Explicação das Camadas e do Processo de Persistência

1. Camada **View** (Apresentação/UI)

Esta é a camada que o usuário vê e com a qual interage.

- **Arquivos Principais:** `CriaPersonagemScreen.kt` e outros componentes (`RacaClasseSelector`, etc.).
- **Responsabilidade:**
  - Exibir o estado atual dos dados (atributos, raça, classe, valores disponíveis).
  - Capturar a **interação do usuário** (cliques em botões, seleção de chips, digitação de nome).
  - Observar o estado do `ViewModel` (usando `collectAsState()`) e se redesenhar automaticamente quando o estado muda.
- **No Fluxo de Persistência (Salvamento):**
  - O usuário preenche o nome no `OutlinedTextField` e clica no botão "Salvar Personagem no Banco (ROOM)".
  - O clique no botão chama a função correspondente no `ViewModel`: `viewModel.salvarPersonagem(nomePersonagem)`.

2. Camada **Controller / ViewModel** (Lógica de Apresentação e Estado)

Esta camada atua como a ponte entre a View e o Model. Em um aplicativo Android moderno, ela usa o `ViewModel` para sobreviver a mudanças de configuração (como rotação de tela) e o `AndroidViewModel` (como no seu projeto) quando precisa de acesso ao `Application Context`.

- **Arquivo Principal:** `CriaPersonagemViewModel.kt`.
- **Responsabilidade:**
  - Gerenciar o **estado da UI** (`StateFlows` como `_atributosFinais`, `_valoresDisponiveis`).
  - Conter a **lógica de apresentação** (ex: determinar se o botão "Salvar" deve estar habilitado (`valoresDisponiveis.isEmpty()`)).
  - Manipular a lógica do Model (chamar o `GeradorAtributos.gerarValoresSimulados()`).
  - Conectar-se à camada de persistência (o DAO do ROOM).
- **No Fluxo de Persistência (Salvamento):**
  - Recebe a chamada `salvarPersonagem(nomePersonagem)` da View.
  - Verifica a **lógica de validação** (se todos os atributos foram alocados).

- Mapeia os dados do estado do ViewModel (que estão no formato Model `Atributos`, `Raca`, `Classe`) para a **Entidade** exigida pelo ROOM (`PersonagemEntity`).
- Inicia uma **coroutine** (`viewModelScope.launch`) para executar a operação de banco de dados em *background*.
- Chama o método de inserção no DAO: `personagemDao.insert(personagemParaSalvar)`.
- Atualiza um estado de *feedback* (`_mensagemUsuario`) para notificar a View sobre o sucesso ou falha.

3. Camada **Model** (Dados e Lógica de Negócio)

Esta é a camada mais interna e independente, que define as regras do jogo e a estrutura dos dados. É aqui que o ROOM se manifesta como a subcamada de persistência.

3A. Lógica de Negócio

- **Arquivos Principais:** `Atributos.kt`, `Raca.kt`, `Classe.kt`, `GeradorAtributos.kt`.
- **Responsabilidade:** Definir a estrutura dos dados (as `data classes`) e as regras do jogo (como os atributos são gerados).

3B. Persistência (ROOM - Subcamada do Model)

Esta subcamada lida exclusivamente com o armazenamento e recuperação de dados.

Componente ROOM	Arquivo	Responsabilidade no Fluxo
Entidade	`PersonagemEntity.kt`	Define a **estrutura da tabela** no banco de dados SQLite. É a "imagem" do personagem no disco.
DAO (Data Access Object)	`PersonagemDao.kt`	Define a **interface de acesso**. Especifica os métodos de CRUD (`@Insert`, `@Query`, etc.) para interagir com o SQLite.
Database	`AppDatabase.kt`	É o **ponto de entrada** e a configuração do banco de dados (nome do arquivo, entidades, versão).