
Herança

Sumário

- Introdução;
- Herança Simples;
- Tipos de Herança;
- Classe Abstrata;
- Conversão de Tipos entre Base e Derivada;
- Níveis de Herança;
- Herança Múltipla;
- Bibliografia;

Introdução

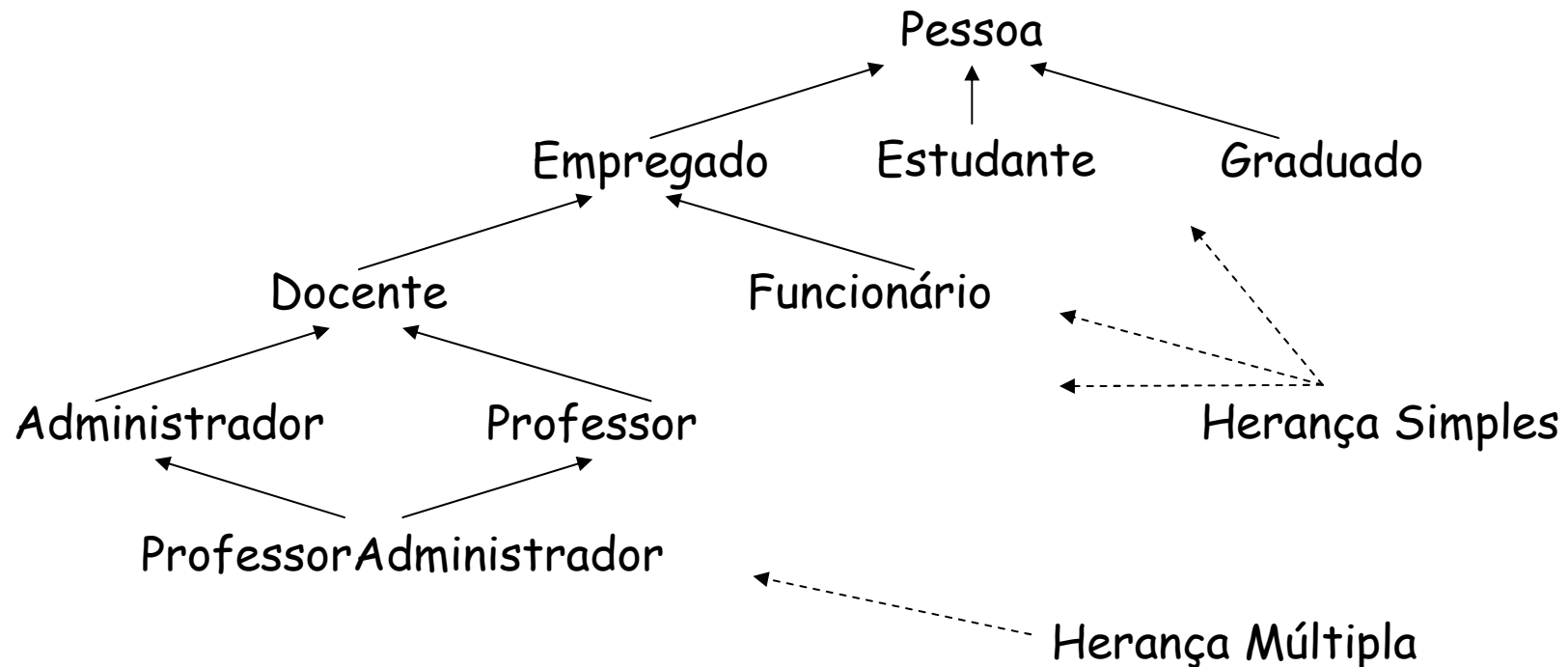
- Herança é o processo que permite criar uma classe que **herde todas as características** de outra existente.
- A nova classe é chamada de **classe derivada** (subclasse) e a classe existente, de **classe-base** (superclasse);
- **É possível incluir**, na classe derivada, características que próprias adicionais além das herdadas;

Introdução

- É importante identificar a diferença entre **composição** e **herança**:
 - Na herança, um objeto da subclasse "é um" objeto da superclasse. Por exemplo, o carro é um veículo;
 - Enquanto que na composição um objeto "tem um" outro objeto. Por exemplo, o carro tem uma direção.
- O processo de herança vai além da derivação simples. Uma classe derivada pode herdar características de **mais de uma classe-base**;
- Uma das maiores vantagens do processo de herança é a **reutilização de código**;

Herança Simples

- A herança costuma formar estruturas hierárquicas do tipo **árvore**, vejamos:



Herança Simples

```
1 public class EmpregadoComissao
2 {
3     protected String codigo;
4     protected double vendas; // vendas brutas semanais
5     protected double comissao; // porcentagem da comissão
6
7     public EmpregadoComissao( String cod, double sales, double rate )
8     {
9         codigo = cod;
10        setVendas( sales ); // valida e armazena as vendas brutas
11        setComissao( rate ); // valida e armazena a taxa de comissão
12    }
13    public void setCodigo( String cod )
14    { ... }
17    public String getCodigo()
18    { ... }
21    public void setVendas( double sales )
22    { ... }
25    public double getVendas()
26    { ... }
29    public void setComissao( double rate )
30    {
31        comissao = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
32    }
33    public double getComissao()
34    { ... }
```

Classe-base (superclasse)

Palavra-chave protected, usada para dar permissão de acesso apenas às classes derivadas

Métodos definidos na classe-base

```
37 public double lucros()
38 {
39     return comissao * vendas;
40 }
41
42 public String toString()
43 {
44     return String.format( "%s: %s \n%s: %.2f\n%s: %.2F",
45         "comissao do empregado", codigo,
46         "Vendas", vendas,
47         "Comissão", comissao );
48 }
49 }
```

Herança Simples

```
1 public class BasePlusCommissionEmployee3 extends EmpregadoComissionado
2 {
3     private double salarioBase; // salário-base por semana
4
5     public BasePlusCommissionEmployee3( String cod, double vendas, double rate, double
6     {
7         super( cod, vendas, rate );
8         setSalarioBase( salario ); // valida e armazena salário-base
9     }
10    public void setSalarioBase( double salario )
11    {
12        salarioBase = ( salario < 0.0 ) ? 0.0 : salario;
13    }
14    public double getSalarioBase()
15    {
16        return salarioBase;
17    }
18    public double lucros()
19    {
20        return salarioBase + ( comissao * vendas );
21    }
22
23    public String toString()
24    {
25        return String.format( "%s: %s \n%s: %.2f\n%s: %.2f\n%s: %.2f",
26            "comissao do empregado", codigo,
27            "Vendas", vendas,
28            "Comissão", comissao,
29            "Salário Base", salarioBase);
30    }
}
```

Palavra-chave extends do Java usada para herdar classes

Outros métodos definidos pela classe derivada

Utilização de método definido na classe base

Herança Simples

```
... * BasePlusCommissionEmployee3.java * BasePlusCommissionEmployeeTest3.java x
1 public class BasePlusCommissionEmployeeTest3
2 {
3     public static void main( String args[] )
4     {
5         BasePlusCommissionEmployee3 empregado =
6             new BasePlusCommissionEmployee3( "333-33-3333", 5000, .04, 300 );
7
8         System.out.println( "Informações do Empregado: \n" );
9         System.out.printf( "%s %s\n", "Código",
10             empregado.getCodigo() );
11         System.out.printf( "%s %.2f\n", "Vendas Brutas",
12             empregado.getVendas() );
13         System.out.printf( "%s %.2f\n", "Comissão",
14             empregado.getComissao() );
15         System.out.printf( "%s %.2f\n", "Salário Base",
16             empregado.getSalarioBase() );
17
18         empregado.setSalarioBase( 1000 ); // configura o salário-base
19
20         System.out.printf( "\n%s:\n\n%s\n",
21             "Atualização de Empregado pelo método toString",
22             empregado.toString() );
23     }
24 }
```

Criação do objeto da classe derivada

Chamada ao método especializado (toString) definido na classe derivada

Herança Simples - C++

```
AulaHeranca.cpp horista.h *Empregado.h X
1#include <cstring>
2#include <cassert>
3#include <iostream>
4using namespace std;
5
6class Empregado{
7public:
8    Empregado( const char *, const char * );
9    void print() const;
10    ~Empregado();
11private:
12    char *primeiroNome;
13    char *ultimoNome;
14};
15
16Empregado::Empregado( const char *primeiro, const char *ultimo )
17{
18    primeiroNome = new char[ strlen( primeiro ) + 1];
19    assert( primeiroNome != 0 );
20    strcpy( primeiroNome, primeiro );
21    ultimoNome = new char[ strlen( ultimo ) + 1];
22    assert( ultimoNome != 0 );
23    strcpy( ultimoNome, ultimo );
24}
25void Empregado::print() const
26{    cout << primeiroNome << ' ' << ultimoNome; }
27
28Empregado::~Empregado()
29{
30    delete [] primeiroNome;
31    delete [] ultimoNome;
32}
```

Herança Simples - C++

```
AulaHeranca.cpp horista.h X
1#include "Empregado.h"
2
3class Horista : public Empregado {
4public:
5    Horista( const char *, const char *, double, double );
6    double getSalario() const;
7    void print() const;
8private:
9    double salHora;
10   double horas;
11};
12
13Horista::Horista( const char *primeiroNome,
14                 const char *ultimoNome,
15                 double initHoras, double iniSalHora ): Empregado ( primeiroNome,
16{
17    horas = initHoras; //Deveria validar
18    salHora = iniSalHora; // Deveria validar
19}
20
21double Horista::getSalario() const { return salHora * horas; }
22
23void Horista::print() const
24{
25    cout << "Horista::print() está sendo executada\n\n";
26    Empregado::print();
27
28    cout << " é um horista com pagamento de R$ " << getSalario() << endl;
29}
```

Herança Simples - C++



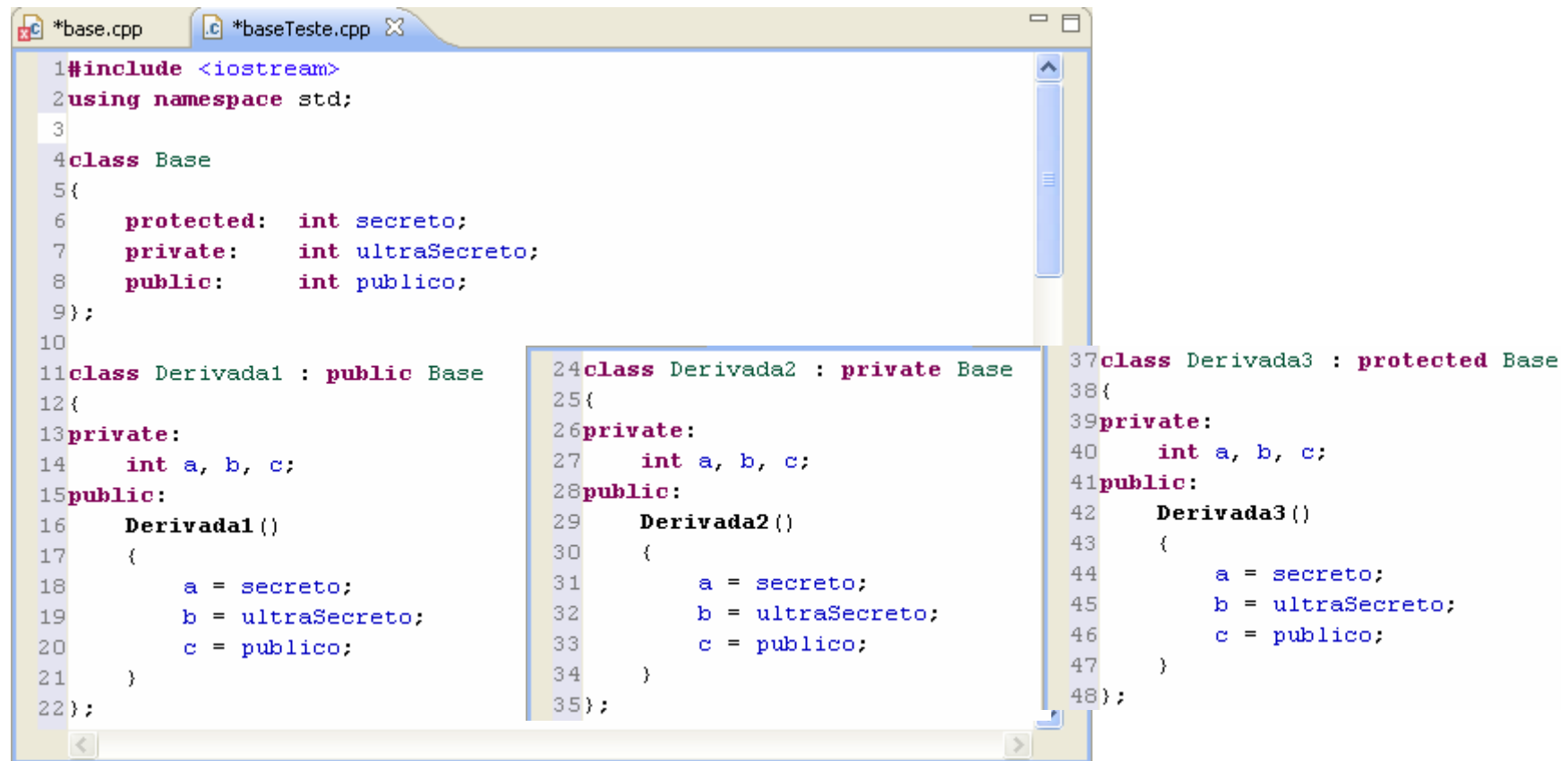
```
1//=====
2// Name      : AulaHeranca.cpp
3// Author    : Leonardo
4// Version   :
5// Copyright : Your copyright notice
6// Description: Hello World in C++, Ansi-style
7//=====
8
9#include "horista.h"
10using namespace std;
11
12int main() {
13    Horista h("Maria", "Silva", 40.0, 10.00);
14    h.print();
15    return 0;
16}
17
```

Tipos de Herança

- Herança Pública:
 - Indica que os membros públicos da classe derivada e os membros protegidos da classe-base serão membros protegidos da classe derivada;
- Herança Privada:
 - Indica que tanto os membros públicos quanto os protegidos da classe-base serão membros privados da classe derivada;
- Herança Protegida:
 - Indica que tanto os membros públicos quanto os protegidos da classe-base serão protegidos da classe derivada;

Tipos de Herança

- Vejamos um exemplo:



```
1#include <iostream>
2using namespace std;
3
4class Base
5{
6    protected:    int secreto;
7    private:      int ultraSecreto;
8    public:       int publico;
9};
10
11class Derivada1 : public Base
12{
13private:
14    int a, b, c;
15public:
16    Derivada1()
17    {
18        a = secreto;
19        b = ultraSecreto;
20        c = publico;
21    }
22};
23
24class Derivada2 : private Base
25{
26private:
27    int a, b, c;
28public:
29    Derivada2()
30    {
31        a = secreto;
32        b = ultraSecreto;
33        c = publico;
34    }
35};
36
37class Derivada3 : protected Base
38{
39private:
40    int a, b, c;
41public:
42    Derivada3()
43    {
44        a = secreto;
45        b = ultraSecreto;
46        c = publico;
47    }
48};
```

Tipos de Herança

- Vejamos um exemplo:



```
*base.cpp  *baseTeste.cpp X
50 int main() {
51     int x;
52
53     Derivada1 Obj1;
54     x = Obj1.secreto;
55     x = Obj1.ultraSecreto;
56     x = Obj1.publico;
57
58     Derivada2 Obj2;
59     x = Obj2.secreto;
60     x = Obj2.ultraSecreto;
61     x = Obj2.publico;
62
63     Derivada3 Obj3;
64     x = Obj3.secreto;
65     x = Obj3.ultraSecreto;
66     x = Obj3.publico;
67
68     return 0;
69 }
70
```

Tipos de Herança

The image shows a C++ IDE window titled 'base.cpp' with the following code:

```
1#include <iostream>
2using namespace std;
3
4class Base
5{
6    protected: int secreto;
7    private:   int ultraSecreto;
8    public:   int publico;
9};
10
11class Derivada1 : public Base
12{
13private:
14    int a, b, c;
15public:
16    Derivada1()
17    {
18        a = secreto;
19        b = ultraSecreto;
20        c = publico;
21    }
22};
```

The IDE also shows a 'Console' window with the following output:

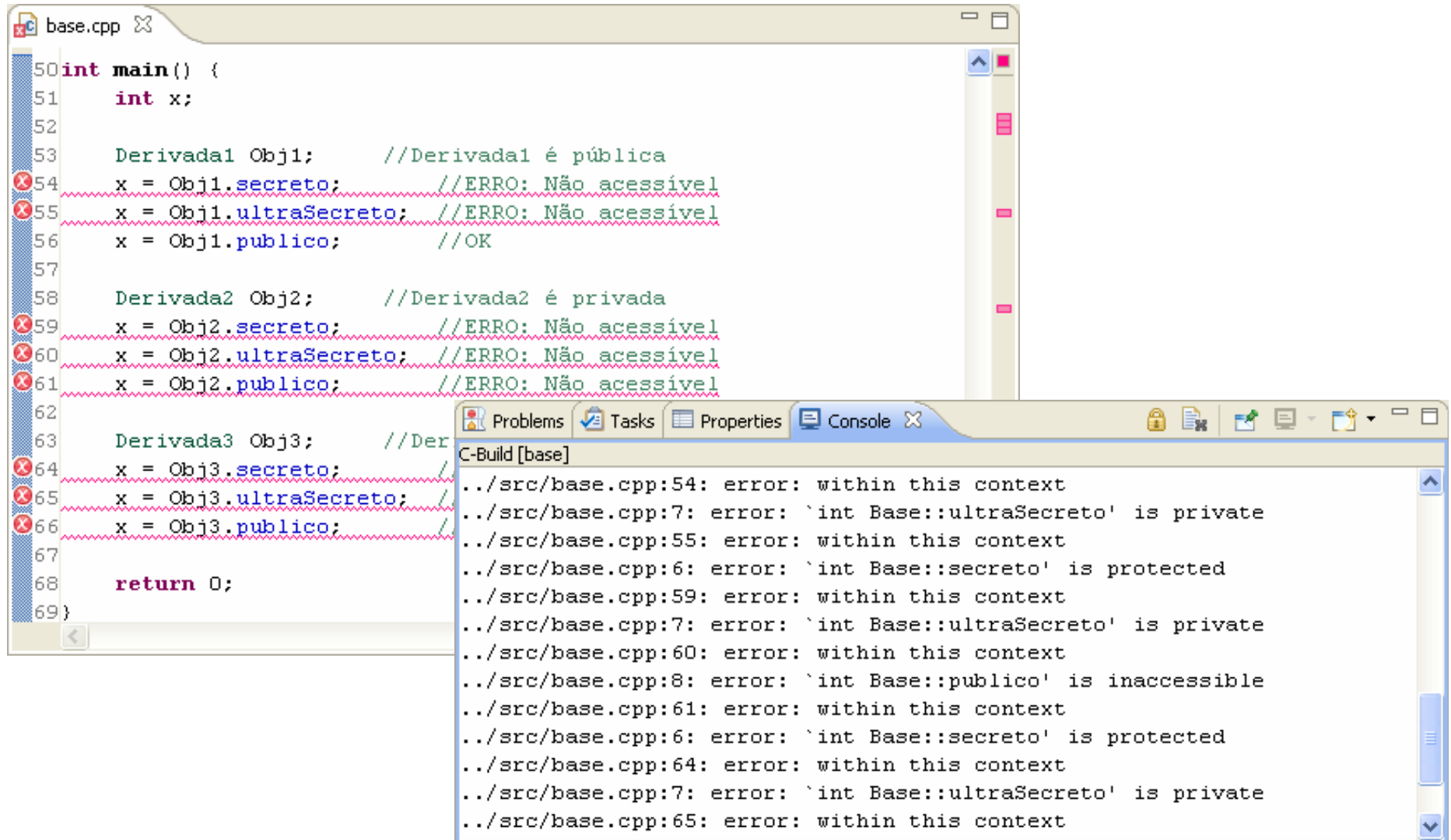
```
C-Build [base]
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/base.d"
-MT"src/base.d" -o"src/base.o" "../src/base.cpp"
../src/base.cpp: In constructor `Derivada1::Derivada1()':
../src/base.cpp:7: error: `int Base::ultraSecreto' is private
../src/base.cpp:19: error: within this context
../src/base.cpp:7: error: `int Base::ultraSecreto' is private
../src/base.cpp:19: error: within this context
```

Tipos de Herança

```
base.cpp
24 class Derivada2 : private Base
25 {
26 private:
27     int a, b, c;
28 public:
29     Derivada2()
30     {
31         a = secreto;
32         b = ultraSecreto;
33         c = publico;
34     }
35 };
36
37 class Derivada3 : protected Base
38 {
39 private:
40     int a, b, c;
41 public:
42     Derivada3()
43     {
44         a = secreto;
45         b = ultraSecreto;
46         c = publico;
47     }
48 };

Problems Tasks Properties Console
C-Build [base]
../src/base.cpp:32: error: within this context
../src/base.cpp: In constructor `Derivada3::Derivada3()':
../src/base.cpp:7: error: `int Base::ultraSecreto' is private
../src/base.cpp:45: error: within this context
../src/base.cpp:7: error: `int Base::ultraSecreto' is private
../src/base.cpp:45: error: within this context
../src/base.cpp: In function `int main()':
../src/base.cpp:6: error: `int Base::secreto' is protected
```


Tipos de Herança



```
base.cpp
50 int main() {
51     int x;
52
53     Derivada1 Obj1;    //Derivada1 é pública
54     x = Obj1.secreto; //ERRO: Não acessível
55     x = Obj1.ultraSecreto; //ERRO: Não acessível
56     x = Obj1.publico; //OK
57
58     Derivada2 Obj2;    //Derivada2 é privada
59     x = Obj2.secreto; //ERRO: Não acessível
60     x = Obj2.ultraSecreto; //ERRO: Não acessível
61     x = Obj2.publico; //ERRO: Não acessível
62
63     Derivada3 Obj3;    //Der
64     x = Obj3.secreto; //
65     x = Obj3.ultraSecreto; //
66     x = Obj3.publico; //
67
68     return 0;
69 }
```

Problems Tasks Properties Console

C-Build [base]

```
./src/base.cpp:54: error: within this context
./src/base.cpp:7: error: `int Base::ultraSecreto' is private
./src/base.cpp:55: error: within this context
./src/base.cpp:6: error: `int Base::secreto' is protected
./src/base.cpp:59: error: within this context
./src/base.cpp:7: error: `int Base::ultraSecreto' is private
./src/base.cpp:60: error: within this context
./src/base.cpp:8: error: `int Base::publico' is inaccessible
./src/base.cpp:61: error: within this context
./src/base.cpp:6: error: `int Base::secreto' is protected
./src/base.cpp:64: error: within this context
./src/base.cpp:7: error: `int Base::ultraSecreto' is private
./src/base.cpp:65: error: within this context
```

Reescrevendo Métodos da Classe-Base

- Quando a classe base e a classe derivada definem funções com o mesmo nome o compilador terá que resolver o escopo das funções:
- A regra é a seguinte:
 - Se duas funções de mesmo nome existem, uma na classe-base e outra na classe derivadas, a função da classe derivada será executada se for chamada por meio de um objeto da classe derivada;
 - Se um objeto da classe-base é criado, usará sempre funções da própria classe-base, pois não conhece nada da classe derivada

Classe Abstrata

- Uma classe é denominada abstrata se nenhuma instância dela é criada. Vejamos:

```
1#include <iostream>
2#include <cstring>
3#include <iomanip>
4using namespace std;
5
6class Conta
7{
8private:
9    char nomeCliente[50];
10   int numConta;
11   float saldoBase;
12public:
13   void Get ()
14   {
15       cout << " Nome: "; cin.ignore(10, '\n'); gets(nomeCliente)
16       cout << " Número da Conta: "; cin >> numConta;
17       cout << " Saldo: "; cin >> saldoBase;
18   }
19   void Print ()
20   {
21       cout << " Nome: " << nomeCliente << endl;
22       cout << " Número da Conta: " << numConta << endl;
23       cout << " Saldo: "
24           << setiosflags(ios::fixed)
25           << setprecision(2) << saldoBase << endl;
26   }
27   float getSaldo ()
28   { return saldoBase; }
29};
```

Classe Abstrata

```
31 class ContaSimples : public Conta
32 {};
33
34 class ContaEspecial: public Conta
35 {
36 private:
37     float limiteConta;
38 public:
39     void Get ()
40     {
41         Conta::Get();
42         cout << " Limite: "; cin >> limiteConta;
43         cin.ignore(10, '\n'); //Limpa teclado
44     }
45     void Print ()
46     {
47         Conta::Print();
48         cout << " Limite: " << limiteConta << endl;
49         cout << " Saldo Total: "
50             << setiosflags(ios::fixed)
51             << setprecision(2) << (getSaldo() + limiteConta) << endl;
52     }
53};
```

Classe Abstrata

```
55 int main() {
56     ContaSimples c1;
57     ContaEspecial c2;
58
59     cout << "\n Digite os dados da conta simples " << endl;
60     c1.Get();
61     cout << "\n Digite os dados da conta especial " << endl;
62     c2.Get();
63
64     cout << "\n\n Conta Simples\n"; c1.Print();
65     cout << "\n\n Conta Especial\n"; c2.Print();
66     return 0;
67 }
```

Conversão de Tipos entre Base e Derivada

- Visto que `ContaEspecial` é um tipo de `Conta`, faz sentido pensar em converter um objeto da `ContaEspecial` em um objeto da classe `Conta`;
 - C++ permite a conversão implícita de um objeto da classe derivada em um objeto da sua classe-base; Por exemplo:

```
59     Conta Base;  
60     Conta Especial Derivada;  
61  
62     Derivada.Get();  
63  
64     Base = Derivada;  
65  
66     Derivada = Base;  
67
```

Níveis de Herança

- Uma classe pode ser derivada de outra classe, por sua vez, é também uma classe derivada. Vejamos:

```
class X
();

class Y : public X
();

class Z : public Y
();
```

- Z é derivada de Y que, por sua vez, é derivada de X;
- A hierarquia de classes pode ser descrita usando-se uma estrutura de árvore.
- A conversão implícita de objetos de classes derivadas para objetos de suas classes-base é a principal característica que permite polimorfismo;

Níveis de Herança

- No exemplo anterior, imagine que tenhamos decidido adicionar uma característica à conta especial;

```
class ContaPremio : public ContaEspecial
{
private:
    float premio;
public:
    void Get ()
    {
        ContaEspecial::Get ();
        cout << " Prêmio: "; cin >> premio
    }
    void Print ()
    {
        ContaEspecial::Print ();
        cout << " Prêmio: " << Premio << endl;
    }
};
```


Herança Múltipla

- Uma classe pode herdar as características de mais de uma classe-base. Esse processo é chamado de herança múltipla;

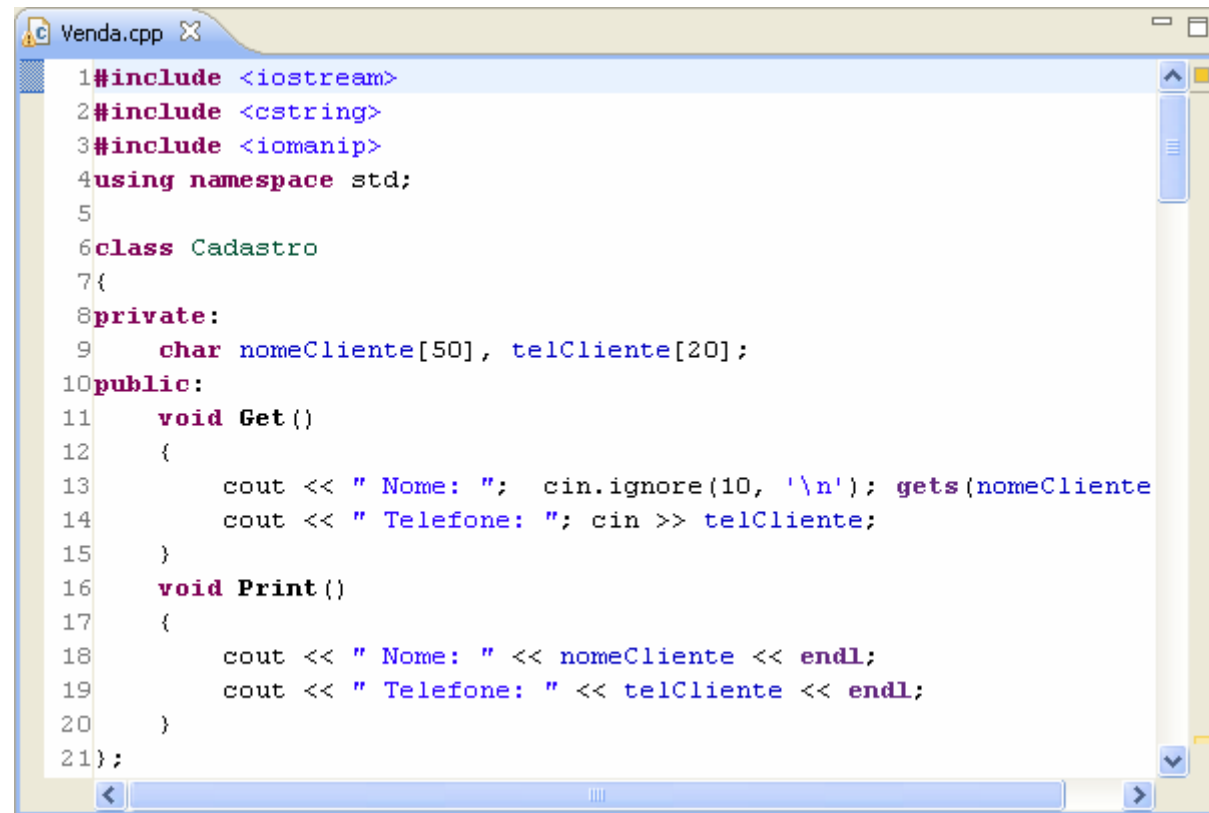
```
class X
();

class Y
();

class Z : public X, public Y //Z é derivada de X e de Y
();
```

- O difícil em herança múltipla é construir as classes e não a sintaxe;
- Java não dá suporte, explícito, à herança múltipla;

Herança Múltipla



```
Venda.cpp X
1#include <iostream>
2#include <cstring>
3#include <iomanip>
4using namespace std;
5
6class Cadastro
7{
8private:
9    char nomeCliente[50], telCliente[20];
10public:
11    void Get ()
12    {
13        cout << " Nome: "; cin.ignore(10, '\n'); gets(nomeCliente);
14        cout << " Telefone: "; cin >> telCliente;
15    }
16    void Print ()
17    {
18        cout << " Nome: " << nomeCliente << endl;
19        cout << " Telefone: " << telCliente << endl;
20    }
21};
```

Herança Múltipla

```
22
23 class Imovel
24 {
25 private:
26     char endImovel[30];
27     float areaUtil, areaTotal;
28     int quartos;
29 public:
30     void Get ()
31     {
32         cout << " Endereço: "; cin.ignore(10, '\n'); gets(endImovel);
33         cout << " Área Útil: "; cin >> areaUtil;
34         cout << " Área Total: "; cin >> areaTotal;
35     }
36     void Print ()
37     {
38         cout << " Endereço: " << endImovel << endl;
39         cout << " Área Útil: " << areaUtil << endl;
40         cout << " Área Total: " << areaTotal << endl;
41     }
42 };
43
```

Herança Múltipla

```
44 class Tipo
45 {
46 private:
47     char tipoImovel[20];
48 public:
49     void Get ()
50     {
51         cout << " Tipo: "; cin.ignore(10, '\n'); gets(tipoImovel)
52     }
53     void Print ()
54     {
55         cout << " Tipo: " << tipoImovel << endl;
56     }
57 };
58
```

Herança Múltipla

```
59 class Venda : private Cadastro, Imovel, Tipo
60 {
61 private:
62     float valor;
63 public:
64     void Get ()
65     {
66         cout << "\nProprietário: " << endl;
67         Cadastro::Get ();
68         cout << "Imóvel: " << endl;
69         Imovel::Get ();
70         Tipo::Get ();
71         cout << "Valor R$: "; cin >> valor;
72     }
73     void Print ()
74     {
75         cout << "\nProprietário: " << endl;
76         Cadastro::Print ();
77         cout << "Imóvel " << endl;
78         Imovel::Print ();
79         Tipo::Print ();
80         cout << "Valor R$: " << valor << endl;
81     }
82};
```

Herança Múltipla

```
113 int main()
114 {
115     Venda V;
116     Aluguel A;
117
118     cout << "Digite os dados do imóvel: Venda." << endl;
119     V.Get();
120     cout << "Digite os dados do imóvel: Aluguel." << endl;
121     A.Get();
122     cout << "Imóvel para Aluguel: " << endl;
123     A.Print();
124     return 0;
125 }
```

Exercício

- Definir uma classe Aluguel usando as classes Cadastro, Imovel e Tipo como classes-base. Inclua aluguelMensal e prazoAluguel. Acrescente duas funções, uma para a entrada de dados, Get(), e uma que imprima os dados, Print();

Herança Múltipla

```
84 class Aluguel : private Cadastro, Imovel, Tipo
85 {
86 private:
87     float aluguelMensal;
88     int prazoAluguel;
89     Cadastro Proprietario;
90 public:
91     void Get ()
92     {
93         cout << " Proprietário: "; Proprietario.Get ();
94         cout << " Inquilino: "; Cadastro::Get ();
95         cout << " Imóvel: "; Imovel::Get (); Tipo::Get ();
96         cout << "Aluguel: "; cin >> aluguelMensal;
97         cout << "Prazo do Contrato: "; cin >> prazoAluguel;
98     }
99     void Print ()
100    {
101        cout << " Proprietário: " << endl;
102        Proprietario.Print ();
103        cout << " Inquilino: " << endl;
104        Cadastro::Print ();
105        cout << " Imóvel: " << endl;
106        Imovel::Print ();
107        Tipo::Print ();
108        cout << "Aluguel: " << aluguelMensal << endl;
109        cout << "Prazo de Contrato: " << prazoAluguel << endl;
110    }
111};
```

Bibliografia

- Mizrahi, Victorine Viviane. *Treinamento em C++, módulo 2*. 2ª ed. São Paulo: 2006.
- Deitel, H. M. & Deitel, P. J. *C++: como programar*, Editora Bookman. 3ª ed. Porto Alegre: 2001.
- Deitel, H. M. & Deitel, P. J. *Java: como programar*, Editora Bookman. 6ª ed. São Paulo: 2005.