Modelagem Funcional (Contratos)

© Prof. Raul Sidnei Wazlawick

UFSC-CTC-INE

2011

Fonte: Análise e Projeto de Sistemas de Informação Orientados a Objetos, 2ª Edição, Elsevier, 2011.

Artefatos necessários

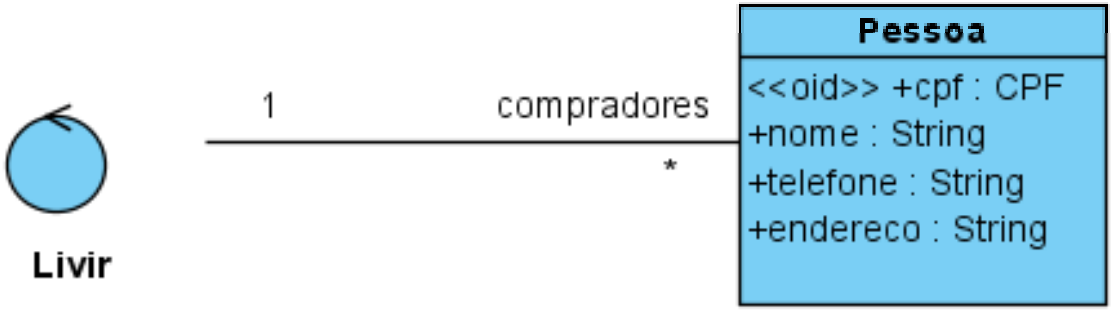
- Modelo conceitual
- Modelo de interação (DSS)

Contratos

- Operação de Sistema
  - Pré-condições (opcional)
  - Pós-condições
  - Exceções (opcional)
- Consulta de Sistema
  - Pré-condições (opcional)
  - Resultados

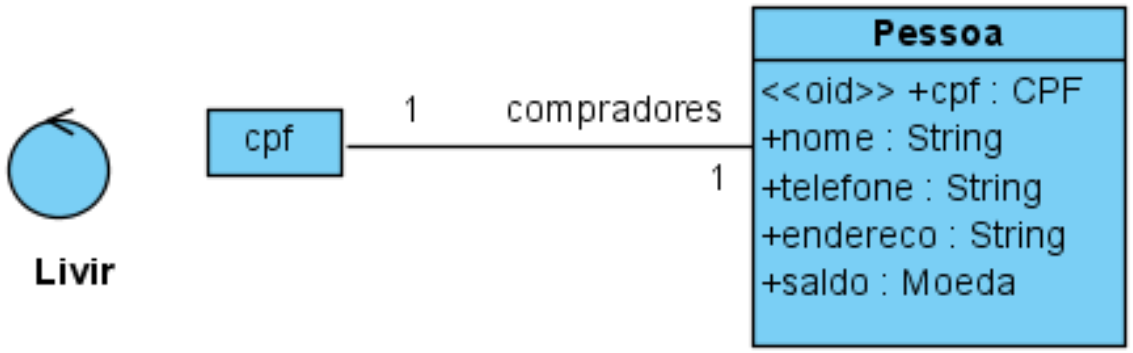
Pré-condições

- Estabelecem o que é verdadeiro quando uma operação ou consulta de sistema for executada.




```
Context Livir::identificaComprador(umCpf)

  pre:
    self.compradores→select(p|p.cpf=umCpf)→notEmpty()
```

... com associação qualificada




```
Context Livir::identificaComprador(umCpf)

  pre:
    self.compradores[umCpf]→notEmpty()
```

Tipos de pré-condição

- *Garantia de parâmetros*: pré-condições que garantem que os *parâmetros* da operação ou consulta correspondem a elementos válidos do sistema de informação, como, por exemplo, que existe cadastro para o comprador cujo CPF corresponde ao parâmetro da operação ou consulta.
- *Restrição complementar*: pré-condições que restringem ainda mais o modelo conceitual para a execução da operação ou consulta, de forma a garantir que a informação se encontra em uma determinada situação desejada, por exemplo, que o endereço para entrega informado pelo comprador não esteja no estado *inválido*.

Garantia de parâmetros

- Devem ser semânticas, não sintáticas.
- Para garantir que um parâmetro seja, por exemplo, um número maior do que zero, basta usar *tipagem* (por exemplo, "x:InteiroPositivo"), não sendo necessário escrever isso como pré-condição.

Restrição complementar

- Tipos:
  - Fazer uma *afirmação específica* sobre uma instância ou um conjunto de instâncias.
  - Fazer uma *afirmação existencial* sobre um conjunto de instâncias.
  - Fazer uma *afirmação universal* sobre um conjunto de instâncias.

Afirmação específica

```
Context Livir::operacaoQualquer()

  pre:
    compradores[12345678910].saldo = 0
```

Afirmação Existencial

```
Context Livir::operacaoQualquer()

  pre:
    compradores→exists(c|c.saldo = 0)
```

Afirmação Universal

```
Context Livir::operacaoQualquer()

  pre:
    compradores→forAll(c|c.saldo = 0)
```

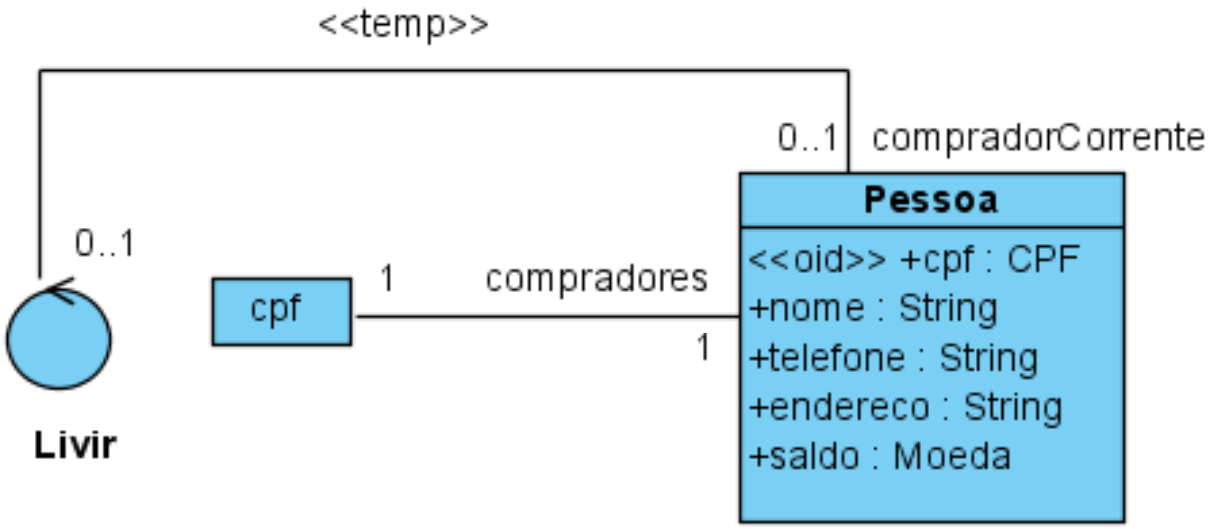
Garantia das pré-condições

- Como as pré-condições não são testadas pela operação admite-se que algum mecanismo externo as garanta.
  - Pode-se, por exemplo, antes de chamar a operação, executar uma consulta que testa a pré-condição e só chama a operação se o resultado for positivo.
  - Pode-se ainda criar mecanismos restritivos de interface que garantam que a operação só é executada se a pré-condição for observada.

Pré-condição x Invariante

- Usa-se invariantes no modelo conceitual para regras que valem sempre, independentemente de qualquer operação.
- Usa-se precondições para regras que valem apenas quando determinada operação ou consulta está sendo executada.
- Quando já existir uma invariante para determinada situação não é necessário escrever pré-condições para a mesma situação.
- Assume-se que o projeto deva ser efetuado de forma que tanto a invariante quanto as eventuais pré-condições nunca sejam desrespeitadas.
- Mecanismos de teste de projeto poderão verificar as invariantes e pré-condições durante a fase de teste do sistema.

Associações temporárias para statefull



Retorno de Consulta

- Ex.: Saldo do comprador corrente:

```
Context Livir::saldoCompradorCorrente():Moeda

  body:
    compradorCorrente.saldo
```

Ex.:

- Nome e telefone do comprador cujo CPF é dado:

```
Context Livir::nomeTelefoneComprador(umCpf):Tuple

body:
  Tuple{
    nome = compradores[umCpf].nome,
    telefone = compradores[umCpf].telefone
  }
```

Usando cláusula def:

```
Context Livir::nomeTelefoneComprador(cpfComprador):Tuple

  def: comprador =
    compradores[cpfComprador]

  body:
    Tuple{
      nome = comprador.nome,
      telefone = comprador.telefone
    }
```

Projeção

- Nomes de todos os compradores

```
Context Livir::listaNomeCompradores():Set

  body:
     compradores->collect(p|p.nome)
```

```
Context Livir::listaNomeCompradores():Set

  body:
    compradores.nome
```

Filtro e projeção

- Nome de todos compradores com saldo zero

```
Context Livir::listaNomeCompradoresSaldoZero():Set

  body:
    compradores→select(saldo=0).nome
```

Ex.:

- CPF, nome e telefone de todos compradores com saldo zero

```
Context Livir::listaCpfNomeTelefoneCompradoresSaldoZero():Set

  body:
    compradores→select(saldo=0)→collect(c|
      Tuple {
        cpf = c.cpf,
        nome = c.nome,
        telefone = c.telefone
      }
    )
```

Pós-Condições

- As pós-condições estabelecem o que muda nas informações armazenadas no sistema após a execução de uma operação de sistema.

```
Context Livir::operacaoX()
  post:
    <expressão OCL>

Context Livir::operacaoX()
  post:
    <expressão 1> AND
    <expressão 2> AND
       ...
    <expressão n>
```

Tipos de pós-condições

- Modificação de valor de atributo.
- Criação de instância.
- Criação de associação.
- Destruição de instância.
- Destruição de associação.

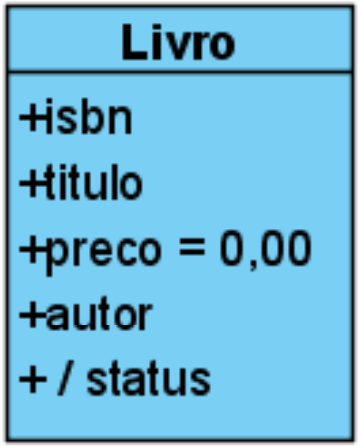
Operações básicas

- São aquelas que em orientação a objetos operam diretamente sobre os elementos básicos do modelo conceitual.
- Seu significado e comportamento são definidos por padrão.
- Elas realizam as pós-condições.

Modificação de valor de atributo

```
pessoa^setDataNascimento(novaData)
```

Criação de instância




```
Livro::newInstance()
```

Atributos com valores iniciais e atributos derivados

- Assume-se que atributos com valores iniciais (cláusula init) já sejam definidos automaticamente pela operação de criação (sem necessidade de especificar novamente pós-condições para dar valor a estes atributos).
- Atributos derivados são calculados e não podem ser modificados diretamente.

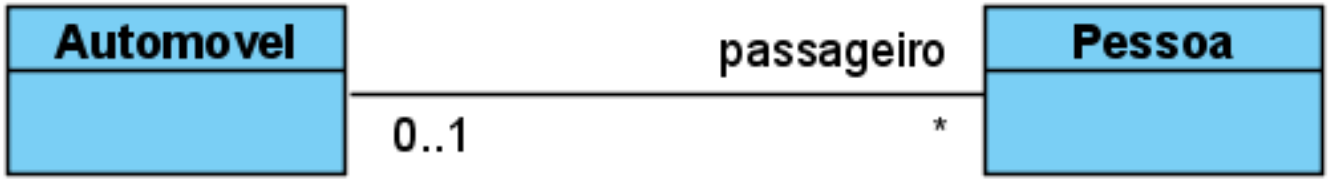
Demais atributos

- Há dois padrões:
  - A operação básica de criação de instância simplesmente produz a instância, sem inicializar seus atributos e associações obrigatórias.
    - Neste caso a checagem de consistência é feita no nível da operação de sistema, e não no nível da operação básica.
  - A operação básica de criação de instância inicializa atributos e associações obrigatórias de forma que a instância não fique inconsistente em relação ao modelo conceitual.
    - Neste caso, a operação básica já produz uma instância consistente.

Aplicando o primeiro padrão

```
Context Livir::criaLivro(umIsbn, umTitulo, umAutor)

  def: novoLivro =
    Livro::newInstance()

  post:
     ...
    novoLivro^setIsbn(umIsbn) AND
    novoLivro^setTitulo(umTitulo) AND
    novoLivro^setAutor(umAutor)
```

Criação de associação




```
jipe^addPassageiro(joao)

Ou:

joao^addAutomovel(jipe)
```

Criação de instância e associação

- É necessário que uma instância recém criada seja associada a outra instância.
- Pode ser feito assim:

```
venda^addPagamento(Pagamento::newInstance())
```

Retornando a um exemplo anterior:

```
Context Livir::criaLivro(umIsbn, umTitulo, umAutor)

  def: novoLivro =
    Livro::newInstance()

  post:
    self^addLivro(novoLivro) AND
    novoLivro^setIsbn(umIsbn) AND
    novoLivro^setTitulo(umTitulo) AND
    novoLivro^setAutor(umAutor)
```

Destruição de instância

- Duas abordagens:
  - *Explícita*: declara-se que um objeto foi destruído através do envio de uma mensagem explícita de destruição.
  - *Implícita*: removem-se todas as associações para o objeto de forma que ele passe a não ser mais acessível.
    - Em linguagens de programação é possível implementar coletores de lixo (*garbage collection*) para remover da memória objetos que não são mais acessíveis.

Abordagem explícita

```
objeto^destroy()
```

Assume-se que todas as associações deste objeto também são destruídas com ele

Destruição de associação

```
v^removePagamento(p1)
Ou:
p1^removeVenda(v)
```

Duas conseqüências possíveis:

1. O pagamento p1 também é destruído.
2. O pagamento p1 é associado a outra venda.

Multiplicidade 1 ou 0..1 não exige parâmetro


```
p1^removeVenda()
```

Pós-condições bem formadas

- Considerando-se que as operações básicas mais elementares não comportam checagem de consistência nos objetos em relação ao modelo conceitual, o conjunto de pós-condições precisa ser verificado para se saber se ao final da execução das operações os objetos estão em um estado consistente com as definições do modelo.

Regra 1

- Uma instância recém criada deve ter pós-condições indicando que todos seus atributos foram inicializados, exceto:
  - Atributos derivados (que são calculados).
  - Atributos com valor inicial (que já são definidos por uma cláusula *init* no contexto da classe e não precisam ser novamente definidos para cada operação de sistema).
  - Atributos que possam ser nulos (neste caso a inicialização é opcional).

Regra 2

- Uma instância recém criada deve ter sido associada a alguma outra que por sua vez possua um caminho de associações que permita chegar à controladora de sistema.
- Caso contrário ela é inacessível e não faz sentido criar um objeto que não possa ser acessado por outros.

Regra 3

- Todas as associações afetadas por criação ou destruição de instância ou associação devem estar com seus papeis dentro dos limites inferior e superior.

Regra 4

- Todas as invariantes afetadas por alterações em atributos, associações ou instâncias devem continuar sendo verdadeiras.

@pre

- vendaCorrente^setSaldo(vendaCorrente.saldo@pre+1)

Combinações de pós-condições

- AND
- OR
- IMPLIES

Exemplo de IMPLIES

```
post:
  self.vendaCorrente.saldo@pre = 0 IMPLIES
  self.vendaCorrente^setSaldo(1)
```

Equivalente a:

```
post:
  if vendaCorrente.saldo@pre = 0 then
    vendaCorrente^setSaldo(1)
  endIf
```

Pós-condições sobre coleções de objetos

```
Context Livir::reduzPrecoLivros(x)

  post:
    self.livros→forAll(livro|
      livro^setPreco(livro.preco@pre * (100-x)/100)
    )



self.livros^setPreco(100)
```

Exceções

- As exceções em contratos são estabelecidas como situações de falha que não podem ser evitadas ou verificadas antes de iniciar a execução da operação propriamente dita.

Exemplo

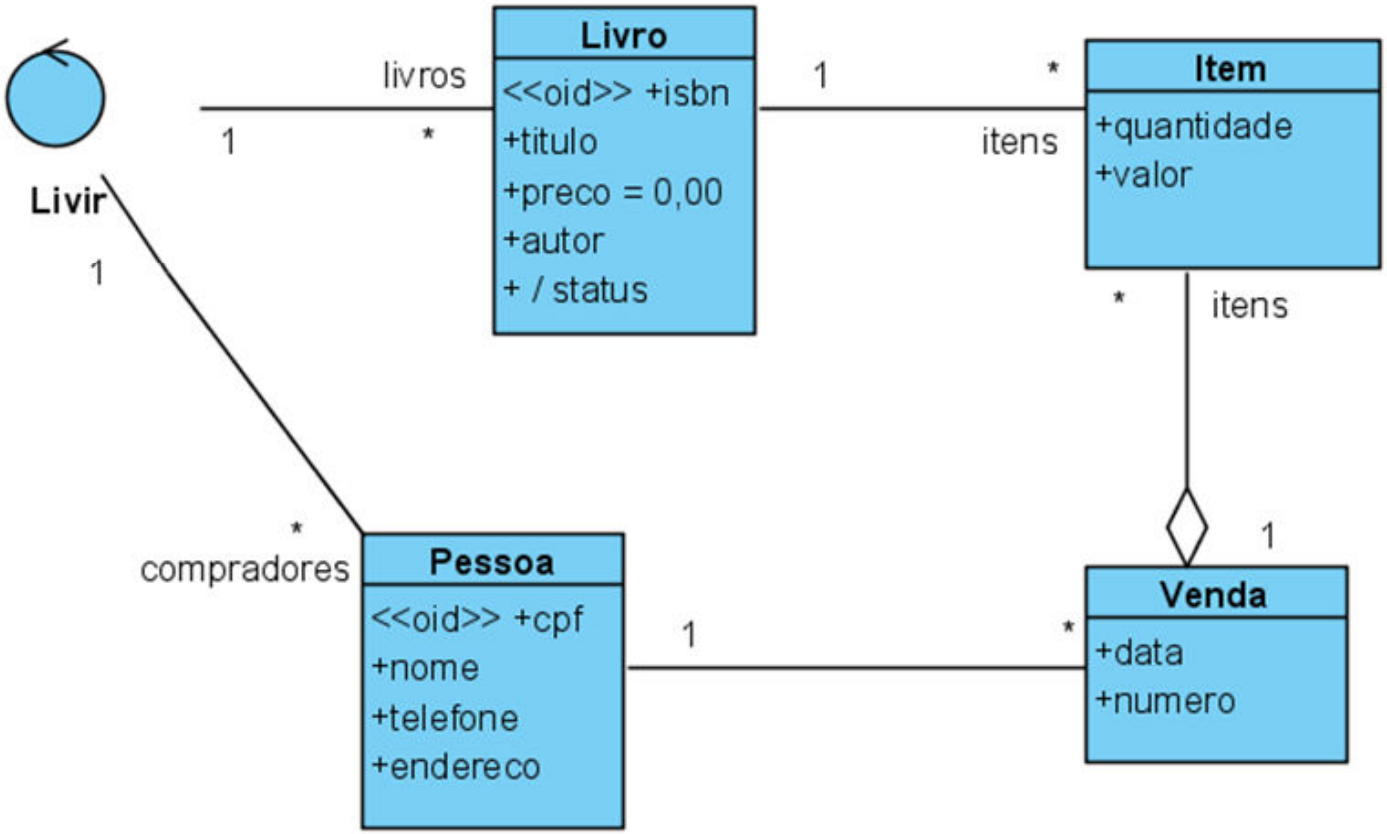
```
Context Livir::identificaComprador(umCpf)

  def: comprador =
    compradores→select(cpf = umCpf)

  post:
    self^addCompradorCorrente(comprador)

  exception:
    comprador→size() = 0 IMPLIES
      self^throw("cpf inválido")
```

Conversão em pré-condição

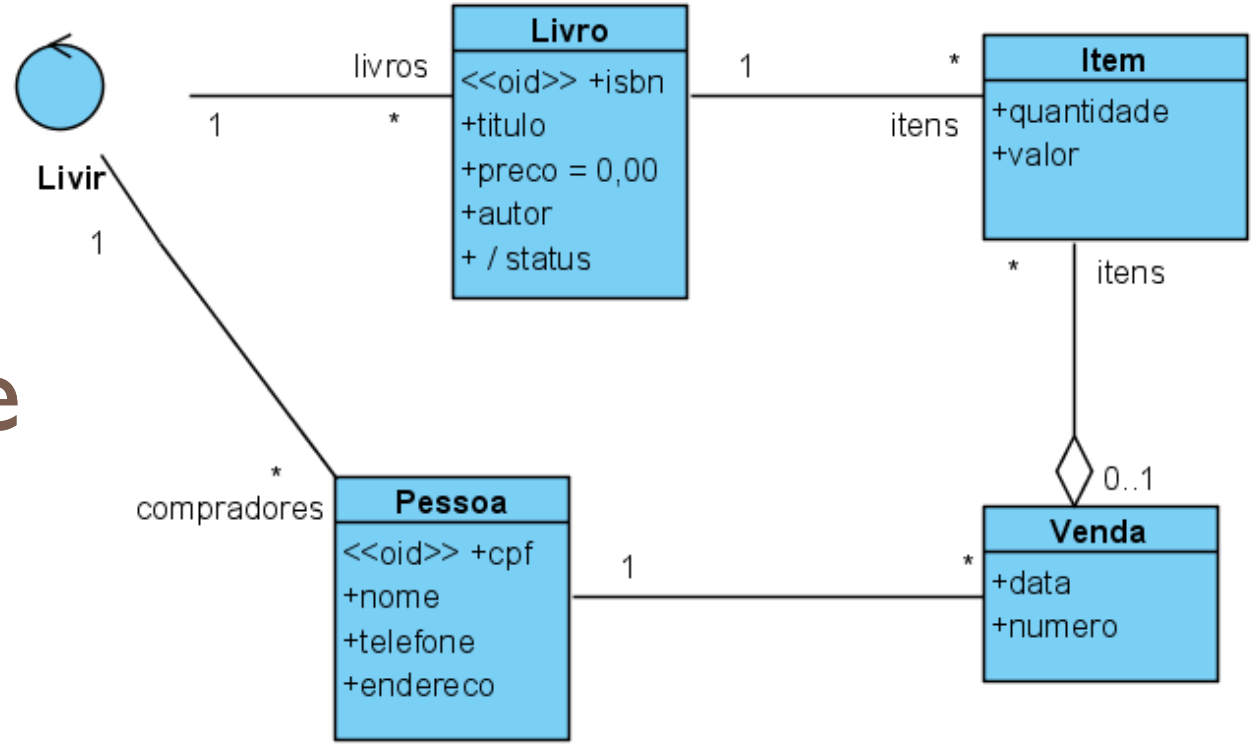
```
Context Livir::identificaComprador(umCpf)

  def: comprador =
    compradores→select(cpf = umCpf)

  pre:
    comprador→size() = 1

  post:
    self.addCompradorCorrente(comprador)
```

Contratos padrão CRUD

- Modelo de referência



Create




```
Context Livir::criaLivro(umIsbn, umTitulo, umAutor)

  def: novoLivro =
    Livro::newInstance()

  post:
    self^addLivros(novoLivro) AND
    novoLivro^setIsbn(umIsbn) AND
    novoLivro^setTitulo(umTitulo) AND
    novoLivro^setAutor(umAutor)
```

ISBN não pode ser repetido

```
Context Livir criaLivro(umIsbn, umTitulo, umAutor)

  def: novoLivro =
    Livro::newInstance()

  pre:


  post:
    self^addLivros(novoLivro) AND
    novoLivro^setIsbn(umIsbn) AND
    novoLivro^setTitulo(umTitulo) AND
    novoLivro^setAutor(umAutor)

  exception:
    livros→select(isbn=umIsbn)→notEmpty() IMPLIES
      self^throw("Este ISBN já está cadastrado")
```

Caso o ISBN seja garantidamente novo:

```
Context Livir criaLivro(umIsbn, umTitulo, umAutor)

  def: novoLivro =
    Livro::newInstance()

  pre:
    livros→select(isbn=umIsbn)→isEmpty()

  post:
    self^addLivros(novoLivro) AND
    novoLivro^setIsbn(umIsbn) AND
    novoLivro^setTitulo(umTitulo) AND
    novoLivro^setAutor(umAutor)
```

Update

```
Context Livir alteraLivro(umIsbn, novoIsbn, umTitulo, umAutor)

  def: livro =
    livros→select(isbn=umIsbn)

  pre:
    livro→size()=1 AND
    livros->select(isbn=novoIsbn)->isEmpty()

  post:
    livro^setIsbn(novoIsbn) AND
    livro^setTitulo(umTitulo) AND
    livro^setAutor(umAutor)
```

Opções com atributos <<oid>>

- *Não se permite que sejam alterados.*
  - O objeto deve ser destruído e um novo objeto criado.

OU:

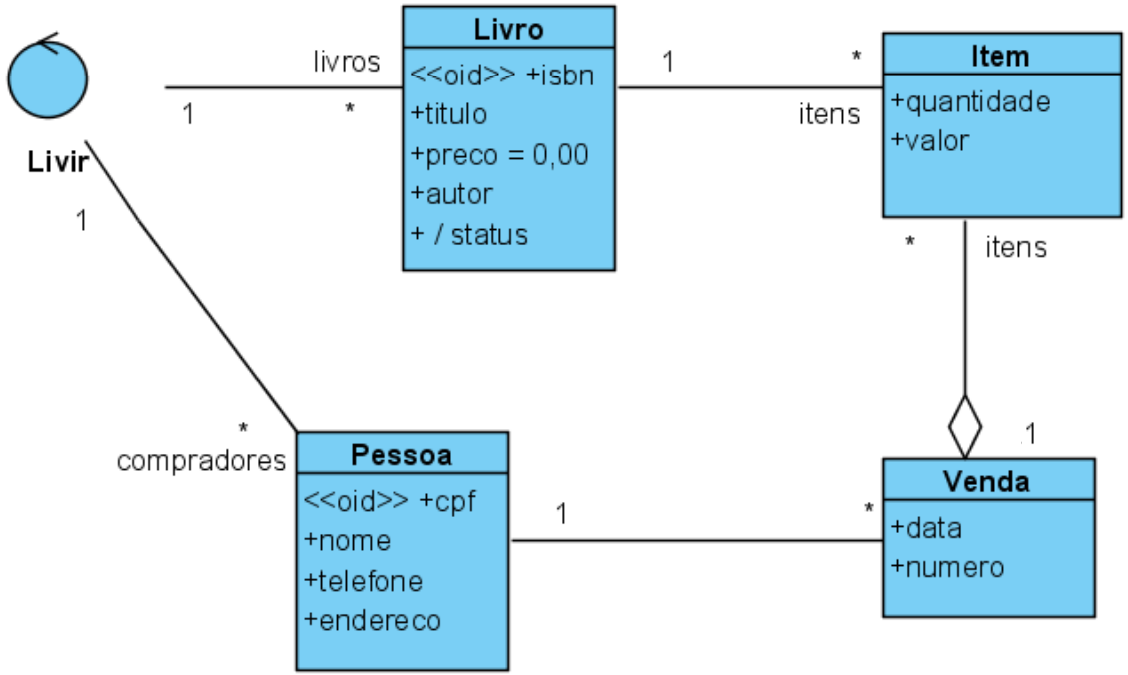
- *Permite-se a alteração.*
  - Neste caso, a operação passa dois argumentos: o ISBN anterior e o novo, como foi feito no exemplo acima.
  - Se o novo ISBN corresponder a um livro já existente haverá uma exceção porque este atributo foi marcado como oid.

Delete

- As operações de sistema que incluem exclusão de informações vão ter que considerar as regras de restrição estrutural do modelo conceitual antes de decidir se um objeto pode ou não ser destruído e, caso possa, verificar se outros objetos também devem ser destruídos junto com ele.

Exemplo

- Para excluir um livro deve-se determinar o que acontece com os itens que tem associação obrigatória para ele



Abordagens

- Garantir por *pré-condição* que o livro sendo excluído não possui nenhum item ligado a ele.
  - A associação do ponto de vista do livro é opcional.
  - Por esta abordagem, apenas livros que não tem itens associados podem ser selecionados para exclusão.
- Garantir por *pós-condição* que todos os itens ligados ao livro também serão excluídos.
  - Usa-se essa abordagem quando se quer que a operação de exclusão se propague para todos os objetos (no caso itens) que tem associações obrigatórias com o livro.
  - Essa propagação continua atingindo outros objetos em cadeia até que não haja mais ligações baseadas em associações obrigatórias.
- Utilizar uma *exceção* para abortar a operação caso seja tentada sobre um livro que tenha itens associados a ele.

Exclusão com a abordagem "pré"

```
Context Livir::excluiLivro(umIsbn)

  def: livro =
    livros→select(isbn=umIsbn)

  pre:
    livro→size() = 1 AND
    livro.itens→isEmpty()

  post:
    livro^destroy()
```

Exclusão com a abordagem "pós"

```
Context Livir::excluiLivro(umIsbn)

  def: livro =
    livros→select(isbn=umIsbn)

  pre:
    livro→size() = 1

  post:
    livro.itens^destroy() AND
    livro^destroy()
```

Exclusão com abordagem "exceção"

```
Context Livir::excluiLivro(umIsbn)

  def: livro =
    livros→select(isbn=umIsbn)

  pre:
    livro→size() = 1

  post:
    livro^destroy()

  exception:
    livro.itens→notEmpty() IMPLIES
      self^throw("não é possível excluir este livro")
```

Retrieve

```
Context Livir::consultaLivro(umIsbn):Tuple

  def: livro =
    livros→select(isbn = umIsbn)

  body:
    Tuple{
      isbn = livro.isbn,
      titulo = livro.titulo,
      preco = livro.preco,
      autor = livro.autor,
      status = livro.status
    }
```

Contrato padrão “listagem”

- Freqüentemente é necessário utilizar operações de listagem de um ou mais atributos de um conjunto de instâncias de uma determinada classe para preencher listas ou menus em interfaces.

Ex. ISBN dos livros catalogados

```
Context Livir::listaIsbn():Set

  body:
    livros.isbn
```

Ex. Lista múltipla (ISBN e título)

```
Context Livir::listaIsbnTitulo():Set

  body:
    self.livros→collect(livro|
      Tuple{
        isbn = livro.isbn,
        titulo = livro.titulo
      }
    )
```

Ex. Lista com filtro

```
Context Livir::listaIsbnTituloNaoVendidos():Set

  body:
    self.livros→select(livro|
      livro.itens→isEmpty()
    )→ collect(livro|
      Tuple{
        isbn = livro.isbn,
        titulo = livro.titulo
      }
    )
```

Contratos e casos de uso

- Para as operações associadas com casos de uso, freqüentemente haverá uma cadeia de execução ao longo de um dos fluxos, onde duas ou mais operações de sistema serão executadas.
- Possivelmente cada operação poderá deixar informações para as demais na forma de associações temporárias.
- Para melhor construir os contratos destas operações uma abordagem possível é seguir a seqüência das operações de sistema do caso de uso expandido.

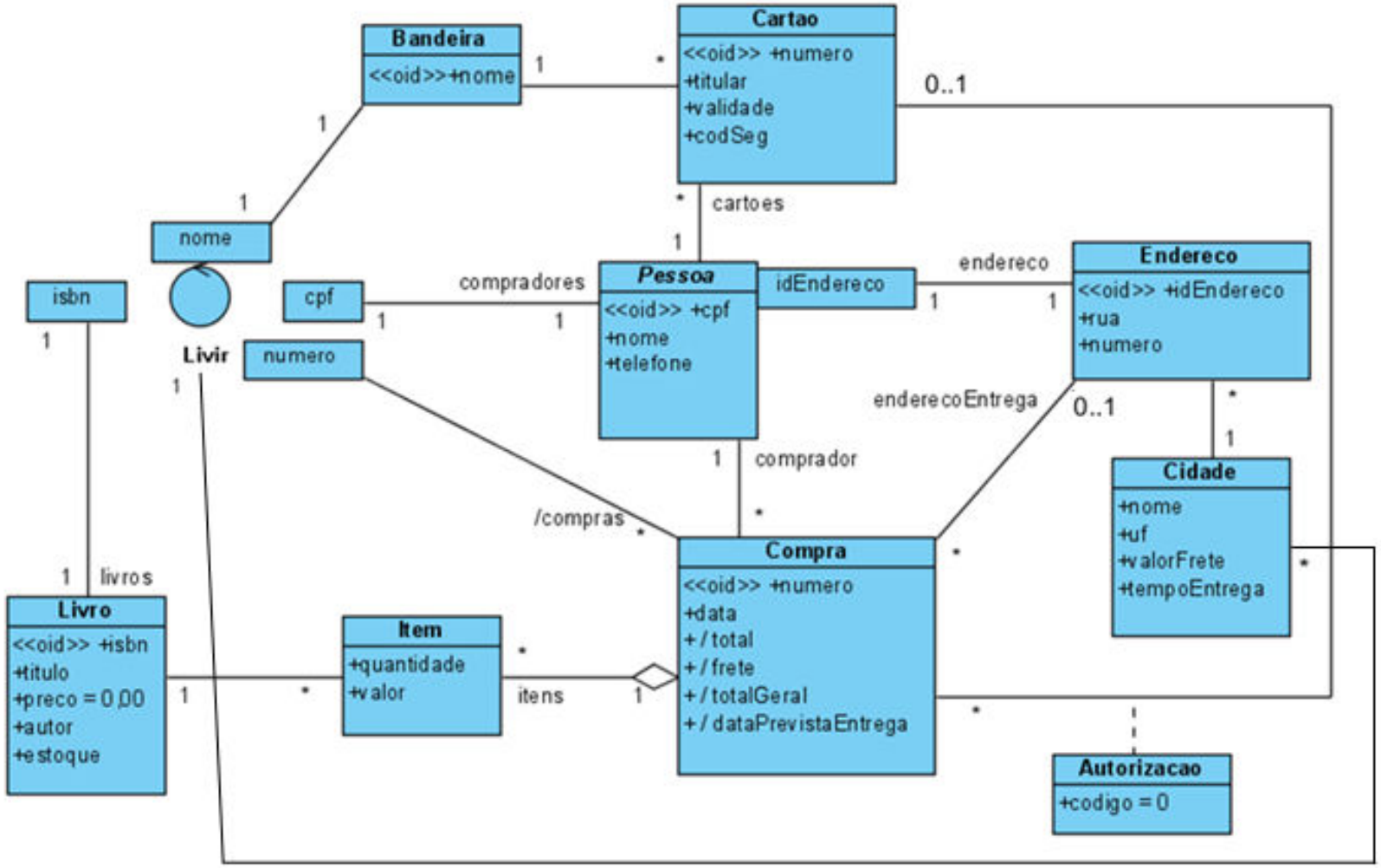
Coisas a perguntar:

- Qual é o objetivo de cada operação?
- O que cada uma delas produz?
- O que cada uma delas espera que tenha sido produzido pelas anteriores?
- Que exceções poderiam ocorrer durante a execução?
- A operação segue algum padrão como *CRUD*, Listagem ou *REP*?

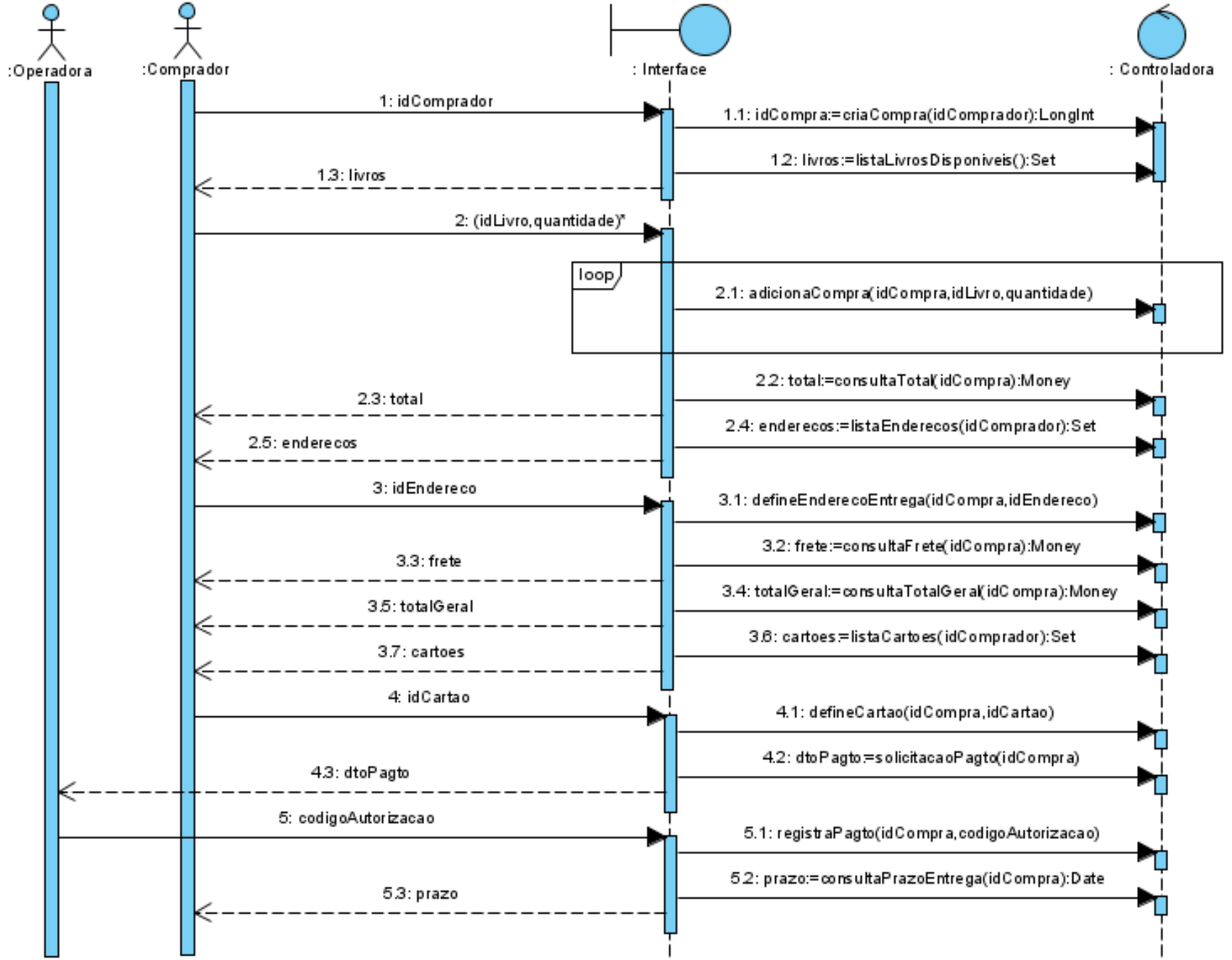
Exemplos

- Stateless
- Statefull

Modelo conceitual de referência stateless



Stateless


:Operadora
:Comprador
: Interface
: Controladora
1: idComprador
1.1: idCompra:=criaCompra(idComprador):LongInt
1.2: livros:=listaLivrosDisponiveis():Set
1.3: livros
2: (idLivro,quantidade)*
loop
2.1: adicionaCompra(idCompra,idLivro,quantidade)
2.2: total:=consultaTotal(idCompra):Money
2.3: total
2.4: enderecos:=listaEnderecos(idComprador):Set
2.5: enderecos
3: idEndereco
3.1: defineEnderecoEntrega(idCompra,idEndereco)
3.2: frete:=consultaFrete(idCompra):Money
3.3: frete
3.4: totalGeral:=consultaTotalGeral(idCompra):Money
3.5: totalGeral
3.6: cartoes:=listaCartoes(idComprador):Set
3.7: cartoes
4: idCartao
4.1: defineCartao(idCompra,idCartao)
4.2: dtoPagto:=solicitacaoPagto(idCompra)
4.3: dtoPagto
5: codigoAutorizacao
5.1: registraPagto(idCompra,codigoAutorizacao)
5.2: prazo:=consultaPrazoEntrega(idCompra):Date
5.3: prazo

Operações stateless

- `criaCompra(idComprador):LongInt`
- `listaLivrosDisponiveis():Set`
- `adicionaCompra(idCompra, idLivro, quantidade)`
- `consultaTotal(idCompra):Money`
- `listaEnderecos(idComprador):Set`
- `defineEnderecoEntrega(idCompra, idEndereco)`
- `consultaFrete(idCompra):Money`
- `consultaTotalGeral(idCompra):Money`
- `listaCartoes(idComprador):Set`
- `defineCartao(idCompra,idCartao)`
- `solicitacaoPagto(idCompra):Tuple`
- `registraPagto(idCompra, codigoAutorizacao)`
- `consultaPrazoEntrega(idCompra):Date`

```
Context Livir::criaCompra(idComprador):LongInt

  def: novaCompra =
    Compra::newInstance()

  def: comprador =
    compradores[idComprador]

  post:
    novaCompra^setNumero(System::novoNumeroAutomatico()) AND
    novaCompra^setData(System::dataAtual()) AND
    novaCompra^addComprador(comprador) AND
    return: novaCompra.numero

  exception:
    comprador→size() = 0 IMPLIES
      self^throw("Comprador não cadastrado")
```

```
Context Livir::listaLivrosDisponiveis():Set

  body:
    livros→select(
      estoque>0
    )→collect(livro|
       Tuple{
         isbn = livro.isbn,
         titulo = livro.titulo,
         preco = livro.preco,
         autor = livro.autor
       }
    )
```

```
Context Livir::adicionaCompra(idCompra, idLivro, quantidade)

  def: compra =
    compras[idCompra]

  def: livro =
    livros[idLivro]

  def: item =
    Item::newInstance()

  pre: compra→size() = 1 AND
       livro→size() = 1

  post:
    item^setQuantidade(quantidade) AND
    item^setValor(livro.preco) AND
    item^addCompra(compra) AND
    item^addLivro(livro) AND
    livro^setEstoque(livro.estoque@pre - quantidade)

  exception:
    quantidade > livro.estoque IMPLIES
      self^throw("quantidade insuficiente em estoque")
```

```
Context Livir::consultaTotal(idCompra):Money

  pre:
    compras[idCompra]→size()=1

  body:
    compras[idCompra].total
```

```
Context Livir::listaEnderecos(idComprador):Set

  def: comprador =
    compradores[idComprador]

  pre:
    comprador→size() = 1

  body:
    comprador.enderecos→collect(endereco|
      Tuple {
        id = endereco.idEndereco,
        rua = endereco.rua,
        numero = endereco.numero,
        cidade = endereco.cidade.nome,
        uf = endereco.cidade.uf
      }
    )
```

```
Context Livir::defineEnderecoEntrega(idCompra, idEndereco)

  def: compra =
    compras[idCompra]

  def: comprador =
    compra.comprador

  def: endereco =
    comprador.enderecos[idEndereco]

  pre:
    compra→size()=1 AND
    endereco→size()=1

  post:
    compra^addEnderecoEntrega(endereco)
```

```
Context Livir::consultaFrete(idCompra):Money

  pre:
    compras[idCompra]→size() = 1

  body:
    compras[idCompra].frete
```

```
Context Livir::consultaTotalGeral(idCompra):Money

  pre:
    compras[idCompra]→size() = 1

  body:
    compras[idCompra].totalGeral
```

```
Context Livir::listaCartoes(idComprador):Set

  pre:
    compradores[idComprador]→size() = 1

  body:
    compradores[idComprador].cartoes→collect(cartao|
      Tuple {
        bandeira = cartao.bandeira.nome,
        numero = cartao.numero
      }
    )
```

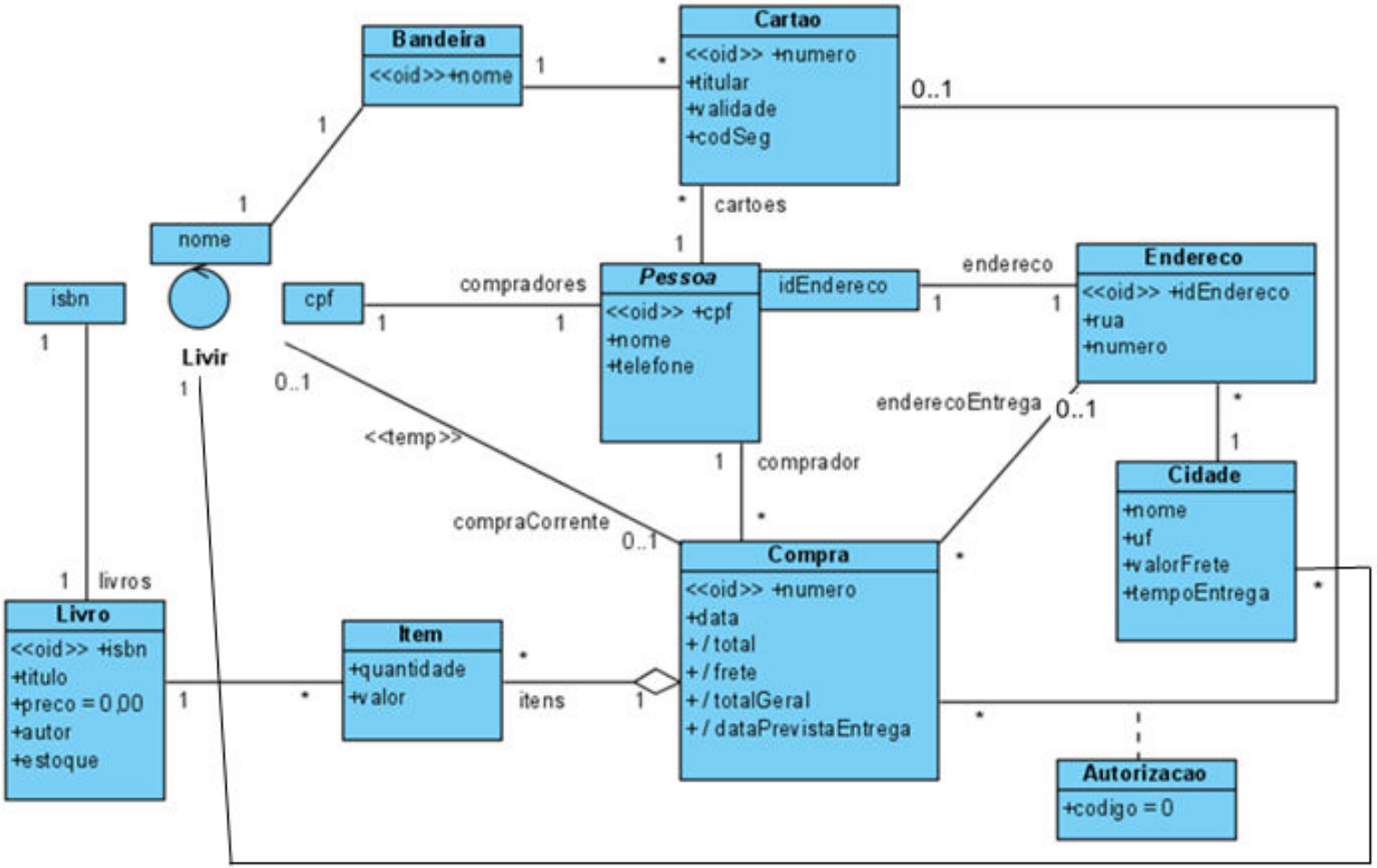
```
Context Livir::defineCartao(idCompra,idCartao)

def: compra =
  compras[idCompra]

def: cartao =
  compra.comprador.cartoes→select(numero=idCartao)

pre:
  compra→size()=1 AND
  cartao→size()=1

post:
  compra^addCartao(cartao)
```

```
Context Livir::solicitacaoPagto(idCompra):Tuple

  def: compra =
    compras[idCompra]

  pre:
    compra→size() = 1

  body:
    Tuple {
      numero = compra.cartao.numero,
      titular = compra.cartao.titular,
      validade = compra.cartao.validade,
      codSeg = compra.cartao.codSeg,
      valor = compra.valorTotal
    }
```

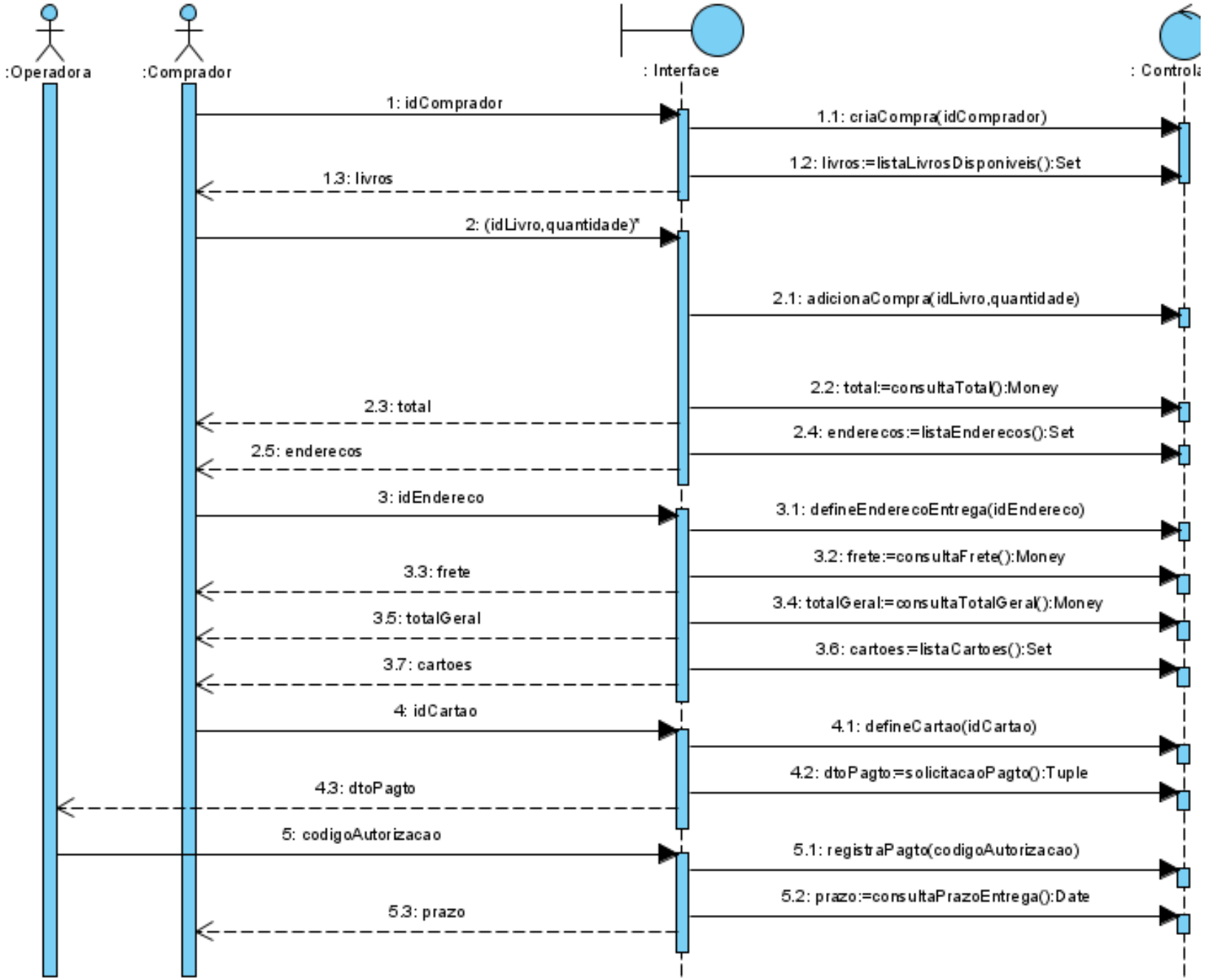
```
Context Livir::registraPagto(codigoAutorizacao, idCompra)

  def: compra =
    compras[idCompra]

  pre:
    compra→size() = 1

  post:
    compra.autorizacao^setCodigo(codigoAutorizacao)
```

```
Context Livir::consultaPrazoEntrega(idCompra):Date

  pre:
    compras[idCompra]→size() = 1

  body:
    compras[idCompra].enderecoEntrega.cidade.tempoEntrega
```

Modelo conceitual de referência para statefull



Statefull


:Operadora
:Comprador
: Interface
: Controla
1: idComprador
1.1: criaCompra(idComprador)
1.2: livros:=listaLivrosDisponiveis():Set
1.3: livros
2: (idLivro,quantidade)*
2.1: adicionaCompra(idLivro,quantidade)
2.2: total:=consultaTotal():Money
2.3: total
2.4: enderecos:=listaEnderecos():Set
2.5: enderecos
3: idEndereco
3.1: defineEnderecoEntrega(idEndereco)
3.2: frete:=consultaFrete():Money
3.3: frete
3.4: totalGeral:=consultaTotalGeral():Money
3.5: totalGeral
3.6: cartoes:=listaCartoes():Set
3.7: cartoes
4: idCartao
4.1: defineCartao(idCartao)
4.2: dtoPagto:=solicitacaoPagto():Tuple
4.3: dtoPagto
5: codigoAutorizacao
5.1: registraPagto(codigoAutorizacao)
5.2: prazo:=consultaPrazoEntrega():Date
5.3: prazo

Operações statefull

- `criaCompra(idComprador)`
- `listaLivrosDisponiveis():Set`
- `adicionaCompra(idLivro, quantidade)`
- `consultaTotal():Money`
- `listaEnderecos():Set`
- `defineEnderecoEntrega(idEndereco)`
- `consultaFrete():Money`
- `consultaTotalGeral():Money`
- `listaCartoes():Set`
- `definecartao(idCartao)`
- `solicitacaoPagto():Tuple`
- `registraPagto(codigoAutorizacao)`
- `consultaPrazoEntrega():Date`

```
Context Livir::criaCompra(idComprador)

  def: novaCompra =
    Compra::newInstance()

  def: comprador =
    compradores[idComprador]

  post:
    novaCompra^setNumero(System::novoNumeroAutomatico()) AND
    novaCompra^setData(System::dataAtual()) AND
    novaCompra^addComprador(comprador) AND
    self^addCompraCorrente(novaCompra)

  exception:
    comprador→size() = 0 IMPLIES
      self^throw("Comprador não cadastrado")
```

```
Context Livir::adicionaCompra(idLivro, quantidade)

  def: livro =
    livros[idLivro]

  def: item =
    Item::newInstance()

  pre:
    livro→size()=1 AND
    compraCorrente→size()=1

  post:
    item^setQuantidade(quantidade) AND
    item^setValor(livro.preco) AND
    item^addCompra(compraCorrente) AND
    item^addLivro(livro) AND
    livro^setEstoque(livro.estoque@pre - quantidade)

  exception:
    quantidade>livro.estoque IMPLIES
      self^throw("quantidade insuficiente em estoque")
```

```
Context Livir::consultaTotal():Money

  pre:
    compraCorrente→size() = 1

  body:
    compraCorrente.total
```

```
Context Livir::listaEnderecos():Set

  pre:
    compraCorrente→size() = 1

  body:
    compraCorrente.comprador.enderecos→collect(endereco|
      Tuple {
        id = endereco.idEndereco,
        rua = endereco.rua,
        numero = endereco.numero,
        cidade = endereco.cidade.nome,
        uf = endereco.cidade.uf
      }
    )
```

```
Context Livir::defineEnderecoEntrega(idEndereco)

  def: endereco =
    compraCorrente.comprador.enderecos[idEndereco]

  pre:
    endereco→size() = 1 AND
    compraCorrente→size() = 1

  post:
    compraCorrente^addEnderecoEntrega(endereco)
```

```
Context Livir::consultaFrete():Money

  pre:
    compraCorrente→size() = 1

  body:
    compraCorrente.frete
```

```
Context Livir::consultaTotalGeral():Money

  pre:
    compraCorrente→size() = 1

  body:
    compraCorrente.totalGeral
```

```
Context Livir::listaCartoes():Set

  pre:
    compraCorrente→size() = 1

  body:
    compraCorrente.comprador.cartoes→collect(cartao|
      Tuple {
        bandeira = cartao.bandeira.nome,
        numero = cartao.numero
      }
    )
```

```
Context Livir::definecartao(idCartao)

  def: cartao =
    compraCorrente.comprador.cartoes
      →select(numero=idCartao)

  pre:
    cartao→size() = 1 AND
    compraCorrente→size() = 1

  post:
    compraCorrente^addCartao(cartao)
```

```
Context Livir::solicitacaoPagto():Tuple

  pre:
    compraCorrente→size() = 1 AND
    compraCorrente.cartao->size() = 1
  body:
    Tuple {
      numero = compraCorrente.cartao.numero,
      titular = compraCorrente.cartao.titular,
      validade = compraCorrente.cartao.validade,
      codSeg = compraCorrente.cartao.codSeg
    }
```

```
Context Livir::registraPagto(codigoAutorizacao)

  pre:
    compraCorrente→size()=1 AND
    compraCorrente.cartao->size()=1

  post:
    compraCorrente.autorizacao^setCodigo(codigoAutorizacao)
```

```
Context Livir::consultaPrazoEntrega():Date

  pre:
    compraCorrente→size() = 1

  body:
    compraCorrente.enderecoEntrega.cidade.tempoEntrega
```