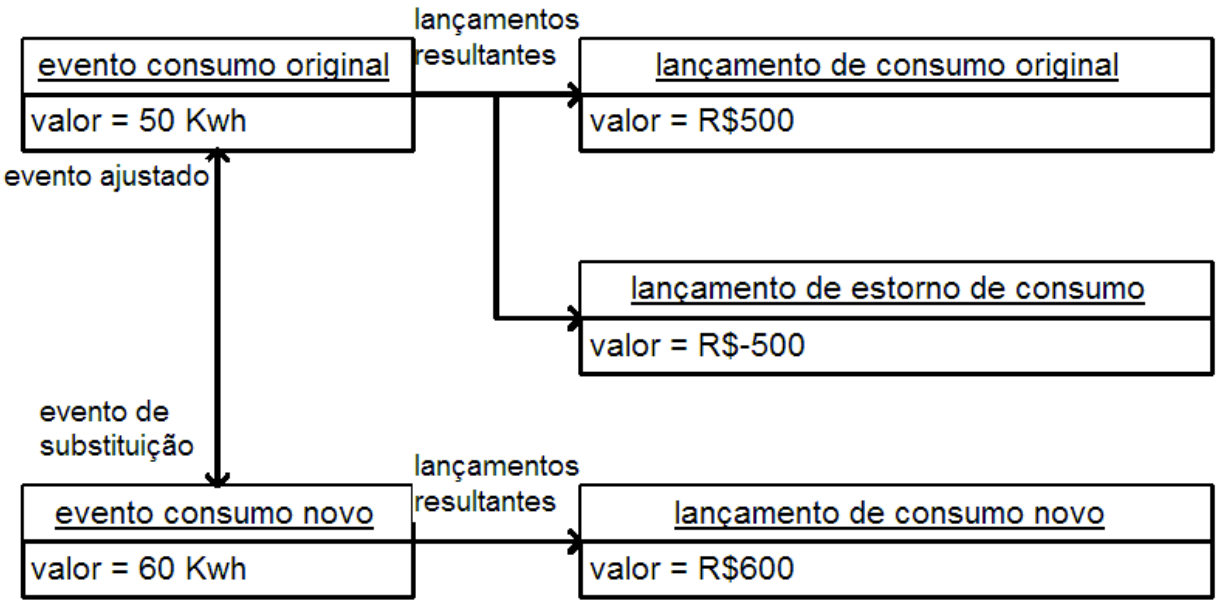
Analysis Pattern: Estorno (Reversal Adjustment)

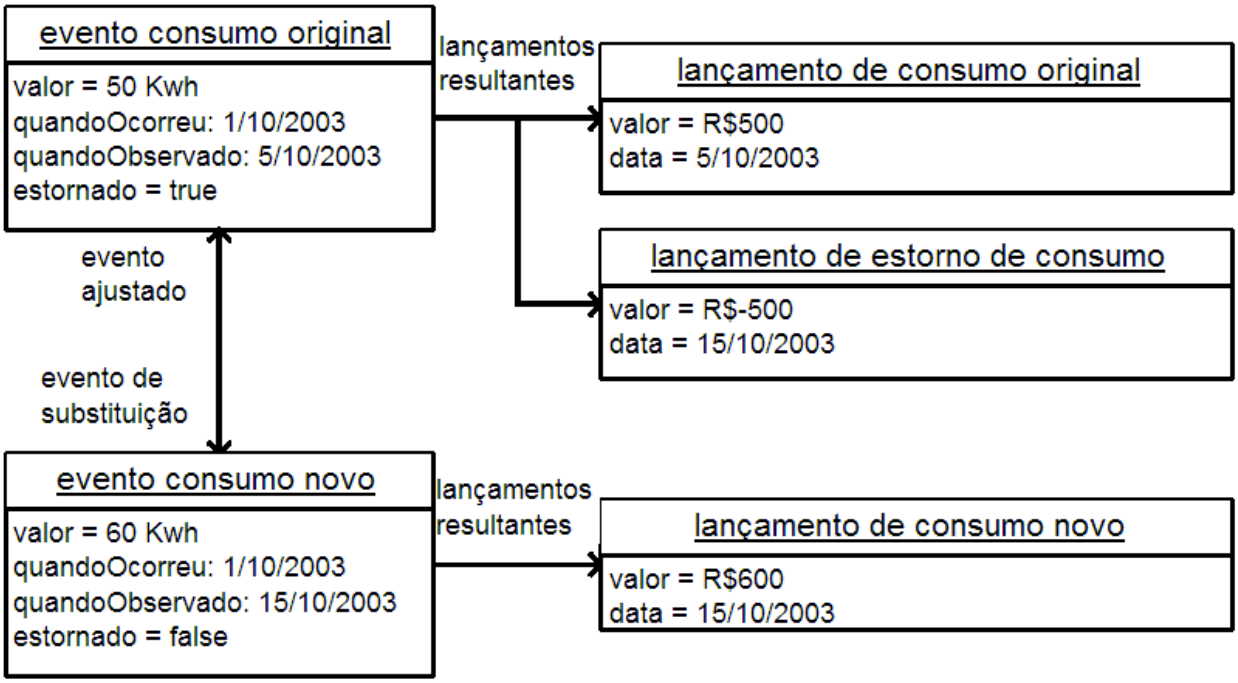
O que é

- Ajuste a lançamentos existentes pela criação de lançamentos inversos (estornos) seguida da criação de lançamentos corretos



Detalhes de funcionamento

- Para cada lançamento que deve ser ajustado, você cria dois novos lançamentos
    - Um (o estorno) reverte o lançamento original
        - Mantenha o valor com sinal oposto
        - Mantenha o atributo quandoOcorreu
        - quandoObservado é a data do lançamento da correção
    - Um é o lançamento como deveria ter sido
- Associamos o primeiro lançamento ao evento original
    - Se ajustarmos o novo evento no futuro, não vamos estornar o estorno!
- E marcamos o evento original como "estornado" para que não seja estornado novamente
- Veja o resultado abaixo:



- É freqüente ter que fornecer relatórios em que lançamentos de estorno foram expurgados

Quando deve ser usado

- É a alternativa mais simples se tivermos lançamentos imutáveis
- Use ajuste de diferencial apenas se o Domain Expert preferir ver a informação desta forma

Código exemplo

- Veja que comportamento final nós gostaríamos de ter no código de testes abaixo
    - Basicamente, ao pedir um saldo, é como se o evento original nunca tivesse existido

```
class Testador {
        ...
        public void setUp() {
                configClienteNormal();
                configClienteBaixaRenda();
                eventoConsumo =
                        new Consumo(
                                Unit.KWH.valor(50),
                                criaCalendar(2003, 10, 1),
                                criaCalendar(2003, 10, 1),
                                cam);
                listaLancamentos.add(eventoConsumo);
                listaLancamentos.processa();
        }

        public void testAjuste() {
                Consumo ajuste1 =
                        new Consumo(
                                Unit.KWH.valor(70),
                                criaCalendar(2003, 10, 1),
                                criaCalendar(2003, 10, 15),
                                eventoConsumo);
                listaLancamentos.add(ajuste1);
                listaLancamentos.processa();
```

```
                assertEquals(
                        Money.reais(700),
                        cam.saldoDe(TipoLancamento.CONSUMO_BASICO));
                assertEquals(Money.reais(38.5), cam.saldoDe(TipoLancamento.IMPOSTO));
        }
}
```

- Na nossa implementação a classe do evento de ajuste é a mesma do evento original
  - A diferença está no construtor chamado
  - Este construtor recebe o evento a ser ajustado e cria o novo evento (veja abaixo)

```
class EventoContabil {
        ...
        private EventoContabil eventoAjustado, eventoDeSubstituicao;

        EventoContabil(
                TipoEvento tipo,
                Calendar quandoOcorreu,
                Calendar quandoObservado,
                EventoContabil eventoAjustado) {

                if (eventoAjustado.foiAjustado())
                        throw new IllegalArgumentException(
                                "O " + eventoAjustado + " ja foi ajustado ");
                this.tipo = tipo;
                this.quandoOcorreu = quandoOcorreu;
                this.quandoObservado = quandoObservado;
                this.eventoAjustado = eventoAjustado;
                eventoAjustado.eventoDeSubstituicao = this;
        }

        protected boolean foiAjustado() {
                return (eventoDeSubstituicao != null);
        }
}
```

- Observe acima que uma ligação é mantida entre os dois eventos
- Agora, devemos cuidar do processamento de um evento
  - Devemos reverter o evento original

```
class EventoContabil {
        ...
        public void processa() {
                assert(!foiProcessado); //Nao pode processar um evento duas vezes
                if (eventoAjustado != null)
                        eventoAjustado.reverte();
                achaRegra().processa(this);
                foiProcessado = true;
        }

        void reverte() {
                Collection lancamentos = new HashSet(getLancamentosResultantes());
                Iterator it = lancamentos.iterator();
                while (it.hasNext()) {
                        Lancamento umLancamento = (Lancamento) it.next();
                        Lancamento lancamentoParaReverter =
                                new Lancamento(
                                        umLancamento.getValor().inverte(),
                                        quandoObservado,
                                        umLancamento.getTipo());
                        getCliente().addLancamento(lancamentoParaReverter);
                        this.addLancamentoResultante(lancamentoParaReverter);
```

```
                }
                reverteEventosSecundarios();
        }

        private void reverteEventosSecundarios() {
                Iterator it = getEventosSecundarios().iterator();
                while (it.hasNext()) {
                        EventoContabil umEvento = (EventoContabil) it.next();
                        umEvento.reverte();
                }
        }
}
```

programa