UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Fundamentos de Programação

ET43I

Orientada a Objetos

Tratamento de exceções e UML

Aula 9

Prof. Daniel Cavalcanti Jeronymo

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia Eletrônica – 3º Período
2016.1

Plano de Aula

- Tratamento de Exceções

- Diagramas UML

Exceção

- Situações **excepcionais** que ocorrem na execução do código

- Realizam transferência de controle entre:
  - uma parte do código - com erro
  - outra parte do código - com tratamento do erro

- Podem ser implementadas em:
  - nível de Sistema (hardware ou S.O.)
  - nível de linguagem de programação

Exceção

- A nível de sistema

- Provocando uma divisão por zero
  - No Windows – Erro 0xC0000094
  - No Linux – SIGFPE

```
#include <iostream>

int main()
{
   std::cout << 1/0 << std::endl;
   return 0;
}
```

Exceção

- A nível de linguagem de programação
  - Em C++ – Usando blocks de try-catch com throw

```
try
{
   // codigo com possível lançamento (throw) de exceção aqui
}
catch(Tipo1 param)
{
   // tratamento 1
}
catch(Tipo2 param)
{
   // tratamento 2
}
catch (...)
{
   // tratamento padrão
}
```

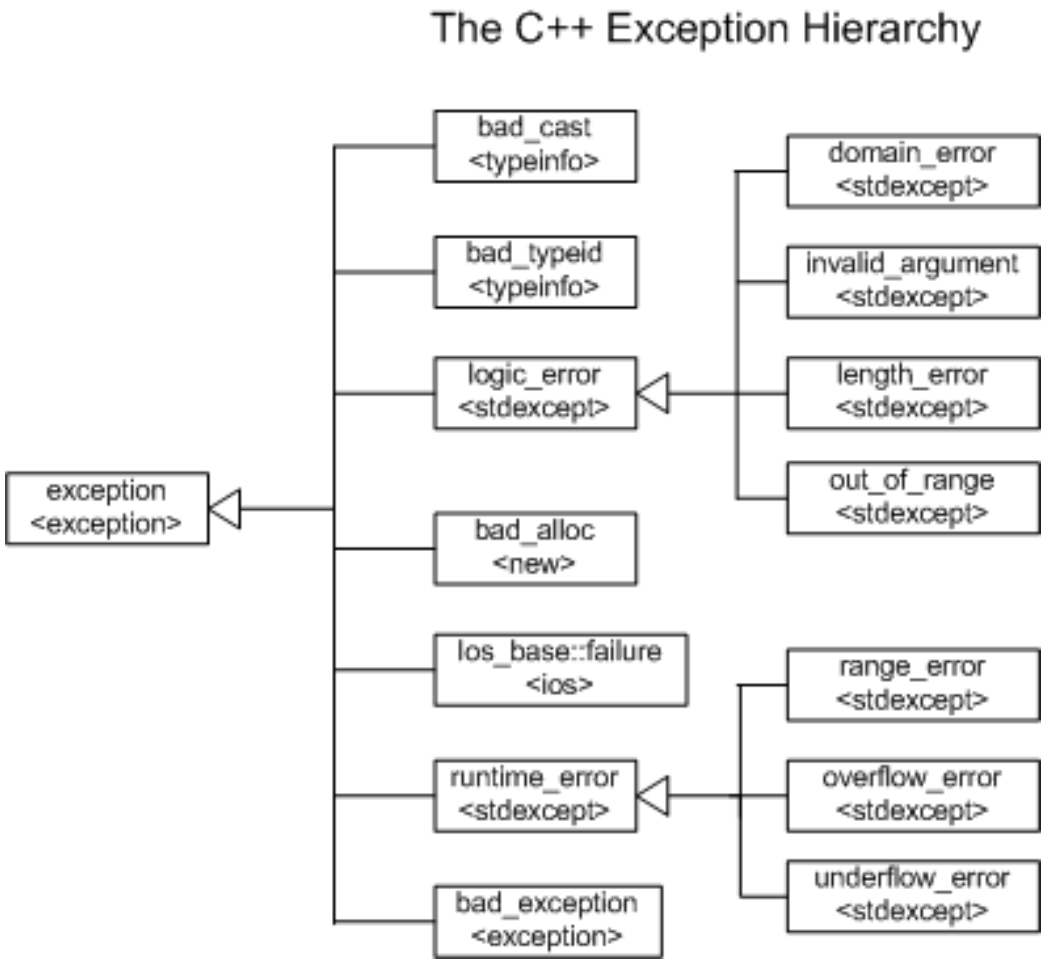
Exceção

- Exemplo

```
#include <iostream>

int main()
{
   try
   {
      std::cout << "Linha 1" << std::endl;
      throw 10;
      std::cout << "Linha 2" << std::endl;
   }
   catch(int e)
   {
      std::cout << "Linha 3" << std::endl;
      std::cout << "Excecao: " << e << std::endl;
      std::cout << "Linha 4" << std::endl;
   }

   return 0;
}
```

Exceção

- Exceções padrões da STL



Exceção

- Código seguro em relação a exceções (*exception safe code)*
  - *Exemplo vector*

std::**vector::at** <vector>

```
      reference at (size_type n);
const_reference at (size_type n) const;
```

Access element

Returns a reference to the element at position *n* in the vector.

The function automatically checks whether *n* is within the bounds of valid elements in the vector, throwing an out_of_range exception if it is not (i.e., if *n* is greater or equal than its `size`). This is in contrast with member operator[], that does not check against bounds.

Exceção

- Código seguro em relação a exceções (*exception safe code)*
  - *Exemplo vector*

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
   vector<int> a;

   // insere um elemento em 'a'
   a.push_back(10);

 // le a posição do 3o elemento de 'a' - não lança exceção mas pode ocorrer erro de acesso inválido em tempo de execução
   cout << a[2] << endl;

   return 0;
}
```

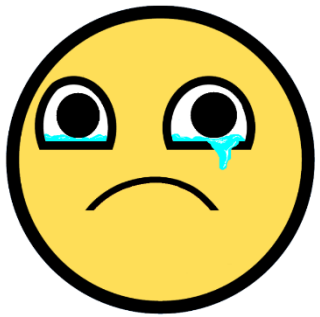
Exceção

- Código seguro em relação a exceções (*exception safe code)*
  - *Exemplo vector*

```
#include <iostream>
#include <vector>
#include <stdexcept>

using namespace std;

int main()
{
   vector<int> a;

   // insere um elemento em 'a'
   a.push_back(10);

   // le a posição do 3o elemento de 'a' - lança exceção
   try
   {
      cout << a.at(2) << endl;
   }
   catch(const out_of_range &a)
   {
      cout << "Catch (what): " << a.what() << endl;
   }

   return 0;
}
```

Exceção

- Código seguro em relação a exceções (*exception safe code)*
  - *Exemplo vector*
  - Ainda não é seguro *(push_back)*



```
#include <iostream>
#include <vector>
#include <stdexcept>

using namespace std;

int main()
{
   vector<int> a;

   // insere um elemento em 'a'
   a.push_back(10);

   // le a posição do 3o elemento de 'a' - lança exceção
   try
   {
      cout << a.at(2) << endl;
   }
   catch(const out_of_range &a)
   {
      cout << "Catch (what): " << a.what() << endl;
   }

   return 0;
}
```

Exceção

- Código seguro em relação a exceções (*exception safe code)*
  - *Exemplo vector*

```
int main()
{
   vector<int> a;

   try
   {
      // insere um elemento em 'a'
      a.push_back(2);

      // le a posição do 3o elemento de 'a'
      cout << a.at(2) << endl;
   }
   catch(const bad_alloc &a)
   {
      cout << "Catch (what bad_alloc): " << a.what() << endl;
   }
   catch(const out_of_range &a)
   {
      cout << "Catch (what out_of_range): " << a.what() << endl;
   }

   return 0;
}
```

Exceção

- Vantagens:
  - V1 – Torna o código mais legível e robusto
  - V2 – Precisa ser tratada ou gera erro de execução (ao contrário de códigos de erro)
  - V3 – O objeto da exceção pode armazenar informações sobre o erro
  - **V4 – É a única maneira de registrar erro num construtor seguindo RAII**
- Desvantagens:
  - D1 – Torna o código difícil de seguir e inspecionar
  - D2 – Facilita o vazamento de recursos
  - D3 – Escrever código seguro a exceções é difícil
  - D3.1 – O contrário é verdade, escrever código abusando de exceções é fácil
  - D4 – O tratamento de exceções tem um custo computacional

- D3.1 – O contrário é verdade, escrever código abusando de exceções é fácil
  - Considere o problema inicial da divisão de inteiros, como isso pode ser resolvido por exceções?
  - Escrever uma função para realizer a divisão e verificar se é segura

- D3.1 – O contrário é verdade, escrever código abusando de exceções é fácil

  - Esta abordagem não poderia ser trocada por uma sem exceções?

```
#include <iostream>
#include <stdexcept>

using namespace std;

int divisao_segura(int num, int den)
{
    if(den == 0)
        throw overflow_error("Excecao de
divisao por zero");

    return num / den;
}
```

```
int main()
{
    try
    {
        int ret = divisao_segura(10, 0);

        cout << "Divisao: " << ret << endl;
    }
    catch(const overflow_error &e)
    {
        cout << "Catch: " << e.what() <<
endl;
    }

    return 0;
}
```

Exceção

- D4 – O tratamento de exceções tem um custo computacional
  - Considere o problema inicial da divisão de inteiros, como isso pode ser resolvido na **linguagem C *de maneira similar* ao tratamento de exceção**?

Este é apenas um exemplo! Evite usar setjmp / longjmp.

```c
#include <stdio.h>
#include <setjmp.h>

int divisao_segura(int num, int den, jmp_buf buf)
{
    // verifica se a divisao é segura
    // caso não seja, volta
    if(den == 0)
        longjmp(buf, 1);

    // executa a divisao apenas se (den != 0)
    return num / den;
}
```

```c
int main()
{
    jmp_buf buf;
    int num = 10, den = 0, ret = 0;

    // setjmp - define a posição pro longjmp e armazena em 'buf'
    if(setjmp(buf) == 0) // retorna zero em fluxo normal de código, diferente de zero quando vem do longjmp
    {
        // tenta realizar a divisão segura
        ret = divisao_segura(num, den, buf);
        printf("Valor da divisao: %d\n", ret);
    }
    else
    {
        // caso a divisao não seja segura, chega nesse ponto
        printf("Catch: a divisao nao foi segura!\n");
    }

    return 0;
}
```

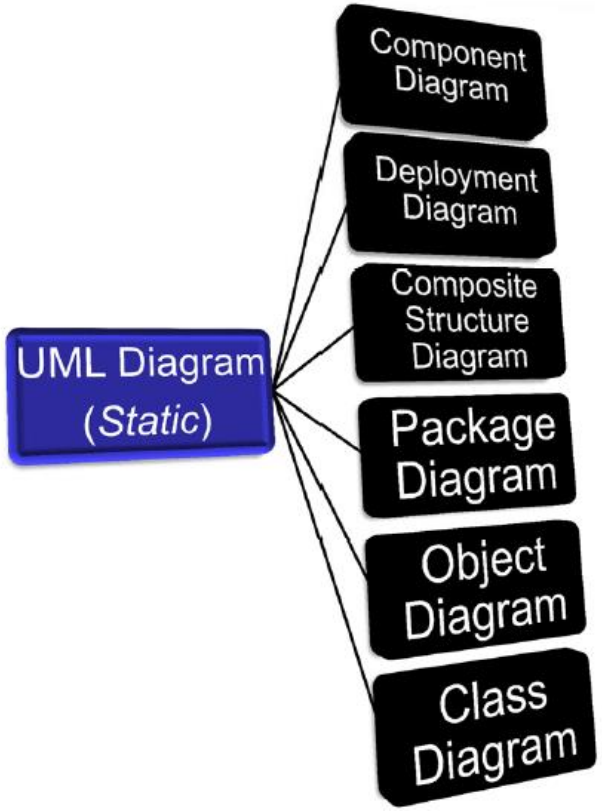
- **V4 – É a única maneira de registrar erro num construtor seguindo RAII**
  - Verdadeira motivação para a existência de exceções

```
class LePalavra
{
public:
    string palavra;

    LePalavra(const string &nome)
    {
        ifstream arquivo(nome.c_str());

        arquivo >> palavra;

        // como reportar erro caso o arquivo não seja aberto???
    }
};
```

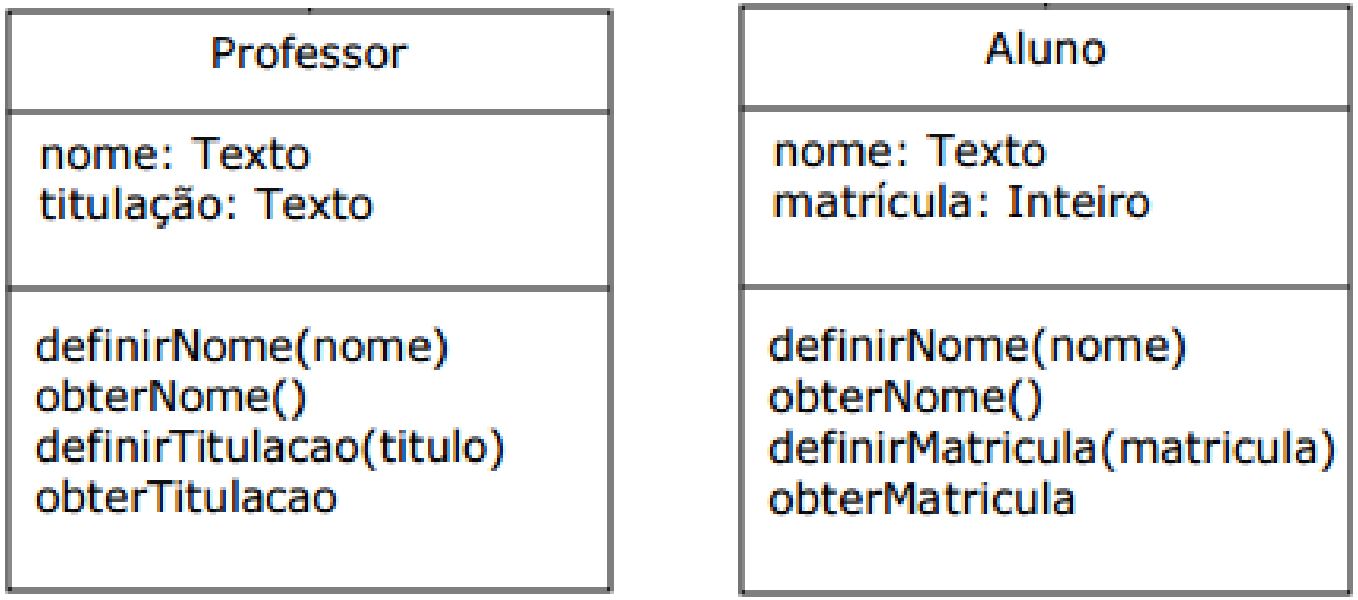
- Leituras adicionais:
  - https://isocpp.org/wiki/faq/exceptions#why-not-exceptions
  - https://isocpp.org/blog/2012/12/systematic-error-handling-in-c-andrei-alexandrescu
  - Cap. 5.4 (Exception Handling) do TR 18015/2006

http://www.open-std.org/jtc1/sc22/wg21/docs/TR18015.pdf

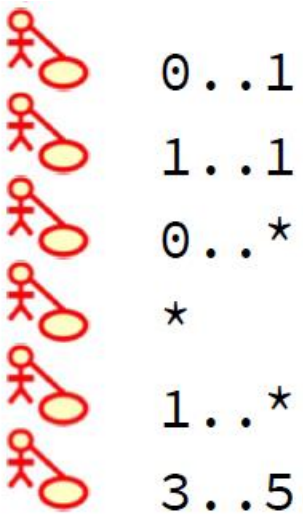
- UML – Unified Modeling Language
  - Linguagem de modelagem utilizada na engenharia de software para visualizar o projeto de um sistema

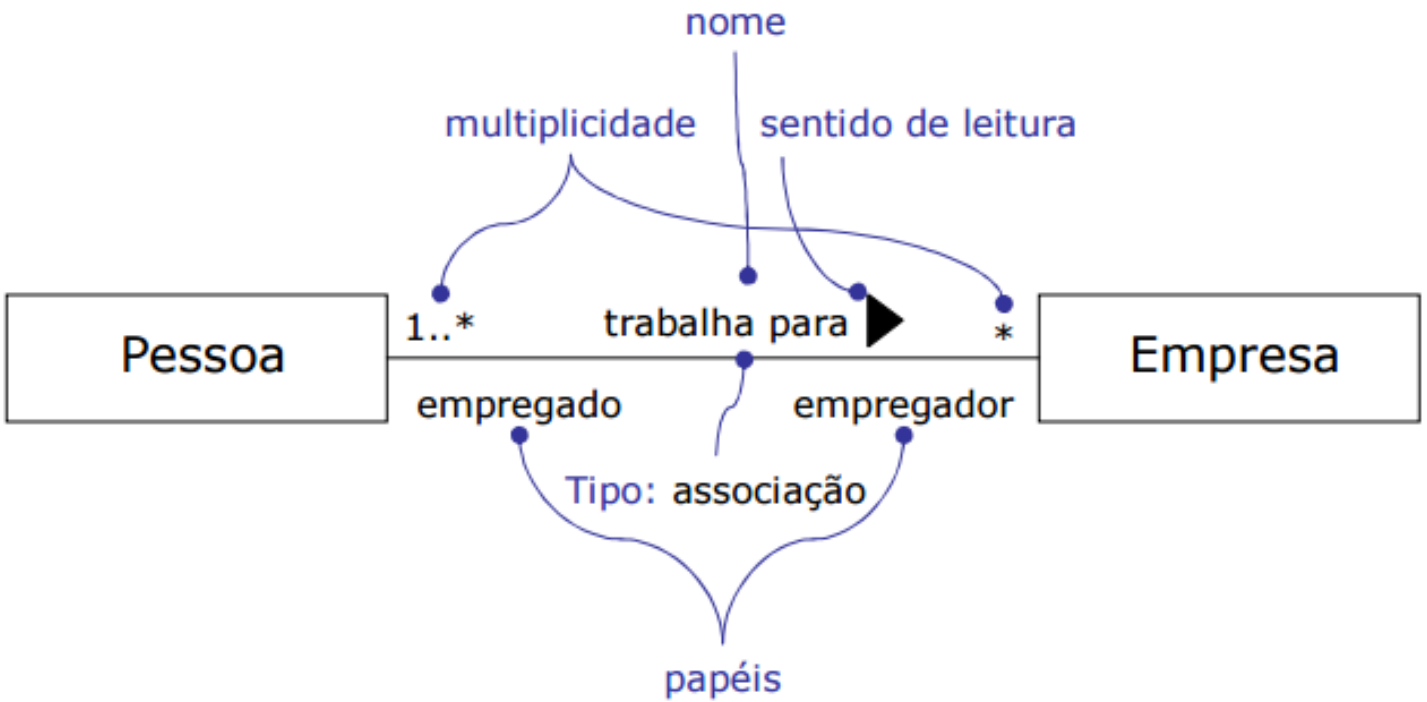


- Diagrama de classes, especifica:
  - Nome da classe
  - Atributo
  - Métodos

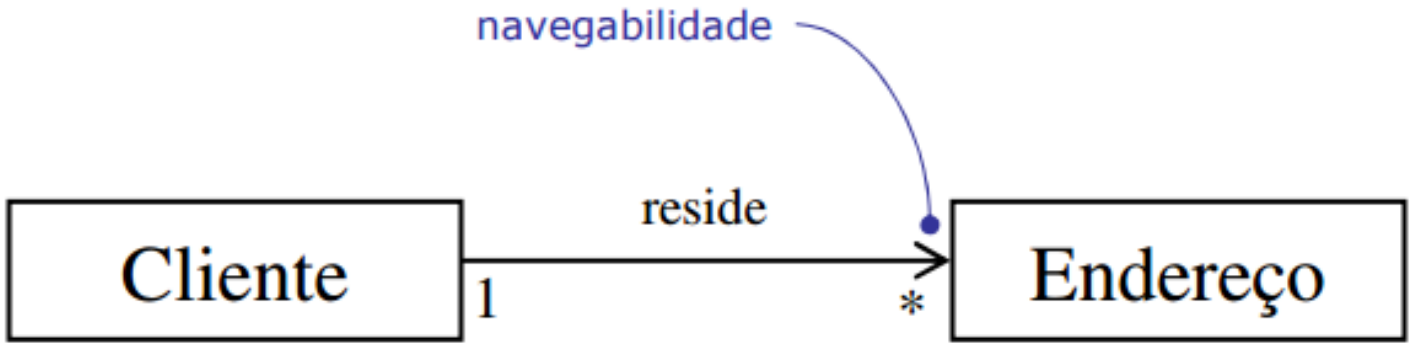


- Relacionamentos
  - Grau de multiplicidade

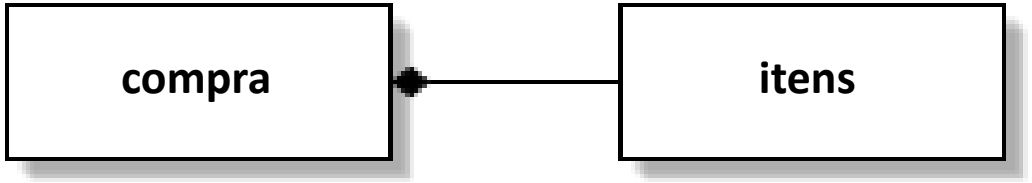




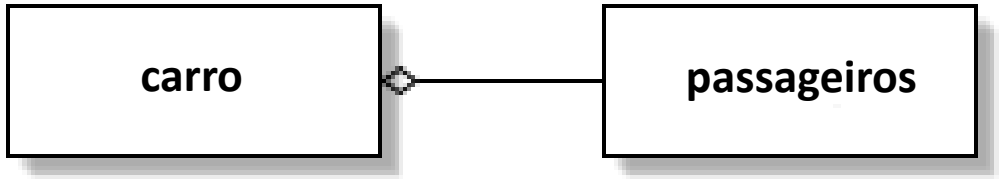
- Relacionamentos
  - O cliente sabe quais são os seus endereços mas o endereço não sabe a quais clientes pertence



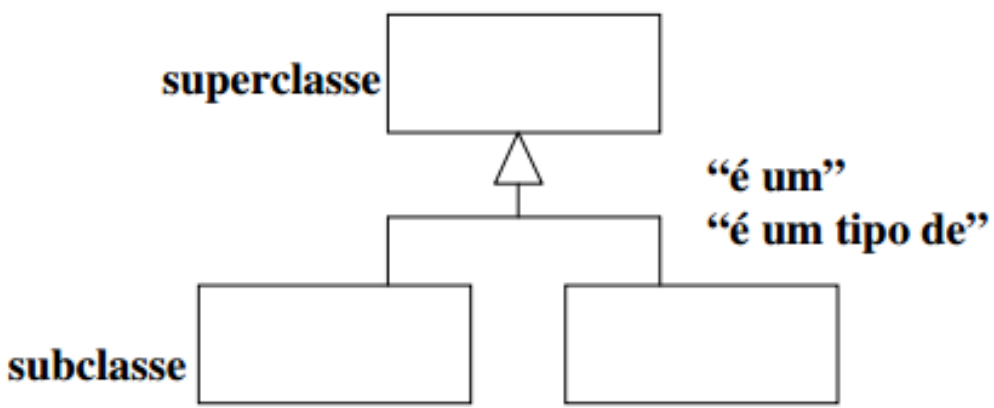
- Relacionamentos
  - Composição – a parte não existe sem o todo (toda compra tem itens comprados)

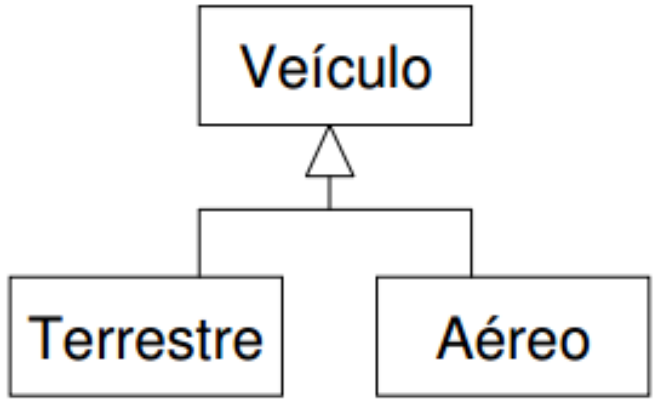


  - Agregação – a parte existe sem o todo (um carro pode ou não ter passageiros)

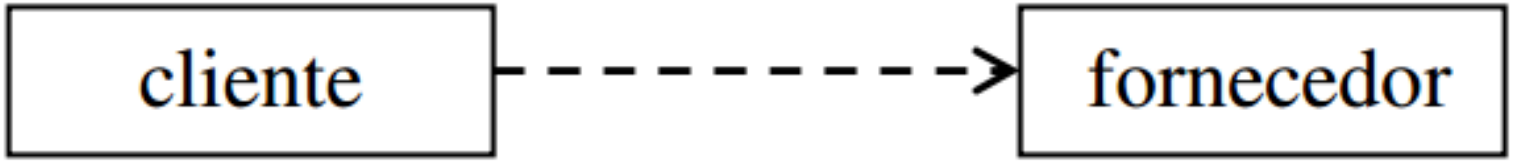


- Relacionamentos
  - Generalização – relacionamento entre generalizações (superclasses) e especializações (subclasses)

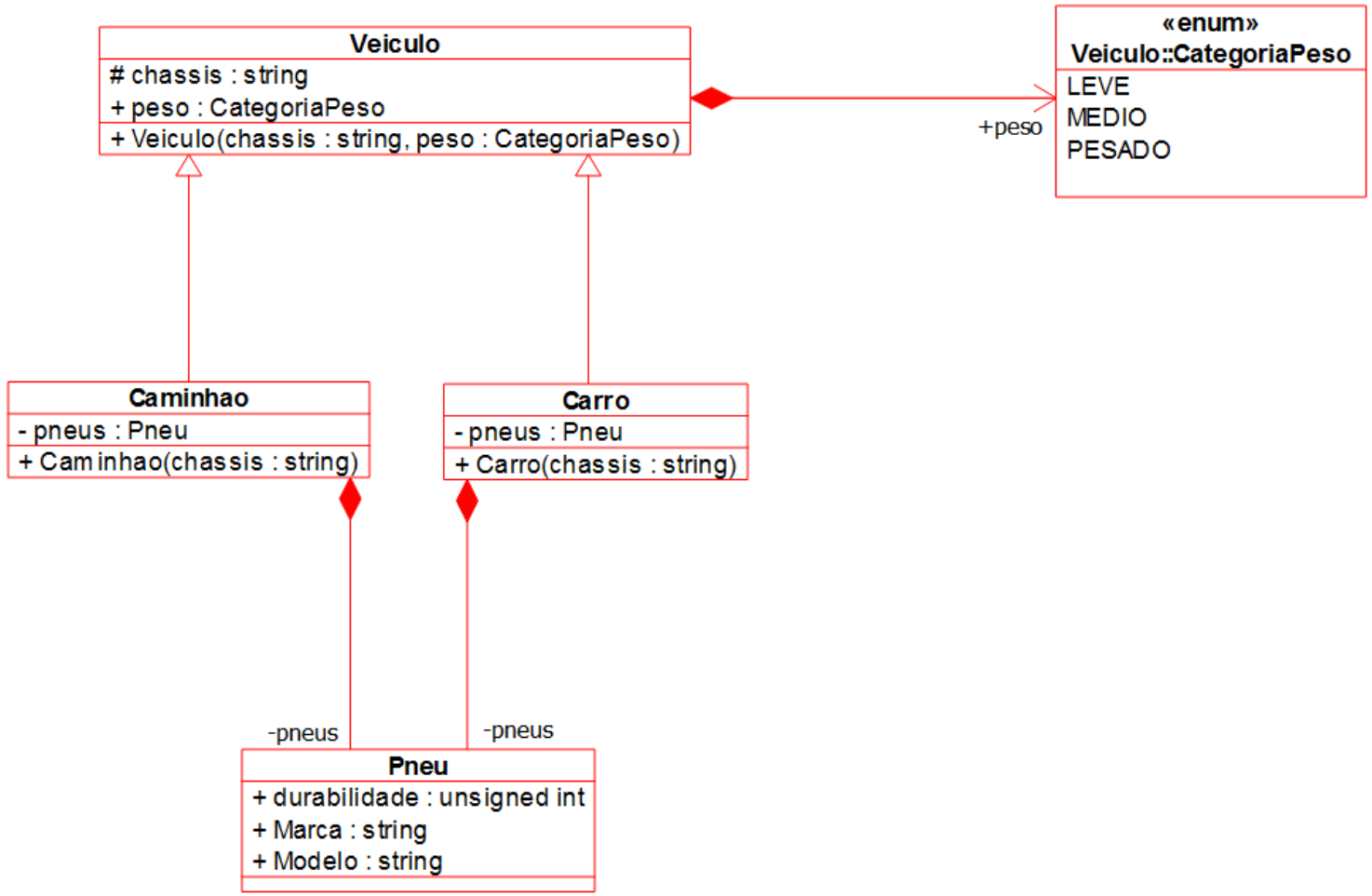




- Relacionamentos
  - Dependência – a alteração de um objeto (independente) pode afetar outro objeto (dependente)



- Exemplo de veículos (**umlExemploVeiculo.rar**)



UML

- Tarefa para quarta-feira (Lista 8):
  - Prepare um esboço de diagrama UML da sua APS
  - Procure apontar os itens avaliados na APS pelo diagrama