Analysis Pattern: Regra Secundária de Lançamento (Secondary Posting Rule)

- Permitir que várias Regras de Lançamento chamem uma outra Regra para compartilhar seu efeito

O que é

- Às vezes, é necessário que uma Regra de Lançamento chame outra
- Isso ocorre freqüentemente com impostos
    - Uma empresa de distribuição de energia elétrica deve pagar, digamos, 5.5% sobre todas suas faturas de consumo e de serviço
- Não queremos duplicar essa lógica em todas as Regras de Lançamento
- É melhor criar uma Regra que seja chamada por outra Regra que precise do efeito

Detalhes de funcionamento

- Há duas formas de realizar o trabalho
    - Primeira forma: crie uma Regra Secundária e chame-a passando o evento original
    - Segunda forma: crie um segundo evento e processe-o normalmente
        - Isso cria um segundo evento, o que pode ser indesejável em certas situações

Quando deve ser usado

- Use o padrão "Regra Secundária de Lançamento" quando há lógica duplicada entre Regras de Lançamento
    - Fatore a lógica duplicada em Regras Secundárias

Código exemplo

- Alteraremos o código anterior para incluir um imposto fixo de 5.5% sobre tudo
    - Usaremos a "segunda forma", criando um novo evento
    - É exercício de casa fazer o trabalho sem criar segundo evento
- Na realidade, podemos usar uma regra existente para realizar o cálculo: RegraFormulaSimples

```
class Testador {
        ...
        public void configuraClienteNormal() {
                cam = new Cliente("Cafe A Margot");
                AcordoServico padrao = new AcordoServico();
                ...
                padrao.addRegraLancamento(
                        TipoEvento.IMPOSTO,
                        new RegraFormulaSimples(
                                0.055,
                                Money.reais(0),
                                TipoLancamento.IMPOSTO),
                        criaCalendar(2003, 10, 1));
                cam.setAcordoServico(padrao);
        }
}
```

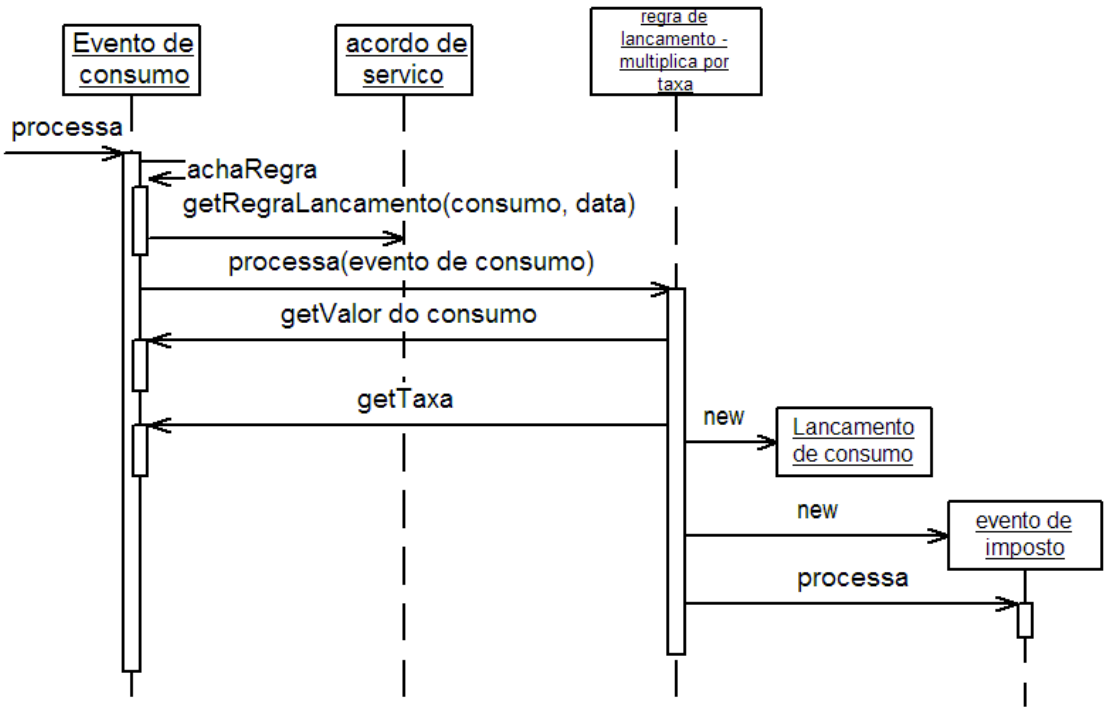
- Precisamos de um novo TipEvento e de um novo TipoLancamento para impostos
- Depois, precisamos criar um evento para passá-lo à Regra de Lançamento
    - Para tanto, podemos alterar o método processa() da superclasse RegraLancamento

```
class RegraLancamento {
        ...
        public void processa(EventoContabil evento) {
                facaLancamento(evento, calculaValor(evento));
                if (isTributavel()) {
                        new EventoTributo(evento, calculaValor(evento)).processa();
                }
        }
}
```

- Temos que ter cuidado para não criar eventos de imposto a partir de eventos de imposto, para evitar recursão infinita
    - Veja uma forma de fazer isso abaixo

```
class RegraLancamento {
        ...
        private boolean isTributavel() {
                return !(tipo == TipoLancamento.IMPOSTO);
        }
}
```

- Em UML, temos a seguinte situação:



- O evento de imposto é simples:

```
class EventoImposto extends EventoMonetario {
        private EventoContabil base;
        public EventoImposto(EventoContabil base, Money valorTributavel) {
                super(
                        valorTributavel,
                        TipoEvento.IMPOSTO,
                        base.getQuandoOcorreu(),
                        base.getWhenNoticed(),
                        base.getCliente());
                this.base = base;
                assert(base.getTipoEvento() != getTipoEvento());// Provavel recursao infinita
        }
}
```

- Observe a programação defensiva no construtor acima
- Exercício de casa:
    - Use o padrão de projeto Decorator para implementar os lançamentos de impostos, tanto para a situação em que se cria um novo evento como para a situação em que nenhum novo evento é criado)

- Compare nossa solução acima com sua solução com Decorator

Mantendo o Rastreio de Eventos

- Como veremos à frente com Estorno (Reversal Adjustment), é necessário manter um rastreio completo entre eventos e lançamentos
- No nosso caso, teremos que achar os resultados dos eventos secundários a partir dos eventos primários
- Uma forma de fazer isso é de manter uma ligação bidirecional entre os eventos

Código exemplo

- Já tínhamos uma ligação entre o evento secundário e o evento base acima (atributo "base")
- Agora, precisamos de uma ligação bidirecional, para achar os eventos secundários a partir dos eventos primários

```
class EventoImposto {
        ...
        public EventoImposto(EventoContabil base, Money valorTributavel) {
                super(
                        valorTributavel,
                        TipoEvento.IMPOSTO,
                        base.getQuandoOcorreu(),
                        base.getQuandoObservado(),
                        base.getCliente());
                this.base = base;
                base.addEventoSecundario(this);
                assert(base.getTipoEvento() != getTipoEvento());// Provavel recursao infinita
                ...
        }
}
class EventoContabil {
        ...
        private List eventosSecundarios = new ArrayList();
        void addEventoSecundario(EventoContabil arg) {
                //so deve ser chamado pelo metodo set do evento secundario
                eventosSecundarios.add(arg);
        }
}
```

- Agora, podemos prover métodos que usem este comportamento:

```
class EventoContabil {
        ...
        Set getTodosLancamentosResultantes() {
                Set resultado = new HashSet();
                resultado.addAll(lancamentosResultantes);
                Iterator it = eventosSecundarios.iterator();
                while (it.hasNext()) {
                        EventoContabil evento = (EventoContabil) it.next();
                        result.addAll(evento.getlancamentosResultantes());
                }
                return result;
        }
}
```

- Aqui está um teste que verifica o funcionamento:

```
class Tester public void testConsumo() {
        Consumo evento =
                new Consumo(
                        Unit.KWH.valor(50),
                        criaCalendar(2003, 10, 1),
                        criaCalendar(2003, 10, 1),
                        cam);
        evento.processa();
        Lancamento lancamentoConsumo = getLancamento(cam, 0);
        Lancamento lancamentoImposto = getLancamento(cam, 1);
        assertEquals(Money.reais(500), lancamentoConsumo.getValor());
```

```
        assertEquals(TipoLancamento.CONSUMO_BASICO, lancamentoConsumo.getTipo());
        assertEquals(Money.reais(27.5), lancamentoImposto.getValor());
        assertEquals(TipoLancamento.IMPOSTO, lancamentoImposto.getTipo());
        assert(evento.getlancamentosResultantes().contains(lancamentoConsumo));
        assert(evento.getTodosLancamentosResultantes().contains(lancamentoImposto));
}
```

programa