Herança

Roland Teodorowitsch

Programação Orientada a Objetos - ECo - Curso de Engenharia de Computação - PUCRS

29 de maio de 2024

Herança

Herança

- Ao modelar um conjunto de classes, é possível encontrar classes semelhantes na estrutura e no comportamento
  - Solução 1: modelar as classes de forma independente (duplicação de código)
  - Solução 2: extrair características estruturais e comportamentais que sejam comuns e colocá-las em classes mais gerais a partir das quais são definidas classes mais específicas (**herança**)

Herança

- É a propriedade da programação orientada a objetos que permite expressar o compartilhamento de características comuns entre as classes
- Permite a reutilização de código através especialização de soluções genéricas já existentes

Sintaxe da Herança

- Em C++, pode-se criar uma classe **derivada** a partir de uma classe **base** usando:

```
class Base {
  // Membros
};

class Derivada : /* public, private, protected */ Base {
  // Outros membros
};
```

- O acesso aos membros da classe base será no máximo o definido na herança
- Se o modificador de acesso for omitido, será usado `private`

Aspectos da Herança

- Os seguintes aspectos devem ser considerados quando se usa herança:
  - Acesso a Atributos e Métodos Privados
  - Construtor
  - Sobrescrita de Métodos
  - Herança Múltipla

Acesso a Atributos e Métodos Privados

- Atributos e métodos privados na superclasse, usualmente, NÃO são acessíveis na subclasse. Assim, se:

```
class Aluno {
  private:
    int matricula;
    string nome;
  public:
    string obtemNome();
};
```

- E a herança for declarada usando `public`, deve-se usar a interface pública da classe base:

```
class AlunoRegular : public Aluno {
  ...
  string obtemNomeComPrefixo() {
    return "Aluno: " + obtemNome();
  }
};
```

Membros Protected

- A subclasse **herda** (tem acesso) apenas os métodos e atributos públicos da superclasse
- O acesso aos atributos privados deve ser feito através de métodos públicos
- É possível declarar atributos da superclasse como `protected`, o que faz com que eles sejam herdados pela subclasse

Construtor

- Uma classe derivada pode usar os construtores da classe base. O construtor padrão é usado, mas isso pode ser modificado

```
class Base {
  public:
    Base(int b);
};

class Derivada : /* public, private, protected */ Base {
  public:
    Derivada(int d);
};

Derivada::Derivada(int d) : Base(d) {
  // Implementacao do construtor da classe Derivada
}
```

- Note que o construtor de `Base` é chamado com parâmetro `d`, recebido na chamada do construtor de `Derivada`
- O construtor da classe base sempre será executado antes do construtor da derivada

Sobrescrita de Métodos

- Uma classe derivada pode redefinir métodos da classe base, simplesmente declarando um novo método com a mesma assinatura
- O método da classe base ainda estará disponível para ela através do operador : :
- Os métodos são ditos sobrescritos

```
class Base {
  public:
    Base(int b);
    void funcao();   // Metodo da base
};

class Derivada : /* public, private, protected */ Base {
  public:
    Derivada(int d);
    void funcao();   // Metodo sobrescrito
};

void Derivada::funcao() {
  Base::funcao();
}
```

Sobrescrita x Sobrecarga

- A sobrescrita ocorre quando temos métodos com o mesmo nome e assinatura (mesmas variáveis paramétricas)
- A sobrecarga ocorre quando temos métodos com o mesmo nome, porém assinaturas diferentes

Herança Múltipla

- A linguagem C++ admite herança múltipla, ou seja, classes derivadas podem herdar mais de uma classe base
- Se houver conflitos entre atributos e métodos das classes bases, o operador :: pode ser usado para resolvê-los

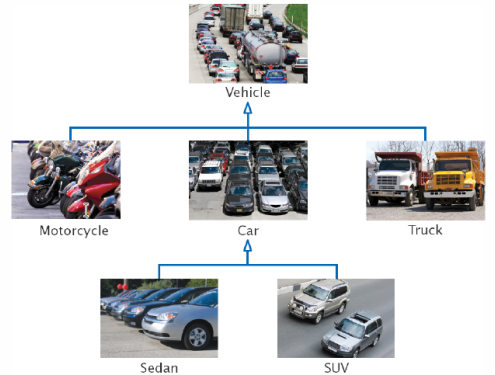
```
class Base {
  public:
    Base(int b);
};

classe OutraBase {
  public:
    OutraBase();
}

class Derivada : /* public, private, protected */ Base, OutraBase {
  public:
    Derivada(int d);
};

Derivada::Derivada(int d) : Base(d),OutraBase() {
  // Implementacao do construtor
}
```

Taxonomia de Classes



Fonte: Java For EveryOne (HORSTMANN, p. 456)

- Se X é subclasse de Y, então Y é superclasse de X e todo X é Y
- As classes que herdam características de outras são ditas classes filhas, classes derivadas ou subclasses
- As classes a partir das quais outras são derivadas são ditas classes pai ou superclasses
- Uma hierarquia de classes pode ter vários níveis
- Em uma mesma hierarquia uma classe pode ser considerada superclasse em relação a uma classe e subclasse em relação à outra

Exercício

Exercício

Identifique a **superclasse** e a **subclasse** em cada um dos seguintes pares de classes:

- Empregado x Gerente

Exercício

Identifique a **superclasse** e a **subclasse** em cada um dos seguintes pares de classes:

- Empregado x Gerente
  - Superclasse = Empregado / Subclasse = Gerente
- EstudanteDeGraduacao x Estudante

Exercício

Identifique a **superclasse** e a **subclasse** em cada um dos seguintes pares de classes:

- Empregado x Gerente
  - Superclasse = Empregado / Subclasse = Gerente
- EstudanteDeGraduacao x Estudante
  - Superclasse = Estudante / Subclasse = EstudanteDeGraduacao
- Pessoa x Estudante

Exercício

Identifique a **superclasse** e a **subclasse** em cada um dos seguintes pares de classes:

- Empregado x Gerente
  - Superclasse = Empregado / Subclasse = Gerente
- EstudanteDeGraduacao x Estudante
  - Superclasse = Estudante / Subclasse = EstudanteDeGraduacao
- Pessoa x Estudante
  - Superclasse = Pessoa / Subclasse = Estudante
- Empregado x Professor

Exercício

Identifique a **superclasse** e a **subclasse** em cada um dos seguintes pares de classes:

- Empregado x Gerente
  - Superclasse = Empregado / Subclasse = Gerente
- EstudanteDeGraduacao x Estudante
  - Superclasse = Estudante / Subclasse = EstudanteDeGraduacao
- Pessoa x Estudante
  - Superclasse = Pessoa / Subclasse = Estudante
- Empregado x Professor
  - Superclasse = Empregado / Subclasse = Professor
- ContaBancaria x ContaComChequeEspecial

Exercício

Identifique a **superclasse** e a **subclasse** em cada um dos seguintes pares de classes:

- Empregado x Gerente
  - Superclasse = Empregado / Subclasse = Gerente
- EstudanteDeGraduacao x Estudante
  - Superclasse = Estudante / Subclasse = EstudanteDeGraduacao
- Pessoa x Estudante
  - Superclasse = Pessoa / Subclasse = Estudante
- Empregado x Professor
  - Superclasse = Empregado / Subclasse = Professor
- ContaBancaria x ContaComChequeEspecial
  - Superclasse = ContaBancaria / Subclasse = ContaComChequeEspecial
- Carro x Veiculo

Exercício

Identifique a **superclasse** e a **subclasse** em cada um dos seguintes pares de classes:

- Empregado x Gerente
  - Superclasse = Empregado / Subclasse = Gerente
- EstudanteDeGraduacao x Estudante
  - Superclasse = Estudante / Subclasse = EstudanteDeGraduacao
- Pessoa x Estudante
  - Superclasse = Pessoa / Subclasse = Estudante
- Empregado x Professor
  - Superclasse = Empregado / Subclasse = Professor
- ContaBancaria x ContaComChequeEspecial
  - Superclasse = ContaBancaria / Subclasse = ContaComChequeEspecial
- Carro x Veiculo
  - Superclasse = Veiculo / Subclasse = Carro
- Veiculo x Caminhao

Exercício

Identifique a **superclasse** e a **subclasse** em cada um dos seguintes pares de classes:

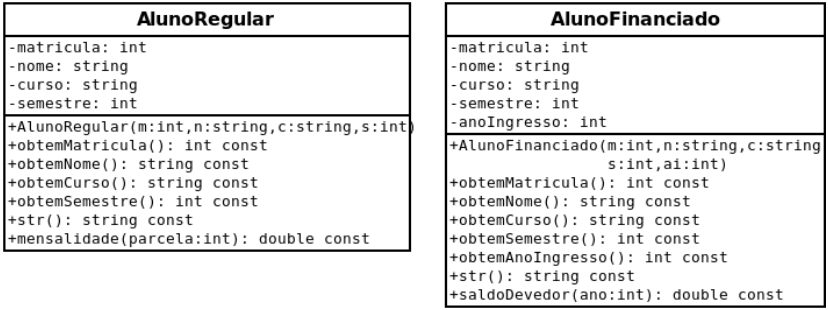
- Empregado x Gerente
  - Superclasse = Empregado / Subclasse = Gerente
- EstudanteDeGraduacao x Estudante
  - Superclasse = Estudante / Subclasse = EstudanteDeGraduacao
- Pessoa x Estudante
  - Superclasse = Pessoa / Subclasse = Estudante
- Empregado x Professor
  - Superclasse = Empregado / Subclasse = Professor
- ContaBancaria x ContaComChequeEspecial
  - Superclasse = ContaBancaria / Subclasse = ContaComChequeEspecial
- Carro x Veiculo
  - Superclasse = Veiculo / Subclasse = Carro
- Veiculo x Caminhao
  - Superclasse = Veiculo / Subclasse = Caminhao

Exemplo

Exemplo

- A Universidade XYZ trabalha com dois tipos de alunos: Regular e Financiado
- `AlunoRegular`
  - Custo da mensalidade é baseado no curso/semestre que está cursando, bem como no número da parcela.
  - São 6 parcelas por semestre, sendo que as 3 primeiras custam 20% a mais que as demais (fazendo o papel de matrícula).
- `AlunoFinanciado`
  - Não paga mensalidade durante o curso. Começará a pagar após o término do mesmo.
  - A qualquer momento pode consultar seu saldo devedor. Este é calculado tendo por base o curso/semestre em que se encontra e o ano de ingresso na Instituição.

Exemplo

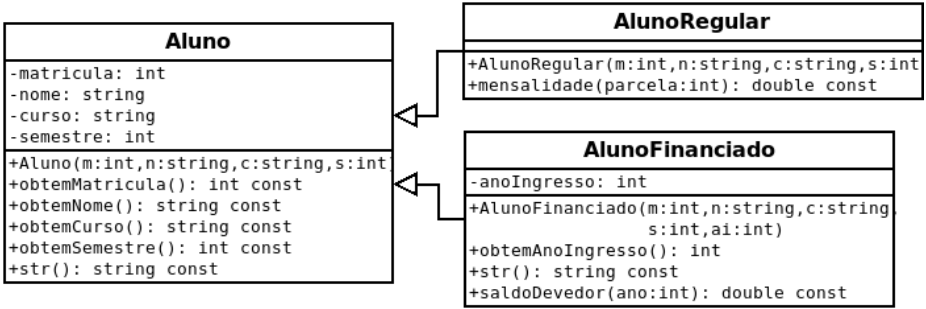


- As duas classes **têm atributos e métodos iguais**!
- Se há **código redundante**, a manutenção será problemática, pois a probabilidade de fazer uma alteração em uma das classes e esquecer de fazer a mesma alteração na outra é grande

Exemplo: Herança

- As classes `AlunoRegular` e `AlunoFinanciado` podem ser entendidas como um tipo especializado da classe Aluno
- Pois compartilham as características comuns a todos os tipos de alunos, porém possuem algumas características a mais que particularizam sua categoria ou classe
- É necessário encontrar uma forma de expressar que certas classes **compartilham** características (**atributos** e **métodos**) com outras

Exemplo: Herança



- `AlunoRegular` e `AlunoFinanciado` estendem `Aluno`
- `AlunoRegular` e `AlunoFinanciado` herdam atributos e métodos de `Aluno`
- Logo, o objeto instanciado "carrega" todos atributos e métodos

Exemplo: Herança

- Note que as propriedades comuns dos dois tipos de alunos foram concentradas em uma classe chamada `Aluno`
- Diz-se, então, que as classes `AlunoRegular` e `AlunoFinanciado` herdam as propriedades (atributos e métodos) de `Aluno` e acrescentam suas próprias propriedades a essa descrição
- Observe os exemplos de construção dos dois objetos a seguir:

```
AlunoRegular     ar(100001,"Beatriz␣Silva","4/600",8);
AlunoFinanciado  af(100002,"Augusto␣Alcantara","4/450",6,2018);
```

Exemplo: Aluno.hpp

```
#ifndef _ALUNO_HPP
#define _ALUNO_HPP
#include <string>

using namespace std;

class Aluno {
private:
  int matricula;
  string nome;
  string curso;
  int semestre;
public:
  const double valorMensal = 1000.0;
  Aluno(int m, string n, string c, int s);
  int obtemMatricula() const;
  string obtemNome() const;
  string obtemCurso() const;
  int obtemSemestre() const;
  string str() const;
};
#endif
```

Exemplo: Aluno.cpp

```
#include <sstream>
#include "Aluno.hpp"

Aluno::Aluno(int m, string n, string c, int s) {
  matricula = m;
  nome = n;
  curso = c;
  semestre = s;
}

int Aluno::obtemMatricula() const { return matricula; }

string Aluno::obtemNome() const { return nome; }

string Aluno::obtemCurso() const { return curso; }

int Aluno::obtemSemestre() const { return semestre; }

string Aluno::str() const {
  stringstream ss;
  ss << "Aluno: " << nome << endl;
  ss << "- matricula: " << matricula << endl;
  ss << "- curso: " << curso << endl;
  ss << "- semestre: " << semestre;
  return ss.str();
}
```

Exemplo: AlunoRegular.hpp

```
#ifndef _ALUNOREGULAR_HPP
#define _ALUNOREGULAR_HPP
#include "Aluno.hpp"

class AlunoRegular : public Aluno {
public:
  AlunoRegular(int m, string n, string c, int s);
  double mensalidade(int parcela) const;
};
#endif
```

Exemplo: AlunoRegular.cpp

```
#include "AlunoRegular.hpp"

AlunoRegular::AlunoRegular(int m, string n, string c, int s) : Aluno(m,n,c,s) {}

double AlunoRegular::mensalidade(int parcela) const {
  if (parcela>=1 && parcela<=3)
     return valorMensal * 1.2;
  return valorMensal;
}
```

Exemplo: AlunoFinanciado.hpp

```
#ifndef _ALUNOFINANCIADO_HPP
#define _ALUNOFINANCIADO_HPP
#include "Aluno.hpp"

class AlunoFinanciado : public Aluno {
private:
  int anoIngresso;
public:
  AlunoFinanciado(int m, string n, string c, int s, int ai);
  int obtemAnoIngresso() const;
  string str() const;
  double saldoDevedor(int ano) const;
};
#endif
```

Exemplo: AlunoFinanciado.cpp

```cpp
#include <sstream>
#include "AlunoFinanciado.hpp"

AlunoFinanciado::AlunoFinanciado(int m, string n, string c, int s, int ai) : Aluno(m,n,c,s) {
  anoIngresso = ai;
}

int AlunoFinanciado::obtemAnoIngresso() const {
  return anoIngresso;
}

string AlunoFinanciado::str() const {
  stringstream ss;
  ss << Aluno::str() << endl;
  ss << "-␣ingresso:␣" << anoIngresso;
  return ss.str();
}

double AlunoFinanciado::saldoDevedor(int ano) const {
  int dif = ano - anoIngresso;
  return dif * 12 * valorMensal;
}
```

Exemplo: main.cpp

```
#include <iostream>
#include "AlunoRegular.hpp"
#include "AlunoFinanciado.hpp"

int main() {
  AlunoRegular    ar(100001,"Beatriz␣Silva","4/600",8);
  AlunoFinanciado af(100002,"Augusto␣Alcantara","4/450",6,2018);
  cout << ar.str() << endl;
  cout << "-␣mensalidades:";
  for (int i=1; i<=6; ++i)
      cout << "␣" << ar.mensalidade(i);
  cout << endl;
  cout << af.str() << endl;
  cout << "-␣saldo␣devedor:␣" << af.saldoDevedor(2023) << endl;
  return 0;
}
```

Lista de Exercícios

Exercício 1

1. Considere a solução para o exercício 1 proposto na aula sobre "Makefile e Princípios de POO" (Livraria), e reimplemente-o para que ele tenha suporte a Herança, tornando-o assim mais simples e evitando códigos redundantes.

Exercício 2

2. Crie uma hierarquia de classes para representar pessoas, alunos e professores:
   a. Uma pessoa tem um nome e um RG.
   b. Um aluno tem os mesmos atributos que uma pessoa, mas também tem um número de matrícula e um ano de ingresso.
   c. Um professor tem os mesmos atributos que uma pessoa, mas também tem uma unidade ("EP" para Escola Politécnica, "EN" para Escola de Negócios, etc.), um ano de ingresso, e um salário fixo.

   Crie os métodos básicos de cada classe: construtor, *getters*, *setters* e `str()`.
   Ao final, escreva um programa para testar a hierarquia, criando uma pessoa, um aluno e um professor.

Exercício 3

3. Um sistema apresenta duas classes:
   a. `ProfessorTI`: professor de tempo integral, 40 horas, com salário fixo;
   b. `ProfessorH`: professor horista, com até 20 horas e salário calculado com base no número de horas.

As classes são definidas como:

ProfessorTI
-nome: string -matricula: long -cargaHoraria: int = 40 -salario: double
+ProfessorTI(nome:string,matribula:long) +obtemNome(): string const +obtemMatricula(): long const +obtemCargaHoraria(): int const +obtemSalario(): double const +defineSalario(salario:double): void

ProfessorH
-nome: string -matricula: long -cargaHoraria: int -salarioHora: double
+ProfessorH(nome:string,matribula:long,cargaHoraria:int) +obtemNome(): string const +obtemMatricula(): long const +obtemCargaHoraria(): int const +obtemSalario(): double const +defineSalarioHora(salarioHora:double): void

Exercício 3

3. (Continuação)
   Ambas têm diversos membros em comum. Então:
   - Crie uma classe `Professor` com o que há de comum entre `ProfessorTI` e `ProfessorH`;
   - Redefina `ProfessorTI` e `ProfessorH` como subclasses da classe `Professor`;
   - Implemente as três classes e um programa de teste.

Créditos

Créditos

- Estas lâminas contêm trechos de materiais disponibilizados pelos professores Rafael Garibotti, Matheus Trevisan, Daniel Callegari, Sandro Fiorini e Bernardo Copstein.

Soluções

Exercício 1: Makefile

```
CC = g++
CFLAGS = -c #-DDEBUG
EXEC = app

all:                                ${EXEC}

${EXEC}:                            app.o Promocao.o Produto.o ProdutoComPaginas.o ProdutoComPaginasEAno.o
                                    @${CC} -o ${EXEC} app.o Promocao.o Produto.o ProdutoComPaginas.o ProdutoComPaginasEAno.o

app.o:                              app.cpp Produto.hpp ProdutoComPaginas.hpp ProdutoComPaginasEAno.hpp
                                    @${CC} ${CFLAGS} app.cpp

Promocao.o:                         Promocao.cpp Promocao.hpp
                                    @${CC} ${CFLAGS} Promocao.cpp

Produto.o:                          Produto.cpp Produto.hpp Promocao.hpp
                                    @${CC} ${CFLAGS} Produto.cpp

ProdutoComPaginas.o:                ProdutoComPaginas.cpp ProdutoComPaginas.hpp
                                    @${CC} ${CFLAGS} ProdutoComPaginas.cpp

ProdutoComPaginasEAno.o:            ProdutoComPaginasEAno.cpp ProdutoComPaginasEAno.hpp
                                    @${CC} ${CFLAGS} ProdutoComPaginasEAno.cpp

clean:
                                    @rm -rf *.o ${EXEC}
```

Exercício 1: Promocao.hpp

```
#ifndef _PROMOCAO_HPP
#define _PROMOCAO_HPP
#include <string>

using namespace std;

class Promocao {
private:
  double percDesconto;
public:
  Promocao(double pD = 0.0);
  Promocao(string nomePromocao);
  ~Promocao();
  void definePromocao(string nomePromocao);
  void definePercDesconto(double pD);
  double obtemPercDesconto() const;
};
#endif
```

Exercício 1: Promocao.cpp

```cpp
#ifdef DEBUG
#include <iostream>
#endif
#include "Promocao.hpp"

Promocao::Promocao(double pD) {
  percDesconto = pD;
  #ifdef DEBUG
  cerr << "+ Promocao(" << percDesconto << ") criado..." << endl;
  #endif
}

Promocao::Promocao(string nomePromocao) {
  definePromocao(nomePromocao);
  #ifdef DEBUG
  cerr << "+ Promocao(" << percDesconto << ") criado..." << endl;
  #endif
}

Promocao::~Promocao() {
  #ifdef DEBUG
  cerr << "- Promocao(" << percDesconto << ") destruído..." << endl;
  #endif
}

void Promocao::definePromocao(string nomePromocao) {
  if ( nomePromocao == "Regular" ) percDesconto = 0.10;
  else if ( nomePromocao == "Liquidação" ) percDesconto = 0.30;
  else percDesconto = 0.0;
}

void Promocao::definePercDesconto(double pD) { percDesconto = pD; }
double Promocao::obtemPercDesconto() const { return percDesconto; }
```

Exercício 1: Produto.hpp

```
#ifndef _PRODUTO_HPP
#define _PRODUTO_HPP
#include "Promocao.hpp"

class Produto {
protected:
  string nome;
  double precoBase;
  Promocao *promocao;
public:
  Produto(string n="", double pB=0.0, Promocao *p=nullptr);
  ~Produto();
  string obtemNome() const;
  double obtemPrecoBase() const;
  double obtemPrecoFinal() const;
  Promocao *obtemPromocao() const;
  void defineNome(string n);
  void definePrecoBase(double pB);
  void definePromocao(Promocao *p);
  string str() const;
};
#endif
```

Exercício 1: Produto.cpp

```cpp
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include "Produto.hpp"

Produto::Produto(string n, double pB, Promocao *p) {
  nome = n;  precoBase = pB;  promocao = p;
  #ifdef DEBUG
  cerr << "+ Produto(" << str()  << ") criado..." << endl;
  #endif
}

Produto::~Produto() {
  #ifdef DEBUG
  cerr << "- Produto(" << str() << ") destruído..." << endl;
  #endif
}

string Produto::obtemNome() const { return nome; }
double Produto::obtemPrecoBase() const { return precoBase; }
double Produto::obtemPrecoFinal() const { return (promocao==nullptr)? precoBase : precoBase-precoBase*promocao->obtemPercDesconto(); }
Promocao *Produto::obtemPromocao() const { return promocao; }
void Produto::defineNome(string n) { nome = n; }
void Produto::definePrecoBase(double pB) { precoBase = pB; }
void Produto::definePromocao(Promocao *p) { promocao = p; }

string Produto::str() const {
  stringstream ss;
  ss << nome << "," << precoBase << ",";
  if ( promocao == nullptr ) ss << "nullptr";
  else ss << promocao->obtemPercDesconto();
  return ss.str();
}
```

Exercício 1: ProdutoComPaginas.hpp

```
#ifndef _PRODUTOCOMPAGINAS_HPP
#define _PRODUTOCOMPAGINAS_HPP
#include "Produto.hpp"

class ProdutoComPaginas : public Produto {
protected:
  int paginas;
public:
  ProdutoComPaginas(string n="", double pB=0.0, Promocao *p=nullptr, int pg=0);
  ~ProdutoComPaginas();
  int obtemPaginas() const;
  void definePaginas(int pg);
  string str() const;
};
#endif
```

Exercício 1: ProdutoComPaginas.cpp

```
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include "ProdutoComPaginas.hpp"

ProdutoComPaginas::ProdutoComPaginas(string n, double pB, Promocao *p, int pg) : Produto(n,pB,p) {
  paginas = pg;
  #ifdef DEBUG
  cerr << "+ ProdutoComPaginas(" << str() << ") criado..." << endl;
  #endif
}

ProdutoComPaginas::~ProdutoComPaginas() {
  #ifdef DEBUG
  cerr << "- ProdutoComPaginas(" << str() << ") destruído..." << endl;
  #endif
}

int ProdutoComPaginas::obtemPaginas() const { return paginas; }
void ProdutoComPaginas::definePaginas(int pg) { paginas = pg; }

string ProdutoComPaginas::str() const {
  stringstream ss;
  ss << Produto::str() << "," << paginas;
  return ss.str();
}
```

Exercício 1: ProdutoComPaginasEAno.hpp

```
#ifndef _PRODUTOCOMPAGINASEANO_HPP
#define _PRODUTOCOMPAGINASEANO_HPP
#include "ProdutoComPaginas.hpp"

class ProdutoComPaginasEAno : public ProdutoComPaginas {
protected:
  int ano;
public:
  ProdutoComPaginasEAno(string n="", double pB=0.0, Promocao *p=nullptr, int pg=0, int a=0);
  ~ProdutoComPaginasEAno();
  int obtemAno() const;
  void defineAno(int pg);
  string str() const;
};
#endif
```

Exercício 1: ProdutoComPaginasEAno.cpp

```cpp
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include "ProdutoComPaginasEAno.hpp"

ProdutoComPaginasEAno::ProdutoComPaginasEAno(string n, double pB, Promocao *p, int pg, int a) : ProdutoComPaginas(n,pB,p,pg) {
  ano = a;
  #ifdef DEBUG
  cerr << "+ ProdutoComPaginasEAno(" << str() << ") criado..." << endl;
  #endif
}

ProdutoComPaginasEAno::~ProdutoComPaginasEAno() {
  #ifdef DEBUG
  cerr << "- ProdutoComPaginasEAno(" << str() << ") destruído..." << endl;
  #endif
}

int ProdutoComPaginasEAno::obtemAno() const { return ano; }
void ProdutoComPaginasEAno::defineAno(int a) { ano = a; }

string ProdutoComPaginasEAno::str() const {
  stringstream ss;
  ss << ProdutoComPaginas::str() << "," << ano;
  return ss.str();
}
```

Exercício 1: app.cpp

```cpp
#include <iostream>
#include "Produto.hpp"
#include "ProdutoComPaginas.hpp"
#include "ProdutoComPaginasEAno.hpp"

using namespace std;

int main() {
  Promocao semPromocao(0.0);
  Promocao regular("Regular"); // 0.10
  Promocao liquidacao("Liquidação"); // 0.30

  Produto lapis1("Lápis 1",1.00,&semPromocao),
          lapis2("Lápis 2",1.00,&regular),
          lapis3("Lápis 3",1.00,&liquidacao);
  cout << lapis1.str() << endl;
  cout << lapis2.str() << endl;
  cout << lapis3.str() << endl;

  ProdutoComPaginas caderno1("Caderno1",10.00,&semPromocao,100),
                    caderno2("Caderno2",20.00,&regular,200),
                    caderno3("Caderno3",30.00,&liquidacao,300);
  cout << caderno1.str() << endl;
  cout << caderno2.str() << endl;
  cout << caderno3.str() << endl;

  ProdutoComPaginasEAno agenda1("Agenda 2023",50.00,&semPromocao,400,2023),
                        agenda2("Agenda 2024",50.00,&regular,400,2024),
                        agenda3("Agenda 2025",50.00,&liquidacao,400,2025);
  cout << agenda1.str() << endl;
  cout << agenda2.str() << endl;
  cout << agenda3.str() << endl;

  return 0;
}
```

Exercício 2: Makefile

```
CC = g++
CFLAGS = -c #-DDEBUG
EXEC = app

all:                            ${EXEC}

${EXEC}:                        app.o Professor.o Aluno.o Pessoa.o
                                @${CC} -o ${EXEC} app.o Professor.o Aluno.o Pessoa.o

app.o:                          app.cpp Pessoa.hpp Aluno.hpp Professor.hpp
                                @${CC} ${CFLAGS} app.cpp

Pessoa.o:                       Pessoa.cpp Pessoa.hpp
                                @${CC} ${CFLAGS} Pessoa.cpp

Aluno.o:                        Aluno.cpp Aluno.hpp
                                @${CC} ${CFLAGS} Aluno.cpp

Aluno.hpp:                      Pessoa.hpp

Professor.o:                    Professor.cpp Professor.hpp
                                @${CC} ${CFLAGS} Professor.cpp

Professor.hpp:                  Pessoa.hpp

clean:
                                @rm -rf *.o ${EXEC}
```

Exercício 2: Pessoa.hpp

```
#ifndef _PESSOA_HPP
#define _PESSOA_HPP

#include <string>

using namespace std;

class Pessoa {
protected:
  string nome;
  int rg;
public:
  Pessoa(string nome="", int rg=0);
  ~Pessoa();
  void defineNome(string nome);
  void defineRG(int rg);
  string obtemNome() const;
  int obtemRG() const;
  string str() const;
};

#endif
```

Exercício 2: Pessoa.cpp

```
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include "Pessoa.hpp"

Pessoa::Pessoa(string n, int r) {
  nome = n;
  rg = r;
  #ifdef DEBUG
  cerr << "+ Pessoa(" << nome << "," << rg << ") criado..." << endl;
  #endif
}

Pessoa::~Pessoa() {
  #ifdef DEBUG
  cerr << "- Pessoa(" << nome << "," << rg << ") destruído..." << endl;
  #endif
}

void Pessoa::defineNome(string n) {  nome = n;  }
void Pessoa::defineRG(int r) {  rg = r;  }
string Pessoa::obtemNome() const {  return nome;  }
int Pessoa::obtemRG() const {  return rg;  }

string Pessoa::str() const {
  stringstream ss;
  ss << "Nome: " << nome << endl;
  ss << "RG: " << rg << endl;
  return ss.str();
}
```

Exercício 2: Aluno.hpp

```
#ifndef ALUNO_HPP
#define ALUNO_HPP

#include "Pessoa.hpp"

class Aluno : public Pessoa {
private:
  int matricula;
  int ano;
public:
  Aluno(string n="", int r=0, int m=0, int a=0);
  ~Aluno();
  void defineMatricula(int m);
  void defineAno(int a);
  int obtemMatricula() const;
  int obtemAno() const;
  string str() const;
};

#endif
```

Exercício 2: Professor.cpp

```
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include "Professor.hpp"

Professor::Professor(string n, int r, string u, int a, double s) : Pessoa(n, r) {
  unidade = u;
  ano = a;
  salario = s;
  #ifdef DEBUG
  cerr << "+ Professor(" << nome << "," << rg << "," << unidade << "," << ano << "," << salario << ") criado..." << endl;
  #endif
}

Professor::~Professor() {
  #ifdef DEBUG
  cerr << "- Professor(" << nome << "," << rg << "," << unidade << "," << ano << "," << salario << ") destruído..." << endl;
  #endif
}

void Professor::defineUnidade(string u) {  unidade = u;  }
void Professor::defineAno(int a) {  ano = a;  }
void Professor::defineSalario(double s) {  salario = s;  }
string Professor::obtemUnidade() const {  return unidade;  }
int Professor::obtemAno() const {  return ano;  }
double Professor::obtemSalario() const {  return salario;  }

string Professor::str() const {
  stringstream ss;
  ss << Pessoa::str();
  ss << "Unidade: " << unidade << endl;
  ss << "Ano de ingresso: " << ano << endl;
  ss << "Salario: " << salario << endl;
  return ss.str();
}
```

Exercício 2: Professor.hpp

```
#ifndef PROFESSOR_HPP
#define PROFESSOR_HPP

#include "Pessoa.hpp"

class Professor : public Pessoa {
private:
  string unidade;
  int ano;
  double salario;
public:
  Professor(string n, int r, string u, int a, double s);
  ~Professor();
  void defineUnidade(string u);
  void defineAno(int a);
  void defineSalario(double s);
  string obtemUnidade() const;
  int obtemAno() const;
  double obtemSalario() const;
  string str() const;
};

#endif
```

Exercício 2: Aluno.cpp

```cpp
#ifdef DEBUG
#include <iostream>
#endif
#include <sstream>
#include "Aluno.hpp"

Aluno::Aluno(string n, int r, int m, int a) : Pessoa(n, r) {
  matricula = m;
  ano = a;
  #ifdef DEBUG
  cerr << "+ Aluno(" << nome << "," << rg << "," << matricula << "," << ano << ") criado..." << endl;
  #endif
}

Aluno::~Aluno() {
  #ifdef DEBUG
  cerr << "- Aluno(" << nome << "," << rg << "," << matricula << "," << ano << ") destruído..." << endl;
  #endif
}

void Aluno::defineMatricula(int m) {  matricula = m;  }
void Aluno::defineAno(int a) {  ano = a;  }
int Aluno::obtemMatricula() const {  return matricula;  }
int Aluno::obtemAno() const {  return ano;  }

string Aluno::str() const {
  stringstream ss;
  ss << Pessoa::str();
  ss << "Matrícula: " << matricula << endl;
  ss << "Ano: " << ano << endl;
  return ss.str();
}
```

Exercício 2: app.cpp

```cpp
#include <iostream>
#include "Pessoa.hpp"
#include "Aluno.hpp"
#include "Professor.hpp"

int main () {
  Pessoa pessoa("Pessoa",1);
  Aluno aluno("Estudante", 1234, 1111, 2016);
  Professor professor("Professor", 2222, "EP", 2015, 10000.01);
  cout << pessoa.str() << endl;
  cout << aluno.str() << endl;
  cout << professor.str() << endl;
  return 0;
}
```