Classes e Objetos

Sumário

- Introdução;
- Escopo de Classe e Acesso a Membros de Classes;
  - Exercício;
- Os Métodos get e set;
- Separação de Interface e Implementação;
- Construtores e Destrutores;
- Uso do ponteiro `this`;

Sumário

- Objetos `const` e funções membro `const`;
- Composição;
- Funções `friend` e classes `friend`;
- Alocação dinâmica de memória;
- Classes Proxy;

Introdução

- Gênesis: No princípio criaram as estruturas:

ClassesObjetos.cpp

```
 9 #include <iostream>
10 using namespace std;
11
12 struct cliente
13 {
14     double vendas[12];
15     double totalAno;
16     int codigo;
17 };
18
19 void setComprasCliente( cliente &t )
20 {
21     for( int i = 1; i <= 12; i++ )
22     {
23         cout<< "Digite o volume de vendas para o mês " << i << ": ";
24         do{
25             cin >> t.vendas[i];
26             if ( t.vendas[i] < 0 )
27             {
28                 cout << "Valor de vendas Inválido!!! " << endl
29                      << "Redigite o volume de vendas para o mês " << i << ": ";
30             }
31         }while (t.vendas[i] < 0);
32     }
33 }
```

Tipo de dados

Campos da estrutura

Função que irá manipular a estrutura

Introdução

- Gênesis: No princípio criaram as estruturas:
  - Continuação...

ClassesObjetos.cpp

```
34
35 void printComprasAnual( cliente &t )
36 {
37     for (int i = 0; i < 12; i++)
38         t.totalAno += t.vendas[i];
39
40     cout << "O total anual de vendas foi : " << t.totalAno;
41 }
42 int main() {
43     cliente c;
44     c.totalAno = 0;
45     c.codigo = 0;
46     for ( int i = 0; i < 12; i++)
47         c.vendas [i] = 0.0;
48     cout << "Este programa é a primeira aplicação de Classes (Hello Class!!!)"
49     setComprasCliente(c);
50     printComprasAnual(c);
51     return 0;
52 }
```

Função que irá manipular a estrutura

c é uma variável do tipo cliente

Introdução

- Novo Testamento: Vieram as Classes

ClassesObjetos.cpp

```
 9 #include <iostream>
10 using namespace std;
11
12 class Cliente {
13 public:
14     Cliente();
15     void setComprasCliente();
16     void printComprasAnual();
17
18 private:
19     double vendas[12];
20     double totalAno;
21     int codigo;
22 };
23
24 Cliente::Cliente()
25 {
26     for ( int i = 0; i < 12; i++)
27         vendas [i] = 0.0;
28     codigo = 0;
29     totalAno = 0;
30 }
```

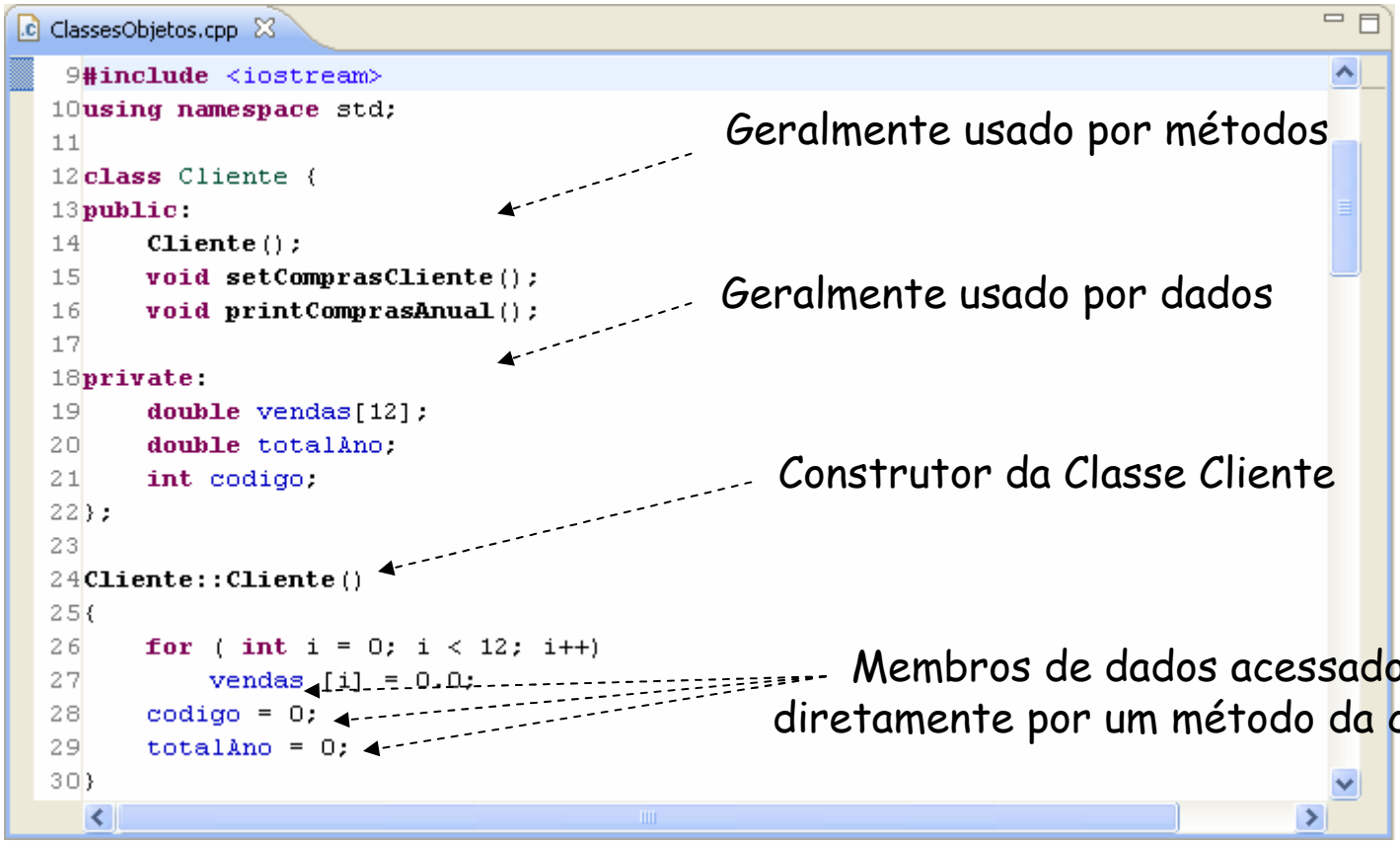
Definição da Classe Cliente

`public` torna os membros abaixo dela e antes do `private` públicos

`private` torna os membros abaixo dela privados, ou seja, acessados apenas por métodos da classe

Introdução

- Novo Testamento: Vieram as Classes



Introdução

ClassesObjetos.cpp

```
32 void Cliente::setComprasCliente()
33 {
34     for( int i = 1; i <= 12; i++ )
35     {
36         cout<< "Digite o volume de vendas para o mês " << i << ": ";
37         do{
38             cin >> vendas[i];
39             if ( vendas[i] < 0 )
40             {
41                 cout << "Valor de vendas Inválido!!! " << endl
42                      << "Redigite o volume de vendas para o mês " << i << ": ";
43             }
44         }while (vendas[i] < 0);
45     }
46 }
47 void Cliente::printComprasAnual()
48 {
49     for (int i = 0; i < 12; i++)
50         totalAno += vendas[i];
51
52     cout << "O total anual de vendas foi : " << totalAno;
53 }
54 int main() {
55     Cliente c;
56     cout << "Este programa é a primeira aplicação de Classes (Hello Class!!!)"
57     c.setComprasCliente();
58     c.printComprasAnual();
59     return 0;
60 }
```

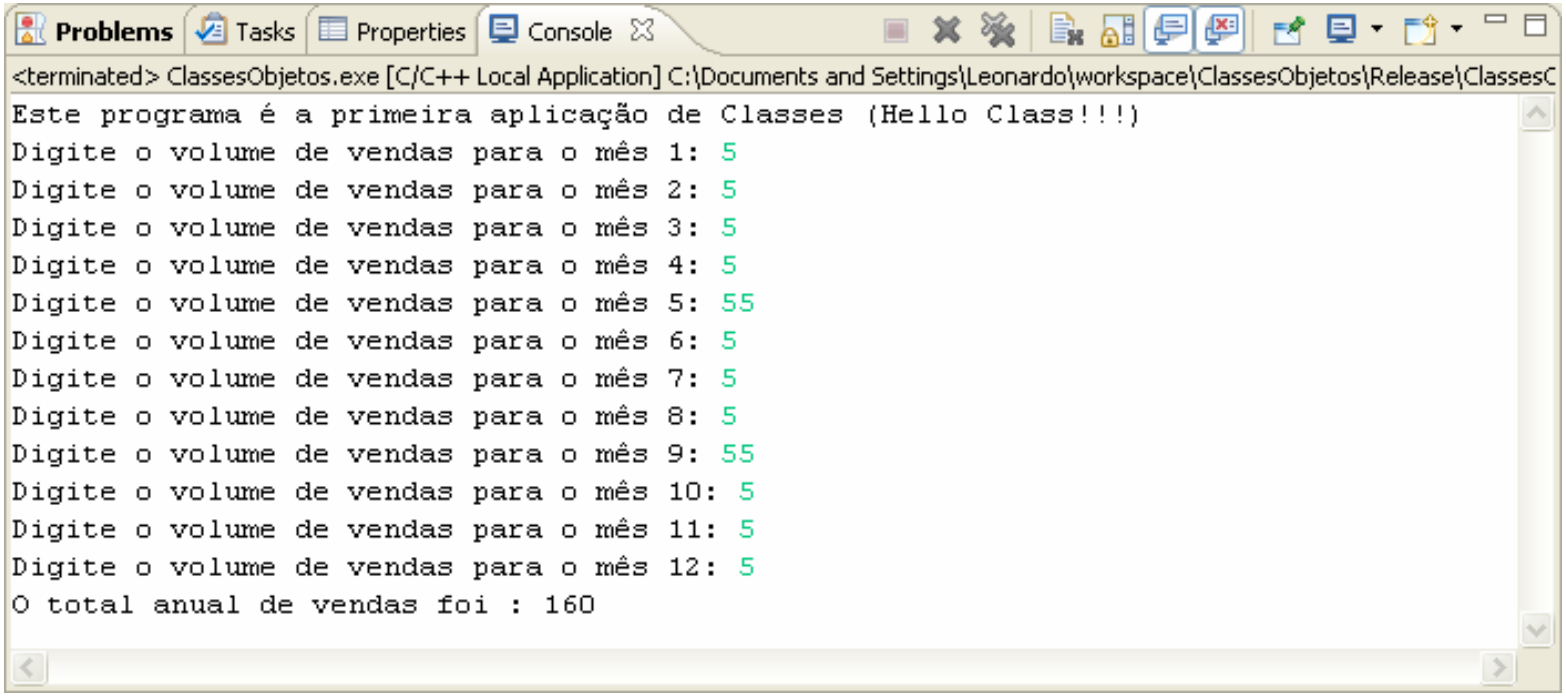
:: Operador binário de resolução de escopo

Criação do objeto c da classe Cliente

Chamada a métodos da classe Cliente

Introdução

- Saída para os programas Anteriores:



Escopo de Classe e Acesso a Membros de Classes

- Acesso aos membros privados:

Escopo de Classe e Acesso a Membros de Classes

- Acesso através de Ponteiros e com Referência:

Cria um Ponteiro para a classe c

Cria um Referência para a classe c

Operador de seleção de membro ponto (.) é combinado com o nome ou com uma referência a um objeto para acessar os membros do objetoCria um Referência para a classe c

Operador de seleção de membro seta (->) é combinado com um ponteiro para um objeto a fim de acessar os membros daquele objeto

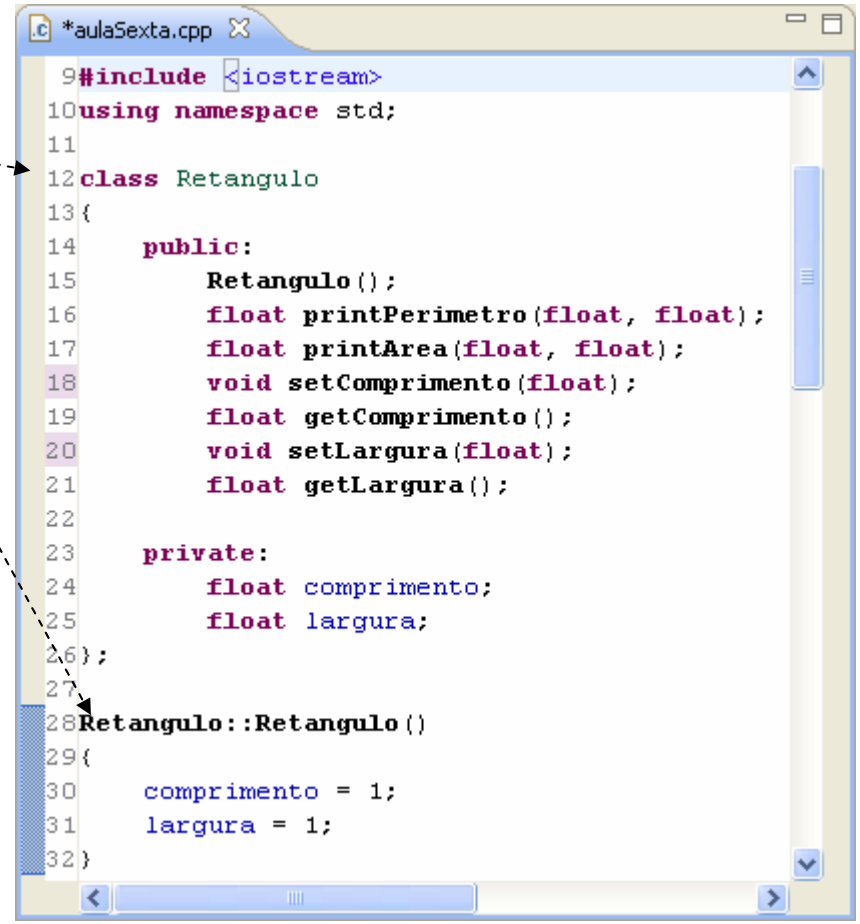
```
11
12 int main() {
13     Cliente c,
14         *cPonteiro = &c,
15         &cReferencia = c;
16     cout << "Este programa é a primeira aplicação de Classes (Hello Class!!!)"
17     cReferencia.setComprasCliente();
18     cPonteiro->printComprasAnual();
19     return 0;
20 }
```

Exercício

- Crie uma classe Retângulo. A classe tem atributos comprimento e largura, cada um com valor default igual a 1.
  - Ela tem funções membro que calculam o perímetro e a área do retângulo.
  - Forneça funções set e get, tanto para o comprimento como para a largura.
  - As funções set devem verificar se o comprimento e a largura são números de ponto flutuante maiores que 0.0 e menores que 20.0

Exercício

- Classe e Construtor sugerido:



Os Métodos get e set

- Como os campos privados de uma classe só podem ser manipulados pelos métodos dessa classe, costumamos ter métodos públicos para permitir aos clientes da classe configurar:
  - set : atribuir valores;
  - get : obter valores;

- Um método set pode – e deve – avaliar cuidadosamente as tentativas de modificar o valor da variável a fim de assegurar que o novo valor é apropriado para esse item de dados;
  - Mês acima de 12; idade e medidas abaixo de 0 (zero);

Os Métodos get e set

- Exemplo de um método get:

```
44float Retangulo::getComprimento()
45{
46    return comprimento;
47}
48
```

- Exemplo de um método set:

```
49void Retangulo::setComprimento(float comp)
50{
51    comprimento = ( (comp>0) ? comp : 0 );
52}
53
```

Separação de Interface e Implementação

- Implementação:

ClassesObjetos.cpp | dadosCliente.h

```
1//============================================================================
2// Name        : ClassesObjetos.cpp
3// Author      : Leonardo
4// Version     :
5// Copyright   : Your copyright notice
6// Description : Hello World in C++, Ansi-style
7//============================================================================
8#include <iostream>
9
10#include "dadosCliente.h"
11
12int main() {
13    Cliente c,
14        *cPonteiro = &c,
15        &cReferencia = c;
16    cout << "Este programa é a primeira aplicação de Classes (Hello Class!!!)"
17    cReferencia.setComprasCliente();
18    cPonteiro->printComprasAnual();
19    return 0;
20}
```

Arquivo cabeçalho para inclusão da classe na Implementação

Separação de Interface e Implementação

- Interface:

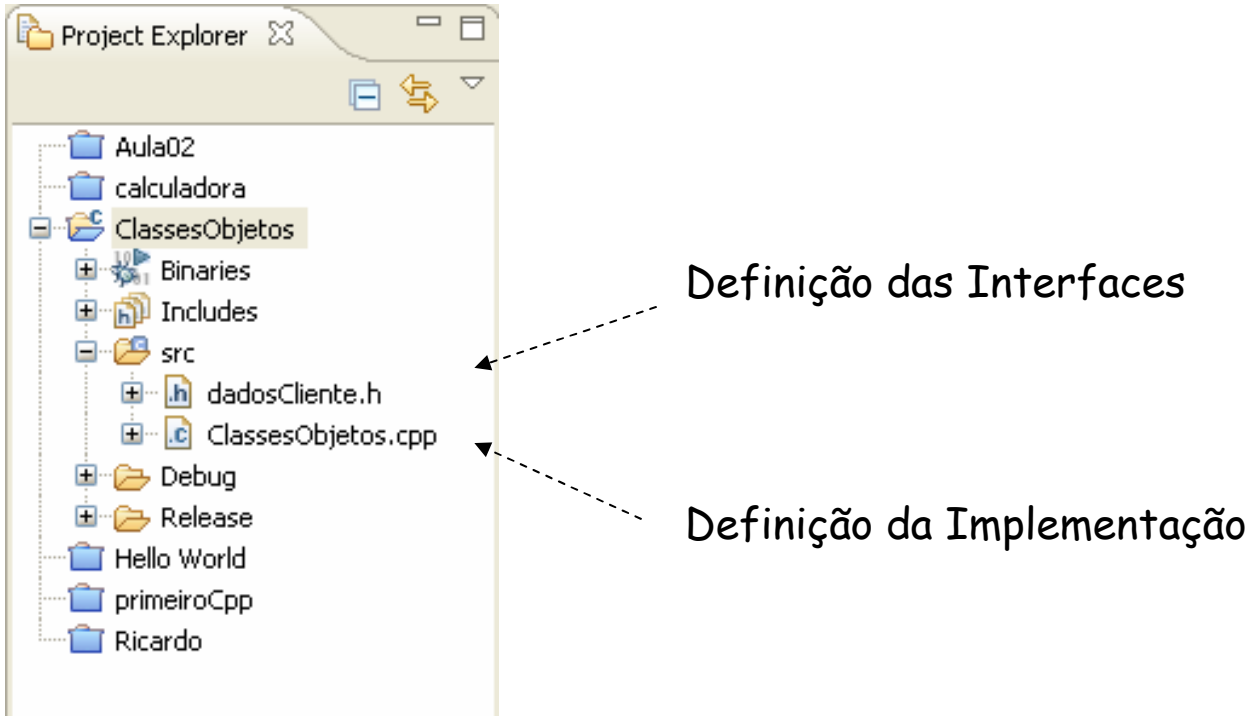
ClassesObjetos.cpp | dadosCliente.h

```
1 #ifndef DADOSCLIENTE_H_
2 #define DADOSCLIENTE_H_
3
4 #include <iostream>
5 using namespace std;
6
7 class Cliente {
8 public:
9     Cliente();
10    void setComprasCliente();
11    void printComprasAnual();
12
13 private:
14    double vendas[12];
15    double totalAno;
16    int codigo;
17 };
18
19 Cliente::Cliente()
20 {
```

Evita a inclusões múltiplas de arquivo de cabeçalho

Separação de Interface e Implementação

- Distribuição do Projeto no Eclipse:



Construtores e Destrutores

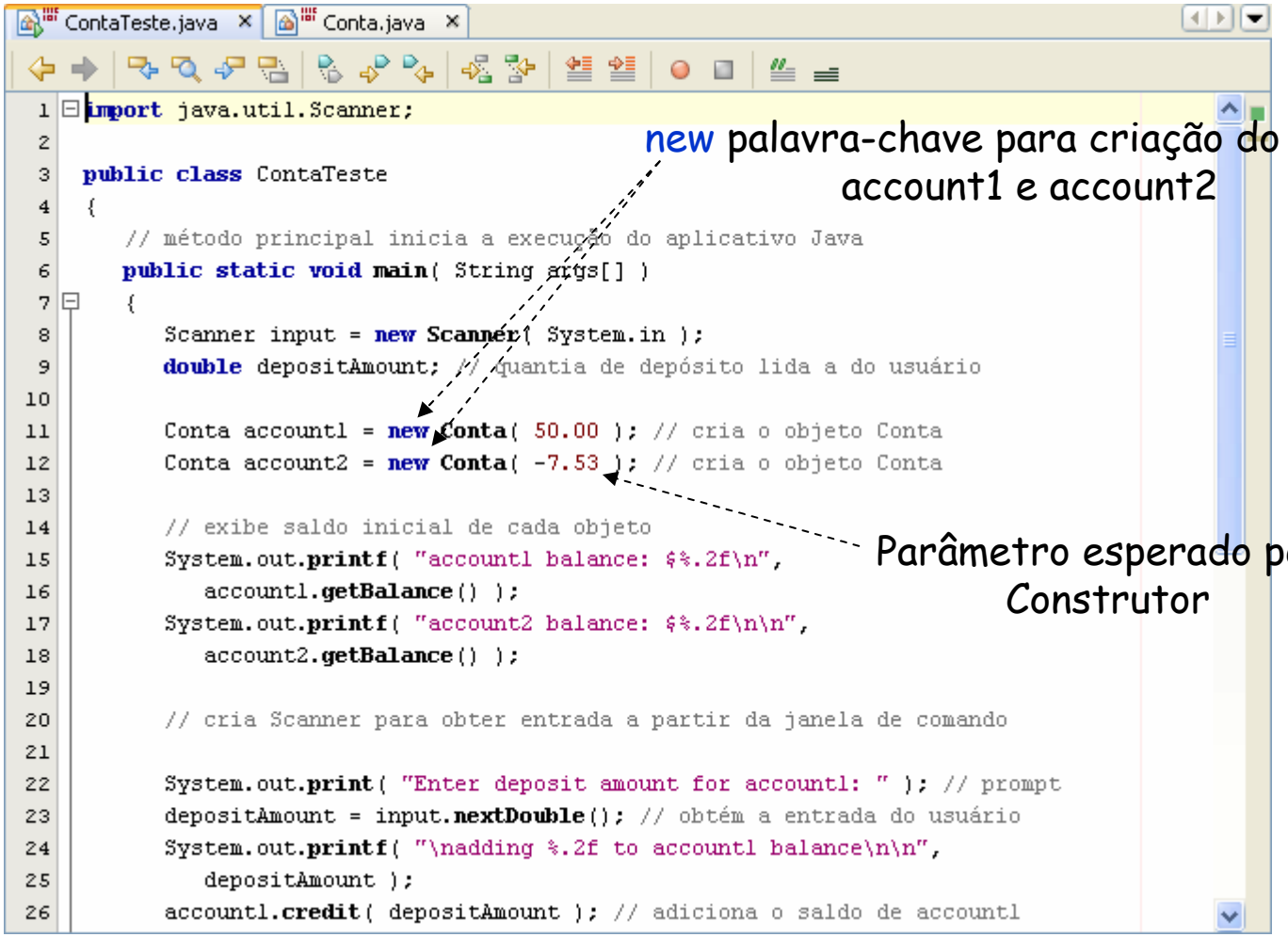
- Construtor:
  - Função membro da classe com o mesmo nome;
  - Invocado automati-camente quando o oobjeto é criado;
  - Subporta Sobrecar-ga

ContaTeste.java | Conta.java

```
1  public class Conta
2  {
3     private double balance; // variável de instância que armazena o saldo
4
5     // construtor
6     public Conta( double initialBalance )
7     {
8        // valida que initialBalance é maior que 0,0;
9        // se não, o saldo é inicializado como o valor padrão 0.0
10       if ( initialBalance > 0.0 )
11          balance = initialBalance;
12    } // fim do construtor Account
13
14    // credita (adiciona) uma quantia à conta
15    public void credit( double amount )
16    {
17       balance = balance + amount; // adiciona quantia ao saldo
18    } // fim do método credit
19
20    // retorna o saldo da conta
21    public double getBalance()
22    {
23       return balance; // fornece o valor de saldo ao método chamador
24    } // fim do método getBalance
25
26 } // fim da classe Conta
```

Palavra-chave precedendo as variáveis da classe

Construtores e Destrutores




```
1  import java.util.Scanner;
2
3  public class ContaTeste
4  {
5     // método principal inicia a execução do aplicativo Java
6     public static void main( String args[] )
7     {
8        Scanner input = new Scanner( System.in );
9        double depositAmount; // quantia de depósito lida a do usuário
10
11       Conta account1 = new Conta( 50.00 ); // cria o objeto Conta
12       Conta account2 = new Conta( -7.53 ); // cria o objeto Conta
13
14       // exibe saldo inicial de cada objeto
15       System.out.printf( "account1 balance: $%.2f\n",
16          account1.getBalance() );
17       System.out.printf( "account2 balance: $%.2f\n\n",
18          account2.getBalance() );
19
20       // cria Scanner para obter entrada a partir da janela de comando
21
22       System.out.print( "Enter deposit amount for account1: " ); // prompt
23       depositAmount = input.nextDouble(); // obtém a entrada do usuário
24       System.out.printf( "\nadding %.2f to account1 balance\n\n",
25          depositAmount );
26       account1.credit( depositAmount ); // adiciona o saldo de account1
```

Construtores e Destrutores

- Fim do Programa:

ContaTeste.java | Conta.java

```
22        System.out.print( "Enter deposit amount for account1: " ); // prompt
23        depositAmount = input.nextDouble(); // obtém a entrada do usuário
24        System.out.printf( "\nadding %.2f to account1 balance\n\n",
25           depositAmount );
26        account1.credit( depositAmount ); // adiciona o saldo de account1
27
28        // exibe os saldos
29        System.out.printf( "account1 balance: $%.2f\n",
30           account1.getBalance() );
31        System.out.printf( "account2 balance: $%.2f\n\n",
32           account2.getBalance() );
33
34        System.out.print( "Enter deposit amount for account2: " ); // prompt
35        depositAmount = input.nextDouble(); // obtém a entrada do usuário
36        System.out.printf( "\nadding %.2f to account2 balance\n\n",
37           depositAmount );
38        account2.credit( depositAmount ); // adiciona ao saldo de account2
39
40        // exibe os saldos
41        System.out.printf( "account1 balance: $%.2f\n",
42           account1.getBalance() );
43        System.out.printf( "account2 balance: $%.2f\n",
44           account2.getBalance() );
45     } // fim de main
46
47  } // fim da classe AccountTest
```

Construtores e Destrutores

- Saída do programa anterior:

```
Saída - Construtor (run)
Compiling 2 source files to C:\Documents and Sett
compile:
run:
account1 balance: $50,00
account2 balance: $0,00

50
Enter deposit amount for account1:
adding 50,00 to account1 balance

account1 balance: $100,00
account2 balance: $0,00

80
Enter deposit amount for account2:
adding 80,00 to account2 balance

account1 balance: $100,00
account2 balance: $80,00
EXECUTADO COM SUCESSO (tempo total: 5 segundos)
```

Construtores e Destrutores

- Destrutor ou Destruidor é um método especial da classe que excluir o objeto criado (no escopo em que ele foi criado);

- O nome do destrutor é formado pelo caracter ~ (til) seguido pelo nome da classe;
  - Não recebe e nem passa parâmetros;

- Geralmente o destrutor é chamado em ordem inversa ao construtor;

Construtores e Destrutores

- Destrutor:

ClassesObjetos.cpp | *dadosCliente.h

```
7  class Cliente {
8  public:
9      Cliente();
10     ~Cliente();
11     void setComprasCliente();
12     void printComprasAnual();
13
14 private:
15     double vendas[12];
16     double totalAno;
17     int codigo;
18 };
19
20 Cliente::Cliente()
21 {
22     for ( int i = 0; i < 12; i++)
23         vendas [i] = 0.0;
24     codigo = 0;
25     totalAno = 0;
26 }
27
28 Cliente::~Cliente()
29 {
30     cout << "Destrutor do Objeto!!!";
31 }
```

Construtores e Destrutores

- Os destrutores de objetos definidos no escopo global e objetos locais `static` são chamados quando a **`main`** termina ou a função **`exit`** é chamada;
  - Não é chamados destruidores para objetos globais e locais `static` se o programa é terminado com a função **`abort`**;

- Os destrutores de objetos locais são chamados quando os objetos deixam o escopo;

Construtores e Destrutores

- Exemplo:
  - Definição da classe, do construtor e do destrutor

ClassesObjetos.cpp | dadosCliente.h

```
4  #include <iostream>
5  using namespace std;
6
7  class Cliente {
8  public:
9      Cliente(int);
10     ~Cliente();
11     void setComprasCliente();
12     void printComprasAnual();
13
14 private:
15     double vendas[12];
16     double totalAno;
17     int codigo;
18 };
19
20 Cliente::Cliente( int num )
21 {
22     for ( int i = 0; i < 12; i++)
23         vendas [i] = 0.0;
24     codigo = num;
25     totalAno = 0;
26     cout << "Construtor do Objeto " << codigo << endl;
27 }
28
29 Cliente::~Cliente()
30 {
31     cout << "Destrutor do Objeto " << codigo << endl;
32 }
```

Construtores e Destrutores

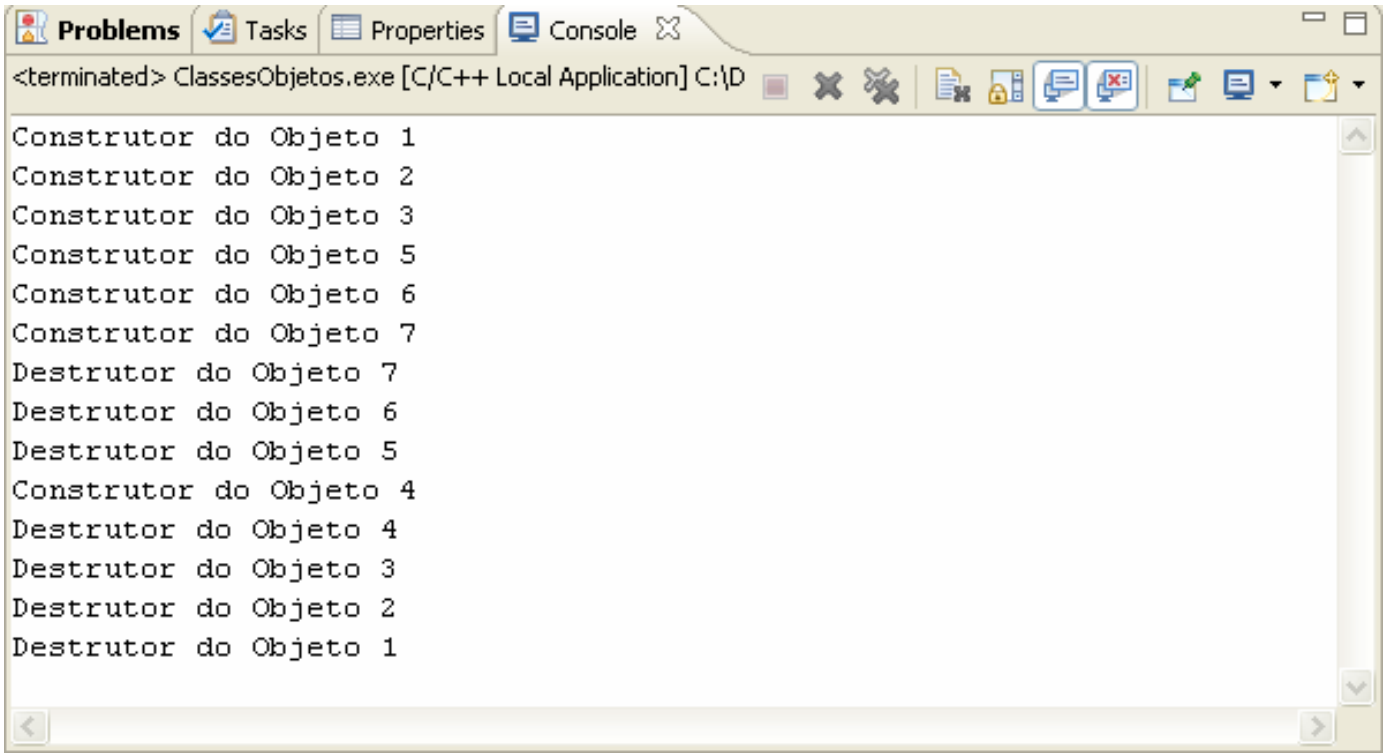
- Exemplo:
  - Definição do main e de uma classe que cria alguns objetos como teste;

*ClassesObjetos.cpp | dadosCliente.h

```
 8 #include <iostream>
 9
10 #include "dadosCliente.h"
11
12 void criaConstrutores();
13
14 Cliente primeiro( 1 );
15
16 int main()
17 {
18     Cliente segundo ( 2 );
19     Cliente terceiro ( 3 );
20     criaConstrutores();
21     Cliente quarto ( 4 );
22     return 0;
23 }
24
25 void criaConstrutores()
26 {
27     Cliente quinto ( 5 );
28     Cliente sexto ( 6 );
29     Cliente setimo ( 7 );
30 }
31
```

Construtores e Destrutores

- Saída do programa anterior:



Construtores e Destrutores

- É possível permitir que objetos sejam criados com diferentes comportamentos através da sobrecarga de construtores, vejamos:

Time2.java * | Time2Test.java *

```
1  public class Time2
2  {
3     private int hour;   // 0 - 23
4     private int minute; // 0 - 59
5     private int second; // 0 - 59
6
7     public Time2()
8     {  this( 0, 0, 0 ); }
9
10    public Time2( int h )
11    {  this( h, 0, 0 ); }
12
13    public Time2( int h, int m )
14    {  this( h, m, 0 ); }
15
16    public Time2( int h, int m, int s )
17    {  setTime( h, m, s ); }
18
19    public void setTime( int h, int m, int s )
20    {
21       setHour( h );   // configura hour
22       setMinute( m ); // configura minute
23       setSecond( s ); // configura second
24    } // fim do método setTime
```

Construtores e Destrutores

- Continuação:

Time2.java * | Time2Test.java *

```
26    public void setHour( int h )
27    {  hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }
28
29    public void setMinute( int m )
30    {  minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }
31
32    public void setSecond( int s )
33    {  second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }
34
35    public int getHour()
36    {  return hour; }
37
38    public int getMinute()
39    {  return minute; }
40
41    public int getSecond()
42    {  return second; }
43
44    public String toUniversalString()
45    {  return String.format( "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
46
47    public String toString()
48    {  return String.format( "%d:%02d:%02d %s",
49          ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
50          getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
51    } // fim do método toString
52 } // fim da classe Time2
```

Construtores e Destrutores

- Main:

```
1  public class Time2Test
2  {
3     public static void main( String args[] )
4     {
5        Time2 t1 = new Time2();             // 00:00:00
6        Time2 t2 = new Time2( 2 );          // 02:00:00
7        Time2 t3 = new Time2( 21, 34 );     // 21:34:00
8        Time2 t4 = new Time2( 12, 25, 42 ); // 12:25:42
9        Time2 t5 = new Time2( 27, 74, 99 ); // 00:00:00
10
11       System.out.println( "Construido com:" );
12       System.out.println( "t1: Todos os argumentos Default" );
13       System.out.printf( "   %s\n", t1.toUniversalString() );
14       System.out.printf( "   %s\n", t1.toString() );
15
16       System.out.println( "t2: hora especificada; minuto e segundo default" );
17       System.out.printf( "   %s\n", t2.toUniversalString() );
18       System.out.printf( "   %s\n", t2.toString() );
19
20       System.out.println( "t3: hora e minuto especificado; segundo default" );
21       System.out.printf( "   %s\n", t3.toUniversalString() );
22       System.out.printf( "   %s\n", t3.toString() );
23
24       System.out.println( "t4: hora, minuto e segundo especificado" );
25       System.out.printf( "   %s\n", t4.toUniversalString() );
26       System.out.printf( "   %s\n", t4.toString() );
27
28       System.out.println( "t5: Todos os dados especificados, INVÁLIDOS!!!!" );
29       System.out.printf( "   %s\n", t5.toUniversalString() );
30       System.out.printf( "   %s\n", t5.toString() );
31    } // fim de main
32 } // fim da classe Time2Test
```

Construtores e Destrutores

- Saída para o programa anterior:

```
Saída - Tempo (run)
run:
Construido com:
t1: Todos os argumentos Default
    00:00:00
    12:00:00 AM
t2: hora especificada; minuto e segundo default
    02:00:00
    2:00:00 AM
t3: hora e minuto especificado; segundo default
    21:34:00
    9:34:00 PM
t4: hora, minuto e segundo especificado
    12:25:42
    12:25:42 PM
t5: Todos os dados especificados, INVÁLIDOS!!!!
    00:00:00
    12:00:00 AM
EXECUTADO COM SUCESSO (tempo total: 0 segundos)
```

Uso do ponteiro this

- Em Java:
  - Cada objeto pode acessar uma referência com a palavra-chave this;
  - Quando um método não-static é chamado por um objeto particular, o corpo do método utiliza implicitamente a palavra-chave this para referenciar as variáveis de instância do objeto e outro métodos.

- Em C++:
  - Todo objeto tem acesso ao seu próprio endereço através de um ponteiro this;

Objetos const e funções membro const

- A palavra-chave const pode ser usada para especificar que um objeto não é modificável:
  - Dessa forma, qualquer tentativa de modificar o objeto deve ser considerada um erro de sintaxe;

```
4 class Time{
5 public:
6     Time( int = 0, int = 0, int = 0 );
7     void setTime( int, int, int );
8     void setHour( int );
9     void setMinute ( int );
10    void setSecond ( int );
11
12    int getHour() const;
13    int getMinute() const;
14    int getSecond() const;
15
16    void printMilitary() const;
17    void printStandard();
18
19 private:
20    int hour;
21    int minute;
22    int second;
23 };
```

Objetos const e funções membro const

- O compilador só permite chamadas de função membro para objetos `const` apenas se essa função também for `const`;
- As funções declaradas `const` não podem modificar o objeto;
- Uma função é especificada `const` tanto em seu protótipo como em sua definição;

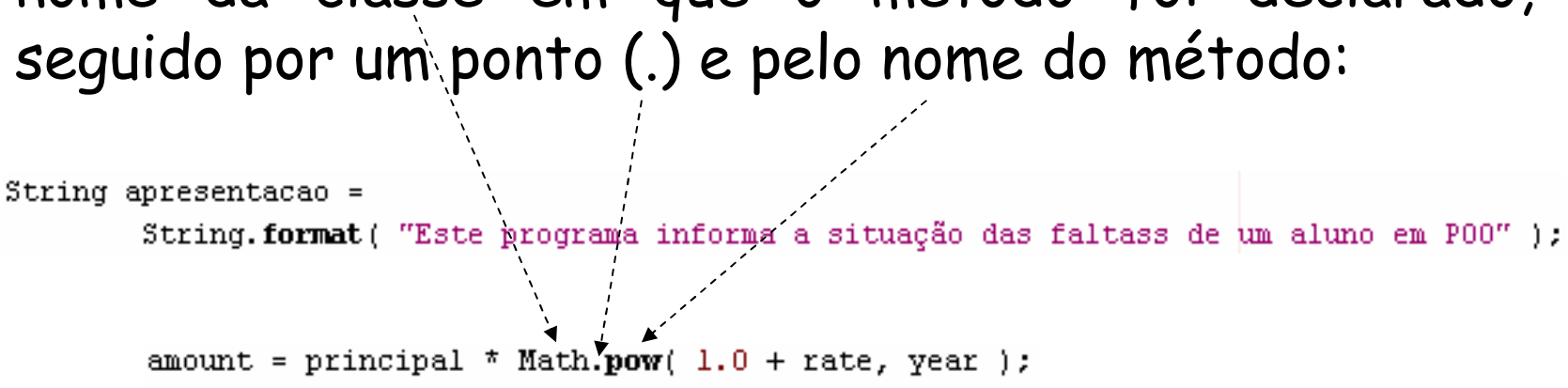
Métodos static e campos static

- Métodos static realizam uma tarefa que não depende do conteúdo de nenhum objeto;
- Esses métodos são usados, normalmente, para realizar uma tarefa comum;
- Para declarar um método como `static` no Java, coloque a palavra-chave `static` antes do tipo de retorno;

```
public static void main( String args[] )
{
```

Métodos static e campos static

- Para chamar um método `static` basta especificar o nome da classe em que o método foi declarado, seguido por um ponto (.) e pelo nome do método:

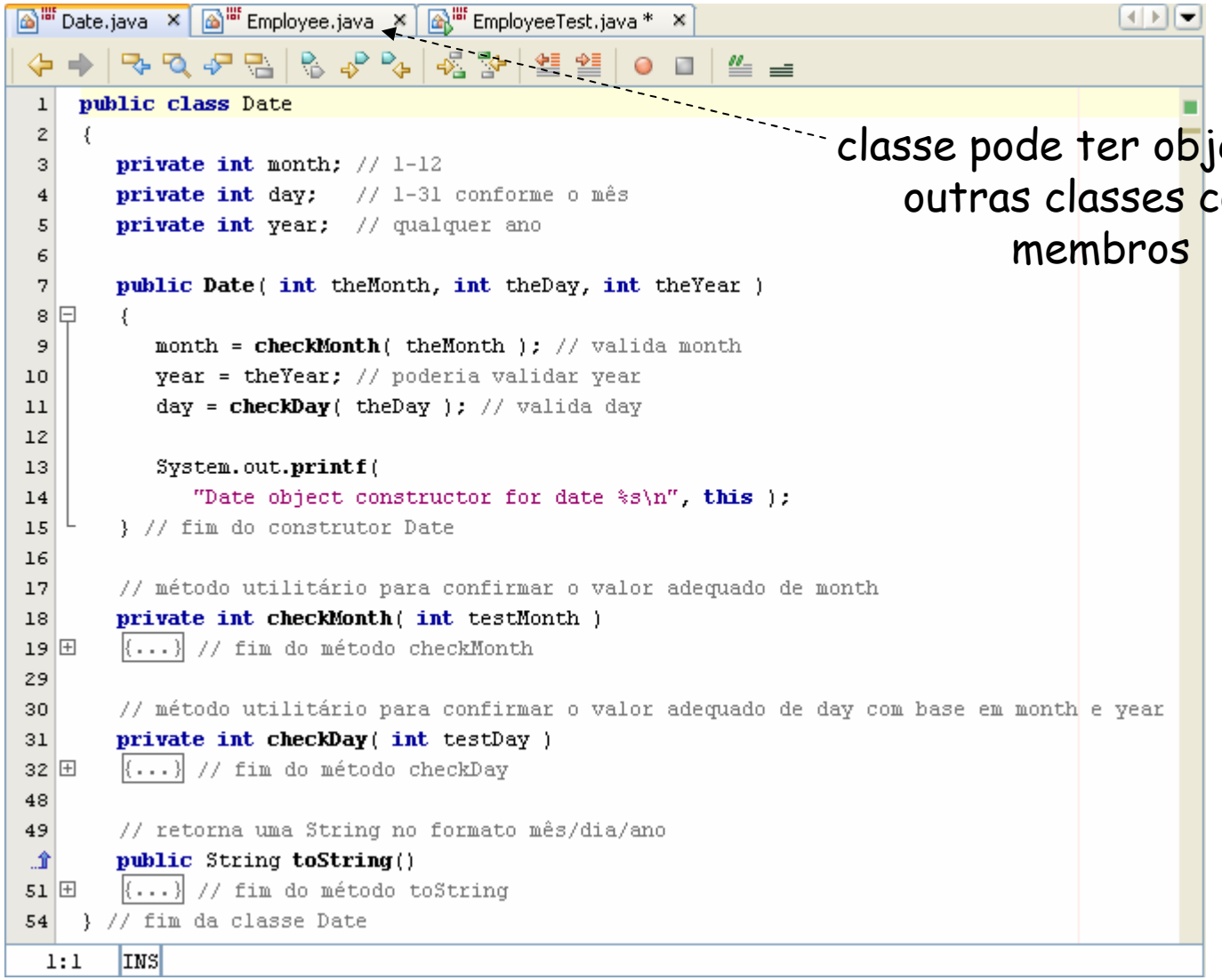


```
String apresentacao =
        String.format( "Este programa informa a situação das faltass de um aluno em POO" );


amount = principal * Math.pow( 1.0 + rate, year );
```

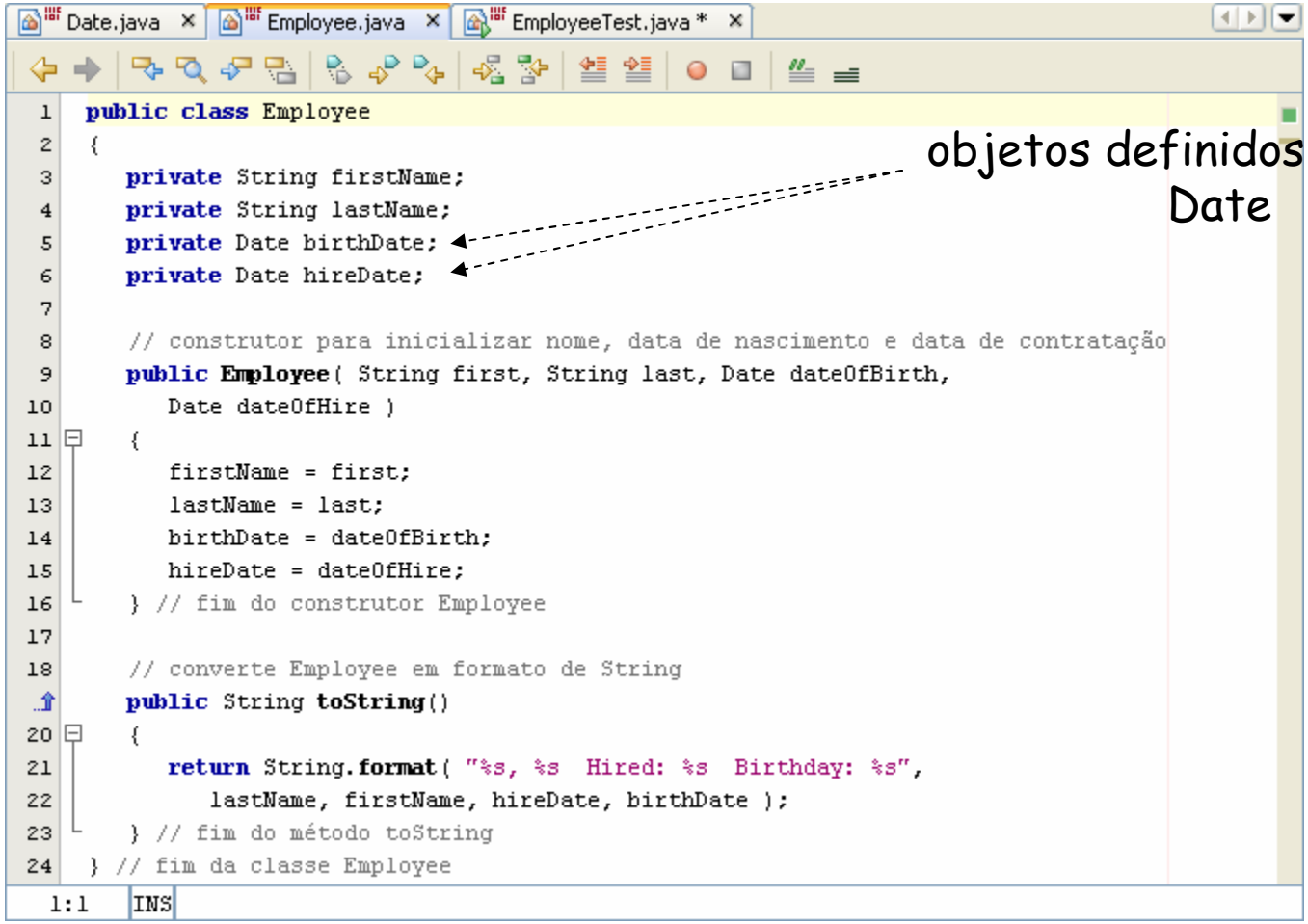
- Em Java, por que o método main é declarado `static`?
  - Porque possibilita à JVM invocar o método main sem criar um objeto para a classe.

Composição



classe pode ter objetos de outras classes como membros

Composição


Date.java
Employee.java
EmployeeTest.java *
objetos definidos na classe Date
1 public class Employee
2 {
3 private String firstName;
4 private String lastName;
5 private Date birthDate;
6 private Date hireDate;
7
8 // construtor para inicializar nome, data de nascimento e data de contratação
9 public Employee(String first, String last, Date dateOfBirth,
10 Date dateOfHire)
11 {
12 firstName = first;
13 lastName = last;
14 birthDate = dateOfBirth;
15 hireDate = dateOfHire;
16 } // fim do construtor Employee
17
18 // converte Employee em formato de String
public String toString()
20 {
21 return String.format("%s, %s Hired: %s Birthday: %s",
22 lastName, firstName, hireDate, birthDate);
23 } // fim do método toString
24 } // fim da classe Employee
1:1
INS

Composição

Date.java | Employee.java | EmployeeTest.java

```
1  public class EmployeeTest
2  {
3     public static void main( String args[] )
4     {
5        Date birth = new Date( 7, 24, 1949 );
6        Date hire = new Date( 3, 12, 1988 );
7        Employee employee = new Employee( "Bob", "Blue", birth, hire );
8
9        System.out.println( employee );
10    } // fim de main
11 } // fim da classe EmployeeTest
```

1:1 INS

Saída - Aula04 (run)

```
init:
deps-jar:
Compiling 3 source files to C:\Documents and Sett
Copied 1 empty directory to 1 empty directory und
compile:
run:
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Blue, Bob  Hired: 3/12/1988  Birthday: 7/24/1949
EXECUTADO COM SUCESSO (tempo total: 0 segundos)
```

Composição

- Em C++, a composição tem a mesma semântica que no Java, a diferença é sintática, vejamos:

```
16class Date{
17public:
18     Date ( int = 1, int = 1, int = 1900 );
19     void print() const;
20     ~Date();
21private:
22     int month;
23     int day;
24     int year;
25     int checkDay( int );
26};
```

```
 4class Employee{
 5public:
 6     Employee( char *, char *, int, int, int, int, int, int );
 7     void print() const;
 8     ~Employee();
 9private:
10     char fristName[ 25 ];
11     char lastName[ 25 ];
12     const Date birthDate;
13     const Date hireDate;
14};
```

```
29Employee::Employee ( char *fname, char *lname,
30                     int bmonth, int bday, int byear,
31                     int hmonth, int hday, int hyear )
32     :birthDate( bmonth, bday, byear ),
33      hireDate( hmonth, hday, hyear )
34
```

Os dois pontos (:) no cabeçalho separa os inicializadores de membros da lista de parâmetros

Composição

- No exemplo anterior os argumentos `bmonth`, `bday` e `byear` são passados para o construtor o objeto `birthDate`;

```
29 Employee::Employee ( char *fname, char *lname,
30                      int bmonth, int bday, int byear,
31                      int hmonth, int hday, int hyear )
32     :birthDate( bmonth, bday, byear ),
33      hireDate( hmonth, hday, hyear )
34
```

- Os argumentos `hmonth`, `hday` e `hyear` são passados para o construtor do objeto `hireDate`.

Funções `friend` e classes `friend`

- Uma função `friend` de uma classe é definida fora do escopo daquela classe, mas ainda tem o direito de acessar membros `private` da classe;
- Uma função ou uma classe inteira podem ser declaradas como `friend` de outra classe;
- O uso de `friends` freqüentemente é apropriado quando uma função membro não puder ser usada para certas operações;

Funções friend e classes friend

- Para declarar uma função como um `friend` de uma classe, preceda o protótipo da função na definição da classe com a palavra-chave `friend`;

```
4class Count{
5    friend void setX ( Count &, int ); //Declaração de friend
6public:
7    Count() { x = 0; }
```

- Uma boa prática de programação é colocar todas as declarações `friend` no início da classe, logo depois do cabeçalho da classe;

Funções friend e classes friend

Aula04.cpp

```
1#include <iostream>
2using namespace std;
3
4class Contador{
5    friend void setNumero( Contador &, int );
6public:
7    Contador() { numero = 0; }
8    void print() const { cout << numero << endl; }
9private:
10    int numero;
11};
12
13void setNumero ( Contador &cont, int num )
14{
15    cont.numero = num;
16}
17
18int main() {
19    Contador c;
20    cout << "Contador.numero após instanciação: ";
21    c.print();
22    cout << "Contador.numero após chamada à função friend setNumero: ";
23    setNumero ( c, 8 );
24    c.print();
25    return 0;
26}
```

Método friend acessando um membro privado

Alocação dinâmica de memória

- Para criar um objeto usando o C puro teríamos:

```
4c *Contador;
5Contador = malloc( sizeof( c ) );
6
```

- Os operadores `new` e `delete` oferecem um meio mais agradável de executar a alocação dinâmica de memória do que `malloc` e `free`;

```
18int main() {
19    Contador *c;
20    c = new Contador;
```

Classes Proxy

- Fornece aos cliente da classe acesso apenas à interface `public`;
- Permite que os clientes usem serviços da classe sem lhes dar acesso aos detalhes de implementação da classe;
- Vejamos a implementação de uma classe proxy;

Classes Proxy

interface.h | implementacao.cpp

```
 4 class Implementation{
 5 public:
 6     Implementation( int v ) { valor = v; }
 7     void setValor( int v ) { valor = v; }
 8     int getValor() const { return valor; }
 9 private:
10         int valor;
11 };
12 class Interface{
13 public:
14     Interface( int );
15     void setValor( int );
16     int getValor() const;
17     ~Interface();
18 private:
19     Implementation *ptr;
20 };
21
22 Interface::Interface( int v )
23     :ptr ( new Implementation ( v ) ) { }
24
25 void Interface::setValor ( int v ) { ptr->setValor( v ); }
26
27 int Interface::getValor() const { return ptr->getValor(); }
28
29 Interface::~Interface() { delete ptr; }
30
```

Definição da classe cujo dados `private` queremos esconder

Classe `proxy` com uma interface `public` idêntica à da classe `Implementation`

Classes Proxy

interface.h | implementacao.cpp

```
1 #include <iostream>
2 using namespace std;
3 #include "interface.h"
4
5 int main()
6 {
7     Interface i( 5 );
8     cout << "Interface contém: " << i.getValor()
9          << " antes de setValor" << endl;
10    i.setValor( 10 );
11    cout << "Interface contém: " << i.getValor()
12         << " após setValor" << endl;
13    return 0;
14 }
```

Problems | Tasks | Properties | Console

<terminated> Aula04.exe [C/C++ Local Application] C:\[

```
Interface contém: 5 antes de setValor
Interface contém: 10 após setValor
```

Bibliografia

- 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.