Aluno: João Vitor da Silva

Resumo Final para a Prova 23/06

No desenvolvimento de apps Android, seguir boas práticas é essencial: organizar o código, adicionar comentários, usar recursos eficientemente, garantir a responsividade da interface, realizar testes, garantir segurança e documentar o projeto.

A classe View representa elementos de interface, como botões e textos, e pode ser configurada com propriedades de tamanho, posição, cor e transparência, além de responder a eventos do usuário.

Componentes comuns incluem ScrollView para rolar conteúdo maior que a tela, CardView para exibir informações organizadas e orientação para definir a direção do app (horizontal ou vertical).

Atributos comuns são layout_width e layout_height para definir largura e altura, background para cor de fundo, text para texto exibido e src para imagem em ImageView. O layout organiza elementos na tela, usando contêineres como LinearLayout, RelativeLayout, ConstraintLayout e FrameLayout.

Outros termos importantes são atividades, constantes, fragmentos, visualizações, intenções, recursos, manifesto e Gradle. O método onCreate() é essencial e chamado ao criar a tela. A classe 'R' contém referências a recursos e ViewGroup armazena views para layouts. Medição em DP garante consistência entre dispositivos.

Match_Parent ocupa todo o espaço do pai, enquanto Wrap_Content ajusta ao conteúdo. É recomendado usar nomes genéricos para dimensões e considerar categorias de tamanho de tela.

TextView exibe textos estáticos, e é possível acessar componentes por identificadores. É comum prefixar entidades com "Main_" para facilitar acesso.

- Cadastro de atividades no androidManifest é importante para o funcionamento do app.
- Padrão de nome para recursos: usar título da tela para facilitar organização
- Personalização de botões: alterar cor com backgroundtint.
- Identificação e resolução de erros: usar Alt+Enter para detalhes.
- Uso do databinding: conexão entre interface e lógica.
- Configuração do databinding: habilitar no gradle e modificar layouts.
- Lateinit: declarar variáveis não nulas.
- Tratamento de eventos: setOnClickListener() com funções lambda ou métodos.

- Estruturas de decisão: if else, if else if, when. Executar blocos de código com base em condições.
- Estruturas de repetição: while, do...while, for. Repetir código enquanto condição for atendida.
- Definição de funções: blocos de código reutilizáveis. Criação, chamada e passagem de parâmetros.
- Programação orientada a objetos (POO): organização em objetos com atributos e métodos. Classes, relacionamentos (associação, agregação, composição, herança).
- Três pilares da POO: encapsulamento, herança, polimorfismo.
- Paradigmas de programação: imperativo (estruturado, procedural, orientado a objetos) e declarativo (funcional, lógico).

- Servidores HTTP: Apache, Tomcat, Nginx e IIS. Responsáveis por receber requisições e fornecer respostas.
- Banco de Dados Relacional: MySQL, SQL Server, Postgres. Armazenamento de dados estruturados e relacionais.
- Banco de Dados NoSQL: Flexíveis e escaláveis para dados não estruturados.
- Comunicação do servidor de aplicação: Pode ser conjunta ou separada entre dispositivo e servidor.
- Arquitetura Orientada a Serviço: Comunicação por meio de serviços específicos.
- SOAP (Simple Object Access Protocol): Protocolo que utiliza XML para comunicação.
- XML (Extensible Markup Language): Representação de documento semi-estruturado.
- REST (Representational State Transfer): Uso dos métodos HTTP para interação.
- Clientes magros: Dependentes do servidor, sem código de aplicação personalizado.
- Clientes gordos: Interface, lógica de negócio e acesso a dados. Baixa comunicação com o servidor.
- Nativo: Alto desempenho, utiliza ecossistema da plataforma.
- Compile-to-native: Compilação para diversas plataformas usando frameworks.
- Híbrida: Executam em webview, mais lentas que nativas.
- Progressive Web App: Aplicativos desenvolvidos para navegador, experiência semelhante a nativos.

- Volley: Biblioteca do Google para chamadas de API, evitando bloquear a interface do usuário.
- Atualização do SDK: Necessária para se manter atualizado com as funcionalidades e correções.
- Classe anônima: Criação de uma classe anônima para lidar com callbacks de API de uso ocasional.

- Dinâmica das APIs: Cada API possui sua própria dinâmica e métodos específicos.
- Componentes e classes: Os componentes da resposta da API devem ser mapeados para a classe correspondente.
- Callback: Mecanismo para receber chamadas de retorno com a resposta da API.
- Callback na API Service: Implementação do callback na classe de serviço da API.
- OnResponse: Método acionado na resposta bem-sucedida da API, permitindo manipular os dados retornados.
- OnFailed: Método acionado em caso de falha na resposta da API, possibilitando tratar erros e exibir mensagens apropriadas ao usuário.

- API: Acessa dados pela internet, obtidos de um servidor remoto através de requisições.
- Área de trabalho privada: Cada aplicativo possui sua própria área de armazenamento de arquivos, podendo optar por salvar na área pública. Arquivos locais podem ser em formatos como txt (csv, json, xml) ou dat (binário).
- SQLite: Banco de dados embutido que suporta uma grande quantidade de dados, facilitando a localização e as operações de inserção, atualização e exclusão. Requer mais código para utilizar, e a leitura e relacionamento entre tabelas podem ser mais lentos e complexos.
- Shared Preference: Forma de armazenamento de chave-valor para gravar configurações locais, preferências do usuário ou cache. As informações nas shared preferences são recriadas quando o aplicativo é reinstalado, e seu uso é fácil e não requer configurações adicionais.

Consuming API:

```
dependencies {
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
}

data class User(
    val id: Int,
    val name: String,
    val email: String
)

import retrofit2.Call
import retrofit2.http.GET

interface ApiService {
    @GET("users")
    fun getUsers(): Call<List<User>>
}

import android.os.Bundle
import android.util.Log
import androidx.appcompat.app.AppCompatActivity
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val retrofit = Retrofit.Builder()
            .baseUrl("https://api.example.com/") // Substitua pela URL base da sua API
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        val apiService = retrofit.create(ApiService::class.java)
        val call = apiService.getUsers()
```