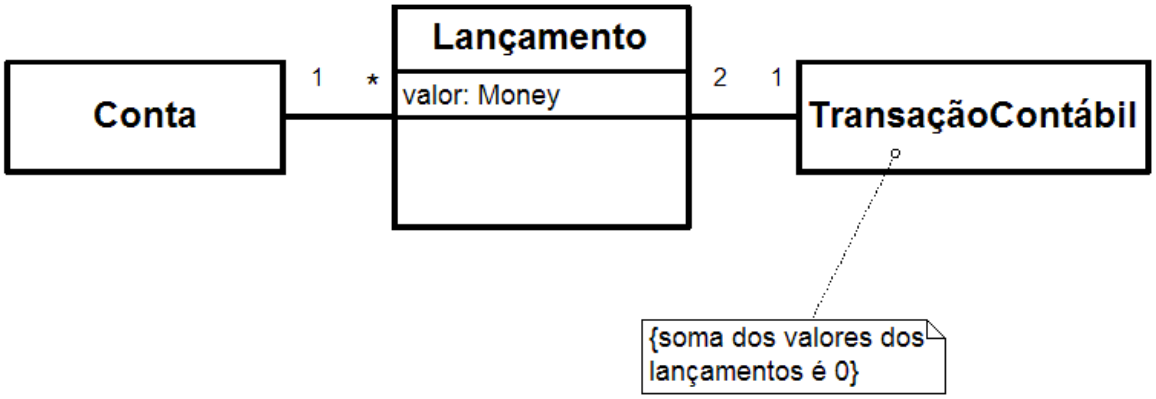
Analysis Pattern: Transação Contábil (Accounting Transaction)

O que é

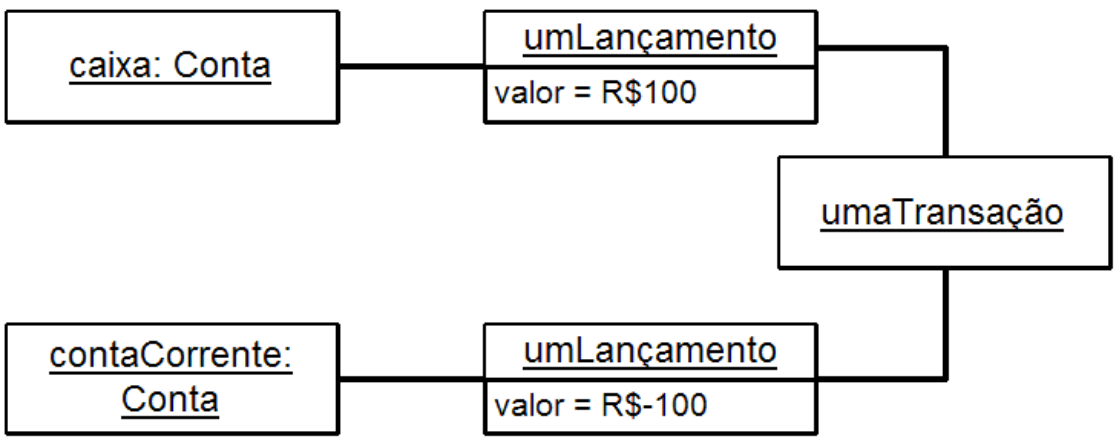
- Liga dois ou mais lançamentos de forma a que a soma dos lançamentos de uma transação seja zero



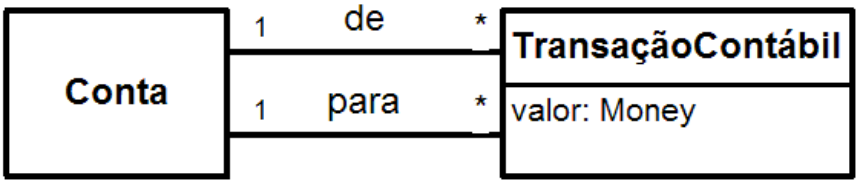
- Uma das práticas mais importantes da contabilidade é o uso de "partidas dobradas" (*double entry bookkeeping*)
  - Cada saque deve ser balançado por um depósito
    - Daí, as palavras Balanço, Balancete, ...
  - Tudo que é feito nos livros contábeis tem dois elementos: subtração de uma conta e adição numa outra
  - Isso é "conservação do dinheiro", mais ou menos como a conservação da energia
  - Dinheiro não é criado, só é movido de um lugar para outro

Detalhes de funcionamento

- Há transações de dois tipos:
  - Transações de dois lançamentos
    - Dois lançamentos, com sinais opostos
    - Movimento de uma conta para outra
  - Transações multi-lançamento
    - Mais de dois lançamentos
    - A soma dos valores é zero



- É mais simples implementar apenas transações de dois lançamentos
    - Então não complique seu sistema com transações multi-lançamento, a não ser que sejam realmente usadas
- Se a informação dos dois lançamentos, incluindo as datas, for sempre igual (com exceção do sinal do valor), então pode-se eliminar os lançamentos e colocar todos os dados na transação
    - Veja o diagrama abaixo



- A construção de uma transação de dois lançamentos é simples e pode ser feita numa única operação
- Para transações multi-lançamento, a operação é mais complicada, sendo feita aos poucos
    - Pode ser útil usar o padrão "Proposed Object" em que se registra os detalhes de um objeto antes de ele ser oficialmente considerado "criado e pronto para uso"
    - Depois de completamente criado, pode ser inserido nas contas apropriadas

Quando deve ser usado

- Por que a técnica de partidas dobradas foi inventada?
    - Para facilitar a descoberta de vazamentos e, assim, evitar fraudes
    - Sem partidas dobradas, é mais fácil fazer dinheiro aparecer e desaparecer misteriosamente
    - Hoje a técnica é sempre usada nas empresas
- Só use o padrão "Transação Contábil" se você usar o padrão "Conta"
- Deixe os Domain Experts decidirem se este padrão será usado
- Só use transações multi-lançamentos quando forem estritamente necessárias
    - São mais genéricas porém mais complexas

Código Exemplo

Transações com dois lançamentos

- Nosso objeto de transação é simples:

```
public class TransacaoContabil {
        private Collection lancamentos = new HashSet();

        public TransacaoContabil(Money valor, Conta de, Conta para, Calendar data) {
                Lancamento lancamentoDe = new Lancamento(valor.negate(), data);
                de.addLancamento(lancamentoDe);
                lancamentos.add(lancamentoDe);
                Lancamento lancamentoPara = new Lancamento(valor, data);
                para.addLancamento(lancamentoPara);
                lancamentos.add(lancamentoPara);
        }
}
```

- Em vez de usar o construtor de TransacaoContabil diretamente, é melhor

prover métodos que expressem o contexto da operação:

```
void saque(Money valor, Conta alvo, Calendar data) {
        new TransacaoContabil(valor, this, alvo, data);
}
```

- Agora, as manipulações ficam mais simples, como podemos ver no código de teste abaixo

```
public void testBalancoUsandoTransacoes() {
        receitas = new Conta(Moeda.BR);
        contasProteladas = new Conta(Moeda.BR);
        contasAReceber = new Conta(Moeda.BR);
        receitas.saque(Money.reais(500), contasAReceber, criaCalendar(2003, 10, 1));
        receitas.saque(Money.reais(200), contasProteladas, criaCalendar(2003, 10, 1));
        assertEquals(Money.reais(500), contasAReceber.saldo());
        assertEquals(Money.reais(200), contasProteladas.saldo());
        assertEquals(Money.reais(-700), receitas.saldo());
}
```

Transações multi-lançamento

- Tais transações são mais difíceis de criar e requerem validação
- No código que segue, usamos o padrão "Proposed Object" de forma que podemos montar a transação aos poucos e lançá-la nas contas no final
    - Chamadas separadas de métodos serão usadas para adicionar lançamentos na transação em vez de fazer tudo no construtor
    - No final, depois dos lançamentos individuais, a transação é lançada nas contas associadas
        - Antes de fazer o lançamento final, precisamos nos certificar de que há balanço na transação (soma de valores = zero)
        - Também, uma vez a transação lançada, não podemos adicionar novos lançamentos a ela
- Vamos primeiro ver os atributos e o construtor:

```
public class TransacaoContabil {
        private Calendar data;
        private Collection lancamentos = new HashSet();
        private boolean fechada = false;
        public TransacaoContabil(Calendar data) {
                this.data = data;
        }
}
```

- Temos uma única data para toda a transação
    - Embora mais raro, em outras situações mais complexas, poderemos talvez precisar de datas separadas para cada lançamento
- O método add segue:

```
class TransacaoContabil {
        ...
        public void add(Money valor, Conta Conta) {
                if (fechada) {
                        throw new TransacaoImutavelException(
                         "Nao pode adicionar lancamento numa transacao fechada");
            }
                lancamentos.add(new Lancamento(valor, data, Conta, this));
        }
}
```

- A seguir, usaremos uma classe de Lancamento diferente para manter

associações bidirecionais entre o lançamento e ambos a transação e a conta:

```
class Lancamento {
        ...
        private Money valor;
        private Calendar data;
        private Conta conta;
        private TransacaoContabil transacao;
        Lancamento(
                Money valor,
                Calendar data,
                Conta conta,
                TransacaoContabil transacao) {

                this.valor = valor;
                this.data = data;
                this.conta = Conta;
                this.transacao = transacao;
        }
}
```

- Podemos agora ver como lançar a transação inteira:

```
class TransacaoContabil {
        ...
        public void lançar() {
                if (!podeLançar())
                        throw new NaoPodeLancarException();
                Iterator it = lancamentos.iterator();
                while (it.hasNext()) {
                        Lancamento umLancamento = (Lancamento) it.next();
                        umLancamento.lançar();
                }
                fechada = true;
        }
        public boolean podeLançar() {
                return saldo().isZero();
        }
        private Money saldo() {
                Money resultado = Money.reais(0);
                Iterator it = lancamentos.iterator();
                while(it.hasNext()) {
                        Lancamento umLancamento = (Lancamento) it.next();
                        resultado = resultado.add(umLancamento.valor());
                }
                return resultado;
        }
}
class Lancamento {
        ...
        void lançar() {
                conta.addLancamento(this);
        }
}
```

- Agora podemos usar a transação como segue:

```
void testMulti() {
        TransacaoContabil multi = new TransacaoContabil(criaCalendar(2003, 10, 25));
        multi.add(Money.reais(-700), receitas);
        multi.add(Money.reais(500), contasAReceber);
        multi.add(Money.reais(200), contasProteladas);
        multi.lançar();
        assertEquals(Money.reais(500), contasAReceber.saldo());
        assertEquals(Money.reais(200), contasProteladas.saldo());
```

```
        assertEquals(Money.reais(-700), receitas.saldo());
}
```

- Deve ter ficado claro por que usar transações multi-lançamento é tão mais complicado do que transações de dois lançamentos ...
- Por outro lado, se você tiver muitas transações multi-lançamento e poucas transações de dois lançamentos, pode valer a pena implementar a interface de transações de dois lançamentos como uma transação multi-lançamento:

```
class Conta {
        ...
        void saque(Money valor, Conta alvo, Calendar data) {
                TransacaoContabil trans = new TransacaoContabil(data);
                trans.add(valor.negate(), this);
                trans.add(valor, alvo);
                trans.lançar();
        }
}
```

programa