Java
Introdução

Wagner G. Al-Alam
wgalam@gmail.com

Java

- Linguagem de Programação Orientada a Objetos
- Linguagem de alto nível que pode ser compilada ou interpretada
- Possui Garbage Collection
  - Coleta automática de lixo de memória.
- Multi-plataforma
- Grande variedade de bibliotecas disponíveis

Java

- Java combina a idéia do compilador e interpretador
- A linguagem é *compilada para bytecodes*
  - **Java Bytecode: é uma linguagem de máquina genérica,** que não é aceita por nenhum hardware
- Máquina Virtual Java: (Java Virtual Machine) Programa que executa programas em Java compilados para *bytecode.*
- O mesmo programa Java compilado para *bytecode* pode rodar em máquinas diferentes, basta existir uma JVM para a plataforma em que o programa será executado

Introdução
Comentários

- Existem 3 tipos permitidos de comentários nos programas feitos em Java:

```
// comentário de uma linha

/* comentário de uma ou mais linhas */

/** comentário de documentação */ (Arquivos de documentação)
```

Introdução

Ponto e vírgula, Blocos e o espaço em branco

- No java, os comandos são terminados com o sinal de ponto e vírgula (;)
- Um bloco tem iníco e tem o seu fim representados pelo uso das chaves {};
- O uso do espaço em branco permite uma melhor visualização dos comandos e em consequencia facilita a sua manutenção.

Introdução
Tipos básicos

Type	Size in bits	Values	Standard
`boolean`	8	`true` or `false`	
`char`	16	`'\u0000'` to `'\uFFFF'` (0 to 65535)	(ISO Unicode character set)
`byte`	8	–128 to +127 (-2^7 to $2^7 - 1$)	
`short`	16	–32,768 to +32,767 (-2^{15} to $2^{15} - 1$)	
`int`	32	–2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$)	
`long`	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (-2^{63} to $2^{63} - 1$)	
`float`	32	*Negative range:* –3.4028234663852886E+38 to –1.40129846432481707e–45 *Positive range:* 1.40129846432481707e–45 to 3.4028234663852886E+38	(IEEE 754 floating point)
`double`	64	*Negative range:* –1.7976931348623157E+308 to –4.94065645841246544e–324 *Positive range:* 4.94065645841246544e–324 to 1.7976931348623157E+308	(IEEE 754 floating point)

Introdução
Listas de Tipos Básicos

- Arrays:
- int [] numInteiro = new int[10];
- Int [] n = {1,2,3,4,5,6,7,8,9,10};
- String [] str = new String[];

- Arrays multidimensionais:
- int [][] numInteiro = new int[10][10];

Introdução
Operações Aritméticas

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	bm	`b * m`
Division	`/`	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Modulus	`%`	$r \bmod s$	`r % s`

Introdução
Ordem de Precedência

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses on the same level (i.e., not nested), they are evaluated left to right.
*, / and %	Multiplication Division Modulus	Evaluated second. If there are several of this type of operator, they are evaluated from left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several of this type of operator, they are evaluated from left to right.

Introdução
Operadores de Igualdade e Relacionais

Standard algebraic equality or relational operator	Java equality or relational operator	Example of Java condition	Meaning of Java condition
Equality operators			
=	==	`x == y`	**x** is equal to **y**
≠	!=	`x != y`	**x** is not equal to **y**
Relational operators			
>	>	`x > y`	**x** is greater than **y**
<	<	`x < y`	**x** is less than **y**
≥	>=	`x >= y`	**x** is greater than or equal to **y**
≤	<=	`x <= y`	**x** is less than or equal to **y**

Introdução
Operadores de Atribuição

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Introdução
Operadores

- Operadores Lógicos
  - AND:
    - &&
  - OR
    - ||
- Diferença entre o Pré-Incremento e o Pós-Incremento:

x = 10

++x => neste exato momento a variável a vale 11

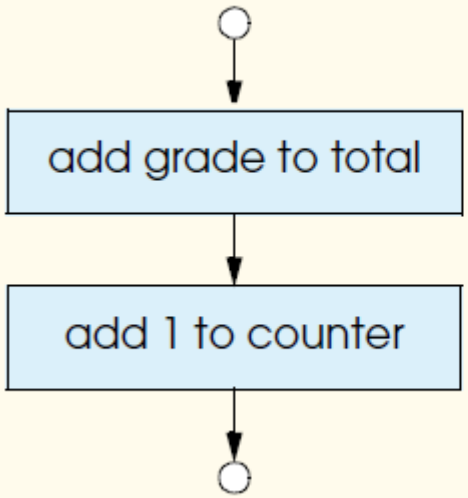
x =10

x++ => neste exato momento a variável x vale 10

Introdução
Programação Estruturada

- A programação é estruturada a partir de 3 sequências de controle:
  - Estrutura de sequência;
  - Estrutura de seleção;
  - Estrutura de repetição.

Introdução
Estrutura de Sequência



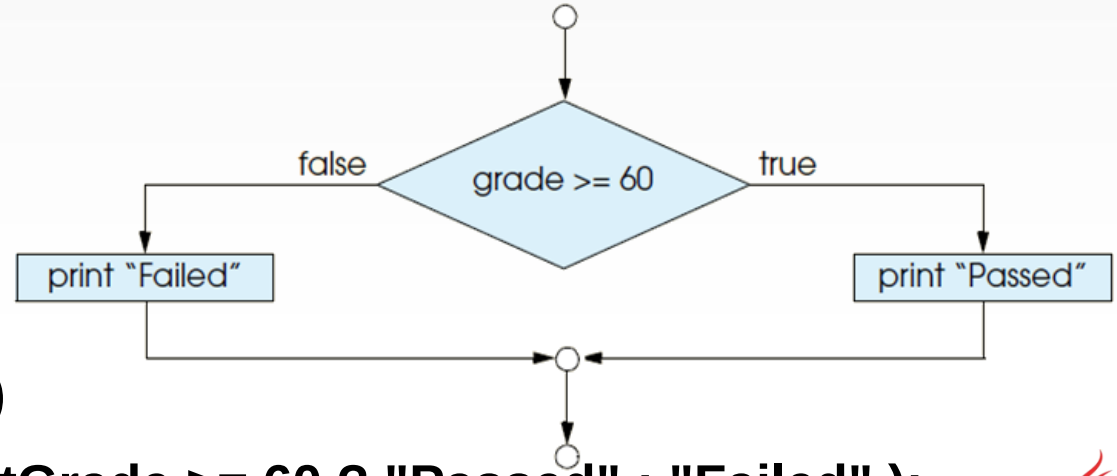

```
total = total + grade;
counter = counter + 1;
```

Introdução
Estrutura de Seleção

- Estrutura If/else

```
if(predicado){
   //Ação caso verdadeiro
   //Segunda ação se verdadeiro executada na sequencia.
} else
   //Ação caso falso(1 linha), não precisa de  { }
```

Ou



Operador condicional (?:)

System.out.println(studentGrade >= 60 ? "Passed" : "Failed");

Introdução
Estrutura de Seleção

- **Estrutura if/else aninhados:**

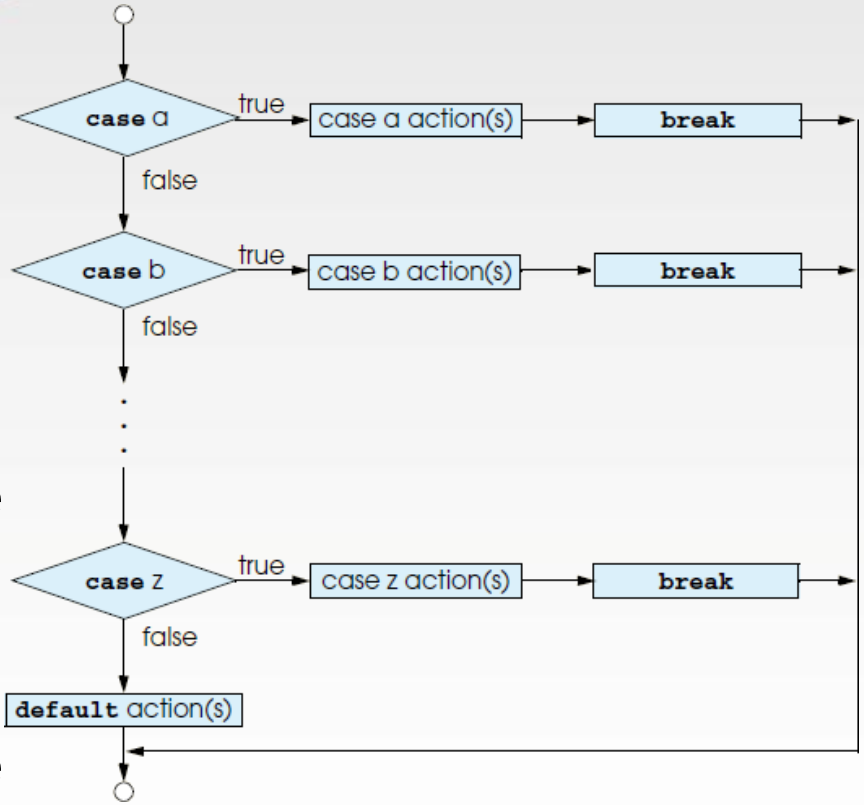
```
If student's grade is greater than or equal to 90
    Print "A"
else
    If student's grade is greater than or equal to 80
        Print "B"
    else
        If student's grade is greater than or equal to 70
            Print "C"
        else
            If student's grade is greater than or equal to 60
                Print "D"
            else
                Print "F"
```

```
if ( studentGrade >= 90 )
   System.out.println( "A" );
else
   if ( studentGrade >= 80 )
      System.out.println( "B" );
   else
      if ( studentGrade >= 70 )
         System.out.println( "C" );
      else
         if ( studentGrade >= 60 )
            System.out.println( "D" );
         else
            System.out.println( "F" );
```

Introdução
Estrutura de Seleção Múltipla

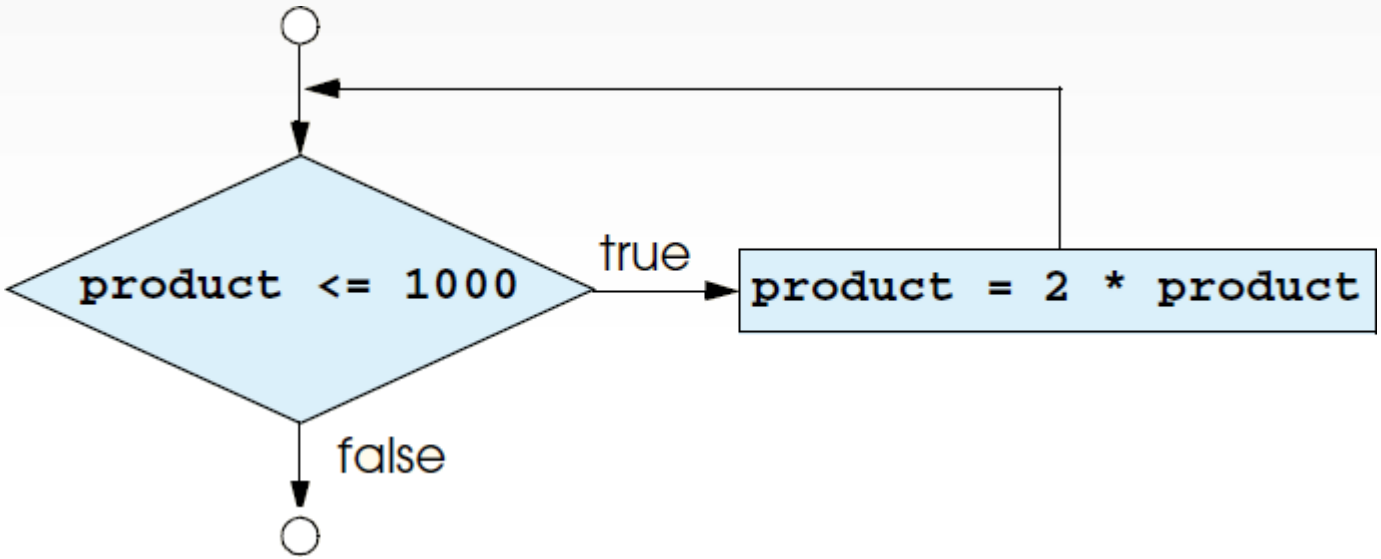
- **Switch/case**

```
switch ( choice ) {
        case 1:
              //Ação caso 1
              break; // done processing case
        case 2:
              //Ação caso 2
              break; // done processing case
        default:
              //Ação caso não satisfaça anteriores
}
```

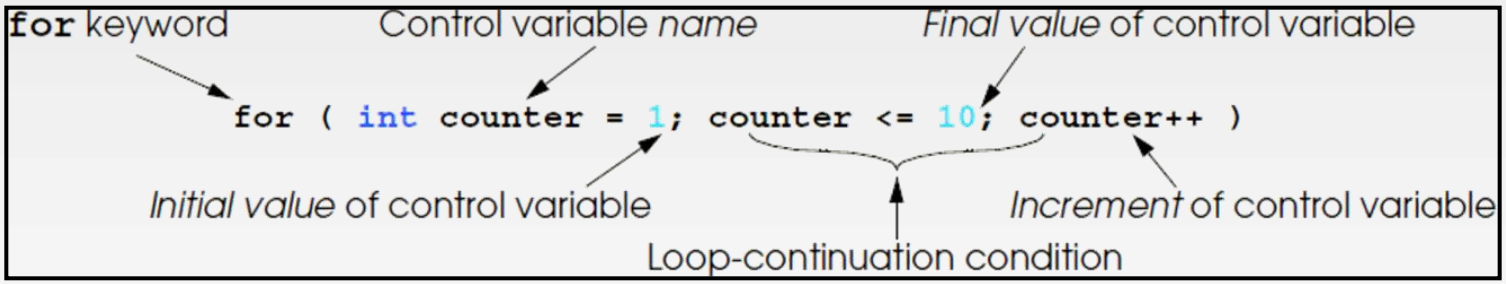


- **Estrutura de repetição *While*:**

```
int product = 2;
while ( product <= 1000 )
        product = 2 * product;
```



Introdução
Estruturas de Repetição



- Formato geral para a estrutura For:

for (*expression1; expression2; expression3)*

//Ação

- Equivalente usando a estrutura While:

expression1;

while (*expression2) {*

//Ação

expression3;

}

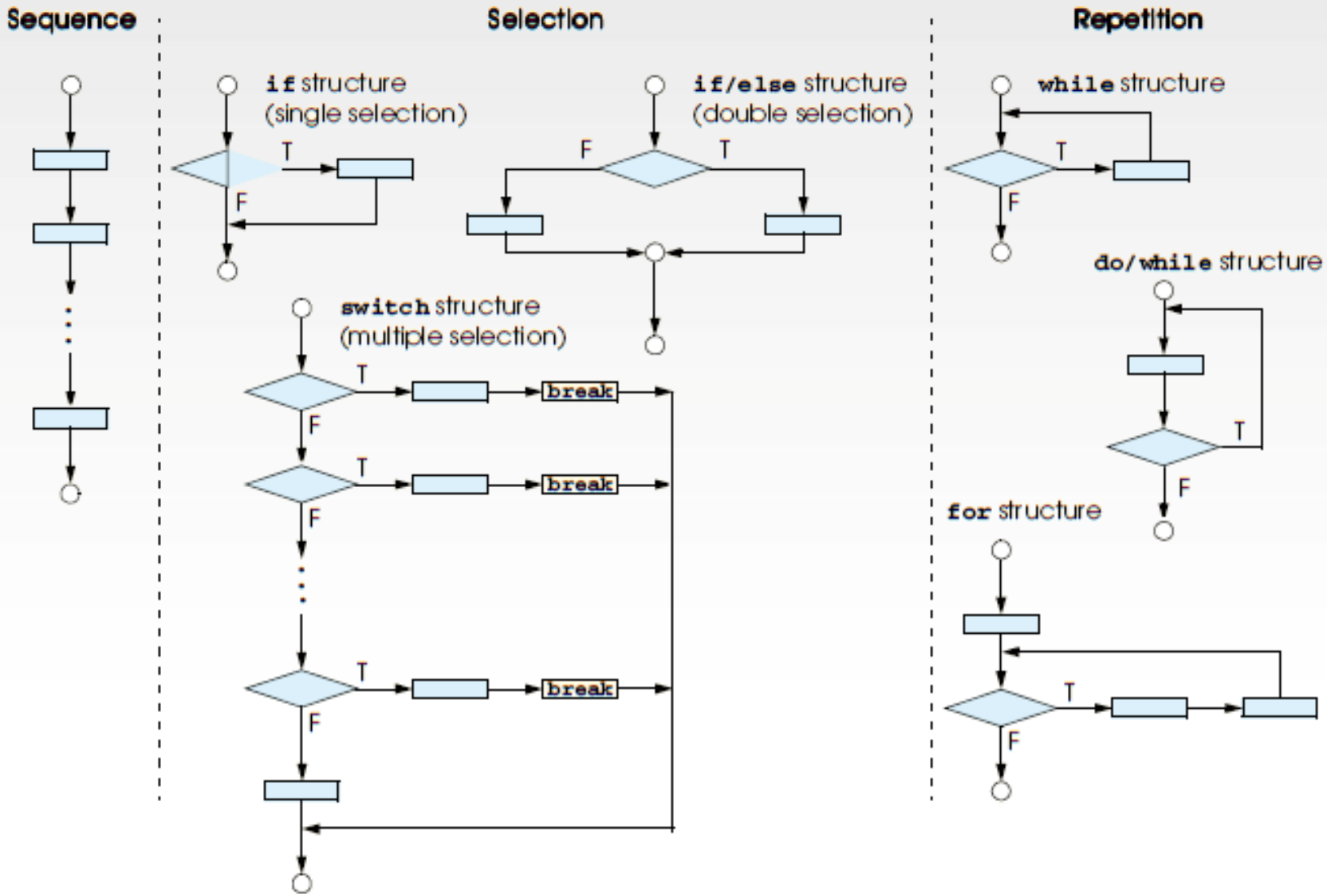
- **Foreach**
  - Percorrer uma lista
  - Exemplo:

```
int [] teste  = new int[] {1,2,3,4,5};
for(int i: teste){
   System.out.println(i);
```

- **Estrutura de repetição Do / *While*:**

```
int product = 2;
do{
    product = 2 * product;
} while ( product <= 1000 )
```

Introdução
Estruturas de Repetição - Resumo



Introdução
Métodos e Classes

- Módulos em Java são chamados de métodos e classes:
  - Os programas em Java são escritos combinando-se novos métodos e classes que o programador escreve, com outros métodos e classes (pré-empacotados) disponíveis na Java API (também conhecida como biblioteca de classes Java) e em várias outras bibliotecas de métodos e classes.
  - O programador pode escrever métodos para definir tarefas específicas que um programador pode utilizar muitas vezes durante sua execução.

Construtores

- Construtores são métodos especiais que são executados automaticamente quando um objeto é criado
- Um construtor **sempre possui o mesmo nome de sua classe**
- O construtor é diferente de outros métodos pois ele não possui tipo de retorno
- Se uma classe não possui construtor, o compilador gera um construtor automaticamente, sendo este em branco;

- Declaração de métodos:

```
int multiplicaPor2(int i){
   return 2*i;
}
```

- Método sem retorno e sem argumento:

```
void naoFazNada(){
   //Método que não faz nada
}
```

- **Sobrecarga de métodos:**

```
public static int multiplica2(int i){
   return 2 * i;
}
public static double multiplica2(double f){
   return 2 * f;
}
```

Método toString()

- Geralmente uma classe posssui um método que é usado para transformar o conteúdo do objeto em uma String
- Permite que se imprima facilmente o conteúdo do objeto
- Esse método em Java geralmente é chamdo de toString()

```
class Point {
private int x, y;
public Point (int x, int y) {(...) }
public int getX () { return x;}
(...)
public String toString() {
      return ("(" + x + ","+ y+")" );
      }
}
```

Tipos Abstratos de Dados

```
private class Aluno{
   public String nome;
   public int idade;
   public String matricula;

public aluno(String nome, int idade, String matricula){
   this.nome = nome;
   this.idade = idade;
   this.matricula = matricula;
   }
}
```

Usando tipos abstratos de dados

- Exemplo do uso do tipo personalizado:

```
class TestaAluno{
public static void main(String [] args){
   Aluno aluno = new Aluno(“Wagner”, 28, “00123456”);
}
}
```

Tipos Abstratos de Dados
Modificadores de Acesso de Usuários

- Modificadores:
  - Private
    - Acesso somente local
    - Para acesso externo necessita de métodos *getters* e *setters.*
  - Public
    - Acesso público fora da Classe

Coleções Dinâmicas em Java

- **COLEÇÃO DE OBJETOS: Vector, ArrayList**
- Um Vector é uma coleção de objetos que cresce automaticamente, ou seja, não é necessário declarar-se um tamanho para um Vector
- Para usar um Vector é preciso importar “ import java.util.*; ”
- Métodos principais:
  - size() que retorna o tamanho atual do Vector
  - add(Object o) adiciona um objeto ao Vector,
  - Object get(int index) que devolve um objeto na posição index
  - boolean remove (Object o) que remove a primeira ocorrência do objeto *o* dentro do vetor
  - Object remove(int index) , que remove o objeto na posição index
-

Coleções Dinâmicas em Java

- Declarando um vector:
  - Vector v = new Vector();
- Declarando um vector com declaração de tipo:
  - Vector<String> = New Vectror<String>()

Lendo Valores do Teclado

- Exemplo:

```
public static void main(String[] args) throws IOException {
   BufferedReader keyboard = new BufferedReader(new InputStreamReader(System. in) );
   String name;
   System.out.println ("Name? ");
   name = keyboard.readLine();
   System.out.println ("Hello " + name);
}
```

- Ler do teclado pode gerar uma Exceção, por isso usa-se throws IOException .
  - Voltaremos a esse assunto mais tarde

Lendo Números

- Usa-se o método estático Integer.parseInt:
- number = Integer.parseInt(keyboard.readLine( ));
- rate = Double.parseDouble(keyboard.readLine( ));

Exceções

- ***O que é uma Exception ?***

- A classe de Exceção define condições de erro moderados que seus programas podem encontrar.
- Em vez de você deixar o programa terminar, você pode escrever um código para tratar as exceções e continuar a execução do programa.

Manipulação de Exceções

- Uma maneira de manipular possíveis erros, é usando as declarações try e catch. A declaração try indica a parte do código aonde poderá ocorrer uma exception, sendo que para isso você deverá delimitar esta parte do código com o uso de chaves.

```
try {
        // código que pode ocasionar uma exception
}catch{
        // tratamento do erro
}
```

Manipulação de Exceções com Finally

- A declaração finally é utilizada para definir o bloco que irá ser executado tendo ou não uma exception, isto após o uso da declaração de try e catch.

```
try {
        // código que pode ocasionar uma exception
}catch{
        // tratamento do erro
}finally {
        // código
}
```

Primeiro programa em Java

- Hello World

```
class HelloWorld{
public static void main(String [] args){
   System.out.println(“Hello World”);
}
}
```

Compilando e Executando um Programa

- Gerando bytecode:
  - > Javac HelloWorld.java
  - Cria o arquivo HelloWorld.class
- Executando na JVM
  - > java HelloWorld
-

Exemplos

// Calculando a soma dos elementos de um array.

```
public class SumArray
{
   public static void main( String args[] )
   {
      int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;
      // adiciona o valor de cada elemento ao total
      for ( int counter = 0; counter < array.length; counter++ )
         total += array[ counter ];
      System.out.printf( "Total of array elements: %d\n", total );
   } // fim de main
} // fim da classe SumArray
```

Exemplos

```
//Calculando o Fatorial Recursivo
public class Fatorial {
   public static long fatorial(long number) {
            if (number == 0) {
                     return 1;
            }
            return number * fatorial(number - 1);
   }
   public static void main(String[] args) {
            System.out.println(Fatorial.fatorial(5));
   }
}
```

Exemplos

```
//Fatorial Iterativo
public class Fatorial{
   public static void main(String[] args) {
            fatorial(5);
   }
   public static void fatorial(int n){
            int fat = 1;
            for (int i = 1; i <= n; i++){
                     fat = fat * i;
                     System.out.println(fat);
            }
   }
}
```

Referências

- [1] Java Como Programar. Deitel 4ª edição.
- [2] Programação Java. André Rauber Du Bois. Universidade Católica de Pelotas.
- [3] http://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)