



# Java Foundations

8-4

Conceitos e Técnicas de Depuração

**ORACLE**  
Academy



Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

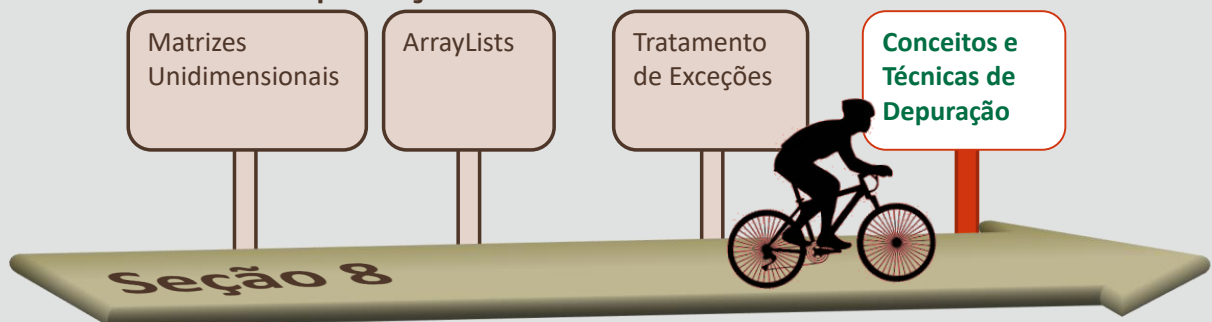
# Objetivos

- Esta lição abrange os seguintes objetivos:
  - Testar e depurar um programa Java
  - Identificar três tipos de erros
  - Aplicar técnicas de depuração
    - Instruções print
    - Depurador do NetBeans
  - Aplicar algumas dicas e técnicas de depuração



# Tópicos

- **Testando e Depurando um Programa Java**
- Três Tipos de Erros
- Técnicas de Depuração: instruções de impressão
- Técnicas de Depuração: Depurador do NetBeans
- Dicas de Depuração



**ORACLE**  
Academy

JFo 8-4  
Conceitos e Técnicas de Depuração

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

# Testando um Programa Java

- Richie criou um programa Java para encontrar o valor máximo entre três inteiros:

```
public static void main(String[] args) {  
    int num1 = 3, num2 = 3, num3 = 3;  
    int max = 0;  
    if (num1 > num2 && num1 > num3) {  
        max = num1;  
    }//fim if  
    if (num2 > num1 && num2 > num3) {  
        max = num2;  
    }//fim if  
    if (num3 > num1 && num3 > num2) {  
        max = num3;  
    }//fim if  
    System.out.println("O máximo de 3 números é " + max);  
}//fim do método main
```

## Testando um Programa Java

- Richie testou-o em muitos conjuntos de dados, como  $\langle 3, 5, 9 \rangle$ ,  $\langle 12, 1, 6 \rangle$  e  $\langle 2, 7, 4 \rangle$
- O programa funciona para todos os dados
- No entanto, ele foi informado que o programa não funciona e não conseguiu descobrir por quê



## Exercício 1

- Importe e abra o projeto DebuggingEx
- Observe `MaxIntegers.java`
  - Você consegue identificar o que Richie esqueceu em seu teste?

## Identificar o Erro

- O programa falha quando é testado com valores duplicados, como  $\langle 3,3,3 \rangle$  e  $\langle 7,2,7 \rangle$ , e exibe a saída como zero
  - Você identificou o erro
  - A próxima etapa é corrigir o erro



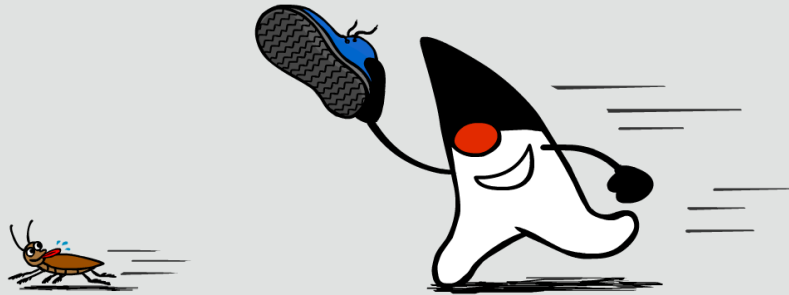
## Corrigir o Erro

- Modifique o programa e teste-o em muitos conjuntos de dados, inclusive em valores duplicados

```
public static void main(String[] args) {  
    int num1 = 3, num2 = 3, num3 = 3;  
    int max = 0;  
    if (num1 > max) {  
        max = num1;  
    }//fim if  
    if (num2 > max) {  
        max = num2;  
    }//fim if  
    if (num3 > max) {  
        max = num3;  
    }//fim if  
    System.out.println("O máximo de 3 números é " + max);  
}//fim do método main
```

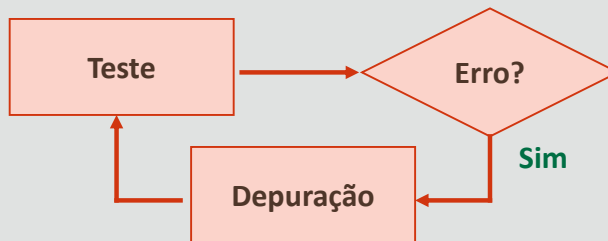
# Testando e Depurando

- Como você observou no exemplo anterior, o teste e a depuração são atividades importantes no desenvolvimento do software



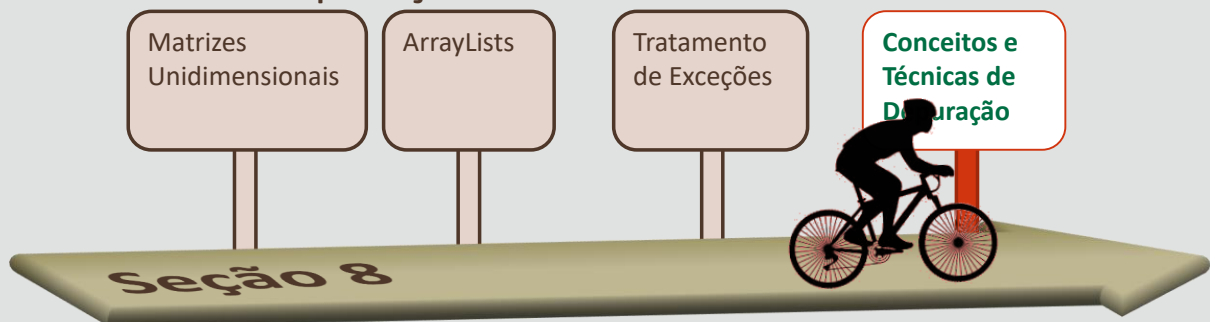
# Testando e Depurando

- Teste:
  - Para determinar se um código contém erros
- Depuração:
  - Para identificar um erro e corrigi-lo



# Tópicos

- Testando e Depurando um Programa Java
- **Três Tipos de Erros**
- Técnicas de Depuração: instruções de impressão
- Técnicas de Depuração: Depurador do NetBeans
- Dicas de Depuração



**ORACLE**  
Academy

JFo 8-4  
Conceitos e Técnicas de Depuração

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

12

# Três Tipos de Erros

- Erros

- Erros de compilação
- Erros lógicos
- Erros de run-time

# Erros de Compilação

- Erro de sintaxe
- Os tipos de erros mais fáceis de serem corrigidos
- Exemplos:
  - Exemplo 1: ausência de ponto e vírgula
    - `int a = 5 // falta o ponto e vírgula`
  - Exemplo 2: erros na expressão
    - `x = ( 3 + 5; // falta o parêntese de fechamento`
    - `y = 3 + * 5; // falta um argumento entre '+' e  
// '*'`

# Erros Lógicos

- O programa é executado, mas produz um resultado incorreto
- Difícil de caracterizar e, portanto, mais difícil ainda de corrigir
- Exemplo: variável não inicializada

- `int i;`
- `i++; // a variável i não foi inicializada`

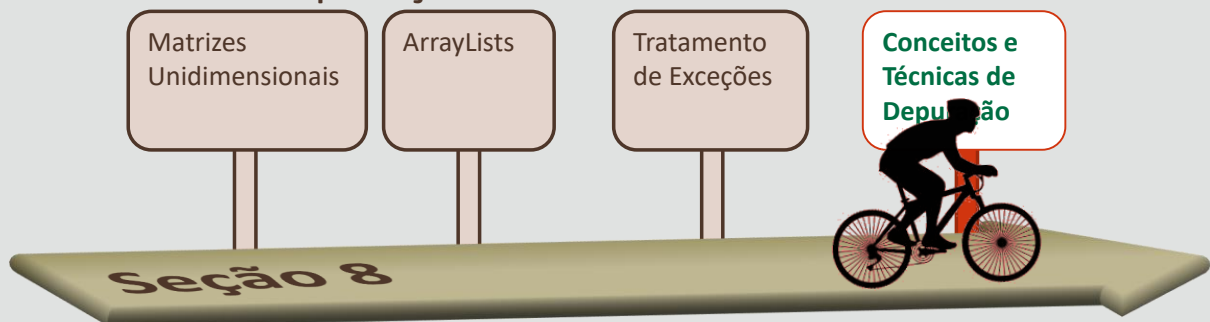
# Erros de Run-time

- Esses erros ocorrem no run-time
- O mecanismo de tratamento de exceções do Java pode detectar esses erros
- Algumas das exceções comuns:
  - `ArrayIndexOutOfBoundsException`
  - `NullPointerException`
  - `ArithmeticException`



# Tópicos

- Testando e Depurando um Programa Java
- Três Tipos de Erros
- **Técnicas de Depuração: instruções de impressão**
- Técnicas de Depuração: Depurador do NetBeans
- Dicas de Depuração



**ORACLE**  
Academy

JFo 8-4  
Conceitos e Técnicas de Depuração

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

17

# Técnicas de Depuração

- Vamos analisar duas técnicas de depuração:
  - Uso de instruções print
  - Uso do depurador do NetBeans

## print: vantagens

- Fáceis de serem adicionadas
- Fornecem informações
  - Quais métodos foram chamados
  - O valor de parâmetros
  - A ordem em que os métodos foram chamados
  - Os valores de campos e variáveis locais em pontos estratégicos

## print: desvantagens

- Não é prático adicionar instruções de print a cada método
- Um número excessivo de instruções print gera uma sobrecarga de informações
- A remoção de instruções print é uma atividade monótona

## print: Exemplo

- Considere este código Java:

```
int n = 10;
int sum = 10;
while (n > 1){
    sum = sum + n;
    n--;
} //fim while
System.out.println("A soma dos inteiros de 1 a 10 é " + sum);
```

- Quando esse programa é executado, ele não funciona corretamente
- Para saber o que está errado, você pode rastrear o valor das variáveis n e sum inserindo instruções print

# Programa Modificado com Instruções print Adicionais para Depuração

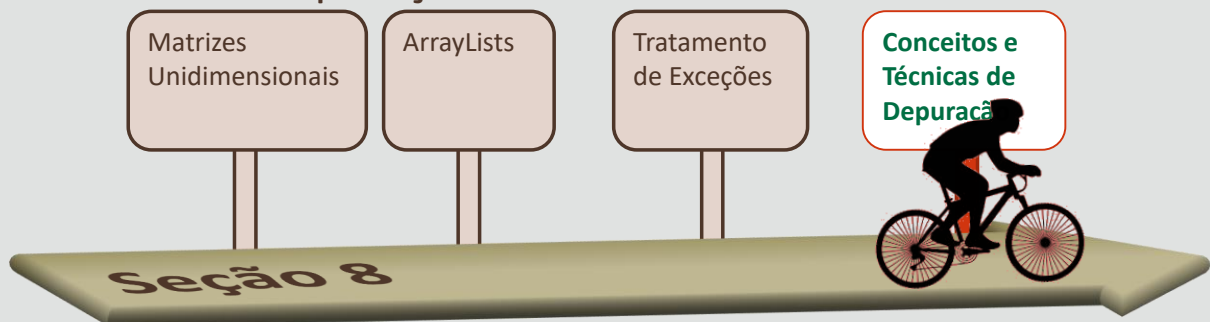
```
int n = 10;
int sum = 10;
while (n > 1) {
    System.out.println("No início do loop: n = " + n);
    System.out.println("No início do loop:sum= " + sum);
    n--;
    System.out.println("No fim do loop: n = " + n);
    System.out.println("No fim do loop: sum = " + sum);
}
System.out.println("A soma dos inteiros de 1 a 10 é " + sum);
```

## Saída

- Veja a seguir as quatro primeiras linhas da saída depois da primeira iteração do loop:
  - No início do loop:  $n = 10$
  - No início do loop:  $\text{sum} = 10$
  - No fim do loop:  $n = 9$
  - No fim do loop:  $\text{sum} = 20$
- Você pode ver que algo está errado:
- A variável `sum` foi definida como 20. Já que ela foi inicializada como 10, é definido  $10 + 10$ , o que será incorreto se você quiser adicionar números de 1 a 10

# Tópicos

- Testando e Depurando um Programa Java
- Três Tipos de Erros
- Técnicas de Depuração: instruções de impressão
- **Técnicas de Depuração: Depurador do NetBeans**
- Dicas de Depuração



**ORACLE**  
Academy

JFo 8-4  
Conceitos e Técnicas de Depuração

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

24



## O Depurador do NetBeans

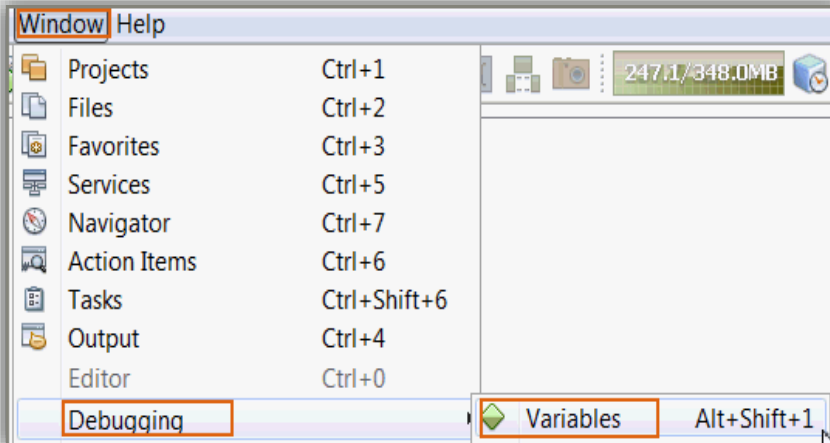
- Você já usou o ambiente gráfico de depuração do NetBeans
- Você usou os seguintes recursos do depurador:
  - Definir pontos de interrupção
  - Percorrer um programa linha a linha
- Vamos usar outro recurso muito importante para exibir o conteúdo de variáveis

## Janela Variables

- Quando atinge um ponto de interrupção definido, você pode usar a janela Variables para ver o valor das variáveis naquele momento
- Você pode descobrir valores de variáveis sem precisar inserir várias instruções print em seu programa

# Acessando a Janela Variables

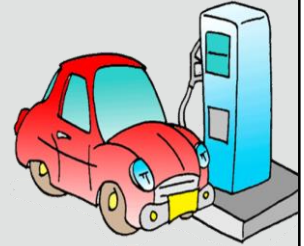
- Para ver a janela Variable, no menu principal do NetBeans:
  - Clique em Window > Debugging > Variables





## Exercício 2: cenário

- Suponha que você tenha um carro e queira ir até o posto de combustível
- Você tem os seguintes detalhes:
  - A posição atual do carro:  $x1$  e  $y1$
  - A localização do posto de combustível:  $x2$  e  $y2$
  - A velocidade do carro
- Você quer calcular o tempo que levará para o carro deslocar-se da posição atual ( $x1$ ,  $y1$ ) e chegar no posto de combustível ( $x2$ ,  $y2$ )
- Um programa Java para calcular o tempo usando a fórmula  $\text{tempo} = \text{distância} / \text{velocidade}$  está disponível no projeto ComputeTime





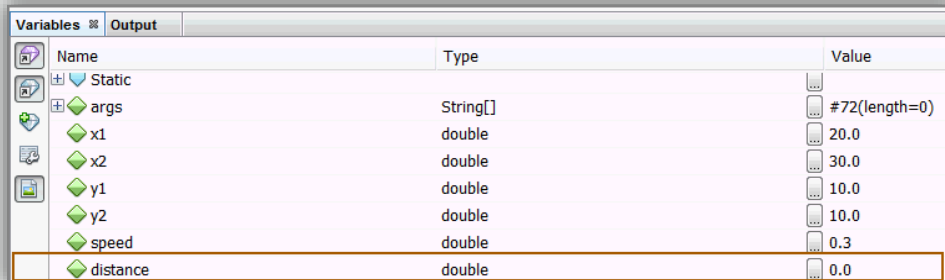
## Exercício 2

- Importe e abra o projeto `DebuggingEx`
- Examine `ComputeTime.java`
- Execute o programa com o depurador do NetBeans para depurar este programa:
  - Defina o ponto de interrupção no método `getDistance`
  - Pressione `Step In` para passar para a linha seguinte
  - Observe os valores das variáveis `x1`, `x2`, `y1`, `y2`, `speed`, `distance` e `time`
- Você consegue identificar o bug?



## Observe o Valor de distance

- No exercício anterior usando os recursos de depuração do NetBeans, você identificou o bug:



Name	Type	Value
Static		
args	String[]	#72(length=0)
x1	double	20.0
x2	double	30.0
y1	double	10.0
y2	double	10.0
speed	double	0.3
distance	double	0.0

- Como você pode ver, distance é 0
- A fórmula para calcular a distância estava errada e isso causou um valor de retorno incorreto para a variável distance

# Identificando o Possível Bug

```
public static void main(String[] args) {  
    double x1 = 20;  
    double x2 = 30;  
    double y1 = 10;  
    double y2 = 10;  
    double speed = 0.3;  
    double distance = getDistance(x1, x2, y1, y2);  
    double time = distance/speed;  
    System.out.println("Time taken to reach the gas station is " + time);  
  
} //fim do método main  
  
static double getDistance(double x1, double x2, double y1, double y2){  
    return Math.sqrt((x1 - x1) * (x1 - x2) + (y1 - y2) * (y1 - y2));  
} //fim do método getDistance
```

Possível bug

## Corrigindo o Bug

- Como identificou o bug, você pode alterar a localização do ponto de interrupção para o local em que o método `getDistance()` é chamado
- Dessa forma, você não precisa percorrer o código que já analisou
- Então, vamos modificar o código e executar o depurador novamente com o novo ponto de interrupção para ver o que obtemos



# Executando Novamente o Depurador

```
public static void main(String[] args) {  
    double x1 = 20;  
    double x2 = 30;  
    double y1 = 10;  
    double y2 = 10;  
    double speed = 0.3;  
    double distance = getDistance(x1, x2, y1, y2);  
    double time = distance/speed;  
    System.out.println("Time taken to reach the gas station is " + time);  
  
} //fim do método main  
  
static double getDistance(double x1, double x2, double y1, double y2){  
    return Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));  
} //fim do método getDistance
```

Novo ponto de interrupção

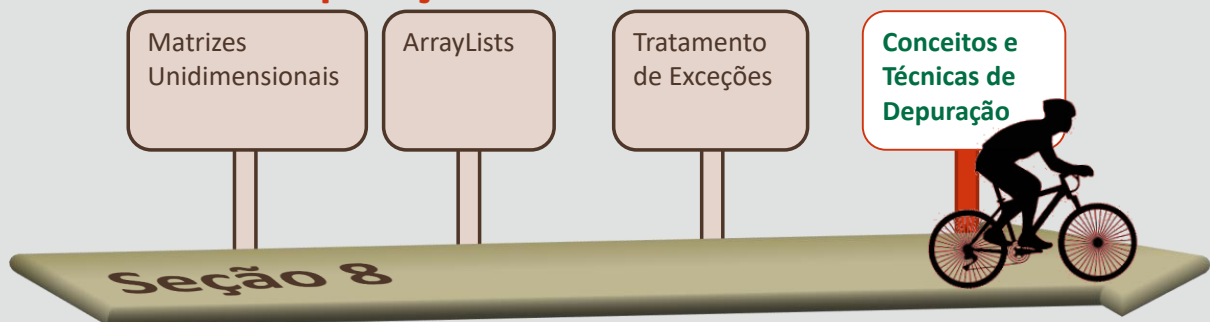
Código  
modificado

# Observando as Variáveis

Variables		Output
Name	Type	Value
args	String[]	#/2(length=0)
x1	double	20.0
x2	double	30.0
y1	double	10.0
y2	double	10.0
speed	double	0.3
distance	double	10.0
time	double	33.33333333333336

# Tópicos

- Testando e Depurando um Programa Java
- Três Tipos de Erros
- Técnicas de Depuração: instruções de impressão
- Técnicas de Depuração: Depurador do NetBeans
- **Dicas de Depuração**



**ORACLE**  
Academy

JFo 8-4  
Conceitos e Técnicas de Depuração

Copyright © 2020, Oracle e/ou suas empresas afiliadas. Todos os direitos reservados.

35

## Operador com um sinal de igual x dois sinais de igual

- Operador de Atribuição (=) x Operador de Comparação (==)

- 1. Operador de comparação

- `if( x = 0 )`, em vez de `if(x == 0)`
    - Observe-o nas instruções `if`, `for` e `while`

- 2. Operador de atribuição

- `int x == 1; ,` em vez de `int x = 1;`

## Ponto e Vírgula na Posição Incorreta

- Verifique o ponto e vírgula depois da instrução if ou das instruções de loop for/while

```
if (x == 0); {  
  <instruções>  
}
```

em vez de

```
if(x == 0) {  
  <instruções>  
}
```

```
while(<expressão booliana>); {  
  <instruções>  
}
```

em vez de

```
while(<expressão booliana>) {  
  <instruções>  
}
```

# Chamando Métodos com Argumentos Errados

- Os tipos de parâmetros de chamadas de método devem coincidir com os tipos de parâmetros de definição
- Por exemplo:
  - Dada a definição do método:
  - `void methodName(int x, char y) { }`
  - Chame este método:
  - `methodName(a, b)`



a deve ser int e b deve ser char

# Condições Limite

- É importante testar as condições limite
- A lógica por trás desses testes é que os erros tendem a ocorrer perto dos valores limite de uma variável de entrada
- Por exemplo, a condição limite para:
  - Dados de entrada (testar com dados válidos x dados inválidos)
  - Loops (início e fim de loops)

## Testando Condições Limite para Loops

- Isso permite que testes de caso limite como “menor que” e “maior que” para condições de iteração de loop sejam realizados com precisão
- Por exemplo, dado este loop:

```
if ( num >= 50 && num <= 100 ) {  
    //fazer testes  
} //fim if
```

- Para testar condições limite, você faria testes com números próximos de 50 e 100, ou seja, 49, 50, 51, 99, 100 e 101





## Exercício 3

- Importe e abra o projeto DebuggingEx
- Observe `BoundaryTesting.java`
- Valide a entrada executando o programa com os seguintes valores de teste limite do ano e mês:

Ano	Mês
1582	2
1583	0
1583	13
1583	1
1583	12

# Resumo

- Nesta lição, você deverá ter aprendido a:
  - Testar e depurar um programa Java
  - Identificar três tipos de erros
  - Aplicar técnicas de depuração
    - Instruções print
    - Depurador do NetBeans
  - Aplicar algumas dicas e técnicas de depuração



