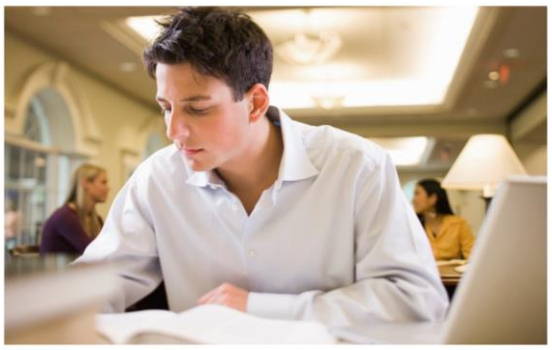

ORACLE
Academy

Java Fundamentals

4-3

Tipos de dados e operadores



ORACLE
Academy

Objetivos

- Esta aula abrange os seguintes tópicos:
  - Usar tipos de dados primitivos em código Java
  - Especificar literais dos tipos primitivos e das Strings
  - Demonstrar como inicializar variáveis
  - Descrever as regras do escopo de um método
  - Reconhecer quando uma expressão exige uma conversão de tipo

Visão geral

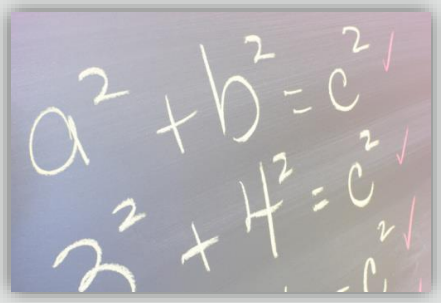
- Esta aula abrange os seguintes tópicos:
  - Aplicar casting no código Java
  - Usar operadores aritméticos
  - Usar o operador de atribuição
  - Usar um método da classe Math
  - Acessar um método da classe Math da API do Java

Tipos de Programação Java

- Em Java, tipos de dados:
  - São usados para definir o tipo de dados que podem ser armazenados em uma variável
  - Verificam se somente os dados corretos são armazenado.
  - São declarados ou inferidos
  - Podem ser criados pelo programador

Variáveis Devem Ter Tipos de Dados

- Todas as variáveis devem ter um tipo de dado por razões de segurança
- O programa não será compilado se o usuário tentar armazenar dados de um tipo incorreto
- Os programas devem aderir às restrições de tipo para serem executados:
  - Tipos incorretos em expressões ou dados são marcados como erros no tempo de compilação

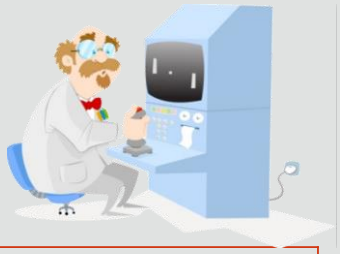


Tipos de Dados Primitivos

- O Java tem oito tipos de dados primitivos que são usados para armazenar dados durante uma operação de programação
- Tipos de dados primitivos são um grupo especial de tipos de dados que não usam a palavra-chave nova quando inicializados
- O Java cria-os como variáveis automáticas que não são referências, as quais são armazenadas na memória com o nome da variável
- Os tipos primitivos mais comuns usados neste curso são int (inteiros) e double (decimais)

Os tipos primitivos são simples e armazenam apenas um valor, enquanto um objeto é mais complexo e pode armazenar muitos valores (campos).
inteiro e duplo são os tipos primitivos mais comuns usados em Java. valores booleanos também são muito usados.

Tipos de Dados Primitivos

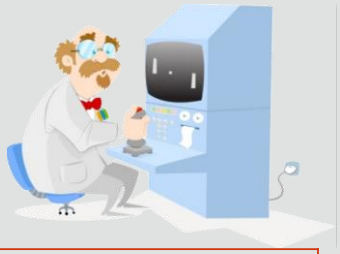


Tipo de Dado	Tamanho	Exemplo de Dados	Descrição dos Dados
boolean	1 bit	true, false	Armazena sinalizadores de true e false.
byte	1 byte (8 bits)	12, 128	Armazena inteiros de -128 a 127.
char	2 bytes	'A', '5', '#'	Armazena um caractere Unicode de 16 bits de 0 a 65.535
short	2 bytes	6, -14, 2345	Armazena inteiros de -32.768 a 32.767.

ORACLE Academy

Dependendo do sistema operacional do computador com o qual o Java esteja sendo usado, os tipos de dados menores poderão, na verdade, usar 4 ou até 8 bytes de memória real.

Tipos de Dados Primitivos



Tipo de Dado	Tamanho	Exemplo de Dados	Descrição dos Dados
int	4 bytes	6, -14, 2345	Armazena inteiros de: -2.147.483.648 a 2.147.483.647
long	8 bytes	3459111, 2	Armazena inteiros de: - 9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
float	4 bytes	3.145, .077	Armazena um número decimal positivo ou negativo de: $1.4023x10^{-45}$ a $3.4028x10^{+38}$
double	8 bytes	.0000456, 3.7	Armazena um número decimal positivo ou negativo de: $4.9406x10^{-324}$ a $1.7977x10^{308}$

Declarando Variáveis e Usando Literais

- A nova palavra-chave não é usada ao inicializar uma variável de um tipo primitivo.
- Em vez disso, um valor de literal deve ser designado a cada variável na inicialização
- Um literal pode ser qualquer número, texto ou outra informação que represente um valor
- Exemplos de declaração de uma variável e designação de um valor de literal a ela:

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

Declarando Variáveis e Usando Literais - Exemplo

- Os valores de d1 e d2 são iguais
- A inicialização de d2 mostra como a notação científica pode ser usada para definir o valor
- Total e ss_num recebem o mesmo valor
- A inicialização de ss_num mostra que sublinhados podem ser usados para separar números para legibilidade

```
long total=999999999;
long ss_num = 999_99_9999;
double d1 = 123.4;
double d2 = 1.234e2;
float f1  = 123.4f;
```


Em geral, o Java faz muita distinção entre maiúsculas e minúsculas. No entanto, você pode usar as letras maiúsculas ou minúsculas e, f ou l ao trabalhar com valores literais e notação científica.

Exemplos de Literais Numéricos

- 0x e 0b são usados para denotar um valor de literal hexadecimal ou um valor de literal binário
- Os literais melhorarão o desempenho do processo

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_fff_fff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Para se divertir, de quais outras palavras Hex (somente as letras A-F), como DEAD_BEEF, você pode cria?

Literais Binários

- Literais binários podem ser expressados usando o sistema binário adicionando os prefixos 0b ou 0B ao número
- Literais binários são valores Java int
- Valores de byte e curto Java requerem um casting para impedir um erro de perda de precisão do compilador

Um casting sinaliza um tipo de conversão. Se você aplicar casting de um tipo em outro tipo, você está explicitamente convertendo o item do seu tipo original para o tipo especificado. Um casting não arredonda um número decimal, mas trunca-o. Um exemplo de aplicação de casting em um double como um int é:

```
double x = 5.745;
int y = (int)x;  //y is now equal to 5
```

ORACLE Academy

Observe que a conversão de float ou duplo em um tipo de valor inteiro trunca o valor. O valor não é arredondado.

Para que os Literais Binários São Usados

- Literais binários são usados para:
  - Cálculos
  - Comparações
  - Programação de baixo nível, como:
    - Gravação de drivers de dispositivo
    - Gráficos de baixo nível
    - Montagem do pacote do protocolo de comunicações
    - Decodificação

Por que Usar Literais Binários?

- Usar Literais Binários para representar valores para comparações e cálculos é substancialmente mais rápido do que usar valores do tipo de dados reais
- Processadores de alto desempenho modernos geralmente fazem cálculos em inteiros tão rápido quanto o uso de literais binários, então, por que usar literais?
- Ainda é o ideal usar literais para o poder e o desempenho geral, pois eles usam menos recursos

Exemplo de Casting

```
// An 8-bit 'byte' value:
byte aByte = (byte)0b00100001;

// A 16-bit 'short' value:
short aShort = (short)0b1010_0001_0100_0101;

// Some 32-bit 'int' values:
int anInt1=0b1010_0001_0100_0101_1010_0001_0100_0101;
int anInt2=0b101;
int anInt3=0B101;   //The B can be upper or lower case
```

Este é um exemplo de casting. Neste exemplo, o valor binário o valor é convertido para um tipo de byte.

Regras para Nomes de Variáveis

- Siga as próximas regras ao escolher o nome de uma variável:
  - Não use uma palavra reservada ou palavra-chave Java
  - Não use espaço no nome da variável
  - Use uma combinação de letras ou uma combinação de letras e números
  - não é possível começar com um número
  - Os únicos símbolos permitidos são o sublinhado (_) e o sinal de dólar (**$**)

Os elementos _ e $ em identificadores Java raramente são usados.

Convenções para Nomes de Variáveis

- Embora as convenções não sejam regras, a maioria dos programadores Java seguem essas convenções:
  - Use palavras completas em vez de abreviações criptografadas
  - Não use variáveis de uma única letra.
  - Se todas as variáveis tiverem uma única letra, o código poderá parecer muito confuso
    - Uma exceção a esta convenção é para as variáveis de controle de loop, que são geralmente as letras i, j ou k
  - Se um nome de variável consistir em uma palavra, escreva essa palavra com todas as letras minúsculas
  - Se um nome de variável consistir em mais de uma palavra, use lowerCamelCase

Usem bons nomes de variáveis!

Convenções de Nomenclatura Adicionais para Nomes de Variáveis

- Convenções de nomenclatura adicionais:
  - Se uma variável for um valor constante, use todas as letras maiúsculas e separe-as com o sublinhado
  - Use nomes que expressem a finalidade da variável
  - No exemplo a seguir, PI é uma boa escolha para nomear esse número, pois permite que você lembre o que é a variável

```
double PI = 3.14159;
```

Escopo da Variável

- O escopo é usado para descrever o bloco de código, no qual existe uma variável em um programa
- É possível existir múltiplas variáveis com o mesmo nome em um programa Java
  - Na maioria dos casos, a variável mais interna tem precedência
- Uma variável existe somente dentro do bloco de código no qual é declarada
- Quando a chave final do bloco } é atingida:
  - A variável sai do escopo
  - Ela não é mais reconhecida como uma variável declarada

Quando uma variável é declarada, é alocada memória para ela. Quando a variável sai do escopo, a memória é liberada e o valor é perdido.

Exemplo 1 de Escopo de Variável

- No exemplo a seguir, o nome não será exibido, pois ele deixará de existir quando a chave que marcou o Ponto B é atingida

```
public void someMethod()
{
   if(gameOver && score>highScore)
   {
      String name;                  //Point A
      System.out.println("Please enter your name:");
      name=reader.next();
   }//fim if                        //Point B

   System.out.println("Thank you " + name + ", ");
   System.out.println("your high score has been saved.");
}//fim do método someMethod
```


Não só o nome não será impresso, mas o programa não será compilado.

Exemplo 2 de Escopo de Variável

- Neste exemplo, o nome da variável foi movido para fora do bloco de instrução para que possa ser usado em todo o método

```
public void someMethod()
{
   String name;                    //Point A
   if(gameOver && score>highScore)
   {
      System.out.println("Please enter your name:");
      name=reader.next();
   }//fim if                       //Point B
   System.out.println("Thank you " + name + ", ");
   System.out.println("your high score has been saved.");
} //fim do método someMethod
```

Exemplo 3 de Escopo de Variável

- O Java permitirá que uma variável de classe e uma variável de método com o mesmo nome existam em um programa
- Você pode prever o que este programa exibirá?

```
public class Counting{
  public static int counter=5;
  public static void main(String[] args){
    System.out.println("At the start of this program, counter is "+ counter);
    count();
    System.out.println("At the end of this program, counter is "+ counter);
  }//fim do método main
  public static void count(){
    int counter=10;
    System.out.println("At the end of this method, counter is "+ counter);
  }//fim do método count
}//fim da classe Counting
```


Embora esse exemplo seja compilado e executado, pode ser muito confuso. Se for necessário ter variáveis de local e campo com o mesmo nome, seu escopo deverá ser bem documentado com comentários. Seria melhor ainda usar nomes diferentes e evitar a confusão.

Solução do Exemplo 3 de Escopo de Variável

- Solução:
  - 5, 10, 5
- O programa inicia o main () e exibe o contador de variável de classe global
- O método count() é chamado e um novo contador de variável local é criado para essa chamada de método
- É fornecido o valor 10 e, depois, exibido
- Quando a chave no final de count() for atingida, a variável local sairá do escopo (deixará de existir)
- O programa retorna para o método main() e exibe o contador de variável global que não mudou seu valor

Operadores boolianos

- Os operadores boolianos são um conjunto de operadores que podem ser usados para comparar expressões com verdadeiro ou falso

Operador	Operador	Descrição	Exemplo
&&	E	Se os dois forem verdadeiros, retornará verdadeiro	(A&&B)
||	OU	Se um deles for verdadeiro, retornará verdadeiro	(A||B)
!	NÃO	Inverte o estado de lógica (V para F, F para V)	!(A&&B)
==	igual a	Se os dois forem iguais, retornará verdadeiro	(A==B)
!=	diferente de	Se os dois forem diferentes, retornará verdadeiro	(A!=B)
>	maior que	Se o esquerdo for maior que o direito, retornará verdadeiro	(A>B)
>=	maior que ou igual a	Se o esquerdo for maior que ou igual ao direito, retornará verdadeiro	(A>=B)
<	menor que	Se o esquerdo for menor que o direito, retornará verdadeiro	(A<B)
<=	menor que ou igual a	Se o esquerdo for menor que ou igual ao direito, retornará verdadeiro	(A<=B)

Operadores Aritméticos

- O Java tem vários operadores aritméticos para executar operações matemáticas

Símbolo	Descrição do Operador
+	Operador de adição
-	Operador de subtração
*	Operador de multiplicação
/	Operador de divisão (encontra o quociente)
%	Operador modular (encontra o restante)
+ +	Operador de incremento (adiciona um) É um operador unário.
- -	Operador de decremento (subtrai um) É um operador unário.

Em código Java, não há espaço entre + ou -.

Precedência de Operadores Aritméticos

- Todas as expressões matemáticas são avaliadas seguindo a ordem de precedência:
  - As expressões entre parênteses são tratadas primeiro
  - Todas as operações de multiplicação, divisão e modulares são tratadas a seguir, trabalhando da esquerda para a direita
  - Finalmente, todas as operações de adição e subtração são tratadas, trabalhando da esquerda para a direita

Uma tabela de precedência completa:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Incrementos e Decrementos

- Incrementos e decrementos são tratados primeiro para a notação pré-incremento e por último para a notação pós-incremento
- Incremento em Java significa adicionar um ao valor da variável
- Decremento em Java significa subtrair um da variável
- Notação pré-incremento:

```
++x;
```

- Notação pós-incremento:

```
x++;
```

Exemplo de Precedência de Incremento e Decremento

- Notação pré-incremento:

```
int x = 3;
++x;          //x is equal to 4
z = ++x;      //x is equal to 4, THEN z is equal to 5
```

- Notação pós-incremento:

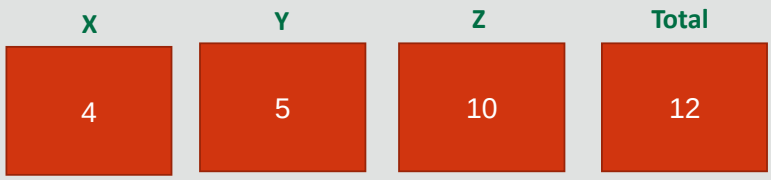
```
int x = 3;
x++;          //x is equal to 4
z = x++;      //z is equal to 4, THEN x is equal to 5
```


Você pode criar as mesmas instruções neste slide sem usarem o operador ++?

Operador de Atribuição

- O Java usa o = (sinal de igual) como operador de atribuição A avaliação da expressão à direita é designada ao local da memória à esquerda

```
int x = 4;
int y = 5;
int z = 10;
int total = 12;
```

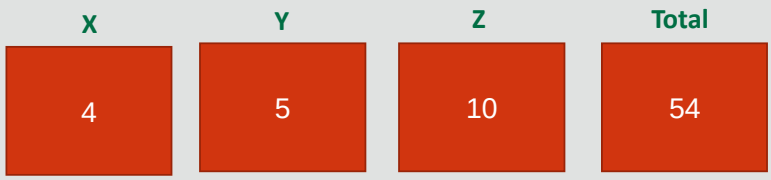


ORACLE Academy

Exemplo 1 de Operador de Atribuição

- Quando esta linha de código é executada, o valor total muda
- As caixas mostram o que está em cada local da memória para x, y, z e total
- Agora, o total do local da memória receberá o valor 4 + 5 * 10, que é 54

```
int x = 4, y = 5, z = 10;
int total = x + y * z;
```



ORACLE Academy

Exemplo 2 de Operador de Atribuição

- Pense no operador de atribuição como uma seta apontando para a esquerda
- Tudo à direita irá para o local da memória à esquerda
- Como a memória mudará quando esse código for executado?

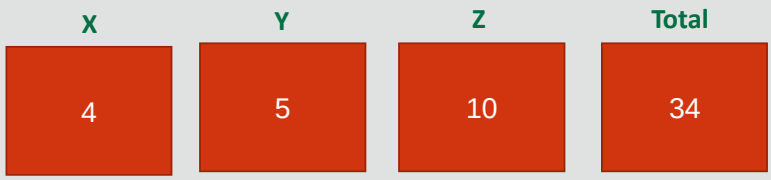
```
int x = 4, y = 5, z = 10;
int total = x + y * (z - x);
```

ORACLE Academy

Solução do Exemplo 2 de Operador de Atribuição

- A resposta é que o total receberá o valor da expressão
- Isso significa que o valor da expressão será armazenado no endereço da memória associado ao total da variável

```
int x = 4, y = 5, z = 10;
int total = x + y * (z - x);
```



ORACLE Academy

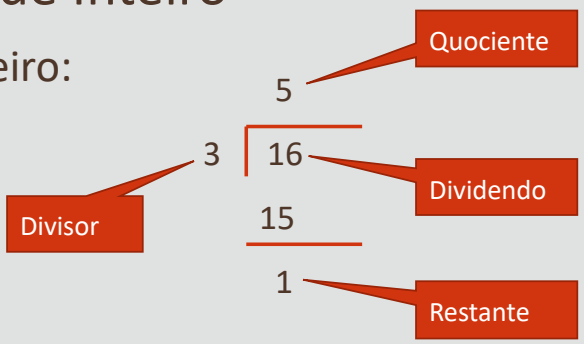
Truncamento e Divisão de Inteiro

- A divisão de dois inteiros SEMPRE produzirá um inteiro
- Por exemplo, a fórmula do volume de um cone de Geometria é:
  - V = ⅓ r2 h
  - Se 1/3 for usado em uma expressão Java, ele será avaliado como 0, devido a divisão do inteiro
  - Os resultados da divisão de inteiros são o quociente sem decimais

Truncamento é o conceito de remover a parte fracionária ou decimal de um número. Por exemplo: O truncamento de 7,8 produzirá o resultado 7, e o truncamento de -3,2 produzirá o resultado -3.

Truncamento e Divisão de Inteiro

- Exemplos de Divisão de Inteiro:
  - 16 / 3 = 5 (Truncamento)
  - 10 / 3 = 3 (Truncamento)
  - 16 % 3 = 1
  - 12 % 3 = 0
  - 11 % 3 = 2



- Para calcular um quociente, sem truncamento (resultados decimais), converta o dividendo ou o divisor para um decimal
- Por exemplo:
  - 11 / 5,0 = 2,2 e, do mesmo modo, 11,0 / 5 = 2,2

Além de adicionar .0 aos literais inteiros para transformá-los em duplos, também é possível converter os valores inteiros.

Exemplo 1 de Truncamento e Divisão de Inteiro

- O que é exibido quando o código a seguir é executado?

```
int x = 4, y = 5, z = 10;
int total = z / (x * y);

System.out.println("The total is " + total + ".");
```

- A resposta é 0
- Por que não exibe 0,5?
- Como o total é um inteiro, o sistema armazenará um valor inteiro como o resultado do cálculo
- A parte decimal da resposta será truncada em vez de arredondada para produzir a resposta de inteiro final 0

Exemplo 2 de Truncamento e Divisão de Inteiro

- Um programador pode escrever o seguinte código para criar um programa para calcular o volume
- O programa Java calculará incorretamente a resposta como 0
- Trabalhando da esquerda para a direita, o programa divide 1 por 3.
- O Java considera 1 e 3 como inteiros literais e faz a divisão de inteiro, onde 0,33... é truncado para 0
- Como você corrigiria isso?

```
double height = 4, radius = 10, volume;
volume = 1 / 3 * 3.14 * radius * radius * height;
System.out.println("The volume is " + volume + ".");
```

Noções Básicas sobre Tipos e Conversões

- Existem algumas formas de impor uma fórmula para não truncar um valor:
  - Mova a fração até o fim para que o Java sempre use um double e um integer e converta implicitamente a resposta para um double, não truncado

```
double volume = 3.14 * radius * radius * height * 1 / 3;
```

  - Transforme um dos integers de literais em um double de literal de modo que o Java sempre use um double e um integer e converta implicitamente a resposta em um double, não truncado

```
double volume = 1 / 3.0 * 3.14 * radius * radius * height;
```

Conversões de Tipo Implícitas

- No exemplo anterior, o Java fez conversões de tipo implícitas
- Isso acontece sempre que um tipo de dado menor (como int) é colocado em um tipo maior (como double)
- O Java percebe que os tipos são diferentes e faz automaticamente a conversão para o tamanho maior para você
- No entanto, o Java não converterá de um tamanho maior (como double) para um tamanho menor (como int) automaticamente

Usando Casting de Tipo

- Usando o método aleatório da biblioteca de Matemática, podemos gerar um número aleatório entre 1 e 10
- O método aleatório gera um double entre 0 e (não incluindo) 1
- Valores como 0, 0,4567 ou 0,901306 podem ser gerados

```
int number;
number = Math.random() * 10;
System.out.println("The random number is " + number + ".");
```

Usando Casting de Tipo

- Multiplicar esses valores por 10 e, em seguida, truncar o adicional resultará nos valores 0, 4 ou 9
- No entanto, o Java não deixará este programa compilar em seu estado atual
- Os dados são perdidos, indo de um valor maior (double) a um valor menor (int)
- Desse modo, o tipo de casting é necessário para esse tipo de conversão

```
int number;
number = Math.random() * 10;
System.out.println("The random number is " + number + ".");
```

Operador de Casting de Tipo

- Para converter um valor duplo em inteiro, use (int) na frente do valor
- Para que o resultado de double de nossa fórmula vá para o contêiner de integer, use o operador (int) de casting de tipo na frente do valor
- A aplicação de casting no tipo de dado truncará o valor
- Desse modo, a aplicação de casting no literal double 4.567 para um int resultará em 4, e 9.01306 resultará em 9

```
int number;
number = (int)(Math.random() * 10);
System.out.println("The random number is " + number + ".");
```

Convertendo Tipos de Dados

- É possível converter um tipo de dado (primitivo ou de referência) em outro tipo de dado, apenas colocando o nome do tipo de dado entre parênteses na frente do valor ou da variável, conforme exibido no exemplo a seguir

```
int number;
Object o;
char firstInitial = 'A';
number = (int)firstInitial;
o = (Object)firstInitial;
```


Os objetos relacionados por herança também pode ser convertidos. Isso será abordado posteriormente neste curso.

Convertendo Tipos de Dados de String

- Observe que o casting não funcionará em todas as situações
- Por exemplo
  - Converter um char em uma String resultará em um erro do compilador
  - Em situações como essa, é necessário reclassificar para fazer a conversão de tipo de outra maneira
  - Há métodos na biblioteca java.lang para converter caracteres em strings

Usando Conversões de Tipo

- Usar conversões de tipo é outra opção para corrigir o problema de truncamento com a fórmula do volume mostrado anteriormente
- Use o casting de tipo para transformar um dos integers literais em um double

```
double volume = (double) 1 / 3 * 3.14 * radius * radius * height;
```

Noções Básicas sobre Tipos e Conversões

- Quando o Java está convertendo de um tipo primitivo menor para um tipo primitivo maior, a conversão é implícita


```
double d1  = 5;
```

Noções Básicas sobre Tipos e Conversões

- No entanto, quando o Java converte de um tipo primitivo maior para um tipo primitivo menor, a conversão deve ser explícita por meio do casting de tipo
- O Java não converterá implicitamente um tipo maior em um tipo menor devido à perda de precisão


```
int x  =
(int)12.95;
```

Pesquisando na API do Java

- Os exemplos e exercícios deste curso exigirão o uso dos métodos nas classes Java Math e String
- Você encontra uma descrição de todos os métodos Java na API do Java on-line
- Compreender como navegar nessa vasta biblioteca de métodos e classes padrão vai ajudá-lo a escrever programas em Java e a reutilizar blocos de código que já foram criados por outros

Por que Usar a API do Java?

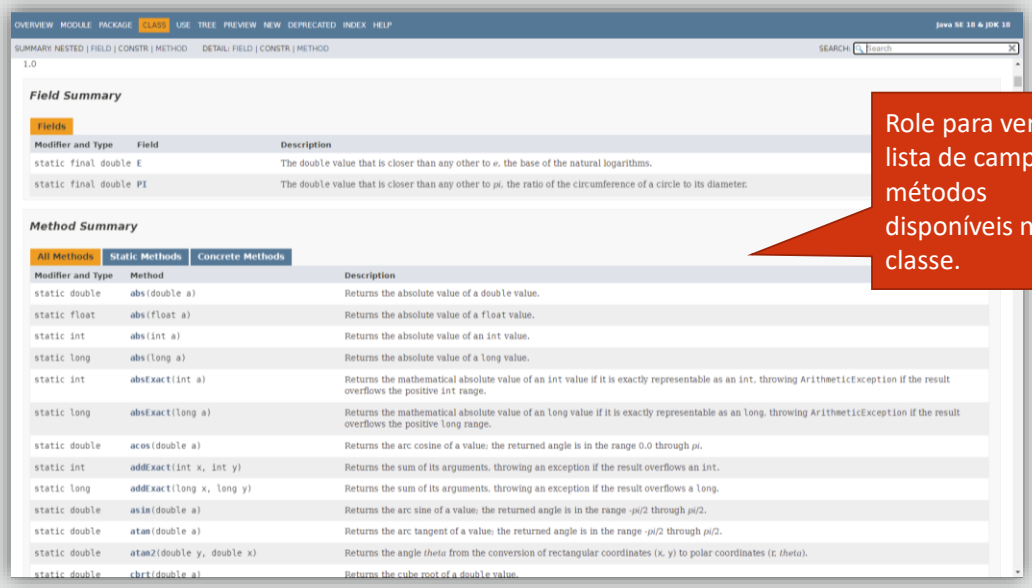
- Um dos principais benefícios de ter acesso à API do Java é um conceito comum para programadores chamado reutilização de código
- Em vez de codificar itens de excesso, você pode usar a API para saber como acessar o código existente, que faz exatamente o que você quer
- Isso reduzirá o tempo gasto com a reprodução de código já existente

Revisar a API do Java

- Volte à API do Java, usando um mecanismo de busca para procurá-la
- Há muitas edições. Revise a Standard Edition para Java 18: https://docs.oracle.com/en/java/javase/18/docs/api/allclasses-index.html
  - Examine a class Math
  - Veja se você pode encontrar um valor para PI e um método para calcular a raiz quadrada de um número

Classe Math

- Encontre a Classe Math na janela de quadros da classe

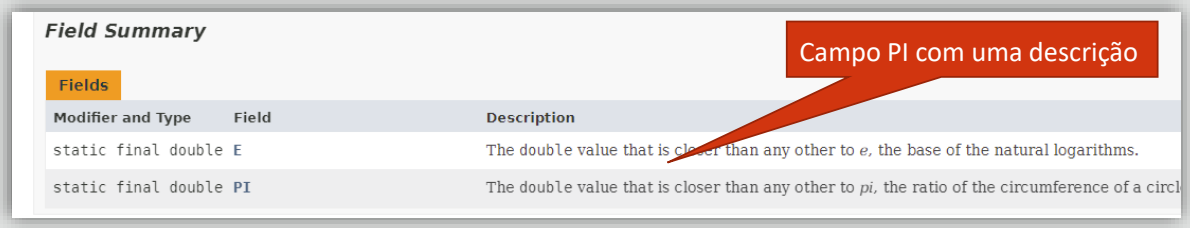


Role para ver uma lista de campos e métodos disponíveis nessa classe.

Campo PI

- Role para encontrar o campo PI
- Para usar PI em um programa, especifique o nome da classe (Math) e PI separados pelo operador ponto
- Por exemplo, esse campo renderá um cálculo de volume mais preciso no exemplo anterior, se for usado da seguinte maneira:

```
double volume = (double) 1 / 3 * Math.PI * radius * radius * height;
```



Calculando Raízes Quadradas

- O método de cálculo de raízes quadradas

`static double`	**`sqrt`**`(double a)` Returns the correctly rounded positive square root of a `double` value.

- Para calcular a raiz quadrada de 625, use o nome da classe e o método separado novamente pelo operador ponto

```
double answer = Math.sqrt(625);
```

- O método sqrt requer um double
- Por que o integer literal 625 não dá erro?

Se tiver tempo, explore o que acontece quando é solicitado ao Java calcular a raiz quadrada de um número negativo. As exceções serão abordadas na Seção 6.

Calculando Raízes Quadradas - Solução

- Pergunta:
  - O método sqrt requer um double
  - Por que o integer literal 625 não dá erro?
- Solução:
  - Conversão implícita
  - O int 625 é implicitamente convertido em um double e, assim, não ocorrem erros

Prática de Operadores e Tipos de Dados

- No papel, avalie as seguintes instruções Java e registre o resultado

```
double x = 3.25;
double y = -4.5;
int m = 23;
int n = 9;
System.out.println(x + m * y - (y + n) * x);
System.out.println(m / n + m % n);
System.out.println(5 * x - n / 5);
System.out.println(Math.sqrt(Math.sqrt(n)));
System.out.println((int)x);
System.out.println(Math.round(y));
double x = 3.25;
double y = -4.5;
int m = 23;
int n = 9;
System.out.println((int)Math.round(x) + (int)Math.round(y));
System.out.println(m + n);
System.out.println(1-1-((1-(1-(1-n)))));
```

Terminologia

- Estes são os principais termos usados nesta aula:
  - Operador aritmético
  - Operador de atribuição
  - boolean
  - char
  - Convenções
  - Declaração
  - Double
  - float
  - Inicialização
  - Int
  - Literais

Terminologia

- Estes são os principais termos usados nesta aula:
  - long
  - Ordem da Operação
  - Tipos de dados primitivos
  - Escopo
  - Curto
  - Truncamento
  - Casting de tipo
  - Conversão de tipo
  - Variáveis

Resumo

- Nesta aula, você deverá ter aprendido a:
  - Usar tipos de dados primitivos em código Java
  - Especificar literais dos tipos primitivos e das Strings
  - Demonstrar como inicializar variáveis
  - Descrever as regras do escopo de um método
  - Reconhecer quando uma expressão exige uma conversão de tipo
  - Aplicar casting no código Java
  - Usar operadores aritméticos
  - Usar o operador de atribuição
  - Usar um método da classe Math
  - Acessar um método da classe Math da API do Java


ORACLE
Academy