Polimorfismo

Sumário

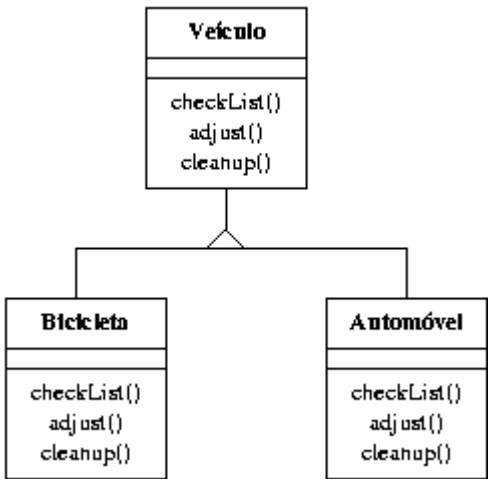
- Introdução;
- Polimorfismo;
- Polimorfismo – Java;
  - Métodos Abstratos – Java
  - Classes Abstratas – Java
  - Exercício - Java
- Polimorfismo – C++
  - Classe Abstrata – C++;
  - Funções Virtuais Puras – C++
  - Classe-Base Virtual – C++
  - Classes Amigas – C++
- Bibliografia;

Introdução

- A característica de chamar métodos de um objeto sem especificar o tipo exato dele é conhecido como polimorfismo;
- A palavra polimorfismo significa "assumir várias formas".
- O polimorfismo permite escrever programas que processam objetos que compartilham a mesma superclasse em uma hierarquia de classes como se todas fossem objetos da superclasse;

Polimorfismo

- Exemplo:
  - Considere uma classe `veículo` com duas classes derivadas, `Automóvel` e `Bicicleta`.



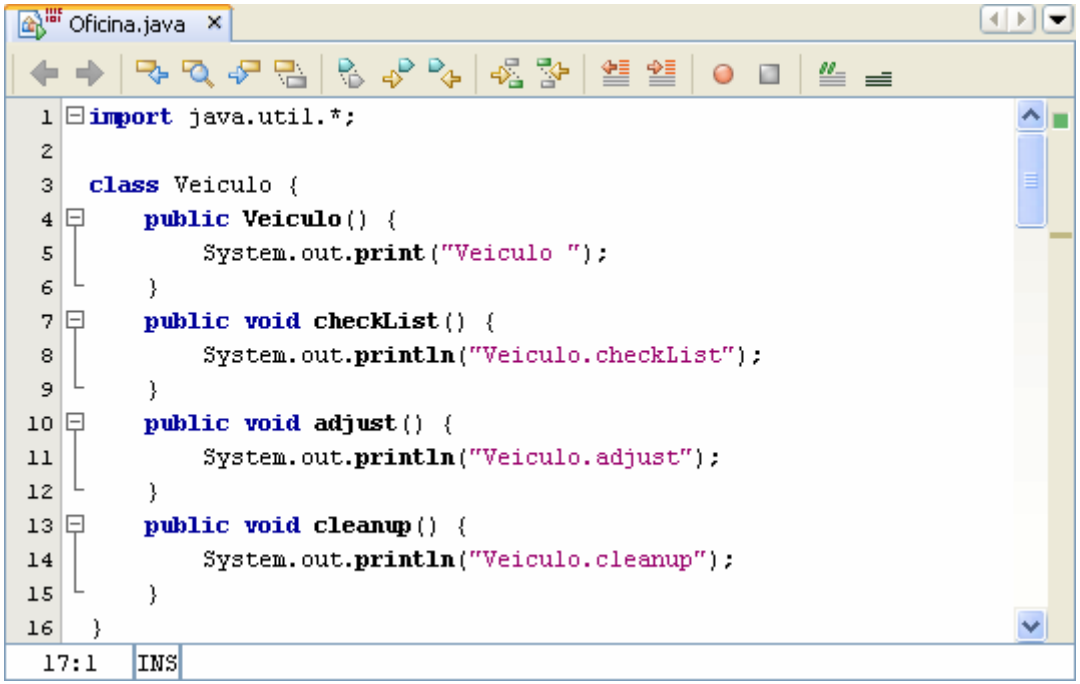
  - Essas classes têm três métodos, definidos para veículos de forma geral e redefinidos mais especificamente para automóveis e bicicletas;

Polimorfismo

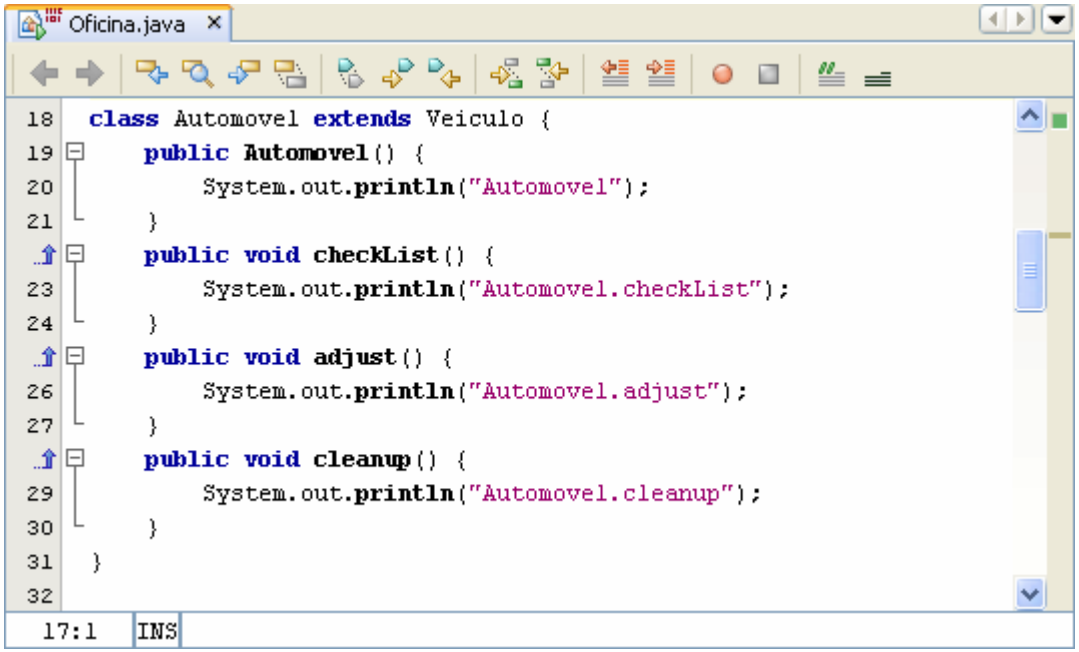
- As funções são:
  - `checkList()`, para verificar o que precisa ser analisado no veículo;
  - `adjust()`, para realizar os reparos e a manutenção necessária;
  - `cleanup()`, para realizar procedimentos de limpeza do veículo.

- A aplicação Oficina define um objeto que recebe objetos da classe Veículo.
  - Para cada veículo recebido, a oficina executa na seqüência os três métodos da classe Veículo.

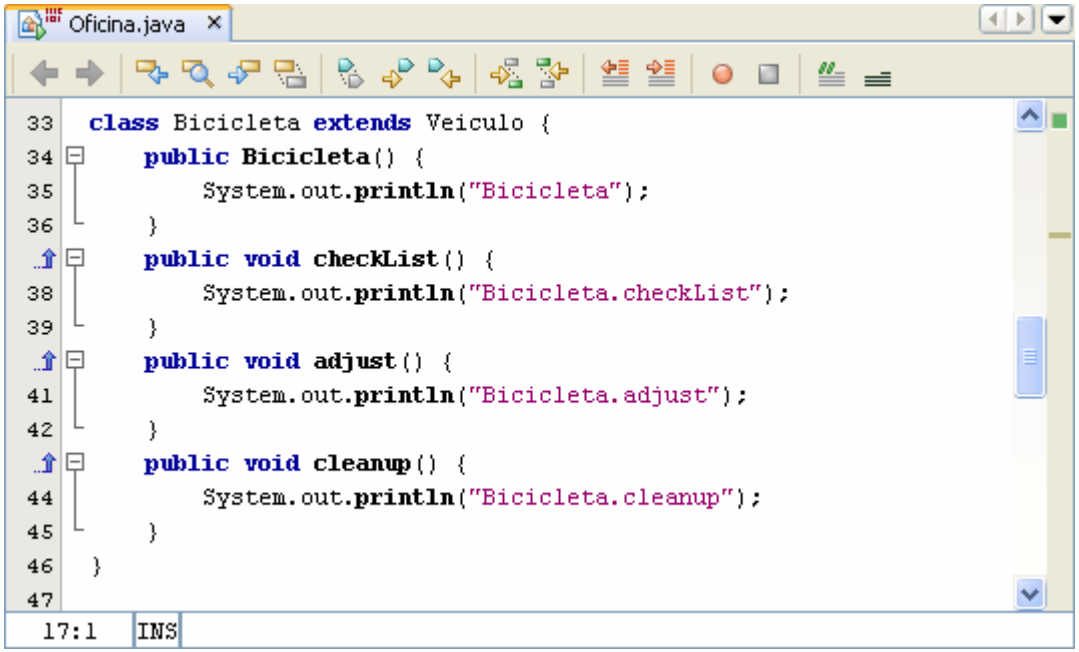
Polimorfismo - Java



Polimorfismo - Java



Polimorfismo - Java



Polimorfismo - Java

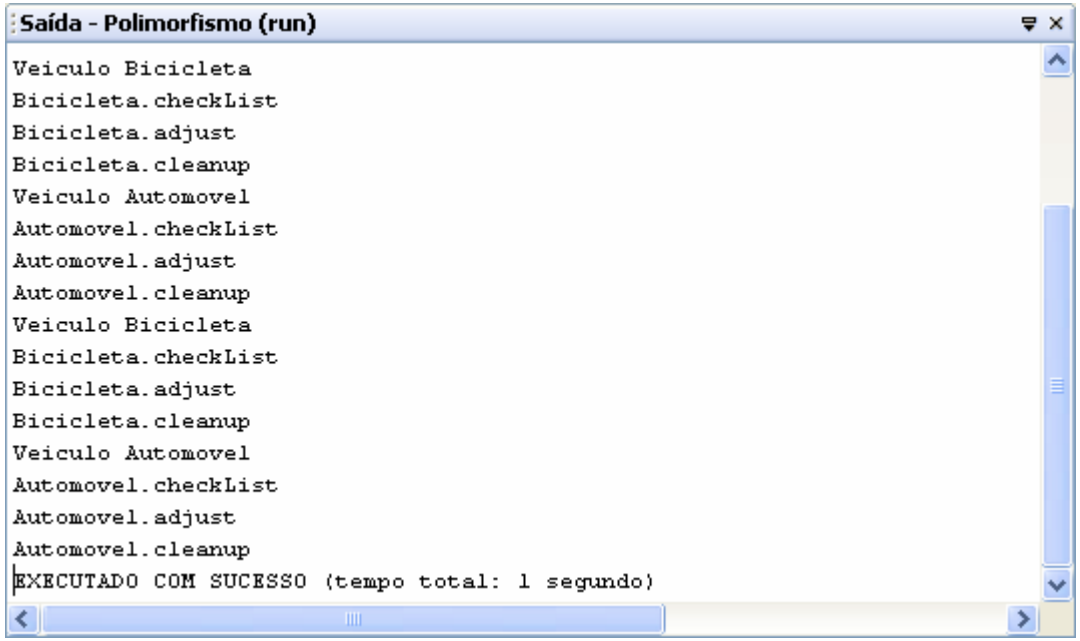
Não há como saber no momento da programação se a Oficina estará recebendo um automóvel ou uma bicicleta -- assim, o momento de decisão sobre qual método será aplicado só ocorrerá durante a execução do programa.

Oficina.java

```
49  public class Oficina {
50      Random r = new Random();
51
52      public Veiculo proximo() {
53          Veiculo v;
54          int code = r.nextInt();
55          if (code%2 == 0)
56              v = new Automovel();
57          else
58              v = new Bicicleta();
59
60          return v;
61      }
62
63      public void manter(Veiculo v) {
64          v.checkList();
65          v.adjust();
66          v.cleanup();
67      }
68
69      public static void main(String[] args) {
70          Oficina o = new Oficina();
71          Veiculo v;
72
73          for (int i=0; i<4; ++i) {
74              v = o.proximo();
75              o.manter(v);
76          }
77      }
78  }
```

17:1 INS

Polimorfismo - Java



Polimorfismo

- Considerações:
  - **Upcasting**
    - O método Oficina.proximo() realiza uma atribuição de um objeto Automóvel à variável (referência para objeto Veiculo) v quando o valor do número aleatório gerado é par (o resto da divisão inteira por 2 é igual a 0). Essa atribuição de um objeto de uma classe mais especializada para uma referência de uma classe ancestral é denominada **upcast**. Esse mesmo tipo de atribuição é realizado de Bicicleta para veículo quando o número aleatório gerado é ímpar.

Polimorfismo

- Considerações:
  - **Uso de métodos abstratos**
    - Apesar de métodos da classe Veículo terem sido definidos, estes nunca são invocados nesse exemplo. (Se fossem, algo estaria errado.) Isso ilustra uma situação onde **métodos abstratos** poderiam ser utilizados, pois a definição do corpo desses métodos é de fato irrelevante. Ainda mais, se uma classe como essa só contém métodos abstratos, ela poderia ser implementada como uma **interface** Java.

Classe Abstrata - Java

- Conforme vimos em herança, uma classe abstrata é útil quando não se quer instanciar objetos.

- As superclasses abstratas são utilizadas para que outras classes possam herdar e assim compartilhar um projeto comum.

- Em java, cria uma classe abstrata declarando-a com a palavra-cahve `abstract`:

```
public abstract class Quadrilatero{
```

Métodos Abstratos - Java

- Uma classe abstrata normalmente contém um ou mais métodos abstratos;
- Um método abstrato é um com a palavra-chave abstract na sua declaração, como em:

```
public abstract float calcularArea();
```

- Métodos abstratos não fornecem implementações. Uma classe que contém métodos abstratos deve ser declarada como classe abstrata esmo que contenha métodos concretos (não abstratos);

Exercício - Java

- Exercício:
  - **1)** Implemente, em Java, uma classe abstrata de nome *Quadrilatero* onde são declarados dois métodos abstratos:
    - *float calcularArea();*
    - *float calcularPerimetro();*
  - **2)** Crie, como subclasse de *Quadrilatero*, uma classe de nome *Retangulo* cujas instâncias são caracterizadas pelos atributos *lado* e *altura* ambos do tipo *float*. Implemente na classe *Retangulo* os métodos herdados de *Quadrilatero* e outros que ache necessários.

Exercício - Java

- Exercício:
  - **3)** *Crie, como subclasse de Quadrilatero, uma classe de nome Circulo cujas instâncias são caracterizadas pelo atributo raio do tipo float. Implemente na classe Circulo os métodos herdados de Quadrilatero e outros que ache necessários. Nota: poderá aceder ao valor de Pi fazendo Math.Pi.*
  - ***4)*** *Crie, como subclasse de Retangulo, uma classe de nome Quadrado cujas instâncias são caracterizadas por terem os atributos lado e altura com o mesmo valor.*

Exercício - Java

- Exercício:
  - **5)** Elabore um programa de teste onde é declarado um array, de dimensão 5, do tipo estático *Quadrilatero*. Nesse array devem ser guardadas instâncias de *Rectangulo*, *Circulo* e *Quadrado* seguindo uma ordem aleatória. Nota: para gerar números aleatórios crie primeiro uma instância da classe *Random* (presente na biblioteca *java.util*) e para extrair um inteiro entre 0 e *n* efectue a evocação *nextInt(n)*. Depois implemente um ciclo que percorra o array evocando, relativamente a cada um dos objetos guardados, os métodos *calcularArea* e *calcularPerimetro*.

Exercício - Java

- Classe Quadrilátero:

```
1  import java.util.*;
2  import java.util.Scanner;
3
4  abstract class Quadrilatero {
5      String nome;
6      protected float largura;
7      protected float altura;
8      protected double raio;
9
10     public Quadrilatero() {
11         System.out.print("Construtor de Quadrilátero");
12     }
13     public abstract double calcularArea();
14     public abstract double calcularPerimetro();
15     public void setAltura(float alt)
16     {   altura = alt;    }
17     public float getAltura()
18     {   return altura;   }
19     public void setLargura(float larg)
20     {   largura = larg;  }
21     public float getLargura()
22     {   return largura;  }
```

Exercício - Java

- Continuação da Classe Quadrilátero:

```
23      public double getRaio()
24      {   return raio;     }
25      public void setRaio(double r)
26      {   raio = r;        }
27      public void setNome(String n)
28      {   nome = n;        }
29      public String getNome()
30      {   return nome;     }
31      public void setDados()
32      {
33          Scanner entrada = new Scanner( System.in );
34          System.out.println("Digite a altura do Quadrilatero: ");
35          altura = entrada.nextFloat();
36          System.out.println("Digite a largura do Quadrilatero: ");
37          largura = entrada.nextFloat();
38          System.out.println("Digite o raio do Círculo: ");
39          raio = entrada.nextDouble();
40      }
41  }
42
```

Exercício - Java

- Classe Retangulo:

```
43  class Retangulo extends Quadrilatero{
44      public Retangulo()
45      {
46          System.out.println("Construtor de Retângulo");
47          setNome("Retângulo");
48      }
        public double calcularArea()
50      {
51          return (getLargura()*getLargura());
52      }
        public double calcularPerimetro()
54      {
55          return (2*getLargura())+(2*getAltura());
56      }
57  }
58
```

Exercício - Java

- Classe Circulo:

```
59  class Circulo extends Quadrilatero {
60      private double pi = 3.14159;
61      public Circulo() {
62          System.out.println("Construtor de Circulo");
63          setNome("Círculo");
64      }
        public double calcularArea()
66      {
67          return (pi*(getRaio()*getRaio()));
68      }
        public double calcularPerimetro()
70      {
71          return (2*pi*getRaio());
72      }
73  }
74
```

Exercício - Java

- Classe Quadrado:

```
75  class Quadrado extends Retangulo {
76      public Quadrado() {
77          System.out.println("Construtor de Quadrado");
78          setNome("Quadrado");
79          super.setLargura(super.getAltura());
80      }
        public double calcularArea()
82      {
83          return (getLargura() * getLargura());
84      }
        public double calcularPerimetro()
86      {
87          return (4*getLargura());
88      }
89  }
90
```

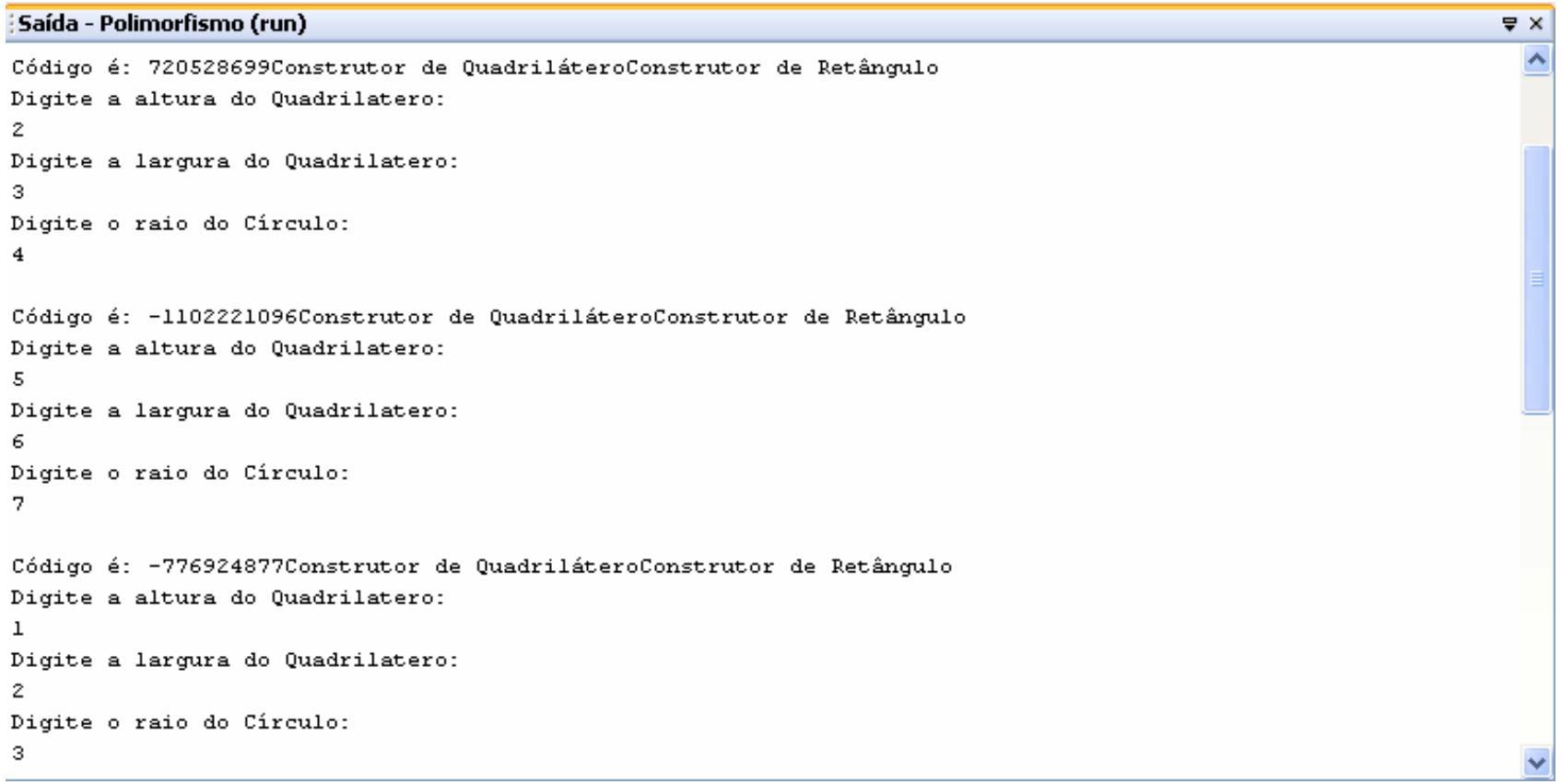
Exercício - Java

- Programa de Teste:

```
91  public class TesteQuadrilatero {
92      public static void main(String[] args) {
93          Quadrilatero q[] = new Quadrilatero[5];
94          Random r = new Random();
95
96          for(int i=0; i<5; i++, r = new Random())
97          {
98              int codigo = r.nextInt();
99              System.out.printf("\nCódigo é: %d", codigo);
100
101             if (codigo%3 == 0)
102                 q[i] = new Retangulo();
103             if ( (codigo%3 == 1)||(codigo%3==-1) )
104                 q[i] = new Quadrado();
105             if ( (codigo%3 == 2)||(codigo%3==-2) )
106                 q[i] = new Circulo();
107             q[i].setDados();
108         }
109
110         for(int i=0; i<5; i++)
111         {
112             System.out.printf("\nA área do %s é: %f", q[i].getNome(), q[i].calcularArea());
113             System.out.printf("\nO Perímetro do %s é: %f", q[i].getNome(), q[i].calcularPerimetro());
114         }
115     }
116 }
```

Exercício - Java

- Saída no Console do programa anterior:



Exercício - Java

```
Saída - Polimorfismo (run)
Código é: -338696095Construtor de QuadriláteroConstrutor de Retângulo
Construtor de Quadrado
Digite a altura do Quadrilatero:
9
Digite a largura do Quadrilatero:
8
Digite o raio do Círculo:
7

Código é: 1390259292Construtor de QuadriláteroConstrutor de Retângulo
Digite a altura do Quadrilatero:
4
Digite a largura do Quadrilatero:
5
Digite o raio do Círculo:
6

A área do Retângulo é: 9,000000
O Perímetro do Retângulo é: 10,000000
A área do Retângulo é: 36,000000
O Perímetro do Retângulo é: 22,000000
A área do Retângulo é: 4,000000
O Perímetro do Retângulo é: 6,000000
A área do Quadrado é: 64,000000
O Perímetro do Quadrado é: 32,000000
A área do Retângulo é: 25,000000
O Perímetro do Retângulo é: 18,000000
EXECUTADO COM SUCESSO (tempo total: 28 segundos)
```

Exercício - Java

- Considerações:
  - Entenda porque imprime a mensagem dentro do construtor Retangulo na criação de um objeto Quadrado;
  - Refaça o exemplo excluindo o "quadrilátero" círculo e inserindo os quadriláteros Trapézio e Losango;
  - Retire as leituras desnecessárias, por exemplo, altura e raio para Quadrado;
  - Imprima os dados personalizados na apresentação dos cálculos, por exemplo, "A área do **retângulo** de **altura x** e **largura y** é: z";

Classe Abstrata – C++

- Em C++, uma classe abstrata é uma classe que define uma função virtual pura;
- Mas o que é uma função virtual?
  - É uma função que impede a classe declarar qualquer objeto dela;
- Em C++, o polimorfismo é possível através de funções virtuais, vejamos como:

Polimorfismo – C++

Polimorfismo.cpp

```
1#include <iostream>
2using namespace std;
3
4class Base
5{
6public:
7    virtual void Print()
8    {
9        cout << "Base" << endl;
10   }
11};
12
13class DerivadaO : public Base
14{
15public:
16   void Print()
17   {
18       cout << "Derivana O" << endl;
19   }
20};
```

`virtual` palavra-chave incluída no início da declaração do método

Classe derivada que implementa o método `Print()` novamente

Polimorfismo - C++

Polimorfismo.cpp

```
22 class Derivada1 : public Base
23 {
24 public:
25     void Print()
26     {
27         cout << "Derivana 1" << endl;
28     }
29 };
30
31 class Derivada2 : public Base
32 {
33 public:
34     void Print()
35     {
36         cout << "Derivana 2" << endl;
37     }
38 };
39
```

Demais classes derivadas que também implementam o método `Print()` novamente

Polimorfismo – C++

Polimorfismo.cpp

```
39
40int main()
41{
42    Base *p[3];
43
44    Derivada0 d0;
45    Derivada1 d1;
46    Derivada2 d2;
47
48    p[0] = &d0;
49    p[1] = &d1;
50    p[2] = &d2;
51
52    for (int i=0; i<3; i++)
53        p[i] -> Print();
54    return 0;
55}
56
```

Matriz de ponteiros para a classe Base

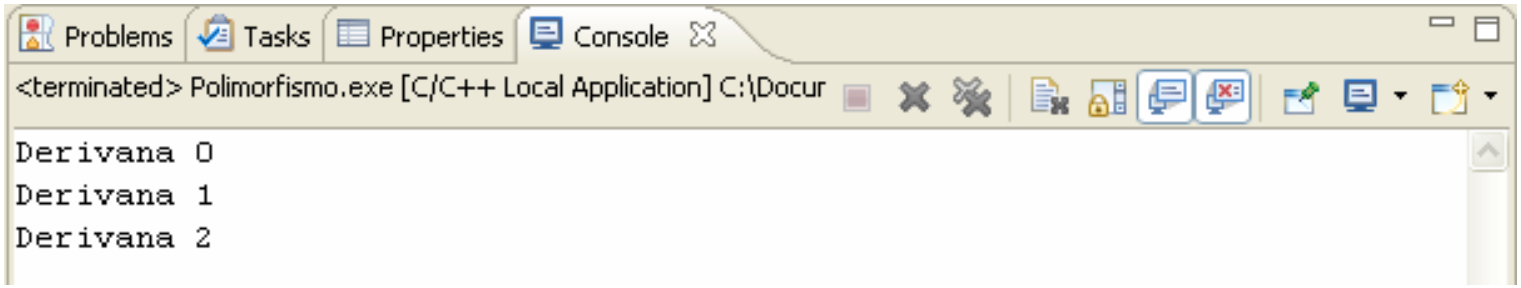
Objetos das classes Derivadas

Ponteiros recebendo os endereços de cada um dos elementos objetos

Execução da função Print() dependendo do conteúdo de p[i] e não do seu tipo

Polimorfismo – C++

- Saída no console do programa anterior:

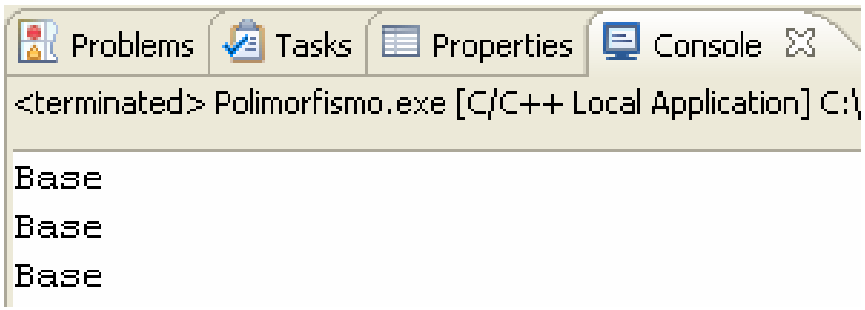


Polimorfismo - C++

- Caso retirássemos a palavra-chave `virtual` da função `Print()` o compilador iria ignorar o conteúdo de `p[i]` e usa o tipo do ponteiro para identificar o método a ser chamado, vejamos:

```
4 class Base
5 {
6 public:
7     void Print()
8     {
9         cout << "Base" << endl;
10    }
11 };
```

```
52    for (int i=0; i<3; i++)
53        p[i] -> Print();
54    return 0;
```



Funções Virtuais Puras – C++

- Faz sentido termos um método virtual com corpo?

  Não!!!

- Dessa forma, temos a função virtual pura que não tem bloco de código;

- Para criar uma função virtual pura basta usar o operador de atribuição seguido de um zero após o seu protótipo:

```
1 class Base
2 {
3 public:
4     virtual void Print() = 0;
5 };
6
```

Exercício - C++

- Exercício:
  - **1)** Implemente, em C++, uma classe abstrata de nome *Paciente* com um atributo `char Nome[40]` e dois métodos:
    - `void GetNome()`
    - `void Print()`
  - **2)** Crie um classe derivada de *Paciente* de nome *Assegurado:*
    - cujas instâncias são caracterizadas pelos atributos `char Seguradora[40]` e `int NumSeguro`;
    - Implemente na classe *Assegurado* o método `Print()` herdado de *Paciente* e `GetSeguro()` para obter os atributos (dados) da classe;

Exercício – C++

- Exercício:
  - **3)** Crie um classe derivada de *Paciente* de nome *NaoAssegurado:*
    - cujas instâncias são caracterizadas pelos atributos `float Consulta, int Banco e int Cheque;`
    - Implemente na classe Nao*Assegurado* o método `Print()` herdado de P*aciente* e `GetValor()` para obter os atributos (dados) da classe;
  - ***4)*** Elabore um programa de teste onde é declarado um array, de dimensão 100, do tipo *Paciente.* Nesse array devem ser guardadas instâncias de *Assegurado* e Nao*Assegurado* a partir de uma seleção inicial (if). Após o usuário optar em não incluir mais pacientes, o programa de teste deve imprimir a relação de pacientes cadastrados;

Exercício - C++

Polimorfismo.cpp

```
1  #include <iostream>
2  #include <iomanip>
3  #include <cstdio>
4  using namespace std;
5
6  class Cliente
7  {
8  private:
9      char Nome[40];
10 public:
11     void GetNome()
12     {
13         cout << "Digite Nome: "; //cin.ignore(10, '\n');
14         cin.getline( Nome, 40 , '\n' );
15         //gets(Nome);
16     }
17     virtual void Print()
18     {
19         cout << Nome << endl;
20     }
21 };
22
```

Exercício - C++

Polimorfismo.cpp

```
23 class Assegurado : public Cliente
24 {
25 private:
26     char Seguradora[40];
27     int NumSeguro;
28 public:
29     void GetSeguro()
30     {
31         cout << "Digite o nome da Seguradora :";
32         cin.getline( Seguradora, 40 , '\n' );
33         //gets(Seguradora); cin.ignore(10, '\n');
34         cout << "Digite número do seguro: ";
35         cin >> NumSeguro;//cin.ignore(10, '\n');
36     }
37     void Print()
38     {
39         Cliente::Print();
40         cout << "     Seguradora: " << Seguradora << endl;
41         cout << "     Núm. do Seguro: " << NumSeguro << endl;
42     }
43 };
44
```

Exercício - C++

Polimorfismo.cpp

```
45class NaoAssegurado : public Cliente
46{
47private:
48    float Consulta;
49    int Banco, Cheque;
50public:
51    void GetValor()
52    {
53        cout << "Valor da Consulta: ";  cin >> Consulta;
54        cout << "Banco: ";              cin >> Banco;
55        cout << "Cheque: ";             cin >> Cheque;
56    }
57    void Print()
58    {
59        Cliente::Print();
60        //cout << setprecision(2) << setiosflags(ios::fixed);
61        cout << "       Valor da Consulta: " << Consulta << endl;
62        cout << "       Banco: " << Banco << endl;
63        cout << "       Cheque: " << Cheque << endl;
64    }
65};
66
```

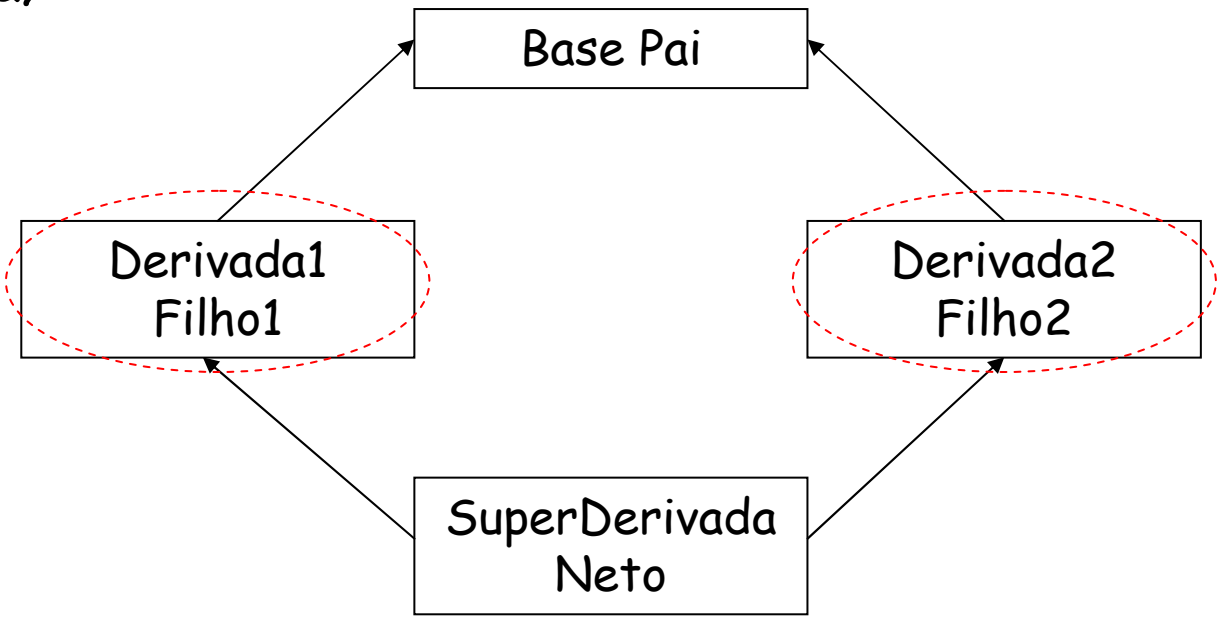
Exercício C++

Polimorfismo.cpp

```
67int main()
68{
69    Cliente *Paciente[100];
70    Assegurado *Ass;
71    NaoAssegurado *NaoAss;
72    int n=0;
73    char opcao;
74
75    do{
76        cout << "\nAssegurado (s/n) ? ";
77        cin >> opcao; cin.ignore(5, '\n');
78        if (opcao == 's')
79        {
80            Ass = new Assegurado;
81            Ass->GetNome();
82            Ass->GetSeguro();
83            Paciente[n++] = Ass;
84        }
85        else
86        {
87            NaoAss = new NaoAssegurado;
88            NaoAss->GetNome();
89            NaoAss->GetValor();
90            Paciente[n++] = NaoAss;
91        }
92        cout << "\nInserir outro paciente (s/n)?";cin.ignore(5, '\
93        cin >> opcao;
94        }while(opcao == 's');
95
96        for (int i =0; i<n; i++)
97            Paciente[i] -> Print();
98
99        return 0;
100}
```

Classe-Base Virtual

- Além de declararmos funções virtuais, podemos usar a palavra-chave `virtual` para declarar uma classe inteira;



Classe-Base Virtual

- O trecho do programa seguinte mostra o erro de ambigüidade caso a palavra-chave `virtual` não seja usada:

```
1 class Base
2 {
3 protected:
4     int BasInt;
5 };
6
7 class Derivada1 : public Base{};
8 class Derivada2 : public Base{};
9
10 class SuperDerivada : public Derivada1, public Derivada2
11 {
12 public:
13     int GetInt()
14     {
15         return BasInt;      // ERRO: Ambiguidade
16     }
17 };
```

Classe-Base Virtual

- A solução é simples e evita o erro de ambigüidade, vejamos:

```
1 class Base
2 {
3 protected:
4     int BasInt;
5 };
6
7 class Derivada1 : virtual public Base{};
8 class Derivada2 : virtual public Base{};
9
10 class SuperDerivada : public Derivada1, public Derivada2
11 {
12 public:
13     int GetInt()
14     {
15         return BasInt;       // ERRO: Ambiguidade
16     }
17 };
```

Classes Amigas

- Além de ser possível declararmos funções independentes como amigas, podemos declarar uma classe inteira como amiga da outra;
- Nesse caso, os métodos da classe será todos amigos da outra classe;
- Ou seja, os métodos terão acesso à parte privada ou protegida da outra classe.

Classes Amigas

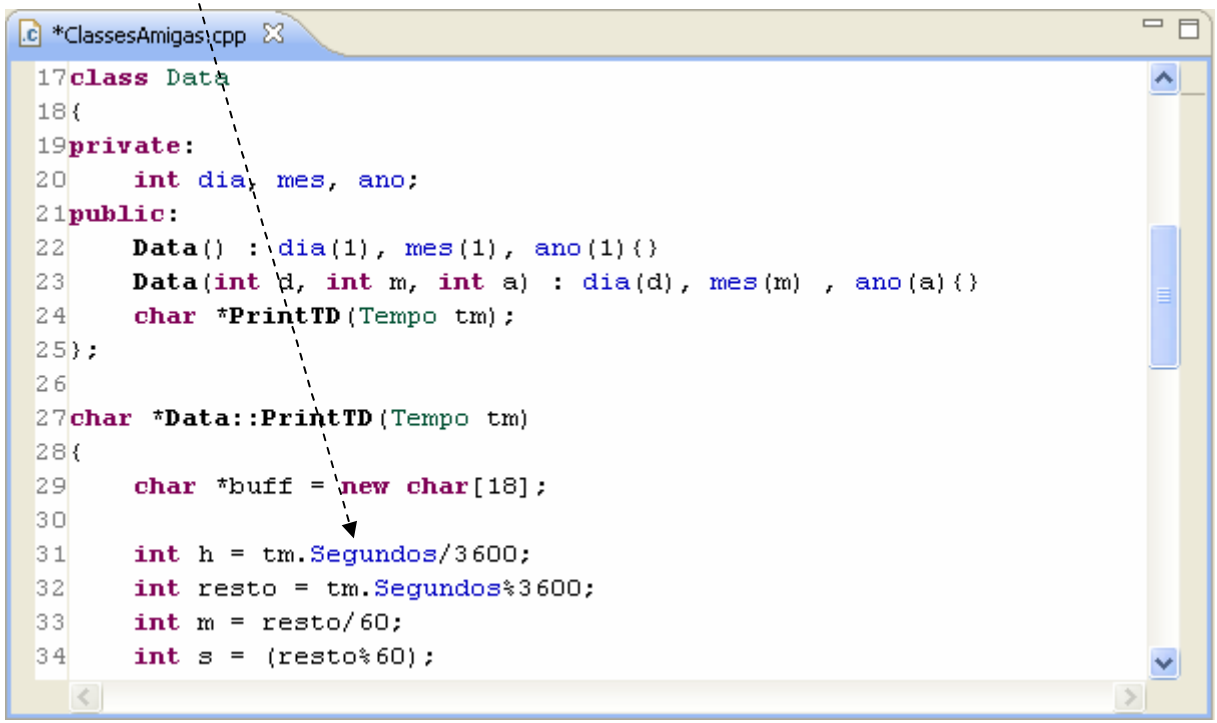
- Vejamos um exemplo de classes amigas (Tempo e Data):

*ClassesAmigas.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  class Tempo
5  {
6  private:
7      long Segundos;
8  public:
9      Tempo() : Segundos(0){}
10     Tempo(long h, long m, long s)
11     {
12         Segundos = h*3600 + m*60 + s;
13     }
14     friend class Data;  // Declaração de uma classe amiga
15 };
16
```

Classes Amigas

- Vejamos a classe Data:
  - Acesso à dados privados da classe Tempo.



Classes Amigas

- Continuação da classe Data:

*ClassesAmigas.cpp

```
35
36      buff[0] = h/10+'0';
37      buff[1] = h%10+'0';
38      buff[2] = ':';
39      buff[3] = m/10+'0';
40      buff[4] = m%10+'0';
41      buff[5] = ':';
42      buff[6] = s/10+'0';
43      buff[7] = s%10+'0';
44      buff[8] = '\n';
45
46      buff[9] = dia/10+'0';
47      buff[10] = dia%10+'0';
48      buff[11] = '/';
49      buff[12] = mes/10+'0';
50      buff[13] = mes%10+'0';
51      buff[14] = '/';
52      int Ano = ano %100;
53      buff[15] = Ano/10+'0';
54      buff[16] = Ano%10+'0';
55      buff[17] = '\0';
56
57      return buff;
58  }
```

Classes Amigas

- Programa teste:

*ClassesAmigas.cpp

```
59
60int main()
61{
62    Tempo tm(15, 10, 20);
63    Data dt(27,10,2007);
64    char *tmdt = dt.PrintTD(tm);
65    cout << tmdt << endl;
66    delete tmdt;
67    return 0;
68}
69
```

- Saída no Console:

Problems | Tasks | Properties | Console

<terminated> ClassesAmigas.exe [C/C++ Local Application] C

```
15:10:20
27/10/07
```

Aprenda Java com o BOPE

```
/**
 * O Bope foi criado para atuar quando a policia perde o controle
 * E no rio de janeiro isso acontece com bastante frequencia
 */

class Bope{

        private String nome;
        private int qtdeVitimas = 1; // ja começa bem!
        public Bope(String nome){
                this.nome = nome;

        }

        // sobrecarga do método ondeTaOBaiano
        public void ondeTaOBaiano(Estudante e){
                e.sabeVoarEstudante();
        }

        public void ondeTaOBaiano(Traficante t){
                t.levaSacoNaCabeca();
        }

        // exemplo de método final!

        public final Doze encontreiOBaiano(Baiano b){
                return b.naCaraNaoQueEhPraNaoEstragarVelorio();
        }

}
```

Aprenda Java com o BOPE

```
public class Treinamento{

        public void missao(CapitaoNascimento cn){
                cn.sentaOdedoNessaPorra();
        } // Sugestão do leitor Luciano Silva

        public static void main(String [] xxx){
//Apresento o capitão nascimento

        Bope capitao = new Bope("Ol"); //Capitao Nascimento
        try{

//de cada 100 policiais que fazem o curso do Bope,
//so se formam 5, e eu, quando me formei parceiro,
//eramos apenas 3.

        Turma.tentaFazerCursoBopeCom(capitao);

while(aluno.count >= 3){
        aluno.pedePraSair();
           if(aluno istanceof Cafetao){
                 capitao.say("Pede pra sair!");
                 capitao.say("Seu Lugar Eh Com Puta!");
           }
           if(aluno istanceof PorraLoka){
                   capitao.say("Tira essa farda preta!");
                   capitao.say("Voce nao eh cavera. voce eh MULEQUE
           }
           if( aluno.isXerife() ) aluno.desiste();
}
} catch (PolicialCorruptoEncontrado pce){
} catch (PolicialFracoEncontrado pfe){
} catch (PolicialSemABandoleiraNessaAlturadoCampeonato e) {
}
}
}
```

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.