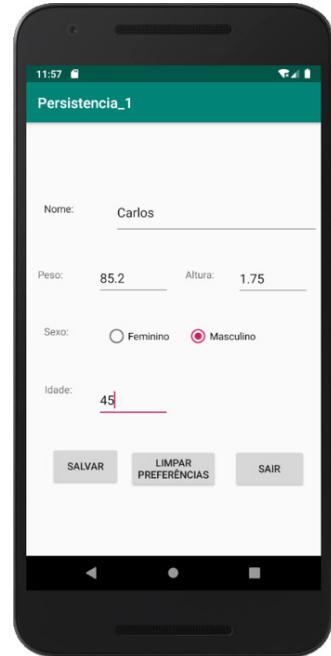
Testes realizados



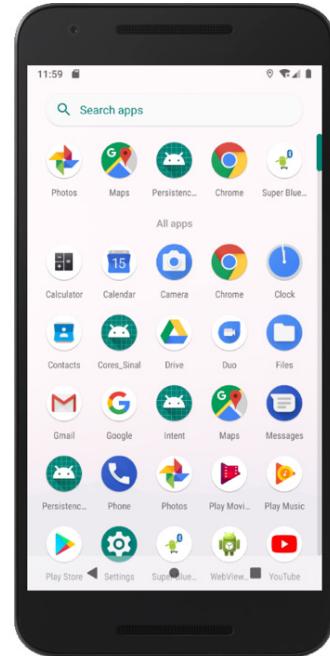
Tela principal sem dados.



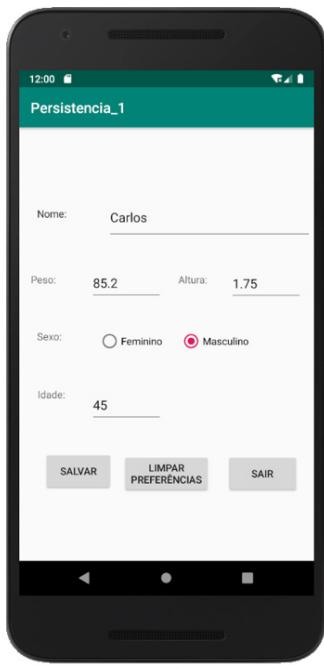
Preenchimento e gravação com botão SALVAR.



Após a gravação, sair do aplicativo.



Retorno ao Android para executar novamente.



O aplicativo inicia com dados já preenchidos.



Limpeza do arquivo de preferências e sair do aplicativo.



Retorno ao Android para executar novamente.



O aplicativo retorna na forma original, sem as preferências.

Você pode trabalhar com diferentes arquivos de preferências para um mesmo aplicativo, basta definir nomes diferentes para cada arquivo. É possível, inclusive, criar diferentes arquivos para diferentes usuários, bastando realizar a mudança de usuário, alternando os arquivos de preferências.



Dica

A atualização do arquivo de preferências pode demorar um pouco para ser efetivada. Isso ocorre porque esse processo é executado de forma assíncrona (não sincronizada) com o aplicativo. Como na maioria dos aplicativos a gravação das preferências ocorre durante o seu uso, isso não é notado pelo usuário. Sendo assim, é necessário aguardar um pouco para sair; caso contrário, uma mensagem do aplicativo pode ser exibida.

De qualquer forma, não se preocupe!

A tarefa será realizada mesmo após o encerramento da aplicação.



MEDIATECA

Veja na midiateca da Unidade 4 o conteúdo complementar selecionado pelo professor sobre **como usar SharedPreferences no Android**.



MEDIATECA

Veja na midiateca da Unidade 4 o conteúdo complementar selecionado pelo professor sobre o **desenvolvimento do aplicativo Twitter Searches**.

Trabalhando com a persistência de dados em arquivos locais

É muito comum que tenhamos aplicativos que:

- Precisam armazenar um conjunto pequeno de dados.
- São inadequados para um armazenamento em arquivos de preferências.
- Necessitam de um banco de dados.

Como temos a linguagem Java completa disponível para o desenvolvimento com o Android, podemos criar arquivos de uso comum em Java para uso em nossos aplicativos.

A linguagem Java disponibiliza um conjunto de classes para trabalhar com diferentes tipos de arquivos.

Streams

Streams são objetos utilizados para manipular dados em arquivos. Para manipular streams, devemos conhecer as três classes a seguir:

- i. *File*.
- ii. *FileInputStream*.
- iii. *FileOutputStream*.

i. *File*

Representação lógica de um arquivo cujos métodos construtores são:

- *File(String arquivo)* – Nome do arquivo para uso no mesmo diretório da aplicação.
- *File(String path, String arquivo)* – Caminho e nome do arquivo para locais diferentes.

Seus principais métodos são:

canRead()

Verifica e retorna verdadeiro (*true*) caso o arquivo possua permissão de leitura.

| | |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| canWrite() | Verifica e retorna verdadeiro (<i>true</i>) caso o arquivo possua permissão de gravação. |
| delete() | Apaga o arquivo. |
| exists() | Verifica e retorna verdadeiro (<i>true</i>) caso o arquivo exista no diretório. |
| getName() | Retorna uma <i>String</i> contendo apenas o nome do arquivo. |
| getPath() | Retorna uma <i>String</i> contendo o caminho (<i>path</i>) do arquivo. |
| isFile() | Verifica e retorna verdadeiro (<i>true</i>) se o arquivo é um arquivo "normal" (não é um diretório, por exemplo). |
| length() | Verifica e retorna um valor do tipo: • " <i>long</i> ", contendo o tamanho do arquivo em bytes. • <i>0L</i> , se o arquivo não existir. |
| lastModified() | Verifica e retorna um valor do tipo " <i>long</i> " contendo o tempo em que houve a última modificação do arquivo. |
| list() | Retorna um vetor (<i>array</i>) de <i>Strings</i> com os nomes de todos os arquivos do diretório. |
| list(<i>FileNameFilter filtro</i>) | Retorna um vetor (<i>array</i>) de <i>Strings</i> com os nomes de todos os arquivos do diretório e que pertençam ao "filtro". |
| mkdirs() | Cria um novo diretório no final do caminho especificado pelo objeto <i>File</i> . |
| renameTo(<i>File novo_nome</i>) | Renomeia o arquivo para "novo-nome". |



Observação

Como esses métodos podem lançar exceções de *SecurityException*, eles devem ser utilizados dentro de um bloco *try/catch*.

ii. *FileInputStream*

Representação de uma *stream* para leitura de dados de um arquivo.

Seus principais métodos são:

| | |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| close() | Fecha o arquivo, liberando todos os recursos associados. |
| read(byte b[]) | Faz uma leitura no arquivo com o total de bytes especificados por <i>b.length</i> , retornando quantos bytes foram lidos, ou <i>-1</i> se o fim de arquivo foi alcançado. |



Observação

Como esses métodos podem lançar exceções de *IOException*, eles devem ser utilizados dentro de um bloco *try/catch*.

iii. *FileOutputStream*

Representação de uma *stream* para escrita de dados em um arquivo.

Seus principais métodos são:

| | |
|--------------------------|--------------------------------------------------------------------------------------|
| close() | Fecha o arquivo, liberando todos os recursos associados. |
| write(byte b[]) | Faz uma gravação no arquivo com o total de bytes especificados por <i>b.length</i> . |



Observação

Como esses métodos podem lançar exceções de *IOException*, eles devem ser utilizados dentro de um bloco *try/catch*.

Utilizando streams

Para uso de *streams*, é necessário:

1. Importar as classes de "java.io":

```
import java.io.*;
```

2. Associar um arquivo em disco (por meio de uma *String* contendo o seu nome) a um dispositivo lógico (uma variável do tipo "*File*"):

```
File f = new File (nome_arq);
```

3. Instanciar objetos que permitam operar (ou ler ou escrever) o arquivo.

Para abrir o arquivo para **leitura**:

- FileInputStream fis = new FileInputStream(f);

Para abrir o arquivo para **escrita**:

- FileOutputStream fos = new FileOutputStream(f);

Você pode utilizar *streams* para realizar cópias de arquivos para:

- Realização de backups.
- Armazenamento de textos.
- Organização de conjuntos de dados.

Para trabalhar com o armazenamento e recuperação de dados, podemos usar as classes: **DataOutputStream** e **DataInputStream**.

Os seus principais métodos são:

| DataOutputStream | DataInputStream |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| writeBoolean (boolean b) writeByte (byte b) writeChar (nt caracter) writeDouble (double d) writeFloat(float f) writeInt(int i) writeLong(long e) writeShort(short s) writeUTF(String str) | readBoolean() readByte() readChar() readDouble() readFloat() readInt() readLong() readShort() readUTF() |

Agora, em mais um exemplo prático, trabalharemos com um conjunto de dados para o armazenamento de um conjunto de disciplinas e suas respectivas notas de A1, A2 e A3.

Vamos aproveitar a oportunidade para explorar o uso da linguagem Java no desenvolvimento de projetos para o Android!

Nosso aplicativo fará acesso a um arquivo local e, para isso, será necessário requisitar o acesso à leitura e à gravação do arquivo por meio da liberação do recurso no *AndroidManifest*.

Vamos criar uma classe Java para definir a disciplina com os seguintes atributos:

- Nome (texto).
- A1 (real).
- A2 (real).
- A3 (real).

Além disso, serão criados:

- Os métodos de acesso (setters e getters).
- Um construtor para inicialização dos atributos com valores-padrão.
- Um método para formatar os itens da lista de disciplinas.

A aplicação:

- Apresentará ao usuário uma lista de disciplinas com os valores-padrão – e o usuário poderá selecionar qualquer item da lista para alterar os valores-padrão pelos valores de seu interesse.

- Carregará os dados do arquivo toda vez que inicializar – isso fará com que na primeira vez seja apresentada uma exceção pela ausência do arquivo, mas, após a primeira gravação, essa mensagem não será mais apresentada.

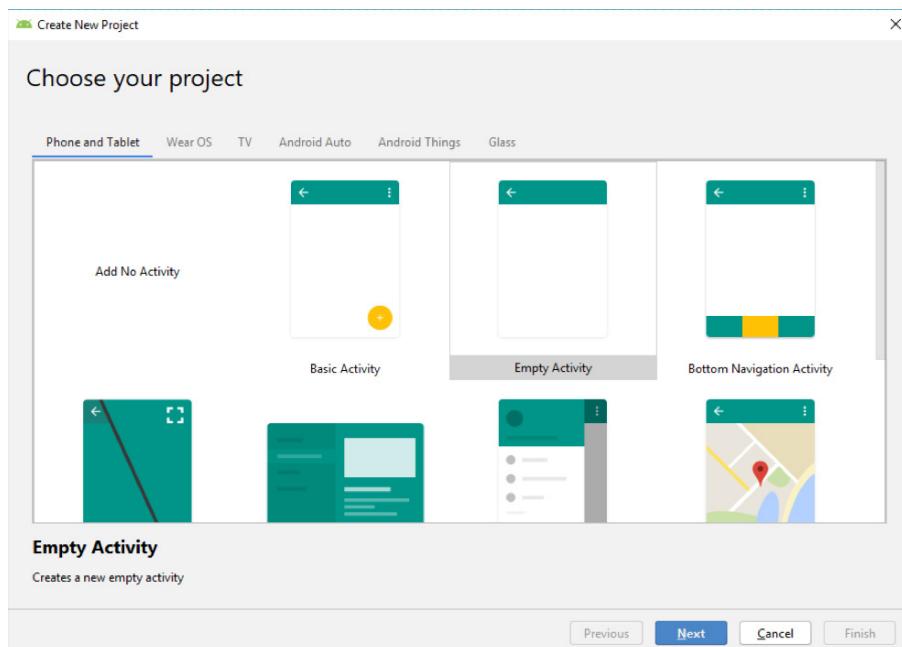
Os dados serão armazenados e lidos a partir de um arquivo do tipo *DataStream* (input/output).

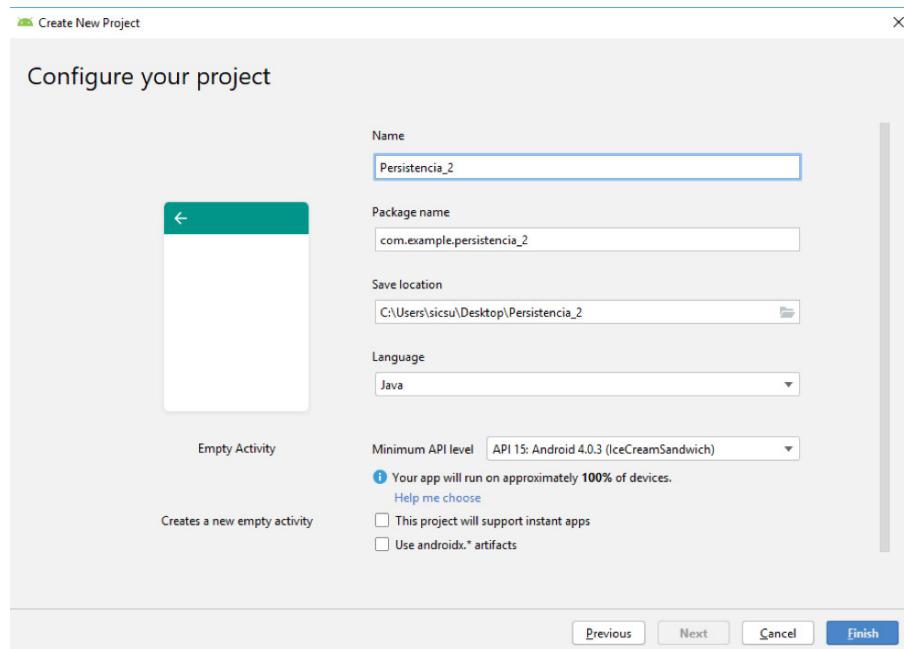
Iremos usar também o conceito de troca de mensagens entre a **atividade principal** e a **atividade de tratar disciplina**.

Vamos praticar?

- I. Crie um novo projeto compatível com a API 15 e codificação em Java.
- II. O projeto terá o nome de **Persistencia_2**.
- III. Caso você altere os nomes da sua atividade, certifique-se de que fará o mesmo com a classe da atividade e com a chamada da view.

Criação do projeto **Persistência_2**





Android Studio

Importante

O *AndroidManifest* é responsável pela configuração geral do aplicativo e definição dos recursos necessários.

Não podemos simplesmente utilizar recursos que possam afetar a segurança do sistema sem requerer permissão para isso.

As solicitações de uso de recursos devem ser definidas na seção `<uses-permission>` do *AndroidManifest*.

O arquivo *AndroidManifest.xml* está disponível na aba “manifests” do app.

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.persistencia_2">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Persistencia_2"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".TratarDisciplina" />
    </application>
    <uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE" />
</manifest>

```

Localização e edição do arquivo *AndroidManifest.xml*

Devemos incluir após: </application>

Permissão: <uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE"/>

Antes de: </manifest>



Dica

Existem as permissões de leitura (**READ**) e de gravação (**WRITE**), mas a permissão de gravação já inclui a permissão de leitura, não sendo necessário incluir as duas.

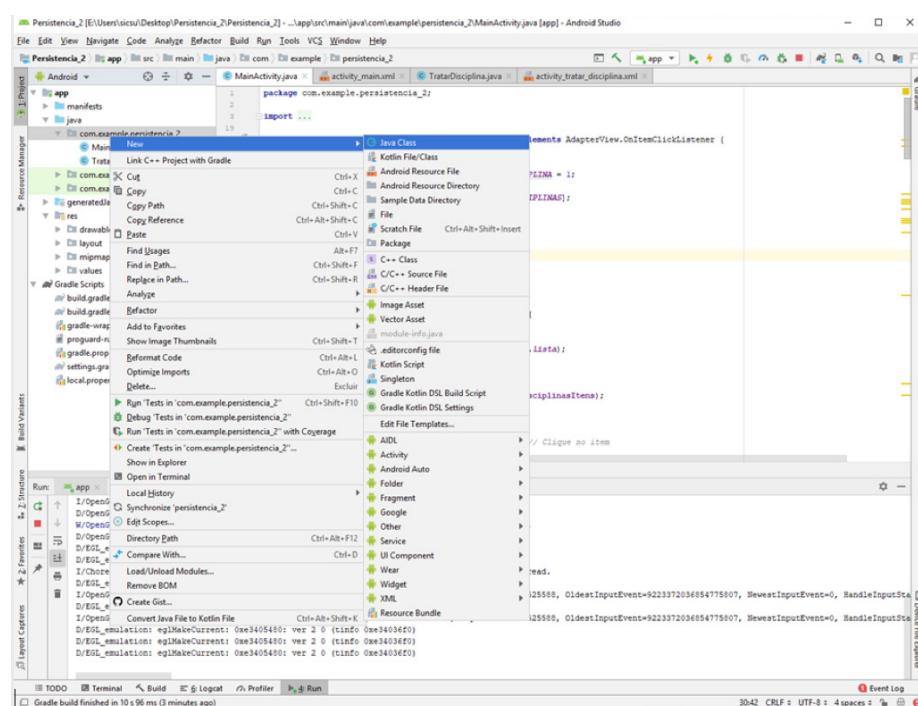
Atividade principal

a. Classe Disciplina

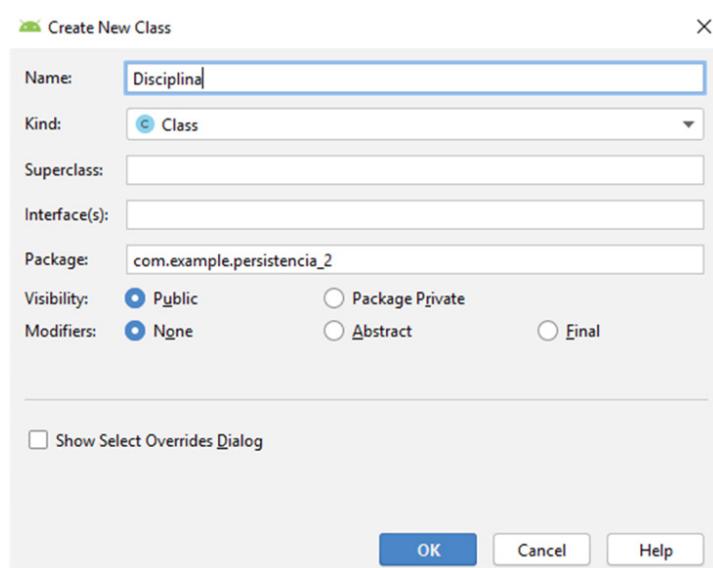
Devemos incluir uma classe Java em nosso projeto.

A **classe Disciplina** será usada pela **atividade principal**, por isso deve ser criada antes de codificarmos a atividade principal.

Observe como incluir uma nova classe Java:



1. Selecione a aplicação e escolha **New > Java Class**.



2. Defina o nome da classe, ***Disciplina***, e confirme com **OK**.

b. Codificação da nova classe Java: ***Disciplina***

```
package com.example.persistencia_2;

public class Disciplina {
    // Definição dos atributos
    private String nome;
    private double a1, a2, a3;

    // Métodos de acesso (setters & getters)
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        if (!nome.isEmpty()) {
            this.nome = nome;
        }
    }

    public double getA1() {
        return a1;
    }

    public void setA1(double a1) {
        if (a1 >= 0) {
            this.a1 = a1;
        }
    }

    public double getA2() {
        return a2;
    }

    public void setA2(double a2) {
        if (a2 >= 0) {
            this.a2 = a2;
        }
    }

    public double getA3() {
        return a3;
    }

    public void setA3(double a3) {
        if (a3 >= 0) {
```

```

        this.a3 = a3;
    }

}

// Método construtor com definição dos valores-padrão
public Disciplina() {
    nome = "Nome Disciplina";
    a1 = 0.0;
    a2 = 0.0;
    a3 = 0.0;
}

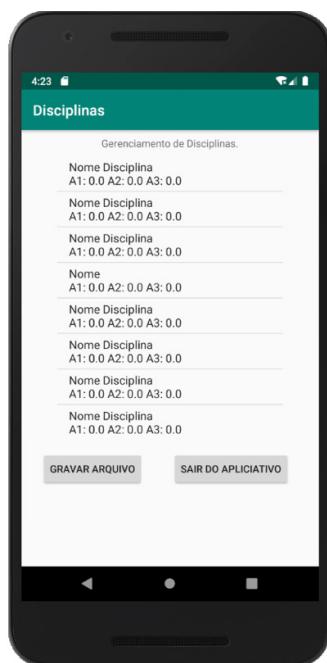
// Método de formatação dos dados para exibição na lista
public String textoLista() {
    String item;
    item = getNome();
    item += "\nA1: " + String.format("%3.1f", getA1());
    item += "\tA2: " + String.format("%3.1f", getA2());
    item += "\tA3: " + String.format("%3.1f", getA3());
    return item;
}
}

```

c. MainActivity

Criação da tela (view).

Layout e codificação da view do projeto:



```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/lista"
        android:layout_width="312dp"
        android:layout_height="388dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="36dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"></ListView>

    <Button
        android:id="@+id/btSair"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="20dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:onClick="sair"
        android:text="Sair do Aplicativo"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/btIncluir"
        app:layout_constraintTop_toBottomOf="@+id/lista" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginBottom="8dp"
        android:text="Gerenciamento de Disciplinas."
        app:layout_constraintBottom_toTopOf="@+id/lista"
        app:layout_constraintEnd_toEndOf="parent"

```

```

        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.457" />

<Button
    android:id="@+id/btIncluir"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="20dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:onClick="gravarArquivo"
    android:text="Gravar Arquivo"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.078"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/lista" />

</android.support.constraint.ConstraintLayout>

```

Notas:

c.1. Foram criados dois botões:

- Um botão para a gravação do arquivo de dados (**GRAVAR ARQUIVO**).
- Um botão para encerrar o aplicativo (**SAIR DO APLICATIVO**).

c.2. Foi utilizada UMA LISTA (**ListView**) em que cada item pode ser clicado de forma a chamar a atividade de **TratarDisciplina** para realizar as alterações dos dados, de acordo com o índice do elemento na lista.

d. Codificação

Código de programação da aplicação (controller):

```

package com.example.persistencia_2;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;

```

```

import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity implements AdapterView.OnItemClickListener {
    // Componente lista da view
    ListView listaDisciplinas;
    // ArrayAdapter para realizar o preenchimento da lista com os itens
    // do vetor (array)
    ArrayAdapter adapter;
    // Determina o código da intent
    public static final int ACTIVITY_REQUEST_DISCIPLINA = 1;
    // Determina o número total de itens (disciplinas)
    public static final int TOTAL_DISCIPLINAS = 50;
    // Determina o vetor de textos para os itens da lista
    public String[] disciplinas = new String[TOTAL_DISCIPLINAS];
    // Determina o vetor de objetos Disciplina
    public Disciplina[] vetDisciplinas = new Disciplina[TOTAL_DISCIPLINAS];
    // Declaração da intent
    Intent intent;
    // Armazena o índice do item da lista
    public int indiceItem = 0;
    // Define o nome do arquivo de dados
    private final String NOMEARQUIVO = "dadosDisciplinas.dat";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listaDisciplinas = (ListView) findViewById(R.id.lista);

        // Instancia os objetos Disciplina para poderem
        // ser usados na carga dos dados
        iniciarVetor();
        // Carrega o arquivo e prepara os itens da lista
        try {
            if (carregarDisciplinasArquivo(NOMEARQUIVO)) {

```

```

        // Mensagem de informação de sucesso
        Toast.makeText(getApplicationContext(),
        "Dados carregados na lista!", Toast.LENGTH_LONG).show();
    }
} catch (FileNotFoundException fnfe) {
    // Poderá ser exibida até a primeira gravação
    Toast.makeText(getApplicationContext(),
    "Arquivo INEXISTENTE!", Toast.LENGTH_SHORT).show();
} catch (IOException e) {
    Toast.makeText(getApplicationContext(),
    "Erro de IO!", Toast.LENGTH_SHORT).show();
}
// Atualiza a lista (view) com os itens do arquivo
atualizarLista();
// Altera o título da view
setTitle("Disciplinas");
// Determina que os itens da lista serão clicáveis e
// prepara o método de controle: onItemClick(..)
listaDisciplinas.setOnItemClickListener(this); // Clique no item
}
// Método para atualizar a lista após o carregamento ou
// alguma alteração em um item
private void atualizarLista() {
    for (int i = 0; i < TOTAL_DISCIPLINAS; i++) {
        // Para cada disciplina, é preparado o vetor de
        // itens com os dados formatados
        disciplinas[i] = vetDisciplinas[i].textoLista();
    }
    // Prepara o adapter com os itens do vetor de disciplinas
    adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, disciplinas);
    // Preenche a lista com os itens do vetor de disciplinas
    listaDisciplinas.setAdapter(adapter);
}
// Método para instanciação (criação) dos objetos Disciplina
private void iniciarVetor() {
    for (int i = 0; i < TOTAL_DISCIPLINAS; i++) {
        // Instancia cada objeto
        vetDisciplinas[i] = new Disciplina();
    }
}
// Método para carregar os dados do arquivo
// Recebe o nome do arquivo
private boolean carregarDisciplinasArquivo(String arq) throws
    FileNotFoundException, IOException {
    // Define o objeto File para identificar e abrir o arquivo
    // getFilesDir() retorna o diretório corrente
}

```

```

        File file = new File(getFilesDir(), arq);
        // Cria o objeto de leitura dos dados
        DataInputStream dis = new DataInputStream(new FileInputStream(-
file));
        // Realiza a leitura e atribuição dos dados do arquivo
        for (int i = 0; i < TOTAL_DISCIPLINAS; i++) {
            // Para cada objeto do vetor de Disciplinas
            vetDisciplinas[i].setNome(dis.readUTF());
            vetDisciplinas[i].setA1(dis.readDouble());
            vetDisciplinas[i].setA2(dis.readDouble());
            vetDisciplinas[i].setA3(dis.readDouble());
        }
        // Encerra o recurso
        dis.close();
        return true;
    }

    // Trata a escolha de um item pelo usuário
    public void onItemClick(AdapterView<?> parent, View view, int posi-
tion,
                           long id) {
        // Position é a posição do item na lista
        // Armazena o índice do elemento escolhido na lista
        indiceItem = position;
        // Preparação da intent com previsão de retorno de dados
        intent = new Intent(getApplicationContext(), TratarDisciplina.
class);
        // Passa os dados do item escolhido para a atividade de trata-
mento
        intent.putExtra("indice", indiceItem);
        intent.putExtra("nome", vetDisciplinas[indiceItem].getNome());
        intent.putExtra("a1", vetDisciplinas[indiceItem].getA1());
        intent.putExtra("a2", vetDisciplinas[indiceItem].getA2());
        intent.putExtra("a3", vetDisciplinas[indiceItem].getA3());
        // Chama a atividade aguardando valores de retorno
        startActivityForResult(intent, ACTIVITY_REQUEST_DISCIPLINA);
    }

    // Método para o tratamento do retorno dos dados da atividade se-
cundária
    protected void onActivityResult(int requestCode, int resultCode,
                                    Intent data) {
        // Identifica a atividade de retorno
        if (requestCode == ACTIVITY_REQUEST_DISCIPLINA) {
            if (resultCode == RESULT_OK) {
                // Altera os dados do objeto por meio do índice
                int i = data.getIntExtra("indice", 0);
                vetDisciplinas[i].setNome(data.getStringExtra("nome"));
                vetDisciplinas[i].setA1(data.getDoubleExtra("a1", 0.0));
            }
        }
    }
}

```

```

        vetDisciplinas[i].setA2(data.getDoubleExtra("a2", 0.0));
        vetDisciplinas[i].setA3(data.getDoubleExtra("a3", 0.0));
        // Atualiza a lista na atividade principal com os novos
dados
        atualizarLista();
    }
}
}

// Método para a chamada a tratamento de exceções da gravação
public void gravarArquivo(View v) {
    // Carrega o arquivo e prepara os itens da lista
    try {
        if (armazenarDisciplinasArquivo(NOMEARQUIVO)) {
            // Mensagem de sucesso na gravação
            Toast.makeText(getApplicationContext(),
                "Arquivo gravado!", Toast.LENGTH_LONG).show();
        }
    } catch (FileNotFoundException fnfe) {
        Toast.makeText(getApplicationContext(),
            "Arquivo INEXISTENTE!", Toast.LENGTH_SHORT).show();
    } catch (IOException e) {
        Toast.makeText(getApplicationContext(),
            "Erro de IO!", Toast.LENGTH_SHORT).show();
    }
}

// Método para o armazenamento dos dados
private boolean armazenarDisciplinasArquivo(String arg) throws
    FileNotFoundException, IOException {
    // Define o objeto File para identificar e abrir o arquivo
    // getFilesDir() retorna o diretório corrente
    File file = new File(getFilesDir(), arg);
    // Cria o objeto para a gravação dos dados
    DataOutputStream dos = new DataOutputStream(new FileOutputStream(-
file));
    // Realiza a gravação dos dados de todas as disciplinas
    for (int i = 0; i < TOTAL_DISCIPLINAS; i++) {
        dos.writeUTF(vetDisciplinas[i].getNome());
        dos.writeDouble(vetDisciplinas[i].getA1());
        dos.writeDouble(vetDisciplinas[i].getA2());
        dos.writeDouble(vetDisciplinas[i].getA3());
    }
    // Libera o recurso
    dos.close();
    return true;
}
}

// Método de encerramento do aplicativo
public void sair(View v) {

```

```

    // Encerra o aplicativo
    finish();
}
}

```

Notas:

- Não se esqueça de confirmar a programação dos métodos *OnClick* dos dois botões na atividade principal e na secundária.
- O aplicativo inicia apresentando dados-padrão, uma vez que ainda não existe o arquivo de dados.
- Após a gravação, o aplicativo irá sempre iniciar com os dados alterados que foram gravados.
- Qualquer item da lista pode ser escolhido para ser alterado.
- A lista permite que seja realizada a rolagem dos itens.
- O aplicativo foi projetado inicialmente para 50 disciplinas, mas basta alterar o parâmetro (**TOTAL_DISCIPLINAS = 50**) para mudar a quantidade de disciplinas.

TratarDisciplinaActivity (atividade secundária)

a. Criação da tela (view)

Layout e codificação da view secundária do projeto:



```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".TratarDisciplina">

    <EditText
        android:id="@+id/editText4"
        android:layout_width="91dp"
        android:layout_height="41dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginBottom="8dp"
        android:ems="10"
        android:inputType="numberDecimal"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.093"
        app:layout_constraintStart_toEndOf="@+id/textView7"
        app:layout_constraintTop_toBottomOf="@+id/editText3"
        app:layout_constraintVertical_bias="0.103" />

    <EditText
        android:id="@+id/editText3"
        android:layout_width="91dp"
        android:layout_height="41dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginBottom="8dp"
        android:ems="10"
        android:inputType="numberDecimal"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.093"
        app:layout_constraintStart_toEndOf="@+id/textView6"
        app:layout_constraintTop_toBottomOf="@+id/editText2"
        app:layout_constraintVertical_bias="0.069" />

```

```

<TextView
    android:id="@+id/textView7"
    android:layout_width="52dp"
    android:layout_height="42dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:text="A3:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.088"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView6"
    app:layout_constraintVertical_bias="0.096" />

<TextView
    android:id="@+id/textView6"
    android:layout_width="52dp"
    android:layout_height="42dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:text="A2:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.088"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView5"
    app:layout_constraintVertical_bias="0.064" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="28dp"
    android:layout_marginLeft="28dp"
    android:layout_marginTop="127dp"
    android:layout_marginEnd="258dp"
    android:layout_marginRight="258dp"
    android:layout_marginBottom="585dp"
    android:text="Nome da Disciplina:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<EditText
    android:id="@+id/editText"
    android:layout_width="356dp"
    android:layout_height="48dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="172dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="511dp"
    android:ems="10"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="89dp"
    android:layout_height="31dp"
    android:layout_marginTop="20dp"
    android:text="Notas:"
    app:layout_constraintTop_toBottomOf="@+id/editText"
    tools:layout_editor_absoluteX="30dp" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="52dp"
    android:layout_height="42dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:text="A1:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.088"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView4"
    app:layout_constraintVertical_bias="0.057" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="91dp"
    android:layout_height="41dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"

```

```
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.093"
    app:layout_constraintStart_toEndOf="@+id/textView5"
    app:layout_constraintTop_toBottomOf="@+id/textView4"
    app:layout_constraintVertical_bias="0.081" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="30dp"
    android:layout_marginLeft="30dp"
    android:layout_marginTop="552dp"
    android:layout_marginEnd="277dp"
    android:layout_marginRight="277dp"
    android:layout_marginBottom="131dp"
    android:onClick="tratarVoltar"
    android:text="Confirmar"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="237dp"
    android:layout_marginLeft="237dp"
    android:layout_marginTop="552dp"
    android:layout_marginEnd="86dp"
    android:layout_marginRight="86dp"
    android:layout_marginBottom="131dp"
    android:onClick="voltar"
    android:text="Voltar"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Notas:

a.1. Foram criados dois botões:

- Um para a confirmação da alteração (**CONFIRMAR**).
- Outro botão para encerrar a atividade secundária, retornando para a atividade principal do aplicativo (**VOLTAR**).

a.2. Os dados foram enviados da atividade principal, e a tela foi preenchida com esses valores.

a.3. Após as alterações, os dados alterados são retornados à atividade principal, realizando uma troca de mensagens entre as duas atividades.

b. Codificação

Código de programação da aplicação (controller):

```
package com.example.persistencia_2;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class TratarDisciplina extends AppCompatActivity {
    EditText ed1, ed2, ed3, ed4;
    private int indice;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tratar_disciplina);
        ed1 = (EditText) findViewById(R.id.editText);
        ed2 = (EditText) findViewById(R.id.editText2);
        ed3 = (EditText) findViewById(R.id.editText3);
        ed4 = (EditText) findViewById(R.id.editText4);
        // Recebe os dados da atividade principal
        // e preenche a tela (view)
        indice = getIntent().getExtras().getInt("indice");
        ed1.setText(getIntent().getExtras().getString("nome"));
        ed2.setText(String.format("%.1f", getIntent().getExtras().getDouble("a1")));
    }
}
```

```

    ed3.setText(String.format("%.1f", getIntent().getExtras().getDouble("a2")));
    ed4.setText(String.format("%.1f", getIntent().getExtras().getDouble("a3")));
    // Altera o título da tela
    setTitle("Alterar Disciplina");
}
// Método para preparar os dados para retornar à atividade principal
public void tratarVoltar(View v) {
    // Prepara a intent de retorno da disciplina nova
    Intent returnIntent = new Intent();
    // Define os parâmetros de retorno
    returnIntent.putExtra("indice", indice);
    returnIntent.putExtra("nome", ed1.getText().toString());
    returnIntent.putExtra("a1", Double.parseDouble(ed2.getText().toString()));
    returnIntent.putExtra("a2", Double.parseDouble(ed3.getText().toString()));
    returnIntent.putExtra("a3", Double.parseDouble(ed4.getText().toString()));
    // Retorna os dados à atividade principal
    setResult(RESULT_OK, returnIntent);
    // Encerra a atividade
    finish();
}
public void voltar(View v) {
    // Encerra a atividade, retornando à atividade principal
    finish();
}
}

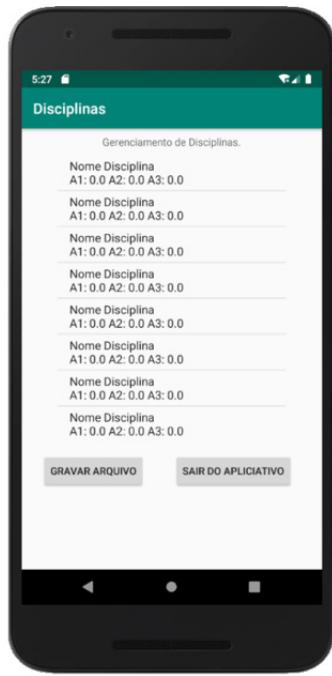
```

Notas:

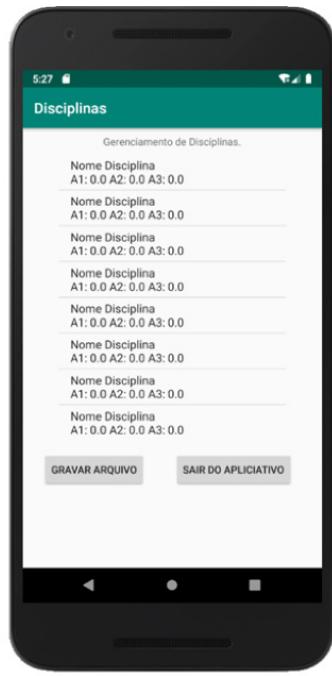
- b.1. Não se esqueça de confirmar a programação dos métodos *OnClick* dos dois botões na atividade secundária.
- b.2. A tela é configurada de acordo com os dados enviados da atividade principal.
- b.3. Após as alterações, o botão **CONFIRMAR** prepara os dados para o retorno à atividade principal.

Realize os seus testes e compare com os resultados obtidos!

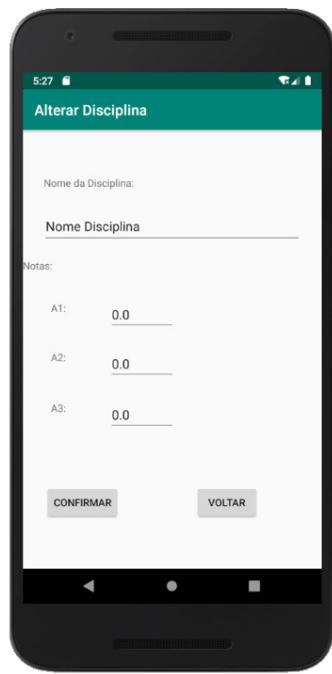
Testes realizados



Tela principal com dados-padrão



Escolha um item da lista para alterar

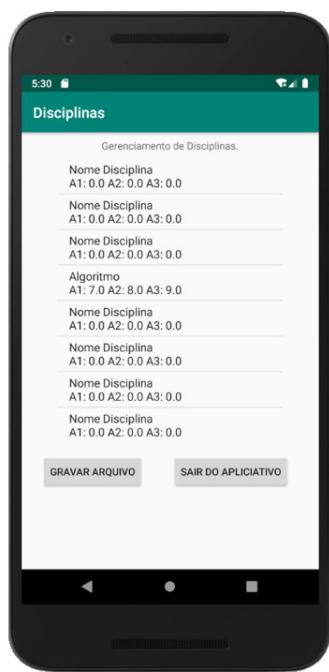


Os dados do item preenchem a tela

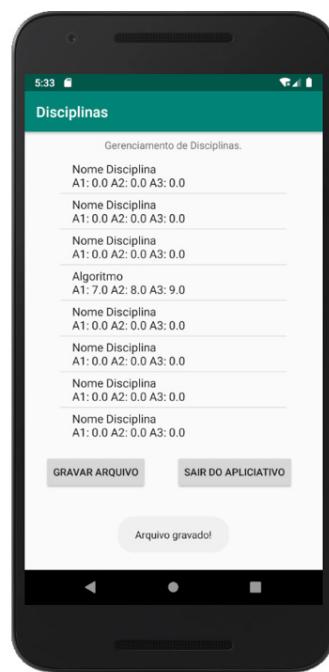


Realize as alterações e retorne, confirmando

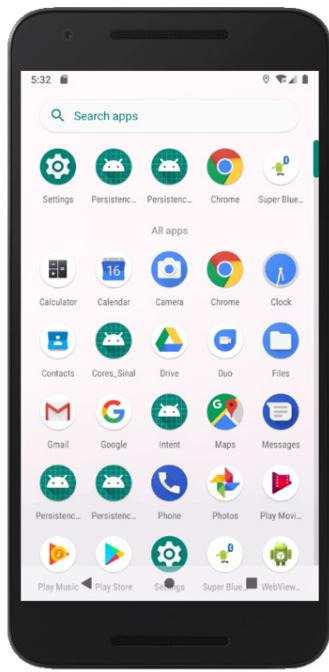
Executar novamente



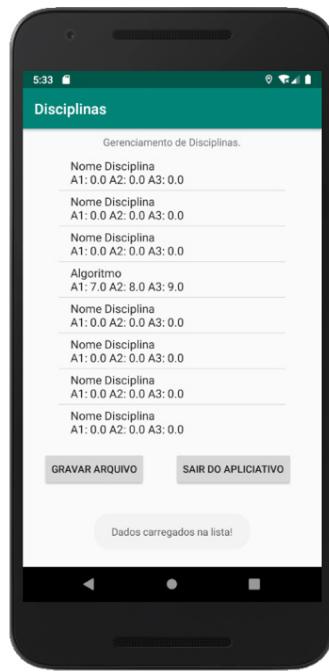
Os dados da lista foram atualizados



Realize a gravação e saia do aplicativo



Retorno ao Android para reiniciar a aplicação



O aplicativo retorna com os dados atualizados

Você ainda tem toda a potencialidade da linguagem Java para desenvolver projetos mais robustos para seus aplicativos Android.



MIDIA TECA

Veja na midiateca da Unidade 4 o conteúdo complementar selecionado pelo professor sobre **opções de armazenamento**.



MIDIA TECA

Veja na midiateca da Unidade 4 o conteúdo complementar selecionado pelo professor sobre **como trabalhar com arquivos na linguagem Java**.

Trabalhando com a persistência de dados em banco de dados local (**SQLite**)

Vimos, até o momento, formas de armazenamento de dados (persistência) com o uso da *SharedPreferences* e de arquivos como *internal storage* ou *external storage*.

Agora, chegou a hora de conhecer o uso do banco de dados **SQLite.**



Observação

O **SQLite** existe em todas as versões do Android e não é necessário instalá-lo, bem como não é necessária nenhuma ação para utilizá-lo — uma vez que não precisa de nenhum tipo de configuração, basta usá-lo.

O **SQLite** foi desenvolvido na **linguagem C** e não necessita de licenciamento. Comparado ao **MySQL**, possui algumas limitações, mas é muito mais simples e leve.

Vamos relembrar o conceito de banco de dados?

Um banco de dados é simplesmente um arquivo organizado de forma estruturada para armazenar dados de forma persistente por meio de um conjunto de tabelas.

- Cada tabela de um banco de dados é responsável por armazenar um conjunto de registros de mesma estrutura, em que cada atributo da tabela possui o seu próprio tipo de dados.
- Na programação orientada a objetos, uma tabela pode representar um objeto (atributos da tabela correspondem aos atributos de uma classe).
- Dessa forma, cada coluna de tabela representa um atributo de classe.

Assim, cada registro em uma tabela pode representar um objeto específico.

Características do **SQLite**:

- Sistema de gerenciamento de banco de dados – SGBD de código aberto.
- Atomicidade, com suporte a transações em que um conjunto de operações só será validado se todas as ações forem completadas.

- Autocontido, pois possui dependência mínima do Android.
- Acesso exclusivamente local — não permite acesso remoto por razões de segurança.
- Autoinstalado, pois é parte do sistema operacional Android.
- Suporte ao SGBD, uma vez que, com ele, podemos criar chaves primárias e estrangeiras, views e índices, dando suporte, inclusive, ao uso de transações e *triggers* (“gatilhos”), *JOINS GRANT* e *REVOKE*.
- Banco de dados com a maioria dos recursos encontrados em qualquer banco de dados relacional.

Programação em camadas

Conforme visto no início da nossa disciplina, com o SQLite podemos trabalhar em três camadas.

Versões do SQLite no Android:

| API do Android | Versão do SQLite |
|----------------|------------------|
| API 27 | 3.19 |
| API 26 | 3.18 |
| API 24 | 3.09 |
| API 21 | 3.08 |
| API 11 | 3.07 |
| API 8 | 3.06 |
| API 3 | 3.05 |
| API 1 | 3.04 |

Quando o aplicativo cria um banco de dados, este, por padrão, é salvo no diretório *DATA/data/APP_NAME/databases/Filename*.

A forma para a definição do caminho para a identificação do diretório é feita de acordo com as seguintes regras:

- *DATA*: o caminho retornado pelo método *Environment.getDataDirectory()* da aplicação.
- *APP_NAME*: nome do seu aplicativo.
- *FILENAME*: nome que você definiu no código do seu aplicativo para o banco de dados.

Tipos de dados no *SQLite*

a. Afinidades de tipos

O *SQLite* possui poucos tipos de dados, mas tem suporte ao conceito de **afinidade de tipo em colunas**.



Saiba mais

Qualquer coluna pode armazenar qualquer tipo de dados, mas o tipo preferencial para os dados de uma coluna é chamado de **afinidade**.

Para cada coluna de tabela em um banco de dados *SQLite* pode ser atribuída uma das seguintes afinidades de tipo:

| Tipos de dados mais comuns | Afinidade no <i>SQLite</i> | Descrição de afinidade em <i>SQLite</i> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------|------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none">• INT• INTEGER• TINYINT• SMALLINT• MEDIUMINT• BIGINT• UNSIGNED BIG INT• INT2• INT8 | INTEGER | Trabalha da mesma forma que uma coluna com afinidade numérica, forçando o armazenamento para tipo inteiro. |

| | | |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|----------------|---------------------------------------------------------------------------------------------------------------------------------|
| • BLOB • no datatype specified | NONE | Uma coluna com afinidade <i>NONE</i> não apresenta preferência de armazenamento. |
| • NUMERIC • DECIMAL (10,5) • BOOLEAN • DATE • DATETIME | NUMERIC | Essa coluna pode conter valores usando todos os tipos numéricos de armazenamento. |
| • REAL • DOUBLE • DOUBLE PRECISION • FLOAT | REAL | Trabalha da mesma forma que uma coluna com afinidade numérica, forçando valores inteiros para representação em ponto flutuante. |
| • CHARACTER (20) • VARCHAR (255) • VARYING CHARACTER (255) • NCHAR (55) • NATIVE CHARACTER (70) • NVARCHAR (100) • TEXT • CLOB | TEXT | Essa coluna armazena os dados usando tipos de armazenamento <i>NULL</i> , <i>TEXT</i> ou <i>BLOB</i> . |

b. Tipo de dados *booleano*

O *SQLite* não possui uma forma padrão de armazenamento de dados do tipo *booleano*, mas valores *booleanos* podem, por exemplo, ser armazenados como números inteiros 0 (*false*) e 1 (*true*).

c. Tipos de dados de data e hora

Apesar de não possuir uma forma de armazenamento para **data** e **hora**, o *SQLite* pode armazenar esses dados com valores dos tipos **TEXT**, **REAL** ou **INTEGER**, conforme tabela a seguir.

| Tipos | Classe de armazenamento & data formate |
|----------------|-------------------------------------------------------------------------------|
| TEXT | Podemos armazenar uma data no formato: "AAAA-MM-DD HH:MM:SS.SSS" |
| REAL | O número de dias desde o meio-dia em Greenwich em 24 de novembro de 4714 a.C. |
| INTEGER | O número de segundos desde 1970-01-01 00:00:00 UTC. |

Você pode utilizar:

- Qualquer das formas que constam na tabela para o armazenamento de data e hora.
- Métodos da linguagem Java para converter esses formatos para os formatos suportados pela linguagem.

Vamos ver um exemplo prático usando o banco de dados *SQLite*?

Para podermos identificar os registros, usaremos o padrão do **atributo *id***, que é um atributo do tipo autoincrementado – contador que recebe mais uma unidade a cada nova inclusão e é usado para identificar o registro dentro da tabela.

Por isso, tanto a tabela quanto a classe *Disciplina* terão a inclusão desse atributo.

A aplicação fará a **alteração** ou **exclusão** a partir da escolha de um item da lista de disciplinas. Já a **inclusão** ocorrerá a partir de um botão.

Em todos os três casos, a tela secundária irá atender à ação solicitada, uma vez que ela será preenchida em função da escolha:

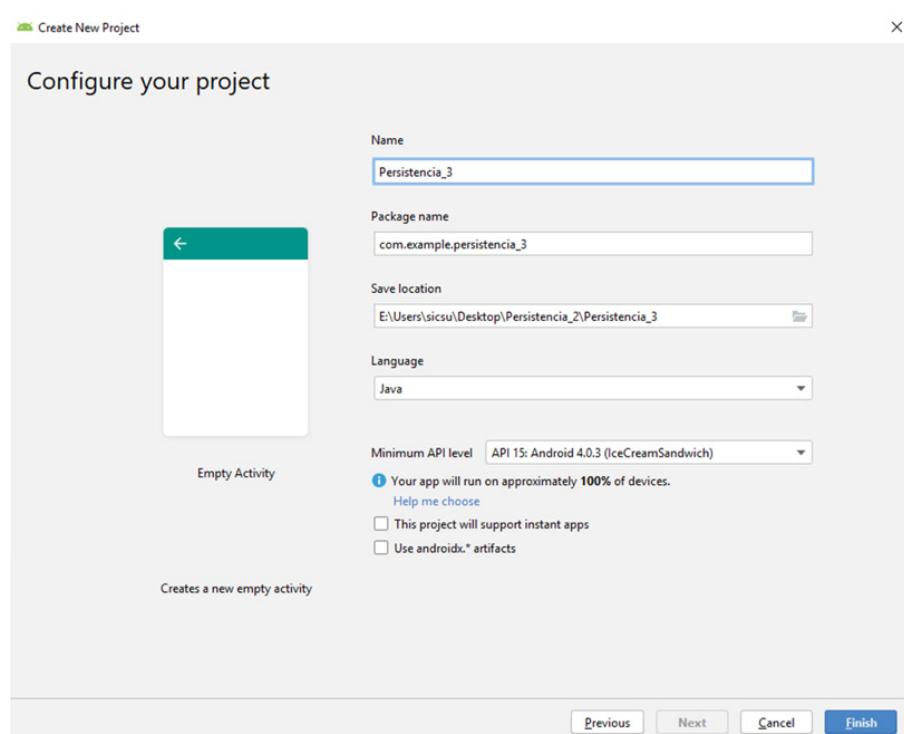
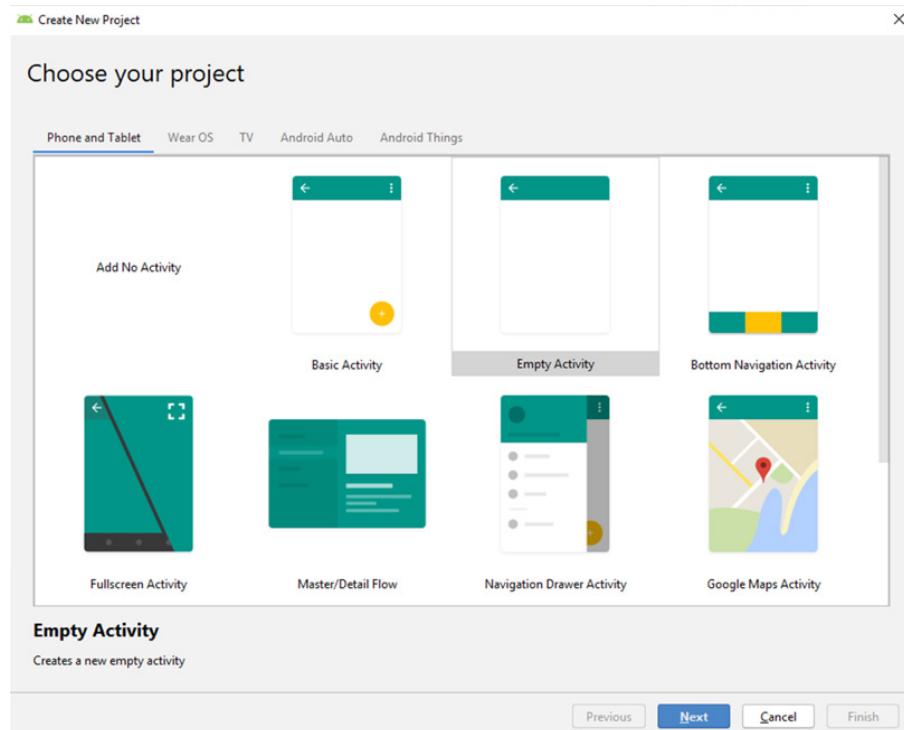
- -1 para inclusão.
- 0 para alteração ou exclusão.

O projeto foi desenvolvido em três camadas, com as classes de acesso à manipulação dos dados do banco de dados por meio de classes próprias.

Vamos praticar?

- I. Crie um novo projeto compatível com a API 15 e codificação em Java.
- II. O projeto terá o nome de **Persistencia_3**.
- III. Caso você altere os nomes da sua atividade, certifique-se de que fará o mesmo com a classe da atividade e com a chamada da view.

Criação do projeto Persistencia_3



Android Studio

Atividade principal

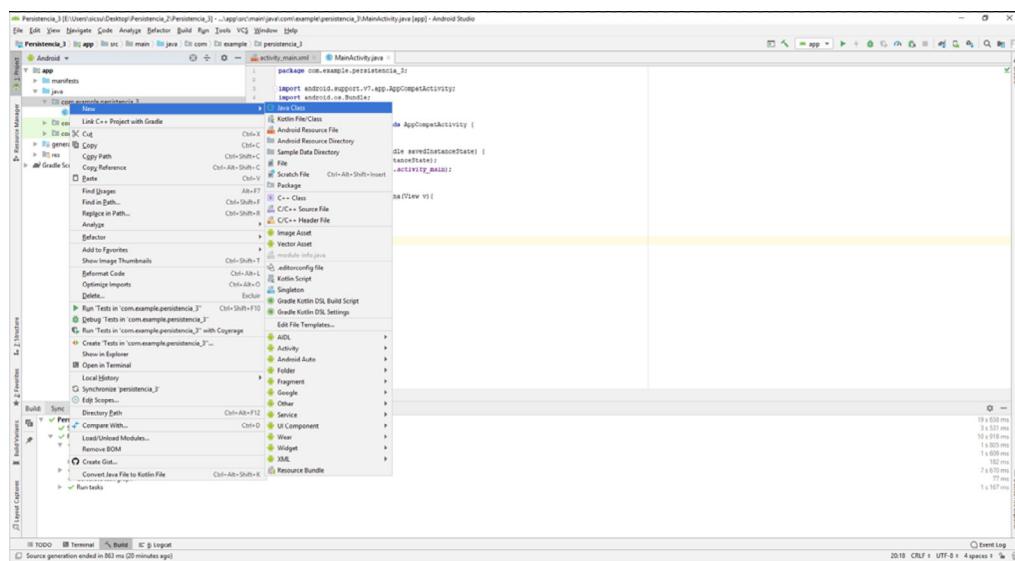
a. Classe Disciplina

Devemos incluir uma classe Java em nosso projeto.

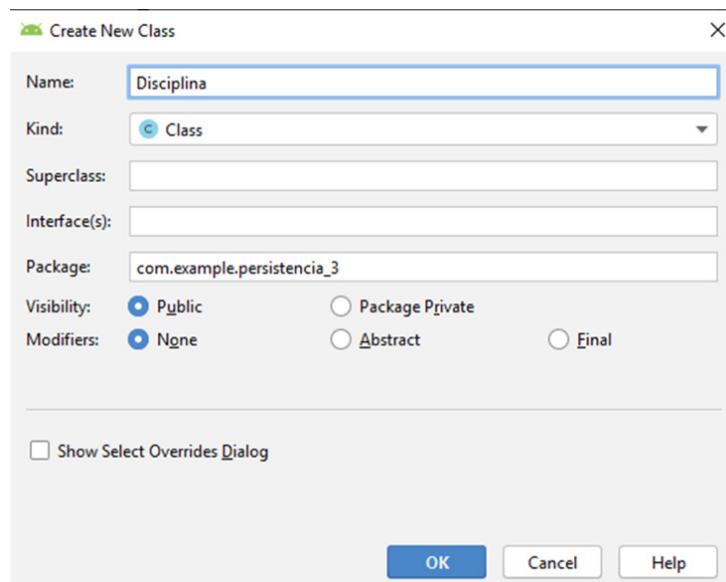
A classe *Disciplina* será usada pela atividade principal. Por isso, deve ser criada antes de codificarmos a atividade principal. Além disso, foi criado um novo atributo, *long id*, que será responsável por identificar o *id* do registro na tabela do banco de dados.

Observe, a seguir, a inclusão de uma nova classe Java.

1. Selecione a aplicação e escolha **New > Java Class**.



2. Defina o nome da classe, **Disciplina**, e confirme com **OK**.



a.1. Codificação da nova classe Java: *Disciplina*.

```
package com.example.persistencia_3;

public class Disciplina {
    // Definição dos atributos
    // O atributo id não existia na versão anterior
    private long id;
    private String nome;
    private double a1, a2, a3;

    // Métodos novos, de acesso do atributo id
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    // Métodos de acesso (setters & getters)
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        if (!nome.isEmpty()) {
            this.nome = nome;
        }
    }
    public double getA1() {
        return a1;
    }
    public void setA1(double a1) {
        if (a1 >= 0) {
            this.a1 = a1;
        }
    }
    public double getA2() {

        return a2;
    }
    public void setA2(double a2) {
        if (a2 >= 0) {
            this.a2 = a2;
        }
    }
    public double getA3() {
        return a3;
    }
}
```

```

public void setA3(double a3) {
    if (a3 >= 0) {
        this.a3 = a3;
    }
}

// Método construtor com definição dos valores-padrão
public Disciplina() {
    nome = "Nome Disciplina";
    a1 = 0.0;
    a2 = 0.0;
    a3 = 0.0;
}

// Método de formatação dos dados para exibição na lista
public String textoLista() {
    String item;
    item = getNome();
    item += "\nA1: " + String.format("%3.1f", getA1());
    item += "\tA2: " + String.format("%3.1f", getA2());
    item += "\tA3: " + String.format("%3.1f", getA3());
    return item;
}
}

```

Notas:

- Foi incluído o atributo *id* para armazenar o *id* do registro.
- Foram acrescentados os métodos de acesso para o novo atributo.

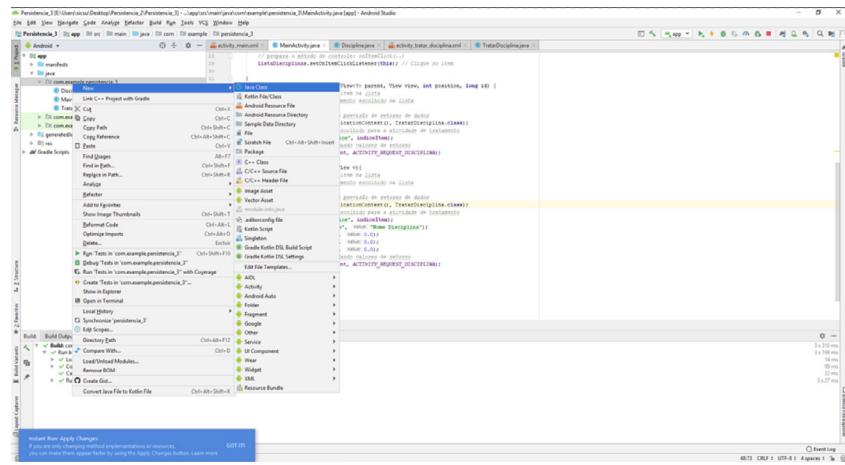
b. Classe *DisciplinaSQLiteOpenHelper*

Devemos incluir uma classe Java em nosso projeto.

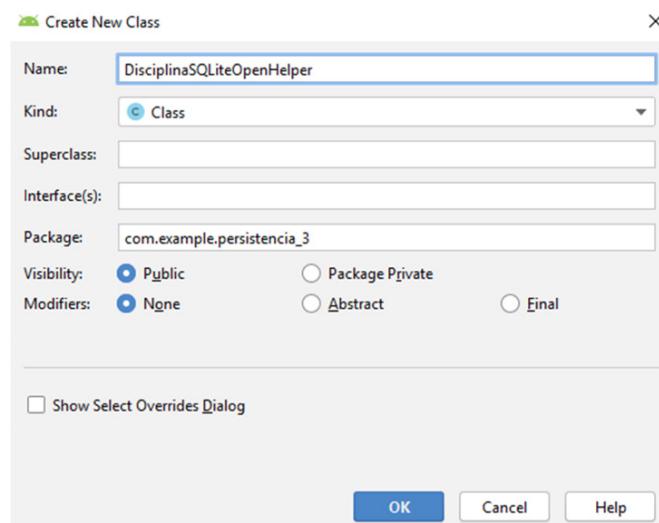
A classe *DisciplinaSQLiteOpenHelper* será responsável pelo acesso ao banco de dados *SQLite*.

Observe, a seguir, a inclusão de uma nova classe Java:

1. Selecione a aplicação e escolha **New > Java Class**.



2. Defina o nome da classe, **DisciplinaSQLiteOpenHelper**, e confirme com **OK**.



b.1. Codificação da nova classe Java: **DisciplinaSQLiteOpenHelper**

```
package com.example.persistencia_3;

import android.content.Context ;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DisciplinaSQLiteOpenHelper extends SQLiteOpenHelper {
    // Definição da tabela e dos atributos
    // Os espaços antes e depois dos nomes são importantes,
    // pois será criada uma QUERY (consulta) com esses identificadores
```

```

public static final String TABELA = " Disciplina ";
public static final String COLUNA_ID = " id ";
public static final String COLUNA_NOME = " nome ";
public static final String COLUNA_A1 = " a1 ";
public static final String COLUNA_A2 = " a2 ";
public static final String COLUNA_A3 = " a3 ";
// Define o nome do banco de dados
private static final String DATABASE_NAME = "disciplinas.db";
// Define a versão do banco
// O valor deve ser incrementado somente em casos de alterações
// na estrutura do banco (tabelas e atributos)
private static final int DATABASE_VERSION = 1;
// String de criação da tabela no banco de dados
private static final String CRIAR_BANCO = " create table "
    + TABELA + "("
    + COLUNA_ID + " integer primary key autoincrement , "
    + COLUNA_NOME + " text not null , "
    + COLUNA_A1 + " double not null , "
    + COLUNA_A2 + " double not null , "
    + COLUNA_A3 + " double not null ) ;";
// Método construtor da classe para criação ou atualização do banco
public DisciplinaSQLiteOpenHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
// Cria o banco de dados, caso não exista
@Override
public void onCreate ( SQLiteDatabase database ) {
    database . execSQL ( CRIAR_BANCO );
}
// Atualiza o banco de dados, caso seja uma nova versão
@Override
public void onUpgrade ( SQLiteDatabase db, int oldVersion, int new-
Version) {
    db. execSQL ( " DROP TABLE IF EXISTS " + TABELA );
    onCreate (db);
}
}

```

Notas:

- Essa classe é responsável pela manutenção do banco, criando-o, caso não exista, ou atualizando-o, caso seja uma nova versão.
- As estruturas das tabelas e os respectivos atributos devem ser determinados. Contudo, neste exemplo, temos apenas uma tabela no banco de dados.

- O método `onCreate` criará a tabela caso ela não exista.
 - O método `onUpgrade` atualizará a tabela caso seja identificada uma nova versão (incremento do atributo: **`DATABASE_VERSION`**).
 - Os dados não são perdidos durante o processo de atualização.

c. Classe *Disciplina.DAO*

Devemos incluir uma classe Java em nosso projeto.

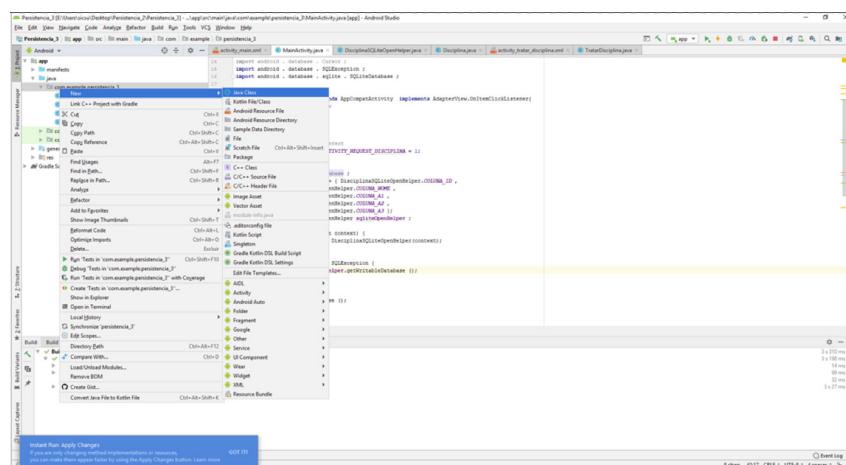
A classe *Disciplina_DAO* será responsável pela manipulação da tabela de disciplinas.

Essa classe possuirá os métodos de:

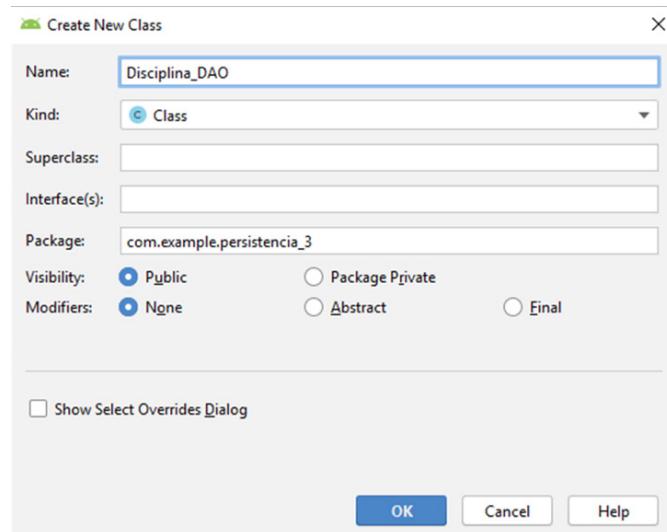
- Abertura e fechamento do banco.
 - Inclusão, alteração, exclusão, busca e montagem da lista de disciplinas.

Observe, a seguir, a inclusão de uma nova classe Java:

1. Selecione a aplicação e escolha **New > Java Class**.



2. Defina o nome da classe, ***Disciplina.DAO***, e **confirme com OK**.



c.1. Codificação da nova classe Java: ***Disciplina.DAO***

```
package com.example.persistencia_3;

import java.util.ArrayList ;
import java.util.List ;

import android.content.ContentValues ;
import android.content.Context ;
import android.database.Cursor ;
import android.database.SQLException ;
import android.database.sqlite . SQLiteDatabase ;

// Camada de dados (Persistência)
// Classe de definição para os acessos ao banco de dados
public class Disciplina _ DAO {
    // Define um objeto banco de dados
    private SQLiteDatabase database;
    // Define as colunas da tabela
    private String [] columns = { DisciplinaSQLiteOpenHelper.COLUNA _ ID
        ,
        DisciplinaSQLiteOpenHelper.COLUNA _ NOME ,
        DisciplinaSQLiteOpenHelper.COLUNA _ A1 ,
        DisciplinaSQLiteOpenHelper.COLUNA _ A2 ,
        DisciplinaSQLiteOpenHelper.COLUNA _ A3 };
    private DisciplinaSQLiteOpenHelper sqliteOpenHelper ;
    // Método construtor
    public Disciplina _ DAO(Context context) {
```

```

    sqliteOpenHelper = new DisciplinaSQLiteOpenHelper(context);
}

// Método para abrir o recurso de banco de dados
public void open () throws SQLException {
    database = sqliteOpenHelper.getWritableDatabase();
}

// Método para fechar o recurso de banco de dados
public void close() {
    sqliteOpenHelper.close ();
}

// Método de inclusão de registro
public void inserir ( String nome, double a1, double a2, double a3
) {
    // Prepara os valores das colunas da tabela para a inserção
    ContentValues values = new ContentValues ();
    values.put ( DisciplinaSQLiteOpenHelper.COLUNA_NOME , nome );
    values.put ( DisciplinaSQLiteOpenHelper.COLUNA_A1 , String.
    valueOf(a1) );
    values.put ( DisciplinaSQLiteOpenHelper.COLUNA_A2 , String.
    valueOf(a2) );
    values.put ( DisciplinaSQLiteOpenHelper.COLUNA_A3 , String.
    valueOf(a3) );
    // Efetua a inclusão com retorno do id do registro
    long insertId = database.insert ( DisciplinaSQLiteOpenHelper.TA-
BELA ,
                                    null , values );
}

// Método para atualização de um registro da tabela
// Recebe os dados para a inserção
public void alterar(long id, String nome, double a1, double a2,
double a3){
    // Prepara os dados para a atualização
    ContentValues values = new ContentValues ();
    values.put ( DisciplinaSQLiteOpenHelper.COLUNA_NOME , nome );
    values.put ( DisciplinaSQLiteOpenHelper.COLUNA_A1 , String.
    valueOf(a1) );
    values.put ( DisciplinaSQLiteOpenHelper.COLUNA_A2 , String.
    valueOf(a2) );
    values.put ( DisciplinaSQLiteOpenHelper.COLUNA_A3 , String.
    valueOf(a3) );
    // Efetua a alteração do registro, de acordo com o id do registro
    database.update(DisciplinaSQLiteOpenHelper.TABELA , values,
                    DisciplinaSQLiteOpenHelper.COLUNA_ID + "=" + id, null);
}

// Método para a exclusão de um registro
// Recebe o id do registro que será excluído
public void apagar ( long id ) {

```

```

    // Exclui o registro a partir do id
    database.delete ( DisciplinaSQLiteOpenHelper.TABELA ,
                      DisciplinaSQLiteOpenHelper.COLUNA_ID
                      + " = " + id , null );
}

// Método de busca de registro a partir do id
// Busca os dados de cada registro a partir do id
// Retorna um objeto Disciplina com os dados do registro
public Disciplina buscar ( long id ) {
    // Realiza a busca a partir do id do registro
    // Usado para buscar os dados para a montagem da tela e
    // para a alteração ou exclusão do registro
    // O objeto cursor armazena os registros da consulta
    // Como o filtro é o id, apenas um registro será selecionado
    Cursor cursor = database.query( DisciplinaSQLiteOpenHelper.TABELA,
                                    columns , DisciplinaSQLiteOpenHelper.COLUNA_ID + " = " + id,
                                    null , null , null , null );
    cursor . moveToFirst ();
    // Cria um objeto Disciplina auxiliar para retornar o objeto
    Disciplina disciplina = new Disciplina ();
    // Preenche os dados do registro do banco
    // nas propriedades do objeto auxiliar
    disciplina.setId ( cursor.getLong (0) );
    disciplina.setNome ( cursor.getString (1) );
    disciplina.setA1 ( cursor.getDouble (2) );
    disciplina.setA2 ( cursor.getDouble (3) );
    disciplina.setA3 ( cursor.getDouble (4) );
    // Fecha o recurso do cursor
    cursor.close();
    // Retorna o objeto auxiliar do tipo Disciplina
    return disciplina ;
}

// Método de montagem da lista de registros das disciplinas
// Monta a lista de disciplinas para a carga da lista na tela principal
// O método retornará uma lista com todas as disciplinas armazenadas
// no banco
public List <Disciplina > getAll () {
    // Prepara um ArrayList para retorno dos registros armazenados
    no banco
    List <Disciplina > disciplinas = new ArrayList <Disciplina>() ;
    // Objeto cursor para armazenar temporariamente os dados
    // retornados pela consulta
    Cursor cursor = database . query ( DisciplinaSQLiteOpenHelper .
                                         TABELA , columns , null , null , null , null );

```

```

// Passa o ponteiro para o primeiro registro do cursor
cursor . moveToFirst ();
// Para cada registro, os dados da tabela são copiados para
// o objeto Disciplina da lista
while (!cursor.isAfterLast ()) {
    Disciplina disciplina = new Disciplina ();
    disciplina.setId ( cursor . getLong (0) );
    disciplina.setNome ( cursor . getString (1) );
    disciplina.setA1 ( cursor.getDouble (2) );
    disciplina.setA2 ( cursor.getDouble (3) );
    disciplina.setA3 ( cursor.getDouble (4) );
    disciplinas.add ( disciplina );
    cursor . moveToNext ();
}
// Fecha o cursor
cursor . close ();
// Retorna à lista de disciplinas
return disciplinas;
}
}

```

Notas:

- Essa classe é responsável pela manutenção da tabela de disciplinas.
- Essa classe apresenta os métodos de abertura e fechamento do banco.
- Essa classe apresenta os métodos de inserção, alteração e exclusão de registros.
- O método *buscar* é responsável por retornar os dados de um determinado registro, por meio do *id* do registro.
- O método *getAll* é responsável por retornar todos os registros da tabela de disciplinas para a montagem da lista na tela principal.

d. *MainActivity*

Criação da tela (view).

Layout e codificação da view do projeto:



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/lista"
        android:layout_width="312dp"
        android:layout_height="388dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="36dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"></ListView>
```

```

<Button
    android:id="@+id/btSair"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="20dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:onClick="sair"
    android:text="Sair do Aplicativo"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/btIncluir"
    app:layout_constraintTop_toBottomOf="@+id/lista" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:text="Gerenciamento de Disciplinas."
    app:layout_constraintBottom_toTopOf="@+id/lista"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.457" />

<Button
    android:id="@+id/btIncluir"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="20dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:onClick="incluirDisciplina"
    android:text="Inserir Disiplina"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.078"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/lista" />

</android.support.constraint.ConstraintLayout>

```

Notas:

- Dois botões foram criados – um para a inclusão de uma nova disciplina (INSERIR DISCIPLINA) e outro para encerrar o aplicativo (SAIR DO APLICATIVO).
- Uma LISTA (*ListView*) foi utilizada – cada item pode ser clicado de forma a chamar a **atividade TratarDisciplina** para realizar as alterações dos dados, de acordo com o índice do elemento na lista, ou para a exclusão de uma disciplina.
- O botão de inclusão usa a mesma **atividade TratarDisciplina**, por isso a atividade secundária fará uma organização da tela em função da ação a ser executada (inclusão, alteração ou exclusão).

d.1. Codificação

Código de programação da aplicação (*controller*):

```
package com.example.persistencia_3;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.Iterator;
import java.util.List;

public class MainActivity extends AppCompatActivity implements AdapterView.OnItemClickListener{
    // Componente lista da view
    ListView lista;

    // Declaração da intent
    Intent intent;
    // Determina o código da intent
    public static final int ACTIVITY_REQUEST_DISCIPLINA = 1;
    // Declara o objeto de persistência (acesso ao banco)
    private Disciplina_DAO dao ;

    // Vetor com os dados das disciplinas para apresentar na lista
    private String[] disciplinas;
    // Vetor com o id dos registros para identificar o registro
    // para os casos de seleção na lista para alteração ou
```

```

// exclusão
private long[] idDisciplinas;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    lista = (ListView) findViewById(R.id.lista);
    // Altera o título da janela da atividade
    setTitle("Banco de Dados com SQLite!");
    // Cria o objeto de acesso ao banco
    dao = new Disciplina_DAO(this);
    // Abre o banco
    dao.open();

    // Determina que os itens da lista serão clicáveis e
    // prepara o método de controle: onItemClick(..)
    lista.setOnItemClickListener(this); // Clique no item
}

// Sempre que a atividade passar pelo método onResume, a lista
// será atualizada
@Override
protected void onResume () {
    dao.open ();
    super.onResume ();
    // Prepara a lista que será exibida ao usuário na tela
    // Busca no banco todos os registros
    List<Disciplina> listaDisciplinas = dao.getAll();
    // Define o tamanho dos vetores de disciplinas e id
    // em função do tamanho (quantidade de registros) da tabela
    disciplinas = new String[listaDisciplinas.size()];
    idDisciplinas = new long[listaDisciplinas.size()];
    // Primeiro índice dos vetores
    int i =0;
    // Cria um objeto iterator para preencher o vetor de disciplinas
    // com os dados dos registros
    Iterator<Disciplina> iterator = listaDisciplinas.iterator();
    // Para cada registro preencher os vetores
    while (iterator.hasNext()) {
        // Objeto disciplina auxiliar
        Disciplina aux = new Disciplina();
        // Recebe no objeto auxiliar os dados de cada registro
        aux = (Disciplina) iterator.next();
        // Preenche o vetor de disciplinas para visualização na lista
        disciplinas[i] = aux.textoLista();
        // Preenche o vetor de id para identificação do registro
        // ao se escolher um elemento da lista para
}

```

```

        // a alteração ou exclusão
        idDisciplinas[i] = aux.getId();
        // Próximo item dos vetores
        i++;
    }
    // Carrega o ArrayAdapter com os dados do vetor de disciplinas
    ArrayAdapter <String> adapter = new ArrayAdapter<String>(
this ,
        android.R.layout.simple_list_item_1 , disciplinas );
    // Preenche a lista com os dados do ArrayAdapter
    lista.setAdapter( adapter );
}
// Fecha o recurso de acesso ao banco sempre que a atividade passar
// por uma pausa
@Override
protected void onPause () {
    // Libera o recurso de acesso ao banco
    dao.close ();
    super.onPause ();
}
// Método para seleção de um item da lista para
// alteração ou exclusão (atende à interface: AdapterView.OnItem
ClickListener)
public void onItemClick(AdapterView<?> parent, View view, int posi
tion, long ident) {
    // Position é a posição do item na lista
    // Armazena o índice do elemento escolhido na lista
    long id = idDisciplinas[position];
    // Preparação da intent com previsão de retorno de dados
    intent = new Intent(getApplicationContext(), TratarDisciplina.
class);
    // Passa os dados do item escolhido para a atividade de trata
mento
    intent.putExtra("acao", 0);
    intent.putExtra("id", id);
    // Chama a atividade
    startActivity(intent);
}
// Método de controle para chamada da atividade secundária para
inclusão
public void incluirDisciplina(View v){
    // Intent para chamar a atividade secundária
    intent = new Intent(getApplicationContext(), TratarDisciplina.
class);
    // Passa os dados do item escolhido para a atividade de trata
mento
    intent.putExtra("acao", -1);
}

```

```

        intent.putExtra("id", 0L);
        // Chama a atividade
        startActivity(intent);
    }
    // Método para encerrar o aplicativo
    public void sair(View v){
        finish();
    }
}

```

Notas:

- Os métodos `onResume` e `onPause` são responsáveis por atualizar a lista e encerrar recursos, respectivamente.
- Como a lista é atualizada automaticamente no método `onResume`, não precisamos realizar de forma explícita essa ação, pois, sempre que retornar para a atividade principal, a lista será atualizada.

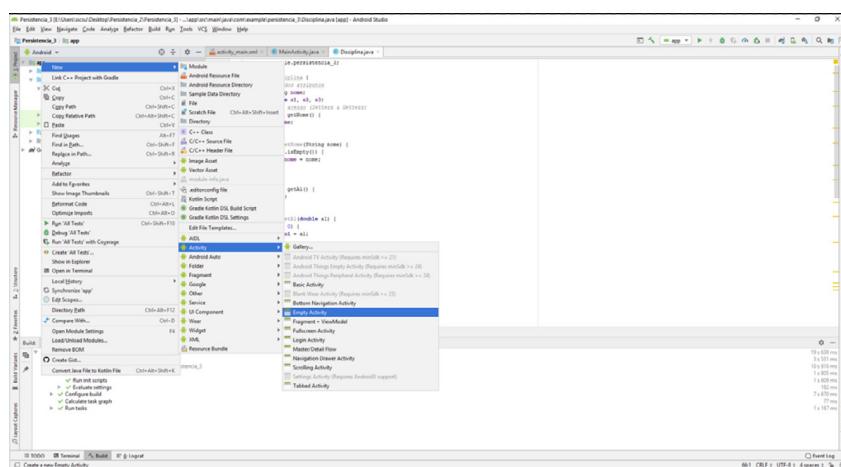
Ainda temos mais dois métodos:

- Um método para tratar a escolha do usuário na lista de disciplinas, que realizará uma alteração ou exclusão.
- Um método para a inclusão de uma nova disciplina.

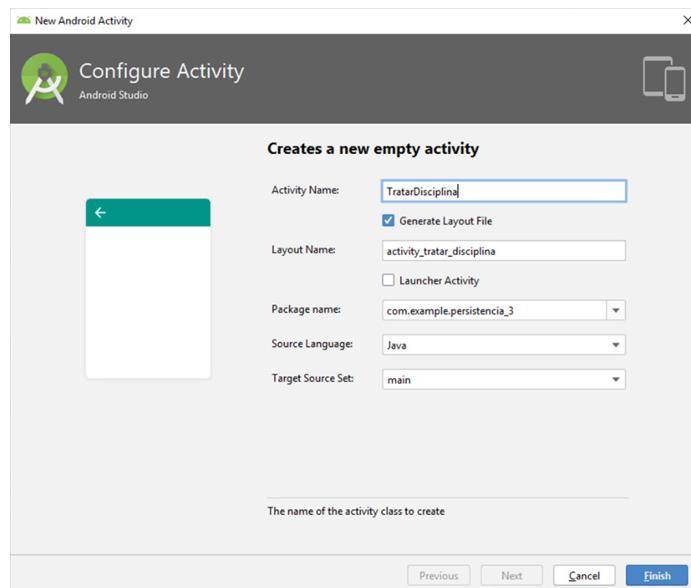
TratarDisciplinaActivity (atividade secundária)

Observe, a seguir, a inclusão de uma nova atividade Java.

1. Selecione a aplicação e escolha **New > Activity > Empty Activity**.



2. Defina o nome da atividade, **TratarDisciplina**, e **confirme com OK**.



Android Studio

a. Criação da tela (view)

Layout e codificação da view secundária do projeto:



```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".TratarDisciplina">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="49dp"
        android:layout_marginLeft="49dp"
        android:layout_marginTop="552dp"
        android:layout_marginEnd="274dp"
        android:layout_marginRight="274dp"
        android:layout_marginBottom="131dp"
        android:onClick="alterarInserir"
        android:text="Alterar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/editText4"
        android:layout_width="91dp"
        android:layout_height="41dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginBottom="8dp"
        android:ems="10"
        android:inputType="numberDecimal"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.093"
        app:layout_constraintStart_toEndOf="@+id/textView7"
        app:layout_constraintTop_toBottomOf="@+id/editText3"
        app:layout_constraintVertical_bias="0.103" />

    <EditText
        android:id="@+id/editText3"

```

```
    android:layout_width="91dp"
    android:layout_height="41dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.093"
    app:layout_constraintStart_toEndOf="@+id/textView6"
    app:layout_constraintTop_toBottomOf="@+id/editText2"
    app:layout_constraintVertical_bias="0.069" />

<TextView
    android:id="@+id/textView7"
    android:layout_width="52dp"
    android:layout_height="42dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:text="A3:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.088"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView6"
    app:layout_constraintVertical_bias="0.096" />

<TextView
    android:id="@+id/textView6"
    android:layout_width="52dp"
    android:layout_height="42dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:text="A2:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.088"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView5"
    app:layout_constraintVertical_bias="0.064" />
```

```

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="28dp"
    android:layout_marginLeft="28dp"
    android:layout_marginTop="127dp"
    android:layout_marginEnd="258dp"
    android:layout_marginRight="258dp"
    android:layout_marginBottom="585dp"
    android:text="Nome da Disciplina:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editText"
    android:layout_width="356dp"
    android:layout_height="48dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="172dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="511dp"
    android:ems="10"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView4"
    android:layout_width="89dp"
    android:layout_height="31dp"
    android:layout_marginTop="20dp"
    android:text="Notas:"
    app:layout_constraintTop_toBottomOf="@+id/editText"
    tools:layout_editor_absoluteX="30dp" />

<TextView
    android:id="@+id/textView5"
    android:layout_width="52dp"
    android:layout_height="42dp"
    android:layout_marginStart="8dp"

```

```
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:text="A1:"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.088"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView4"
    app:layout_constraintVertical_bias="0.057" />

<EditText
    android:id="@+id/editText2"
    android:layout_width="91dp"
    android:layout_height="41dp"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:ems="10"
    android:inputType="numberDecimal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.093"
    app:layout_constraintStart_toEndOf="@+id/textView5"
    app:layout_constraintTop_toBottomOf="@+id/textView4"
    app:layout_constraintVertical_bias="0.081" />

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="284dp"
    android:layout_marginLeft="284dp"
    android:layout_marginTop="552dp"
    android:layout_marginEnd="39dp"
    android:layout_marginRight="39dp"
    android:layout_marginBottom="131dp"
    android:onClick="excluir"
    android:text="Excluir"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```

<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="153dp"
    android:layout_marginLeft="153dp"
    android:layout_marginTop="593dp"
    android:layout_marginEnd="170dp"
    android:layout_marginRight="170dp"
    android:layout_marginBottom="90dp"
    android:onClick="voltar"
    android:text="Voltar"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>

```

Notas:

a.1. Três botões foram criados:

- Um botão para a inclusão ou alteração – o texto será alterado de acordo com a ação (-1, inclusão, ou 0, alteração ou exclusão).
- Um botão para a exclusão que só estará disponível se for uma ação (0 de alteração ou exclusão).
- E outro botão para encerrar a atividade secundária, retornando para a atividade principal do aplicativo (**VOLTAR**).

a.2. Não se esqueça de atribuir os nomes dos métodos ao evento **OnCLick** dos botões.

a.3. Essa tela tem diferentes configurações e textos. Para o caso de:

- Uma inclusão (acao = -1), o botão de **EXCLUIR** é desabilitado, e o texto do primeiro botão será **INCLUIR**.
- Uma ação de alteração ou exclusão (acao = 0), o texto do primeiro botão será **ALTERAR**, e o botão de **EXCLUIR** estará habilitado.

a.4. O usuário:

- Preencherá os dados a partir dos valores-padrão para uma inclusão.
- Realizará as alterações necessárias sobre os dados trazidos da tabela que serão atualizados para a alteração.
- Verificará os dados trazidos da tabela para confirmar a exclusão.
- Poderá retornar sem nenhuma ação caso utilize o botão **VOLTAR**.

b. Codificação

Código de programação da aplicação (*controller*):

```
package com.example.persistencia_3;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class TratarDisciplina extends AppCompatActivity {
    // Declaração dos componentes de tela
    EditText ed1, ed2, ed3, ed4;
    Button bt1, bt2;
    // Atributos de informações de acao: 0 : alteração ou exclusão
    // -1 : inclusão
    private int acao;
    // Id do registro para alteração ou exclusão
    // será usado com o método buscar para trazer os dados do registro
    private long id;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tratar_disciplina);
        bt1 = (Button) findViewById(R.id.button);
        bt2 = (Button) findViewById(R.id.button2);
        ed1 = (EditText) findViewById(R.id.editText);
        ed2 = (EditText) findViewById(R.id.editText2);
        ed3 = (EditText) findViewById(R.id.editText3);
        ed4 = (EditText) findViewById(R.id.editText4);
        // Recebe os dados da atividade principal
        // e preenche a tela (view)
        acao = getIntent().getExtras().getInt("acao");
        id = getIntent().getExtras().getLong("id");
```

```

// Altera o título e prepara a tela
if (acao == -1) {
    // Tela de inclusão
    // Usa dados-padrão
    setTitle("Inserir Disciplina");
    bt1.setText("Incluir");
    bt2.setEnabled(false);
    ed1.setText("Nome Disciplina");
    ed2.setText(String.format("%.1f", 0.0));
    ed3.setText(String.format("%.1f", 0.0));
    ed4.setText(String.format("%.1f", 0.0));
} else {
    // Tela de alteração ou exclusão
    setTitle("Alterar ou Excluir Disciplina");
    // Cria um objeto Disciplina auxiliar para armazenar
    // os dados do registro
    Disciplina aux = new Disciplina();
    // Cria o objeto de acesso ao banco
    Disciplina _ DAO dao = new Disciplina _ DAO(this);
    // Abre o banco
    dao.open();
    // Faz a consulta pelo id do registro para
    // buscar os dados na tabela
    aux = dao.buscar(id);
    // Preenche os dados do registro na tela
    ed1.setText(aux.getNome());
    ed2.setText(String.format("%.1f", aux.getA1()));
    ed3.setText(String.format("%.1f", aux.getA2()));
    ed4.setText(String.format("%.1f", aux.getA3()));
    // Libera o recurso de acesso ao banco de dados
    dao.close();
}
}

// Método para preparar os dados para retornar à atividade principal
public void alterarInserir(View v) {
    String nome;
    double a1, a2, a3;
    // Pega os dados preenchidos na tela,
    // para inclusão ou alteração
    nome = ed1.getText().toString();
    a1 = Double.parseDouble(ed2.getText().toString());
    a2 = Double.parseDouble(ed3.getText().toString());
    a3 = Double.parseDouble(ed4.getText().toString());
    // Cria o objeto de acesso ao banco
    Disciplina _ DAO dao = new Disciplina _ DAO(this);
    // Abre o banco
    dao.open();
}

```

```

// Determina a ação
if (acao == -1) { // Ação de inserção
    // Realiza a inclusão da disciplina na tabela
    dao.inserir(nome, a1, a2, a3);
}
else{ // Ação de alteração
    // Realiza a alteração do registro correspondente na tabela
    dao.alterar(id, nome, a1, a2, a3);
}
// Libera o recurso de acesso ao banco
dao.close();
// Encerra a atividade, retornando à atividade principal
finish();
}

// Método para a exclusão de um registro de acordo com o id da disciplina
public void excluir(View v) {
    // Confere se é uma ação de alteração ou exclusão
    if (acao == 0) {
        // Cria o objeto de acesso ao banco
        Disciplina _ DAO dao = new Disciplina _ DAO(this);
        // Abre o banco
        dao.open();
        // Realiza a exclusão do registro por meio do id
        dao.apagar(id);
        // Libera o recurso de acesso ao banco
        dao.close();
    }
    // Encerra a atividade, retornando à atividade principal
    finish();
}

public void voltar(View v) {
    // Encerra a atividade, retornando à atividade principal
    finish();
}
}

```

Notas:

b.1. Não se esqueça de confirmar a programação dos métodos *OnClick* dos dois botões na atividade secundária.

b.2. A tela é configurada de acordo com os dados enviados da atividade principal:

- Caso seja uma alteração ou exclusão, a tela é preenchida com os dados do registro da tabela – e a tela possui configuração de alteração ou exclusão.
- Caso seja uma ação de inclusão, a tela é configurada para essa ação e é preenchida com os dados-padrão.

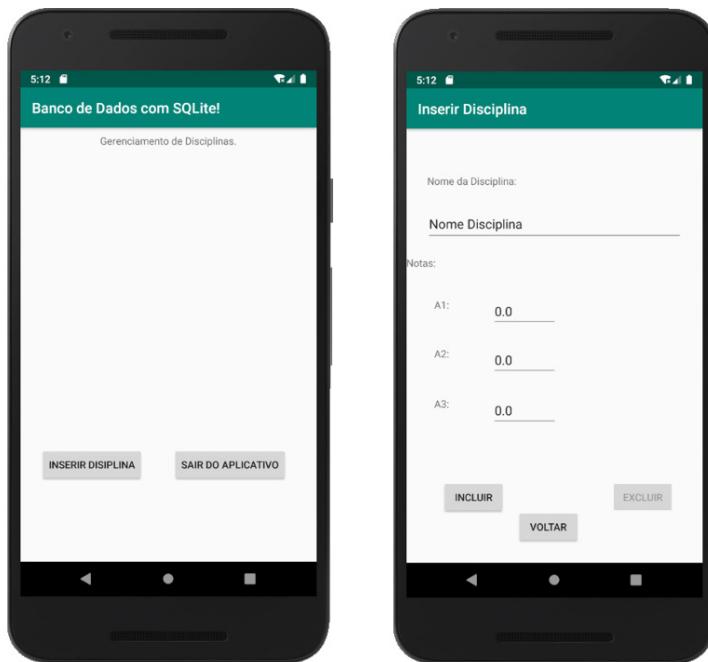
b.3. O botão EXCLUIR fará a ação de exclusão se for uma ação de alteração ou exclusão.

b.4. O primeiro botão pode ter os textos INCLUIR ou ALTERAR, realizando a ação correta de acordo com a ação (-1, inclusão, ou 0, alteração ou exclusão) — a programação desse botão possui uma dependência da ação que poderá ser realizada.

b.5. Não haverá retorno de dados para a atividade principal porque o método *onResume* fará a atualização da lista na atividade principal.

Realize os seus testes e compare com os resultados obtidos!

Testes realizados



Tela principal sem dados

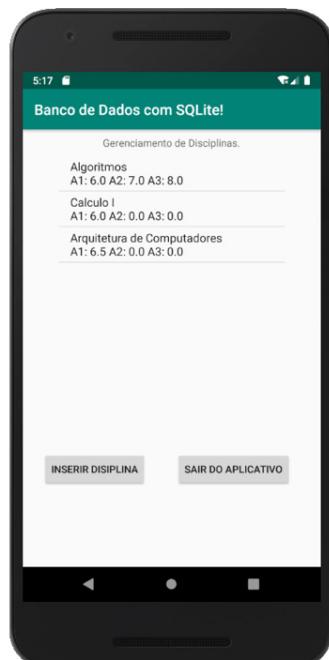
Tela de inclusão com dados preenchidos de forma padrão



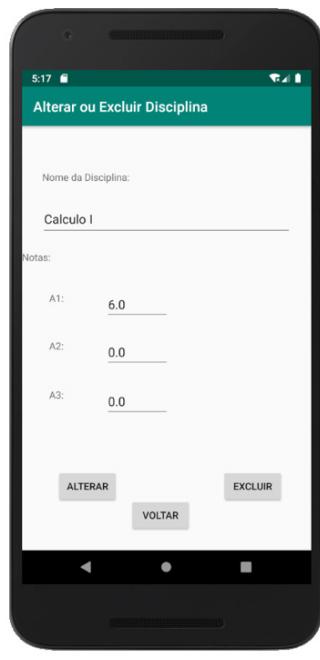
Preenchimento dos dados e inclusão



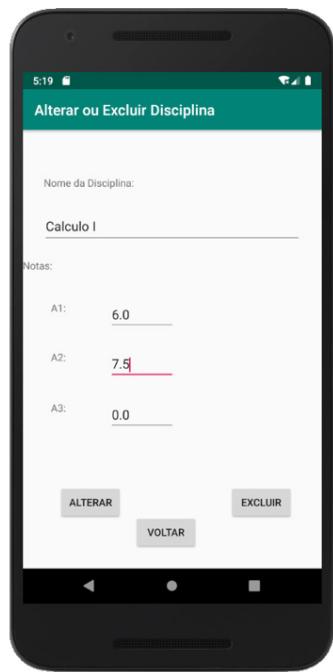
Tela principal com a nova disciplina



O aplicativo inicia com algumas disciplinas



Escolhida a disciplina de Cálculo I para alteração



Realize a alteração da
nota de A2



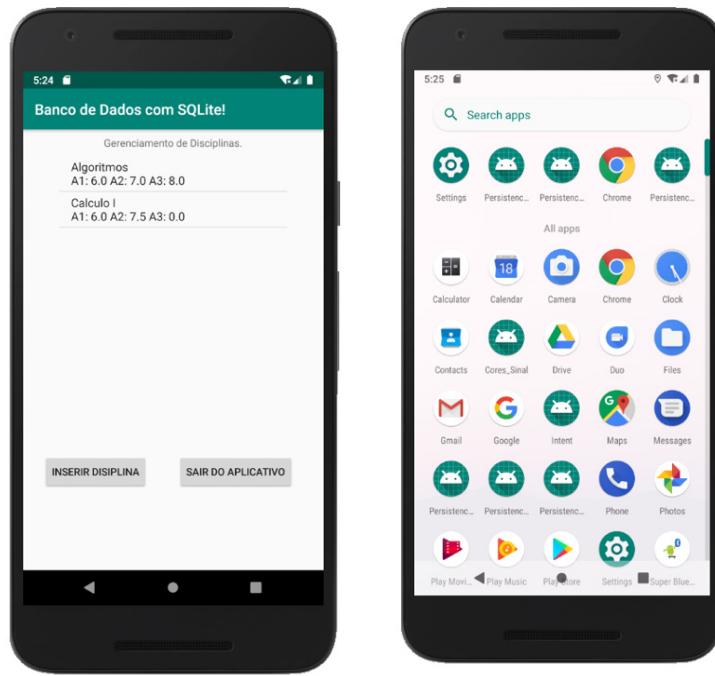
Retorno à tela principal com
a disciplina alterada



Selecione a disciplina de
Arquitetura de computadores
para exclusão



Confirme os dados e
clique em *EXCLUIR* para
excluir a disciplina



A tela principal está atualizada sem a disciplina de Arquitetura de computadores

Você pode sair do aplicativo e, quando voltar, todos os dados estarão disponíveis

Com esse exemplo prático, você aprende:

- Como trabalhar de forma prática com banco de dados.
- A linguagem Java no Android.



MEDIATECA

Veja na mediateca da Unidade 4 o conteúdo complementar selecionado pelo professor sobre **guardar dados com o SQLite**.



MEDIATECA

Veja na midiateca da Unidade 4 o conteúdo complementar selecionado pelo professor sobre **como desenvolver o aplicativo WeatherViewer**.



Saiba mais

Recursos avançados | Programação distribuída com acesso remoto a banco de dados

O desenvolvimento de aplicações para dispositivos móveis se assemelha ao desenvolvimento para a web quanto ao uso de banco de dados remotos.

Normalmente temos:

- Um serviço web (servidor de aplicações) para acomodar a camada de aplicação.
- Um servidor de banco de dados para a camada de persistência.

Esses dois serviços não necessitam de configurações exclusivas para aplicações móveis, sendo suficientes para atender às necessidades de acesso remoto. Isso ocorre porque os aplicativos móveis utilizam APIs de acesso à web por meio de classes de acesso à internet. É necessário incluir no arquivo de manifesto o acesso à internet e utilizar classes de acesso à internet com definição de URLs e passagem de dados para a troca de mensagens entre o aplicativo e o servidor de aplicação. Esses dados podem ser enviados e recebidos através de estruturas de dados empacotadas no formato XML, mas o mais comum é o empacotamento com uso de JSON (formato para a troca de informações entre diferentes sistemas).

Dessa forma, os dados são trocados de forma mais segura.

A linguagem Java e o PHP, por exemplo, dão suporte ao JSON. Você deve, então, criar sua aplicação baseada na troca de mensagens via web por meio do JSON, com um servidor de aplicação remoto, que terá uma camada de persistência com acesso a um servidor de banco de dados.



MEDIATECA

Veja na midiateca da Unidade 4 o conteúdo complementar selecionado pelo professor sobre **como realizar acesso remoto a uma base de dados PostgreSQL ou MySQL**.



NA PRÁTICA

Existem muitos aplicativos que trabalham com a persistência de dados. Muitos deles solicitam os dados de acesso do usuário uma vez e armazenam as preferências para facilitar o acesso do usuário. Em muitos casos, além das preferências, o aplicativo faz o acesso à internet e se conecta a serviços para a troca de mensagens através de bases de dados remotas. Analise os principais aplicativos de redes sociais e de mensagens que você usa. Observe que, após o primeiro acesso, no qual você apresenta os seus dados de acesso, eles não são mais solicitados pelo aplicativo, e é realizada uma atualização das mensagens ou compartilhamentos diretamente pelo aplicativo após realizar o acesso ao seu servidor de aplicação.

Resumo da Unidade 4

Nesta unidade, você pôde compreender como trabalhar com a persistência de dados no Android. Você aprendeu de forma prática como trabalhar com armazenamento de preferências dos usuários nos arquivos de *SharedPreferences*, assim como a trabalhar com arquivos segundo as APIs da linguagem Java por meio do uso das classes de *streams*, com exemplo de uso das classes: *DataOutputStream* e *DataInputStream*. Por final, você teve a oportunidade de trabalhar com uma aplicação baseada na arquitetura em três camadas com o acesso ao banco de dados local *SQLite*.



CONCEITO

Persistência de dados no Android, com uso de arquivos de armazenamento de configurações do aplicativo, gravação e recuperação de dados armazenados em arquivos locais e uso de bases de dados locais (*SQLite*) para o armazenamento e recuperação de dados em tabelas.

Referências

ASCÊNCIO, A. F. G. **Fundamentos da programação de computadores**. São Paulo: Pearson Education do Brasil, 2008. Biblioteca Virtual.

DEITEL, P. J.; DEITEL, H. M.; DEITEL, A. **Android**: como programar. Porto Alegre: Bookman, 2015. Minha Biblioteca.

_____ ; _____; WALD, A. **Android 6 para programadores**: uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2016. Minha Biblioteca.

LEAL, N. G. V. **Dominando o Android**: do básico ao avançado. São Paulo: Novatec, 2015.

MONK, S. **Projetos com Arduino e Android**: use seu smartphone ou tablet para controlar o Arduino. Porto Alegre: Bookman, 2014. Minha Biblioteca.

