

Pelo **Padrão criador**, percebe-se que Banco possui os valores iniciais de Calendario e logo Banco é responsável por criar uma instância de Calendário.

Pelo **Padrão criador**, percebe-se que Banco possui os valores iniciais de GerenciadorDeClientes e logo Banco é responsável por criar uma instância de GerenciadorDeClientes.

Pelo **Padrão criador**, percebe-se que Banco possui os valores iniciais de GerenciadorDeContas e logo Banco é responsável por criar uma instância de GerenciadorDeContas

Pelo **Padrão criador**, percebe-se que Banco possui os valores iniciais de GerenciadorDeTransacoes e logo Banco é responsável por criar uma instância de GerenciadorDeTransacoes.

Pelo **Padrão Especialista da Informação** GerenciadorDeTransacoes conhece Transacao. Logo, GerenciadorDeTransacoes é responsável por efetuarTransacao(Transacao) sem a necessidade de instanciar Transacao.

Pelo **padrão acoplamento**, Transacao possui o método processarTransacao que é implementado em Sacar e Transferir. Isso aumenta o reuso e logo aumenta o aclopamento. Como o acoplamento é aumentado, é possível inferir que pelo **padrão coesão**, a coesão é aumentada, pois a responsabilidade de processarTransacao está bem definida entre os elementos do sistema. Além do padrão acoplamento e coesão, percebe-se o uso do **padrão polimorfismo**, pois Sacar e Transferir podem implementar o método processarTransacao() da maneira que desejarem e isso torna o projeto mais flexível.

Pelo **padrão polimorfismo** Poupanca e ContaCorrente podem implementar o método atualizar() de conta. Logo, o projeto nesse ponto se torna mais flexível e com maior escala de reuso. Nesse caso, **o padrão coesão e acoplamento** surgem também, pois quando existe o padrão polimorfismo, automaticamente existe uma dose significativa do padrão coesão e acoplamento (coesão alta e acoplamento baixo).

Pelo **padrão especialista da informação**, Cliente conhece ContaCorrente e Poupanca. Isso é evidenciado nos métodos sacar(Conta) e transferir(Conta). O objeto Conta está como um parâmetro.

Pelo **padrão controlador**, percebe-se que as classes GerenciadorDeContas, GerenciadorDeClientes, GerenciadorDeTransacoes são responsáveis por processar eventos do sistema. Entretanto, não é possível identificar se o controlador é por fachada (não é garantia que tais classes controlam todos os eventos do sistema), por caso de uso (não é garantia que o controle é por caso de uso) e nem por papel (não é garantia que o controle é próximo do mundo real).

É possível inferir que pelo **padrão indireção**, objetos intermediários compartilham responsabilidades, por exemplo através das classes GerenciadorDeContas, GerenciadorDeClientes, GerenciadorDeTransacoes. Mas, não é possível afirmar que isso ocorra com garantia. Para garantir isso, devia se obter mais detalhes de implementação.

Pelo **padrão Invenção pura** (classe fictícia para aumentar a coesão e diminuir o acoplamento, sem representar no modelo de domínio) pode-se inferir que as classes Sacar e Transferir possui a responsabilidade de guardar os dados em algum lugar, provavelmente um Banco de Dados. Uma classe intermediária para fazer isso (não representada no modelo do domínio) poderia fazer isso e gerar o padrão Invenção pura.

Pelo **padrão variações protegidas** Banco pode conhecer Conta através de Poupanca e ContaCorrente. Mas, para fazer isso tem que passar por GerenciadorDeClientes e Cliente. Nesse caso, recomenda-se que Banco conheça Conta através de GerenciadorDeContas, pois o “caminho é mais curto”.