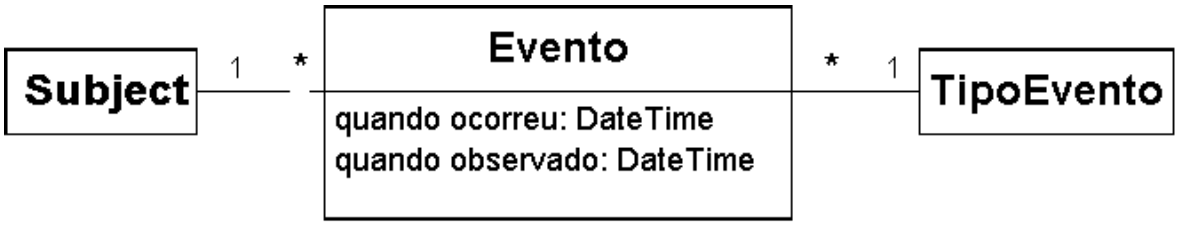
Analysis Pattern: Evento (Event)

O que é

- Um Evento captura a memória de algo interessante que afeta o domínio



- Exemplo:
  - Vou para o restaurante Tábua de Carne na terça-feira e pago com cartão de crédito
  - Isso pode ser modelado como um evento, cujo tipo é "compra", com Subject igual ao cartão de crédito e com data de ocorrência na terça-feira. Se a Tábua de Carne transmitir a transação apenas na sexta-feira, então a data de observação seria sexta-feira
- Para ser interessante e valer a pena ser registrado e modelado, o evento deve causar uma reação no sistema
  - Isto é o motivo de ter eventos num sistema: eles logam coisas que ocorrem
  - Essa idéia é absolutamente central para logs de auditoria que são essencialmente listas de eventos
    - Em outras palavras, uma trilha de auditoria conteria tipicamente eventos e os lançamentos resultantes. Os relacionamentos evento-lançamentos é mais fácil de manter se os eventos forem reificados.

Detalhes de funcionamento

- Lembrando o papel de eventos em logs de auditoria, é importante que eventos sejam objetos imutáveis
  - Uma vez criado, os atributos do evento não podem mudar
  - Se houver motivo para alterar o evento (devido a um erro, por exemplo), técnicas especiais deverão ser usadas (ex. Estorno)
- Em alguns casos, apenas uma data precisa ser mantida, mas sempre analise se ambas são necessárias
- Veremos à frente que tipos de Subjects e EventTypes podemos ter relacionados aos Eventos

Quando deve ser usado

- Quando você deve manter um log das coisas que fazem o sistema mudar
- Cada "coisa" precisa de um evento para gatilhá-la, e o evento mantém a informação relevante
- Desvantagem: pode dar trabalho logar tudo que ocorre
- Vantagem: o "audit trail" fica simples de implementar

Código exemplo

```
interface Evento {
    Event novoEvento (TipoEvento tipo, NumeroConta conta,
                      Calendar quandoOcorreu, Calendar quandoObservado);
    TipoEvento getTipo();
    NumeroConta getConta();
    Calendar getQuandoOcorreu();
    Calendar getQuandoObservado();
}
```

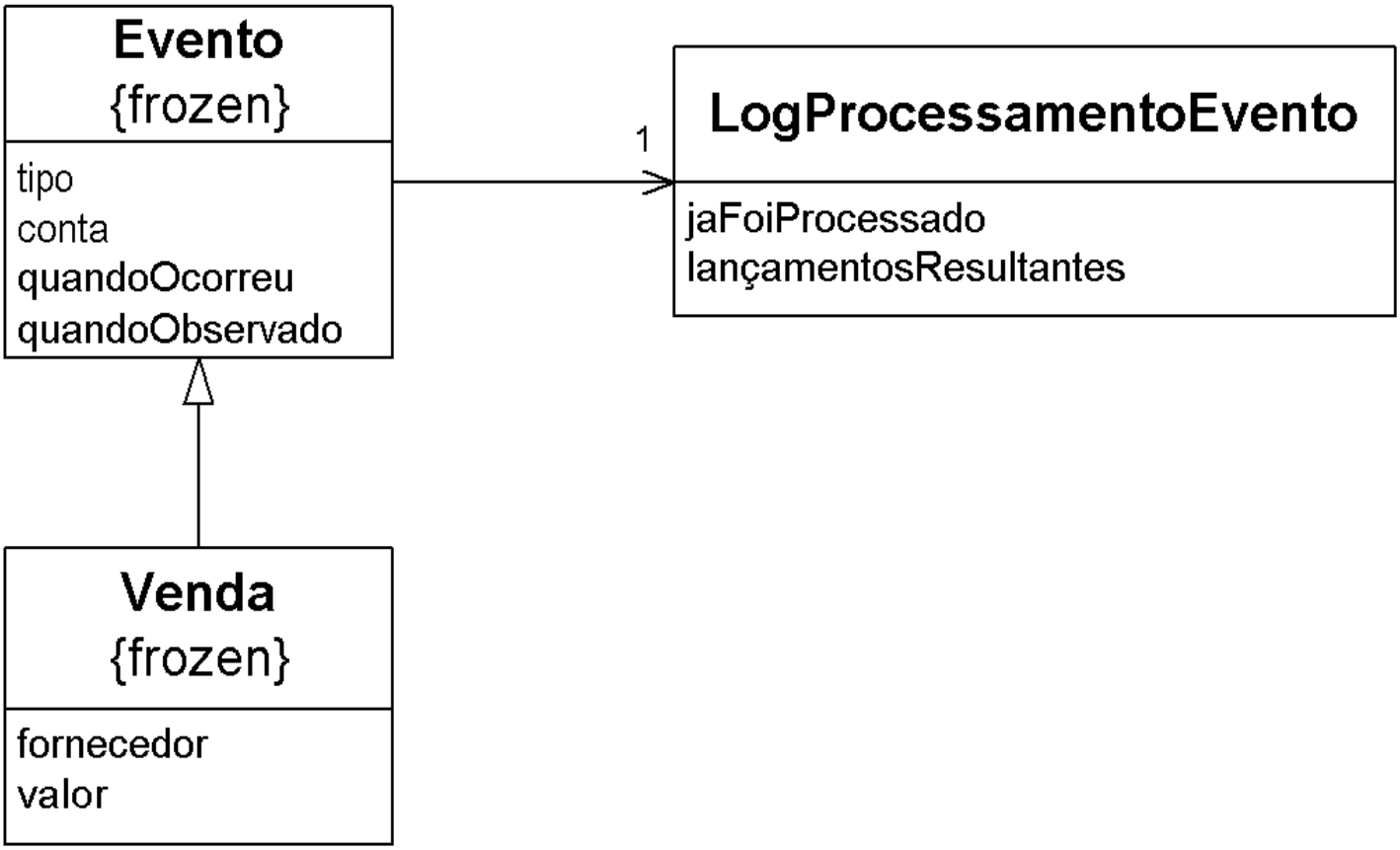
- Observe que a interface não permite alterar os valores dos atributos
- Agora, podemos criar um subtipo útil para nosso sistema

```
interface Venda extends Evento {
    Venda novaVenda (NumeroConta conta,
                     Calendar quandoOcorreu,
                     Calendar quandoObservado,
                     Fornecedor fornecedor,
                     Money valor);
    Fornecedor getFornecedor();
    Money getValor();
}
```

- Observe que o tipo não está no construtor de Venda mas será informado pelo subtipo
- Às vezes, queremos adicionar dados sobre o processamento do Evento
    - Tal informação não precisa ser imutável

```
interface Evento...
   boolean jaFoiProcessado();
   Set getLancamentosResultantes();
   void addLancamentoResultante(Lancamento lancamento);
   ...
```

- Já que há informação mutável e imutável no Evento, pode ser interessante separar as duas partes
    - Veja abaixo (as classes Venda e Evento são imutáveis)



- Também é possível fazer com que a parte mutável referencie a parte imutável
- Lembre que não é absolutamente necessário efetuar a separação das duas partes
- Podemos agora passar para a segunda entidade importante: o lançamento

programa