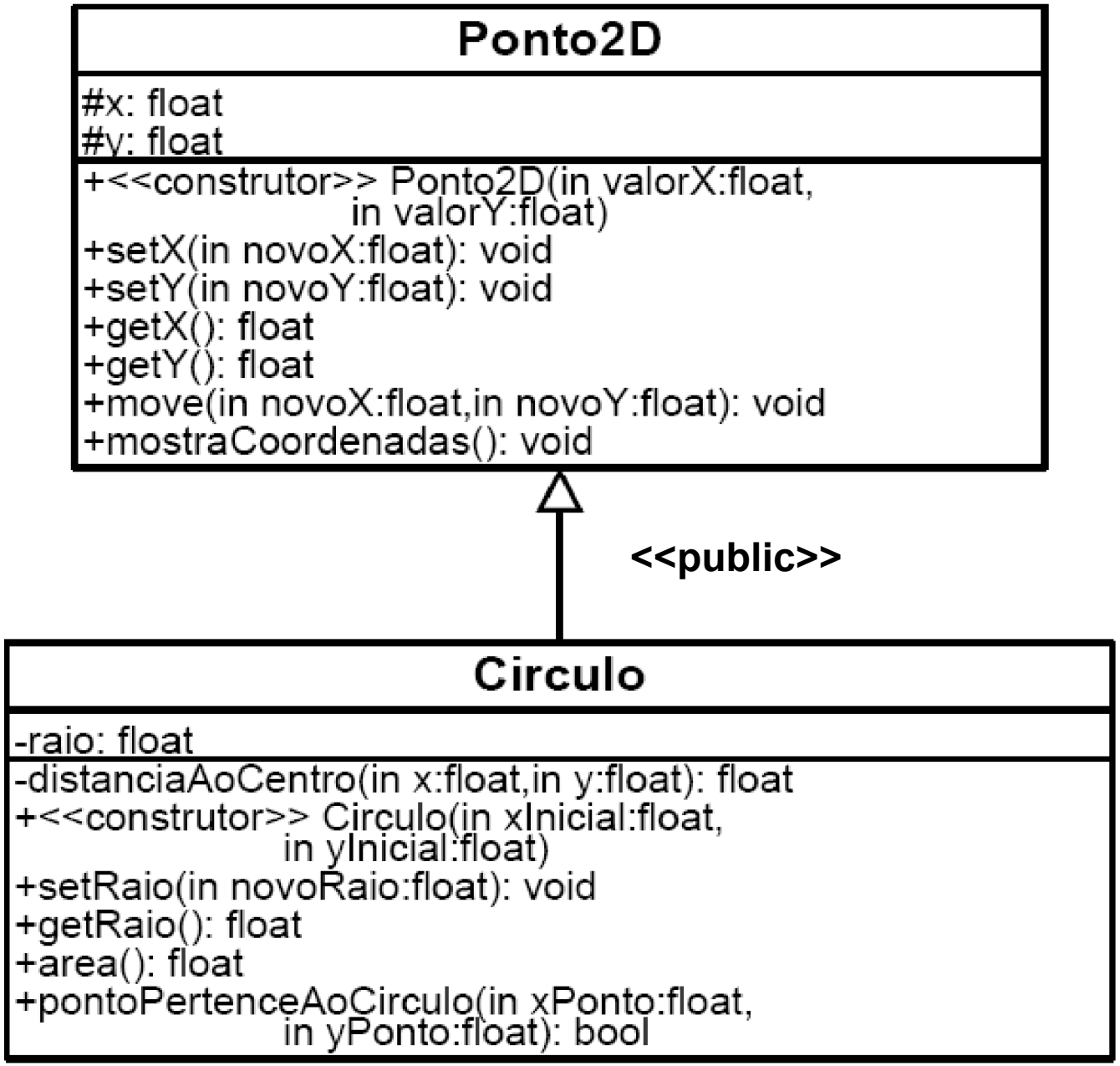
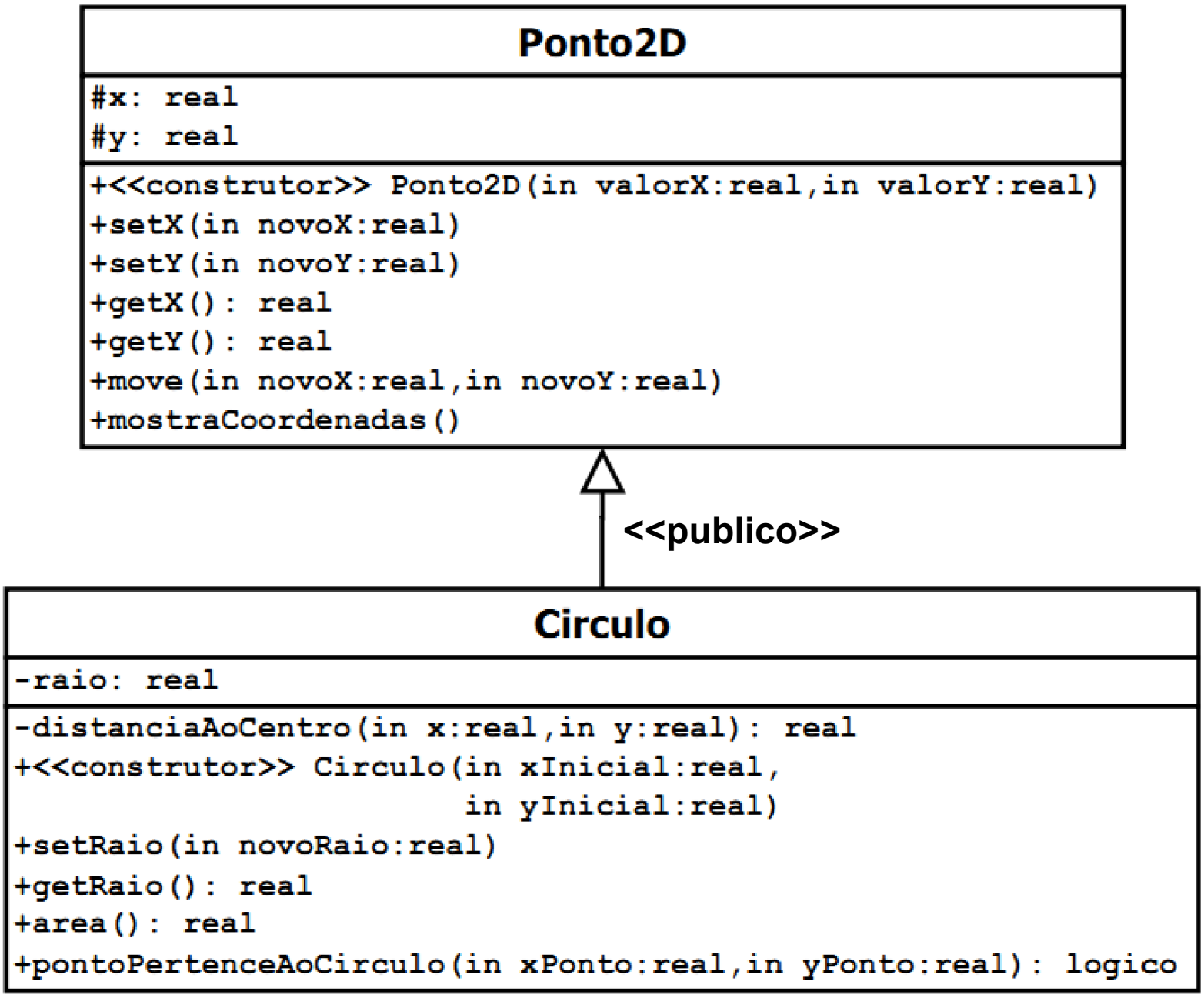
Linguagem de Programação C++



Linguagem de Programação C++



Linguagem de Programação C++

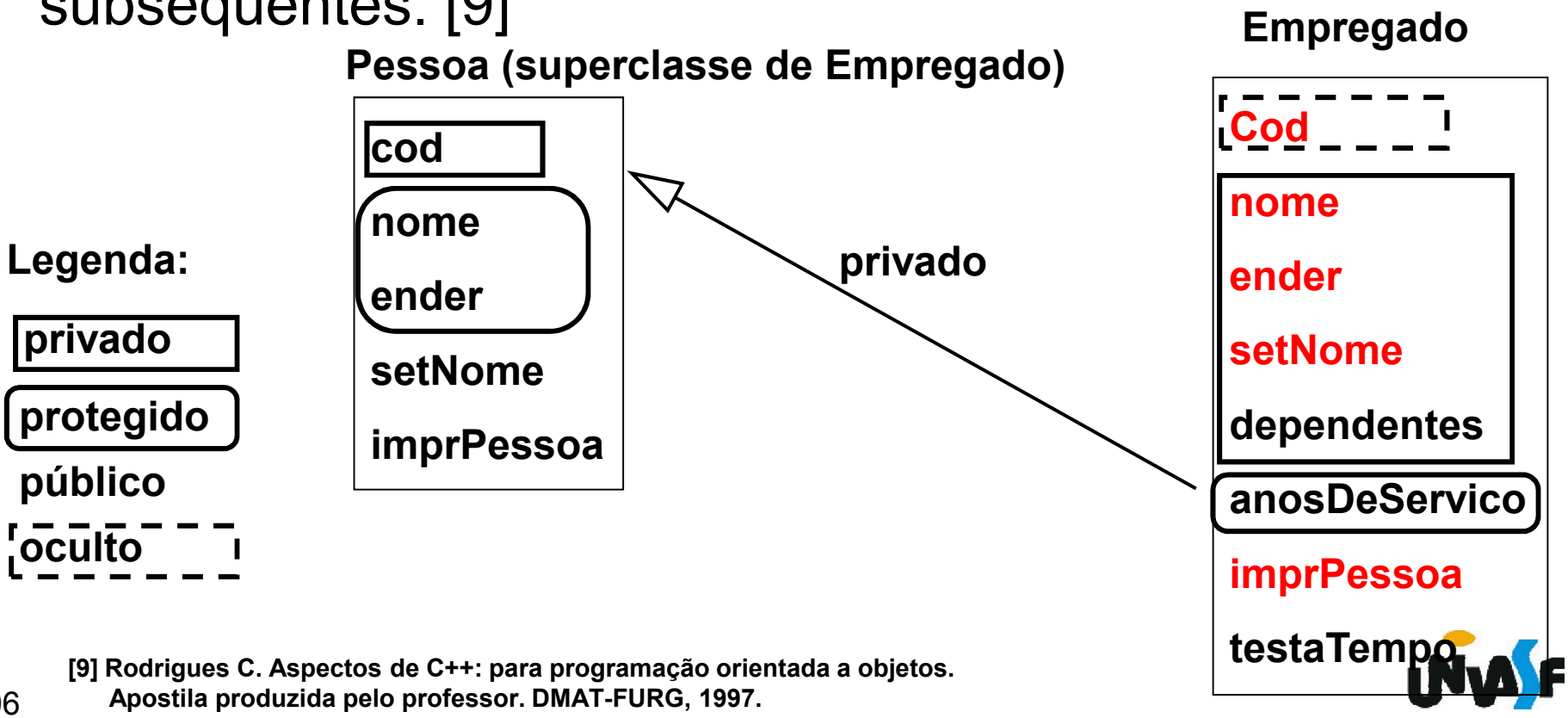
É possível efetuar uma redeclaração individual do modo de acesso, mudando explicitamente o status de um membro de classe derivada.

Vamos analisar o exemplo:

```
class Pessoa
{
  long int cod; //privado – obs. Evitar utilizar a especificação implícita
  protected:
    char nome[30];
    char ender[50];
  public:
    void setNome(novoNome[30]);
    void imprPessoa ();
};
class Empregado:Pessoa //recepção default: private – obs. anterior
{
  public:
    void Pessoa::imprPessoa (); //este método seria privado por default
    int testaTempo(void);
  private:
    int dependentes;
  protected:
    int anosDeServico;
};
```

Linguagem de Programação C++

Assim, o campo nome, de objetos da classe Empregado, será visível fora da classe. Porém, continuara privado nas classes derivadas de empregado. Ou seja, a especificação explícita individual do modo de recepção de um campo privativo **não** se transmite às derivações subseqüentes. [9]



[9] Rodrigues C. Aspectos de C++: para programação orientada a objetos. Apostila produzida pelo professor. DMAT-FURG, 1997.

Linguagem de Programação C++

Em nosso estudo dos conceitos e princípios da OO vimos que uma análise do mundo real nos remete a perceber que em alguns sistemas temos a necessidade de uma subclasse com mais de uma superclasse.

A linguagem C++ possibilita a definição de uma subclasse que possui mais de uma superclasse, em outras palavras, possibilita a **herança múltipla**.

Vamos imaginar o seguinte exemplo: temos duas classes Embalagem e Rotulo, ambas com membros específicos. Pretende-se obter uma classe EmbalagemRotulada, que associa ao recipiente uma descrição.

A sintaxe é:

class EmbalagemRotulada: public Embalagem, public Rotulo

Os objetos desta classe podem receber mensagens de Embalagem e Rotulo.

UNIVASF

Devemos ter ciência de que **problemas** ocorrerão se mais de uma superclasse oferecer funções membros de mesmo nome. Neste caso, será necessário redeclará-las.

Por exemplo, imagine que ambas as superclasses apresentam uma função membro denominada apagar e sem parâmetros, este problema pode ser resolvido da seguinte forma (obs. Este problema se mantém mesmo que os métodos tenham conjuntos de parâmetros distintos):

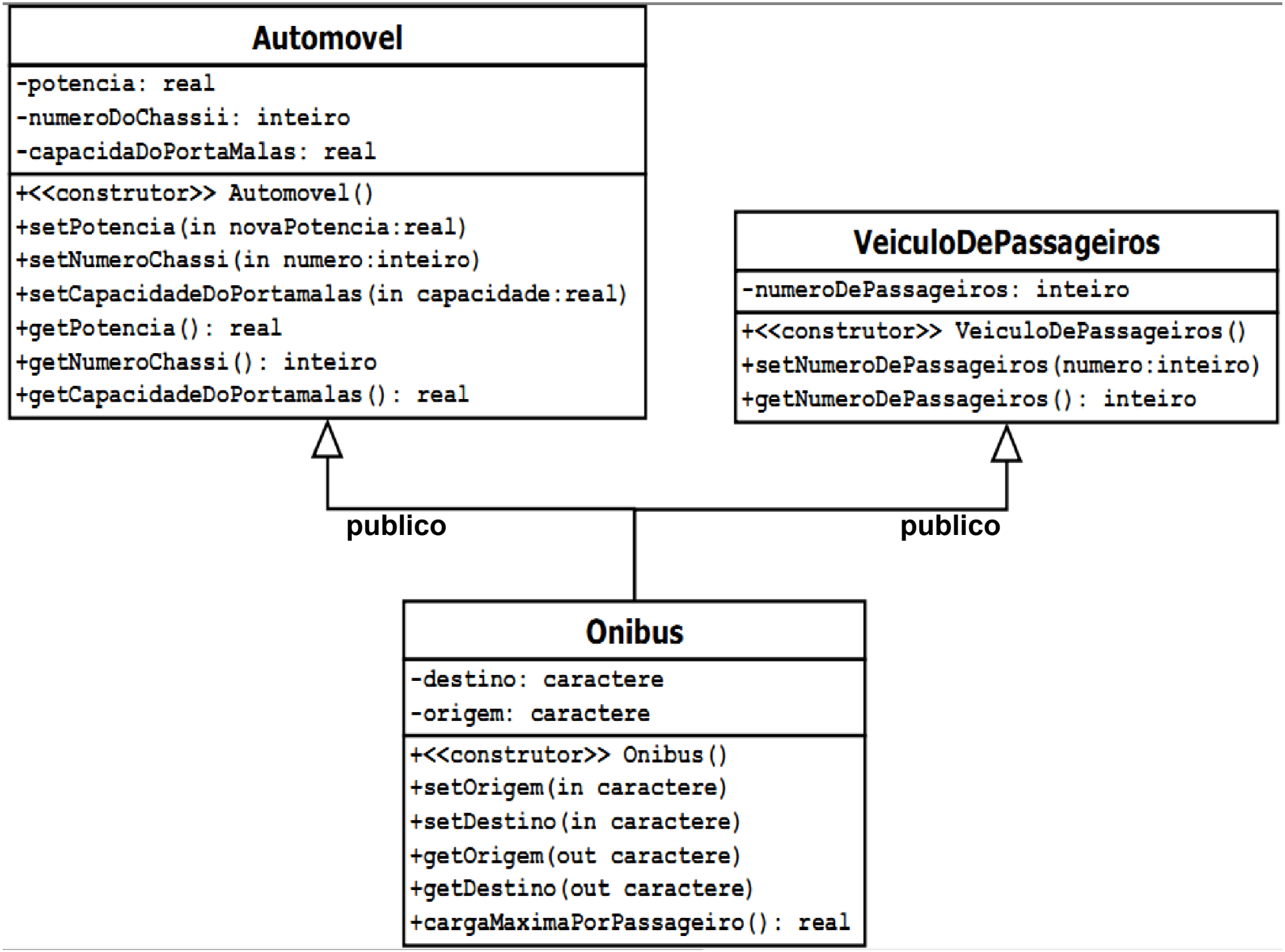
```
class EmbalegemRotulada: public Embalagem, public Rotulo
{
  public:
    void apagar(void);
}
...
void EmbalegemRotulada::apagar(void)
{
  Embalagem::apagar();
  Rotulo::apagar();
}
```

Exercício:

Na oportunidade, em que vimos o conceito de herança múltipla, trabalhamos o exemplo:

De um determinado sistema no qual constavam as classes Automovel e VeiculoDePassageiros, surgindo então a necessidade de definir a classe Onibus.

Sabendo-se que um objeto da classe Onibus é uma instância de Automovel e também é uma instância de VeiculoDePassageiros. Construa um diagrama de classes UML para representar as classes retro aludidas.


Automovel
-potencia: real
-numeroDoChassii: inteiro
-capacidaDoPortaMalas: real
+<<construtor>> Automovel()
+setPotencia(in novaPotencia:real)
+setNumeroChassi(in numero:inteiro)
+setCapacidadeDoPortamalas(in capacidade:real)
+getPotencia(): real
+getNumeroChassi(): inteiro
+getCapacidadeDoPortamalas(): real
VeiculoDePassageiros
-numeroDePassageiros: inteiro
+<<construtor>> VeiculoDePassageiros()
+setNumeroDePassageiros(numero:inteiro)
+getNumeroDePassageiros(): inteiro
publico
publico
Onibus
-destino: caractere
-origem: caractere
+<<construtor>> Onibus()
+setOrigem(in caractere)
+setDestino(in caractere)
+getOrigem(out caractere)
+getDestino(out caractere)
+cargaMaximaPorPassageiro(): real

Linguagem de Programação C++

Com base neste diagrama implemente, em C++, as classes em questão e construa um programa que se utilize da classe Onibus explorando sua interface.

```
//conteúdo do arquivo Automovel.h
#ifndef AUTOMOVEL_H
  #define AUTOMOVEL_H
  class Automovel
  {
    public:
      Automovel();
      void setPotencia(float);
      void setNumeroChassi(int);
      void setCapacidadeDoPortaMalas(float);
      float getPotencia();
      int getNumeroChassi();
      float getCapacidadeDoPortaMalas();
    private:
      float potencia;
      int numeroDoChassi;
      float capacidadeDoPortaMalas;
  };
#endif
```

```
//conteúdo do arquivo Automovel.cpp
#include "Automovel.h"
Automovel::Automovel()
{
  setPotencia(0);
  setNumeroChassi(0);
  setCapacidadeDoPortaMalas(0);
}
void Automovel::setPotencia(float novaPotencia)
{
  if (novaPotencia>0)
    potencia=novaPotencia;
  else
    potencia=0;
}
void Automovel::setNumeroChassi(int numero)
{
  if (numero>0)
    numeroDoChassi=numero;
  else
    numeroDoChassi=0;
}
```

```
void Automovel::setCapacidadeDoPortaMalas(float capacidade)
{
  if (capacidade>0)
    capacidadeDoPortaMalas=capacidade;
  else
    capacidadeDoPortaMalas=0;
}
float Automovel::getPotencia()
{
  return potencia;
}
int Automovel::getNumeroChassi()
{
  return numeroDoChassi;
}
float Automovel::getCapacidadeDoPortaMalas()
{
  return capacidadeDoPortaMalas;
}
```

```
//conteúdo do arquivo VeiculoDePassageiros.h
class VeiculoDePassageiros
{
  public:
    VeiculoDePassageiros();
    void setNumeroDePassageiros(int);
    int getNumeroDePassageiros();
  private:
    int numeroDePassageiros;
};
```

```
//conteúdo do arquivo VeiculoDePassageiros.cpp
#include "VeiculoDePassageiros.h"
VeiculoDePassageiros::VeiculoDePassageiros()
{
  setNumeroDePassageiros(0);
}
void VeiculoDePassageiros::setNumeroDePassageiros(int numero)
{
  if (numero>0)
    numeroDePassageiros=numero;
  else
    numeroDePassageiros=0;
}
int VeiculoDePassageiros::getNumeroDePassageiros()
{
  return numeroDePassageiros;
}
```

```
//conteúdo do arquivo Onibus.h
#ifndef ONIBUS_H
  #define ONIBUS_H
  #include "Automovel.h"
  #include "VeiculoDePassageiros.h"
  class Onibus: public Automovel, public VeiculoDePassageiros
  {
    public:
      Onibus();
      void setOrigem(char [30]);
      void setDestino(char [30]);
      void getOrigem(char [30]);
      void getDestino(char [30]);
      float cargaMaximaPorPassageiro();
    private:
      char destino[30];
      char origem[30];
  };
#endif
```

```
//conteúdo do arquivo Onibus.cpp
#include "Onibus.h"
#include <cstring>
Onibus::Onibus()
{
  setOrigem("");
  setDestino("");
}
void Onibus::setOrigem(char novaOrigem[30])
{
  strcpy(origem, novaOrigem);
}
void Onibus::setDestino(char novoDestino[30])
{
  strcpy(destino, novoDestino);
}
void Onibus::getOrigem(char orig[30])
{
  strcpy(orig, origem);
}
```

```
void Onibus::getDestino(char dest[30])
{
  strcpy(dest, destino);
}
float Onibus::cargaMaximaPorPassageiro()
{
  if (getNumeroDePassageiros())
    return (getCapacidadeDoPortaMalas()/getNumeroDePassageiros());
  else
    return 0;
}
```

```
//conteúdo do arquivo principalOnibus.cpp
#include "Onibus.h"
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
int main(){
   char str1[30], str2[30];
   Onibus onibus;
   onibus.setOrigem("Juazeiro");
   onibus.setDestino("Salvador");
   onibus.setCapacidadeDoPortaMalas(2500);
   onibus.setNumeroDePassageiros(55);
   cout << endl;
   cout << endl << "O onibus de ";
   onibus.getOrigem(str1);
   cout << str1 << " para ";
   onibus.getDestino(str2);
   cout << str2 << " possibilita que cada passageiro transporte ate ";
   cout << onibus.cargaMaximaPorPassageiro() << " kg." << endl;
   return 0;
}
```