Parte 2 - Arquitetura: App, Data, Domínio

Status	Lendo - In progress

Como essas camadas se conectam?

Imagine que você está montando um **processo de saque em uma conta bancária**. O usuário da aplicação quer sacar dinheiro, mas você precisa garantir que ele tenha saldo suficiente para fazer isso. Essa lógica de verificar o saldo pertence à **Camada de Domínio**.

Entender a Arquitetura

Antes de começar, você precisa entender como a arquitetura do sistema influencia onde e como as exceções devem ser tratadas.

- **Domain**: É o núcleo do sistema, onde as **regras de negócio** e **lógica principal** residem.
- **App** (Application): Esta camada coordena as operações do sistema, orquestra fluxos e usa os serviços da camada de domínio.
- **Data**: Responsável por **acesso ao banco de dados**, persistência e leitura de dados.

Identifique Onde o Problema Surge

No caso do enunciado, o problema ocorre quando um **registro não está vinculado a um extrato válido**. Esse é claramente um **erro de negócio**. Por isso, esse problema pertence à **camada de domínio** (Domain), já que envolve as regras sobre como os registros e extratos devem ser associados.

> Sempre que você tiver um erro que diz respeito a **regras de negócio**, o lugar certo para tratá-lo é no **Domain**, porque é lá que as regras do sistema são definidas.

> **Pergunte-se**: “Esse erro é algo relacionado à interação com o banco de dados ou é algo mais relacionado às regras do sistema?”

Resposta: "Está relacionado às regras de negócio." Portanto, o **Domain** é onde a lógica da exceção será definida.

Se for outra regra, então identifique!

Dividir a Solução em Atividades

Atividade 1: Criar a Exceção Personalizada no Domínio

Passo 1.1: Criar a exceção na camada **Domain**.

O erro sobre o registro sem extrato pertence à regra de negócio, então, a exceção deve ser definida nessa camada. A exceção será algo como `RegistroSemExtratoException`, e vai estender a classe base de exceções (`Exception`), que faz parte da biblioteca padrão da linguagem.

Exemplo:

```
// Dentro do projeto Domain
public class RegistroSemExtratoException : Exception
{
    public RegistroSemExtratoException(string message)
        : base(message)
    {
```

```
    }
}
```

> Exceções devem ser definidas **no lugar onde as regras são violadas**. Se a lógica de um registro e seu extrato estiver no **Domain**, faz sentido que a exceção personalizada também esteja lá.

A classe de exceção herda de `Exception`, e sua responsabilidade é representar esse erro específico. Quando o erro ocorrer, essa exceção será lançada.

Atividade 2: Usar a Exceção na Camada de Aplicação (App)

Agora que temos uma exceção personalizada definida no **Domain**, precisamos utilizá-la no momento em que a regra de negócio é violada.

2.1: Validar o registro no serviço de aplicação.

> A camada **App** é responsável por coordenar as operações. Quando um registro for adicionado, essa camada vai chamar os métodos da camada de domínio para verificar se ele está vinculado a um extrato válido. Se não estiver, ela lançará a exceção personalizada.

Exemplo:

```
// Dentro do projeto App
public class RegistroService
{
```

```
    private readonly IExtratoRepository _extratoRepository;
    private readonly IRegistroRepository _registroRepository;

    public RegistroService(IExtratoRepository extratoReposito
    {
        _extratoRepository = extratoRepository;
        _registroRepository = registroRepository;
    }

    public void AdicionarRegistro(Registro registro)
    {
        if (!_extratoRepository.ExisteExtratoValido(registro.
        {
            throw new RegistroSemExtratoException("Registro n
        }

        _registroRepository.SalvarRegistro(registro);
    }
}
```

OBS: a **validação** do extrato acontece no **Application Service** (camada **App**), mas a regra e o tratamento do erro ainda estão conectados ao **Domain**.

Quando o método verifica se o extrato é válido e encontra um erro, ele lança a exceção que foi definida no
Domain. Essa é uma boa prática de separação de responsabilidades.

Atividade 3: Persistir o Registro (Camada Data)

3.1: Persistir os dados no banco apenas quando as validações passarem.
(Pense onde seus dados nesse contexto estão?)

Depois de validar que o extrato é válido, o registro pode ser salvo na camada

Data, que lida com a persistência no banco de dados.

Exemplo:

```
// Dentro do projeto Data
public class RegistroRepository : IRegistroRepository
{
    public void SalvarRegistro(Registro registro)
    {
        // Código para persistir o registro no banco de dados
    }
}
```

O **Data** só se preocupa com o armazenamento de dados e **não contém regras de negócio**.

A lógica de validação já foi tratada nas camadas anteriores, então a **Data Layer** só precisa garantir que os dados serão armazenados corretamente.

Conclusão: O Ciclo Completo

- **Domain**: Define as regras de negócio e as exceções personalizadas.
- **App**: Orquestra as operações e lança as exceções quando as regras são violadas.
- **Data**: Persiste os dados após as validações de negócio.

Atividade:

Dentro das atividades, dadas, identifique o que é uma regra de negócio e o que é um dado, ou App interface.