Aula 9

- Organizar os códigos em subpastas para manter organização

- Utilizar domínio para evitar conflitos

- Tudo que for atividade utilizar a pasta Activities

- Primeiro estado da Atividade, onCreate(), utilizando super.onCreate

- Para acessar arqivos resource, utilizar a classe R, é uma classe gerada toda vez que algo é jogado para dentro da pasta

- Padrão seguido, atividade Main carrega a tela Main

- Layout é representado por um arquivo XML

- View: pode ser componente, Scroll View ou Card View, dentro do Scroll View e Card View, pode colocar qualquer coisa que seja View ou View Group

- View Group: pode armazenar uma ou mais views ou uma ou mais view group, tudo que for View Group será Layout

- O LinearLayout terá n filhos

- Toda View e View Group tem que ter dois atributos obrigatórios, layout_width e layout_heigh

- Necessidade de px depende do tamanho de cada dispositivo

- TextView para colocar texto estático

- match_parent, expandir de acordo com o tamanho do pai

- wrap_content, irá se acomodar pelo tamanho do filho

- Pode utilizar um módulo string para cada idioma, por exemplo

- marginStart a margem fica na esquerda da página

- marginEnd a margem fica pra direita da página

- Colocar espaçamento para deixar um bom espaço para os componenetes, sem deixar o espaço desagradável

- imeOptions são as configurações do teclado, como cada botão vai se comportar

- Para alterar o teclado utilizar o inputType, caso não seja declarado, o teclado funcionará de maneira normal

- Para empilhar vários campos utilizar LinearLayout

- Atributo para exibir ou esconder algo: visibility

- SVG a partir do Android 8

- Aceita png, jpeg, gif

- Definir uma restrição na vertical e outra na horizontal

Aula 10 e 11

- Se atentar na organização dentro da hierarquia

- @font/ -> Para utilizar uma fonte baixada fora do Android Studio

- Para acessar os compontentes de tela, pode usar os móbulos, são eles

    1. Tradicional: findViewById (id)

    2. DataBinding: existem bibliotecas que fazem o trabalho de declaração automática dos componentes

    - Vantagens: economizar digitação de código, minimiza erros

    - Dagger: terceiros

    - DataBinding: incluído no Android

- Config do DataBinding

    1. Habilitar no gradle

    2. Modificar os arquivos de layout

    3. Substituir a chamada do SetContentView

- Arquivo de layout sempre terá na raiz um filho que será uma View ou ViewGroup

- Tratamento de eventos: existem diferentes formas de cadastrar um evento em um compontente

- Sempre que termina em OnListener é um evento, todas as funções que estarão ali dentro serão executadas

- Os eventos são disparados pelos componenetes após o usuário interagir

- As funções para cadastro de evento começam com o prefixo <on> e terminam com <Listener>, é uma regra geral

1. Via função lambda

- Utilizado quando o evento não é reaproveitado e quando são poucas linhas de código

```
mBinding.componentes.setOnClickListener {
    Toast.makeText(this, getText(R.string.componentes_btn_ola), Toast.LENGTH_SHORT).show()
}
```

2. Via método

- Utilizado quando o evento tem a necessidade de ser reaproveitado e podem ser funções complexas

```
mBinding.componenteBtnClique.setOnClickListener()
```

Aula 12

- Fragmentos Android é uma atividade que se acopla com uma tela, numa atividade pode colocar vários fragmentos

- Atividade e fragmento possuem ciclos de vida

- O fragmento é criado depois da ativdade

- A maneira que a tela é criada para atividade é a mesma para o fragmento

- Primeiro destruído o fragmento e após disso, a tela

- O fragmento pode ser reaproveitado

- Valor absoluto só é utilizado para imagem

- Res -> New -> AndroidResource Directory -> Menu

- Para construir um fragmento, usamos o padrão builder

- Permite que os fragmentos sejam criados de maneira uniforme

- Padroniza a passagem de parametros

- Companion Object: utilizado para criar função estática para a construção dos fragmentos

```
companion objetc {
   @JvmStatic
   fun newInstancte () = Tela1Fragment()
}
```

- Criar fragmento

```
mTela1Fragment = Tela1Fragment.newInstance()
```

- Carregar o fragmento do container vazio

```
trocarFragmento(mTela1Fragment)
```

- Cadastrar os eventos da bottom navigation para trocar de fragmento

```
mBinding.fragmentoBtmNavigation.setOnItemSelectedListener(::onSelectedBottomnavigationIt
em)
```

Aula 14

- Se atentar na organização dentro da hierarquia

- @font/ -> Para utilizar uma fonte baixada fora do Android Studio

- Para acessar os compontentes de tela, pode usar os móbulos, são eles

    1. Tradicional: findViewById (id)

    2. DataBinding: existem bibliotecas que fazem o trabalho de declaração automática dos componentes

    - Vantagens: economizar digitação de código, minimiza erros

    - Dagger: terceiros

- DataBinding: incluído no Android

- Config do DataBinding

1. Habilitar no gradle
2. Modificar os arquivos de layout
3. Substituir a chamada do SetContentView

- Arquivo de layout sempre terá na raiz um filho que será uma View ou ViewGroup

- Tratamento de eventos: existem diferentes formas de cadastrar um evento em um compontente

- Sempre que termina em OnListener é um evento, todas as funções que estarão ali dentro serão executadas

- Os eventos são disparados pelos componenetes após o usuário interagir

- As funções para cadastro de evento começam com o prefixo <on> e terminam com <Listener>, é uma regra geral

1. Via função lambda

- Utilizado quando o evento não é reaproveitado e quando são poucas linhas de código

```
mBinding.componentes.setOnClickListener {
    Toast.makeText(this, getText(R.string.componentes_btn_ola), Toast.LENGTH_SHORT).show()
}
```

2. Via método

- Utilizado quando o evento tem a necessidade de ser reaproveitado e podem ser funções complexas

```
mBinding.componenteBtnClique.setOnClickListener()
```

Aula 15

- Servidor HTTP:

1. Apache
2. Tomcat
3. IIS

4. Nginx

- Banco de dados relacionais

1. MySQL
2. PostgreSQL
3. Oracle

- Banco de dados noSQL

1. Firebase
2. MongoDB

XML

- Representação de documentos semi-estruturados

- Composto por elementos e atributos

REST

- Usado para criação de web-services

- Não mantém o estado

- Cada consulta é como se fosse a primeira

- Princípios

1. Cliente-Sevidor
2. Staless
3. Cacheable
4. Interface uniforme -> Padronizada
5. Sistema em camadas -> Organização do código
6. Código sob demanda

CLIENTE

- Fornece interação com o usuário

- Cliente magro e gordo

- Magro (apps de banco, claro, vivo, fgts)

1. Uma única camada
2. Não apresenta código personalizado
3. Dependente do servidor
4. Utiliza navegador web

- Gordo (whatsapp é um exemplo)

1. Possue até três camadas

- Apresentação
- Negócios
- Acesso à dados

2. Comunicação entre cliente - servidor é baixa

APLICAÇÕES

- Nativo

1. Desenvolvido para a plataforma
2. Alto desempenho
3. Maior esforço de desenvolvimento

- Compile-to-native (React Native, Native Script, Flutter e Xamarin)

1. Ambiente tercerizado
2. Aplicação para diversas plataformas
3. Dificuldade de domínio de framework

- Híbrida (PhoneGap, Cordova, Sencha e Ionic)

1. Fácil entendimento para desenvolvimento web
2. Executa em webview

- Progressive Web App (PWA) HTML/CSS, React, Angular e Vue

1. Fácil desenvolvimento
2. Não é uma aplicação real
3. Executa em navegador web

4. Não acessa recursos do dispositivo

PROTOCOLOS

- Http

1. Protocolo para troca de mensagens web

- REST

1. Entidades = URL
2. Ações = Comandos HTTP (Get, post, delete, put)
3. Sempre no plural

BOAS PRÁTICAS

1. Nomes no plural
2. Evitar palavras abstratas

Json

1. JavaScript Objetc Notation
2. Compacto
3. Tipos explícitos

Blob

1. Raw
2. URL
3. Encoding Base64

CÓDIGOS DE ERROS

1. Utilizar códigos de erro http

- 1XX- Informativas
- 2XX - Sucesso
- 3XX - Redirecionamentos
- 4XX - Erro na resposta do cliente
- 5XX - Erro no servidor