# Ciência da Computação

## Android - Persistência de dados com o Room

## Aula 08

Professor: André Flores dos Santos

Santa Maria – RS 2025

# Introdução

**O Que é Room?**

•**Tópicos**:

- O **Room** é uma biblioteca de persistência de dados que faz parte do Jetpack do Android.
- Ele facilita a manipulação de dados locais em SQLite de uma maneira mais segura e eficiente.
- O Room abstrai muitos dos detalhes complexos de trabalhar diretamente com SQL, oferecendo uma interface mais amigável.

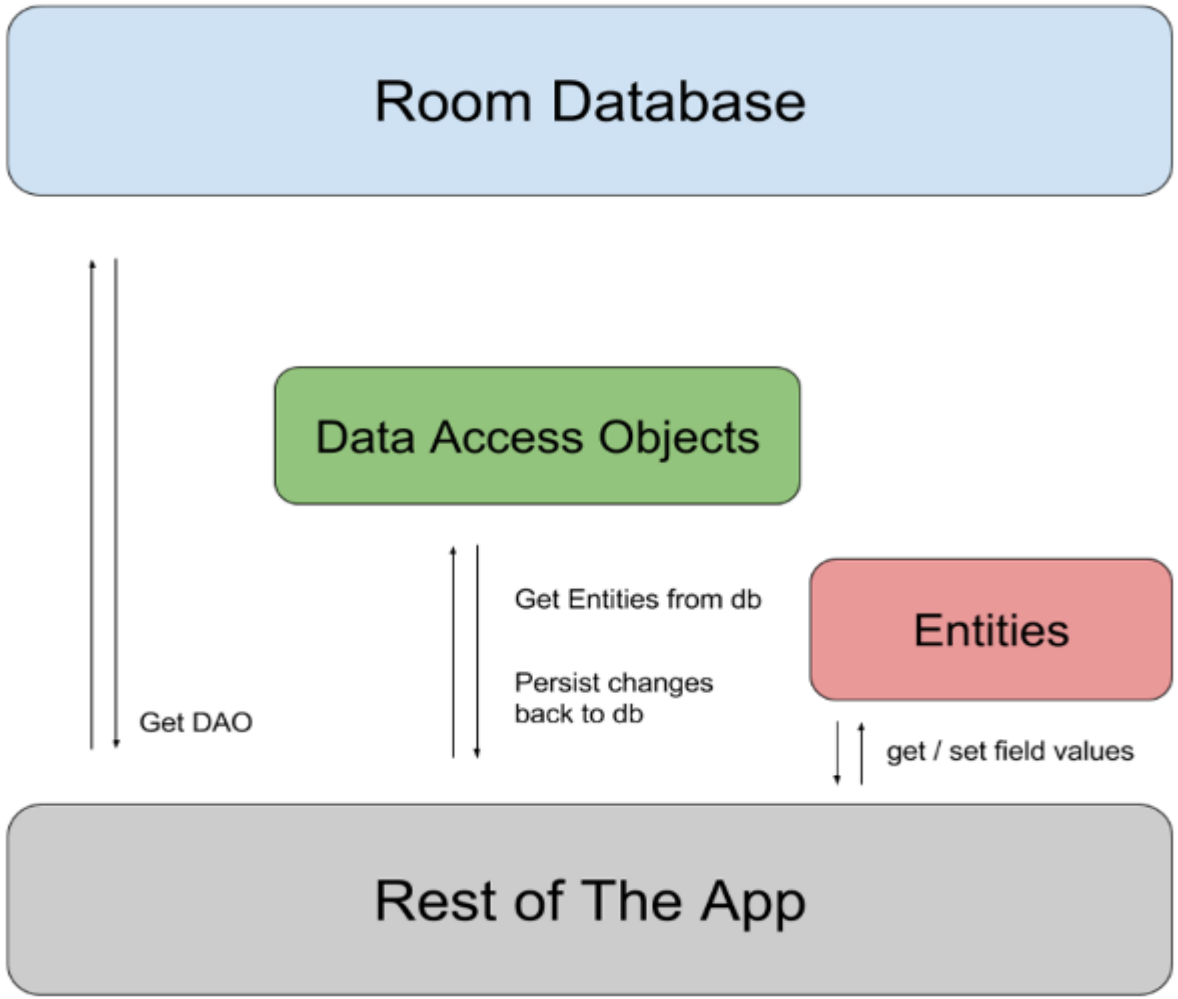
# Introdução

**Componentes do Room**

•**Tópicos**:

- **Entity**: Representa uma tabela no banco de dados.
- **DAO (Data Access Object)**: Interface que contém métodos para acessar o banco de dados.
- **Database**: Classe principal que mantém a conexão com o banco de dados e atua como o ponto central para o acesso aos dados persistidos.

# Introdução




**Salvar dados em um banco de dados local usando Room**

https://developer.android.com/training/data-storage/room?hl=pt-br

# Funcionamento

**Estrutura de uma Entity**

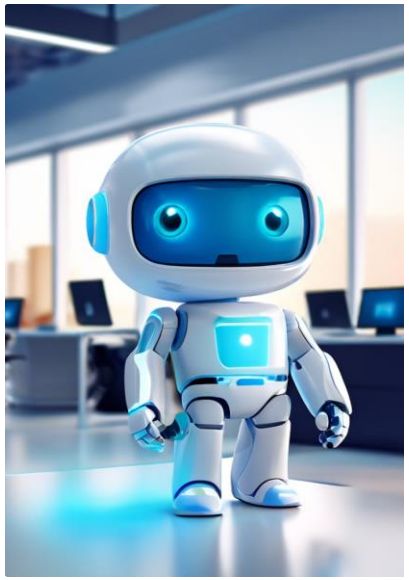
**•Código Exemplo**:

```
@Entity(tableName = "usuarios")
public class Usuario {
    @PrimaryKey(autoGenerate = true)
    private int id;

    @ColumnInfo(name = "nome")
    private String nome;

    @ColumnInfo(name = "email")
    private String email;

    // Getters e Setters...
}
```



Cada campo na classe representa uma coluna no banco de dados.
A anotação **@PrimaryKey** marca o campo que será a chave primária.
**@ColumnInfo** define detalhes das colunas (nome, tipo).

# Funcionamento

**Definindo o DAO**

**•Código Exemplo**:

```
@Dao
public interface UsuarioDao {
    @Insert
    void inserirUsuario(Usuario usuario);

    @Query("SELECT * FROM usuarios")
    List<Usuario> obterTodosUsuarios();
}
```

DAO define os métodos para interagir com o banco de dados.
@Insert: Inserção de dados.
@Query: Executa consultas personalizadas no banco de dados.

# Funcionamento

**Configurando o Database**

•**Código Exemplo**:

```
@Database(entities = {Usuario.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UsuarioDao usuarioDao();
}
```

@Database: Define a classe que será usada como ponto de acesso ao banco de dados (AppDatabase.java). Especifica as entidades (tabelas) e a versão do banco.

# Funcionamento

**Configurando o Database**

**•Código Exemplo (se houver várias entidades)**:

```
@Database(entities = {Usuario.class, Aluno.class, Produto.class,
Pedido.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract UsuarioDao usuarioDao();
    public abstract AlunoDao alunoDao();
    public abstract ProdutoDao produtoDao();
    public abstract PedidoDao pedidoDao();
}
```

Cada entidade deve ter seu próprio DAO, e você deve definir métodos para acessá-los na classe AppDatabase.

# Funcionamento

**Inicializando o Room no App**

**•Código Exemplo**:

```
// Inicializar o banco de dados Room
AppDatabase db = Room.databaseBuilder(getApplicationContext(),
        AppDatabase.class, "banco-de-dados")
        .allowMainThreadQueries() // Evitar em produção, preferir threads
assíncronas
        .build();

// Obter instâncias dos DAOs
UsuarioDao usuarioDao = db.usuarioDao();
AlunoDao alunoDao = db.alunoDao();
```

Room.databaseBuilder: Cria uma instância do banco de dados.
allowMainThreadQueries: Permite execução no thread principal (não recomendado em produção).

# Funcionamento

**Inicializando o Room no App (continuação)**

**•Código Exemplo**:

```
// Exemplo de inserção e obtenção de usuários
Usuario novoUsuario = new Usuario();
novoUsuario.setNome("João");
novoUsuario.setEmail("joao@example.com");
usuarioDao.inserirUsuario(novoUsuario);

// Exemplo de inserção e obtenção de alunos
Aluno novoAluno = new Aluno();
novoAluno.setNome("Maria");
novoAluno.setCpf("12345678900");
novoAluno.setTelefone("999999999");
alunoDao.inserir(novoAluno);
```

# Funcionamento

**Inicializando o Room no App (continuação)**

•**Código Exemplo**:

```
// Obter todos os usuários
List<Usuario> listaUsuarios = usuarioDao.obterTodosUsuarios();
for (Usuario usuario : listaUsuarios) {
    Log.d("Usuario", "Nome: " + usuario.getNome() + ", Email: " +
usuario.getEmail());
}

// Obter todos os alunos
List<Aluno> listaAlunos = alunoDao.obterTodos();
for (Aluno aluno : listaAlunos) {
    Log.d("Aluno", "Nome: " + aluno.getNome() + ", CPF: " +
aluno.getCpf());
}
```

# Funcionamento

- **Conclusão**

❑ Tópicos:

- O Room simplifica o uso do banco de dados SQLite em Android.
- Entidades, DAOs e Database são os blocos principais.
- A implementação prática é direta e traz vantagens como maior segurança e menos propensão a erros.
- Próximo passo: Aplicar esses conceitos ao nosso aplicativo de cadastro de usuários.

# Alteração do nosso projeto para usar o Room

**Passos para adicionar as dependências do Room**
Abrir o arquivo build.gradle (nível do módulo/app):

Esse arquivo geralmente está localizado em: app/build.gradle.
Adicionar as dependências do Room:

Dentro do bloco dependências, você deve incluir as dependências do Room necessárias para trabalhar com a persistência de dados.
Logo após faça:
File>Sync Project with Gradle Files

```
dependencies {
    // Room Database
    implementation 'androidx.room:room-runtime:2.5.2'

    // Annotation Processor para o Room (obrigatório)
    annotationProcessor 'androidx.room:room-compiler:2.5.2'
```

# Alteração do nosso projeto para usar o Room

- **Classe Aluno**

```
@Entity(tableName = "aluno")
public class Aluno implements Serializable {

    @PrimaryKey(autoGenerate = true)
    private Integer id;

    @ColumnInfo(name = "nome")
    private String nome;

    @ColumnInfo(name = "cpf")
    private String cpf;

    @ColumnInfo(name = "telefone")
    private String telefone;

    @ColumnInfo(name = "foto")
    private byte[] fotoBytes;

    // Getters e Setters
    public Integer getId() {
        return id;
    }
//.....
}
```

# Alteração do nosso projeto para usar o Room

- Para migrar nossa classe AlunoDao para o Room, muitas das operações manuais, como manipulação de Cursor, SQLiteDatabase, e ContentValues, serão abstraídas e simplificadas por meio das anotações e métodos do Room. O Room torna as interações com o banco de dados mais seguras e simplifica o código de DAO (Data Access Object).

- **Passo 1: Criando o DAO com Room**
  - Em vez de usar SQLiteDatabase diretamente, o Room define um DAO (Data Access Object) para cada entidade (neste caso, a classe Aluno). O DAO é uma interface onde você define métodos que vão realizar operações de banco de dados.

# Alteração do nosso projeto para usar o Room

```java
@Dao
public interface AlunoDao {

    // Inserir um aluno no banco de dados
    @Insert
    long inserir(Aluno aluno);

    // Atualizar os dados de um aluno
    @Update
    void atualizar(Aluno aluno);

    // Obter todos os alunos do banco de dados
    @Query("SELECT * FROM alunos")
    List<Aluno> obterTodos();

    // Excluir um aluno do banco de dados
    @Delete
    void excluir(Aluno aluno);

    // Verificar se o CPF já existe no banco de dados
    @Query("SELECT COUNT(*) FROM alunos WHERE cpf = :cpf")
    int cpfExistente(String cpf);
}
```

```
//Os métodos de validação de CPF e telefone deverão ser
colocados numa classe chamada 'AlunoValidator.java' !!
```

**Criar uma interface AlunoDao**
**Explicação dos Métodos:**
@Insert: Usado para inserir um novo aluno no banco de dados. Ele pode retornar o ID do novo aluno inserido, assim como você já fazia no método inserir().

@Update: Atualiza um aluno existente no banco de dados, substituindo a lógica manual anterior.

@Delete: Remove um aluno do banco de dados, substituindo o método excluir().

@Query: Define consultas SQL customizadas. No caso de obterTodos(), usamos SELECT * FROM alunos para buscar todos os registros da tabela.

Também usamos @Query no método cpfExistente() para verificar se um CPF já está cadastrado e vai realizar operações de banco de dados.

# Alteração do nosso projeto para usar o Room

Criar a Classe ‘AlunoValidator.java’ e colocar os métodos de validações dentro, para seguir as boas práticas de programação.

Após fazer as alterações na ‘Main’ onde estes métodos eram chamados, com a nova classe criada.

Os métodos de validação de CPF e telefone deverão ser colocados nessa classe ‘AlunoValidator.java’ !!

# Alteração do nosso projeto para usar o Room

**Criar a Classe 'AppDatabase.java'**

Agora, precisamos criar uma classe que representará o banco de dados no Room. Ela será responsável por fornecer a instância de AlunoDao para que possamos realizar operações.

@Database: Define que a classe AppDatabase será o banco de dados do Room. Ela contém uma lista das entidades (Aluno.class) e a versão do banco de dados.

getInstance(): Fornece uma instância do banco de dados, criando uma única instância (singleton) para evitar múltiplas conexões abertas.

```
/ Define a classe como um banco de dados Room
// "entities" especifica as tabelas que pertencem ao banco
// "version" é usada para controle de atualização do banco de dados
@Database(entities = {Aluno.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {

    // Método abstrato que retorna o DAO (Data Access Object) para acessar os dados da tabela "aluno"
    public abstract AlunoDaoRoom alunoDaoRoom();

    // Instância única do banco de dados (Singleton) para evitar múltiplas conexões
    private static AppDatabase INSTANCE;

    // Método para obter a instância do banco de dados
    // "synchronized" garante que apenas uma instância seja criada, mesmo em ambientes com múltiplas threads
    public static synchronized AppDatabase getInstance(Context context) {
        if (INSTANCE == null) { // Se o banco de dados ainda não foi criado, cria uma nova instância
            INSTANCE = Room.databaseBuilder(
                    context.getApplicationContext(), // Usa o contexto da aplicação para evitar vazamento de memória
                    AppDatabase.class, // Define esta classe como o banco de dados Room
                    "banco-de-dados" // Nome do arquivo do banco de dados armazenado no dispositivo
                ).allowMainThreadQueries() // Permite rodar consultas no thread principal (não recomendado em produção)
                .build(); // Cria e retorna a instância do banco de dados
        }
        return INSTANCE; // Retorna a instância única do banco de dados
    }
}
```

# Alteração do nosso projeto para usar o Room

Passo 3: Utilizando o Room no Código
Agora que temos o AlunoDao e o AppDatabase definidos, você pode usá-los facilmente em suas atividades ou fragments para manipular os dados.

Exemplo de como usar o DAO para inserir e recuperar alunos:

```
// Inicializando o banco de dados e o DAO
AppDatabase db = AppDatabase.getInstance(context);
AlunoDao alunoDao = db.alunoDao();

// Inserir um novo aluno
Aluno novoAluno = new Aluno();
novoAluno.setNome("Maria");
novoAluno.setCpf("12345678900");
novoAluno.setTelefone("999999999");
novoAluno.setFotoBytes(null); // Pode adicionar foto se tiver

long id = alunoDao.inserir(novoAluno); // Retorna o ID do aluno inserido

// Recuperar todos os alunos
List<Aluno> alunos = alunoDao.obterTodos();
for (Aluno aluno : alunos) {
    Log.d("Aluno", "Nome: " + aluno.getNome() + ", CPF: " + aluno.getCpf());
}
```

# Alteração do nosso projeto para usar o Room

Devemos declarar a nossa interface AlunoDao que eu chamei de 'AlunoDaoRoom' na 'main activity.java' e depois pegar a instancia do 'AppDataBase' dentro do método 'onCreate'

```
4 usages
public class MainActivity extends AppCompatActivity {

    //campos do EditText
    3 usages
    private EditText nome;
    3 usages
    private EditText cpf;
    3 usages
    private EditText telefone;

    //private AlunoDao dao;
    6 usages
    private AlunoDaoRoom alunoDaoRoom;
```

# Alteração do nosso projeto para usar o Room

Devemos declarar a nossa interface AlunoDao que eu chamei de 'AlunoDaoRoom' na 'main activity.java' e depois pegar a instancia do 'AppDataBase' dentro do método 'onCreate'

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Chama o método onCreate() da classe pai (AppCompatActivity),
    // que configura aspectos essenciais da Activity, como a criação da janela,
    // a restauração do estado salvo e outras inicializações necessárias do ciclo de vida.

    setContentView(R.layout.activity_main);
    // Define qual arquivo de layout XML será usado para esta Activity,
    // ou seja, infla o layout 'activity_listar_alunos.xml' e o torna a interface exibida na tela.

    //Vinculando os campos do layout com as variáveis do Java
    nome = findViewById(R.id.editNome);
    cpf = findViewById(R.id.editCPF);
    telefone = findViewById(R.id.editTelefone);

    //dao = new AlunoDao(this);
    alunoDaoRoom = AppDatabase.getInstance( context: this).alunoDaoRoom();
```

# Alteração do nosso projeto para usar o Room

Devemos atualizar/refatorar o ‘ListarAlunoActivity.java’ se criamos um novo ‘alunoDao’ e testar as adaptações.

Agora já podemos utilizar @anotações para gerenciar o banco de dados através da Room.

# Testar!!

Fazer a adaptação do seu projeto para utilizar a ‘Room’ e entregar um versão funcional na atividade da aula de hoje.

# Referências:


LECHETA, Ricardo R. Google Android: aprenda a criar aplicações para dispositivos móveis com o android SDK. São
Paulo: Novatec, 2009.

TALUKDER, Asoke; YAVAGAL, Roopa. Mobile computing. New Delhi - India: McGraw-Hill, 2007.

SILVA, D. Desenvolvimento para dispositivos móveis. Pearson, 2017.

ANDROID. Android Developers. Disponível em https://developer.android.com. Acesso em agosto de 20018. 2018.

MIKKONEN, T. Programming mobile devices: an introduction for practitioners. Chichester,
England: Wiley, 2007.

ROGERS, Rick et al. Desenvolvimento de aplicações Android. O'Reilly: Novatec, 2009.

SILVA, D. Arquitetura para computação móvel. Pearson, 2018.

YAVAGAL, Roopa R.; TALUKDER, Asoke K. Mobile computing: technology, applications, and service creation. New
York, NY: McGraw-Hill, 2007.

# Thank you for your attention!!


Contato:
Email: **andre.flores@ufn.edu.br**