

# TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

## Linguagem compilada ou interpretada?

**Linguagem compilada** é aquela que se submetem os códigos-fonte gerados pelo programador a um programa compilador que, por sua vez, gera um arquivo executável binário que é compreendido pelo sistema operacional.

Ou seja, esse executável pode ser executado diretamente pelo terminal de linha de comando do SO.

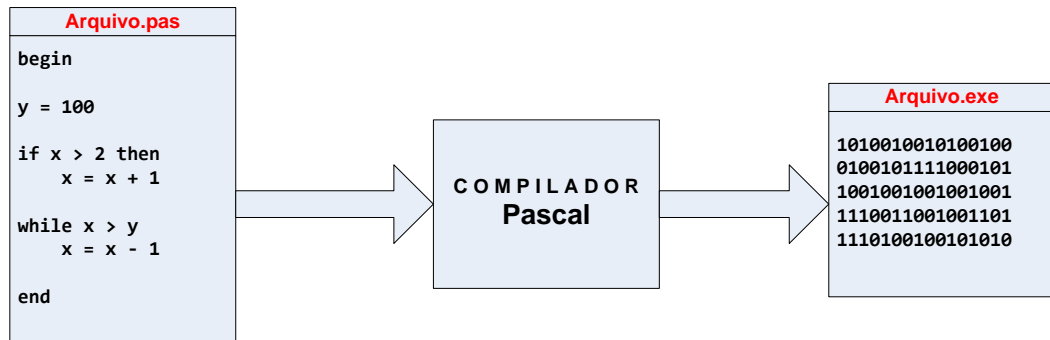


Figura 1

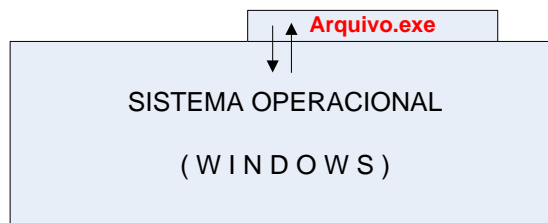


Figura 2

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

**Linguagem interpretada** é aquela cujo programa só é executado pela intermediação de um interpretador. O arquivo ASCII é submetido ao interpretador que, por sua vez, gera, em tempo real, um bloco binário executável em memória.

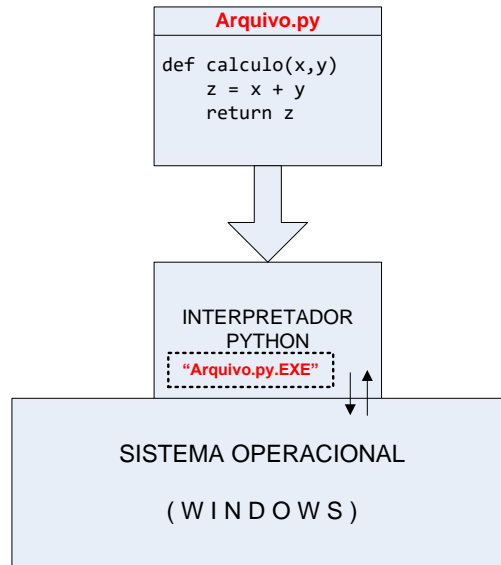


Figura 3

**A linguagem Java** tem uma composição mista de compilação e interpretação. O código-fonte é compilado, porém o binário gerado por esse compilador não é compreendido pelo sistema operacional e sim por um interpretador Java.

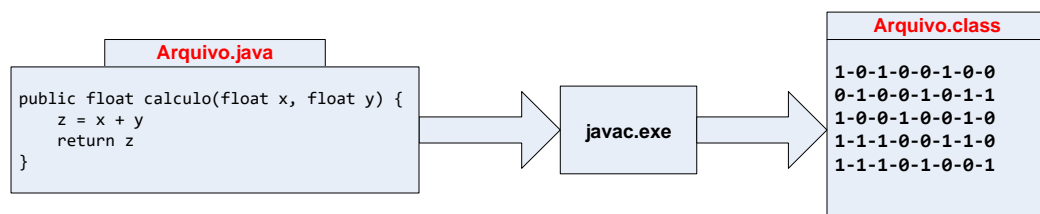


Figura 4

Onde:

- **Arquivo.java** = arquivo de código-fonte
- **javac.exe** = compilador Java
- **Arquivo.class** = arquivo binário compilado Java

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

Considerando uma instalação Java típica, em linha de comando, a compilação se dá por:

```
C:\Program Files\Java\jdk-9.0.4\bin\javac C:\<caminho>\Arquivo.java
```

Observe-se que o compilador *javac* só está disponível no kit de desenvolvimento, o jdk (Java development kit). A instalação Java para *runtime*, a JRE (vide a seguir), não possui esse recurso. Neste exemplo se tem a instalação da JDK versão 9.0.4. Dependendo da versão, o nome desta pasta mudará.

Os IDEs mais poderosos já incluem todo esse procedimento de compilação, desincumbindo o desenvolvedor de trabalhar com a linha de comando.

O interpretador Java é o chamado JRE (Java Runtime Environment) que se trata de uma implementação da chamada JVM (Java Virtual Machine) que, esta última, é a especificação completa de como deva operar esse interpretador Java.

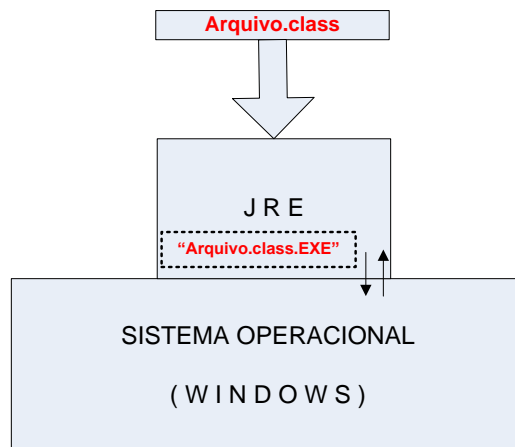


Figura 5

Para execução do programa Java em linha de comando, considerando uma instalação Java típica:

```
C:\Program Files\Java\jre-9.0.4\bin\java C:\<caminho>\Arquivo
```

Observe que o arquivo compilado (.class) entra como argumento do interpretador Java. Ou seja, o programa a ser executado via sistema operacional é o **java.exe**. Observe também que não é necessário adicionar a extensão *.class*.

Obs: A instalação JDK também possui o java.exe.

A ideia por trás do modelo do interpretador é que, dado um conjunto de arquivos compatíveis com o mesmo, sejam códigos-fonte (ASCII) ou binários (compilação intermediária), esses arquivos sejam portáteis e executáveis em qualquer computador com qualquer sistema operacional que execute aquele dado interpretador.

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

### Estrutura de um programa Java

O Java é 100% orientado a objetos, de forma que qualquer programa Java precisa ter todas as suas instruções acondicionadas em uma ou mais classes (excetuam-se as declarações de importação de bibliotecas). Usualmente tem-se um arquivo para cada classe. O nome desse arquivo deve ser idêntico ao da classe (inclusive na maiúsculas/minúsculas) seguido da extensão **.java**.

Em segundo lugar, todas as instruções precisam estar acondicionadas em métodos. Exceção feita à declaração de atributos (variáveis e constantes).

Para que um programa Java possa ser executado, é obrigatório que uma das classes, dita a classe principal, contenha o método **main**. A sintaxe do método **main** é a seguinte:

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

Onde:

- **public** = qualificador de visibilidade (externo à classe)
- **static** = qualificador que desvincula o método ou atributo do paradigma POO, ou seja, elementos *static* não são criados em um objeto instanciado daquela classe.
- **void** = tipo de retorno do método (*void* é um indicador que o método não dá nenhum tipo de retorno).
- **main** = nome do método (deve ser obrigatoriamente esse).
- **String[] args** = *args* é o nome da array que recebe argumentos (string) do teclado.

Exemplo:

```
1  import java.util.Scanner;
2
3
4  public class Aula01 {
5
6      public static void main(String[] args) {
7
8          Scanner scn = new Scanner(System.in);
9          System.out.print("valor de x -> ");
10         float a = scn.nextFloat();
11         System.out.print("valor de y -> ");
12         float b = scn.nextFloat();
13
14         float resultado = calculo(a, b);
15         System.out.println("b / a = " + resultado);
16     }
17
18     private static float calculo(float x, float y) {
19
20         if (x > 100) {
21             x = 100;
22             System.out.println("x foi ajustado para 100");
23         } else if (x < 1) {
24             x = 1;
25             System.out.println("x foi ajustado para 1");
26         }
27
28         return y / x;
29     }
30
31 }
32
33
```

Figura 6

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

Neste exemplo a nossa classe tem o nome de **Aula01**. Logo, o arquivo deverá se chamar **Aula01.java** (linha 4).

- O método *main* está declarado na linha 6 e a chave de fechamento está na linha 17.
- O método *main* começa com a instanciação de um objeto da classe **Scanner** (linha 8), que é a que nos permite obter dados do teclado.
- O método *print*, da classe *System*, atributo *out*, imprime na tela (linhas 9, 11, 23 e 26).
- Nas linhas 10 e 12 obtêm-se através da instância da classe *Scanner* (variável *scn*), os valores a serem enviados ao método *calculo* (início na linha 19, última instrução na linha 29 e chave de fechamento na linha 31).

Observe-se que a classe *Aula01* está operando na íntegra com métodos estáticos e nenhum objeto desta classe foi instanciado. Caso seja instanciado, este objeto não terá conteúdo nenhum, pois todos os componentes da classe (no caso apenas 2 métodos) são estáticos.

Lembrando que um contexto estático não tem acesso a um contexto não estático. Portanto, se o método **calculo** não fosse estático, o método *main* só conseguiria acessá-lo se instanciasse um objeto da sua própria classe. Vide exemplo da Figura 7 (linha 14), a chamada do método **calculo** (linha 16) e o método **calculo** sem a qualificação *static*. Neste caso uma instância da classe *Aula01b* levaria consigo o método **calculo**.

```
1
2  import java.util.Scanner;
3
4  public class Aula01b {
5
6      public static void main(String[] args) {
7
8          Scanner scn = new Scanner(System.in);
9          System.out.print("valor de x -> ");
10         float a = scn.nextFloat();
11         System.out.print("valor de y -> ");
12         float b = scn.nextFloat();
13
14         Aula01b objAb = new Aula01b();
15
16         float resultado = objAb.calculo(a, b);
17         System.out.println("b / a = " + resultado);
18
19     }
20
21     private float calculo(float x, float y) {
22
23         if (x > 100) {
24             x = 100;
25             System.out.println("x foi ajustado para 100");
26         } else if (x < 1) {
27             x = 1;
28             System.out.println("x foi ajustado para 1");
29         }
30
31         return y / x;
32     }
33 }
34
35
36
```

Figura 7

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

Nas estruturas de decisão, além da instrução *if*, tem-se a *switch*.

```
1
2 import java.util.Scanner;
3
4 public class Aula01c {
5
6     public static void main(String[] args) {
7
8         Scanner scn = new Scanner(System.in);
9
10        System.out.print("Valor de x (0, 1 ou 2) -> ");
11        int x = scn.nextInt();
12
13        System.out.println("O valor fornecido para x foi ");
14
15        switch (x) {
16
17            case 0:
18                System.out.println(" 0");
19                break;
20            case 1:
21                System.out.println(" 1");
22                break;
23            case 2:
24                System.out.println(" 2");
25                break;
26            default:
27                System.out.println("FORA DA FAIXA ESPERADA");
28
29        }
30
31    }
32
33 }
```

Figura 8

Laços iterativos:

```
1
2 import java.util.Scanner;
3
4 public class Aula01d {
5
6     public static void main(String[] args) {
7
8         Scanner scn = new Scanner(System.in);
9
10        System.out.print("Valor de x (maior que 0) -> ");
11        int x = scn.nextInt();
12
13        int soma = 0;
14        for (int i = 1; i <= x; i++) {
15            soma += i;
16        }
17        System.out.println("Soma pelo metodo for = " + soma);
18
19        soma = 0;
20        int i = x;
21        while (i > 0) {
22            soma += i;
23            i--;
24        }
25        System.out.println("Soma pelo metodo while = " + soma);
26
27        soma = 0;
28        i = 1;
29        do {
30            soma += i;
31            i++;
32        } while (i <= x);
33        System.out.println("Soma pelo metodo do...while = " + soma);
34
35    }
36
37 }
38 }
```

Figura 9

Na Java temos os laços *for* (linha 14-16), *while* (linha 21-24) e *do..while* (linha 29-32).

## POO em Java

Uma classe contém dois tipos de elementos:

- Atributos
- Métodos

Os atributos são variáveis ou constantes, caracterizando os valores e/ou dados pertencentes a um determinado objeto e os métodos são as funcionalidades da classe ou, em outras palavras, os algoritmos em si.

Uma classe pode representar um objeto do mundo real, concreto ou abstrato, físico ou um registro de um evento passado.

Uma classe pode ser composta por apenas atributos (classes de dados) ou apenas métodos (classes utilitárias).

### Qualificadores de acesso

Em Java existem 3 tipos de qualificadores de acesso, tanto para atributos como para métodos:

**public** – (público) acessível por qualquer método em qualquer classe de todo o sistema.

**private** – (privado) acessível apenas pelos métodos da própria classe.

**protected** - (protegido) acessível apenas pelas classes herdeiras ou classes do mesmo pacote.

**Sem qualificador** – acessível apenas pelas classes do mesmo pacote.

Os métodos da própria classe não tem nenhuma restrição de acesso.

Obs: **pacote**, em Java, em termos práticos, se caracteriza como sendo uma pasta (diretório) no disco.

### Encapsulamento

Usualmente os atributos são declarados como privados e, para serem acessados, são implementados os chamados **métodos de acesso**.

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

```
1 public class Aula01e_c1 {
2
3
4     private int numero;
5     private String nome;
6
7     public int getNumero() {
8         return numero;
9     }
10
11     public void setNumero(int numero) {
12         this.numero = numero;
13     }
14
15     public String getNome() {
16         return nome;
17     }
18
19     public void setNome(String nome) {
20         this.nome = nome;
21     }
22 }
23
24
```

Figura 10

No código da Figura 10, os dois atributos, **numero** e **nome** são declarados nas linhas 4 e 5 respectivamente como privados.

Abaixo têm-se os métodos de acesso, os *get*, para leitura dos conteúdos atuais e os *set*, para alteração desses conteúdos. Por convenção utiliza-se o sufixo *get/set* e a seguir o nome do método.

O objetivo principal do encapsulamento é possibilitar ou inibir o acesso a determinados atributos. Nos casos dos métodos *set*, pode também haver alguma necessidade de verificação de consistência. No exemplo abaixo temos o caso em que, pela regra do negócio, o atributo **numero** não pode ser negativo.

```
public void setNumero(int numero) {
    if (numero < 0) {
        numero = 0;
    }
    this.numero = numero;
}
```

### Herança

O mecanismo da herança permite que uma classe seja estendida pela fusão com outra. Por exemplo, uma determinada classe possui diversos métodos e uma determinada aplicação precisa de todos aqueles métodos e mais um que não tem lá.

Uma solução seria alterar o código fonte dessa classe, enxertando esse novo método, ou então, criar uma nova classe que contenha apenas este método e fazer a herança da maior.



## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

```
1 public class Aula01e_c2 extends Aula01e_c1 {
2
3
4     private String endereco;
5
6     public String getEndereco() {
7         return endereco;
8     }
9
10    public void setEndereco(String endereco) {
11        this.endereco = endereco;
12    }
13
14 }
15
```

Figura 11

Na Figura 11 vemos na declaração da classe (linha 2) a palavra chave *extends* que caracteriza a herança.

Nesta classe temos um único atributo (**endereço**) e, pela herança da classe **Aula01e\_c1**, o objeto resultante passa a se compor do que for da sua classe, acrescido de tudo que for público e protegido da sua classe mãe (superclasse). E também, como dito anteriormente, no caso da linguagem Java, caso a classe herdeira esteja no mesmo pacote da superclasse, elementos sem qualificação também são herdados. Mas, caso a superclasse esteja em outro pacote, elementos não qualificados não são herdados.

### Interfaces

Interface é um tipo de classe que só possui métodos declarados. Ou seja, esses métodos não são implementados.

O objetivo da interface é servir de modelo para as classes que implementam essa interface, de forma que a classe deverá conter todos os métodos relacionados na interface. Uma aplicação bastante frequente para a interface é em sistemas que utilizam o polimorfismo.

```
1 public interface Aula02_i1 {
2
3     public double calculo1(double x, double y);
4
5     public double calculo2(double x, double y);
6
7 }
8
```

Figura 12

```
1 public class Aula02_c1 implements Aula02_i1 {
2
3
4     @Override
5     public double calculo1(double x, double y) {
6         return x / y;
7     }
8
9     @Override
10    public double calculo2(double x, double y) {
11        return x * y;
12    }
13
14 }
15
```

Figura 13

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

```
1
2 public class Aula02_c2 implements Aula02_i1 {
3
4     @Override
5     public double calculo1(double x, double y) {
6         return x + y;
7     }
8
9     @Override
10    public double calculo2(double x, double y) {
11        return x - y;
12    }
13
14 }
15
```

Figura 14

Nos exemplos acima temos a Interface (Figura 12) e duas classes que a implementam (Figura 13 e Figura 14). Observe que ambas as classes têm os mesmos métodos relacionados na interface, porém com implementação diferente.

Obs: a classe que implementa a interface pode ter outros métodos além dos relacionados na interface, mas esses obrigatoriamente têm que constar.

A seguir um método *main* de manipulação dessas classes.

Nas linhas 11 e 12 os objetos foram instanciados com o *data type* da interface.

Das linhas 21 a 27 são chamados os métodos diretamente dos objetos instanciados.

Das linhas 29 a 33 os objetos são enviados para o método cálculos que recebe um parâmetro com o *data type* da interface.

```

1
2 import java.util.Scanner;
3
4 public class Aula02 {
5
6     static double x;
7     static double y;
8
9     public static void main(String[] args) {
10
11         Aula02_i1 c1 = new Aula02_c1();
12         Aula02_i1 c2 = new Aula02_c2();
13
14         Scanner scn = new Scanner(System.in);
15
16         System.out.print("x = ");
17         x = scn.nextDouble();
18         System.out.print("y = ");
19         y = scn.nextDouble();
20
21         System.out.println("Resultados da classe 1");
22         System.out.println("calculol -> " + c1.calculol(x, y));
23         System.out.println("calculol2 -> " + c1.calculol2(x, y));
24
25         System.out.println("Resultados da classe 2");
26         System.out.println("calculol -> " + c2.calculol(x, y));
27         System.out.println("calculol2 -> " + c2.calculol2(x, y));
28
29         System.out.println("\nResultados utilizando o data type da Interface");
30         System.out.println("Classe c1");
31         calculos(c1);
32         System.out.println("Classe c2");
33         calculos(c2);
34     }
35
36     private static void calculos(Aula02_i1 i1) {
37
38         System.out.println("calculol -> " + i1.calculol(x, y));
39         System.out.println("calculol2 -> " + i1.calculol2(x, y));
40
41     }
42 }
43
44

```

## Classes abstratas

Uma classe abstrata é uma classe que não pode ser instanciada, apenas estendida (herdada).

As classes abstratas são utilizadas nos casos em que existe a chamada especialização, por exemplo:

- Classe **Pessoa** (nome, endereço, telefone, e-mail)
- Classe **PessoaFisica** estende Pessoa (CPF, RG)
- Classe **PessoaJuridica** estende Pessoa (CNPJ, Insc. Estad.)

A classe **Pessoa** não faz sentido individualmente, mas sim em conjunto com **PessoaFisica** ou com **PessoaJuridica**.

A classe abstrata tem atributos e métodos como qualquer classe, mas também pode ter métodos abstratos. Esses são equivalentes aos métodos das Interfaces, apenas declarados, ou seja, a classe herdeira tem que implementar.

O Exemplo a seguir mostra uma classe abstrata **Aula02\_c3** (Figura 15) com duas herdeiras - **Aula02\_c3b** (Figura 16) e **Aula02\_c3c** (Figura 17).

A superclasse tem ainda um método abstrato, identif (linha 32), que as herdeiras devem implementar.

A tentativa de instanciamento da **Aula02\_c3** ocasionará uma exceção.

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

```
1
2
3 public abstract class Aula02_c3 {
4     private int nr;
5     private int nm;
6     private String end;
7
8     public int getNr() {
9         return nr;
10    }
11
12    public void setNr(int nr) {
13        this.nr = nr;
14    }
15
16    public int getNm() {
17        return nm;
18    }
19
20    public void setNm(int nm) {
21        this.nm = nm;
22    }
23
24    public String getEnd() {
25        return end;
26    }
27
28    public void setEnd(String end) {
29        this.end = end;
30    }
31
32    public abstract String identif();
33
34 }
35
```

Figura 15

```
1
2 public class Aula02_c3b extends Aula02_c3 {
3
4     private int idB;
5
6     public int getIdB() {
7         return idB;
8     }
9
10    public void setIdB(int idB) {
11        this.idB = idB;
12    }
13
14    @Override
15    public String identif() {
16        return "IDENTIFICACAO: Numero = " + getNr() + " - Nome = " + getNm();
17    }
18
19 }
20
```

Figura 16

```
1
2 public class Aula02_c3c extends Aula02_c3 {
3
4     private int idC;
5
6     public int getIdC() {
7         return idC;
8     }
9
10    public void setIdC(int idC) {
11        this.idC = idC;
12    }
13
14    @Override
15    public String identif() {
16        return "ID = " + getNr() + ", " + getNm() + ", " + getEnd();
17    }
18
19 }
20
```

Figura 17

## Tratamento de exceções

Exceções são situações que o programa não consegue resolver e simplesmente encerra a execução.

Para evitar o encerramento existe uma estrutura para tratamento da exceção, caso ocorra.

Exceções costumam ocorrer em algumas situações como, por exemplo:

- Erros de programação em tempo de execução:
  - Acesso a objetos não instanciados (nulos).
  - Divisão por zero.
  - Tentativa de acesso fora dos limites de um array.
  - Objeto de tipo diferente do esperado.
- Entrada de dados com tipo diferente do previsto:
  - Espera-se um *float* e o usuário digitou um *char*.

A estrutura para tratamento da exceção é o bloco *try..catch*.

```

1
2  import java.util.Scanner;
3
4  public class Aula02_c4 {
5
6      public static void main(String[] args) {
7
8          Scanner scn = new Scanner(System.in);
9          int x;
10
11          try {
12              System.out.print("Valor de x -> ");
13              x = scn.nextInt();
14              System.out.println("x * x = " + x * x);
15          } catch (Exception ex) {
16              System.out.println("ERRO, vide mensagem abaixo.");
17              System.out.println("-----");
18              System.out.println(ex.getMessage());
19              System.out.println(ex.getCause());
20          }
21
22          scn.nextLine();
23          System.out.print("Forneça x novamente -> ");
24          x = scn.nextInt();
25
26          int w = 0;
27          try {
28              double z = x / w;
29          } catch (Exception ex) {
30              System.out.println("ERRO, vide mensagem abaixo.");
31              System.out.println("-----");
32              System.out.println(ex.getMessage());
33          }
34
35      }
36
37  }
38

```

Figura 18

No exemplo da Figura 18, temos um bloco *try..catch* entre as linhas 11 e 20. Qualquer exceção que ocorra no bloco *try*, o fluxo de execução é imediatamente transferido para o bloco *catch*. Neste caso, a única instrução passível de erro é a da linha 13, onde é esperado um *int* e o usuário pode digitar uma *string*.

No bloco *catch* deve ser declarada uma variável que captura os dados da exceção. No caso foi chamada de **ex**.

```
catch(Exception ex) {
```

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

O tipo dessa variável foi declarado como *Exception*, que é o tipo mais genérico de exceção e, por conseguinte, captura todos os subtipos da *Exception*.

No caso da instrução da linha 13, o tipo específico é o *InputMismatchException*. Então, nosso bloco *catch* poderia ser escrito da seguinte forma:

```
catch(InputMismatchException ex) {
```


A carga computacional é muito menor para captura de exceções de tipo específico. Em contrapartida, caso haja a possibilidade de ocorrer algum outro tipo, o bloco *catch* não será acionado e o programa irá parar.

No bloco *catch* da linha 29, poderíamos utilizar o subtipo *ArithmeticException*. Acima foi forçada uma divisão por zero.

### Aplicações de interface gráfica

As interfaces gráficas (GUI - Graphical User Interface) são de utilização muito mais intuitiva, em especial quando se desenvolve para usuários que não são técnicos em TI.

No passado havia ambientes que possibilitavam a construção de telas completas no ambiente de linha de comando.



Clientes	
Código.....:	1
Nome.....:	
CPF.....:	-
Data.....:	17/07/2014
Endereço.....:	
Bairro.....:	
Complemento.:	
Cidade.....:	
Estado.....:	

Figura 19

A tela mostrada na Figura 19 é típica de um sistema de cadastro de clientes desenvolvida para o ambiente DOS.

A seguir, a mesma tela no modo GUI.

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

Figura 20

A tela gráfica é uma classe, definida pelo desenvolvedor, que deve estender a classe JFrame.

O código desta tela tem os seguintes trechos:

```
public class Janela1 extends javax.swing.JFrame {
```

Onde:

- **Janela1** é o nome que o desenvolvedor atribuiu à classe.
- **javax.swing.JFrame** é a classe base das janelas gráficas.

Desenvolvendo-se no IDE Netbeans, o código é todo gerado de forma reversa, ou seja, desenha-se a tela e o IDE gera o código.<sup>1</sup>

O construtor é definido da seguinte forma:

```
public Janela1() {  
    initComponents();  
}
```

O método **initComponents()** é justamente o que faz todo o trabalho de montagem da tela com todos os seus componentes e posicionamentos. Esse método **initComponents()** é gerado pelo IDE à medida em que o desenvolvedor arrasta e solta componentes na tela (labels, caixas de texto, botões, etc.)

A seguir o trecho inicial desse método:

---

<sup>1</sup> Nos primeiros ambientes GUI, o programador precisava escrever o código da tela, o que se transformava em uma atividade extremamente trabalhosa, não só pela definição dos componentes, mas principalmente na definição das dimensões e alinhamentos.

## TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01

Prof. Marcio Feitosa

```
private void initComponents() {

    jTextField1 = new javax.swing.JTextField();
    jTextField2 = new javax.swing.JTextField();
    jTextField3 = new javax.swing.JTextField();
    jTextField4 = new javax.swing.JTextField();
    jTextField5 = new javax.swing.JTextField();
    jTextField6 = new javax.swing.JTextField();
    jTextField7 = new javax.swing.JTextField();
    jTextField8 = new javax.swing.JTextField();
    jTextField9 = new javax.swing.JTextField();
    jButton1 = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    jLabel6 = new javax.swing.JLabel();
    jLabel7 = new javax.swing.JLabel();
    jLabel8 = new javax.swing.JLabel();
    jLabel9 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("Clientes");

    jTextField1.setMaximumSize(new java.awt.Dimension(100, 2147483647));
    jTextField1.setMinimumSize(new java.awt.Dimension(200, 22));

    jTextField9.setMaximumSize(new java.awt.Dimension(100, 20));
    jTextField9.setMinimumSize(new java.awt.Dimension(100, 20));

    jButton1.setText("Salvar");
    jButton1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButton1ActionPerformed(evt);
        }
    });

    jLabel1.setText("Código.....");
    jLabel1.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);

    (.....)
}
```

No primeiro bloco se tem uma sequência de instantiamentos referentes aos componentes de tela. Sim, cada componente vem de uma classe e um objeto, referente a cada um, é instanciado.

A seguir definem-se dimensões, posicionamentos, títulos, etc. No caso dos botões, abre-se um método que deve executar o que o botão deve fazer (vide o JButton1 que, neste exemplo, nenhum código foi acrescentado).

A figura abaixo mostra o ambiente Netbeans durante o desenvolvimento da tela (*design time*).



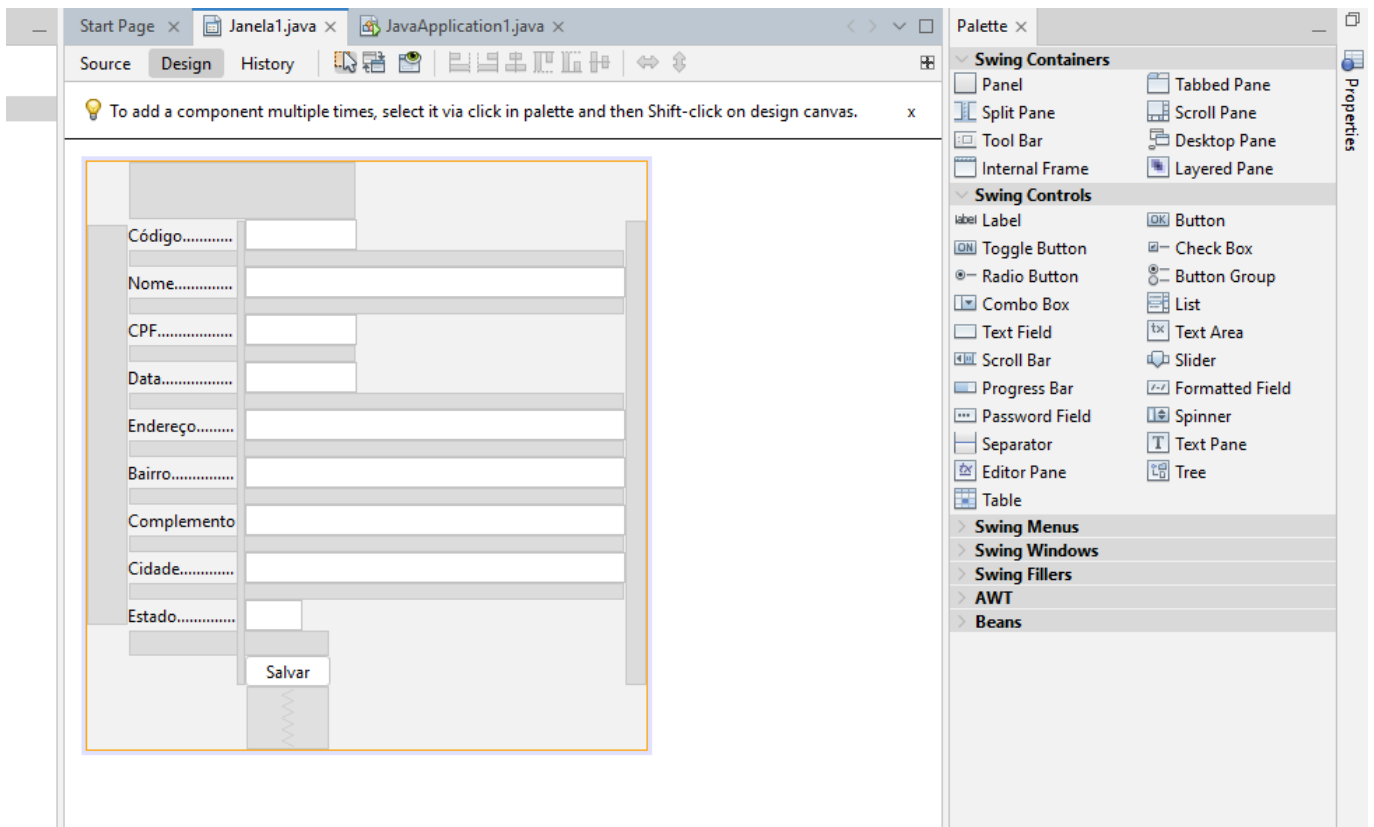


Figura 21

Do lado esquerdo vê-se a tela sendo construída e do lado direito a paleta de componentes. Para inserir componentes, arrasta-se o novo componente da paleta para cima da tela.

Nas telas gráficas se tem, em abundância, o recurso dos chamados **eventos**. Quando se dá um clique em cima da tela, por exemplo, isto é um evento e se pode associar um método à ocorrência desse evento. Se nada for escrito, o evento simplesmente fica sem ação nenhuma.

Alguns outros eventos:

- Tecla pressionada
- Tecla liberada
- Mouse pressionado
- Mouse liberado
- Componente ganhando o foco
- Componente perdendo o foco
- Componente recebendo um clique de mouse
- Mouse passando por cima de um componente (sem clique)
- Tela sendo minimizada
- Tela sendo maximizada
- Tela sendo ativada
- Tela sendo desativada
- ... entre outros

## **TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01**

Prof. Marcio Feitosa

### **Como acessar arquivos no disco**

O acesso a arquivos no disco é fundamental quando se trata de aplicações que precisam salvar e/ou ler dados no disco.

Primeiramente vamos ver como acessar arquivos de texto e em seguida como acessar bancos de dados.

**TOPICOS ESPECIAIS DE PROGRAMAÇÃO – AULA 01**  
Prof. Marcio Feitosa