**01Tipo inteiro: int**

**Transcrição**

Trabalharemos com sintaxes de variáveis e controles de fluxo - laços e condicionais - pela criação de um novo projeto acessando-se "New > Java Project". Poderíamos fazer tudo isto no mesmo arquivo, mas o intuito aqui é de treinar a codar e perder o medo das janelas e suas diversas opções.

Criaremos o "sintaxe-variaveis-e-fluxo", os dois tópicos que começaremos a ver. O novo projeto contendo o diretório "src" estará visível na *view* de "Package Explorer". No prompt, há um diretório "bin" escondido, pois o programa não quer mostrar o .class, e sim o código fonte Java. Reparem que no momento estou usando Mac, o que pouco importa, já que o Eclipse funciona da mesma maneira em todos os sistemas operacionais.

Criaremos nossa classe para começar a trabalhar com variáveis. Clicaremos com o lado direito do mouse em "src" e depois em "New > Class", e a classe se chamará "TestaVariaveis". No Java, um *statement* (ou instrução) não funciona fora dos métodos, portanto precisaremos do ponto inicial, do public static void main(String[] args), após o qual salvaremos:

public class TestaVariaveis {

public static void main(String[] args) {

}

}Copiar código

Poderíamos rodar a aplicação assim como está, mas não aconteceria nada. Então, digitaremos:

public class TestaVariaveis {

public static void main(String[] args) {

System.out.println("ola novo teste");

}

}Copiar código

Salvaremos novamente e rodaremos a aplicação indo à "Run > Run As > Java Application", ou clicando com o lado direito do mouse na classe com main, e em "Run As > Java Application". Também há o atalho "Ctrl + S". O Console mostrará o print, e com isto repetimos o mesmo teste do "ola mundo" feito anteriormente.

As palavras que aparecem em roxo no editor são as palavras chave, reservadas, e deverão estar sempre em caixa baixa. Agora, para criarmos uma variável denominada idade, que armazenará nossas idades, digitaremos:

public class TestaVariaveis {

public static void main(String[] args) {

System.out.println("ola novo teste");

idade = 37;

}

}Copiar código

No Java, como o Eclipse já está dando a entender sublinhando idade com vermelho, não compila isto, pois trata-se de uma linguagem **estaticamente ou fortemente tipada**, ou seja, que necessita da declaração de todas as variáveis e tipos a serem utilizados. Passando o mouse sobre a palavra sublinhada, lê-se a mensagem de erro *"idade cannot be resolved to a variable"*.

Significa que "idade não pode ser entendida como uma variável", pois não foi declarada. O Eclipse inclusive dará algumas opções de "rápido conserto", ou *quick fix*, para a criação local da variável, ou remoção da linha, por exemplo. idade = 37 é uma **atribuição**, em que 37 se encontra dentro de idade.

Precisaremos declará-la informando que ela é do tipo numérico e que guarda um valor inteiro, sem decimais ou pontos flutuantes. int vem de *Integer*:

public class TestaVariaveis {

public static void main(String[] args) {

System.out.println("ola novo teste");

int idade;

idade = 37;

}

}Copiar código

Salvaremos e rodaremos este código. Clicando-se na setinha ao lado do ícone verde que indica *play* na barra de ferramentas superior, vê-se os últimos programas que foram rodados no programa. E clicando no ícone verde, roda-se o último deles.

O valor foi guardado, mas parece que nada aconteceu de fato. Além de atribuirmos uma variável, pode-se usar o valor, mostrando-o na tela. Para isto, utilizaremos o System.out.println de novo, desta vez sem as aspas, pois queremos a *evaluation*, o resultado daquela expressão, e não uma cadeia de caracteres, uma *string*:

public class TestaVariaveis {

public static void main(String[] args) {

System.out.println("ola novo teste");

int idade;

idade = 37;

System.out.println(idade);

}

}Copiar código

Inclusive, é possível ver que todas as menções à variável idade ficam em *highlight*, destacadas para mostrar que tratam-se da mesma variável. Vamos rodar o código acima para imprimirmos o valor de idade! No "Console", obteremos:

ola novo teste

37Copiar código

Poderemos trabalhar com os operadores aritméticos junto a estas variáveis, também:

idade = 30 + 10;

idade = 7 \* 5 + 2;Copiar código

Como na maioria das linguagens, no Java também há precedência, então as operações matemáticas seguem uma determinada ordem de prioridade, mas poderemos usar parênteses, desta forma:

idade = (7 \* 5) + 2;Copiar código

E assim por diante. Imprimiremos a idade três vezes:

int idade;

idade = 37;

System.out.println(idade);

idade = 30 + 10;

System.out.println(idade);

idade = (7 \* 5) + 2;

System.out.println(idade);Copiar código

E obteremos o resultado esperado, na aba "Console":

37

40

37Copiar código

No código, usamos algumas convenções: ao criarmos a classe TestaVariaveis, cuja funcionalidade ainda desconhecemos, usamos a primeira letra em maiúscula e, ao acrescentarmos a segunda palavra, não utilizamos *underscore* ou algo do tipo, e sim a primeira letra em caixa alta de novo. Isto se chama ***Camel Case***, e aparece com frequência no Java e em muitas outras linguagens - é uma **convenção de código**, e seu uso não é obrigatório.

Da mesma forma, a variável iniciando-se com "i" minúsculo é o padrão, bem como não há o costume de se abreviar palavras. No Java, vocês verão nomes gigantescos de variáveis! É legal nos atentarmos a estas práticas para começarmos a nos acostumar com estes hábitos essenciais para quando formos trabalhar com grandes equipes.

Para mostrarmos uma frase antes de um número, basta imprimirmos uma *string*, como "a idade é", juntamente com a variável idade, assim:

System.out.println("a idade é " + idade);Copiar código

O operador **+**, na maioria das vezes, tem a função de somar variáveis de tipo numérico, sendo a única exceção estes casos em que acompanham *strings*, com os números sendo convertidos em letras e tudo sendo concatenado. Este operador, portanto, também serve para concatenar algo com uma palavra ou frase (uma *string*).

Salvando e rodando a aplicação, teremos:

A idade é 37Copiar código

Pode-se acrescentar mais *string*s após a variável usando-se o operador.

Há outra versão do System.out.println(), o System.out.print(), sem o **ln**, isto é, sem o *line*, que pula a linha, que poderá ser utilizado de acordo com sua preferência.

**02Criando uma variável numérica**

Selecione a opção que mostra como podemos criar e definir o valor de uma variável para representar a quantidade de dias de um mês?

Parte superior do formulário

* int Dias;

Dias => 31;

Errado, para definir ou atribuir um valor a variável no java precisamos usar '=' ao invés de '=>' como no exemplo. O nome da variável 'Dias' não dá erro no compilador, mas temos que lembrar que não é a convenção da comunidade Java. Variáveis normalmente são iniciadas com letras minúsculas.

* Alternativa correta

diasDoMes;

diasDoMes = 30;

Errado, no java, por ser uma linguagem estaticamente tipada é necessário definir o tipo das variáveis. Faltou dizer que o tipo é int.

* Alternativa correta

int diasDoMes;

diasDoMes = 30;

Resposta correta.

* Alternativa correta

int diasDoMes

diasDoMes = 30

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**03Mão na massa: utilizando o tipo int**

Está na hora de praticarmos o uso de tipos primitivos, para isso, siga os passos abaixo!

1) Crie um projeto, lembre-se que para fazer isso usamos o **new** e escolhemos **Java project**.

2)Nessa janela, nomeie seu projeto. No curso, escolhemos "**sintaxe-variaveis-e-fluxo**".

3) Agora crie uma classe, utilizando novamente o **new**, só que dessa vez, escolha **class** dentre as opções do Eclipse.

4) Na janela, nomeie sua classe. No curso usamos o nome "**TestaVariaveis**"

5) Escreva um código semelhante ao nosso anterior, ficando dessa forma.

public class TestaVariaveis{

public static void main(String [] args){

System.out.println("ola novo teste");

}

}Copiar código

6) Crie uma variável com nome idade, do tipo **int** e atribua um valor:

**-** Podemos fazer de duas formas, criando a variável e depois inicializando-a:

int idade;

idade = 37;Copiar código

**-** ou, criando a variável e inicializando-a na mesma linha

int idade = 37;Copiar código

7) Imprima a variável idade, nosso código deve ficar assim:

public class TestaVariaveis{

public static void main(String [] args){

System.out.println("ola novo teste");

int idade = 37;

System.out.println(idade);

}

}Copiar código

Repare que quando queremos imprimir o **valor** que está guardado na variável, não devemos usar as aspas.

8) Perceba que podemos usar operadores aritméticos:

+ para Soma

- para Subtração

\* para Multiplicação

/ para Divisão

9) Podemos concatenar valores no System.out.println usando o sinal **+**, dessa forma faça:

public class TestaVariaveis{

public static void main(String [] args){

System.out.println("ola novo teste");

int idade = 37;

System.out.println("a idade é "+ idade + ", parabéns!");

}

}Copiar código

**Opinião do instrutor**

Prossiga para a próxima atividade, onde veremos o uso do tipo **double**!

**04Tipo flutuante: double**

**Transcrição**

Também queremos trabalhar com outros tipos de variáveis, pois se tentarmos colocar no lugar da idade, em idade = 37;, um valor como 37.5, a compilação não irá ocorrer. O erro que se lê ao passarmos o mouse em cima, é "*Type mismatch: cannot convert from double to int*", isto é, a conversão não é possível. Lembrando que no Java nunca usaremos a vírgula para separar o decimal no código fonte.

Vamos criar uma nova classe para testar os números com **ponto flutuante**, clicando em "(default package)" com o lado direito do mouse e em "New > Class", nomeando-a de "TestaPontoFlutuante". Teremos, então:

public class TestaPontoFlutuante {

public static void main(String[] args) {

double salario;

salario = 1250.70;

System.out.println("meu salário é " + salario);

}

}Copiar código

Há dois tipos de variáveis em que cabem o tal de ponto flutuante, sendo que a mais utilizada é o double, como visto acima. Salvaremos o código e o rodaremos! Na aba "Console", obteremos o resultado:

meu salário é 1250.7Copiar código

Dica: as *views* podem ser customizadas de acordo com sua necessidade, sendo possível fechar aquelas que não estão em uso, por exemplo.

O 0 (zero) referente aos centavos não apareceram porque é assim que a variável double é convertida para se juntar à *string*. Poderemos formatar para aparecerem duas, três casas decimais, ou apenas uma, por meio dos *formatters* do Java, inclusos na biblioteca. Não veremos isto neste curso, porém há diversos recursos disponíveis, como o printf, da linguagem C, para colocarmos porcentagens e afins.

No double cabem variáveis do tipo inteiro, isto é, poderemos fazer o caminho inverso, indicando que temos uma variável que guarda idade, com número 37. Reparem que este valor não possui decimal. Não tem problema, um número de tipo inteiro cabe em um tipo double. O inverso, um decimal em um int, é que não é compatível.

O Java tem regras um tanto rígidas, portanto não aceitará 3.0 como int, já que não aceita pontos flutuantes. Ele é uma linguagem com peculiaridades que algumas pessoas podem estranhar, sendo vantajoso para se trabalhar em equipe pois reforça uma padronização, e as pessoas trabalham de formas parecidas.

Para enxergarmos a forma como o double funciona, podemos fazer uma conta de divisão, por exemplo:

double divisao = 3.14 / 2;

System.out.println(divisao);Copiar código

Rodando o código acima, teremos:

1.57Copiar código

Parece básico para quem já conhece linguagem estaticamente tipadas, mas mesmo nessa parte mais básica da linguagem, nos aprofundaremos mais, para sentirmos algumas das características do Java.

O que aconteceria no caso de digitarmos int outraDivisao = 5 / 2;? Alguns podem pensar que isso não é compilado, que dará erro, pois o resultado é 2.5, o que não cabe em int.

No Java, entretanto, há uma regra: quando há uma divisão entre dois números inteiros, ele "forçará" um número inteiro como resultado. Se printarmos esta divisão, obteremos como resultado o valor 2. Estranho, não?

E se quiséssemos que o resultado fosse 2.5 de fato, poderíamos tentar double novaTentativa = 5 / 2;, e pediríamos sua impressão, que traria 2.0. Piorou! O Java irá ler apenas o lado que vem antes da atribuição na linha de código relativa ao double. Ou seja, primeiro, ele irá executar a divisão 5 / 2, e depois ele tentará colocar o resultado em um double.

Na verdade, o que gostaríamos é que a conta tivesse sido feita partindo-se do double e, neste caso, bastaria que um dos valores da divisão fosse deste tipo, como em 5.0 / 2. Desta forma, como trata-se de um double dividido por um int, a conta é feita levando-se em consideração o ponto flutuante.

Esta divisão, sim, trará 2.5 como resultado. Parece pegadinha, mas são características de linguagem que vão te deixar mais a par de como o Java funciona.

Fizemos um truque: estávamos sempre declarando a variável primeiro, e na linha seguinte, fazendo a atribuição. Depois, fizemos double idade = 37;. Ao declararmos variáveis, é muito comum inicializá-las, porque é estranho declararmos uma variável, digitarmos um monte de código e só depois inicializarmos um valor.

O comum é fazermos tudo na mesma linha. Desse modo, as linhas abaixo,

double salario;

salario = 1250.70;Copiar código

pela proximidade, equivalem a escrevermos isto:

double salario = 1250.70;Copiar código

Declaramos a variável informando seu tipo, e a atualizamos, isto é, inicializamos ela, fazendo uma atribuição.

Vamos salvar o código!

**05Operações entre numeros**

Aprendemos que é possível fazer operações entre números, podemos até fazer operações aritméticas com variáveis de tipos diferentes como multiplicar um inteiro por um número de ponto flutuante (int e double).

Verificando cada instrução abaixo diga qual das opções está errada.

Parte superior do formulário

* int dia = 4;
* int outroDia = 4 + dia;
* System.out.println(outroDia);

Deve imprimir 8

* Alternativa correta
* int brindes = 15;
* int pessoas = 2;

System.out.println(brindes / pessoas);

Deve imprimir 7.5

Este código é o errado. Pois quando fazemos operações entre inteiros o resultado sempre será inteiro e não de ponto flutuante. O resultado na verdade seria 7.

* Alternativa correta
* double peso = 4.0;
* int quantidade = 10;

System.out.println(peso \* quantidade);

Deve imprimir 40.0

* Alternativa correta
* double preco = 5.5;
* int ingressos = 4;

System.out.println(preco \* ingressos);

Deve imprimir 22.0

Parte inferior do formulário

Parabéns, você acertou!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**06Mão na massa: utilizando o tipo double**

Vamos praticar um pouco do uso da variável do tipo **double**

1) Crie uma classe "**TestaPontoFlutuante**" e estruture seu método main.

2) Declare uma variável **salario** com o valor de 1250.70, dessa forma:

double salario = 1250.70;Copiar código

Se **salario** fosse do tipo **int** receberíamos um erro, já que temos casas decimais.

3) Imprima a variável **salario** e verifique se o valor está correto, Nosso código deve ficar assim:

public class TestaPontoFlutuante{

public static void main(String[] args){

double salario = 1250.70;

System.out.println(salario);

}

}Copiar código

4) Utilize o **+** para concatenar a variável **salario** com um texto:

System.out.println("meu salario é " + salario);Copiar código

5) Lembre-se que não podemos colocar um valor com ponto flutuante dentro de uma variável do tipo **int**, teste:

public class TestaPontoFlutuante{

public static void main(String[] args){

double salario = 1250.70;

System.out.println(salario);

int valor = 12.5; //não compila.

int valor2 = 0.0; //não compila.

double teste = 125.50; // compila

int valor3 = teste; //não compila.

}

}Copiar código

O Java é bastante tipado, veja que nem com o valor **0.0** nosso código compilará, então tenha atenção com isso!

6) Teste também um cálculo de divisão:

int divisao = 5 / 2;

System.out.println(divisao);Copiar código

Nossa saída será **2**.

É importante ressaltar que quando fazemos uma divisão não exata, mas estamos dividindo números inteiros, o Java sempre **truncará** o resultado. Nesse exemplo a saída será sempre **2**.

Se a variável **divisao** fosse do tipo **double**, teríamos o mesmo problema, a única diferença seria que nossa saída teria casa decimal: 2**.0**

7) Para resolver o problema, coloque em um dos valores da divisão com alguma casa decimal, além de mudar o tipo da variável para double:

double divisao = 5.0 / 2;

System.out.println(divisao);Copiar código

Agora sim obteremos a saída esperada: **2.5**

**Opinião do instrutor**

Na próxima atividade praticaremos um pouco de conversões.

Caso tenha dúvidas, fale conosco no nosso fórum a qualquer momento!

**07Conversões e outros tipos**

**Transcrição**

Haverá momentos em que queremos misturar os tipos de variáveis, como o double e o int. Vimos que um int cabe no double, mas o caminho inverso não funciona. Vamos, então, criar uma classe denominada "TestaConversao".

Incluiremos uma variável do tipo salario com os 1270.50, que por algum motivo queremos que esteja em uma variável do tipo inteiro. E então guardaremos salario em valor:

public class TestaConversao {

public static void main(String[] args) {

double salario = 1270.50;

int valor = salario;

}

}Copiar código

Já vimos que isto não funciona, pois o compilador do Java é rígido e não deixa que isto ocorra sem que afirmemos com total segurança de estarmos cientes de que perderemos o .50. Por conta disso, deixaremos as duas linhas comentadas, e mostraremos que o caminho inverso é possível:

public class TestaConversao {

public static void main(String[] args) {

// double salario = 1270.50;

// int valor = salario;

double valor = 3;

}

}Copiar código

Ou seja, a conversão de um valor inteiro para um tipo double é possível, academicamente chamada de **promoção**, ou "ser promovido a um double", e acontece de maneira automática.

Para tentarmos fazer com que a parte do código comentada acima funcione, poderemos forçar a conversão, moldando um double para que ele se encaixe em um int.

É claro que não haverá encaixe perfeito, resultando em arestas que provavelmente serão perdidas. Faremos isso utilizando uma sintaxe comum a outras linguagens, o *casting*, para que o double seja transformado em um int.

public class TestaConversao {

public static void main(String[] args) {

double salario = 1270.50;

int valor = (int) salario;

System.out.println(valor);

}

}Copiar código

Se printarmos valor, será mostrada apenas a parte inteira daquele número: 1270. É isso que chamamos de *casting* que, nestas variáveis que guardam números, não é algo muito complexo.

Mais adiante, veremos o *casting* de variáveis que são referência, e têm a ver com orientação a objetos, se são compilados ou não, se darão *exceptions*; é um mundo à parte.

Basicamente, para os tipos chamados **primitivos**, as variáveis básicas que estamos vendo aqui e são double com "d" minúsculo, e na cor roxa, possuem funcionamento mais simples. O *casting* faz a conversão quando ela não é possível de forma automática.

Neste caso, sem o (int), assim, entre parênteses, a compilação não ocorre, e a aplicação não rodará.

Como saberemos quais valores se encaixam em quê, e outros tipos numéricos?

No Java, o int e o double são os tipos mais usados, os outros aparecem de maneira muito esporádica. A nível de curiosidade, em int cabem 32bits com sinais, isto é, números positivos e negativos. Mais especificamente, cabem de 2 elevado a 31 negativos, a 2 elevado a 31 positivos menos 1, por conta do 0 (zero), o que dá uma quantidade de cerca de 2 bilhões.

O int pode guardar até 2 bilhões e, passando dessa quantidade, ocorrerá um *overflow*. Caso se queira guardar um número maior ou menor que este, será preciso um número com 64bits, que no Java é o long, e guarda um número de até 2 elevado a 63 menos 1. É um número absurdo, que inclusive precisa de um **L** no fim, em caixa alta ou baixa, para indicar que estouramos os 2 bilhões!

long numeroGrande = 32432423523L;Copiar código

Por padrão, quando não é um double, um número no Java é considerado um int. O L indica "literal", um valor específico, como um long. Em contrapartida, há números menores: o short, que guarda um número de 16bits menos 1, e o byte, que é menor ainda, de até 2 elevado a 8, que dá 256 com 128 negativos, a 127 com 1 a menos:

short valorPequeno = 2131;

byte b = 127;Copiar código

E se o número for maior do que 64bits, um número gigantesco? Daí, não serão usados tipos primitivos, ou estas variáveis. Podem ser objetos, e então usaremos bibliotecas.

Nesse caso, usaremos este exemplo:

double valor1 = 0.2;

double valor2 = 0.1;

double total = valor1+valor2;Copiar código

Esta operação deveria resultar em 0.3, certo? Ao acrescentarmos System.out.println(total); e rodarmos o código, porém, obteremos 0.30000000000000004. Que número maluco é esse?

Há várias questões matemáticas por trás dele. Se pesquisarmos o valor no Google, encontramos diversos resultados de pessoas buscando uma explicação. Existe até o site [0.30000000000000004.com](http://0.30000000000000004.com/), com a explicação matemática para esse *floating point*, do porquê, em muitas linguagens, essa soma dar exatamente esse valor.

Não é à toa - como uma representação de decimal do inteiro é utilizada para se obter um ponto flutuante, fica complicado fazer uma operação aritmética deste tipo e guardar o resultado internamente. Por isto, o Java, como muitas outras linguagens, segue a especificação **IEEE 754**, de leitura complexa, que remete à Engenharia. De qualquer forma, é normal que este resultado apareça quando utilizamos o double.

Para lidarmos com dinheiro sem que apareçam centavos, por exemplo, usaríamos o BigDecimal, de que falaremos mais para a frente. Por ora continuaremos com o double pois ainda estamos iniciando na linguagem, e queremos usar variáveis que são palavras chave do Java.

Os quatro tipos de tipo primitivo são: int, long, byte e short. Quanto aos tipos flutuantes, além do double, há o float e, se tentarmos definir a variável como recebendo 3.14, ocorre o mesmo problema do long, mesmo se tratando de ponto flutuante.

Para o Java, 3.14 é um double com 64bits. É um valor que cabe em um tipo flutuante com 32bits? Não, e informações podem ser perdidas. Neste caso, usa-se o *casting*, o que seria estranho, ou se indica que este literal, o valor 3.14, é um float, colocando-se "**f**" no fim:

float pontoFlutuante = 3.14f;Copiar código

Mais uma vez, o mais importante é o enfoque no double e no int, que aparecem com muito mais frequência. E no long em alguns casos, o qual será visto em alguns exercícios.

**08Imprimindo texto e números**

Agora que já sabemos fazer operações com variáveis e também concatenar texto com números, use o que aprendeu para indicar todas as alternativas abaixo que mostrem o resultado "A idade de Marcos é 30!".

Parte superior do formulário

* int idade = 20;
* System.out.println("A idade de Marcos é 30!");

Opção correta! Neste caso não estamos usando a variável idade que recebe 20.

* Alternativa correta
* int trinta = 10 + 20;

System.out.println("A idade de Marcos é" + trinta);

* Alternativa correta
* double idade = 30.0;

System.out.println("A idade de Marcos é " + (int) idade + "!");

Opção correta! Estamos fazendo o type cast no double que tem seu valor convertido para int. Desta forma não aparece a casa decimal.

* Alternativa correta
* int idade = 3 \* 10

System.out.println("A idade de Marcos é " + idade + "!");

* Alternativa correta
* int idade = 30;

System.out.println("A idade de Marcos é " + idade + "!");

Opção correta!

Parte inferior do formulário

Parabéns, você acertou!

**09Mão na massa: algumas conversões em Java**

Vamos agora praticar algumas conversões em Java, para isso, siga os passos abaixo:

1) Crie uma classe e escreva seu método main, no curso escolhemos o nome "**TestaConversao**".

2) Escreva o seguinte código:

public class TestaConversao{

public static void main(String[] args){

double salario = 1270.50;

int valor = salario;

}

}Copiar código

Esse código não compila, uma vez que estamos tentando atribuir uma variável **double** dentro de uma variável **int**, o que não é possível na linguagem Java.

Lembre-se, entretanto, que o inverso é valido, ou seja, podemos atribuir um valor do tipo **int** dentro de uma variável do tipo **double**.

3) Existe uma forma de fazer com que esse código compile. Para isso usaremos o que chamamos de **casting**, para isso, modifique seu código e adicione (int) antes da variável salario:

public class TestaConversao{

public static void main(String[] args){

double salario = 1270.50;

int valor = (int) salario;

}

}Copiar código

Quando fazemos o **casting**, estamos pedindo ao Java que ache uma forma de transformar o valor no lado direito do **(int)**, no nosso caso, a variável **salario**, em inteiro. Dessa forma o java transformará **double** para **int**.

4) "*Printe*" a variável **valor** no console, para isso, adicione:

System.out.println(valor);Copiar código

A saída será **1270**.

Basicamente, para as variáveis primitivas, o **casting** nada mais é do que fazer com que essa conversão seja feita quando ela não é feita de maneira automática.

5) No momento, nossa variável **salario** é do tipo **double**, será que essa é a melhor escolha? Teste o seguinte:

Escreva 3 variáveis do tipo **double** como feito abaixo, a última variável (**total**) receberá o somatório das outras duas, observe:

public class TestaConversao{

public static void main(String[] args){

double salario = 1270.50;

int valor = (int) salario;

System.out.println(valor);

double valor1 = 0.2;

double valor2 = 0.1;

double total = valor1 + valor2;

System.out.println(total);

}

}Copiar código

Imaginamos que nossa saída seria **0.3**, mas obtemos um número completamente diferente.

Devido essa inconsistência, **double** não seria a melhor escolha para guardarmos salário. Existem outras alternativas melhores que veremos em outros cursos.

6) Vamos testar também o tipo **float**, para isso escreva a variável **pontoFlutuante** do tipo **float**, veja abaixo:

public class TestaConversao{

public static void main(String[] args){

float pontoFlutuante = 3.14; // aqui!

double salario = 1270.50;

int valor = (int) salario;

System.out.println(valor);

double valor1 = 0.2;

double valor2 = 0.1;

double total = valor1 + valor2;

System.out.println(total);

}

}Copiar código

Dessa forma, teremos um erro, já que para o Java, 3.14 é um **double**. Esse é o mesmo problema que tivemos quando tentamos guardar um **double** dentro de um **int** . No tipo **double** podemos armazenar 64 bits, no **float** e no **int** apenas 32 bits. Isso pode acarretar em perda de informação, para que esse código funcione, podemos fazer o **casting**, colocando **(float)** na frente de **3.14**, mas faremos de uma outra forma.

7) Informe ao Java que o valor 3.14 é um float, para isso coloque **f** ao final do número, nosso código ficará assim:

public class TestaConversao{

public static void main(String[] args){

float pontoFlutuante = 3.14f;

double salario = 1270.50;

int valor = (int) salario;

System.out.println(valor);

double valor1 = 0.2;

double valor2 = 0.1;

double total = valor1 + valor2;

System.out.println(total);

}

}Copiar código

Agora nosso código compila!

**Opinião do instrutor**

Eai, alguma dúvida? Fale com nossa comunidade no fórum!

**10Para saber mais: Type Casting**

Como foi visto nos vídeos, quando tentamos colocar um valor inteiro em uma variável do tipo double o Java não mostra erro. Quando tentamos, porém, colocar um double numa variável do tipo inteiro temos um erro de compilação.

Esta propriedade se dá porque o Java faz conversão implícita de um tipo menor para os tipos "maiores". De inteiro para double, por exemplo.

O contrário não é verdade porque existe perda de dados quando é feita a conversão. Acarretando em um "type mismatch" mostrando que esta instrução é de tipos incompatíveis.

Para fazer uma conversão onde pode haver perda de informações é necessário fazer um type casting. Veja a instrução abaixo.

int idade = (int) 30.0;Copiar código

No caso acima, está explícito que será feito o cast de double para inteiro. Veja como funciona o cast implícito e explícito na tabela abaixo.

| **DE/PARA** | **byte** | **short** | **char** | **int** | **long** | **float** | **double** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| byte | ---- | *Impl.* | (char) | *Impl.* | *Impl.* | *Impl.* | *Impl.* |
| short | (byte) | ---- | (char) | *Impl.* | *Impl.* | *Impl.* | *Impl.* |
| char | (byte) | (short) | ---- | *Impl.* | *Impl.* | *Impl.* | *Impl.* |
| int | (byte) | (short) | (char) | ---- | *Impl.* | *Impl.* | *Impl.* |
| long | (byte) | (short) | (char) | (int) | ---- | *Impl.* | *Impl.* |
| float | (byte) | (short) | (char) | (int) | (long) | ---- | *Impl.* |
| double | (byte) | (short) | (char) | (int) | (long) | (float) | ---- |

Para comparar cada tipo primitivo de forma mais clara, a tabela abaixo mostra qual o tamanho de cada um.

| **TIPO** | **TAMANHO** |
| --- | --- |
| boolean | 1 bit |
| byte | 1 byte |
| short | 2 bytes |
| char | 2 bytes |
| int | 4 bytes |
| float | 4 bytes |
| long | 8 bytes |
| double | 8 bytes |

**11O que aprendemos?**

Nesta aula iniciamos nosso aprendizado com variáveis e tipos primitivos do Java. Os tipos vistos com mais detalhe foram int (inteiro) e double (decimal). Que usamos para fazer operações aritméticas e também concatenar com texto.

Durante o capítulo falamos sobre boas práticas na hora de nomear classes e também variáveis. Iniciamos as classes com letra maiúscula e as nossas funções e variáveis com letras minúsculas. Vamos falar mais sobre isto no futuro.

Foi possível entender um pouco de type casting e como podemos passar um valor de um tipo para uma variável de outro. Para alguns casos nós não precisamos fazer nada, pois o casting é implícito e em outros precisamos deixar claro para o compilador que sabemos o que estamos fazendo, mostrando entre parênteses o tipo que queremos que seja usado.

Com isso finalizamos esta aula. Próximo passo: caracteres! Espero vocês lá!

Este conteúdo foi útil para o seu aprendizado?

Sim

Não muito

**12Arquivos do projeto atual**

No link abaixo, você encontra o projeto até o momento atual do curso.

<https://github.com/alura-cursos/Curso-Java-parte-1-Primeiros-passos/archive/capitulo4.zip>

**Opinião do instrutor**

Lembre-se de usar o fórum em caso de dúvidas.