

# Switch\case

- ▶ Estrutura de controle condicional;
- ▶ Usa comparação de valores de inteiros (ou caracteres) para selecionar uma posição na qual entrar;
- ▶ Se um case for selecionado, ele vai executar a partir do case selecionado até encontrar uma instrução de parada ( o “break”);
- ▶ Se nenhum dos cases for selecionado, ele procura por uma opção “default”;



# Exemplo de switch

```
int x = 1;
```

```
switch (x) {
```

```
    case 1:
```

```
        System.out.println("deu um");
```

```
        break;
```

```
    case 2:
```

```
        System.out.println("deu 2");
```

```
    case 3:
```

```
        System.out.println("Continuando...");
```

```
        break;
```

```
    default:
```

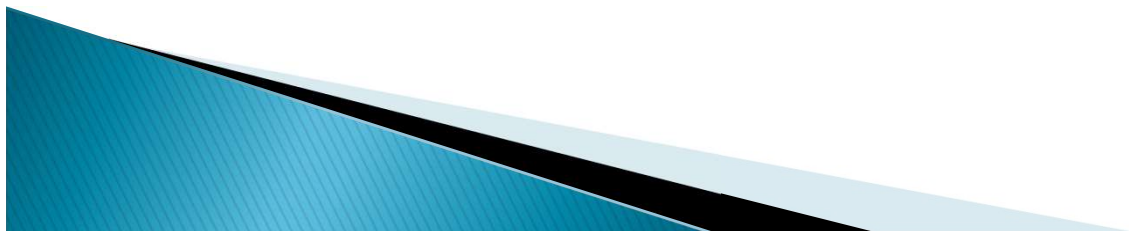
```
        System.out.println("Não foi ninguém");
```

```
}
```



# Em um switch...

- ▶ Só se usam **inteiros** ou **caracteres**;
- ▶ Se não houver um “break” ele cai para o próximo case (ou default);
- ▶ Nunca esqueça de colocar os breaks;
- ▶ O default poder vir em qualquer lugar (e não só no final);



# Passada rápida – classe Scanner

- ▶ A classe Scanner implementa um leitor de uma interface de texto (como uma linha de comando);
- ▶ É um modo simples de se receber instruções de um usuário (ou de outro programa);
- ▶ Foi adicionada no Java 1.5;



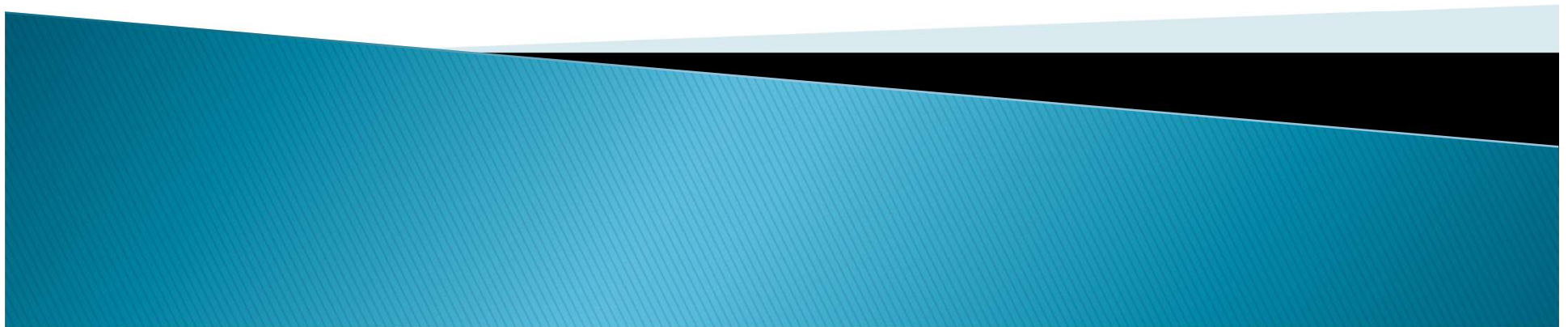
# Usando scanners

```
public class ScannerTest {  
  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner( System.in );  
        System.out.println("Escreva um inteiro");  
        System.out.println("O inteiro lido foi -> " +  
scanner.nextInt());  
        System.out.println("Escreva um double");  
        System.out.println("O double lido foi -> " +  
scanner.nextDouble());  
        System.out.println("Escreva qualquer coisa");  
        System.out.println("O que você escreveu foi -> " +  
scanner.next());  
    }  
}
```



# Variáveis em Java

Rodrigo Fujioka –  
fujiokabr@gmail.com



# Variáveis em Java

Não pode!

```
Cachorro cachorro = new Gato();  
cachorro.late();
```

```
Cachorro outroCachorro =  
    new Cachorro();  
outroCachorro.late();
```

Pode!



# Declaração de variáveis

- ▶ As variáveis devem ter um tipo;
- ▶ As variáveis devem ter um nome;
- ▶ Tipo e classe são a mesma coisa (aprenda isso);





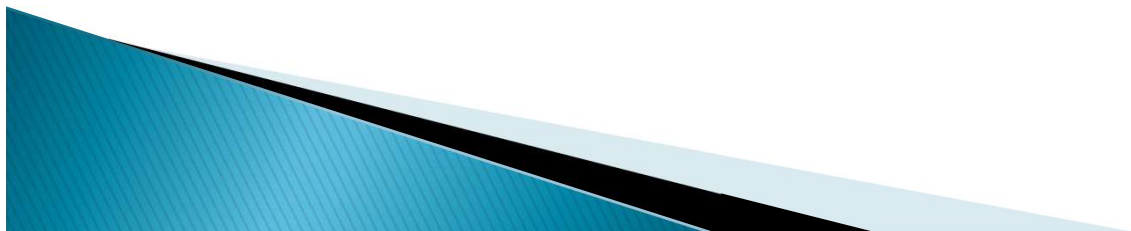
# Nomes de identificadores do Java

- Identificadores são palavras utilizadas para nomear variáveis, métodos e classes.
- Não se pode utilizar palavras reservadas, devendo começar com uma letra, \$ ou \_, podendo contudo utilizar números como parte do identificador.
- Válidos: louco intGrande \$valor1
- Inválidos: ~~3\_variavel~~ ~~!verdade~~
- Java é uma linguagem “case-sensitive”, ou seja, MAIÚSCULAS minúsculas
- Total e total são identificadores distintos

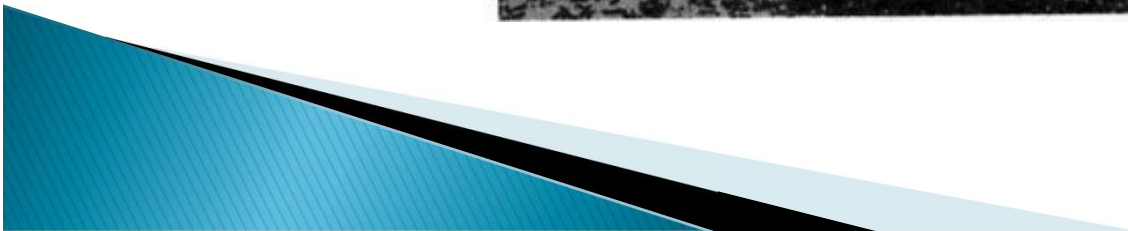
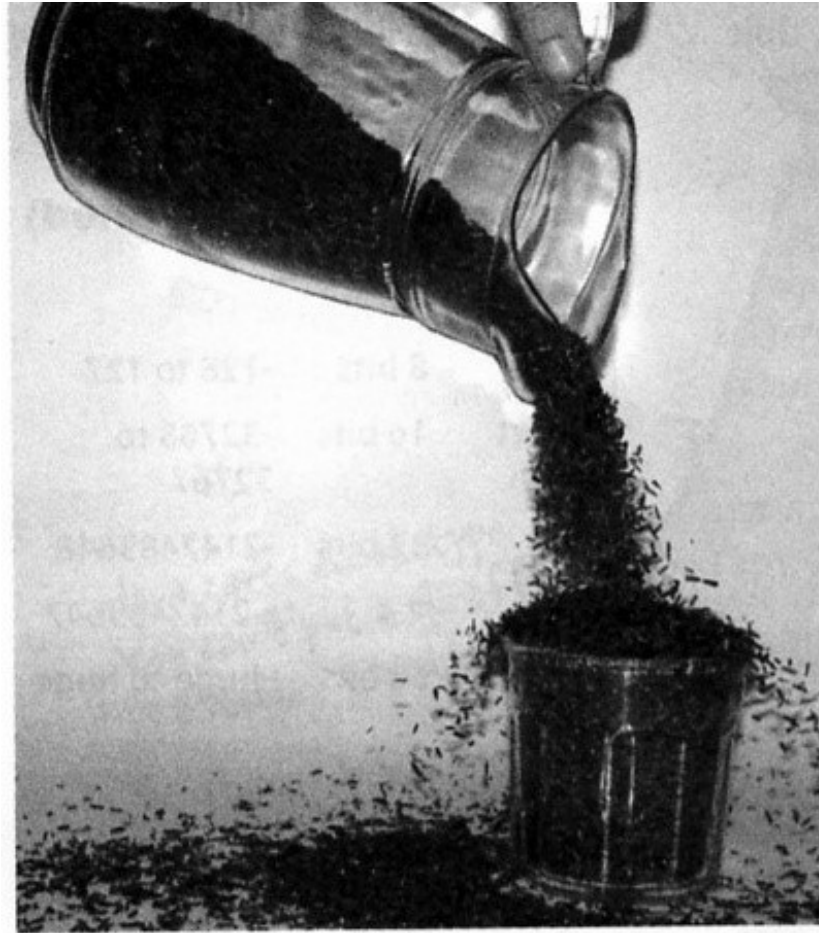


# Palavras reservadas da linguagem

abstract continue for new switch assert  
default goto package synchronized boolean  
do if private this break double implements  
protected throw byte else import public  
throws case enum instanceof return transient  
catch extends int short try char final interface  
static void class finally long strictfp volatile  
const float native super while



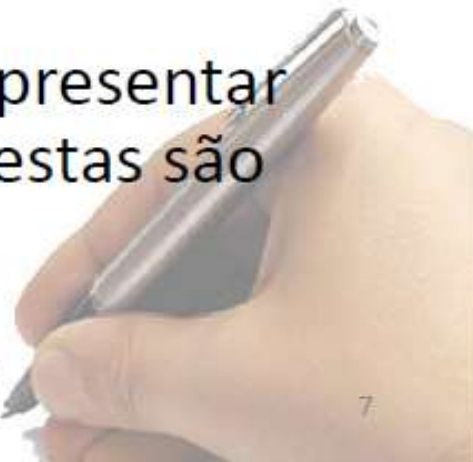
# O que é uma variável?



# Tipos de Dados

- Java é uma linguagem fortemente tipada. Assim, cada variável declarada no escopo da aplicação deve ter um tipo definido, seja ele **primitivo** ou **objeto**.
  - **Tipos Primitivos** – São unidades indivisíveis de dados, que possuem tamanho fixo
  - **Tipos Objetos** – São estruturas complexas, que representam diversos dados
- Algumas classes são utilizadas para representar tipos primitivos sob forma de objeto, estas são denominadas classes **Wrappers**.

Rodrigo Fujieka - Ling de Prog 2

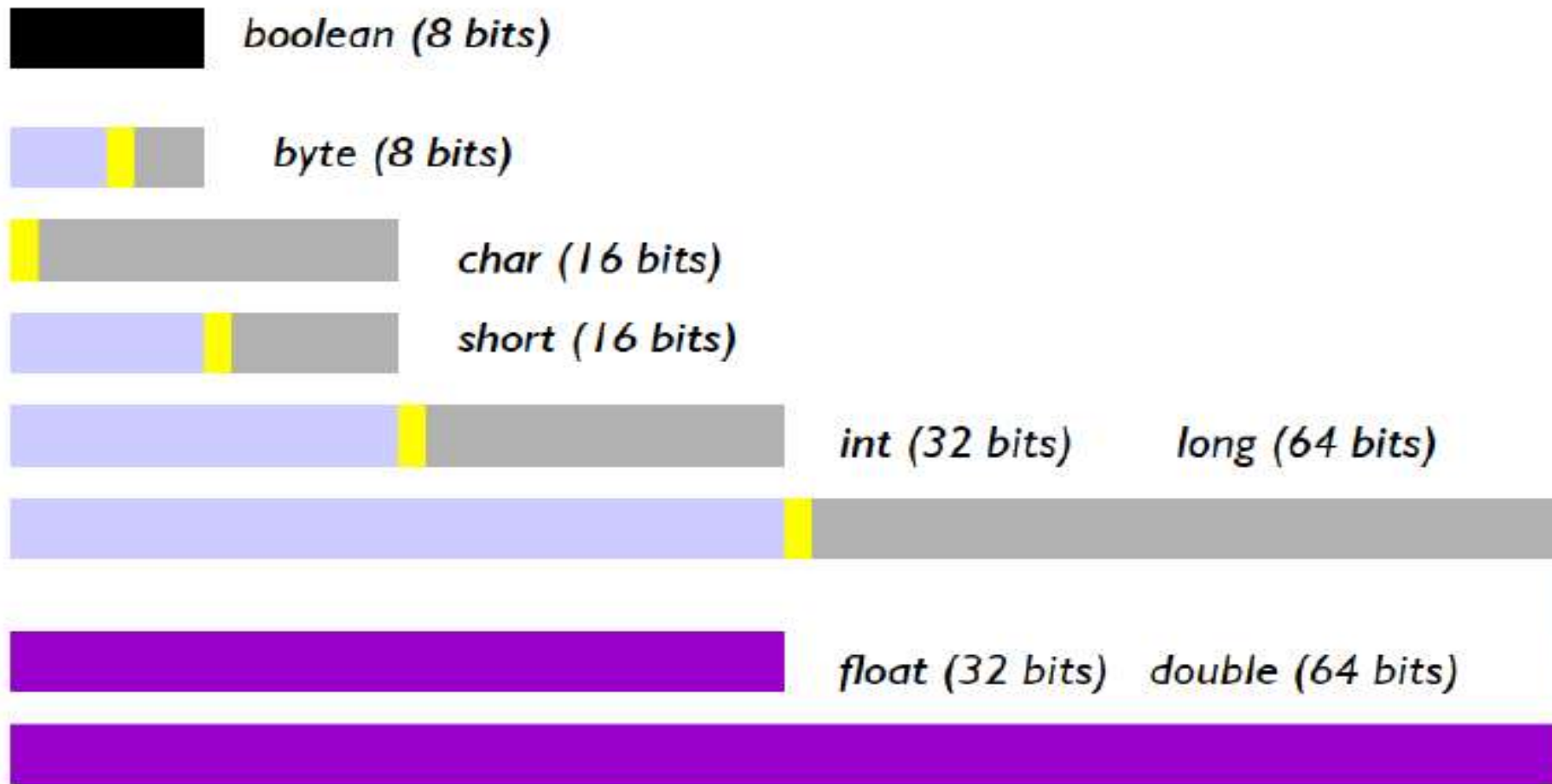




# Tamanhos dos tipos primitivos

boolean	1 bit	true ou false
character	16 bits	0 a 65535
byte	8 bits	-128 a 127
short	16 bits	-32768 a 32767
int	32 bits	-2147483648 a 2147483647
long	64 bits	Não cabe aqui 😊
float	32 bits	-
double	64 bits	-

# Tamanhos dos tipos primitivos



Redrigo Fujioka - Ling de Prog 2

# Declarando Variaveis

<tipo> <nome da variável> = <valor da variável>;



# Declarando Variaveis

- Uma variável é um identificador que representa uma posição na memória capaz de armazenar um tipo específico de dado
- As variáveis devem ser declaradas antes de serem utilizadas:  
**TipoVariavel nomeVariavel;**
- Variáveis podem ser declaradas na mesma linha
- Variáveis podem ser inicializadas no momento da declaração

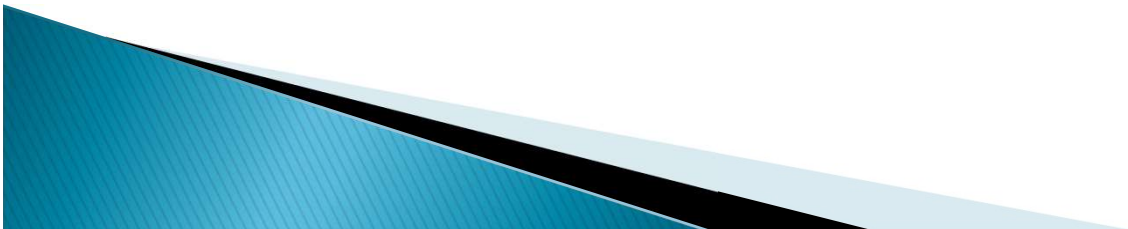
```
int total, count, sum;  
double unitPrice = 57.25;
```

*Rodrigo Fujieka - Ling de Prog 2 Q*



# Atribuição de variáveis

```
public class TesteDeAtribuicoes {  
  
    public static void main(String[] args) {  
        int valor = 10;  
        boolean verdadeiro = false;  
        int soma = valor + 8;  
        int outroValor = valor;  
        double numeroEstranho = 234.987;  
        char caractere = 'P';  
    }  
}
```



# Declarando inteiros

```
public class DeclarandoInteiros {  
  
    public static void main(String[] args) {  
        int inteiro = 1;  
        short pequeno = -20;  
        long grande = 123455;  
        long outroLong = 123456663345L;  
    }  
}
```



# Declarando números de ponto flutuante

```
public class DeclarandoFloats {  
  
    public static void main(String[] args) {  
        float preco = 3.34F;  
        double medida = 12342.45566;  
    }  
}
```



# Declarando caracteres

```
public class DeclarandoCaracteres {  
  
    public static void main(String[] args) {  
        char character = 'Y';  
        char unicode = '\\u0059';  
    }  
}
```



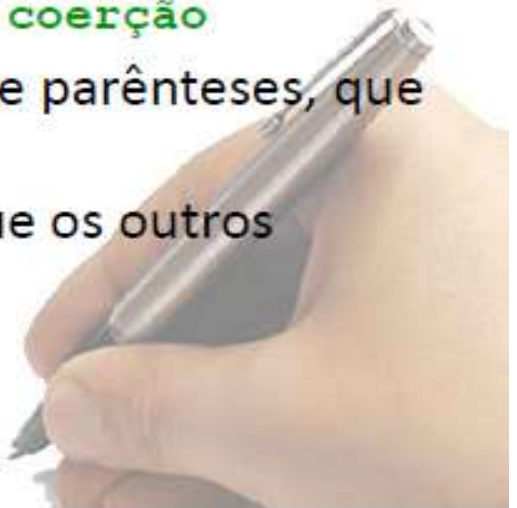
# Perda de precisão

- ▶ Ao tentar colocar um número grande em uma variável que não é grande o suficiente, perdemos precisão;
- ▶ O compilador não aceita a perda de precisão a não ser que o programador se responsabilize por ela;
- ▶ Você não vai querer isso em um sistema que lide com dinheiro;



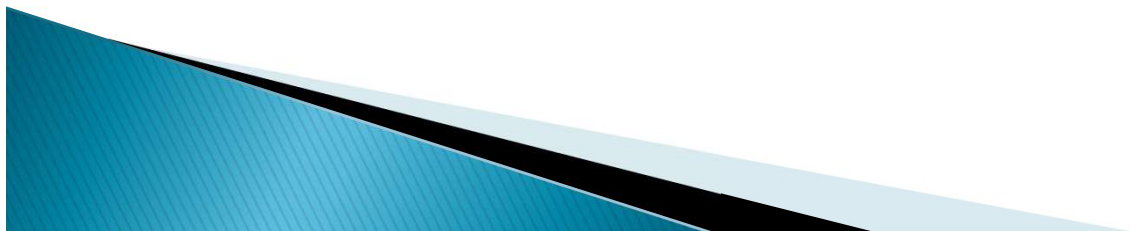
# Perda de dados

- Na coerção (cast), o **programador** assume os riscos da conversão de dados
  - No tipo byte cabem inteiros até 127
  - No tipo short cabem inteiros até 32767
  - Não há risco de perda de informação na atribuição a seguir  
**short s = 100; byte b = s;**  
pois (100 cabe em byte) mas o compilador acusará erro porque um short não pode ser atribuído a byte
  - Solução: **byte b = (byte) s; // operador de coerção**
  - O programador “assume o risco”, declarando entre parênteses, que o conteúdo de **s** cabe em **byte**
  - O operador de coerção tem maior precedência que os outros operadores!



# Perda de precisão

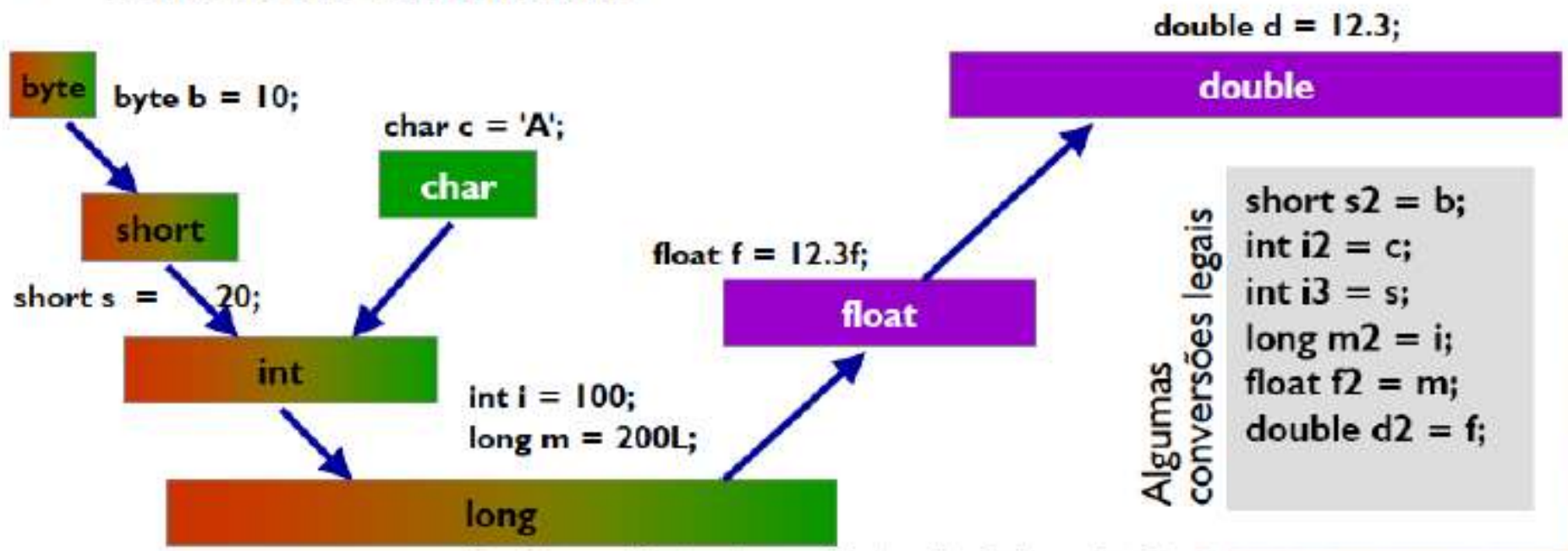
```
public class PrecisaoTest {  
  
    public static void main(String[] args) {  
        int grande = 40000;  
        short menor = (short) grande;  
        //correto  
  
        short pequeno = grande;  
        //errado  
    }  
}
```





# Castings possíveis

- Java converterá um tipo de dados em outro sempre que isto for apropriado
- As conversões ocorrerão automaticamente quando houver garant de não haver perda de informação
  - Tipos menores em tipos maiores
  - Tipos de menor precisão em tipos de maior precisão
  - Inteiros em ponto-flutuante
- Conversões automáticas





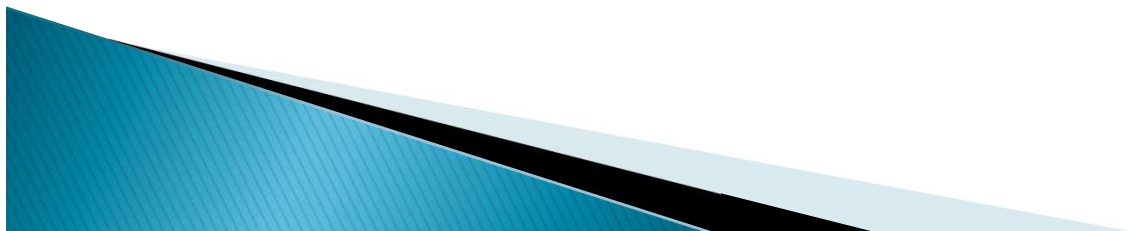
# Castings possíveis

PARA:	byte	short	char	int	long	float	double
DE:	byte	short	char	int	long	float	double
byte	----	<i>Impl.</i>	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
short	(byte)	----	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
char	(byte)	(short)	----	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
int	(byte)	(short)	(char)	----	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
long	(byte)	(short)	(char)	(int)	----	<i>Impl.</i>	<i>Impl.</i>
float	(byte)	(short)	(char)	(int)	(long)	----	<i>Impl.</i>
double	(byte)	(short)	(char)	(int)	(long)	(float)	----



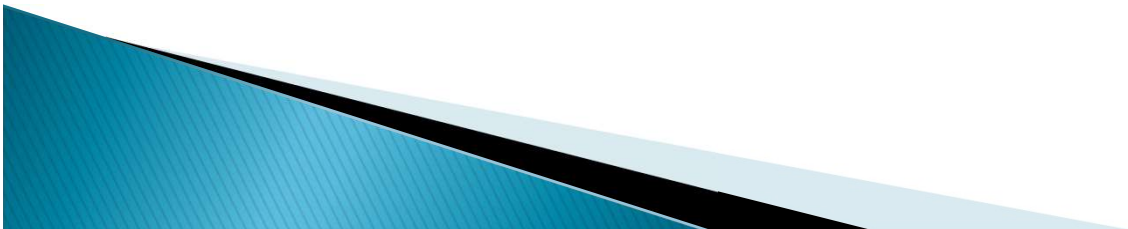
# Matemática de ponto flutuante no Java

- ▶ Operações matemáticas com ponto flutuante no Java não são precisas;
- ▶  $1.01 + 1.17$  não é  $2.18 \rightarrow 2.179999999999999997$ ;
- ▶ Não use números flutuantes para matemática exata no Java (dinheiro? dos outros? Nem pensar!);



# BigDecimal e matemática precisa no Java

```
public class MatematicaComBigDecimal {  
  
    public static void main(String[] args) {  
        BigDecimal numero = new BigDecimal("10.7");  
        BigDecimal outroNumero = new BigDecimal("9.5");  
        System.out.println( numero.add( outroNumero ) );  
    }  
}
```



# Operadores

- Um operador produz um novo valor a partir de um ou mais argumentos
- Os operadores em Java são praticamente os mesmos encontrados em outras linguagens
  - `+`, `-`, `/`, `*`, `=`, `==`, `<`, `>`, `>=`, `&&`, etc.
- A maior parte dos operadores só trabalha com valores de tipos primitivos
- Exceções:
  - `+` e `+=` são usados na concatenação de Strings
  - `!=`, `=` e `==` são usados também com objetos (embora não funcionem da mesma forma quanto aos valores armazenados nos objetos)

# Operadores relacionais

<code>==</code>	<i>igual</i>
<code>!=</code>	<i>diferente</i>
<code>&lt;</code>	<i>menor</i>
<code>&lt;=</code>	<i>menor igual</i>
<code>&gt;</code>	<i>maior</i>
<code>&gt;=</code>	<i>maior igual</i>

Sempre produzem um resultado booleano

- Comparam os valores de duas variáveis ou de uma variável e uma constante Comparam as referências de objetos (apenas `==` e `!=`)

# Operadores relacionais

- ▶ **&&** *E(and)*
- ▶ **||** *Ou(or)*
- ▶ **!** *Negação(not)*
- ▶ Produz sempre um valor booleano
  - true ou false
  - Argumentos precisam ser valores booleanos ou expressões com resultado booleano
  - Ex. **(3 > x) && (y <= 10)**

# Operador ternário (if-else)

- ▶ Retorna um valor ou outro, dependendo do resultado de uma expressão booleana

`variável = expressão ? valor, se true`

`variável = expressão : valor, se false;`

- ▶ Exemplo:

`int x = (y != 0) ? 50 : 500;`

`String tit = (sex == 'f') ? "Sra." : "Sr.";`

`num + " pagina" + (num != 1) ? "s" : "";`

- ▶ Apesar de poder levar o código difícil de entender, é bastante útil em alguns casos

# Operador de concatenação

- ▶ Em uma expressão usando o “+” com dois operandos, se um deles for String, o outro será convertido automaticamente para String e ambos serão concatenados
- ▶ A operação de concatenação, assim como a de adição, ocorre da direita para a esquerda  
`String s = 1 + 2 + 3 + “=” + 4 + 5 + 6;`
- ▶ Resultado: s contém a String “6=456”



# instanceof

- ▶ **instanceof** é um operador usado para comparar uma referência de uma classe
  - A expressão será true se a referência for do tipo de uma classe ou subclasse testada e false, caso contrário
  - Só pode ser usado com valores de tipo de referência
  - Sintaxe: `referência instanceof Classe`

- ▶ Exemplo:

```
if (obj instanceof Point) {  
    System.out.println("Descendente de Point");  
}
```

# Operadores

OPERADOR	FUNÇÃO	OPERADOR	FUNÇÃO
+	Adição	~	Complemento
-	Subtração	<<	Deslocamento à esquerda
*	Multiplicação	>>	Deslocamento à direita
/	Divisão	>>>	Desloc. a direita com zeros
%	Resto	=	Atribuição
++	Incremento	+=	Atribuição com adição
--	Decremento	-=	Atribuição com subtração
>	Maior que	*=	Atribuição com multiplicação
>=	Maior ou igual	/=	Atribuição com divisão
<	Menor que	%=	Atribuição com resto
<=	Menor ou igual	&=	Atribuição com AND
==	Igual	=	Atribuição com OR
!=	Não igual	^=	Atribuição com XOR
!	NÃO lógico	<<=	Atribuição com desl. esquerdo
&&	E lógico	>>=	Atribuição com desloc. direito
	OU lógico	>>>=	Atrib. C/ desloc. a dir. c/ zeros
&	AND	? :	Operador ternário
^	XOR	(tipo)	Conversão de tipos (cast)
	OR	instanceof	Comparação de tipos

# Operadores de Atribuição

- Uma atribuição segue a seguinte sintaxe:
    - **variable-name** = **expression**;
  - A expressão é avaliada e o resultado é armazenado na variável, sobrescrevendo o valor existente.
  - '=' serve apenas para atribuição, em comparações usa-se '=='.
    - Copia o valor da variável ou constante do lado direito para a variável do lado esquerdo
- ```
x = 13; // copia a constante inteira 13 para x
y = x;  // copia o valor contido em x para y
```

*Rodrigo Fujieka - Ling de Prog 2*

# Operadores Aritméticos

- + adição
- - subtração
- \* multiplicação
- / divisão
- Operadores unários
  - -n e +n (ex.: -23) (em uma expressão: 13+ -12)
  - Melhor usar parênteses: 13 + (-12)
- Atribuição com operação
  - +=, -=, \*=, /=, %=
  - `x = x + 1;` equivale a `x += 1;`

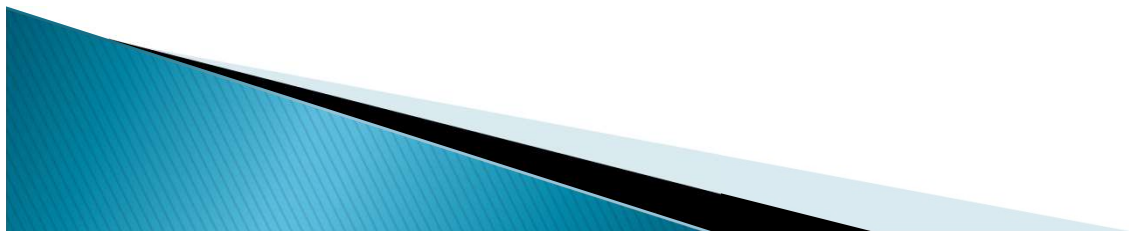
Rodrigo Fújeka - Ling de Prog 2





# Operadores Aritméticos

| Operador | Utilização   | Descrição                                     |
|----------|--------------|-----------------------------------------------|
| +        | $op1 + op2$  | Soma $op1$ e $op2$                            |
| -        | $op1 - op2$  | Subtrai $op2$ de $op1$                        |
| *        | $op1 * op2$  | Multiplica $op1$ por $op2$                    |
| /        | $op1 / op2$  | Divide $op1$ por $op2$                        |
| %        | $op1 \% op2$ | Calcula o resto da divisão de $op1$ por $op2$ |



# Operadores de incremento

- Exemplo

```
int a = 10;  
int b = 5;
```

- Incrementa ou decrementa antes de usar a variável

```
int x = ++a; // a contém 11, x contém 11  
int y = --b; // b contém 4, y contém 4
```

– a atribuição foi feita DEPOIS!

- Incrementa ou decrementa depois de usar a variável

```
int x = a++; // a contém 11, x contém 10  
int y = b--; // b contém 4, y contém 5
```

– a atribuição foi feita ANTES!

*Rodrigo Fujioka - Ling de Prog 2*

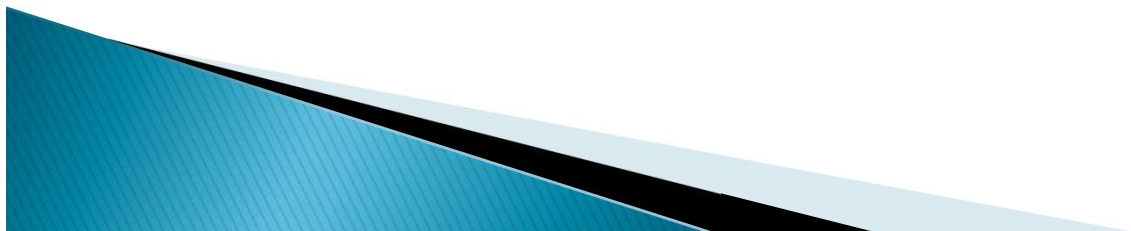
# Caracteres Especiais

| Caractere | Significado                                |
|-----------|--------------------------------------------|
| \n        | Insere uma nova linha                      |
| \t        | Dá uma espaço tipo TAB                     |
| \b        | Apaga o caracter anterior tipo o backspace |
| \r        | Retorno                                    |
| \f        | Avança página na impressora                |
| \\        | Inserir uma barra invertida                |
| \'        | Insere o apóstrofo                         |
| \"        | Insere aspas                               |
| \ddd      | Representa número octal                    |
| \xdd      | Representa número hexadecimal              |



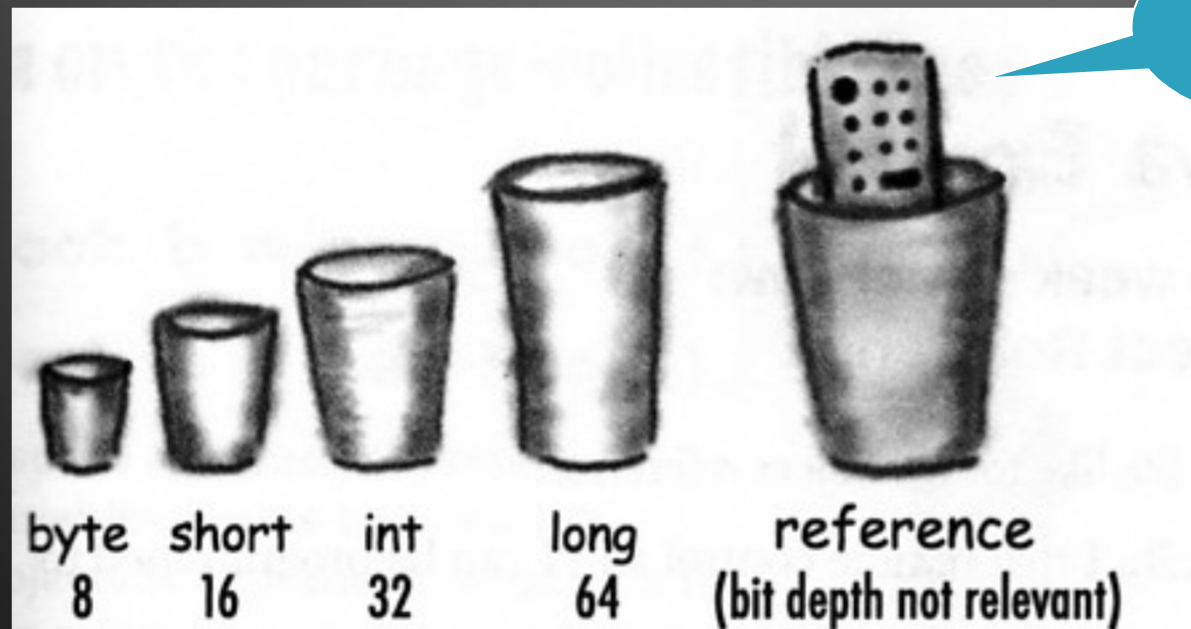
# Exercício 1 Aula 2

- ▶ Crie uma Programa Calculadora que efetue as operações básicas de uma calculadora.
- ▶ Use a classe Scanner para ler as informações do teclado.





# Variáveis que apontam pra objetos





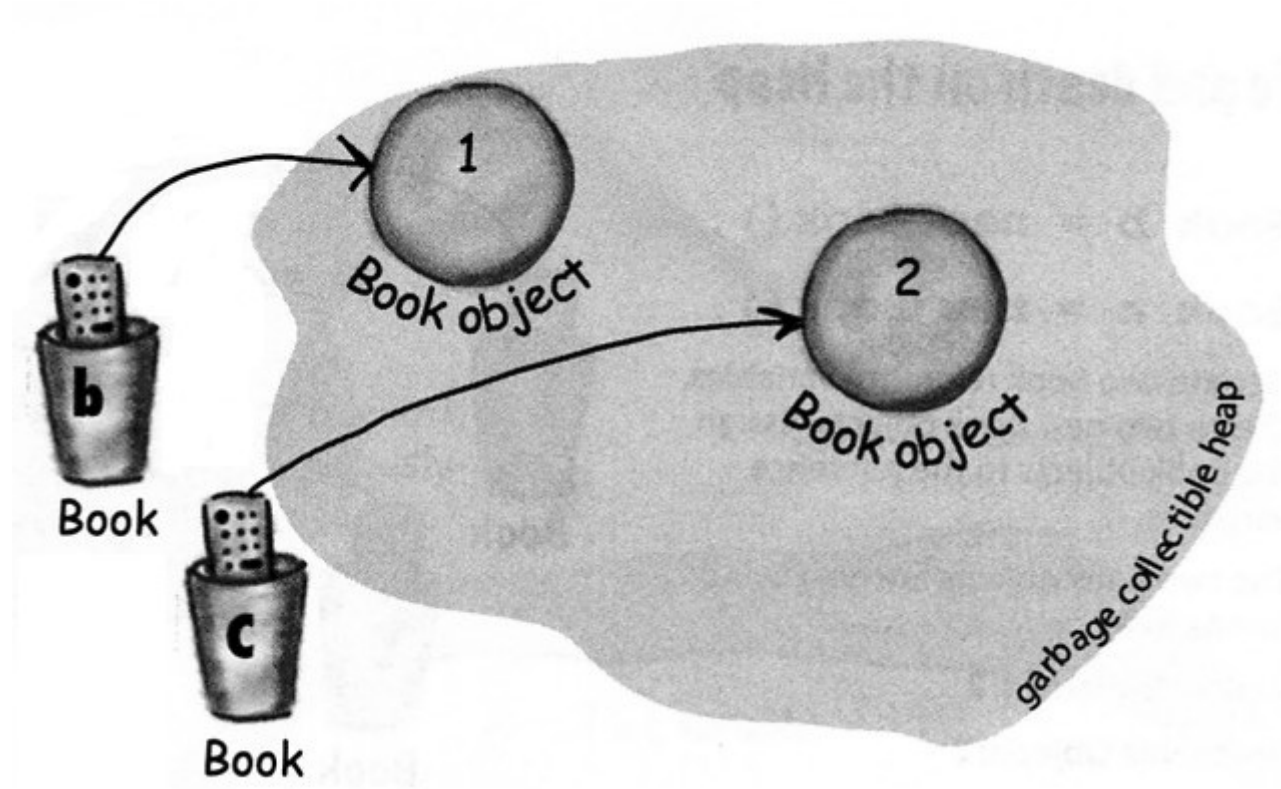
```
Dog d = new Dog();  
d.bark();
```

# Primitivos e objetos

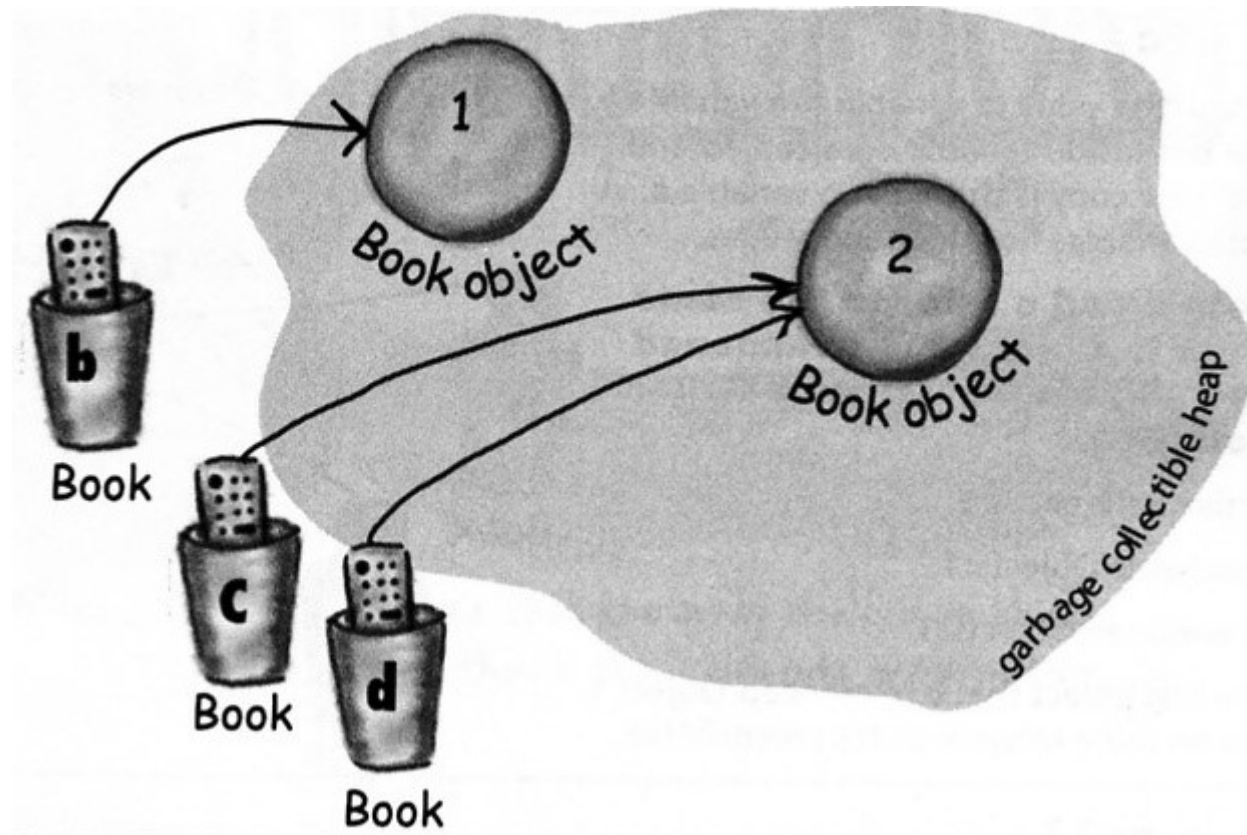
- ▶ Variáveis de tipos primitivos guardam o valor do tipo primitivo;
- ▶ Variáveis de objetos guardam o caminho pra se chegar no objeto (o controle remoto);



# Referências

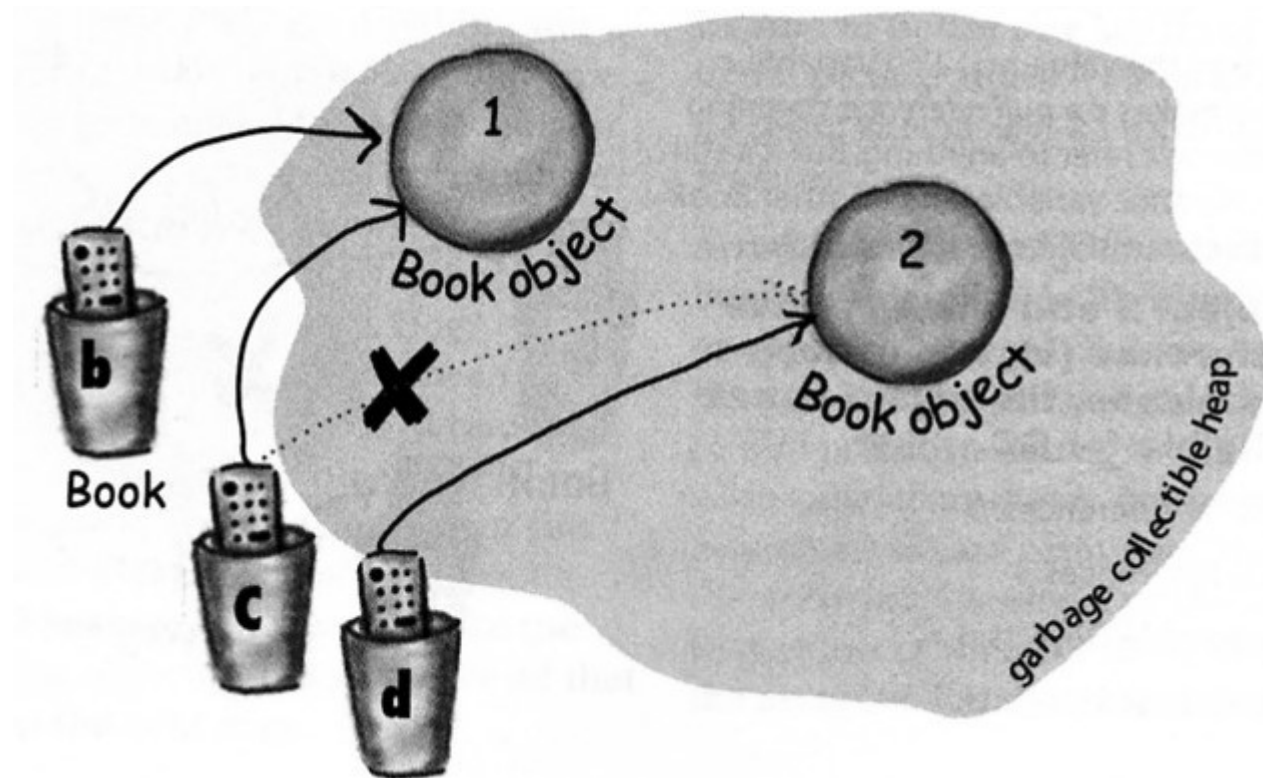


# Uma nova referência

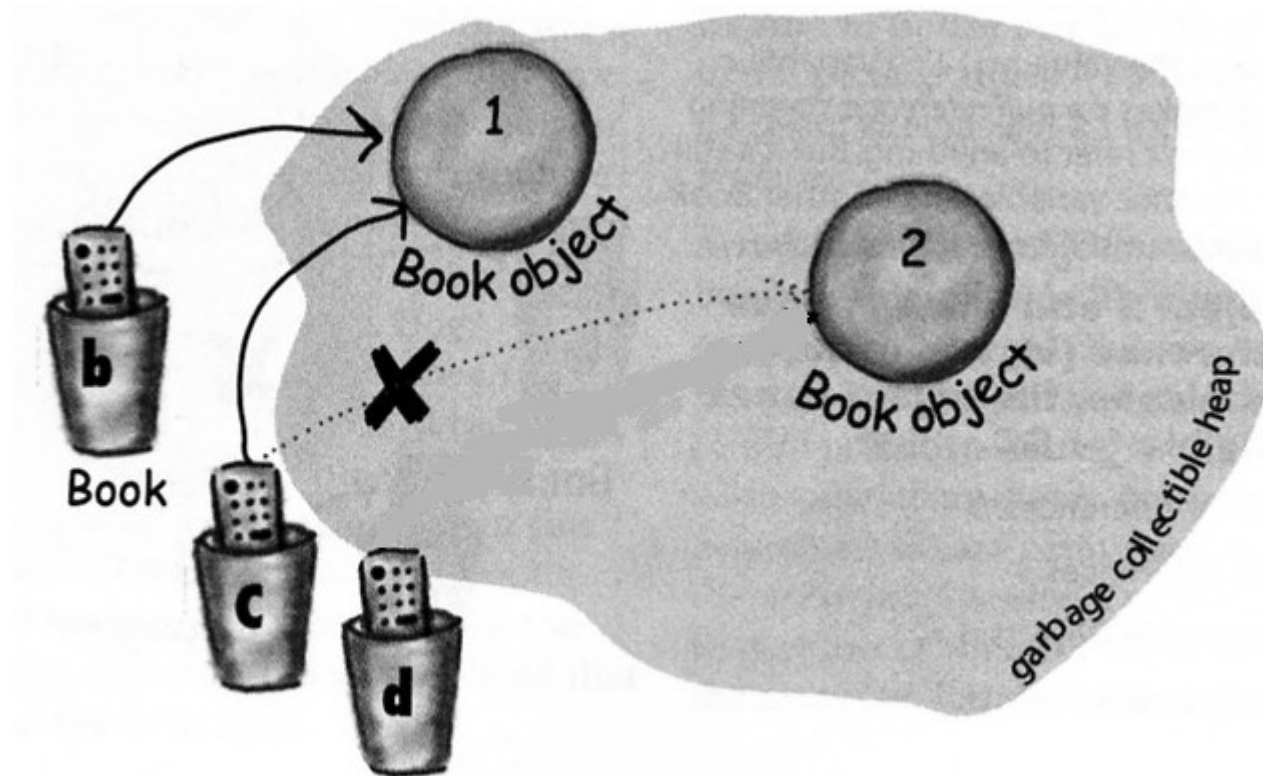




# E uma troca de referências

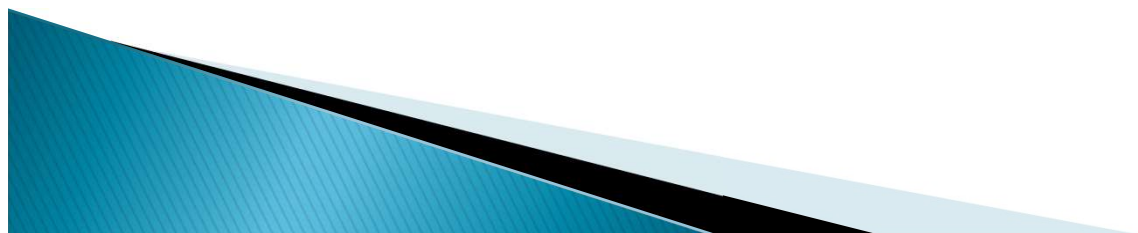


# E agora a televisão está fora do ar





# Mão Massa



# Controle de execução

- ▶ O controle da execução em Java utiliza os mesmos comandos existentes em outras linguagens
  - Repetição: **for**, **while**, **do-while**
  - Seleção: **if-else**, **switch-case**
  - Desvios (somente em estruturas de repetição): **continue**, **break**, rótulos
- ▶ Não existe o comando **goto**
  - **goto**, porém, é uma palavra reservada (keyword) da linguagem

# true e false

- ▶ Todas as expressões condicionais usadas nas estruturas for, if-else, while e do-while são expressões booleanas
  - O resultado das expressões deve ser sempre true ou false
  - Não há conversões automáticas envolvendo booleanos em Java (evita erros de programação comuns)

*Este código não  
compila em Java*

```
int x = 10;  
if (x = 5){  
    ...  
}
```

*Código aceito em  
Java*

```
int x = 10;  
if (x == 5){  
    ...  
}
```

# if-else

## ■ Sintaxe

```
if (expressão booleana)  
    instrução_simples;
```

```
if (expressão booleana) {  
    instruções  
}
```

```
if (expressão booleana) {  
    instruções  
} else if (expressão booleana) {  
    instruções  
} else {  
    instruções  
}
```

## ■ Exemplo

```
if ( ano < 0 ) {  
    System.out.println("Não é um ano!");  
} else if ( (ano%4==0 && ano%100!=0) || (ano%400==0) ) {  
    System.out.println("É bissexto!");  
} else {  
    System.out.println("Não é bissexto!");  
}
```

# return

- ▶ A palavra chave **return** tem duas finalidades
  - Especifica que um método irá retornar (se o método não tiver sido declarado como void)
  - Causa o retorno imediato a linha de controle imediatamente posterior à chamada do método
- ▶ Exemplos de sintaxe:

```
boolean método() {  
    if (condição) {  
        instrução;  
        return true;  
    }  
    resto do método  
    return false;  
}
```

```
void método() {  
    if (condição) {  
        instrução;  
        return;  
    }  
    mais coisas...  
}
```

# while e do-while

## ■ Sintaxe

```
while (expressão booleana )  
{  
    instruções;  
}
```

```
do  
{  
    instruções;  
} while (expressão booleana ) ;
```

## ■ Exemplos

```
int x = 0;  
while (x < 10) {  
    System.out.println ("item " + x);  
    x++;  
}
```

```
int x = 0;  
do {  
    System.out.println ("item " + x);  
    x++;  
} while (x < 10);
```

```
while ( true ) {  
    if (obj.z == 0) {  
        break;  
    }  
}
```

loop  
infinito!

# for

## ■ Sintaxe

```
for (  inicialização;  
      expressões booleanas;  
      passo da repetição )  
{  
    instruções;  
}
```

```
for (  inicialização;  
      expressões booleanas;  
      passo da repetição )  
  
    instrução_simples;
```

## ■ Exemplos

```
for ( int x = 0; x < 10; x++ ) {  
    System.out.println ("item " + x);  
}
```

```
for ( int x = 0, int y = 25;  
      x < 10 && (y % 2 == 0);  
      x++, y = y - 1 ) {  
    System.out.println (x + y);  
}
```

```
for ( ; ; ) {  
    if (obj.z == 0) {  
        break;  
    }  
}
```

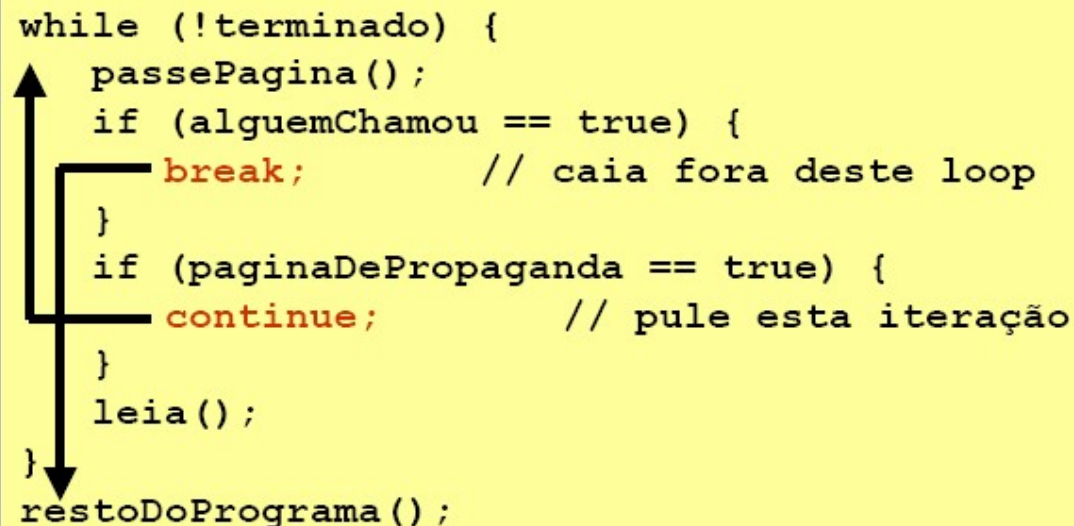
loop  
infinito!



# break e continue

- ▶ **break**: interrompe a execução do bloco de execução
  - Continua com a próxima instrução, logo após o bloco
- ▶ **continue**: interrompe a execução da iteração
  - Testa a condição e reinicia o bloco com a próxima iteração

```
while (!terminado) {  
    passePagina();  
    if (alguemChamou == true) {  
        break;           // caia fora deste loop  
    }  
    if (paginaDePropaganda == true) {  
        continue;       // pule esta iteração  
    }  
    leia();  
}  
restoDoPrograma();
```



# break e continue com rótulos

- ▶ break e continue sempre atuam sobre o bloco de repetição onde são chamados
- ▶ Em blocos de repetição contidos em outros blocos, pode-se usar **rótulos** para fazer break e continue atuarem em **blocos externos**
- ▶ Os rótulos só podem ser usados antes de do, while e for

```
revista: while (!terminado) {  
    for (int i = 10; i < 100; i += 10) {  
        passePagina();  
        if (textoChato) {  
            break revista;  
        }  
    }  
    maisInstrucoes();  
} restoDoPrograma();
```

break sem rótulo quebraria aqui!

## • Sintaxe:

ident: **do** {...}

**ou**

ident: **while** () {...}

**ou**

ident: **for** () { ... }

# Exercício

- ▶ Lista de Exercícios
- ▶ Projeto para Fixação (Registrando uma venda)

