Programação por Objectos

UML

Análise por UML (1)

- Um **sistema de análise** descreve os modelos da aplicação a desenvolver.
  - Aumenta legibilidade (menos informação que o código, permitindo visualizar globalmente a aplicação).
  - Mostra estrutura da aplicação, sem detalhes de implementação.
  - Representação gráfica incrementa clareza semântica.
  - Devido à complexidade, a abordagem OO é descrita por vários modelos, onde cada modelo aborda um aspecto particular.

Análise por UML (2)

- **UML** (*Unified Modeling Language*) resulta da fusão de vários sistemas de análise:
  - Booch (G. Booch)
  - OOSE (I. Jacobson)
  - OMT (J. Rumbaugh)

- 1ª proposta divulgada em 1997 (1999 - v1.1, 2000 - v1.3, 2001 - v1.4, 2003 - v1.5, Abril 2004 - **v2.0**)

- Ferramentas:
  - Rational Rose
  - Visual Paradigm
  - Plug-in do eclipse (**http://www.soyatec.com/euml2/**)
  - ArgoUML (**http://argouml.tigris.org/**)

Análise por UML (3)

- UML v2 disponibiliza 13 diagramas, agrupados em:
  - Modelação estrutural (pacotes, classes, objectos, estrutura composicional, componentes, aplicação física).
  - Modelação de comportamento (*use case*, actividade, máquina de estados, comunicação, sequência de mensagens, temporização, enquadramento das interacções).

- Em POO são abordados apenas os diagramas centrais do UML, na sequência:

  conceito ⇒ representação em UML ⇒ implementação em Java

Classes – definição

- Uma **classe** é um padrão de objectos, definida por:
  - **Identificador**
  - **Atributos** (que definem o estado dum objecto) cujos valores podem ser:
    - tipos primitivos (inteiros,…)
    - referências a outros objectos (identificando relações entre objectos)
  - **Métodos** (operações que podem alterar o estado do objecto)

- Os atributos e os métodos são designados por **membros da classe**.

Classes – exemplo (1)

"Uma conta bancária contém sempre uma quantia e pertence a uma pessoa. A pessoa tem um nome, telefone e um número de BI. Na conta pode ser depositado ou levantado dinheiro. Sempre que entender, o dono pode consultar o saldo da conta".

Classes

- Conta
- Pessoa

Atributos primitivos

- Conta: quantia (float)
- Pessoa: nome (string), numTelf (long), numBI (long)

Classes – exemplo (2)

Atributos referência

- Conta: dono (instância de uma Pessoa)

Métodos

- Conta:
  - levantamento (parâmetro: quantia a levantar)
  - deposito (parâmetro: quantia a depositar)
  - saldo (retorna: quantia da conta)
- Pessoa:
  - numTelf (retorna: número de telefone)
  - numBI (retorna: número do Bilhete de Identidade)

Métodos – definição (1)

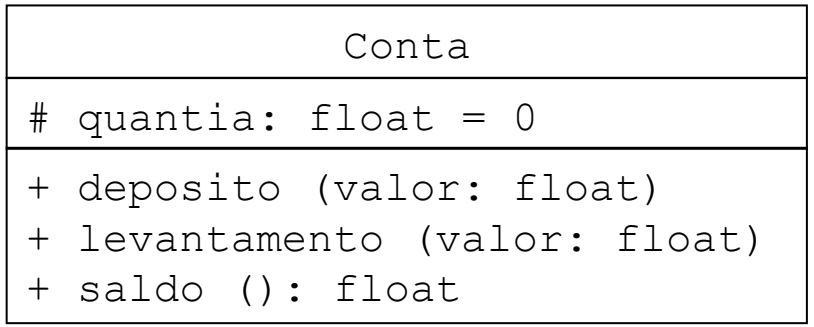
- Um **método** é uma sequência de acções, executada por um objecto, que pode alterar ou dar a conhecer o seu estado (valor dos atributos).
- Enquanto o valor dos atributos reside no objecto, o método reside na classe.
- **Assinatura** dum método:
  - identificador do método;
  - identificador e tipo dos parâmetros;
  - valor de retorno.

Métodos – definição (2)

- Os métodos são catalogados em:
  - **Construtor**: executado na criação do objecto.
    - Têm usualmente o mesmo identificador da classe.
    - Nunca devolvem tipos.
    - Não podem ser chamados.
    - Normalmente usados para inicializar os atributos.
  - **Destrutor**: executado na destruição do objecto.
  - **Modificador**: altera valor dos atributos.
  - **Selector**: dá a conhecer o valor dos atributos, sem os alterar.

Classes – UML

- Representada por um rectângulo, dividido em 3 zonas:
  - Identificador da classe
  - Atributos (primitivos)
  - Métodos



Nota: Apenas se representam os atributos primitivos, os atributos referência serão representados com associações.

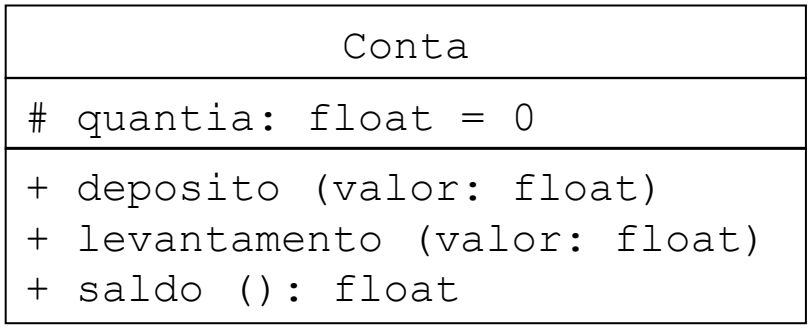
- As zonas dos atributos e métodos são opcionais.
- Um método e um atributo podem ter o mesmo identificador.

Atributos – UML

Sintaxe

Visib Ident: Tipo [=Init][{Prop}]

- Visib: visibilidade
- Ident: identificador do atributo
- Tipo: tipicamente os tipos de dados incluem os tipo primitivos (boolean, int, char, float,…)
- **Init**: inicialização
- Prop: propriedade adicional

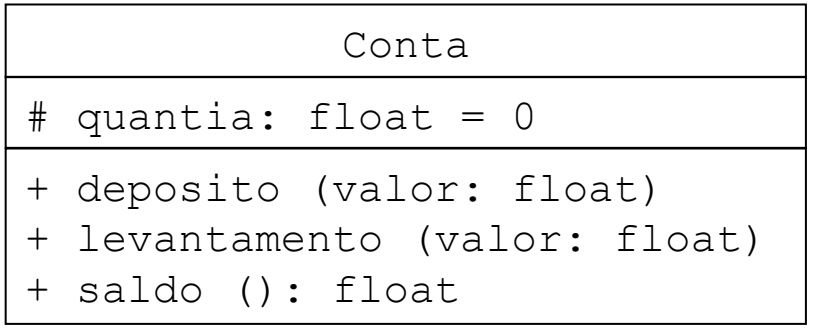


Métodos – UML

Sintaxe

Visib Ident([id:TipoParam [, id:TipoParam]*])
[:TipoRet] [{Prop}]

- Visib: visibilidade
- Ident: identificador do método
- id: identificador de parâmetro
- TipoParam: tipo do parâmetro
- TipoRet: tipo de retorno
- Prop: propriedade adicional



Visibilidade de atributos e métodos – UML

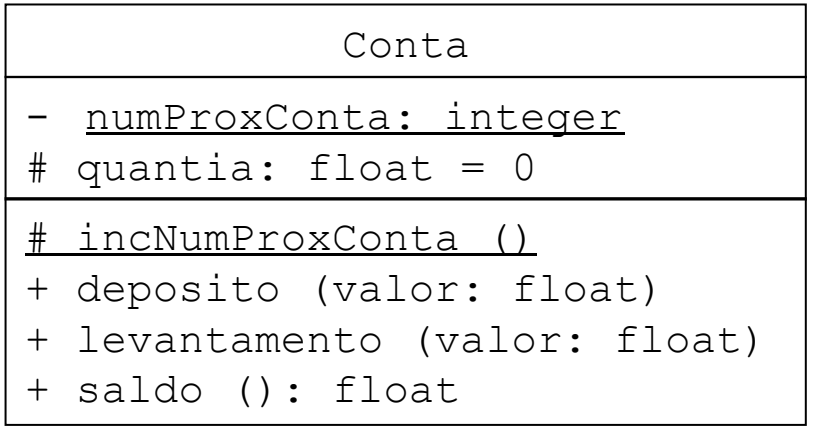
- Visibilidade de atributos e métodos representada por um caractere antes do identificador, que determina as permissões de acesso:
  - **public**: acessível fora da classe (+)
  - **private**: acessível apenas na classe (−)
  - **protected**: acessível na classe e suas subclasses (#)
  - **package**: acessível nas classes do mesmo pacote (~)

Atributos e métodos de classe – definição

- Atributos e métodos podem ser de:
  - instância: um para cada instância da classe
  - classe: um para todas as instâncias da classe

Atributos e métodos de classe – UML

- Representados com sublinhado no diagrama de classe.

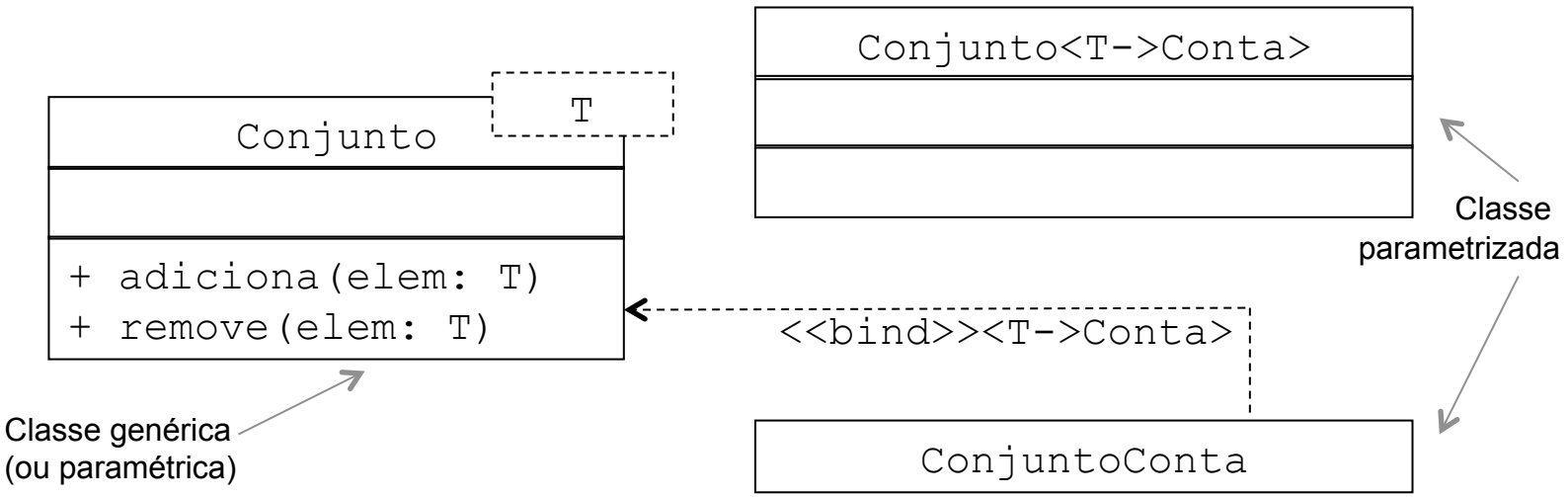


Classes genéricas – definição

- Uma **classe genérica** (ou **classe paramétrica**) é uma classe que recebe como argumento outras classes. As instâncias de **classes genéricas** são denominadas **classes parametrizadas**.
- As classes genéricas são muito utilizadas na definição de colecções (conjuntos, listas, pilhas, árvores, etc).

Classes genéricas – UML

- Representada em UML com uma caixa a tracejado sobre o canto direito de uma representação UML de uma classe. A caixa a tracejado contém uma lista com o tipo de parâmetros a passar à classe genérica.



Relações – definição

- Os objectos não vivem isolados e nos programas são estabelecidas relações de cooperação.
- Uma **relação** é uma conexão entre elementos. Existem diferentes tipos de relações:
  - **Associação**: relaciona objectos entre si.
  - **Composição/Agregação**: relação que denota o *todo* constituido por *partes*.
  - **Herança**: mecanismo de generalização-especialização de classes.
  - **Realização**: uma classe implementa a funcionalidade definida numa interface.

Associação – UML (1)

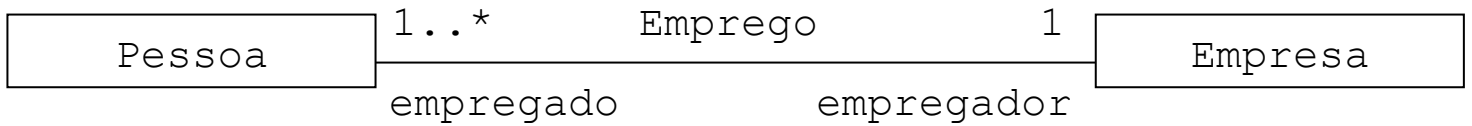
- Uma **associação** representa uma referência entre objectos.
- Numa associação são definidos:
  - **Identificador** – termo descritivo da associação.
  - **Papeis** (*role*) representados pela associação em cada uma das classes relacionadas.
  - **Multiplicidade** – número de objectos associados em cada instância da associação.

Associação – UML (2)

- O identificador e os papeis são opcionais.
- As associações podem ser de multiplicidade diversa:
  - exactamente um (`1`).
  - zero ou um (`0..1`).
  - zero ou mais (`0..*`).
  - um ou mais (`1..*`).
  - são permitidas multiplicidades mais complexas, por exemplo, `0..1, 3..4, 6..*`, para dizer qualquer número excepto 2 e 5.

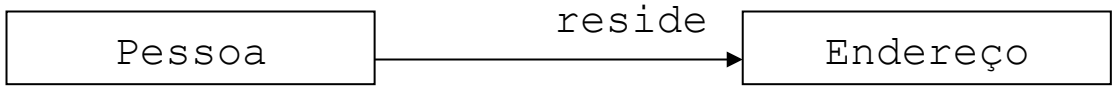
Associação – UML (3)

- A associação é representada por uma linha entre as classes associadas.
- O identificador da associação aparece sobre a associação.
- Os papéis de cada classe na associação aparecem nos respectivos extremos da associação.
- A multiplicidade da associação aparece igualmente nos extremos.



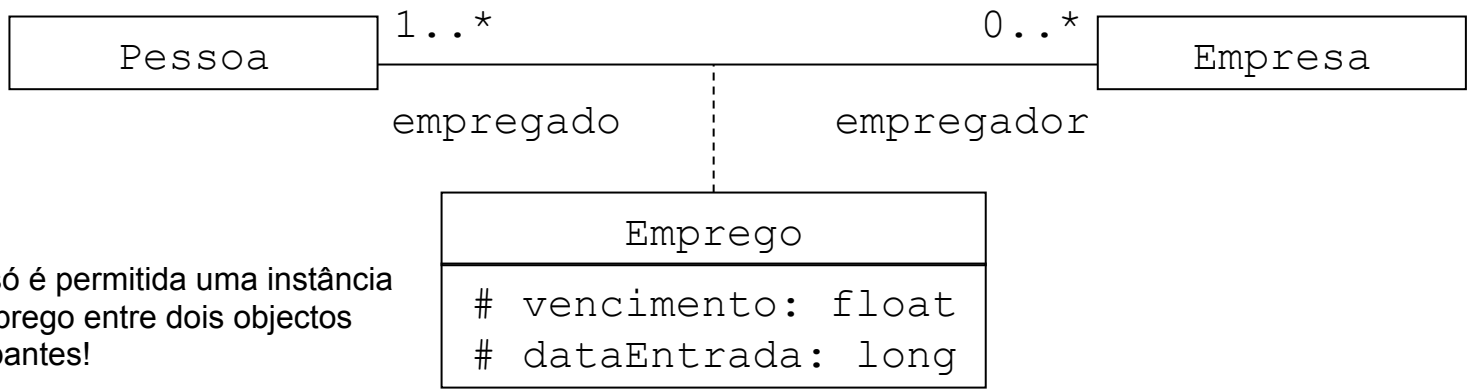
Associação – UML (4)

- As associações podem ser **dirigidas**, e nesse caso são representadas por uma seta.
- A direcção das associações está relacionada com aspectos de implementação.



Associação – UML (5)

- As associações podem, elas próprias, transportar informação, sendo nesse caso a **classe de associação** ligada a tracejado à associação.
- O identificador da associação passa a ser o nome da classe de associação.



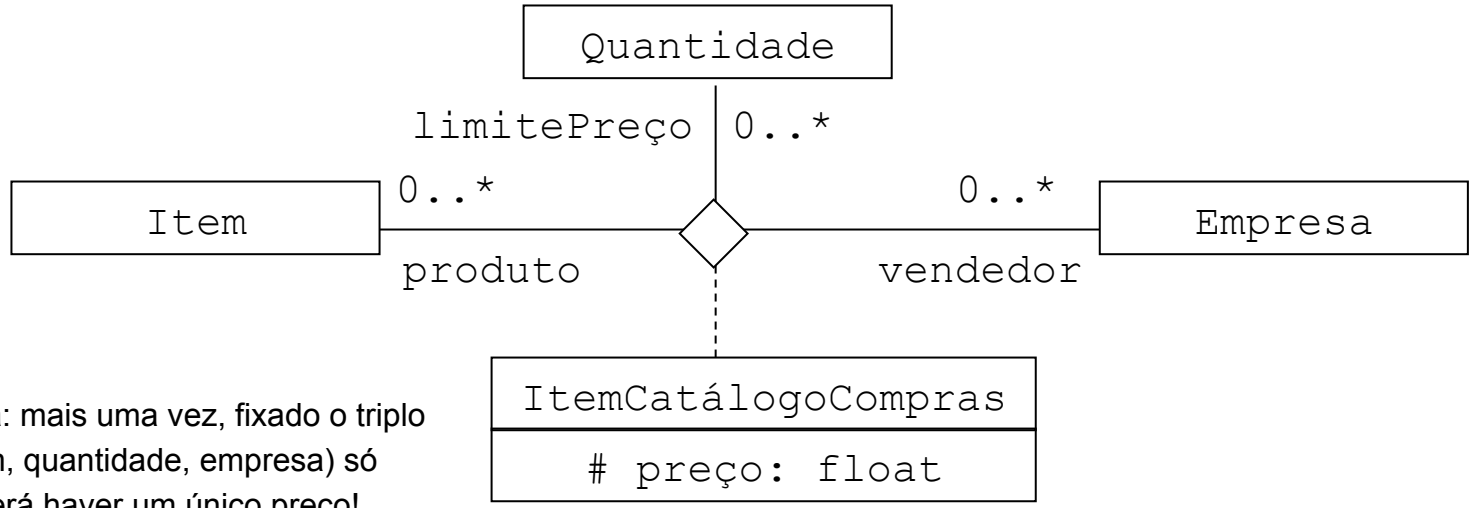
Nota: só é permitida uma instância de Emprego entre dois objectos participantes!

Associação – UML (6)

- Por omissão, a associação é:
  - bi-direccional;
  - de um para um;
  - não possui informação extra.

Associação – UML (7)

- **Associações ternárias** representadas por um diamante não preenchido que liga as diferentes linhas das classes associadas.



Nota: mais uma vez, fixado o triplo (item, quantidade, empresa) só poderá haver um único preço!

Agregação/Composição – UML (1)

- A **agregação** é uma associação, que denota uma relação do *todo* ser formado por *partes*.
- A agregação é dita como sendo uma relação de **"has-a"**.
- Representada como uma associação com um pequeno diamante não preenchido no extremo relativo ao *todo.*

Agregação/Composição – UML (2)

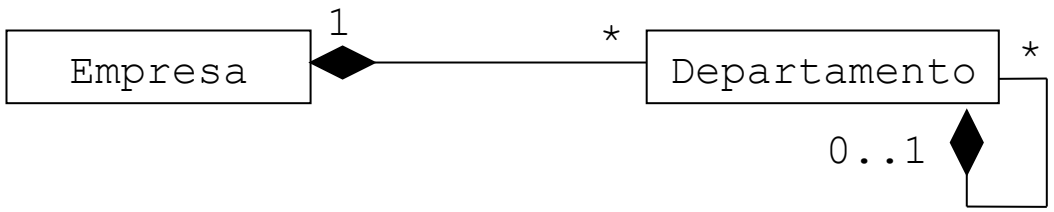
- Na **composição**, o desaparecimento do *todo* conduz ao desaparecimento das *partes* - **ausência de partilha**.
- Representada como uma associação com um pequeno diamante preenchido no extremo relativo ao *todo.*

Agregação/Composição – UML (3)

- De uma maneira geral tanto a agregação como a composição não têm identificador, pois o significado destas relações está representada pelo próprio par *todo-partes*.
- A multiplicidade deve aparecer em ambos os extremos. Quando a multiplicidade é omitida, considera-se que é exactamente 1.

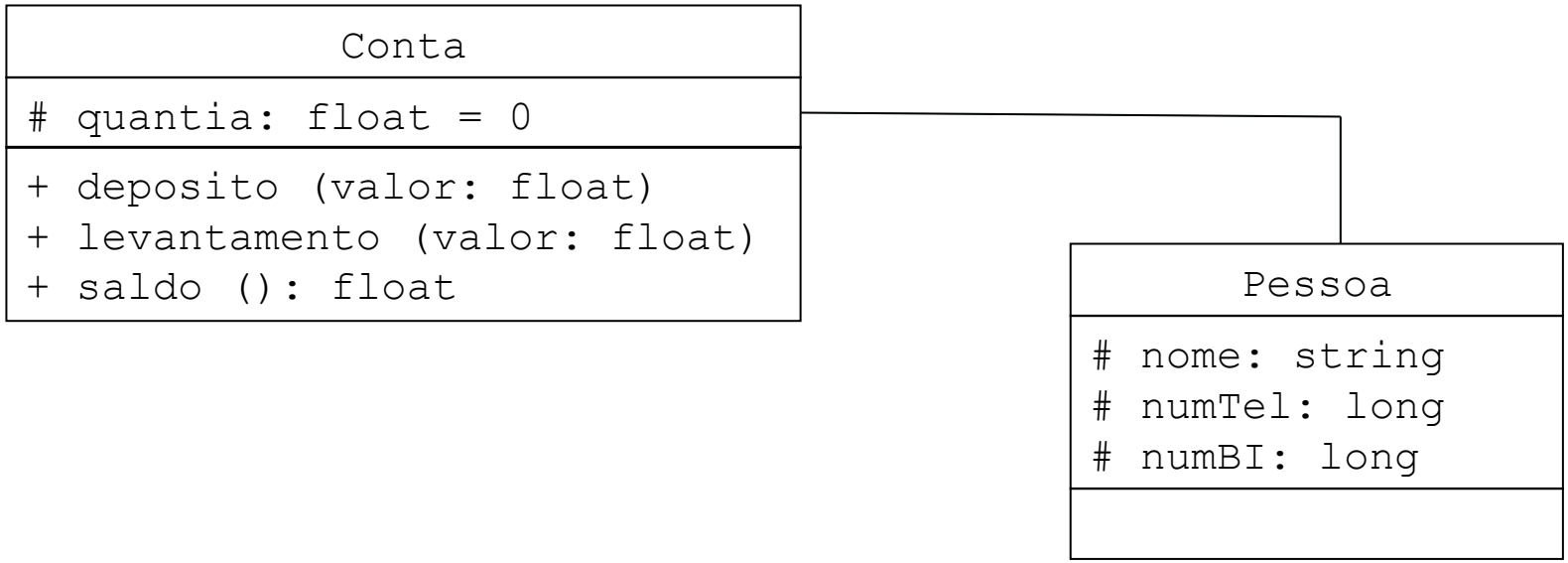
Reflexividade nas relações – UML

- As relações de associação, agregação e composição podem ser reflexivas, com um elemento composto por vários sub-elementos iguais.



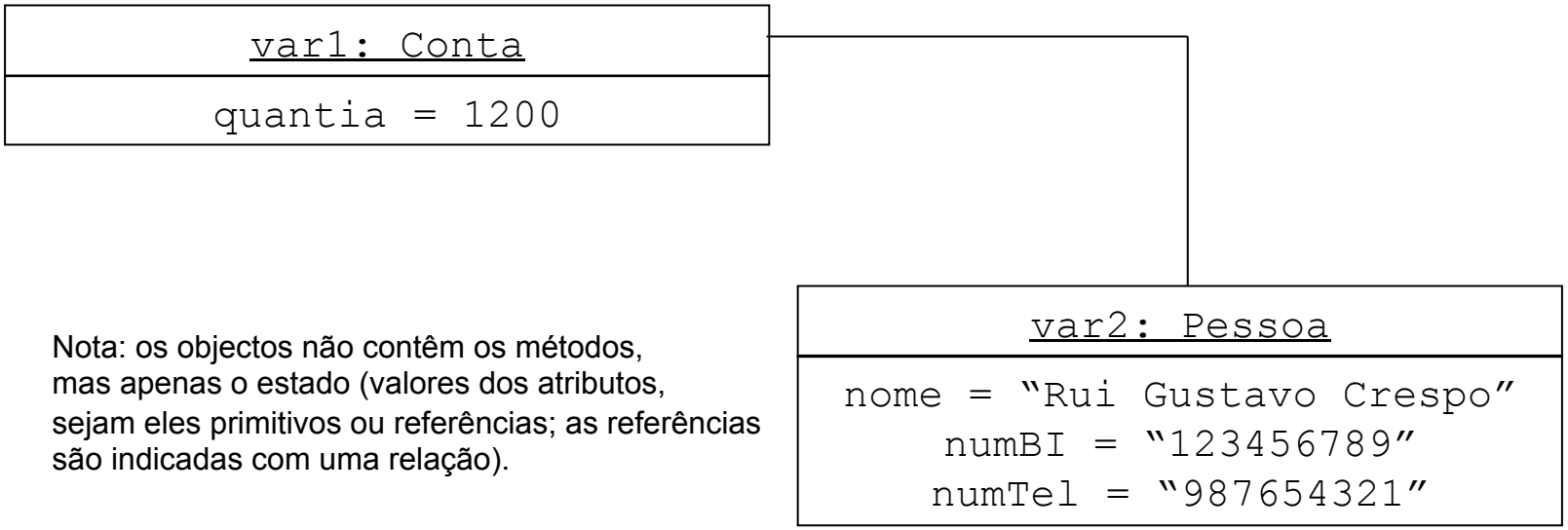
Relações – exemplo (1)

"Uma conta bancária contém sempre uma quantia e pertence a uma pessoa. A pessoa tem um nome, telefone e um número de BI. Na conta pode ser depositado ou levantado dinheiro. Sempre que entender, o dono pode consultar o saldo da conta".



Objectos – UML

- Representado por um rectângulo, dividido em 2 zonas:
  - idObjecto:IdClasse (ou apenas :IdClasse)
  - Inicialização dos atributos, um por linha, na forma Id=Init



Nota: os objectos não contêm os métodos, mas apenas o estado (valores dos atributos, sejam eles primitivos ou referências; as referências são indicadas com uma relação).

Herança – definição (1)

- **Princípio Aberto-Fechado**
  - Fecho: ao desenhar uma classe, todos os atributos e métodos devem estar desenvolvidos para que o programa possa ser usado sem alterações.
  - Abertura: classes devem estar abertas para extensão, por forma a incorporar alterações com o mínimo impacto no sistema.

Herança – definição (2)

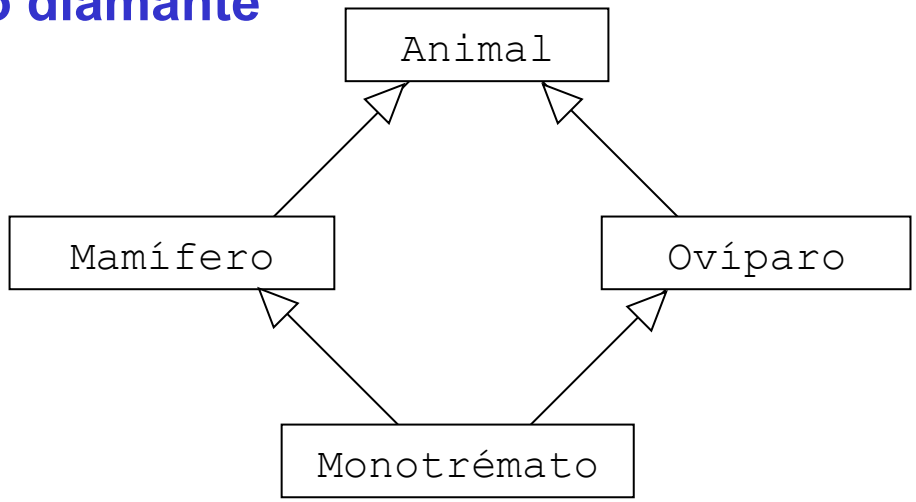
- A herança é um mecanismo em que a **subclasse** constitui uma especialização da **superclasse**. A superclasse pode ser vista como **generalização** das subclasses.
- A herança é dita como uma relação **"is-a"**.
- As subclasses herdam os atributos e métodos das superclasses. Os métodos herdados podem ser modificados. Novos atributos e métodos podem ser adicionados às subclasses.

Herança – definição (3)

- O **polimorfismo** ocorre quando há múltipla definição, ou **redefinição**, de métodos da superclasse nas subclasses, com a mesma assinatura.
- Em OO o polimorfismo é normalmente implementado através de **ligação dinâmica**, i.e., o método a ser executado é determinado apenas em tempo de execução (e não em tempo de compilação).

Herança – definição (4)

- Na **herança simples** cada subclasse tem apenas uma superclasse (directa).
- Na **herança múltipla** uma subclasse pode ter mais do que uma superclasse (directa).
  - **Problema do diamante**



Herança – definição (5)

- Vantagens da herança:
  - Maior legibilidade, pois as superclasses descrevem os aspectos comuns.
  - Facilita alterações, porque normalmente estas incidem apenas nos aspectos particulares.
  - Promovem reutilização do código das superclasses.
- Inconvenientes da herança:
  - Obriga à detecção dos aspectos comuns.

Herança – exemplo (1)

"Uma conta à ordem não recebe juros. A conta a prazo recebe os juros ao fim do prazo, salvo seja movimentada antes do final do período de imobilização reiniciando-se nessa data o período de imobilização".

Subclasses

- ContaOrdem (superclasse: Conta)
- ContaPrazo (superclasse: Conta)

Herança – exemplo (2)

Atributos adicionados

- ContaOrdem: --
- ContaPrazo: juro (tipo float), inicio (tipo long), periodo (tipo int)

Métodos alterados

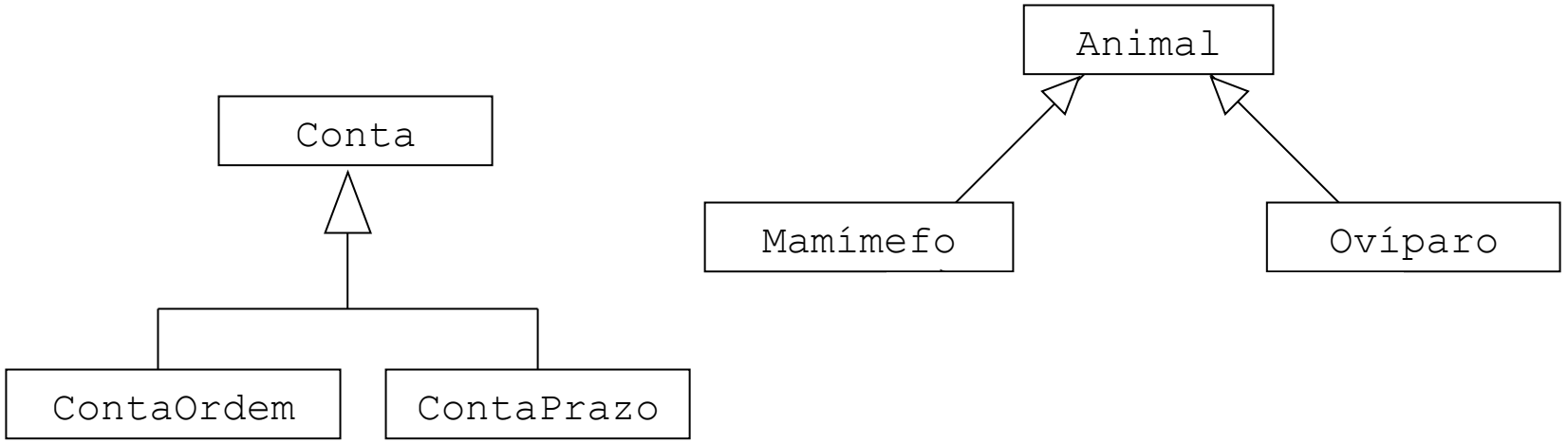
- ContaOrdem: --
- ContaPrazo: levantamento (parâmetro: quantia a levantar)

Métodos adicionados

- ContaOrdem: --
- ContaPrazo: vencimentoJuro

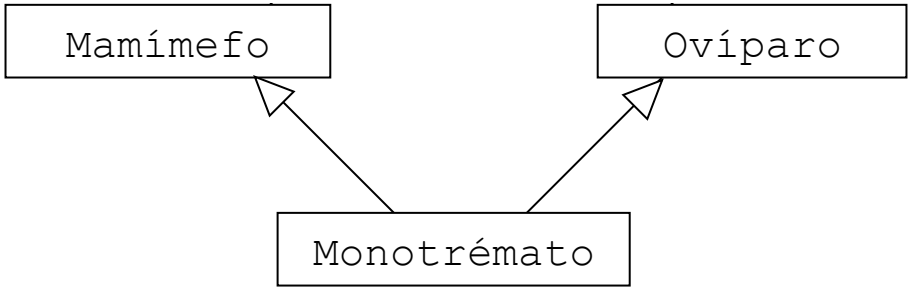
Herança – UML (1)

- A herança simples é representada por uma seta não preenchida das subclasses para a superclasse.



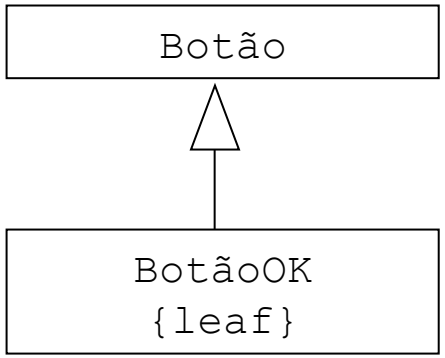
Herança – UML (2)

- A herança múltipla é representada por uma seta não preenchida da subclasse para as superclasses.



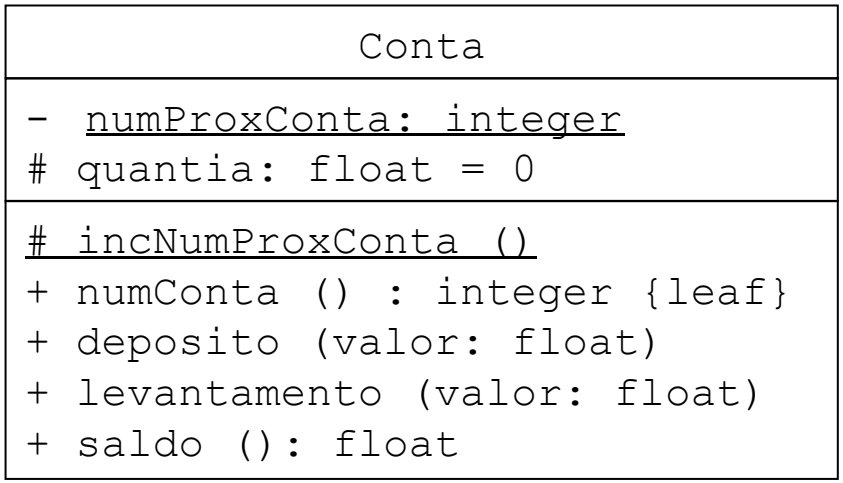
Herança – UML (3)

- **Classe não extensível**:
  - **Classe sem superclasses**: representada com a propriedade {root} escrita por baixo do identificador da classe.
  - **Classe sem subclasses**: representada com a propriedade {leaf} escrita por baixo do identificador da classe.



Herança – UML (4)

- O UML permite ainda especificar que um determinado método não pode ser redefinido em subclasses. Tais métodos são representados com a propriedade {leaf} escrita depois da assinatura do método.



Herança – UML (5)

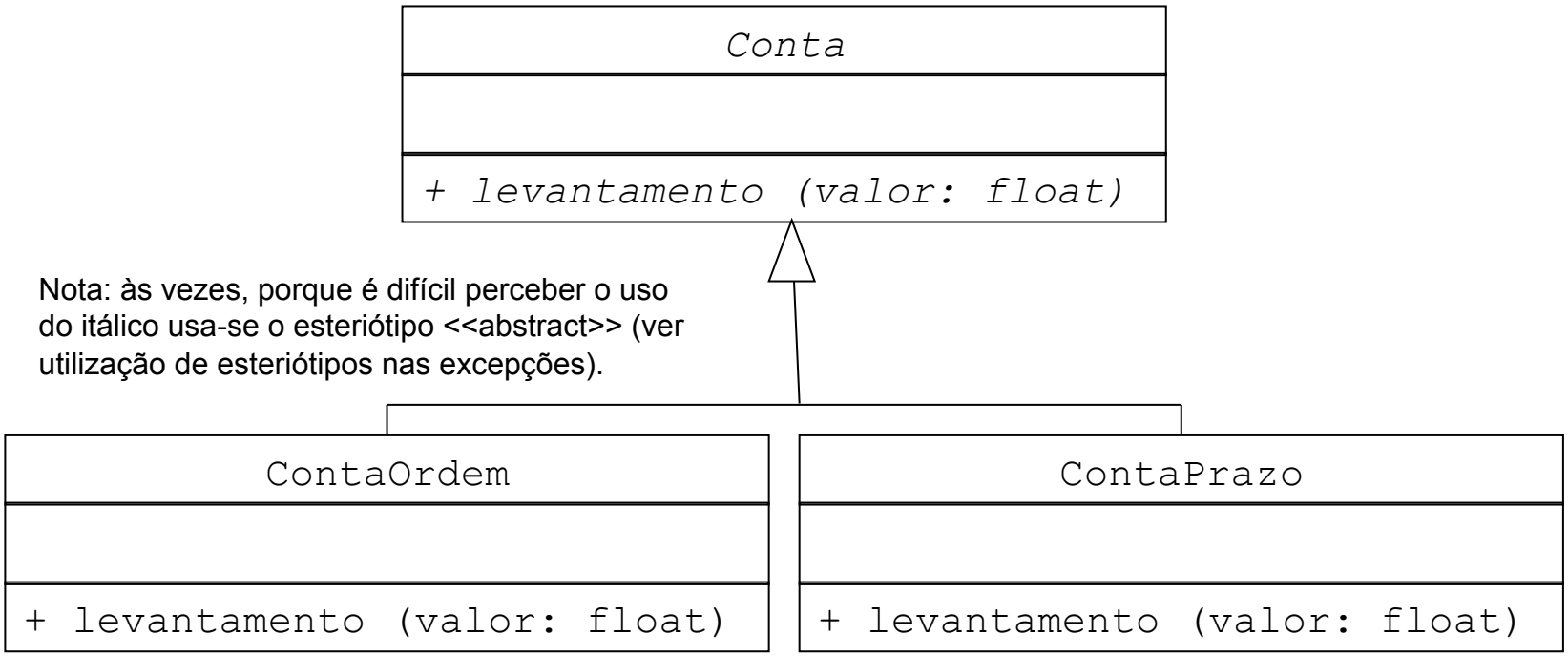
- Visibilidade de atributos e métodos representada por um caractere antes do identificador, que determina as permissões de acesso:
  - **public:** acessível fora da classe (+)
  - **private**: acessível apenas na classe (−)
  - **protected**: acessível na classe e suas subclasses (#)
  - **package**: acessível nas classes do mesmo pacote (~)

Métodos e classes abstractas – definição

- Um **método abstracto** é um método que não tem implementação (é apenas um protótipo).
- Uma **classe abstracta** é uma classe que não pode ser instanciada.
  - Uma classe que tem pelo menos um método abstracto (definido na própria classe ou herdado de uma superclasse, directa ou indirecta, e não implementado), é uma classe abstracta.

Métodos e classes abstractas – UML

- Os métodos/classes abstractas são representados com a assinatura/identificador em itálico.



Excepções – definição (1)

- Frequentemente as aplicações informáticas são sujeitas a situações anómalas:
  - Erros matemáticos (por exemplo, divisões por 0).
  - Dados indicados em formato inválido (por exemplo, inteiro inserido com caracteres inválidos).
  - Tentativa de acesso a uma referência nula.
  - Abertura para leitura de ficheiro inexistente.
  - …

Excepções – definição (2)

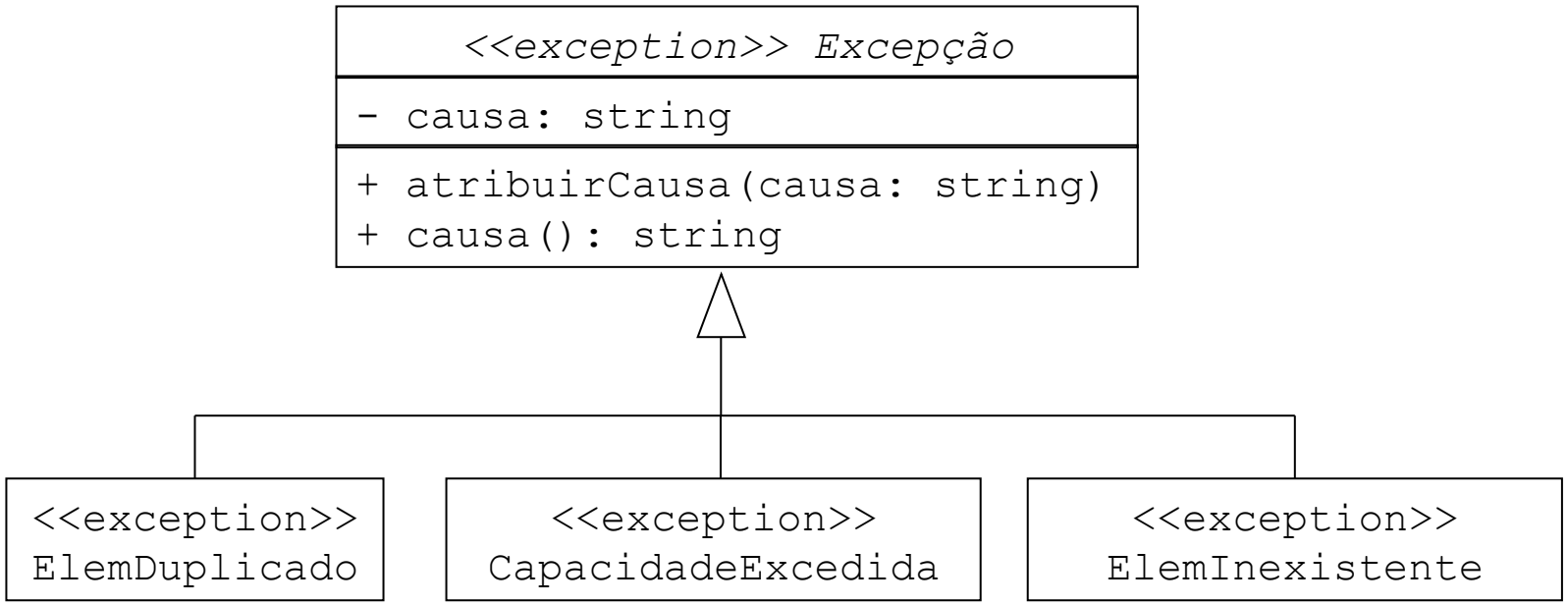
- Uma **excepção** é um sinal lançado ao ser identificada uma condição que impede a execução normal do programa.
  - O sinal é síncrono se ocorrer directamente como resultado de uma instrução em particular.
  - Caso contrário, é assíncrono.
- As excepções podem ser tratadas de formas diversas:
  - Termina o programa com mensagem de aviso e impressão do estado (inaceitável para sistemas críticos).
  - Geridas em locais específicos, designados por **manipuladores** (*handlers*).

Excepções – definição (3)

- Vantagens:
  - Código limpo sem verificação de todos os possíveis erros antes da instrução pretendida.
  - Disponibiliza um mecanismo explícito de assinalar situações de erro, sem recorrer a flags ou efeitos colaterias sobre o valor das variáveis.
  - As condições de erro que um método pode assinalar ficam explícitas na assinatura do método.

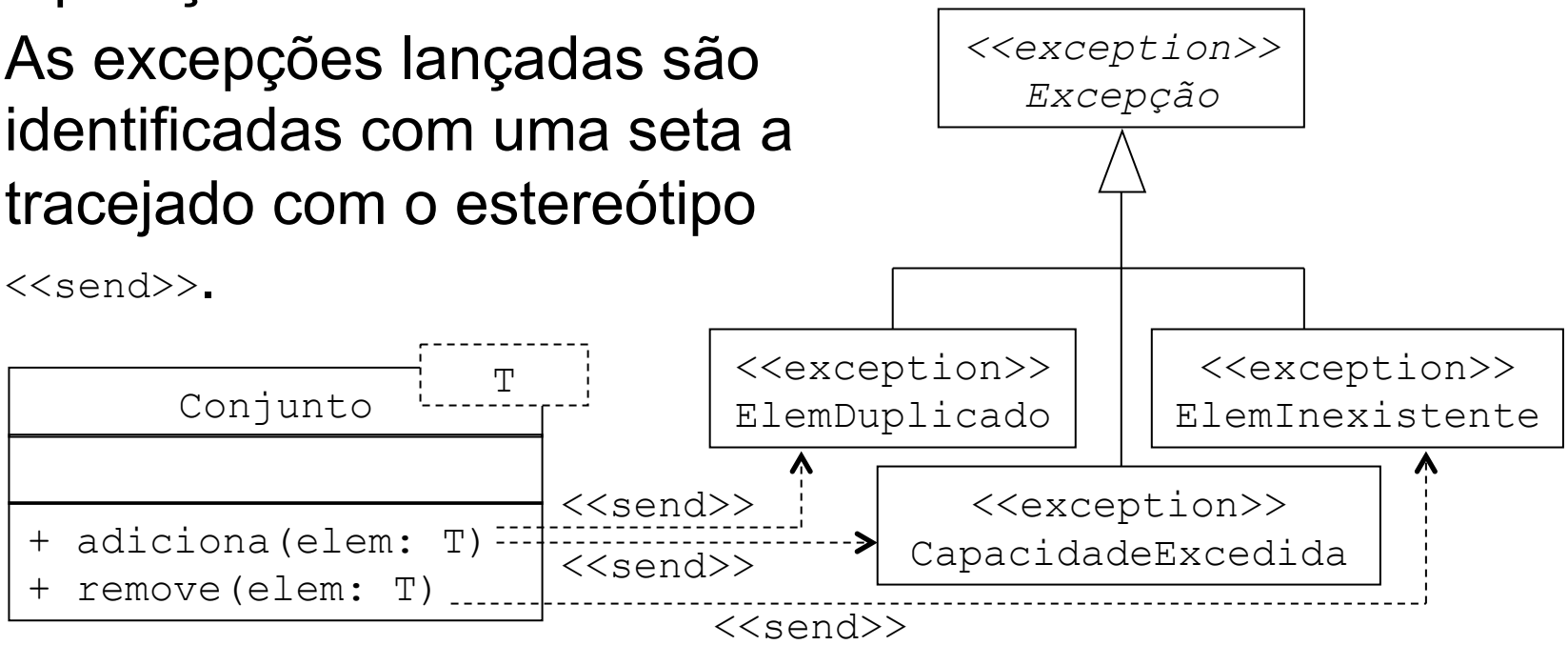
Excepções – UML (1)

- As excepções são representadas como classes com o estereótipo `<<exception>>`.



Excepções – UML (2)

- As classes de excepções modelam as excepções que um objecto pode lançar na execução das suas operações.
- As excepções lançadas são identificadas com uma seta a tracejado com o estereótipo `<<send>>`.



Interfaces e Pacotes – definição

- As interfaces e os pacotes são mecanismos úteis para desenvolvimento de sistemas de grande dimensão:
  - As **interfaces** separam a especificação da implementação.
  - Os **pacotes** agrupam elementos distintos num único grupo.

Interfaces – definição (1)

- Uma **interface** é um conjunto de protótipos de métodos (sem implementações) que especifica um serviço bem definido:
  - As interfaces não podem conter atributos, mas podem conter constantes.
  - A implementação duma interface é realizada pela **classe concreta**. Na **implementação** ou **concretização**:
    - Determina-se os atributos necessários à correcta implementação da interface.
    - Descreve-se o código dos métodos das interface.

Interfaces – definição (2)

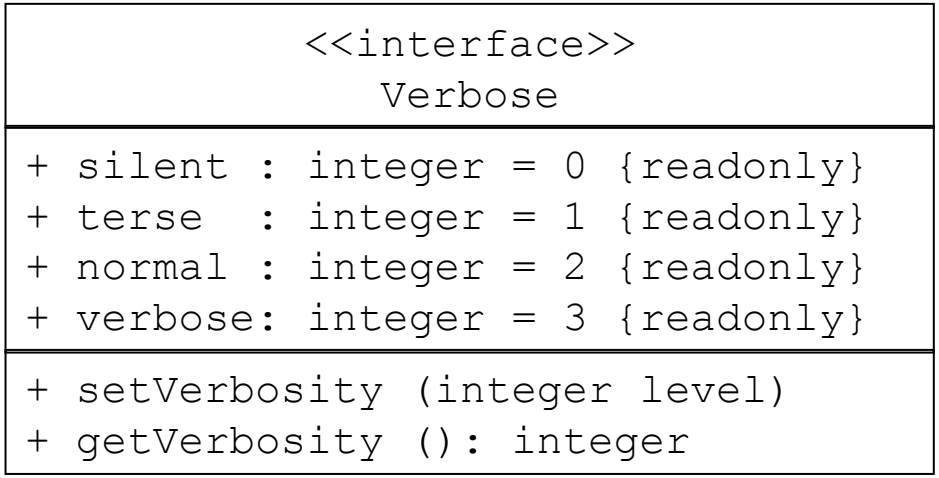
- Uma interface pode herdar as definições de outra interface.
  - Interfaces podem usar polimorfismo.
  - Se uma classe concretizar mais de uma interface, e várias interfaces contiverem métodos com a mesma assinatura, basta definir uma única implementação.
- Uma interface não pode ser instanciada.

Interface vs classe abstracta

- Semelhanças:
  - Não podem ser instanciadas directamente.
- Diferenças:
  - Uma classe abstracta pode ter atributos (constantes ou não), enquanto que uma interface apenas pode ter constantes.
  - Uma classe abstracta pode conter métodos implementados.

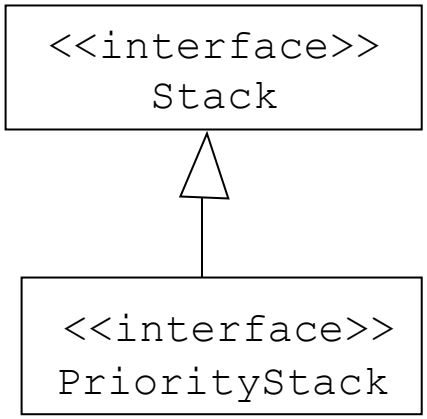
Interfaces – UML (1)

- As interfaces são representadas como classes com o estereótipo `<<interface>>`.
- Uma interface, para além das operações oferecidas, só pode conter **atributos constantes**, representados com a propriedade `{readonly}`.



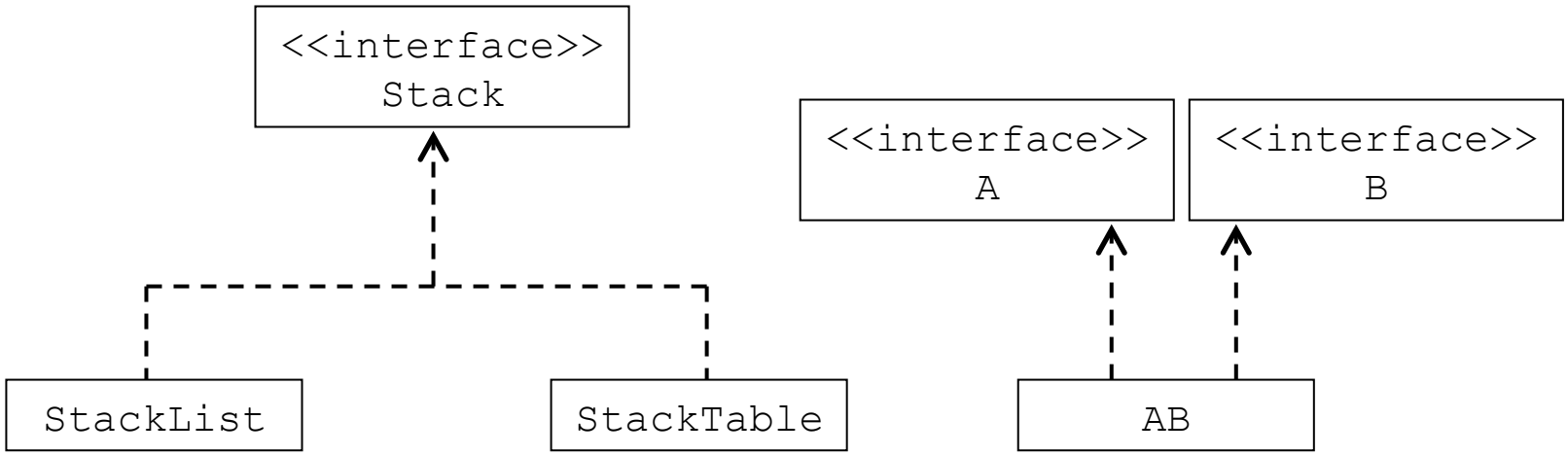
Interfaces – UML (2)

- As interfaces podem participar em relações de herança (simples ou múltipla).



Interfaces – UML (3)

- A classe de concretização está associada à interface por uma relação de **realização**, representada graficamente por uma seta a tracejado.
- Uma classe pode realizar uma ou mais interfaces.



Pacotes – definição (1)

- Um pacote é um mecanismo de agrupamento de informação:
  - Os pacotes podem conter outros pacotes, classes, interfaces e objectos.
  - O pacote forma um **espaço de nomes** (*namespace*), logo os seus membros têm de ter identificadores únicos (por exemplo, num pacote não pode haver duas classes com o mesmo nome).
  - O identificador dum pacote pode consistir num **nome simples** ou num **nome qualificado**. O nome qualificado corresponde ao nome simples prefixado pelo nome do pacote onde este reside, se existir. É usual usar-se `::` para separar os nomes simples.

Pacotes – definição (2)

- A **importação** adiciona o conteúdo do pacote importado ao espaço de nomes do pacote importador, de tal forma que passa a ser desnecessário usar o seu nome qualificado.
- A importação não é transitiva.
  - Se o pacote B importar o pacote A, e o pacote C importar o pacote B, no pacote C não são importados os elementos do pacote A.
  - Caso o pacote C queira importar igualmente os elementos do pacote A, devem ser inseridas duas directivas de importação, uma para o pacote A e outra para o pacote B.

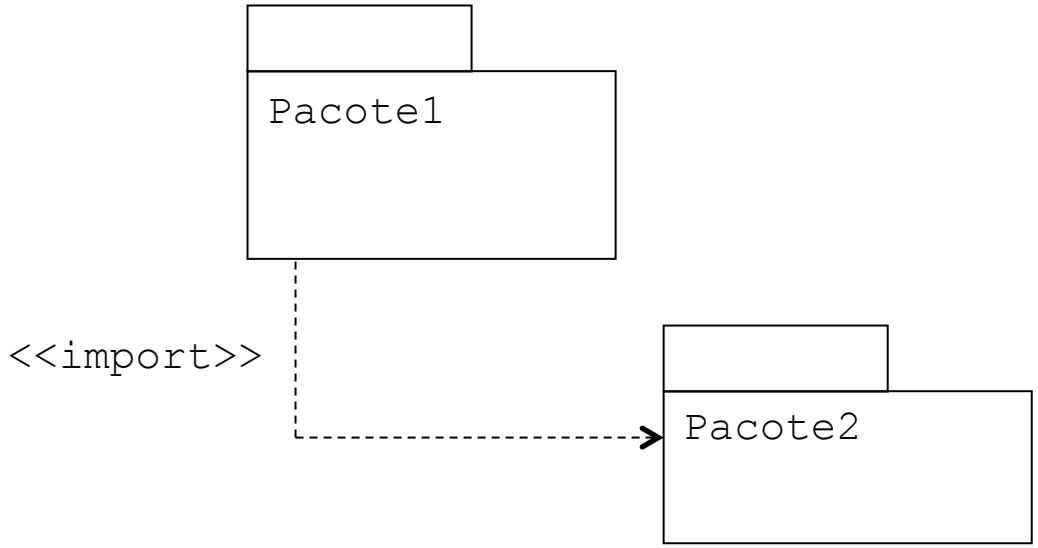
- O conjunto de métodos de um pacote é referido por **API** (*Application Programmer Interface*).

Pacotes – UML (1)

- Os pacotes são representados por pastas, identificados com o respectivo nome.

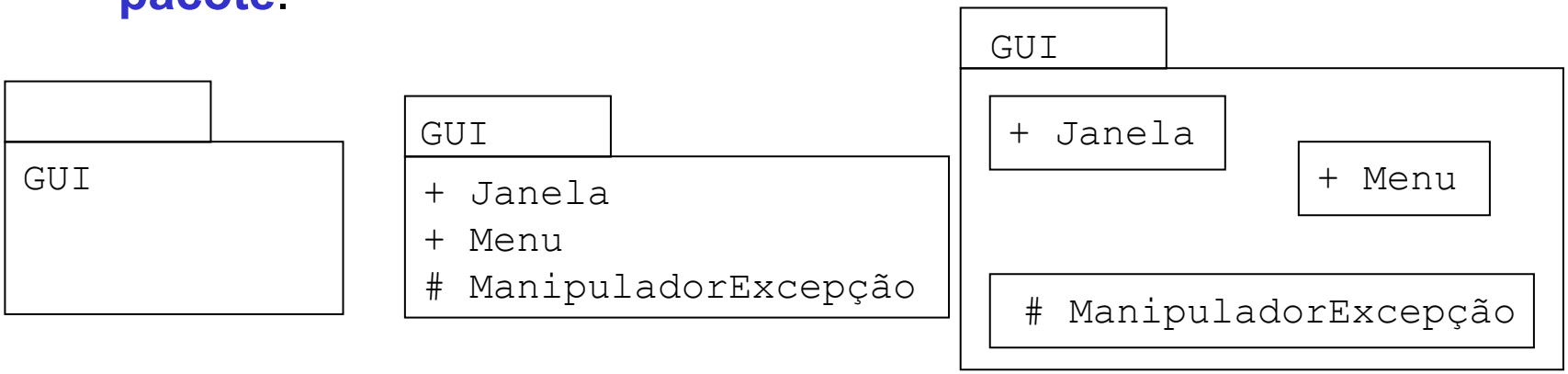
Pacotes – UML (2)

- A importação é representada por uma seta tracejada com o estereótipo `<<import>>`.



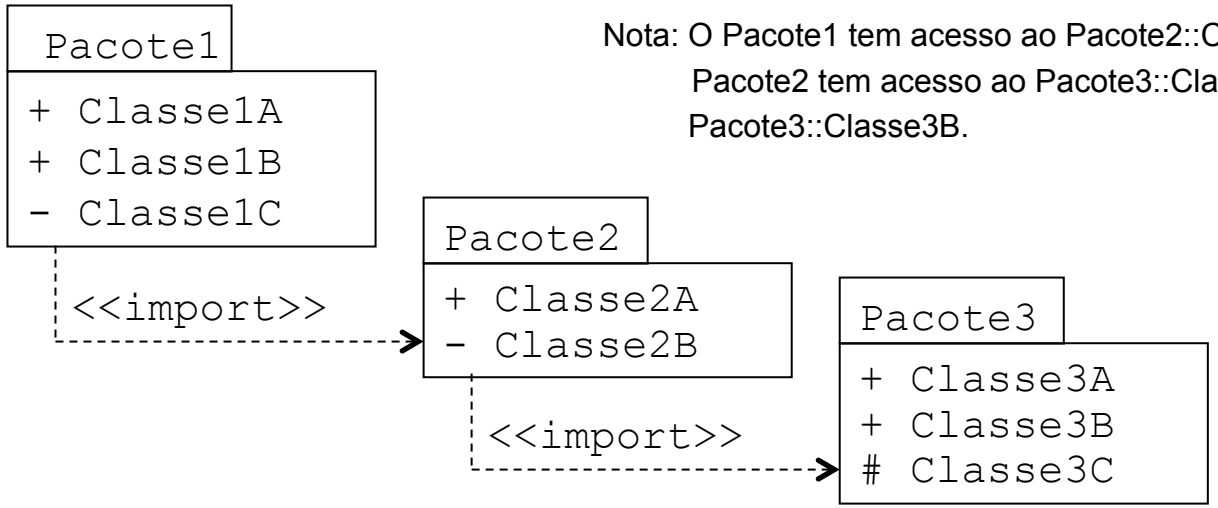
Pacotes – UML (3)

- Os elementos dum pacote podem ter visibilidade diversa:
  - **public**: elementos acessíveis ao pacote e aos pacotes que o importam (+)
  - **private**: elementos inacessíveis fora do pacote (-)
  - **protected**: elementos acessíveis no pacote e subpacotes (#)
  - **package**: elementos acessíveis apenas no pacote (~)
- As partes públicas dum pacote constituem a **interface do pacote**.



Pacotes – UML (4)

- Um pacote **exporta** apenas a sua parte pública.
- As partes exportadas por um pacote são visíveis apenas para os pacotes que explicitamente o importam.



Nota: O Pacote1 tem acesso ao Pacote2::Classe2A, o Pacote2 tem acesso ao Pacote3::Classe3A e Pacote3::Classe3B.

Diagramas UML

- UML v2 disponibiliza 13 diagramas, dos quais apenas os seguintes são abordados na cadeira de PO:
  - Modelação estrutural:
    - **Diagrama de classes**: classes, interfaces e relações.
    - **Diagrama de objectos**: objectos e relações.
    - **Diagrama de pacotes**: pacotes.
  - Modelação de comportamento:
    - **Diagrama de actividades**: mostra a dinâmica de sociedades de objectos, ou o controlo de fluxo de um método.

Diagrama de classes – definição

- O **diagrama de classes** é usado para modelar:
  - Colaboração entre classes.
  - Esquemas de bases de dados:
    - Normalmente, os sistemas contêm objectos persistentes.
    - O diagrama de classes é um superconjunto do **diagrama ER** (*entity-relationship*). Os diagramas ER focam apenas os dados, enquanto que os diagramas de classes permitem, para além dos dados, modelar comportamento.
- Tipicamente, o diagrama de classes contém classes, interfaces e relações. Pode ainda conter pacotes e objectos.
- É o diagrama mais comum na modelação de sistemas orientados a objectos.

Diagrama de classes – UML (1)

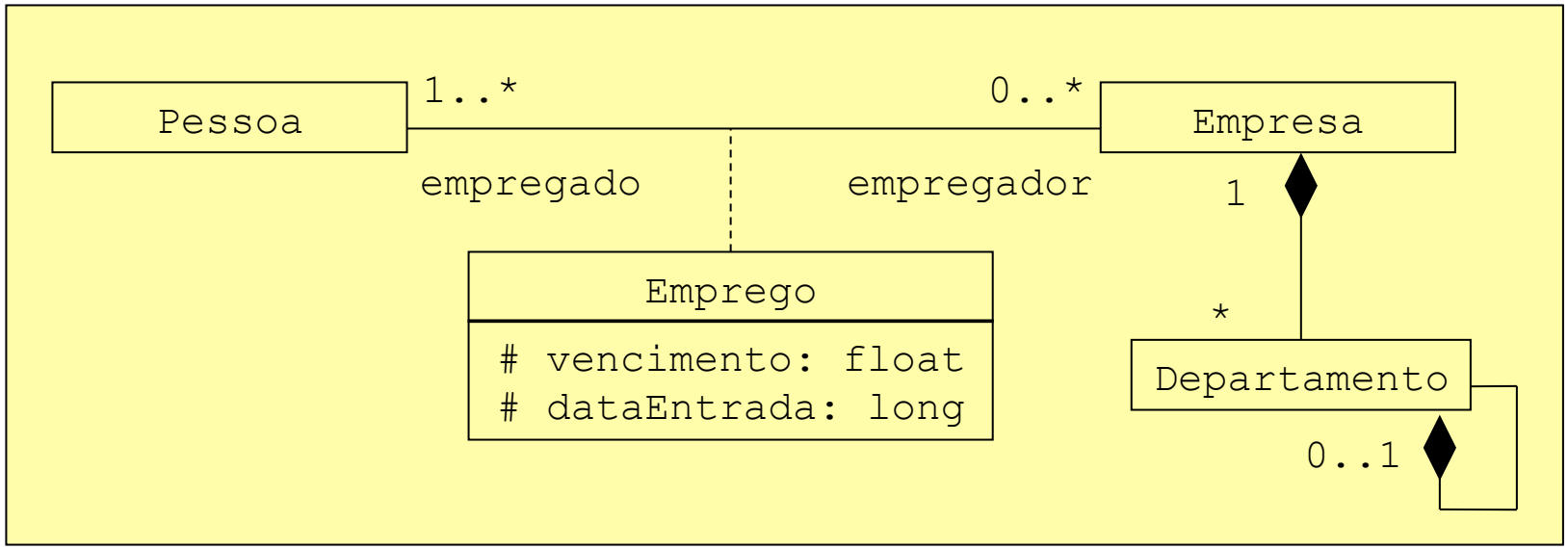


Diagrama de classes – UML (2)

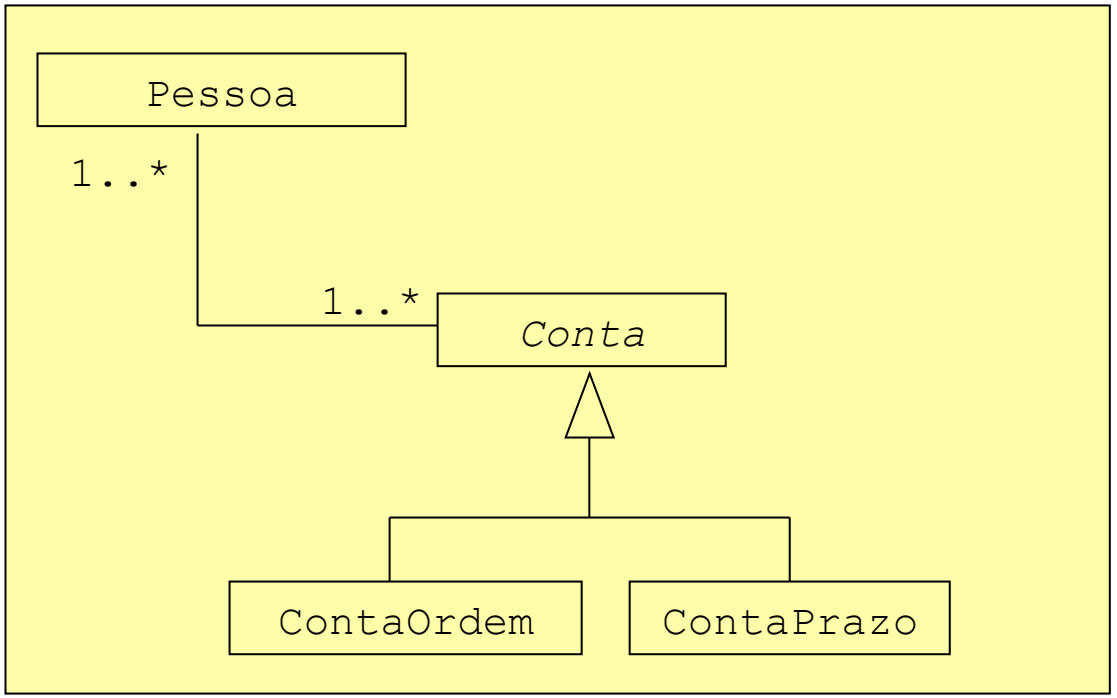


Diagrama de objectos – definição

- O **diagrama de objectos** contém um conjunto de objectos e as suas relações (num determinado instante no tempo).
- Enquanto que com o diagrama de classes é possível especificar completamemente a semântica das abstracções e correspondentes relações num sistema, tal é impossível no diagrama de objectos.
  - O diagrama de objectos apenas consegue capturar subconjuntos interessantes dos objectos do sistema.

Diagrama de objectos – UML

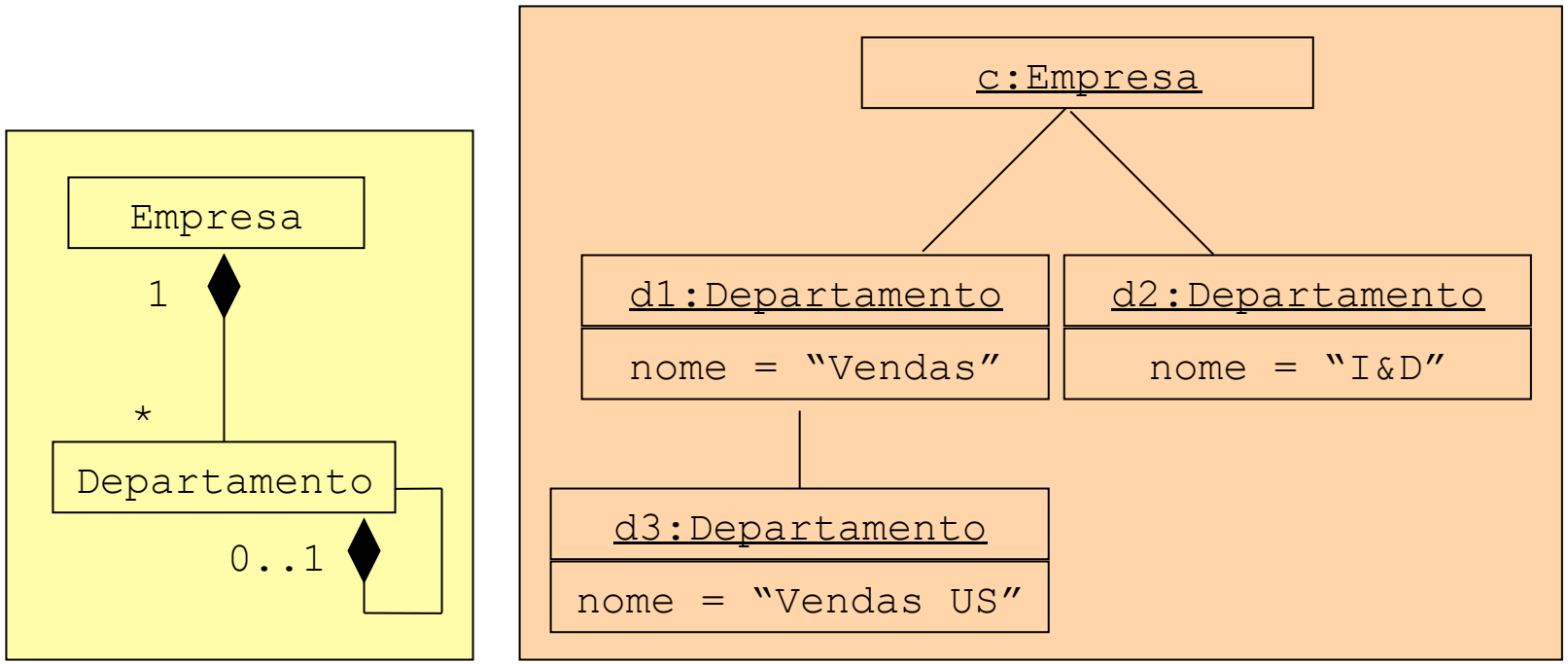
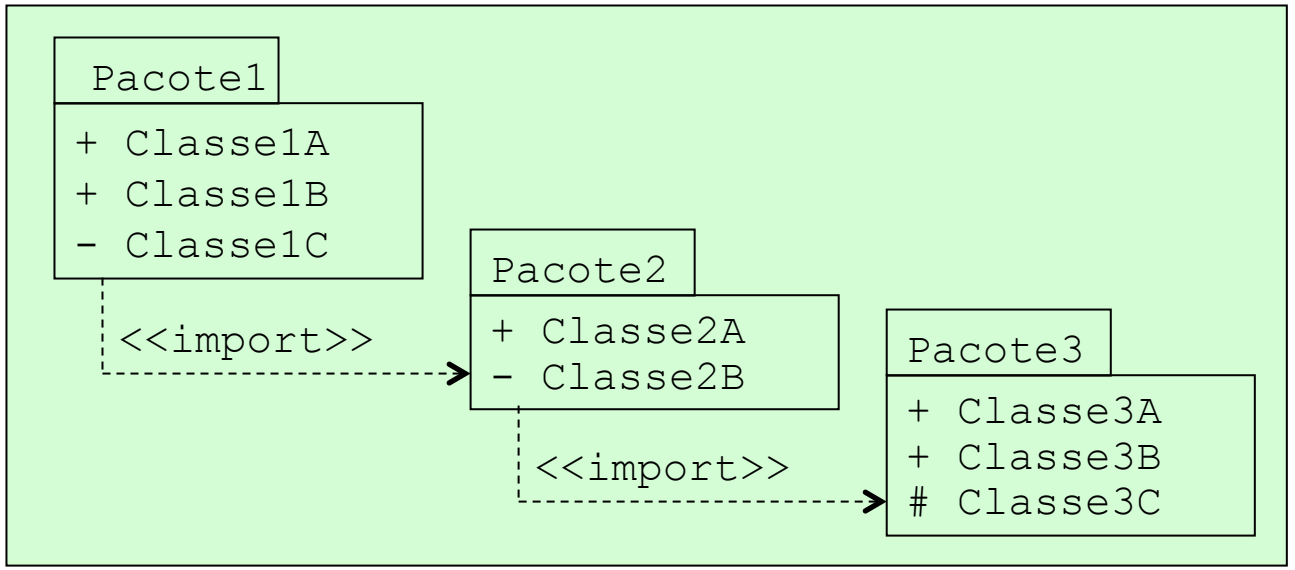


Diagrama de pacotes – UML

- O diagrama de pacotes mostra a decomposição do modelo em unidades de organização (pacotes) e correspondentes dependências (importações).



Curiosidade: não sai no exame.

Diagrama de actividades – definição

- O **diagrama de actividades** é usado para modelar:
  - A dinâmica de sociedades de objectos.
  - O controlo de fluxo de um método.
    - Neste contexto o diagrama de actividades pode ser visto como o equivalente OO dos **fluxogramas**.

Curiosidade: não sai no exame.

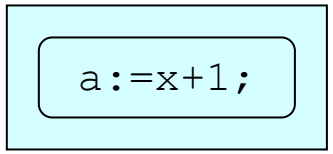
Diagrama de actividades – UML (1)

- Um diagrama de actividades contém:
  - Estados:
    - Estado inicial
    - Estado final
  - Actividades:
    - Actividade atómica
    - Actividade composta
    - Actividade protegida e manipuladores de excepções associados
  - Fluxos entre actividades:
    - Fluxo simples
    - Fluxo alternativo
    - Fluxo paralelo
  - Objectos e fluxo associado

Curiosidade: não sai no exame.

Diagrama de actividades – UML (2)

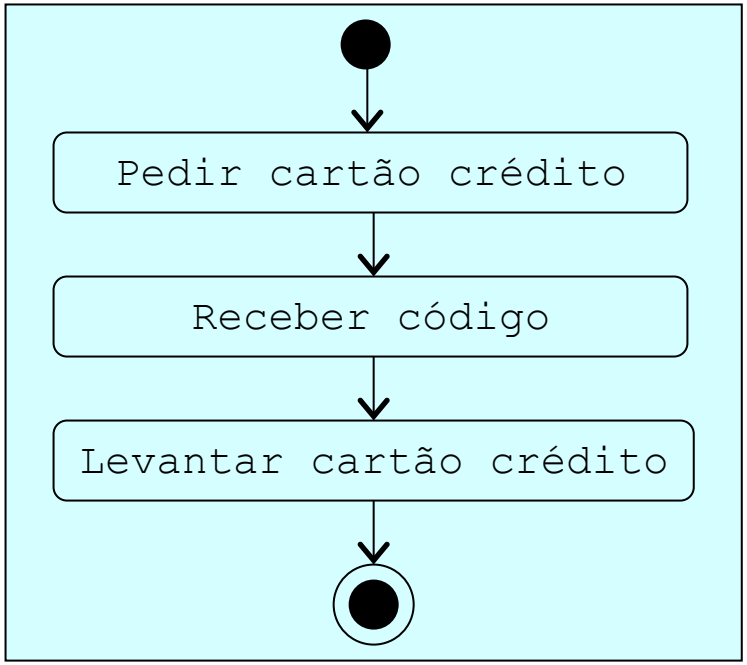
- As **actividades** podem ser:
  - **Atómicas**, i.e., não divisíveis
    - avaliação de expressões
    - atribuição dum valor/expressão a um atributo
    - chamada dum método num objecto
    - enviar um sinal a um objecto
    - criar ou destruir objectos
  - **Compostas**, i.e., podem ser representadas por outros diagramas de actividades.
- As actividades são representadas por um rectângulo com cantos arredondados.
- São permitidas actividades aninhadas.



Curiosidade: não sai no exame.

Diagrama de actividades – UML (3)

- O **fluxo simples** é representado por setas.
- O **estado inicial** é representado por um círculo preenchido e o **estado final** um círculo preenchido dentro de um outro círculo.



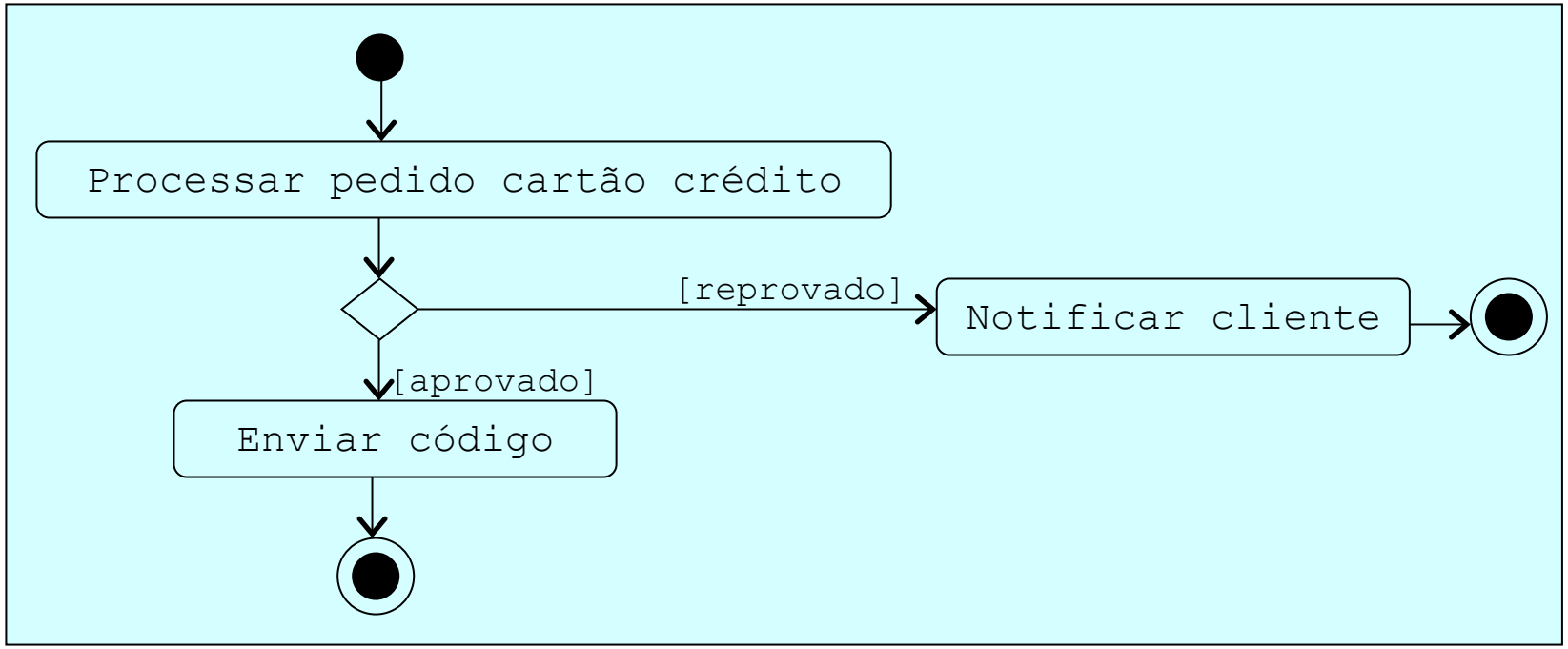
Curiosidade: não sai no exame.

Diagrama de actividades – UML (4)

- O **fluxo alternativo** é representado por um diamante não preenchido de onde parte uma ramificação:
  - A ramificação pode resultar em 2 ou mais fluxos simples.
  - Cada fluxo tem uma guarda (expressão Booleana).
  - As guardas devem ser mutuamente disjuntas (prevenir ambiguidade) e devem cobrir todas as possibilidades (prevenir congelamento da transição).
  - Pode usar-se o `else` para representar o fluxo a seguir caso nenhuma das outras guardas seja verdadeira.

Curiosidade: não sai no exame.

Diagrama de actividades – UML (5)



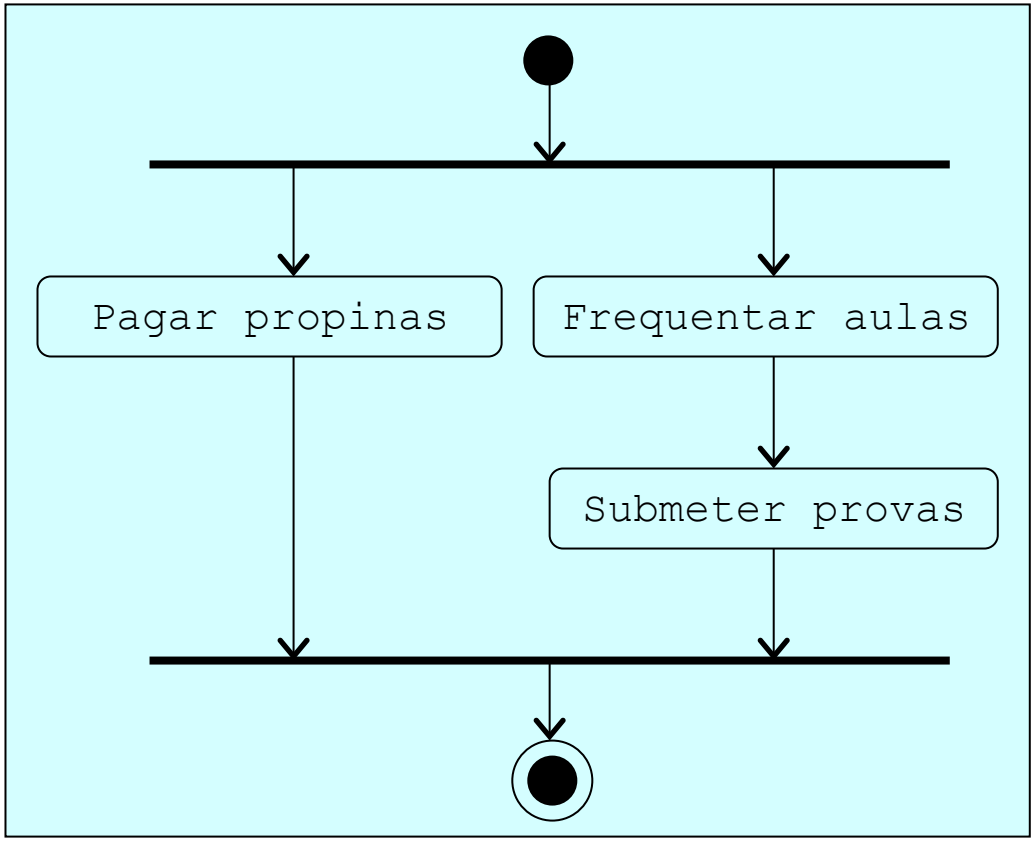
Curiosidade: não sai no exame.

Diagrama de actividades – UML (6)

- O **fluxo paralelo** é representado com uma barra de sincronização, representada por uma barra negra, para especificar o início (*fork*) e fim (*join*) do paralelismo.
- Num diagrama de actividades o número de fluxos iniciados numa barra de sincronização deve igualar o número de fluxos que finaliza a mesma.

Curiosidade: não sai no exame.

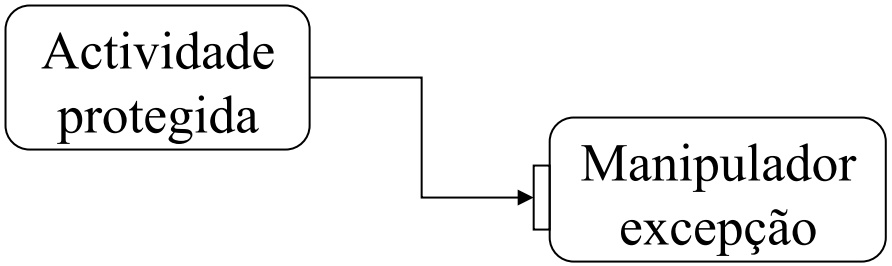
Diagrama de actividades – UML (7)



Curiosidade: não sai no exame.

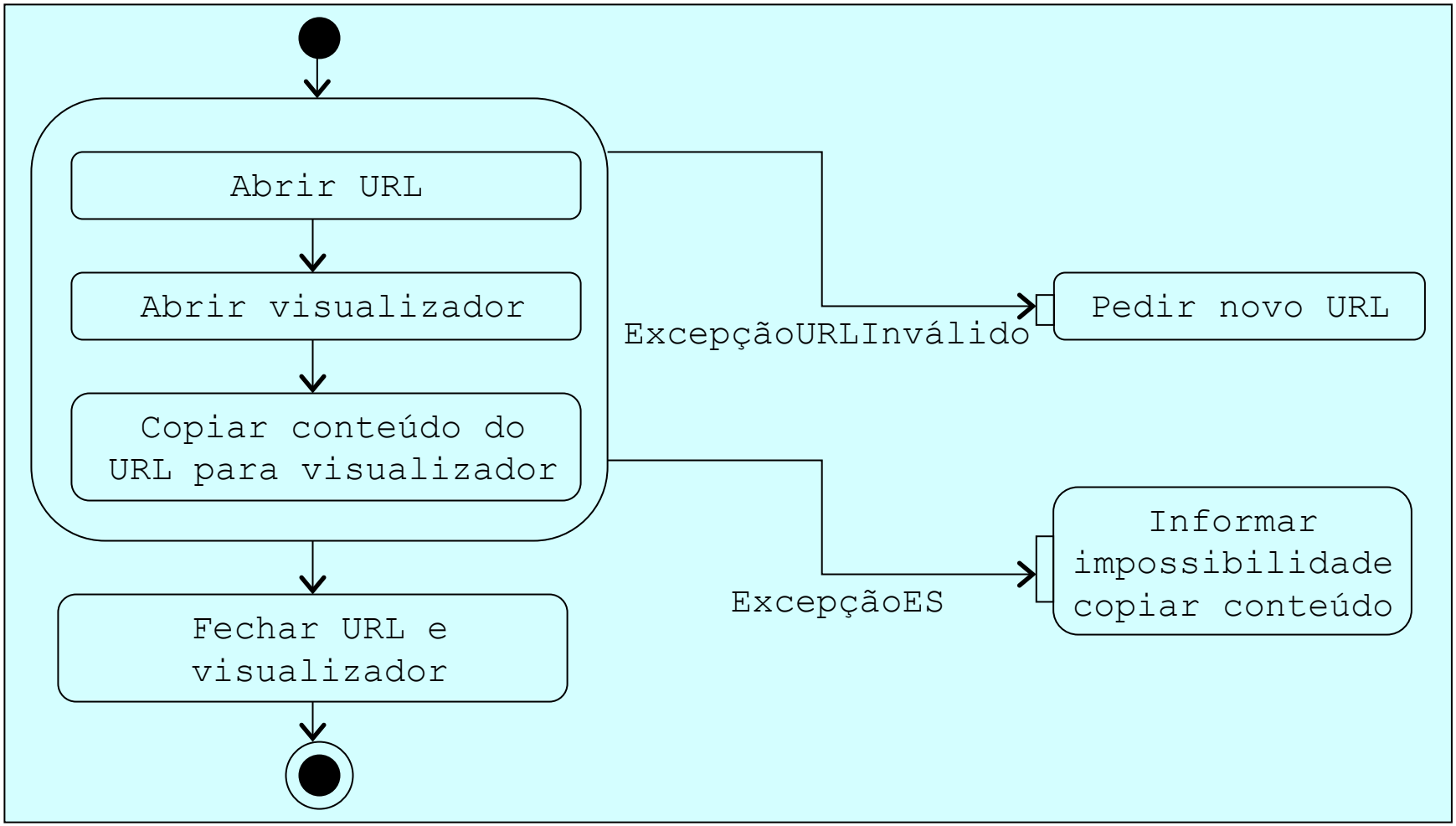
Diagrama de actividades – UML (8)

- O lançamento de excepções é representado por um "raio" partindo da **actividade protegida** em direcção ao **manipulador de excepção**.
- Pode haver mais que uma excepção, cada uma processada pelo seu manipulador.



Curiosidade: não sai no exame.

Diagrama de actividades – UML (9)



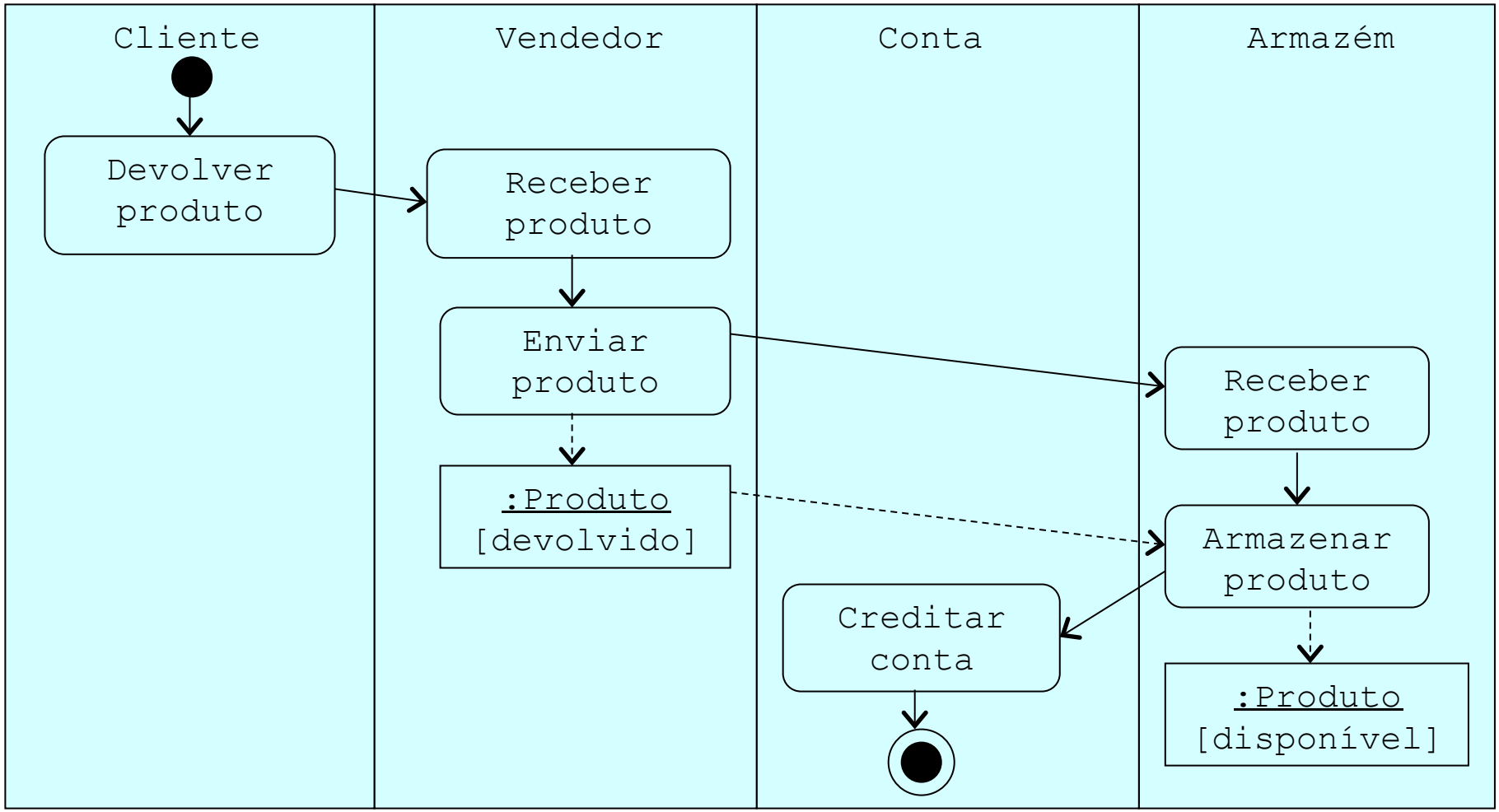
Curiosidade: não sai no exame.

Diagrama de actividades – UML (10)

- O fluxo de controlo associado a um diagrama de actividades pode envolver objectos.
- Os objectos são representados no diagrama de actividades, ligados por uma seta a tracejado a partir da actividade que os criou, modificou, ou destruio.
  - Tal representação é denominada **fluxo de objectos** pois representa a participação dum objecto no fluxo de actividades.
  - Para além da representação do seu fluxo é possível mostrar como o seu estado varia no percurso do mesmo. O estado é representado com o auxílio de parêntesis rectos por baixo do identificador do objecto.
- É usual organizar um diagrama de actividades em faixas que separam as diferentes actividades pela respectiva participação nas diferentes classes do sistema.

Curiosidade: não sai no exame.

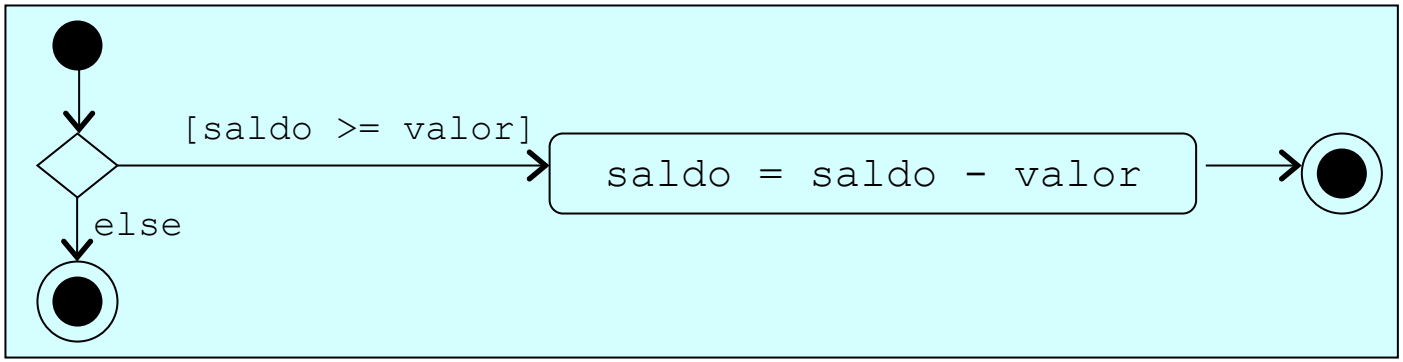
Diagrama de actividades – UML (11)



Curiosidade: não sai no exame.

Diagrama de actividades – UML (12)

- A modelação de um método num diagrama de actividades é particularmente interessante quando o comportamento do método é complexo e difícil de compreender apenas com recurso ao código que o implementa.




```
levantamento(valor: float) {
   if (saldo >= valor)
     saldo = saldo - valor;
}
```