

The Meta logo is displayed in a large, white, sans-serif font. The letter 'a' features a distinctive design element: a small, multi-colored flag (blue, green, and yellow) is attached to its right side, pointing upwards and to the right. The background is a dark blue overlay of a modern office interior with glass partitions, desks, and chairs.

Meta

Digital. Simple. Human.

DIGITAL TRANSFORMATION
CONSULTING & SERVICES

Java SE 11 Developer Certification 1Z0-819

Ministério da Justiça e
Segurança Pública |
MJSP



Apresentação - Unidade Curricular

Unidade Curricular: Java SE 11 Developer Certification
1Z0-819

Carga Horária: 24 horas

Objetivo geral: Permitir que desenvolvedores, profissionais de TI sejam referência desta tecnologia, testem e implementem

Proposta de trabalho

- Enfoque teórico e prático com uso de muitos exercícios, realizados em sala e em casa para fixação;
- Desenvolvimento de projetos com foco no mercado e com o uso da tecnologia estudada.

Planejamento

Semana 1

Introduction to Java

Primitive Types, Operators and Flow Control Statements

Text, Date, Time and Numeric Objects

Classes and Objects

Semana 2

Improved Class Design

Inheritance

Interfaces

Semana 3

Arrays and Loops

Collections

Java Streams API

Planejamento

Semana 4

Handle Exceptions and Fix Bugs

Java IO API

Java Concurrency and Multithreading

Java Modules

Cronograma – Aula

Aula	Conteúdo Programático	Carga Horária
01-06	Semana 1 (segunda, quarta)	6 horas
07-12	Semana 2 (segunda, quarta)	6 horas
Prática de Exercícios		
DOJO (2h)		
13-19	Semana 3 (segunda, quarta)	6 horas
20-24	Semana 4 (segunda, quarta)	6 horas
Prática de Exercícios		
DOJO (2h)		
Prática de Exercícios		
Test Killer		
Certificação		

Prova

Número de questões: 50

Score: 68% (34 questões)

Tempo: 90 minutos

Voucher: R\$ 1.292,00 (6 meses)

Java SE 11 Developer Certification 1Z0-819

[Introdução a Java]



O que é o Java?

- Uma tecnologia;
- Uma linguagem de programação;
- Uma plataforma de desenvolvimento;
- Um software distribuído pela Oracle;
- Um ambiente de execução de programas;
- Uma ilha da Indonésia (e o mar ao norte da ilha).



O que é o Java?

Edições:

- Java Card - Smart Card Edition
- Java ME - Micro Edition
- Java SE - Standard Edition
- Java EE - Enterprise Edition

Java SE é a base de tudo*

O que é o Java? - Java SE

- Ferramentas de desenvolvimento e API núcleo da plataforma (base para as demais);
- Permite o desenvolvimento de aplicações desktop, com interface gráfica, acesso à bancos de dados, I/O, acesso à rede, etc.;
- Dividida em:
 - JRE = *Java Runtime Environment*;
 - JDK = *Java Development Kit*.

O que é o Java? - Versões e edições do Java

- Standard Editions:

- Java 1.0 (1996);
- Java 1.1 (1997);
- J2SE 1.2 (1998);
- J2SE 1.3 (2000);
- J2SE 1.4 (2002);
- Java 1.5 / Java 5 (2004);
- Java SE 6 (2006);
- Java SE 7 (2011);
- Java SE 8 (2014);
- Java SE 9 (2017).
- Java SE 11
- Java SE 17

- Enterprise Editions:

- JPE project (1998);
- J2EE 1.2 (1999);
- J2EE 1.3 (2001);
- J2EE 1.4 (2003);
- Java EE 5 (2006);
- Java EE 6 (2009);
- Java EE 7 (2013);
- Java EE 8 (2016).

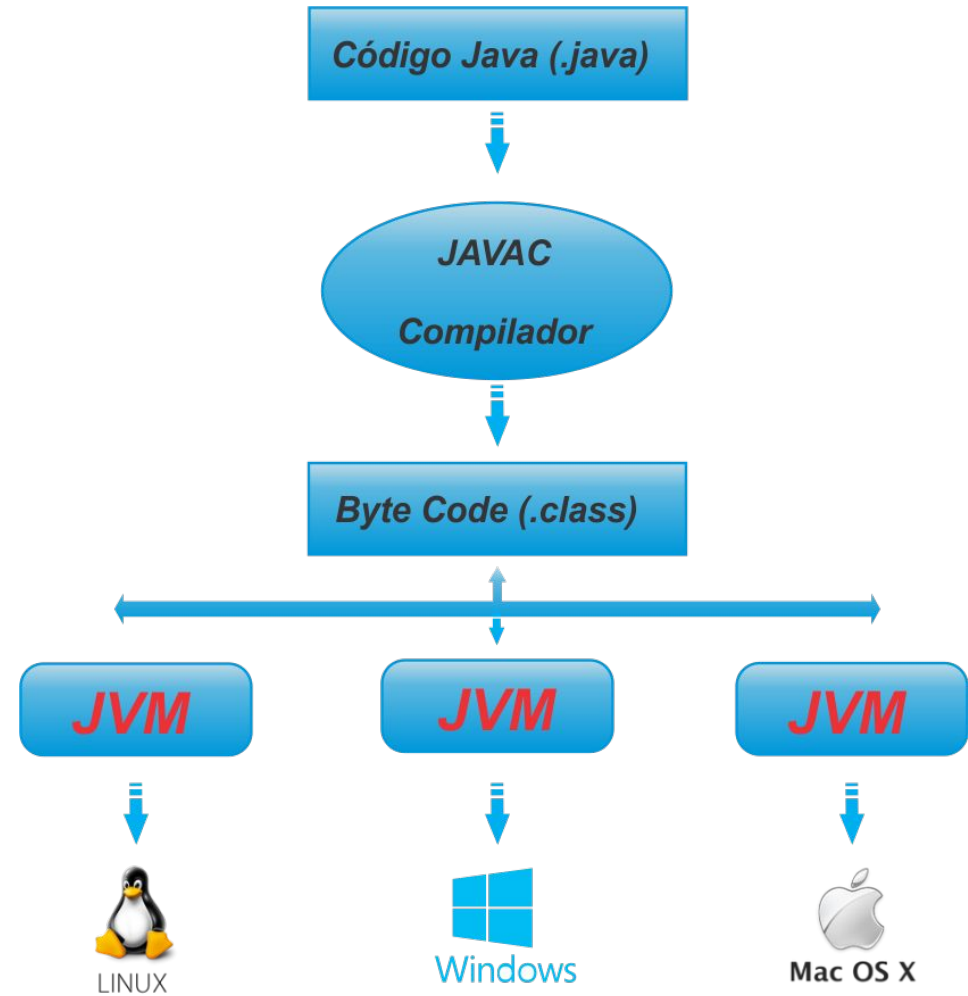
- Mobile Editions...

Como java funciona?

```
1 public class HelloWorld {  
2  
3     Run | Debug  
4     public static void main(String[] args) {  
5         System.out.println("Hello world");  
6     }  
7 }  
8
```

java HelloWorld

javac HelloWorld.java



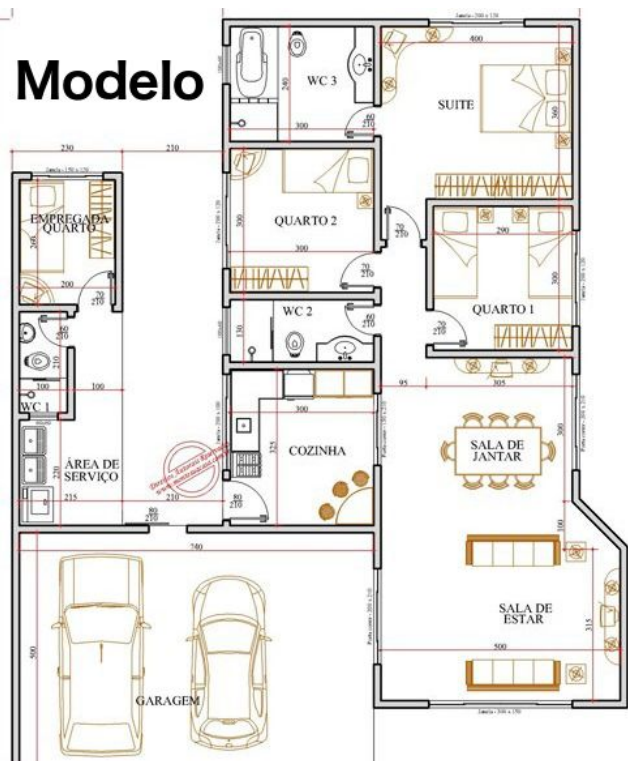
Classes

- Representa um conjunto de objetos com suas próprias características
- Uma **classe** define o comportamento dos **objetos**, através de **métodos** e quais estados (características) ele é capaz de manter, através de **atributos**.
- Java é estruturada em classes.
- Função, procedure, operação, método = comportamento
- variável, atributos, campos (field) = sinônimos = armazena informações

Objetos

- É uma **instância** de uma **classe**
- Um objeto é capaz de armazenar **estados** através de seus **atributos** e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos.

Classe x Objetos



Casa Construída



Classe



Instâncias da classe Casa (objetos)



```
Casa c1 = new Casa();  
c1.numero = 12;  
c1.cor = Color.Yellow;
```



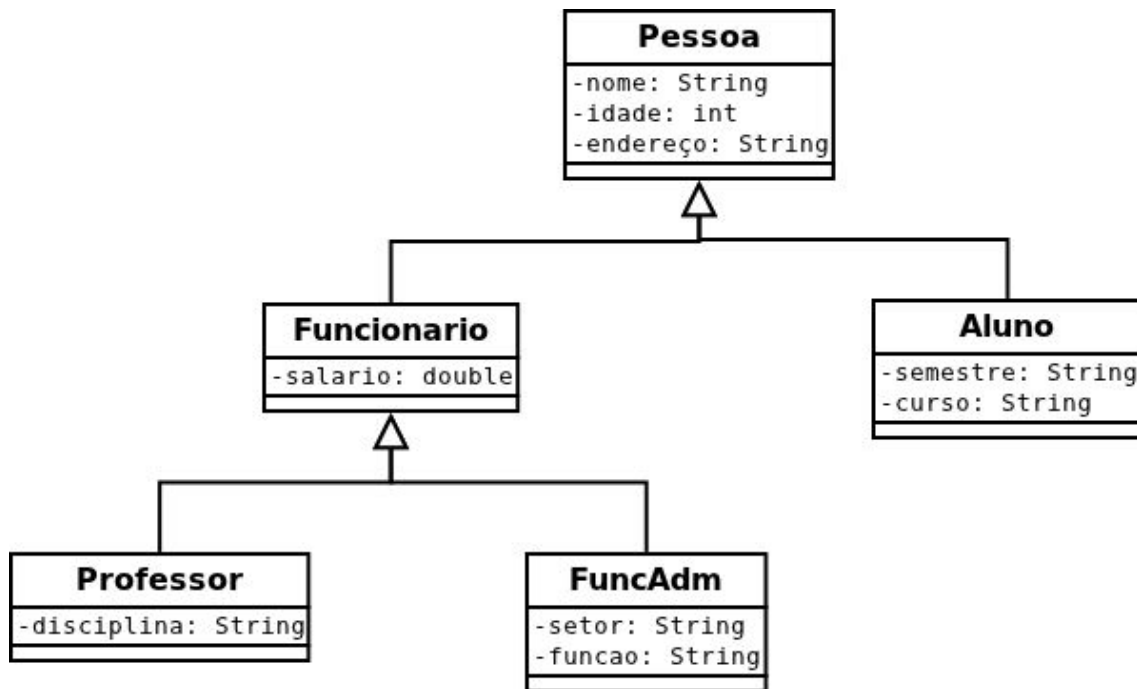
```
Casa c2 = new Casa();  
c2.numero = 56;  
c2.cor = Color.Red;
```



```
Casa c3 = new Casa();  
c3.numero = 72;  
c3.cor = Color.White;  
c3.abrePorta();
```

Herança

Em Java, podemos criar classes que herdem atributos e métodos de outras classes, evitando rescrita de código. Este tipo de relacionamento é chamado de **Herança**.



```
1 public class Pessoa {
2     public String nome;
3     public int idade;
4 }
```

```
1 public class Aluno extends Pessoa {
2     public String matricula;
3 }
```

Herança x Composição

Não está no escopo do treinamento discutir patterns ou anti-patterns.

Curioso sobre o assunto:

Publicado em 2013 -
<https://www.cs.auckland.ac.nz/~ewan/qualitas/studies/inheritance/TemperoYangNobleECOOP2013-pre.pdf>

Pubilcado em 2015 -
<https://www.thoughtworks.com/insights/blog/composition-vs-inheritance-how-choose>

JAVA APIs

JDK provê um conjunto de classes com vários propósitos:

- Básico: String, LocalDateTime, BigDecimal
- Manipular coleções: Enumeration, ArrayList, HashMap
- Tipos genéricos: System, Object, Class
- IO: FileInputStream, FileOutputStream
- Entre outras como: acesso a banco, concorrência, rede, mensageria, segurança, logging, gráfico.

<https://docs.oracle.com/en/java/javase/11/docs/api/>

Java Keywords, Reserved Words e Special Identifier

Keywords available since 1.0

Keywords no longer in use

Keywords added in 1.2

Keywords added in 1.4

Keywords added in 5.0

Keywords added in 9.0

Reserved words for literals values

Special identifier added in 10

if	implements	boolean	assert
else	extends	try	enum
continue	interface	catch	module
break	class	finally	requires
for	static	throw	transitive
do	final	throws	exports to
while	return	new	uses
switch	transient	this	provides
case	void	super	with
default	byte	instanceof	opens to
private	short	native	
protected	int	synchronized	true
public	long	volatile	false
import	char	goto	null
package	float	const	
abstract	double	strictfp	var

Java Naming Conventions

- Java é case-sensitive
- Pacote é um nome reservado para o domínio da empresa, adicionalmente o nome do sistema que a empresa irá adotar
- A convenção de nomenclatura em Java é uma regra a seguir quando se decide nomear seus identificadores, como classes, pacotes, variáveis, constantes, métodos, etc. Mas, não se é forçado a seguir. Portanto, é conhecido como convenção, não regra.

Java Naming Conventions

A seguir, estão as principais regras que devem ser seguidas por todos os identificadores:

- O nome não deve conter espaços em branco.
- O nome não deve começar com caracteres especiais como & (e comercial), \$ (dólar), _(underline).

Java Naming Conventions

- Classe
 - Deve começar com a letra maiúscula.
 - Deve ser um substantivo como Cor, Botão, Sistema, Tópico, etc.
 - Use palavras apropriadas, em vez de siglas.

```
public class Teste {  
    //codificação da classe omitida  
}
```


Java Naming Conventions

- Interface
 - Deve começar com a letra maiúscula
 - Deve ser um adjetivo como Runnable, Remote, ActionListener.
 - Use palavras apropriadas, em vez de siglas.

```
5 interface Testado {  
6     //codificação da classe omitida  
7 }
```

Java Naming Conventions

- Variável
 - Ela deve começar com uma letra minúscula, como id, nome, descricao.
 - Não deve começar com caracteres especiais como & (e comercial), \$ (dólar), _ (underline).
 - Se o nome contiver várias palavras, inicie-o com a letra minúscula seguida por uma letra maiúscula como primeiroNome, ultimoNome.
 - Evite usar variáveis de um caractere, como x, y, z.

```
public class Teste {  
    //variável  
    int id;  
    String descricao;  
    //codificação da classe omitida  
}
```

Java Naming Conventions

- Constante
 - Deve estar em letras maiúsculas, como VERMELHO, AMARELO.
 - Se o nome contiver várias palavras, ele deverá ser separado por um underline (_), como TAMANHO_MINIMO.
 - Pode conter dígitos, mas não como a primeira letra.

```
5  public class Teste {  
6      //constante  
7      final static int TAMANHO_MINIMO = 18;  
8      //codificação da classe omitida  
9  }
```

Java Naming Conventions

- Método
 - Deve começar com letra minúscula.
 - Deve ser um verbo como main(), print(), println().
 - Se o nome contiver várias palavras, inicie-o com uma letra minúscula seguida por uma letra maiúscula, como actionPerformed().

```
5  class Teste {  
6      //método  
7      void imprimir() {  
8          //codificação do método omitida  
9      }  
10 }
```

Java Naming Conventions

- Método - Getters e Setters
 - Os getters devem ter o nome começando por get seguido de uma letra maiúscula, ter o retorno diferente de void e não ter parâmetros. Se o tipo de retorno for boolean (o tipo primitivo), o prefixo pode ser tanto get quanto is. Alguns frameworks permitem que o prefixo is possa ser usado também quando o tipo de retorno é Boolean (o Objeto).
 - Os setters devem ter o nome começando por set seguido de uma letra maiúscula, ter exatamente um parâmetro, e ter retorno void. Alguns os frameworks permitem que o retorno possa ser do tipo da própria classe que declara o método. Além disso, na maioria dos casos, o tipo do parâmetro do setter deve ser o mesmo que o retorno do getter.

Java Naming Conventions

- Método - Getters e Setters

```
1  public class Teste {  
2      // variavel  
3      int id;  
4      String descricao;  
5      // constante  
6      final static int TAMANHO_MINIMO = 18;  
7  
8      public int getId() {  
9          return id;  
10     }  
11  
12     public void setId(int id) {  
13         this.id = id;  
14     }  
15  
16     public String getDescricao() {  
17         return descricao;  
18     }  
19  
20     public void setDescricao(String descricao) {  
21         this.descricao = descricao;  
22     }  
23     // codificacao da classe omitida  
24 }  
25
```

Java Naming Conventions

- Pacote
 - Deve ser uma letra minúscula, como java, lang. - Se o nome contiver várias palavras, ele deverá ser separado por pontos (.) Como java.util, java.lang, br.com.meta.sistemaxpto

```
1  package br.com.meta.sistemaxpto;  
2  
3  public class Teste {  
4      // variavel  
5      int id;  
6      String descricao;  
7      // constante  
8      final static int TAMANHO_MINIMO = 18;  
9  
10     public int getId() {  
11         return id;  
12     }  
13 }
```

Java Naming Conventions

- CamelCase (upper case | lower case)
 - CamelCase é utilizado em convenções de nomenclatura Java.
 - Java segue a sintaxe CamelCase (upper case) para nomear a classe, interface,
 - Classes:
 - Pessoa, PessoaJuridica, PessoaFisica
 - Para métodos e variáveis, camelCase (lower case)
 - Métodos:
 - addPessoa(); removePessoaJuridica();

Java Naming Conventions

- Linhas em Branco
 - Duas linhas em branco devem sempre ser usadas nas seguintes circunstâncias:
 - Entre as seções de um arquivo de código
 - Entre as definições de classe e interface
 - Uma linha em branco deve sempre ser usada nas seguintes circunstâncias:
 - Entre métodos
 - Entre as variáveis locais em um método e sua primeira declaração
 - Antes de um bloco ou comentário de linha única
 - Entre seções lógicas dentro de um método para melhorar a legibilidade

Java Naming Conventions

- Espaços em Branco
 - Espaços em branco devem ser usados nas seguintes circunstâncias:
 - Uma palavra-chave seguida por parênteses deve ser separada por um espaço.

```
while (condicao()) {  
    //Comentários de Linha Única  
    /* Lidar com a condição. */  
}
```

Sintaxe Java

- Final de instrução no java é como “;” (ponto e vírgula)
- Blocos devem estar inseridos entre chaves “{“ e “}”
- Indentação e espaços ajudam na “leitura”, porém semanticamente é irrelevante.

```
package com.oracle.demos.animals;  
class Dog {  
    void fetch() {  
        while (ball == null) {  
            keepLooking();  
        }  
    }  
    void makeNoise() {  
        if (ball != null) {  
            dropBall();  
        } else {  
            bark();  
        }  
    }  
}
```

Definir uma classe Java

- Classe java deve estar contida dentro de um arquivo .java
- Pacotes representam pastas e são onde os arquivos são salvos.
- Pacote deve representar o nome da empresa/compania (domínio) e o nome da aplicação.
 - Exemplo: br.com.meta.mjsp.treinamento
- Pacote e classe devem formar uma combinação única

```
package <package name>;  
class <ClassName> {  
  
}  
  
package com.oracle.demos.animals;  
class Dog {  
    // the rest of this class code  
}
```

```
/somepath/com/oracle/demos/animals/Dog.java
```

Usar modificadores de acesso

Pra começar existem somente 3 modificadores (private, protected e public), e com isso temos 4 níveis de visibilidade

Os níveis são os que você disse: private, default, protected e public

Usar modificadores de acesso

- Private: A única classe que tem acesso ao atributo é a própria classe que o define, ou seja, se uma classe Pessoa declara um atributo privado chamado nome, somente a classe Pessoa terá acesso a ele.
- Default: Tem acesso a um atributo default (identificado pela ausência de modificadores) todas as classes que estiverem no mesmo pacote que a classe que possui o atributo.
- Protected: Esse é o que pega mais gente, ele é praticamente igual ao default, com a diferença de que se uma classe (mesmo que esteja fora do pacote) estende da classe com o atributo protected, ela terá acesso a ele. Então o acesso é por pacote e por herança.
- Public: Todos tem acesso

Usar modificadores de acesso

Modifier	Class	Package	Subclasses	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

Acesso de classes através de pacotes

Para acessar uma classe de outro pacote:

- Prefixo é o nome do pacote, seguido pelo nome;
- Usar a palavra reserva ***import*** para especificar outras classes dentro do contexto de uso



Usar modificadores de acesso

```
package <package name>;  
import <package name>.<class name>;  
import <package name>.*;  
<access modifier> class <ClassName> {  
    <access modifier> <variable definition>  
    <access modifier> <method definition>  
}
```

```
package b;  
import a.*;  
public class Y extends X {  
    public void doThings() {  
        X x = new X();  
        x.y1;  
        x.y2;  
        x.y3;  
        x.y4;  
    }  
}
```

```
package a;  
public class X {  
    public Y y1;  
    protected Y y2;  
    Y y3;  
    private Y y4;  
}
```



Criando uma aplicação

- O método main é o “entry point” (início) de uma aplicação
- Ele é o “ponto de partida” para execução de um programa
- O método **deve** ser chamado de **main**
- Ele deve ser:
 - publico
 - statico
 - void (não deve retornar valor)
 - Deve aceitar uma array de string como parâmetro

```
/**
 * Comentário de documentação
 * @author Nome_Pessoa
 * @version 1.0.0
 * @since
 */
public class ExemploClasse {

    public static void main(String[] args) {

    }

}
```

Compilando um programa java

Java -cp CLASSPATH é necessário se você deseja especificar todo o código no classpath.

- -cp
 - path de outras classes requeridas para compilação
- -d
 - path da compilação resultante

```
javac -cp /project/classes -d /project/classes /project/sources/demos/Whatever.java
```

```
package demos;  
public class Whatever {  
    public static void main(String[] args) {  
        // program execution starts here  
    }  
}
```



javac



```
/project/classes/demos/Whatever.class
```

Executando um programa Java

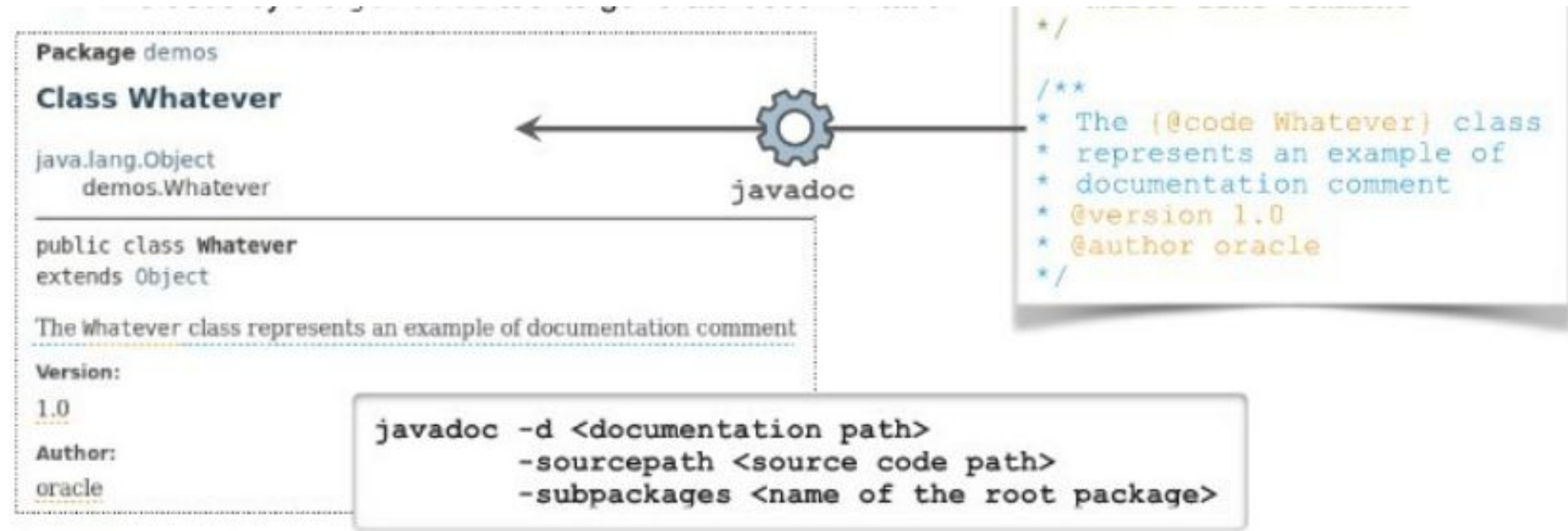
java -cp \$CLASS_PATH \$PACOTE.\$CLASSE \$PARAMETROS

```
package demos;  
public class Whatever {  
    public static void main(String[] args) {  
        String param1 = args[1];  
        System.out.println("Hello "+param1);  
    }  
}
```

```
java -cp /project/classes demos.Whatever Jo John "A Name" Jane  
Hello John
```

```
java /project/sources/demos/Whatever.java
```

Comentários e Documentação



Java SE 11 Developer Certification 1Z0-819

[Tipos primitivos,
operadores e fluxos de
controles]



Declaração e inicialização de variáveis do tipo primitivo

		Valores possíveis		Valor Padrão	Tamanho	Exemplo
Tipos	Primitivo	Menor	Maior			
Inteiro	byte	-128	127	0	8 bits	byte ex1 = (byte)1;
	short	-32768	32767	0	16 bits	short ex2 = (short)1;
	int	-2.147.483.648	2.147.483.647	0	32 bits	int ex3 = 1;
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits	long ex4 = 1l;
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits	float ex5 = 5.50f;
	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits	double ex6 = 10.20d; ou double ex6 = 10.20;
Caractere	char	0	65535	\0	16 bits	char ex7 = 194; ou char ex8 = 'a';
Booleano	boolean	false	true	false	1 bit	boolean ex9 = true;

Declaração e inicialização de variáveis do tipo primitivo

- Para otimização de memória, recomenda-se, sempre que possível, utilizar arrays de **byte** ou **short** ao invés de **int** ou **long**.
- O mesmo é válido para arrays de **float** ao invés de **double**.
- Os tipos float e double **NUNCA** devem ser usados para valores precisos, como moeda, por exemplo. Para isso, recomenda-se utilizar a classe **java.math.BigDecimal**.
- Para cadeias de caracteres a linguagem Java provê suporte através da classe **java.lang.String**.
- A classe String não é *tecnicamente* um tipo primitivo mas devido a sua importância é tratado como se fosse.
- Objetos do tipo String são **imutáveis**.

Literais

Um literal é simplesmente a representação do código-fonte dos tipos de dados primitivos – em outras palavras, um inteiro, um ponto flutuante, um booleano ou caractere que você digite enquanto escreve o código. A seguir temos exemplos de literais primitivos.

Literais

Diagram illustrating variable declarations and assignments with annotations:

- Declaração do Tipo da Variável**: Points to `boolean` in `boolean result = true;`
- Declaração Variável**: Points to `result` in `boolean result = true;`
- Valor Atribuído**: Points to `true` in `boolean result = true;`
- Atribuição de Valor**: Points to the assignment part of `capitalC = 'C';`

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

Representação de tipos numéricos

```
int decimal    = 26;           //O número 26 em decimal
int octal      = 032;          //O número 26 em octal
int hexa       = 0x1a;         //O número 26 em hexadecimal
```



Representação de Pontos Flutuantes

```
double d1 = 123.4;  
double d1 = 123.4D; //ou d  
double d2 = 1.234e2; //o mesmo valor de d1, mas em notação científica  
float f1 = 123.4f; //ou F
```

Representação de Caracteres

```
char    p    =  'C';  
char    a    =  '\u0043';    // Valor    Unicode  
String   s    =  "Linguagem    Java";  
String   x    =  "Linguagem    \u0043#";    //Linguagem    C#
```



`\b`: Backspace

`\t`: TAB

`\n`: Quebra de Linha

`\f`: Alimentação de Folha

`\r`: Retorno de Linha

`\"`: Aspas Duplas


`\'`: Aspas Simples

`\\`: Barra Invertida

Restrições de declaração e inicialização

- Variáveis devem ser inicializadas antes do uso;
- Um *bigger type* não pode ser convertido para um *smaller type*
- Caracteres **não devem** utilizar àspas duplas. Use aspas simples ‘ ‘
- Um valor de caractere não pode conter mais de 1 caractere
- Valores booleanos podem ser expresso como true ou false;

```
byte a;  
byte b = a;  
byte c = 128;  
int d = 42L;  
float e = 1.2;  
char f = "a";  
char g = 'AB';  
boolean h = "true";  
boolean i = 'false';  
boolean j = 0;  
boolean k = False;
```



A expressão da direita é atribuída à variável da esquerda:

```
- int var1 = 0, var2 = 0;  
- var1 = 50; // var1 recebe o valor de 50  
- var2 = var1 + 10; // var2 ficou igual a 60
```

- A expressão da direita é sempre avaliada antes da atribuição.
- As atribuições podem ser agrupadas:

```
var1 = var2 = var3 = 50;
```


Realizam operações aritméticas básicas

- Operam sobre variáveis e literais numéricos

```
int a, b, c, d;
```

```
a = 2 + 2; // adição
```

```
b = a * 3; // multiplicação
```

```
c = b - 2; // subtração
```

```
d = b / 2; // divisão
```

```
e = b % 2; // retorna o resto da divisão
```

-
- A maioria das operações resultam num int ou long:

```
- byte b1 = 1, b2 = 2, b3;
```

```
- b3 = b1 + b2; // error: result is an int // b3 is byte
```

- Valores byte, char, e short são promovidos a int antes da operação.
- Se algum argumento for long, o outro é promovido a long, e o resultado é long.

Operadores Java

Cinco tipos de operadores:

- Atribuição
- Aritméticos
- Manipulação de bits
- Relacionais
- Condicionais

Operadores Java

Operators	Precedence
postfix increment and decrement	++ --
prefix increment and decrement, and unary	++ -- + - ~ !
multiplicative	* / %
additive	+ -
bit shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=

Operadores matemáticos

Matemáticos:

= + - * / %

acumulativos*:

+=

-=

*=

/=

%=

Separador

()

```
int a = 1;    // assignment (a is 1)
int b = a+4;  // addition (b is 5)
int c = b-2;  // subtraction (c is 3)
int d = c*b;  // multiplication (d is 15)
int e = d/c;  // division (e is 5)
int f = d%6;  // modulus (f is 3)
```

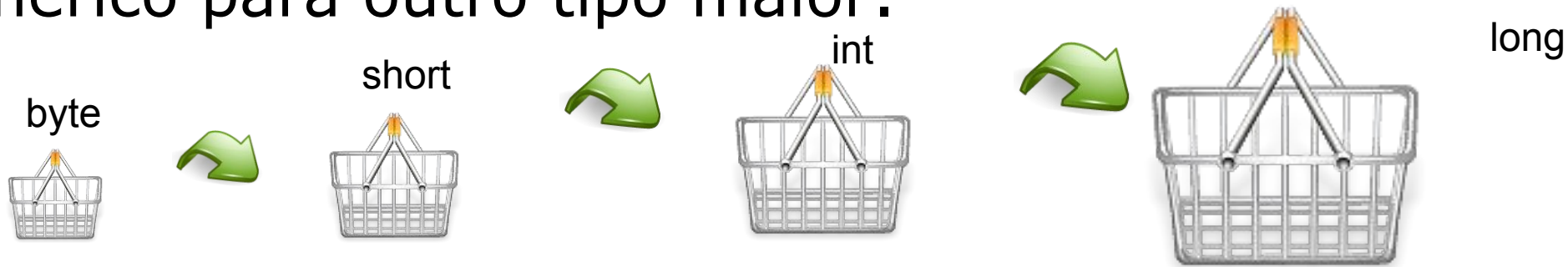
```
int a = 1, b = 3;
a += b; // equivalent of a=a+b (a is 4)
a -= 2; // equivalent of a=a-2 (a is 2)
a *= b; // equivalent of a=a*b (a is 6)
a /= 2; // equivalent of a=a/2 (a is 3)
a %= a; // equivalent of a=a%a (a is 0)
```

```
int a = 2, b = 3;
int c = b-a*b; // (c is -3)
int d = (b-a)*b; // (c is 3)
```

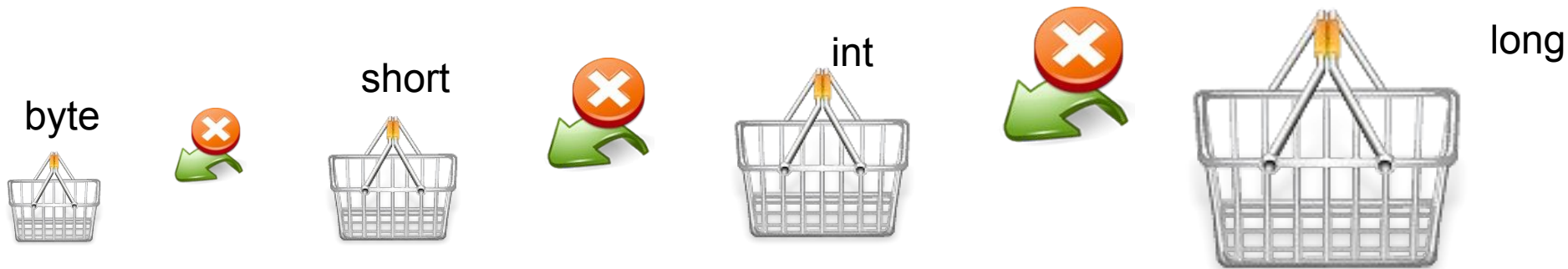
```
int a = 1, b = 0;
a++; // increment (a is 2)
++a; // increment (a is 3)
a--; // decrement (a is 2)
--a; // decrement (a is 1)
b = a++; // increment postfix (b is 1, a is 2)
b = ++a; // increment prefix (b is 3, a is 3)
b = a--; // decrement postfix (b is 3, a is 2)
b = --a; // decrement prefix (b is 1, a is 1)
```

Operadores matemáticos e Type Casting

O Java converte automaticamente valores de um tipo numérico para outro tipo maior.



- O Java não faz automaticamente o “downcast.”



Operadores matemáticos e Type Casting

```
byte a = 127, b = 5;
❌ byte c = a+b;           // compilation fails
    int
✅ int d = a + b;          // d is 132
⚠️ byte e = (byte) (a+b);  // e is -124 (type overflow, because 127 is the max byte value)
⚠️ int f = a/b;            // f is 25 (a/b is 25 because it is an int)
⚠️ float g = a/b;          // g is 25.0F (result of the a/b can be implicitly or
⚠️ float h = (float) (a/b); // h is 25.0F explicitly casted to float, but a/b is still 25)
✅ float i = (float) a/b;   // i is 25.4F (when either a or b
✅ float j = a/(float) b;   // j is 25.4F is float the a/b becomes float)
⚠️ b = (byte) (b+1);        // explicit casting is required, because b+1 is an int
✅ b++;                    // no casting is required for ++ and -- operators
✅ char x = 'x';
✅ char y = ++x;           // arithmetic operations work with character codes
```

Mais Operadores matemáticos

java.lang.Math

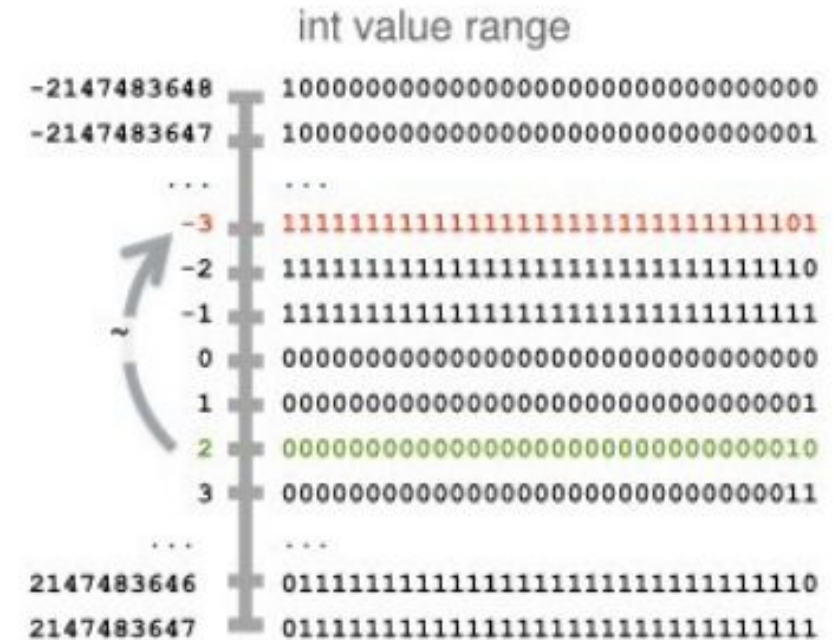
- Módulo
- máximo
- mínimo
- arredondamento
- Exemplo:
 - Math.abs(numero)
 - Math.min(num1,num2)
 - Math.max(num1,num2)
 - Math.ceil(numero)
 - Math.floor(numero)

<https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/lang/Math.html>

Representação de número binários

<https://docs.oracle.com/javase/8/docs/technotes/guides/language/binary-literals.html>

```
int a = 2; //  
int b = ~a; // b is -3
```



<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Representação de número binários

```
public class BinaryLiteralsExample {  
  
    public static void main(String[] args) {  
        // Binary literal in byte type  
        byte b1 = 0b101;    // Using b0, The b can be lower or upper case  
        byte b2 = 0B101;    // Using B0  
        System.out.println("-----Binary Literal in Byte-----");  
        System.out.println("b1 = "+b1);  
        System.out.println("b2 = "+b2);  
  
        // Binary literal in short type  
        short s1 = 0b101;    // Using b0, The b can be lower or upper case  
        short s2 = 0B101;    // Using B0  
        System.out.println("-----Binary Literal in Short-----");  
        System.out.println("s1 = "+s1);  
        System.out.println("s2 = "+s2);  
  
        // Binary literal in int type  
        int i1 = 0b101;    // Using b0, The b can be lower or upper case  
        int i2 = 0B101;    // Using B0  
        System.out.println("-----Binary Literal in Integer-----");  
        System.out.println("i1 = "+i1);  
        System.out.println("i2 = "+i2);  
  
        // Binary literal in long type  
        long l1 = 0b0000011111100001;    // Using b0, The b can be lower or upper case  
        long l2 = 0B0000011111100001;    // Using B0  
        System.out.println("-----Binary Literal in Long-----");  
        System.out.println("l1 = "+l1);  
        System.out.println("l2 = "+l2);  
    }  
}
```

```
-----Binary Literal in Byte-----  
b1 = 5  
b2 = 5  
-----Binary Literal in Short-----  
s1 = 5  
s2 = 5  
-----Binary Literal in Integer-----  
i1 = 5  
i2 = 5  
-----Binary Literal in Long-----  
l1 = 2017  
l2 = 2017
```

Bitwise Operators

- Os operadores de bits trabalham em bits e operam individualmente (bit-by-bit). Os operadores bit a bit podem ser usados com int, short e char. Podemos usar operadores bit a bit quando realizamos uma atualização ou desejamos consultar operadores de uma árvore indexada binária.

Bitwise Operators

- Bitwise OR, representado simbolicamente como `|`.
- Bitwise AND, representado simbolicamente como `&`.
- Bitwise XOR, representado no código como `^`.
- Complemento bit a bit, representado com `~`.
- Operadores de deslocamento bit a bit:
 - Deslocamento à direita assinado, representado como `>>`
 - Deslocamento à esquerda assinado, representado como `<<`.
 - Deslocamento à direita não assinado, indicado com `>>>`.
 - Deslocamento à esquerda sem sinal indicado com `<<<`.

Bitwise Operators - OR

- O OR operador retorna 1 se achar que pelo menos um dos operandos é 1; caso contrário, 0. A seguir está a tabela verdade para dois operandos, X e Y, que podemos usar para entender o OR operador bit a bit.

X	Y	X Y
0	0	0
0	1	1
1	0	1
1	1	1

```
public class BitwiseOr{  
    public static void main(String[] args){  
        int x = 8, y = 9;  
        System.out.println("x | y = " + (x | y));  
    }  
}
```

Resultado:

x | y = 9

Bitwise Operators - AND

Este operador binário é denotado por & e retorna 1 se ambos os bits forem 1; caso contrário, retorna 0. A tabela verdade a seguir demonstra o AND operador onde X e Y são dois operandos e aceitam apenas valores binários (1 ou 0).

X	Y	X&Y
0	0	0
0	1	0
1	0	0
1	1	1

```
public class BitwiseAnd{  
    public static void main(String[] args){  
        int x = 8, y = 9;  
        System.out.println("x & y = " + (x & y));  
    }  
}
```

Resultado:

x & y = 8

Bitwise Operators - XOR

XOR O operador binário (também conhecido como exclusivo OR) é indicado com o símbolo ^ que retorna 1 se ambos os bits forem diferentes. Obtemos 1 se o operando X for 1 e Y for 0, ou Y for 1 e X for 0. Obtemos o resultado como 0 se ambos forem 1 ou 0.

X	Y	X^Y
0	0	0
0	1	1
1	0	1
1	1	0

```
public class BitwiseXor{  
    public static void main(String[] args){  
        int x = 8, y = 9;  
        System.out.println("x ^ y = " + (x ^ y));  
    }  
}
```

Resultado:

$x \wedge y = 1$

Bitwise Operators - ~ (TILDE) - Complemento

O operador de complemento retorna o inverso invertendo cada bit de 0 para 1 e 1 para 0.

X	~X
0	1
1	0

```
public class BitwiseComplement{  
    public static void main(String[] args){  
        int x = 8;  
        System.out.println("~x= " + (~x));  
    }  
}
```

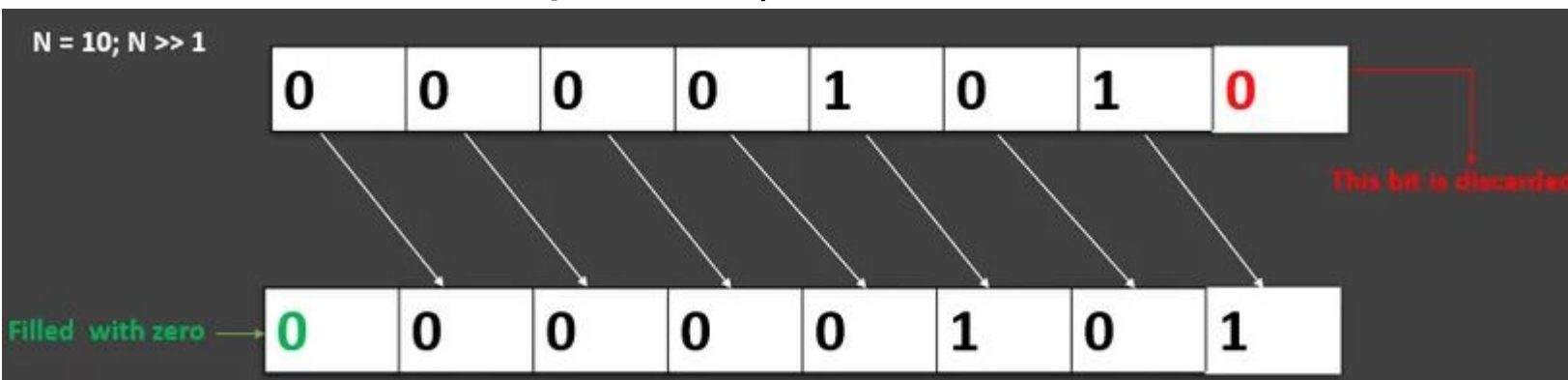
Resultado:

~x= -9

Operador de deslocamento à direita assinado

Variável $\ll n = \text{variavel} / (2^n)$

- O deslocamento à direita com sinal também é chamado de operador de deslocamento à direita bit a bit, denotado com \gg e desloca um padrão de bits para a direita por um determinado número de bits. Os bits mais à direita são descartados e a posição mais à esquerda é preenchida com o bit de sinal.
- Esse deslocamento faz com que algumas posições vagas na extremidade esquerda sejam preenchidas com zeros (porque não temos nenhum bit de sinal neste exemplo que o represente como um número positivo).



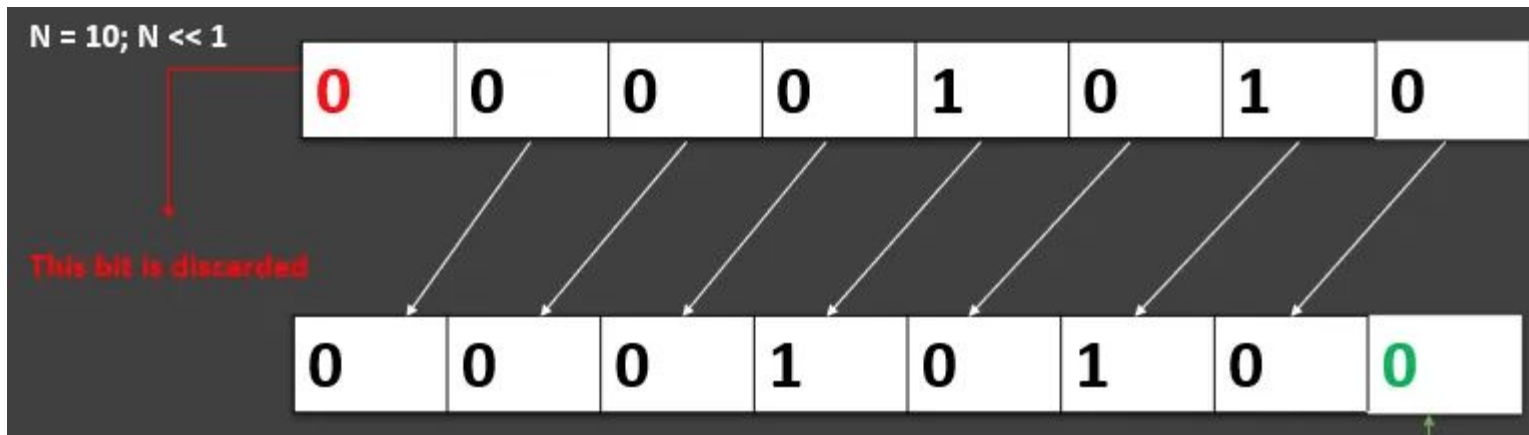
```
public class BitwiseSignedRightShift{  
    public static void main(String[] args){  
        int x = 10;  
        System.out.println("10 >> 1 = " + (x >> 1));  
    }  
}
```

10 >> 1 = 5

Operador de deslocamento à esquerda assinado

- $\text{Variable} \ll n = \text{Variable} \cdot 2^n$

- O operador de deslocamento à esquerda desloca os bits para a esquerda pelo número de vezes especificado pelo lado direito do operando. Após o deslocamento para a esquerda, o espaço vazio à direita é preenchido com 0.
- Outro ponto importante a ser observado é que deslocar um número por um é equivalente a multiplicá-lo por 2, ou, em geral, deslocar um número por n posições à esquerda é equivalente a multiplicar por 2^n .



```
public class BitwiseSignedLeftShift{  
    public static void main(String[] args){  
        int x = 10;  
        System.out.println("10 << 1 = " + (x << 1));  
    }  
}
```

$10 \ll 1 = 20$

Operador de deslocamento à direita não assinado

- Este operador é muito semelhante ao operador de deslocamento à direita com sinal. A única diferença é que os espaços vazios à esquerda são preenchidos com 0, independentemente de o número ser positivo ou negativo. Portanto, o resultado será sempre um número inteiro positivo.

```
public class BitwiseUnsignedRightShift{  
    public static void main(String[] args){  
        int x = 40;  
        System.out.println("40 >>> 2 = " + (x >>> 2));  
    }  
}
```

40 >>> 2 = 10? Como se calcula?

Operador de deslocamento à esquerda não assinado

- O operador de deslocamento à esquerda sem sinal (indicado por <<<) não é suportado na programação Java porque <<e <<<são operações idênticas.

`Prove. Desafio`

Operadores: comparação, igualdade e relacionais

```
int a = 3, b = 2;
boolean c = false;
c = (a == b);           // c is false
c = !(a == b);          // c is true
c = (a != b);           // c is true
c = (a > b);             // c is true
c = (a >= b);            // c is true
c = (a < b);             // c is false
c = (a <= b);            // c is false
c = (a > b && b == 2);   // c is true
c = (a < b && b == 2);   // c is false
c = (a < b || b == 2);   // c is true
c = (a < b || b == 3);   // c is false
c = (a > b ^ b == 2);    // c is false
```

Short-Circuit Evaluation

A operação é realizada quando o “lado esquerdo” é verdadeiro (true).

`&& ||` (short-circuit evaluation)

`& | ^` (full evaluation)

```
true && evaluated
false && not evaluated
false & evaluated
false || evaluated
true || not evaluated
true | evaluated
true ^ evaluated
false ^ evaluated
```

```
int a = 3, b = 2;
boolean c = false;
c = (a > b && ++b == 3); // c is true, b is 3
c = (a > b && ++b == 3); // c is false, b is 3
c = (a > b || ++b == 3); // c is false, b is 4
c = (a < b || ++b == 3); // c is true, b is 4
c = (a < b | ++b == 3); // c is true, b is 5
```

Fluxos de controle: if/else

Sintaxe:

```
if(condição) { // se condição atendida  comandoA();  
// então  
    comandoB();  
} // fim se
```

ou

```
if(condição) comandoA();
```

Operador ternário

(condição) ? <se a condição atendida> : <se a condição não foi atendida> ;

Fluxo de controle: switch

Sintaxe:

```
switch (variavel a ser
avaliada) { case 1:
comando1(); comando2();
break;
case 2:
...
break; case X:
...
break; default:
...
}
```

```
int day = 4;
switch (day) {
case 1:
    System.out.println("Monday");
    break;
case 2:
    System.out.println("Tuesday");
    break;
case 3:
    System.out.println("Wednesday");
    break;
case 4:
    System.out.println("Thursday");
    break;
case 5:
    System.out.println("Friday");
    break;
case 6:
    System.out.println("Saturday");
    break;
case 7:
    System.out.println("Sunday");
    break;
}
// Outputs "Thursday" (day 4)
```

JShell

```
bruno@fedora ~$ jshell
| Welcome to JShell -- Version 11.0.9
| For an introduction type: /help intro

jshell> int x = 1
x ==> 1

jshell> int y = 1
y ==> 1

jshell> x+y
$3 ==> 2

jshell> /exit
| Goodbye
bruno@fedora ~$
```

Obrigado, vamos juntos nessa jornada
de Transformação Digital.

Meta

Digital. Simple. Human.

