Mapeamento de classes em tabelas

Estratégia mais comum - a que vamos adoptar!

Mapeamento de uma tabela por classe

- uma tabela para cada classe da hierarquia - estrutura final das tabelas mais próxima da hierarquia de classes

```
Membro(nº, nome, tipo)      Aluno(nº, te)      Aluno3C(nº, area)
Docente(nº, dataEnt)        Aluno1C(nº)
Funcionario(nº)             Aluno2C(nº)
```

- utilização de chaves estrangeiras para relacionar tabelas (*chave estrangeira*)
- colocação de um identificador de tipo na superclasse
  - permite identificar o tipo concreto dos objetos sem *joins*
  - melhorias na performance
- Problemas:
  - implementação potencialmente mais complexa
  - alguma penalização no desempenho

Mapeamento de relacionamentos

- Associações um-para-um
  - necessitam que uma chave estrangeira seja posta numa das duas tabelas, relacionando o elemento associado na outra tabela.
  - dependendo da navegabilidade da associação, assim será feita a disposição desta chave estrangeira (navegabilidade é sempre da tabela que possui a chave estrangeira para a tabela referenciada).
- Associações um-para-muitos,
  - adota-se a mesma técnica, mas...
  - a chave estrangeira deve ser posta na tabela que contém os objetos múltiplos (para onde se navega)
- Associações muitos-para-muitos
  - cria-se uma tabela intermédia de pares de chaves, identificando os dois lados do relacionamento.

Assumindo que o sistema está a funcionar correctamente (ou seja, como descrito na página 1), assinale as expressões que vão sempre verificar-se:

- ◯ (1)
- ◯ (2)
- ◯ (3)

DAO - Data Access Object

- Padrão para implementar a camada de persistência
- DAOs são classes que:
  - Persistem objectos em Bases de Dados
  - Criam objectos a partir da informação na Base de Dados
  - Encapsulam *queries* SQL
- Podem persistir uma classe ou várias classes
  - Não é obrigatório ter um DAO por classe
- São as *Facades* da Camada de Dados

JDBC - Java DataBase Connectivity

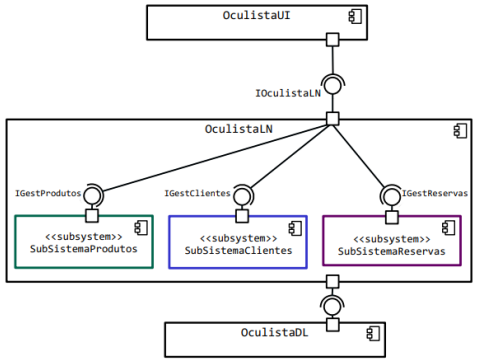
- A comunicação de aplicações Java com bases de dados necessita de:
  - Importar biblioteca para o projecto (através do IDE).
  - Importar no código packages respectivos.
  - Carregar a driver no código.
  - Criar statements e executar.
  - Fechar ligação.
- Transacções devem ser feitas de forma explicita.
- Driver não lida com todos os problemas:
  - É necessário lidar com *timeouts*, *sql injection*, etc.

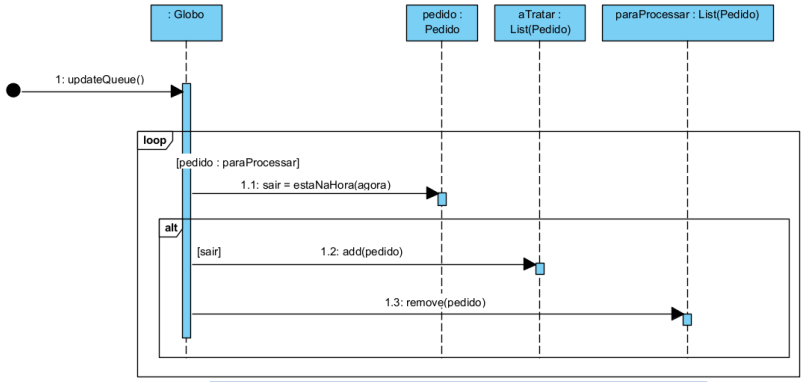
- Uma API para acesso a Bases de Dados, para aplicações Java
  - Independente da Base de Dados utilizada
- Essencialmente um conjunto de classes/interfaces Java distribuídas pelos pacotes: `java.sql` e `javax.sql`
- Exemplos:
  - Abrir uma ligação à Base de Dados (Connection):

```
Connection c = DriverManager.getConnection();
```

  - Criar uma query:

```
PreparedStatement ps =
  c.prepareStatement("SELECT * FROM TABLE WHERE ID < ?")
```





Assumindo que o sistema está a funcionar correctamente (ou seja, como descrito na página 1), assinale as expressões que vão ser sempre falsas:

- ◯ (4)
- ◯ (1)
- ◯ (2)

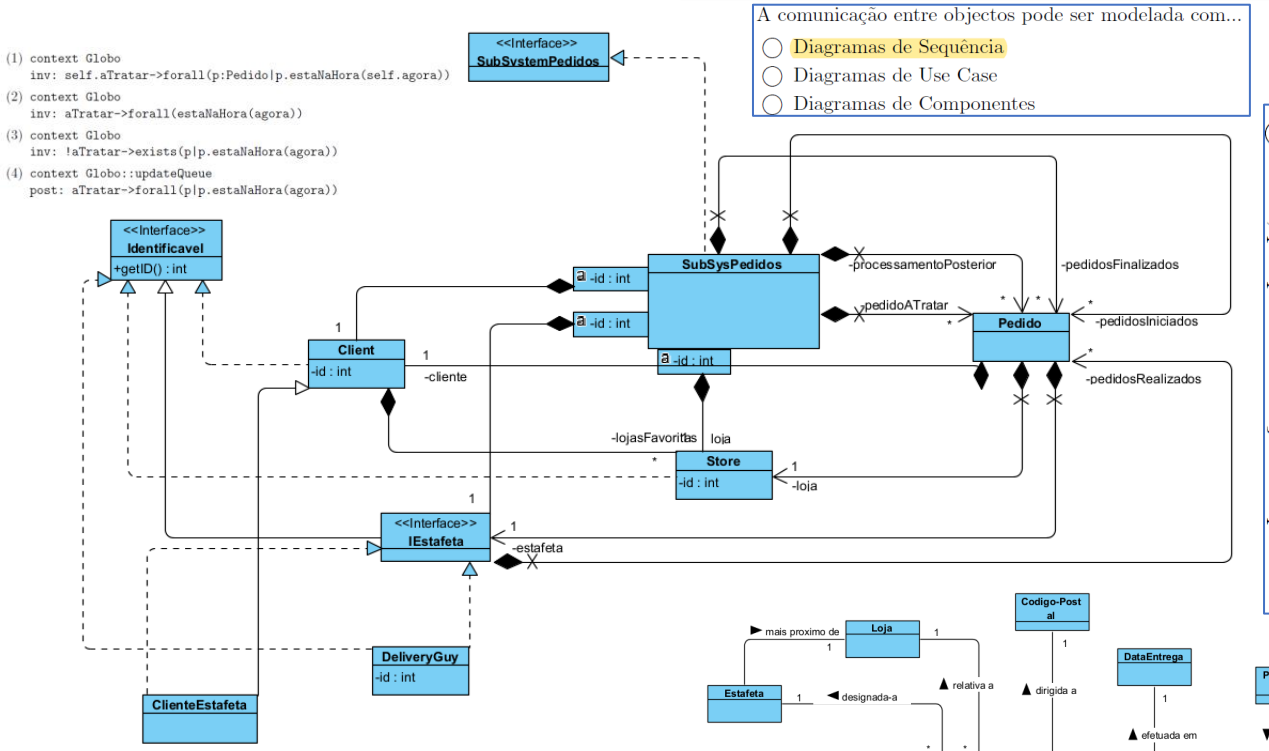
A comunicação entre objectos pode ser modelada com...

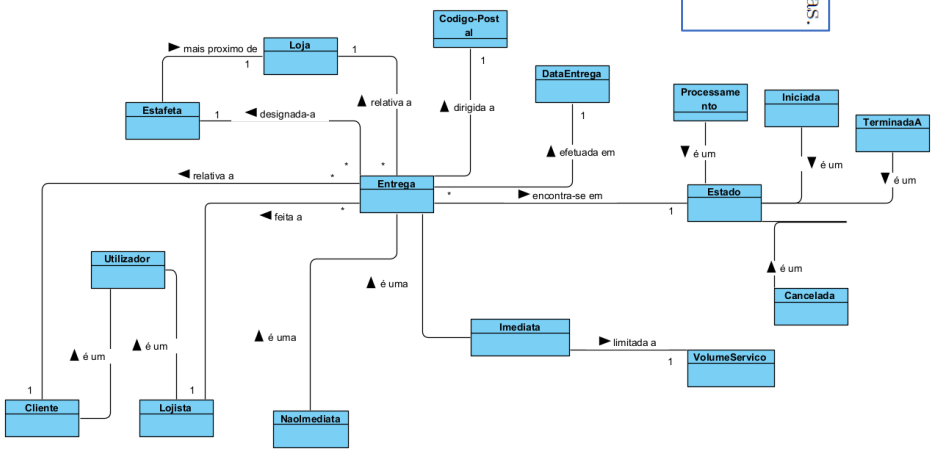
- ◯ Diagramas de Sequência
- ◯ Diagramas de Use Case
- ◯ Diagramas de Componentes

```
(1) context Globo
    inv: self.aTratar->forall(p:Pedido|p.estaNaHora(self.agora))
(2) context Globo
    inv: aTratar->forall(estaNaHora(agora))
(3) context Globo
    inv: !aTratar->exists(p|p.estaNaHora(agora))
(4) context Globo::updateQueue
    post: aTratar->forall(p|p.estaNaHora(agora))
```

Em UML todas as associações são binárias.

- ◯ Verdadeiro.
- ◯ Falso.
- ◯ Falso, porque há herança múltipla.





Interfaces

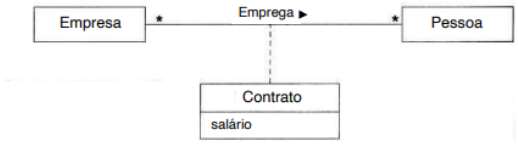
- Uma interface especifica um tipo abstracto - um conjunto de operações externamente visíveis que uma classe (ou componente, subsistema, etc.) deve implementar
- semelhante a classe abstracta só com operações abstractas e sem atributos nem associações
- separação mais explícita entre interface e (classes de) implementação
- interfaces são mais importantes em linguagens que têm herança simples de implementação e herança múltipla de interface (como em Java)

Modelos de Domínio

- O Modelo de Domínio é estático - não representa fluxos de dados
- O Modelo de Domínio representa o problema - **não inclui** o sistema (software/solução) a desenvolver
- As entidades no Modelo de Domínio são **apenas** candidatas a serem classes na solução
- As entidades no modelo de domínio podem ter atributos, mas devem ser de tipos simples (números, strings, etc.) e **nunca** outras entidades
  - Na dúvida, optar por entidades e relacionamentos, em vez de atributos

A UML é...
- ○ um processo de desenvolvimento.
- ○ definida numa norma OMG.
- ○ pensada especificamente para a programação em Java.

Classes de Associação



- A relação entre cada Empresa e cada um dos seus funcionários é caracterizada por um contrato.
- Cada Pessoa, pode ter estado contratada por várias Empresas e para cada uma há um contrato diferente.
- O Contrato não é característica da Empresa, nem da Pessoa, mas da relação entre ambas.

mas... **Dois contratos diferentes com a mesma empresa?!**

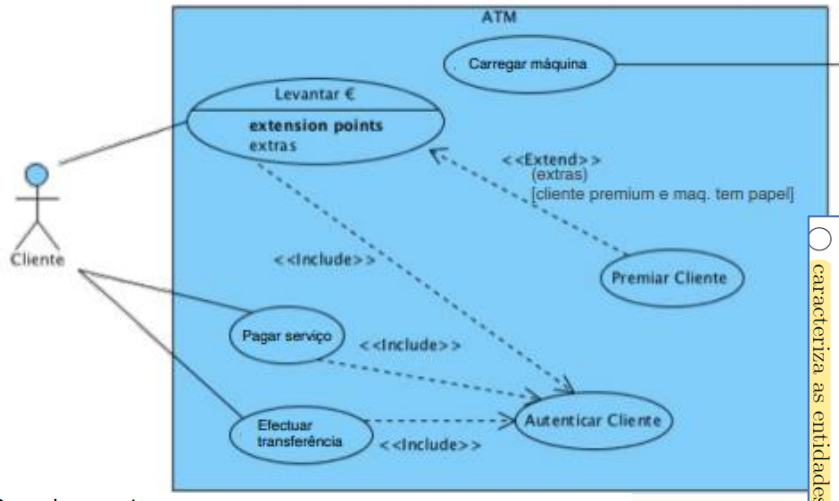
- Podemos caracterizá-los em três tipos:
  - Fluxo Normal (ou Principal)
    O fluxo mais comum. Representa uma situação perfeita em que nada corre mal.
    A pós-condição é satisfeita no final (se pré-condição também o é no início).
  - Fluxos Alternativos
    Fluxos válidos mas menos comuns.
    A pós-condição é satisfeita (se pré-condição também o é no início)
  - Fluxos de Excepção
    Condições de erro suficientemente importantes para serem capturadas no modelo.
    A pós-condição NÃO é satisfeita.

Definição de *Use Case*

- Descreve como os Actores atingem objectivos (realizam os *Use Cases*) utilizando o sistema
  - Definem relação entre *inputs* dos Actores e comportamento do Sistema
- Especificação deve incluir o comportamento tipicamente esperado, bem como variantes:
  - Comportamentos alternativos que ainda levam ao sucesso
  - Comportamentos de insucesso (Excepções)
- Vamos também definir as pré-condições e pós-condições de cada use case (cf. *design by contract*)

Lógica de negócio na Base de dados?

- Um dos princípios base das arquitecturas em camadas é a existência de uma camada de lógica de negócio
- Vantagens de manter a lógica de negócio separada da Base de Dados
  - poder expressivo das linguagens OO
  - facilidade de compreensão (debug, manutenção, ...)
  - escalabilidade
- Justificável colocar lógica na Base de dados
  - Operações *data intensive*
  - Lógica dos Dados
  - Segurança
  - (mas muito disto pode ser implementado numa camada de acesso... ;)



- Operadores mais comuns
  - **alt** - define fragmentos alternativos (mutuamente exclusivos)
  - **loop** / **loop(*n*)** - fragmento é repetido enquanto a guarda for verdadeira / ***n*** vezes
  - **opt** - fragmento opcional (ocorre se a guarda for verdadeira)
  - **par** - fragmentos ocorrem em paralelo
  - **break** - termina o fluxo
  - **ref** - referência a outro diagrama

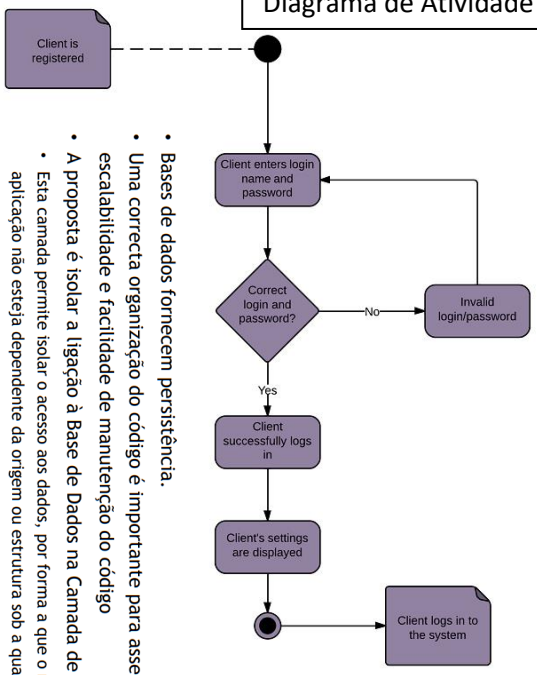
a expressão OCL:

```
context Jogador
inv: emprestados->size()>0 implies not amigos->isEmpty()
```

relativa ao diagrama de classes apresentado na página 7 do teste, expressa o invariante:
- ○ quem tem jogos emprestados, pode ter amigos.
- ○ só quem tem amigos pode ter jogos emprestados.
- ○ quem tem jogos emprestados, não pode ter amigos.

Um Modelo de Domínio...
- ○ é um tipo de diagrama UML.
- ○ pode ser descrito com um Diagrama de Classes.
- ○ caracteriza as entidades de um dado problema.

Diagrama de Atividade



- Bases de dados fornecem persistência.
- Uma correcta organização do código é importante para assegurar e escalabilidade e facilidade de manutenção do código
- A proposta é isolar a ligação à Base de Dados na Camada de Dados.
- Esta camada permite isolar o acesso aos dados, por forma a que o resto da aplicação não esteja dependente da origem ou estrutura sob a qual os dados estão armazenados.

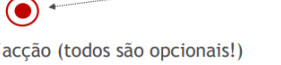
Diagramas de Estado

Notação base

- Estado — define uma possível estado do objecto (normalmente traduz-se em valores específicos dos seus atributos)

- Estado inicial — estado do objecto quando é criado

- Estado final — destruição do objecto



- Transições — evento[guarda]/acção (todos são opcionais!)

Os DAOs...
- ○ encapsulam o acesso à Base de Dados.
- ○ implementam persistência de dados.
- ○ criam o problema do ORM.

Introdução aos Diagramas de Estado — Aplicação

- Os Diagramas de Estado permitem modelar o comportamento de um dado objecto/sistema de forma global.
- A ênfase é colocada no estado do objecto/sistema — modelam-se todos os estados possíveis que o objecto/sistema atravessa em resposta aos eventos que podem ocorrer.
- Úteis para modelar:
  - O comportamento de um objecto de forma transversal aos use case do Sistema
  - O Sistema como um todo
- Devem utilizar-se para entidades/classes em que se torne necessário compreender o comportamento do objecto de forma global ao sistema.
  - Nem todas as entidades/classes vão necessitar de diagramas de estado.