

Refatoração do Startup Game

1. Introdução e Objetivo

O objetivo principal deste projeto foi refatorar um jogo de simulação de gestão de Startups em Java, aplicando princípios de design e arquitetura moderna para aumentar a manutenibilidade, a testabilidade e a escalabilidade. A estrutura atual demonstra uma clara adesão aos conceitos da Arquitetura Limpa (Clean Architecture), isolando o Domínio do Jogo (regras de negócio) da Infraestrutura (persistência) e da Interface (console).

2. Análise da Arquitetura em Camadas

A nova estrutura do projeto está organizada em camadas bem definidas, o que facilita a separação de responsabilidades.

2.1. Domínio (model)

Esta é a camada central e mais isolada.

- **Entidade Principal (Startup.java):** Contém o estado central do jogo (Caixa, Reputação, Moral, Receita Base). A lógica para **aplicar** os Deltas está contida aqui, garantindo que a entidade gerencie sua própria mudança de estado.
- **Value Objects (VO) (model/vo):** A utilização de *Java Records* para Dinheiro, Humor e Percentual é excelente.
- **Objeto de Transferência (Deltas.java):** Representa de forma limpa as alterações de estado resultantes de uma ação, desacoplando a *DecisaoStrategy* da entidade *Startup*.

2.2. Ações de Negócio (actions)

Esta camada implementa o **Padrão Strategy** para modelar as decisões do jogador.

- **DecisaoStrategy.java:** A interface que define a operação *aplicar()*.
- **Estratégias Concretas (MarketingStrategy, EquipeStrategy, etc.):** Cada classe encapsula a lógica específica de uma decisão e seus impactos, lendo os valores de impacto diretamente da *Config*.
- **DecisaoFactory.java:** Implementa o **Padrão Factory**

para centralizar a criação e o mapeamento das estratégias, inicializando-as corretamente com o objeto *Config*.

2.3. Aplicação/Motor (engine)

O *GameEngine* atua como o **coordenador** (Service Layer).

- **GameEngine.java:** Contém a orquestração de alto nível.
 - Gerencia a transação: Carrega o estado, verifica regras (ex: *SaldoInsuficienteException*), executa a *DecisaoStrategy*, persiste o resultado.

- É responsável pela transição de rodadas (finalizarRodada), aplicando a receita e avançando o contador.
- **ScoreService.java:** Lógica de pontuação isolada. O motor delega a responsabilidade de cálculo final a este serviço, mantendo o GameEngine focado na jogabilidade.

2.4. Infraestrutura (persistencia)

A camada de persistência utiliza o **Padrão Repository** para isolar a forma como os dados são armazenados.

- **Interfaces de Repositório (StartupRepository, RodadaRepository, DecisaoAplicadaRepository):** Estas interfaces definem um contrato. O Domínio e a Engine (camadas superiores) interagem **apenas** com estas interfaces, sem saber como os dados são guardados.
- **Implementação Simples::** Uma implementação única que simula o armazenamento de dados usando mapas em memória (DataSourceProvider).

3. Padrões e Princípios Fundamentais

Padrão/Princípio	Aplicação no Projeto	Benefício
Strategy	DecisaoStrategy e suas implementações concretas.	Desacoplamento da lógica de decisão do motor central. Fácil adição de novas ações.
Factory	DecisaoFactory para instanciar estratégias.	Centraliza a criação de objetos e a injeção de dependências (Config).
Repository	Interfaces em persistencia/	Isola a lógica de acesso a dados, facilitando testes e trocas de base de dados.
Value Object	Classes Dinheiro, Humor, Percentual.	Imutabilidade e garantia de estado válido dos dados, reduzindo bugs de estado.
Inversão de Dependência (DI)	O GameEngine recebe interfaces de Repositório via construtor.	O motor de alto nível não depende de implementações de baixo nível (Infraestrutura).

5. Conclusão

O projeto está tecnicamente bem fundamentado. A refatoração resultou numa arquitetura modular e aderente aos padrões de mercado, posicionando o código para fácil manutenção e futura expansão de funcionalidades. A clara separação entre a lógica do jogo (Domínio e Actions) e a infraestrutura (Persistência) é o ponto mais forte do projeto.

[Video Do Startup Funcionando](#)