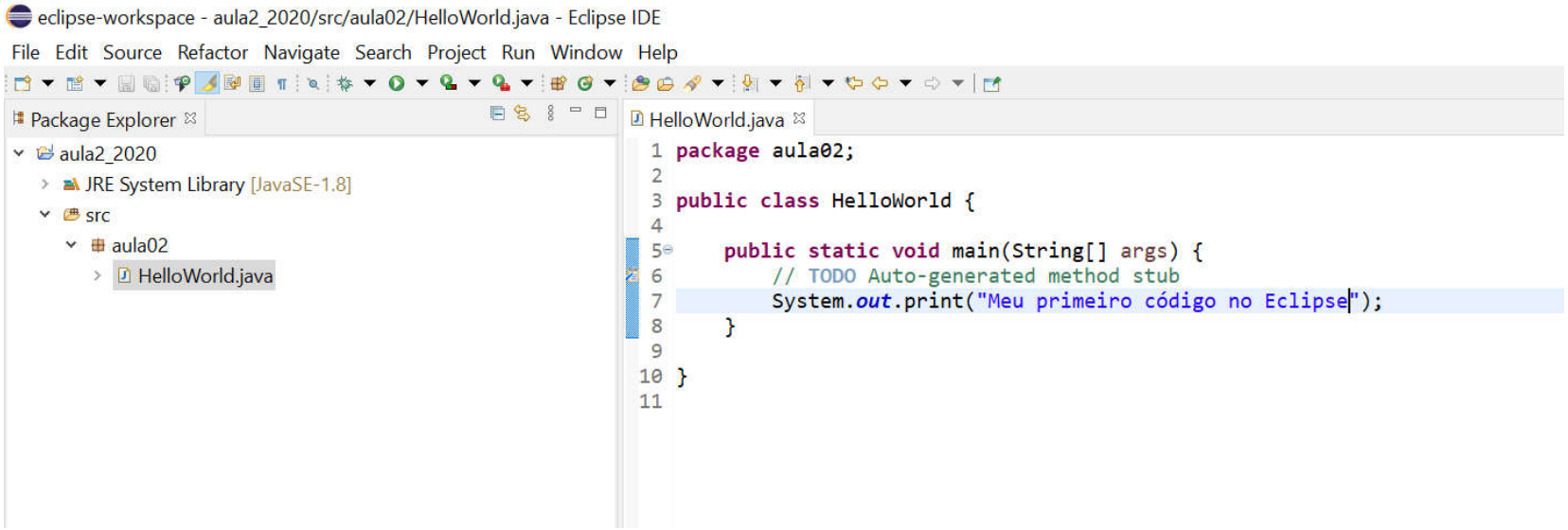
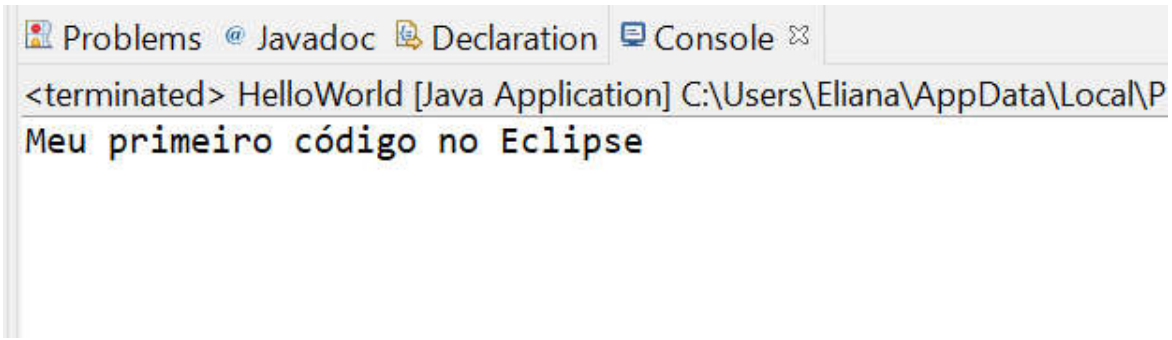
Técnicas especiais de Programação orientada a objetos

Aula 02 – Tipo de variáveis e operadores

UNIP
UNIVERSIDADE PAULISTA


eclipse-workspace - aula2_2020/src/aula02/HelloWorld.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
aula2_2020
JRE System Library [JavaSE-1.8]
src
aula02
HelloWorld.java
HelloWorld.java
1 package aula02;
2
3 public class HelloWorld {
4
5 public static void main(String[] args) {
6 // TODO Auto-generated method stub
7 System.out.print("Meu primeiro código no Eclipse");
8 }
9
10 }
11


Problems
Javadoc
Declaration
Console
<terminated> HelloWorld [Java Application] C:\Users\Eliana\AppData\Local\P
Meu primeiro código no Eclipse

Gerando saída com System.out

- **System.out.print** -> exibe a string e mantem o cursor na mesma linha
- **System.out.println** -> exibe a string e pula linha após a exibição, ou seja, posiciona o cursor de saída no começo da próxima linha na janela de comando.

- Ambas as instruções retomam a exibição dos caracteres a partir de onde a última instrução parou.

Gerando saída com System.out

- **\n** -> nova linha. Posiciona o cursor de tela no ínicio da próxima linha.Ex:

```
System.out.println("Welcome\nto\nJava\nProgramming!");
```

- **\t** -> tabulação horizontal. Move o cursor de tela para a próxima parada de tabulação.
- **** -> barras invertidas. Utilizadas para imprimir um caractere de barra invertida.
- **\”** -> Aspas duplas. Utilizada para imprimir um caractere entre aspas duplas. Ex:

```
System.out.println("\"aspas duplas\"");
```

Gerando saída com System.out

- O método **System.out.printf** (f significa “formato”) exibe a saída do programa com os dados *formatados*.
- O comando abaixo utiliza esse método para gerar a saída em duas linhas de strings “Welcome to" e “Java Programming!”.

```
System.out.printf("%s%n%s%n", "Welcome to", "Java Programming");
```

Gerando saída com System.out

- O primeiro argumento do método **printf** é uma string de formato que pode consistir em texto fixo e especificadores de formato.
- Especificadores de formato iniciam com um sinal de porcentagem (%) seguido por um caractere que representa o *tipo de dados.* Exemplo:
  - **%s** => é um marcador de lugar para uma string.
  - **%n** => espaço (igual ao \n, porém só funciona com printf, não funciona com print nem com println).

Entrada de Dados

- java.lang não tem comando especifico para entrada de dados

```
package teclado;
import java.util.Scanner;

public class Teclado {

    public static void main(String[] args) {
        Scanner teclado = new Scanner (System.in);
        System.out.print ("Digite o nome do aluno ");
        String nome = teclado.nextLine();
        System.out.print ("Digite a nota do aluno ");
        float nota = teclado.nextFloat();
        System.out.print ("Digite a idade do aluno ");
        int idade = teclado.nextInt();
        teclado.close();
  }
```

Tipos Primitivos

- São os tipos de dados que podem:
  - ser apresentados em uma variável, um atributo
  - ser transferidos através de um parâmetro ou retorno de um método

		Valores possíveis				
Tipos	***Primitivo***	***Menor***	***Maior***	***Valor Padrão***	***Tamanho***	***Exemplo***
Inteiro	byte	-128	127	0	8 bits	byte ex1 = (byte)1;
	short	-32768	32767	0	16 bits	short ex2 = (short)1;
	int	-2.147.483.648	2.147.483.647	0	32 bits	int ex3 = 1;
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits	long ex4 = 1l;
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits	float ex5 = 5.50f;

	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits	double ex6 = 10.20d;
						ou
						double ex6 = 10.20;

Caractere	char	0	65535	\0	16 bits	char ex7 = 194; ou char ex8 = 'a';
Booleano	boolean	false	true	false	1 bit	boolean ex9 = true;

Classes wrapper (invólucro)

* As classes wrapper são:

Família	Tipo Primitivo	Classe Invólucro	Tamanho	Exemplo
Lógico	boolean	Boolean	1 bit	true
Literais	char	Character	1 byte	'A'
	–	String	1 byte/ cada letra	"JAVA"
Inteiros	byte	Byte	1 byte	127
	short	Short	2 bytes	32.767
	int	Integer	4 bytes	2.147.483
	long	Long	8 bytes	2^{63}
Reais	float	Float	4 bytes	$3.4e^{+38}$
	double	Double	8 bytes	$1.8e^{+308}$

Declaração de variável

Algoritmos

```
var
idade: inteiro
sal: real
letra: caractere
casado: logico

inicio
idade <- 3
sal <- 1825.54
letra <- "G"
casado <- falso
```

* Em Java: 3 maneiras básicas para fazer a declaração da mesma variável:
* 1ª maneira:

```
int idade = 3;
float sal = 1825.54f;
char letra = 'G';
boolean casado = false;
```

* 2ª maneira (typecast):

```
int idade = (int) 3;
float sal = (float) 1825.54
char letra = (char) 'G';
boolean casado = (boolean) false;
```

* 3ª maneira (classes wrapper):

```
Integer idade = new Integer (3);
Float sal = new Float (1825.54);
Character letra = new Character ('G');
Boolean casado = new Boolean (false);
```

Método Parse

* Converte String em um valor primitivo associado
* **int** i = Integer.parseInt (“4”);
* **double** d = Double.parseDouble (“3.6”);
* **boolean** teste = Bollean.parseBoolean (“true”);

- A linguagem Java é fortemente tipada
  * Existe incompatibilidade entre números <-> string

```
int idade = 30;
String valor = idade; // incompatibilidade
String valor = (String) idade; // typecast tb não funciona
String valor = Integer.toString (idade); // método para converter um inteiro para string
```

```
String valor = “30”;
int idade = valor; // incompatibilidade
int idade = (int) valor; // typecast tb não funciona
int idade = Integer.parseInt (valor);
```

* Tudo o que foi feito para inteiro, também vale para números reais. Exemplo:

```
String valor = “30.5”;
float idade = Float.parseFloat (valor);
```

Operadores Aritméticos

Operação Java	Operador
Adição	+
Subtração	–
Multiplicação	*
Divisão	/
Resto	%

Operadores de Igualdade e Operadores Relacionais

Operador algébrico	Operador de igualdade ou relacional Java	Exemplo de condição em Java	Significado da condição em Java
Operadores de igualdade			
=	==	x == y	x é igual a y
≠	!=	x != y	x é não igual a y
Operadores relacionais			
>	>	x > y	x é maior que y
<	<	x < y	x é menor que y
≥	>=	x >= y	x é maior que ou igual a y
≤	<=	x <= y	x é menor que ou igual a y

Instrução de controle - If

- If / then / else:

```
if (condição) {
        //conjunto de instruções se condição é verdadeira
}
else {
        //conjunto de instruções se condição é falsa
}
```

==	igual a
!=	diferente de
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a
!	não (inverte o resultado de uma expressão booleana)

Instrução de controle – If Exemplo

Exercicio_If1.java > ...

```java
/**
 * Exercicio_If1
 */
import java.util.Scanner;
public class Exercicio_If1 {

    Run | Debug
    public static void main(String[] args) {
        float nota;
        Scanner teclado = new Scanner (System.in);
        System.out.print("Digite a nota final: ");
        nota = teclado.nextFloat();
        teclado.close();
        if (nota >= 5){
            System.out.println("Aprovado!");
        }
    }
}
```

```
C:\Users\Eliana\Desktop\Java>javac Exercicio_If1.java

C:\Users\Eliana\Desktop\Java>java Exercicio_If1
Digite a nota final: 6
Aprovado!
```

Instrução de controle - If Exemplo 2

Exercicio_If1.java > ...

```java
/**
 * Exercicio_If1
 */
import java.util.Scanner;
public class Exercicio_If1 {

    Run | Debug
    public static void main(String[] args) {
        float nota;
        Scanner teclado = new Scanner (System.in);
        System.out.print("Digite a nota final: ");
        nota = teclado.nextFloat();
        teclado.close();
        if (nota >= 5){
            System.out.println("Aprovado!");
        } else {
            System.out.println("Reprovado!");
        }
    }
}
```

```
C:\Users\Eliana\Desktop\Java>java Exercicio_If1
Digite a nota final: 4
Reprovado!

C:\Users\Eliana\Desktop\Java>java Exercicio_If1
Digite a nota final: 6
Aprovado!

C:\Users\Eliana\Desktop\Java>java Exercicio_If1
Digite a nota final: 5
Aprovado!
```

Instrução de controle Operador Condicional

- O operador condicional (**?:**), que pode ser utilizado no lugar de uma instrução if...else. Isso pode tornar o código mais curto e mais claro.
  System.out.println (condição ? “verdadeiro” : “falso”);

Instrução de controle Operador Condicional


```java
/**
 * Exercicio_OpCond
 */
import java.util.Scanner;

public class Exercicio_OpCond {

    Run | Debug
    public static void main(String[] args) {
        Scanner teclado = new Scanner (System.in);
        System.out.print("Digite a nota final:");
        float nota = teclado.nextFloat();
        teclado.close();
        System.out.println(nota >= 5 ? "Aprovado!" : "Reprovado!");
    }
}
```

Instrução de Controle - While

- While:

```
while (condição) {
        //conjunto de instruções
}
```

- Exemplo:

```
a=0;
while (a<3) {
        System.out.println (“TesteA: ” + a);
        a++;
}
```

Instrução de Controle – While


```java
/
 * NotaMediaSala
 */
import java.util.Scanner;
public class NotaMediaSala {

    Run | Debug
    public static void main(String[] args) {
        Scanner input = new Scanner (System.in);
        float total = 0; //inicializa a soma das notas inseridas pelo usuário
        int contador = 1; // inicializa q quantidade de notas a ser inserida em seguida

        while (contador <=10){
            System.out.print("Digite a nota:");
            float nota = input.nextFloat();
            total = total + nota;
            contador = contador + 1;
        }
        input.close();
        float media = total / 10;
        System.out.printf("A media da sala e %.2f%n", media);
    }
}
```

Operadores de Incremento e Decremento

Operador	Nome do Operador	Exemplo	Explicação
++	pré-incremento	++a	Incrementa a por 1, então utiliza o novo valor de a na expressão em que a reside.
++	pós-incremento	a++	Usa o valor atual de a na expressão em que a reside, então incrementa a por 1.
--	pré-decremento	--b	Decrementa b por 1, então utiliza o novo valor de b na expressão em que b reside.
--	pós-decremento	b--	Usa o valor atual de b na expressão em que b reside, então decrementa b por 1.

Operadores de Incremento e Decremento


```
1  // Figura 4.15: Increment.java
2  // Operadores de pré-incremento e de pós-incremento.
3
4  public class Increment
5  {
6     public static void main(String[] args)
7     {
8        // demonstra o operador de pós-incremento
9        int c = 5;
10       System.out.printf("c before postincrement: %d%n", c); // imprime 5
11       System.out.printf("    postincrementing c: %d%n", c++); // imprime 5
12       System.out.printf(" c after postincrement: %d%n", c); // imprime 6
13
14       System.out.println(); // pula uma linha
15
16       // demonstra o operador de pré-incremento
17       c = 5;
18       System.out.printf(" c before preincrement: %d%n", c); // imprime 5
19       System.out.printf("     preincrementing c: %d%n", ++c); // imprime 6
20       System.out.printf("  c after preincrement: %d%n", c); // imprime 6
21    }
22 } // fim da classe Increment
```

Instrução de Controle - For

- For

```
for (variável; condição; operação atribuida) {
        //conjunto de instruções
}
```

- Exemplo:

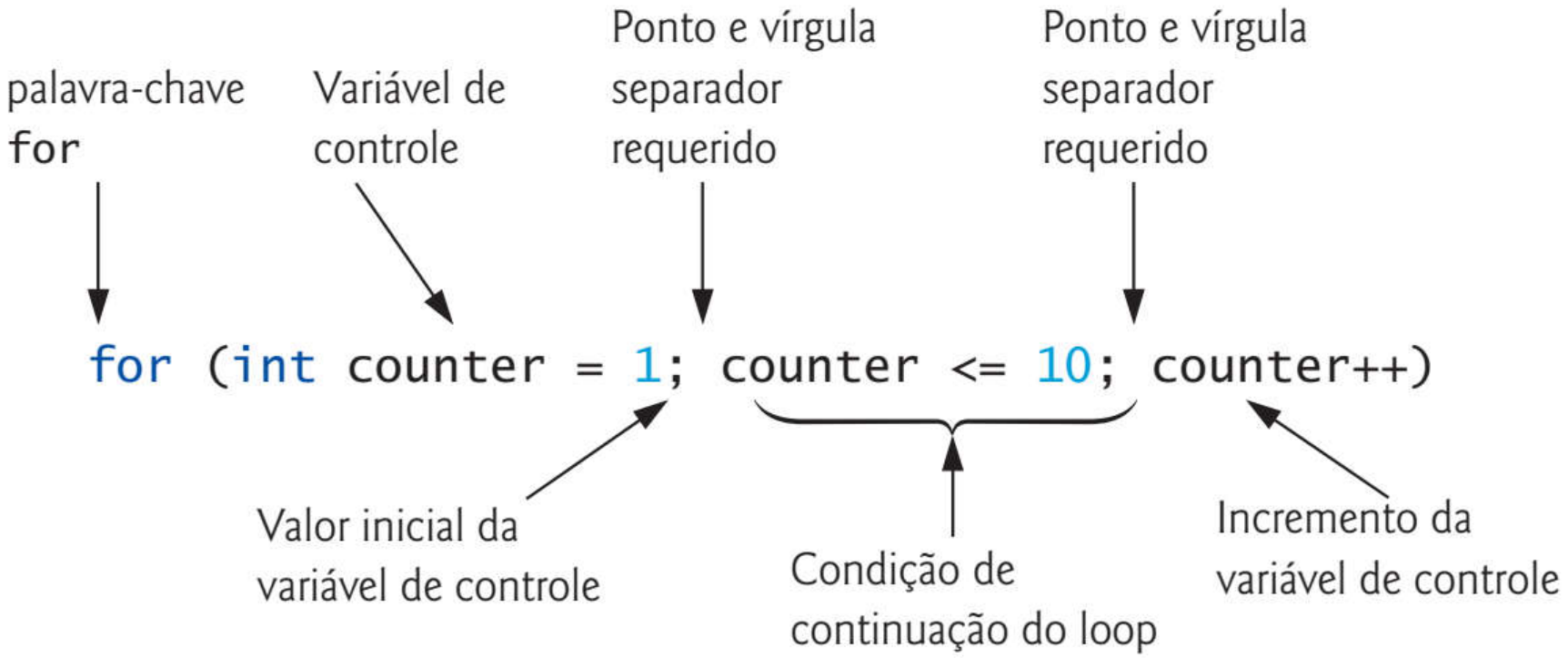
```
for (i=1; i<=10; i++) {
        System.out.println ("Teste: " + i);
}
```

Instrução de Controle – For (cont.)


```
1   // Figura 5.2: ForCounter.java
2   // Repetição controlada por contador com a instrução de repetição for.
3
4   public class ForCounter
5   {
6      public static void main(String[] args)
7      {
8         // o cabeçalho da instrução for inclui inicialização,
9         // condição de continuação do loop e incremento
10        for (int counter = 1; counter <= 10; counter++)
11           System.out.printf("%d  ", counter);
12
13        System.out.println();
14     }
15  } // fim da classe ForCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

Instrução de Controle – For (cont.)



Instruções de Controle – For x While


```
for (inicialização; condiçãoDeContinuaçãoDoLoop; incremento)
    instrução
```

```
inicialização;
while (condiçãoDeContinuaçãoDoLoop)
{
    instrução
    incremento;
}
```

Instrução de Controle – Do While

- Do While:

```
do {
        //conjunto de instruções
} while (condição) ;
```

- Exemplo:

```
do {
        System.out.println (“Teste ”);
} while (false);
```

Instrução de Controle – Do While (cont.)


```
1   // Figura 5.7: DoWhileTest.java
2   // instrução de repetição do...while.
3
4   public class DoWhileTest
5   {
6      public static void main(String[] args)
7      {
8         int counter = 1;
9
10        do
11        {
12           System.out.printf("%d  ", counter);
13           ++counter;
14        } while (counter <= 10); // fim da instrução do...while
15
16        System.out.println();
17     }
18  } // fim da classe DoWhileTest
```

```
1  2  3  4  5  6  7  8  9  10
```

Instrução de Controle – Switch Case

- A instrução de seleção múltipla **switch** realiza diferentes ações com base nos possíveis valores de uma expressão integral constante do tipo *byte, short, int, char* ou *String*.

```
SWITCH (variável) {
CASE valor :
Código a ser executado caso o valor de case seja o mesmo da variável de switch
}
```

- case não gera resultados booleanos, portanto, não há a possibilidade de fazer comparações

Operadores Lógicos

Operador	Significado
&&	E condicional
||	Ou condicional
&	E lógico boleano
|	Ou inclusivo lógico boleano
^	Ou exclusivo lógico boleano
!	Negação

Tipos de exceções

- Erros aritméticos
- Estouro de limite de array
- Entrada de dados inválidos
- Erros na manipulação de arquivos
- Erros na comunicação com bancos de dados
- Falhas de comunicação entre programas distribuídos
- Etc.

Tratamento de exceções

- Palavras chaves em Java:
  - Try
  - Catch
  - Throw
  - Throws
  - Finally
- Vamos focar nas palavras:
  - Try
  - Catch
  - Finally

Tratamento de exceções (cont.)

- Alguns exemplos de exceções já definidas no pacote java.lang incluem:
  - ArithmeticException
  - NumberFormatException
  - IndexOutOfBounds
  - NullPointerException
  - ClassNotFoundException

Capturando um erro genérico


```
try{
        //bloco que é monitorado para erros
}
catch (tipo da exceção1 exception) {
        //tratamento do erro1
}
catch (Throwable exception) {
        //tratamento do erro2
}
```

- Throwable = super classe de todas as exceções e erros no Java

Try-catch-finally


```
try{
        //bloco que é monitorado para erros
}
catch (tipo da exceção1 exception) {
        //tratamento do erro1
}
catch (tipo da exceção2 exception) {
        //tratamento do erro2
}
Finally {
        // executado após o try ou catch
}
```