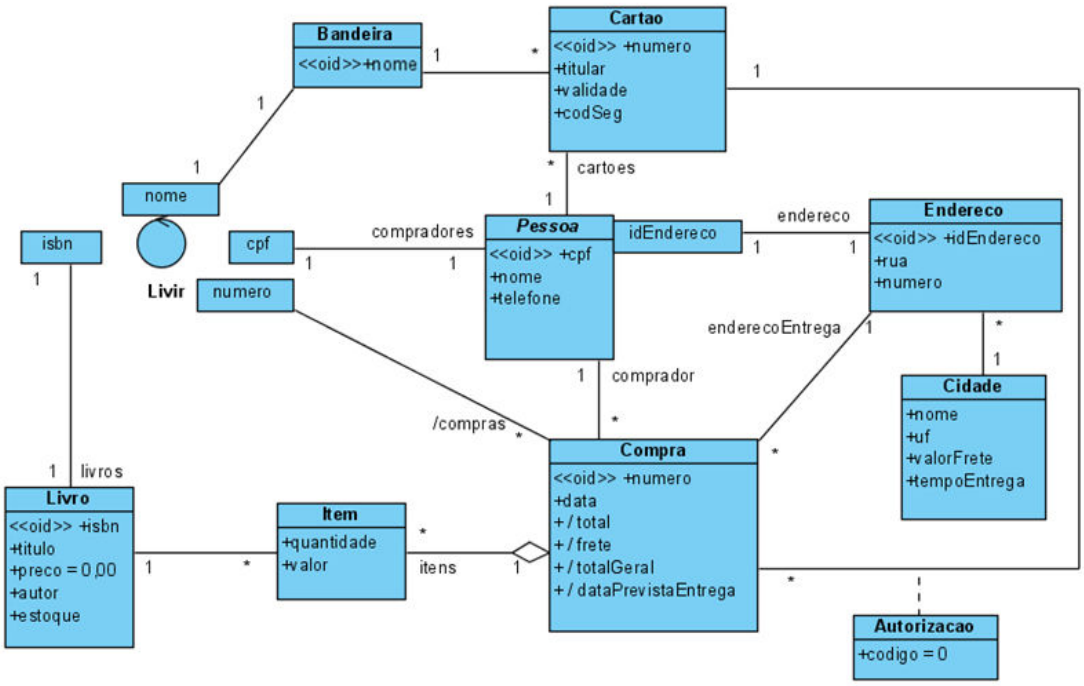
Exercícios de Especificação Formal com OCL

Prof. Raul Sidnei Wazlawick

Considere o seguinte modelo conceitual como base:



1. Escreva uma expressão OCL que defina o atributo derivado “frete” na classe “Compra” como sendo o “valorFrete” da cidade do endereço da compra.

```
Context Compra::frete
    derive:
      self.enderecoEntrega.cidade.valorFrete
```

2. Escreva uma expressão OCL que defina a associação derivada “livrosCarosComprados” entre a classe Pessoa e a classe Livro. A associação é definida como o conjunto dos livros existentes nas compras da pessoa que custaram mais de 100 reais cada.

```
Context Pessoa::livrosCarosComprados
  derive:
    self.compra.itens.livro->select(l|l.preco>100)
```

3. Escreva um contrato OCL para uma consulta de sistema que retorna a lista com os nomes e uf de todas as cidades onde há clientes cadastrados juntamente com o valor do frete e a quantidade de compras já efetuadas para cada cidade. A consulta deve produzir, portanto, uma lista onde cada elemento é uma tupla com nome, uf, valor do frete e quantidade de compras efetuadas para uma determinada cidade. A consulta não tem parâmetros nem pré-condições.

```
Context Livir::consultaCidadeFrete():Set
  body:
    self.compradores.endereco.cidade->collect(c|
      Tuple{
        nome = c.nome,
        uf = c.uf,
        frete = c.valorFrete,
        quantidadeCompras = c.endereco.compra->size()
      }
    )
```

4. Escreva um contrato OCL para uma operação de sistema que cadastre um novo cartão para uma pessoa. Os parâmetros passados são o nome da bandeira, número, titular, código de segurança (todos do tipo string) e validade do cartão (tipo Data) e o CPF da pessoa (tipo CPF). Assuma como pré-condição que a pessoa e a bandeira já estão cadastradas e que o número do cartão ainda não está cadastrado.

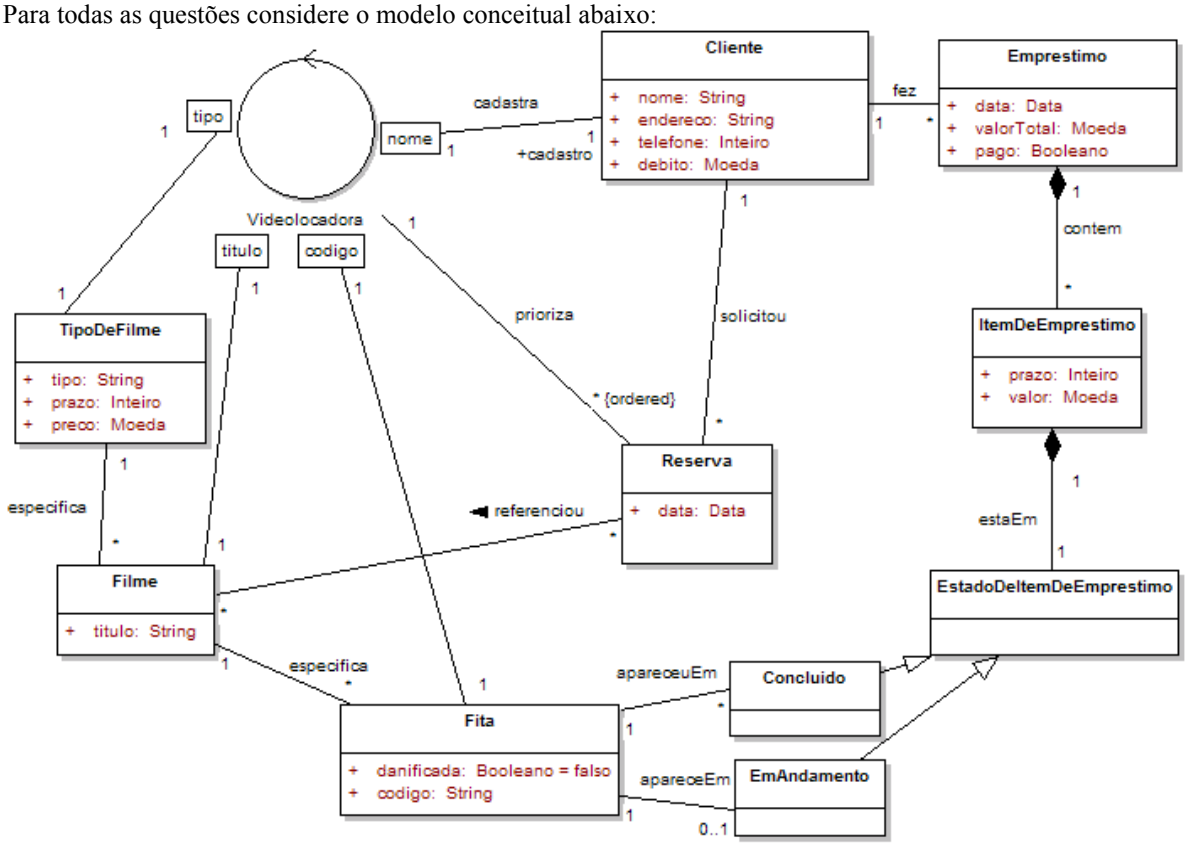
```
Context Livir:cadastraCartao(umCpf:CPF, bandeira, umNumero, umTitular,
umCodSeg:string, umaValidade:Data)
  def: novoCartao = Cartao::newInstance
  def: band = self.bandeira[umaBandeira]
  def: pessoa = self.compradores[umCpf]
  pré:
    pessoa->notEmpty() AND
    bandeira->notEmpty() AND
    self.bandeira.cartao->select(c|
      c.numero = umNumero
    )->isEmpty()
  post:
    novoCartao^setNumero(umNumero) AND
    novoCartao^setTitular(umTitular) AND
    novoCartao^setValidade(umaValidade) AND
    novoCartao^setCodSeg(umCodSeg) AND
    band^addCartao(novoCartao) AND
    novoCartao^addPessoa(pessoa)
```

5. Escreva uma expressão OCL para uma invariante que defina que a data de validade do cartão de uma compra (se existir) deve ser posterior à data da compra (considere que as datas podem ser comparadas com os operadores “=”, “>”, “<”).

```
Context Compra
 inv:
   self.cartao->notEmpty() IMPLIES
     self.cartao.validade > self.data
```

Opção:	a	e	c	c	a	b
Acertos	68%	84%	89%	76%	81%	89%

Para todas as questões considere o modelo conceitual abaixo:



1. O valor inicial do atributo “valor” da classe ItemDeEmprestimo deve ser definido como o valor do tipo de filme correspondente. Qual expressão OCL abaixo representa corretamente este valor inicial?

    a. Context ItemDeEmprestimo::valor
    init:
    self.estadoDeItemDeEmprestimo.fita.filme.tipoDeFilme.preco

    b. Context ItemDeEmprestimo::valor
    init:
    self.emprestimo.cliente.videolocadora.tipoDeFilme.preco

    c. Context ItemDeEmprestimo::valor
    init:
    self.estadoDeItemDeEmprestimo.emAndamento.fita.filme.tipoDeFilme.preco

    d. Context ItemDeEmprestimo::valor
    init:
    self.estadoDeItemDeEmprestimo.concluido.fita.filme.tipoDeFilme.preco

    e. Context ItemDeEmprestimo::valor
    init:
    self.tipoDeFilme.preco

2. O atributo “debito” da classe Cliente será definido como atributo derivado. O débito de um cliente é igual ao somatório do valor total de todos os itens de emprestimo que estão em andamento para este cliente. Qual expressão OCL abaixo representa corretamente este atributo derivado? (a função “isKindOf” retorna true se o objeto receptor é instância da classe passada como parâmetro ou uma de suas subclasses)

    a.
```
Context Cliente::debito
 derive:
  self.emprestimo.itemDeEmprestimo.estadoDeItemDeEmprestimo->select(e|
   e.isKindOf(EmAndamento)
  )->sum(item|
   item.valor
  )
```

    b.
```
Context Cliente::debito
 derive:
  self.emprestimo.itemDeEmprestimo->sum(item|
   item.valor
  )
```

    c.
```
Context Cliente::debito
 derive:
  self.emprestimo.itemDeEmprestimo->sum(item|
   item.estadoDeItemDeEmpresimo.isKindOf(EmAndamento).valor
  )
```

    d.
```
Context Cliente::debito
 init:
  self.emprestimo.itemDeEmprestimo->sum(item|
   item.valor
  )
```

    e.
```
Context Cliente::debito
 derive:
  self.emprestimo.itemDeEmprestimo->select(item|
   item.estadoDeItemDeEmpresimo.isKindOf(EmAndamento)
  )->sum(item|
   item.valor
  )
```

3. Uma associação derivada entre Videolocadora e Cliente deve indicar quais são os clientes vip. Um cliente vip é aquele que já realizou mais de 100 empréstimos (independentemente do número de itens). Qual expressão OCL abaixo corretamente define esta associação derivada?

    a.
```
Context Videolocadora::clientesVip
 derive:
  self.cadastro.emprestimo->select(cliente|
   cliente.emprestimo->size()>=100
  )
```

    b.
```
Context Videolocadora::clientesVip
 derive:
  self.cadastro->select(emprestimo|
   emprestimo->size()>=100
  )
```

    c.
```
Context Videolocadora::clientesVip
 derive:
  self.cadastro->select(cliente|
   cliente.emprestimo->size()>=100
  )
```

    d.
```
Context Cliente::clientesVip
 derive:
  self ->select(cliente|
   cliente.emprestimo->size()>=100
  )
```

e. Context Videolocadora::clientesVip
derive:
self.cadastro.emprestimo->size()>=100

4. Qual das expressões OCL abaixo corretamente expressa a invariante de que um mesmo cliente não pode fazer mais de uma reserva para o mesmo filme? (a operação "excludesAll" retorna *true* se as coleções receptora e passada como parâmetro são disjuntas. Ex.: c1->excludesAll(c2) é verdadeiro se e somente se c1∩c2 = Ø.)

a. Context Cliente
inv:
cliente.reserva->forAll(|reserva1|
cliente.reserva->forAll(reserva2|
reserva1.cliente = reserva2.cliente IMPLIES
reserva1.filme->excludesAll(reserva2.filme)
)
)

b. Context Cliente
inv:
cliente.reserva->forAll(|reserva|
reserva.filme->excludesAll(reserva.filme)
)

c. Context Cliente
inv:
cliente.reserva->forAll(|reserva1|
cliente.reserva->forAll(reserva2|
(reserva1<>reserva2 AND reserva1.cliente = reserva2.cliente) IMPLIES
reserva1.filme->excludesAll(reserva2.filme)
)
)

d. Context Cliente
pre:
cliente.reserva->forAll(|reserva1|
cliente.reserva->forAll(reserva2|
(reserva1<>reserva2 AND reserva1.cliente = reserva2.cliente) IMPLIES
reserva1.filme->excludesAll(reserva2.filme)
)
)

e. Context Cliente
inv:
cliente.reserva->forAll(|reserva1|
cliente.reserva->forAll(reserva2|
reserva1.filme->excludesAll(reserva2.filme)
)
)

5. A consulta de sistema "listaReservas" deve apresentar na ordem definida pela associação "prioriza" as reservas indicando o nome do cliente, o título do filme e a data da reserva. Qual das expressões OCL abaixo corretamente define esta consulta?

a. Context Videolocadora::listaReservas():OrderedSet
body:
self.reserva->collect(reserva|
Tuple {
cliente = reserva.cliente.nome,
filme = reserva.filme.titulo,
data = reserva.data
}
)

b. Context Videolocadora::listaReservas():OrderedSet
body:
self.reserva->collect(reserva|

```
        cliente = reserva.cliente.nome,
        filme = reserva.filme.titulo,
        data = reserva.data
    )
```

c.
```
Context Videolocadora::listaReservas():OrderedSet
 body:
   Tuple {
     cliente = self.reserva.cliente.nome,
     filme = self.reserva.filme.titulo,
     data = self.reserva.data
   }
 )
```

d.
```
Context Videolocadora::listaReservas():OrderedSet
 body:
  self.reserva->select(reserva|
   Tuple {
     cliente = reserva.cliente.nome,
     filme = reserva.filme.titulo,
     data = reserva.data
   }
 )
```

e.
```
Context Videolocadora::listaReservas():OrderedSet
 body:
  self.reserva->select(reserva|
   reserva.cliente.nome = nome
 )
```

6. A operação de sistema "cancelaReserva(cliente,filme)" deve cancelar a reserva de um cliente para um determinado filme da seguinte forma: se o filme é o único que aparece na reserva, remove a instância da reserva, senão, remove a associação entre a reserva e o filme. Assume-se que tanto o cliente quanto o filme são únicos e válidos do ponto de vista do cadastro do sistema. Assume-se também que existe uma reserva para o filme e o cliente informados. Qual contrato OCL abaixo corretamente descreve esta operação?

a.
```
Context Videolocadora::cancelaReserva(cliente,filme)
 pre:
  self.cadastro[cliente]->size()=1 AND
  self.filme[filme]->size()=1AND
  self.reserva->size()=1
  )
 post:
  if reserva.filme->size() = 1 then
   reserva^destroy()
 else
   reserva^removeFilme(self.filme[filme])
 endif
```

b.
```
Context Videolocadora::cancelaReserva(cliente,filme)
 def: reserva =
  self.reserva->select(r|
   r.cliente=self.cadastro[cliente] AND
   r.filme->includes(self.filme[filme])
 pre:
  self.cadastro[cliente]->size()=1 AND
  self.filme[filme]->size()=1AND
  reserva->size()=1
  )
 post:
  if reserva.filme->size() = 1 then
   reserva^destroy()
 else
```

```
      reserva^removeFilme(self.filme[filme])
    endif
```

c.
```
Context Videolocadora::cancelaReserva(cliente,filme)
 def: reserva =
  self.reserva->select(r|
    r.cliente=self.cadastro[cliente] AND
    r.filme->includes(self.filme[filme])
 pre:
  self.cadastro[cliente]->size()>1 AND
  self.filme[filme]->size()>1AND
  reserva->size()>1
  )
 post:
  if reserva.filme->size() = 1 then
    reserva^destroy()
 else
    reserva^removeFilme(self.filme[filme])
 endif
```

d.
```
Context Videolocadora::cancelaReserva(cliente,filme)
 def: reserva =
  self.reserva->select(r|
    r.cliente=self.cadastro.cliente AND
    r.filme->includes(self.filme)
 pre:
  self.cadastro[cliente]->size()=1 AND
  self.filme[filme]->size()=1AND
  reserva->size()=1
  )
 post:
  if reserva.filme->size() = 1 then
    reserva^destroy()
 else
    reserva^removeFilme(self.filme[filme])
 endif
```

e.
```
Context Videolocadora::cancelaReserva(cliente,filme)
 def: reserva =
  self.reserva->select(r|
    r.cliente=self.cadastro[cliente] AND
    r.filme->includes(self.filme[filme])
 pre:
  self.cadastro[cliente]->size()=1 AND
  self.filme[filme]->size()=1AND
  reserva->size()=1
  )
 post:
   reserva^destroy()
```

1	2
a	b

1. Considerando uma classe Pessoa com atributos cpf, nome e endereço, ligada à classe Controladora por uma associação para *, o que a operação xyz abaixo representa?

```
Context Controladora::xyz():Set
        body:
                self.pessoa->select(p|
                        p.endereco->isNull()
                ) ->collect(p|
```

```
        Tuple{
                nome = p.nome,
                cpf = p.cpf
        }
)
```

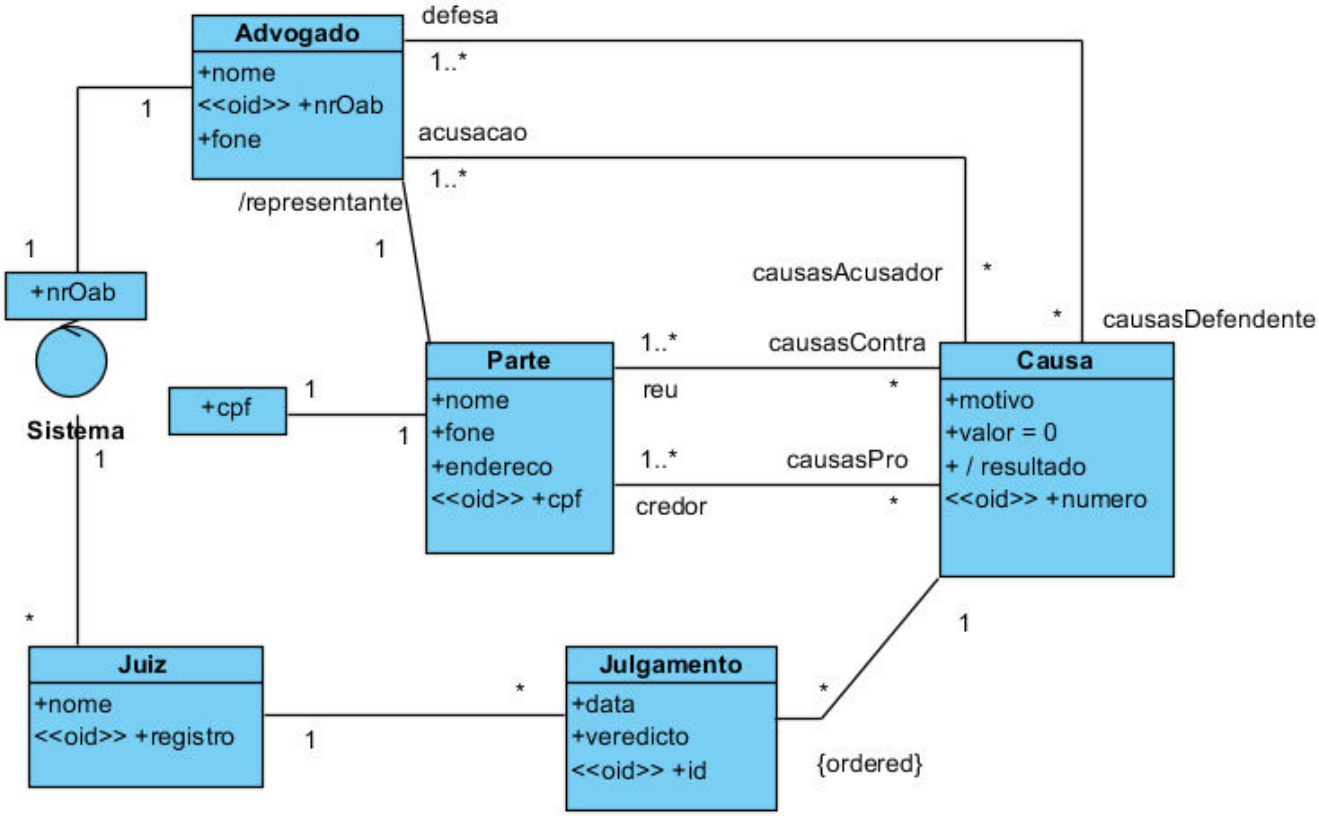
a. Uma consulta que retorna uma lista com o nome e cpf das pessoas que tem endereço definido.
b. Uma consulta que retorna uma lista com o nome e cpf de todas as pessoas.
c. Uma consulta que retorna uma lista com nomes e outra lista com o cpf das pessoas que tem endereço definido.
d. Uma consulta que retorna uma lista de instâncias de Pessoa com endereço definido.
e. Uma consulta que retorna uma tupla com um nome e um endereço da pessoa que não tem endereço definido.
f. Uma consulta que retorna uma tupla com nome e cpf de uma pessoa cujo endereço não é definido.

2. Uma *invariante* deve ser usada em qual das situações abaixo?
   a. Quando uma operação necessita que os dados estejam em um estado particular.
   b. Quando uma regra sobre os dados deve ser obedecida em toda e qualquer situação.
   c. Quando uma operação retorna algum valor pré-definido.
   d. Quando uma classe tem baixa coesão.
   e. Quando os dados são numerosos e deve se restringir sua proliferação.
   f. Quando uma condição temporária deve ser observada para que uma operação possa ser executada.

Respostas:

1	2	3	4	5	6
a	c	e	e	d	b
89%	83%	69%	67%	92%	94%

Média da turma: 8,5
Nota mais alta: 10 (8x)
Nota mais baixa: 5 (2x)



1. Qual expressão OCL abaixo define corretamente e de acordo com o diagrama o valor inicial do atributo **valor** na classe **Causa**?
   a. Context Causa::valor init: 0.
   b. Context Causa init: valor := 0.

c. Context Sistema::Causa::valor init: 0.
d. Context Sistema init: Causa.valor = 0.
e. Context valor:Causa init: 0.

2. Qual expressão OCL abaixo define corretamente e de acordo com o diagrama o atributo derivado **resultado** na classe **Causa** como sendo o **veredicto** do último julgamento desta causa.
   a. Context Causa::resultado derive: julgamento->select(last).veredicto.
   b. Context Causa::resultado derive: julgamento->last()->select(veredicto).
   c. Context Causa::resultado derive: julgamento->last().veredicto.
   d. Context Causa::resultado derive: julgamento[oid].veredicto->last()
   e. Context Causa::resultado derive: julgamento.veredicto[last].

3. Qual expressão OCL abaixo define corretamente e de acordo com o diagrama a associação derivada **representante** de **Parte** para **Advogado**, como sendo o conjunto de advogados que representam uma dada parte em suas causas (se a causa for contra a pessoa é o advogado de defesa, senão é o de acusação).
   a. Context Parte::representante derive: causasContra.acusacao-> includingAll(causasPro.defesa).
   b. Context Parte::representante derive: causasContra.defesa-> includingAll(causasPro.acusacao) -> includingAll(causasContra.acusacao) -> includingAll(causasPro.defesa).
   c. Context Parte::representante derive: causasContra.advogado -> includingAll(causasPro.advogado).
   d. Context Parte::representante derive: causasContra.defesa AND causasPro.acusacao.
   e. Context Parte::representante derive: causasContra.defesa-> includingAll(causasPro.acusacao).

4. Qual expressão OCL abaixo define corretamente e de acordo com o diagrama uma invariante que estabelece que nenhum advogado pode atuar como defesa e acusação para a mesma causa.
   a. Context Causa inv: defesa->forAll(adv| acusacao->includes(adv))
   b. Context Causa inv: causasDefendente->forAll(adv|NOT causasAcusador->includes(adv))
   c. Context Causa inv: defesa->forAll(adv| acusacao-> NOT includes(adv))
   d. Context Causa inv: defesa->exists(adv|NOT acusacao->includes(adv))
   e. Context Causa inv: defesa->forAll(adv|NOT acusacao->includes(adv))

5. Qual expressão OCL abaixo define corretamente e de acordo com o diagrama um contrato de consulta de sistema que retorna o valor total das causas de uma determinada parte. Esta operação deve receber como parâmetro o CPF da parte, e retornar um valor numérico único.
   a. Context Parte::totalCausas(umCpf) body: causasContra->sum(valor) + causasPro -> sum(valor)
   b. Context Parte::totalCausas(umCpf) post: causasContra->sum(valor) + causasPro -> sum(valor)
   c. Context Parte::totalCausas(umCpf) body: causasContra->inclundingAll(causasPro)->sum()
   d. Context Sistema::totalCausas(umCpf) body: parte[umCpf].causasContra->sum(valor) + parte[umCpf].causasPro->sum(valor)
   e. Context Sistema::totalCausas(umCpf) body: parte[umCpf].causasContra->sum() + parte[umCpf].causasPro->sum()

6. Escreva uma expressão OCL para um contrato de operação de sistema que permite o cadastramento (Create de CRUD) de um novo advogado. A operação recebe como parâmetro o nome, número da OAB, e telefone do advogado. Deve ser considerada uma exceção se o número da OAB passado como parâmetro corresponder a um advogado que já tenha cadastro.
   a. Context Sistema::insereAdvogado(umNome,umNrOAB,umFone)
      def: novoAdv = Advogado::newInstance()

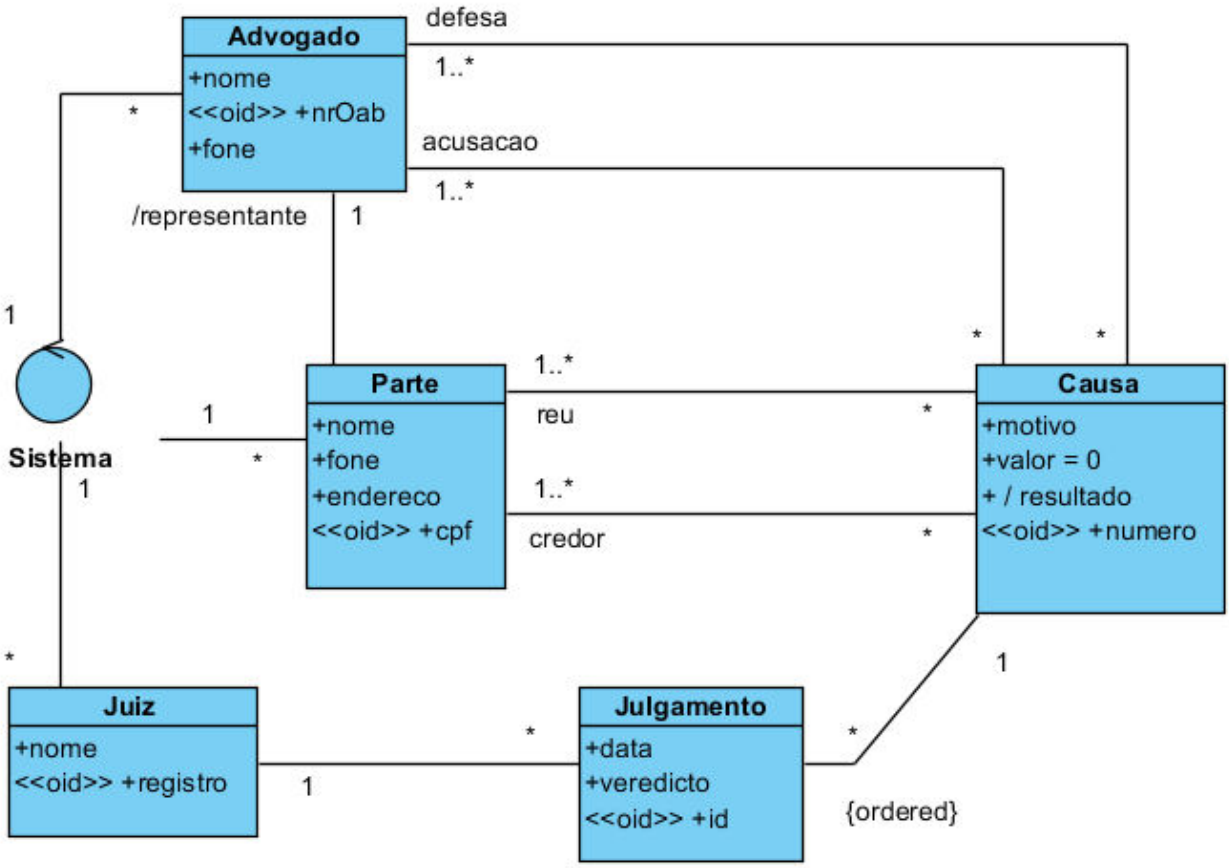
post: novoAdvogado^setNome(umNome) AND
novoAdvogado^setNrOAB(umNrOAB) AND
novoAdvogado^setTelefone(umFone)
exception: advogado[umNrOAB]->size()>0 IMPLIES self^throw("já existe")

b. Context Sistema::insereAdvogado(umNome,umNrOAB,umFone)
def: novoAdvogado = Advogado::newInstance()
post: self^addAdvogado(novoAdvogado) AND
novoAdvogado^setNome(umNome) AND
novoAdvogado^setNrOAB(umNrOAB) AND
novoAdvogado^setTelefone(umFone)
exception: advogado[umNrOAB]->size()>0 IMPLIES self^throw("já existe")

c. Context Sistema::insereAdvogado(umNome,umNrOAB,umFone)
def: novoAdv = Advogado::newInstance()
pre: advogado[umNrOAB]->size()>0
post: self^addAdvogado(novoAdvogado) AND
novoAdvogado^setNome(umNome) AND
novoAdvogado^setNrOAB(umNrOAB) AND
novoAdvogado^setTelefone(umFone)

d. Context Sistema::insereAdvogado(umNome,umNrOAB,umFone)
def: novoAdv = Advogado::newInstance()
post: self^addAdvogado(novoAdvogado) AND
novoAdvogado^setNome(umNome) AND
novoAdvogado^setNrOAB(umNrOAB) AND
novoAdvogado^setTelefone(umFone)
exception: advogado[umNrOAB]->size()=0 IMPLIES self^throw("já existe")

e. Context Sistema::insereAdvogado(umNome,umNrOAB,umFone)
def: novoAdv = advogado[umNrOAB]
post: self^addAdvogado(novoAdvogado) AND
novoAdvogado^setNome(umNome) AND
novoAdvogado^setNrOAB(umNrOAB) AND
novoAdvogado^setTelefone(umFone)
exception: advogado[umNrOAB]->size()>0 IMPLIES self^throw("já existe")



1. Escreva uma expressão OCL que defina o valor inicial do atributo **valor** na classe **Causa**.

2. Escreva a como a expressão OCL que defina um atributo derivado **resultado** na classe **Causa** como sendo o **veredicto** do último julgamento desta causa.

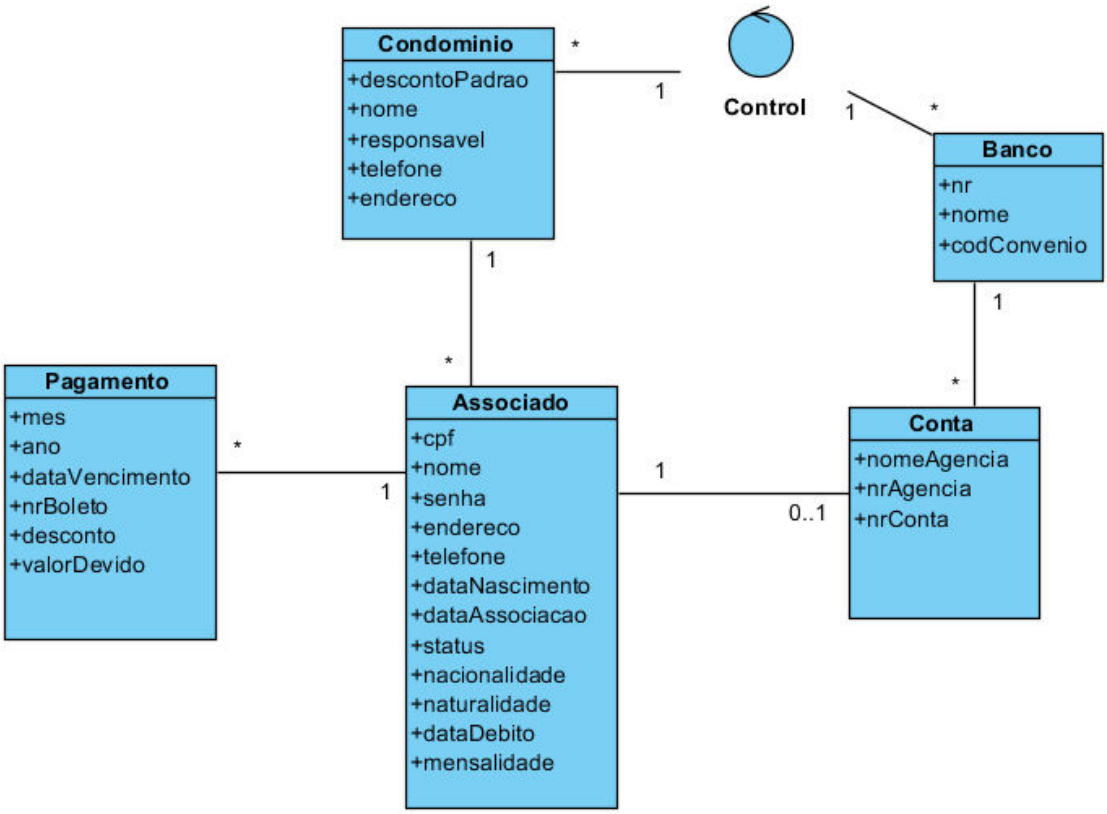
3. Escreva uma expressão OCL que defina a associação derivada **representante** de **Parte** para **Advogado**, como sendo o conjunto de advogados que representam uma dada parte em suas causas.

4. Escreva uma expressão OCL que defina uma invariante que estabelece que nenhum advogado pode atuar como defesa e acusação para a mesma causa.

5. Escreva uma expressão OCL para um contrato de consulta de sistema que retorna o valor total das causas de uma determinada parte. Esta operação deve receber como parâmetro o CPF da parte, e retornar um valor numérico único.

6. Escreva uma expressão OCL para um contrato de operação de sistema que permite o cadastramento (Create de CRUD) de um novo advogado. A operação recebe como parâmetro o nome, número da OAB, e telefone do advogado. Deve ser considerada uma exceção se o número da OAB passado como parâmetro corresponder a um advogado que já tenha cadastro.

Considere o modelo:



1. Considerando o modelo conceitual apresentado no início da prova, escreva uma expressão OCL que defina uma associação derivada entre Associado e Banco, consistindo do banco da conta que o associado pode ter cadastrado.

```
Context Associado::banco
  derive:
    self.conta.banco
```

2. Considerando o modelo conceitual apresentado no início da prova, escreva uma expressão OCL que defina uma invariante para a classe Associado que estabeleça que a data de associação deve ser necessariamente posterior à data de nascimento.

```
Context Associado
  inv:
    dataNascimento < data Associacao
```

3. Considerando o modelo conceitual apresentado no início da prova, escreva uma expressão OCL que defina uma consulta de sistema que retorne o nome e cpf de todos os associados de um condomínio, que não optaram pelo débito em conta: consultaNaoOptantes(nomeCondominio):SET. Considere que o nome do condomínio é válido.

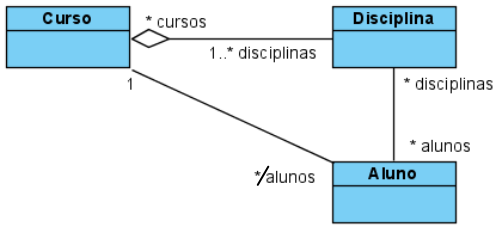
```
Context Control::consultaNaoOptante(nomeCondominio):SET
  def: condominio=self.condominio->select(c|c.nome=nomeCondominio)
  def: naoOptante=condominio.associado->select(a|a.conta->size()=0)
  pre:
    condominio->size()=1
  body:
    naoOptante->collect(no|
      Tuple {
        cpf = no.cpf,
        nome = no.nome
      }
    )
```

4. Considerando o modelo conceitual apresentado no início da prova, escreva uma expressão OCL que represente a operação de sistema que cadastra um novo condomínio com seus dados. Considere como exceção a tentativa de cadastrar um condomínio com um nome que já exista.

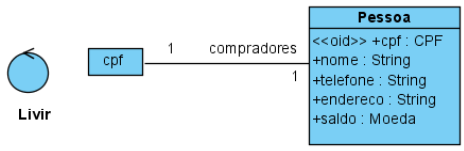
```
Context
Control::insereCondominio(descontoPadrao,nome,responsavel,telefone,endereco)
```

```
def: novoCondominio = Condominio::newinstance()
post:
  novoCondominio^addCondominio(novoCondominio) AND
  novoCondominio^setDescontoPadrao(descontoPadrao) AND
  novoCondominio^setNome(nome) AND
  novoCondominio^setResponsavel(responsavel) AND
  novoCondominio^setTelefone(telefone) AND
  novoCondominio^setEndereco(endereco)
exception:
  self.condominio@pre->select(c|c.nome=nome)->size()>0 IMPLIES
    self^throw("nome já cadastrado")
```

1. Considerando o modelo conceitual da figura abaixo, escreva uma expressão OCL que defina a associação derivada "alunos" de Curso para Aluno, como sendo o conjunto de todos os alunos de todas as disciplinas de um curso.



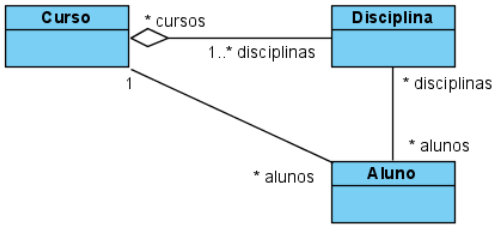
2. Considerando o modelo conceitual da figura ao lado, escreva uma expressão OCL que defina um contrato de uma consulta que liste os nomes (apenas) de todas as pessoas cujo saldo é superior a um valor passado como parâmetro na consulta.



3. Ainda considerando a mesma figura da questão anterior, escreva uma expressão OCL que defina um contrato e operação de sistema CRUD, que atualize os dados de um cliente cujo CPF é passado (apenas o CPF não pode ser atualizado).

4. Qual das expressões OCL abaixo corretamente define a associação derivada "livros", da figura ao lado como o conjunto de livros associados aos itens de uma venda?

   Context Venda::livros derive: self.itens.livro

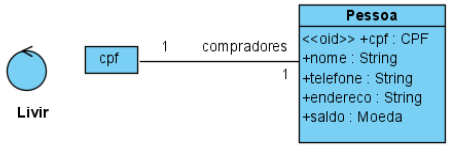


5. Considerando o modelo conceitual da figura ao lado, qual das expressões OCL abaixo reflete a invariante de que um aluno só pode estar em disciplinas que pertençam ao curso ao qual o próprio aluno está ligado?

   Context Aluno inv: self.disciplinas→forAll(d|d.cursos→includes(self.curso))

6. Considerando o modelo conceitual da figura ao lado, qual das expressões OCL abaixo melhor representa a consulta de sistema que retorna nome e telefone das pessoas com saldo acima de um valor limite passado como parâmetro.

   Context Livir::consultaPessoasSaldoAcima(limite):Set
   body:
   self.compradores→select(p|p.saldo>limite)→collect(Tuple{nome=p.nome, telefone=p.telefone})



7. Ainda considerando a mesma figura da questão 4, qual expressão OCL abaixo melhor representa o contrato de operação de sistema que atualiza o saldo de uma pessoa cujo CPF é dado como parâmetro? Considera-se que o CPF passado é válido, ou seja, que há um comprador registrado com este CPF.

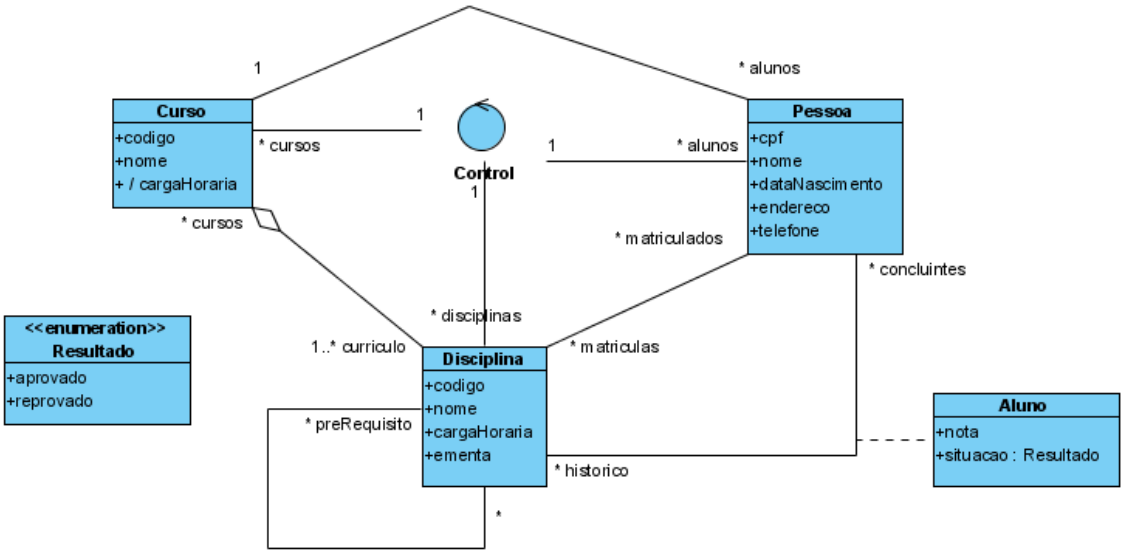
```
//Método para determinar se o número é primo ou não;
public String primim(int x) {

String acumulador="";
int contador=1;

    for (int j=1; j < x; j++) {
            if (x % j == 0)  {
                    contador++;
            }
            if (contador > 2) {
                    acumulador = x + " nao eh primo";
            } else {
                    acumulador = x + " eh primo";
            }
    }
    return acumulador;
}
```

Context Livir::atualizaSaldo(cpf,novoSaldo)
pre: compradores[cpf]→notNull()
post: compradores[cpf]^setSaldo(novoSaldo)

Para as questões abaixo, considere o modelo conceitual:



1. Qual das expressões abaixo representa uma invariante OCL que estabelece que as ementas de todas as disciplinas existentes devem ter no mínimo 50 caracteres?
    a. Context Control inv: self.disciplinas.ementa>=50
    b. Context Disciplina inv: size(self.ementa)>=50
    c. Context Control inv: size(disciplina.ementa)>=50
    d. Context Disciplina inv: self.disciplinas->forAll(d|size(d.ementa)>=50)

2. (DISCURSIVA) Escreva um contrato para matricular uma pessoa em um curso: matricula(cpf,codigoCurso). Considere que o cpf e código do curso são válidos, isto é, existem na base de dados.

```
Context Control::matricula(cpf, codigoCurso)
Def: pessoa = self.alunos->select(p|p.cpf=cpf)
Def: curso = self.cursos->select(p|p.codigo=codigoCurso)
Pre: pessoa->size()=1 AND
     Curso->size()=1
Post: pessoa^setCurso(curso)
```

3. Suponha que uma disciplina vai ser removida (deletada). Quais outras classes devem ser observadas para verificar se a remoção da disciplina deve ser impedida ou propagada em função de associações obrigatórias?
    a. Nenhuma
    b. Curso
    c. Curso e Control
    d. Curso, Pessoa e Aluno
    e. Disciplina

4. Qual das expressões abaixo consiste em um contrato CRUD que consulta os dados de uma pessoa cujo cpf é passado como parâmetro?

a. Context Control:: consultaPessoa(cpf):Tuple
   body: self.alunos->select(p|p.cpf=cpf)->collect(p|Tuple{cpf=p.cpf, nome=p.nome, dataNascimento=p.dataNascimento, endereco=p.endereco, telefone=p.telefone})
b. Context Control:: consultaPessoa(cpf):Tuple
   body: self.alunos-> collect(p|Tuple{cpf=p.cpf, nome=p.nome, dataNascimento=p.dataNascimento, endereco=p.endereco, telefone=p.telefone})
c. Context Control:: consultaPessoa(cpf):Tuple
   body: self.alunos->select(p|p.cpf=cpf)
d. Context Control:: consultaPessoa(cpf):Tuple
   body: self.alunos->select(p|p.cpf=cpf).cpf, nome,dataNascimento,endereco,telefone})
e. Context Pessoa:: consultaPessoa(cpf):Tuple
   body: self.alunos->select(p|p.cpf=cpf)->collect(p|Tuple{cpf=p.cpf, nome=p.nome, dataNascimento=p.dataNascimento, endereco=p.endereco, telefone=p.telefone})