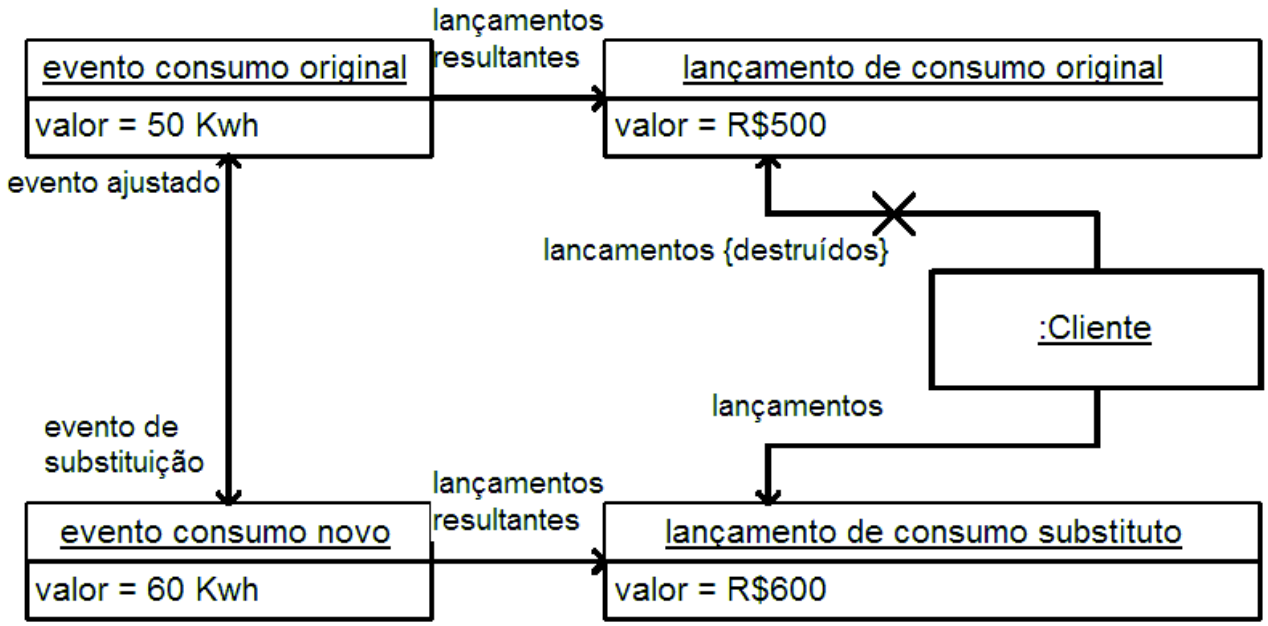
Analysis Pattern: Ajuste por Substituição (Replacement Adjustment)

O que é

- Ajuste a um evento errado removendo lançamentos antigos e substituindo-os por lançamentos novos



- Podem-se editar os valores dos lançamentos originais ou então substitui-los por inteiro

Detalhes de funcionamento

- É uma estratégia muito simples de ajuste
- Ache todos os lançamentos do evento antigo, remove-os e processe o novo evento normalmente
- Você pode manter os lançamentos velhos ainda ligados ao evento original por motivos de auditoria
    - Porém, não ficarão nas coleções principais (como contas, por exemplo)

Quando deve ser usado

- Pró: Minimiza complexidade dos lançamentos
- Contra: Não deixa uma trilha de auditoria clara dos lançamentos
    - Pode haver informação num log, mas não na seqüência de lançamentos nas contas
    - Normalmente não é usado depois que uma ação irreversível foi feita (enviar uma fatura para o cliente, por exemplo)

Código exemplo

- Deve-se tratar da remoção dos eventos antigos
- Iniciamos com um novo construtor e atributos para os eventos

ajustados (antigos) e substitutos

```
class EventoContabil {
        private EventoContabil eventoAjustado, eventoSubstituto;
        public EventoContabil(
                TipoEvento tipo,
                Calendar quandoOcorreu,
                Calendar quandoObservado,
                EventoContabil eventoAjustado) {
                if (eventoAjustado.jaFoiAjustado())
                        throw new IllegalArgumentException(
                                "Nao pode criar "
                                        + this
                                        + "."
                                        + eventoAjustado
                                        + "ja foi ajustado");
                this.tipo = tipo;
                this.quandoOcorreu = quandoOcorreu;
                this.quandoObservado = quandoObservado;
                this.eventoAjustado = eventoAjustado;
                eventoAjustado.eventoSubstituto = this;
        }
        protected boolean jaFoiAjustado() {
                return (eventoSubstituto != null);
        }
}
```

- Agora, no método processa(), verificamos se o evento que estamos processando tem um evento de ajuste e, se tiver, o desfazemos

```
class EventoContabil {
        ...
        public void processa() {
                assert(!foiProcessado); // Nao pode processar um evento duas vezes
                if (eventoAjustado != null)
                        eventoAjustado.undo();
                acharRegra().processa(this);
                foiProcessado = true;
        }
        public void undo() {
                Lancamento[] lancamentos = getLancamentosResultantes();
                for (int i = 0; i < lancamentos.length; i++)
                        getSubject().removeLancamento(lancamentos[i]);
                undoEventosSecundarios();
                lancamentosResultantes = null;
        }
        private void undoEventosSecundarios() {
                Iterator it = getEventosSecundarios().iterator();
                while (it.hasNext()) {
                        EventoContabil umLancamento = (EventoContabil) it.next();
                        umLancamento.undo();
                }
        }
}
```

programa