Apresentação da disciplina

Introdução ao JAVA

Apresentação da disciplina

Área científica: Ciência e Tecnologia da Programação

Cursos: Licenciatura em Engenharia Informática

Escolaridade semanal: 2 horas de aulas teórico-práticas; 2 horas de aulas práticas

Créditos ECTS: 6

Código: 40436

O número de créditos ECTS, atribuído a uma disciplina, **não** indica quantas horas de aulas vão ter. Em vez disso, indica o número de horas expetável que **devem** estudar para esta disciplina.

1 ECTS = 25-30 horas de estudo. 6 ECTS = 150-180 horas de estudo.

O semestre tem ~15 semanas => devem estudar pelo menos 10 horas por semana.

Estas horas incluem: aulas presenciais, leitura de livros, resolução de exercícios, estudo para testes e exames, etc.

Avaliação

Será aplicado o modelo de avaliação "discreta".

Nota final = 0.4 × nota teórica + 0.6 × nota pratica

A nota final resultará de momentos de avaliação seguintes:

- Nota teórica:
  - ❖ Dois testes teórico-práticos (1h) realizados em aulas TP
  - ❖ Datas propostas: 23 de março às 16h30 e na última semana de aulas - 20% + 20%
- Nota prática:
  - ❖ Um teste de avaliação prático realizado no dia 25 de Maio às 16h30 - 40%
  - ❖ Qualidade da participação nas aulas - 20%
  - ❖ O peso do exame prático para estudantes trabalhadores que comprovadamente não tenham frequentado 80% das aulas práticas será de 60%.

A aprovação à disciplina implica uma avaliação global superior ou igual a 9.5 valores sendo que em nenhuma das componentes (teórica e prática) a nota correspondente pode ser inferior a 7.0 valores.

Em regime ordinário as aulas TP e P são de frequência obrigatória.

O aluno que faltar a mais de **3** aulas P e 4 TP ficará automaticamente reprovado, não podendo apresentar-se a qualquer exame da disciplina, durante o ano letivo em curso.

Programa

1) Introdução ao JAVA, tipos primitivos, entrada/saída de dados, tipos de dados, operações, instruções
2) Controlo de fluxo (decisão e ciclos) e vetores.
3) Strings, Introdução a POO: classes, objetos
4) Encapsulamento: atributos e métodos; sobreposição de nomes de métodos; construtores e destrutor; atributos e funções estáticas.
5) Herança: classes base e derivadas; herança; redefinição e sobreposição de métodos.
6) Polimorfismo: generalização versus especialização; ligação dinâmica; classes abstratas.
7) Interfaces, programação para a interface.
8) Tipo paramétricos, enumerados
9) Coleções Java: utilização de estruturas de dados e algoritmos, funções lambda.
10) Entrada e saída de dados: streams, decoradores, serialização.
11) Correção e robustez: programação por contrato e exceções.

Bibliografia recomendada

- B. Eckel, Thinking in Java, 4th edition, Prentice-Hall, 2006, http://mindview.net/Books/TIJ4;
- The Java Tutorials, http://docs.oracle.com/javase/tutorial/;
- Java for Python programmers, https://runestone.academy/runestone/static/java4python/index.html

- F.M. Martins, Java 6 e Programação por Objectos, FCA;
- J. Blosh, Effective Java, 2nd edition, Addison-Wesley, 2008.

Paradigmas de programação

As linguagens de programação têm acompanhado o desenvolvimento dos computadores e evoluído à medida da complexidade dos objetivos cometidos à informática. Todas as linguagens de programação baseiam-se em abstrações.

Um paradigma de programação determina a abstração que o programador pode estabelecer sobre a estruturação e execução do programa.

1) Não estruturada
2) Procedimental
3) Modular
4) Abstração de Tipos de Dados (ADT)
5) Orientada por objetos
6) ...

modelam o problema em termos da estrutura do computador

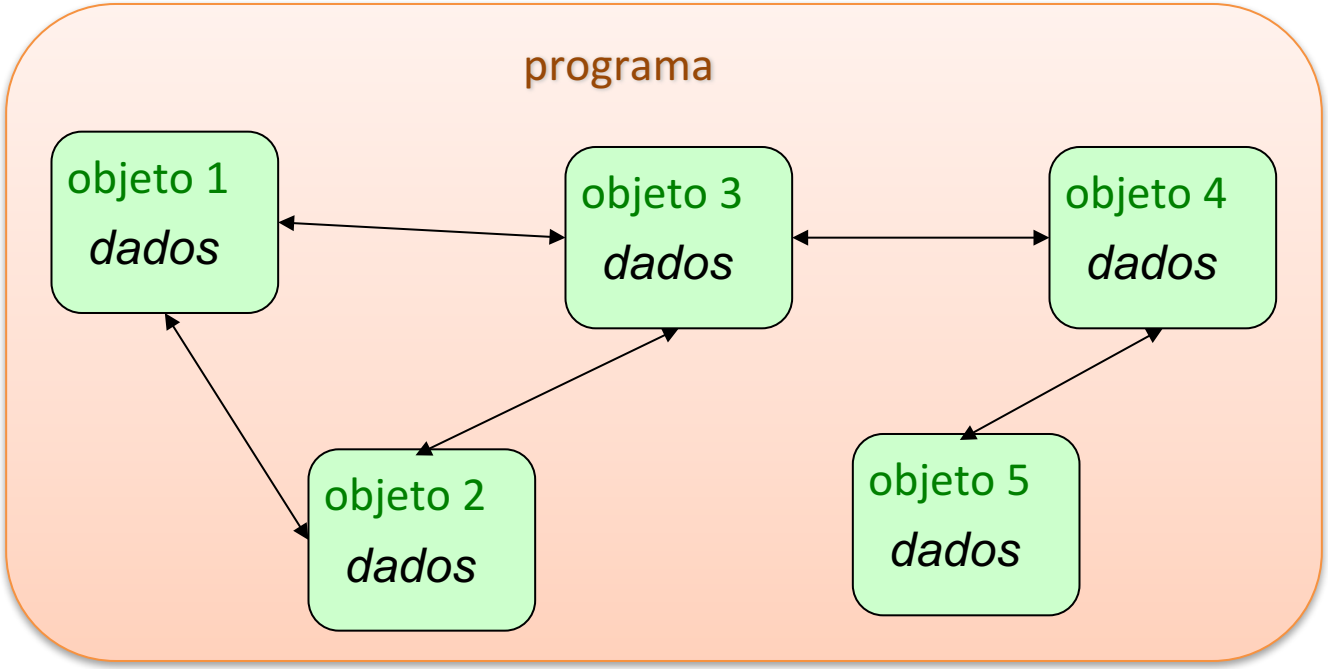
modelam o problema em termos do problema

Algumas linguagens foram desenvolvidas para suportar um paradigma específico, enquanto outras linguagens suportam múltiplos paradigmas.

Programação orientada a objetos

A programação orientada a objetos é um modelo baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Programação orientada por objetos baseia-se num conjunto de objetos que interagem entre si através de mensagens e mantêm os seus próprios dados.



Cada objeto é responsável pela sua própria inicialização e destruição!

Introdução a Java

- Java é uma das linguagens orientadas a objetos (que suporta também outros paradigmas tais como concorrente, genérico e outros).

Java foi desenvolvida na década de 90, na empresa *Sun Microsystems*, e desde aí continuou crescendo e hoje é uma referência no mercado de desenvolvimento de *software*.

Jan 2015	Jan 2014	Change	Programming Language	Ratings	Change
1	1		C	16.703%	-1.24%
2	2		Java	15.528%	-1.00%
3	3		Objective-C	6.953%	-4.14%
4	4		C++	6.705%	-0.86%
5	5		C#	5.045%	-0.80%
6	6		PHP	3.784%	-0.82%
7	9	^	JavaScript	3.274%	+1.70%
8	8		Python	2.613%	+0.24%
9	13	^^	Perl	2.256%	+1.33%
10	17	^^	PL/SQL	2.014%	+1.38%
11	15	^^	MATLAB	1.390%	+0.62%
12	26	^^	ABAP	1.273%	+0.80%
13	27	^^	COBOL	1.267%	+0.81%
14	24	^^	Assembly	1.171%	+0.68%
15	12	v	Ruby	1.130%	+0.07%
16	11	vv	Visual Basic .NET	1.074%	-0.48%
17	-	^^	Visual Basic	1.074%	+1.07%
18	44	^^	R	1.042%	+0.79%
19	10	vv	Transact-SQL	0.874%	-0.68%
20	20		Delphi/Object Pascal	0.837%	+0.24%

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Introdução a Java (cont.)

Pontos fortes de Java:

- *Software* de código aberto, disponível sob os termos da GNU *General Public License*
- Sintaxe similar a C/C++
- Facilidade de internacionalização (suporta nativamente carateres UNICODE)
- Vasto conjunto de bibliotecas
- Facilidades para criação de programas distribuídos e multitarefa
- Desalocação de memória automática por processo de coletor de lixo *(garbage collector)*
- Carga dinâmica de código
- Portabilidade

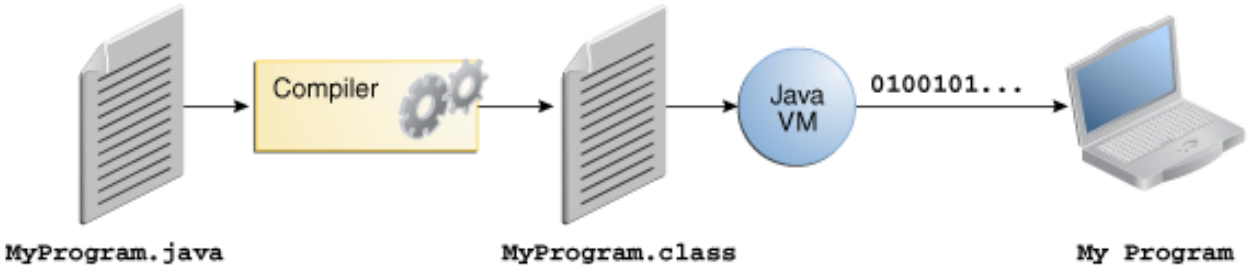
Execução dum programa Java

Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para bytecodes que é executado por uma máquina virtual.

Todo o código fonte é escrito em ficheiros de texto simples que terminam com a extensão .java. Esses ficheiros são compilados em ficheiros .class pelo compilador javac.

Um ficheiro .class não contém código nativo do processador e em vez disso contém bytecodes - a linguagem de máquina virtual.

A aplicação corre com uma instância de Java Virtual Machine - JVM.



Java Virtual Machine

- JVM é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina.

Graças à máquina virtual Java, os programas escritos em Java podem funcionar em qualquer plataforma de hardware e software que possua uma versão da JVM, tornando assim essas aplicações independentes da plataforma onde funcionam.

Logo o mesmo ficheiro .class pode ser executado em máquinas diferentes (que corram Windows, Linux, Mac OS, etc.).

=> Grande portabilidade

Entretanto, já que o ambiente Java é independente do plataforma de hardware, é mais lento comparando com a execução de código nativo.

Introdução à saída de dados

Saída de dados é assegurada com a classe **System** disponível no pacote **java.lang**.

Esta classe inclui, em particular, o dado **out** de tipo **PrintStream** (que modela o *stream* de saída – por omissão consola de saída).

Métodos úteis da classe **PrintStream**:

`println()` – escreve na consóla uma linha vazia

`println(int x)` – escreve na consóla o inteiro `x` e faz "mudança de linha"

`println(String x)` – escreve na consóla a **String** `x` e faz "mudança de linha"

`print(int x)` – escreve na consóla o inteiro `x` e não muda de linha

`print(String x)` – escreve na consóla a String `x` e não muda de linha

Exemplo:

```
System.out.println("O nosso primeiro programa");
```

Introdução à entrada de dados

A classe **System** inclui também o dado **in** de tipo **PrintStream** (que modela o *stream* de entrada – por omissão ligado ao teclado).

Entrada básica de dados é assegurada com a classe **Scanner** disponível no pacote **java.util**, que faz uso de **System.in**.

A classe **Scanner** consegue extrair do *stream* de entrada (teclado) valores de tipos primitivos (*e.g.* **int**) e **String**s. Por omissão, assume-se que valores diferentes são separados uns dos outros por "espaços".

Métodos úteis da classe **Scanner**:

`nextInt()` – extrai da consóla um inteiro e devolve-o

`nextLine()` – extrai da consóla uma linha inteira e devolve a **String** resultante

Exemplo:

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

Estrutura básica dum programa Java

```
//inclusão de pacotes/classes externas
//o pacote java.lang é incluído automaticamente
//em qualquer programa Java
public class Programa
{
   //declaração de dados que compõem a classe
   //declaração e implementação de métodos
   public static void main (String[] args)
   {
       /*ponto de entrada onde começa a execução do programa*/
   }
}
```

Tipos primitivos

- Tipos primitivos servem para criar variáveis *automáticas*, i.e. variáveis que guardam os seus valores *diretamente* e não através de referências.

As variáveis automáticas são criadas pelo compilador na *stack* (pilha - é um tipo de memória mais rápida de alocar e libertar que *heap*).

carateres →

números inteiros

números reais

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	—	—	—	Boolean
char	16 bits	Unicode 0	Unicode $2^{16}-1$	Character
byte	8 bits	-128	+127	Byte
short	16 bits	-2^{15}	$+2^{15}-1$	Short
int	32 bits	-2^{31}	$+2^{31}-1$	Integer
long	64 bits	-2^{63}	$+2^{63}-1$	Long
float	32 bits	IEEE754	IEEE754	Float
double	64 bits	IEEE754	IEEE754	Double
void	—	—	—	Void

B. Eckel, Thinking in Java, 4th edition, Prentice-Hall, 2006

Exemplo com tipos primitivos

```
boolean varBoolean = true; //outro valor possível é false
char varChar = 'A';
byte varByte = 100;
double varDouble = 34.56;

System.out.println(varBoolean);
System.out.println(varChar);
System.out.println(varByte);
System.out.println(varDouble);

Scanner sc = new Scanner(System.in);
float varFloat;
varFloat = sc.nextFloat();
System.out.println(varFloat);
sc.close();
```

Que valor tem a variável varFloat?

Inicialização de variáveis locais

As variáveis locais podem ser inicializadas de modos seguintes:

- na altura da definição:

```
double peso = 50.3;
int dia = 18;
```

- usando uma instrução de atribuição (símbolo '='):

```
double peso;
peso = 50.3;
```

- lendo um valor do teclado ou de outro dispositivo:

```
double km;
km = sc.nextDouble();
```

Estilo recomendado

Para nomear itens no seu programa **não pode** usar nomes reservados em Java (**class**, **new**, **int**, **public**, etc.).

Os identificadores devem começar por uma letra ou por símbolo '_' e só podem conter letras, números e o símbolo '_' (*ex.* `nome`, `idade`, `i`, `j`, `dia_mes`).

Aconselha-se que:

- os nomes das classes comecem sempre por letra maiúscula; se o nome da classe inclui várias palavras, escreva cada uma destas com letra maiúscula: `Pessoa`, `ContaBancaria`;

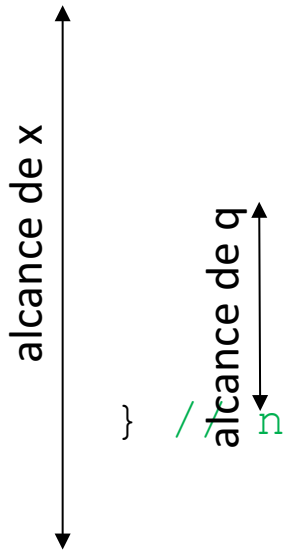
- todas as entidades restantes (variáveis, métodos, etc.) só incluam letras minúsculas exceto se se tratar de palavras composta em que deve escrever cada uma destas palavras com letra maiúscula exceto a primeira: `nomeCompleto`, `capacidadeDeposito`, `ano`, `calcularSaldo`.

Alcance de variáveis

Um alcance é delimitado por um par de parenteses curvos {}.

A variável definida num alcance só existe dentro deste alcance. Quando se sai deste a variável é destruída.

Exemplo:




```
{
        int x = 12;
        // aqui a variável x está disponível
        {
                int q = 96;
                //int x = 2; não se pode declarar mais um x
                // aqui x e q estão disponíveis
        }
        // aqui apenas x está disponível
} // aqui nem x, nem q estão disponíveis
```

Conversão de tipo de variável

Sempre que atribuir um valor com menor capacidade de armazenamento a um valor com maior capacidade de armazenamento, a conversão respetiva será feita automaticamente:

byte -> **short** (ou **char**) -> **int** -> **long** -> **float** -> **double**

A conversão inversa não é admitida e gera um erro de compilação. Entretanto podemos sempre realizar uma conversão explícita através de um operador de conversão:

Exemplo:

```
int a = 3;
double b = 3.3;
double c = a; //conversão automática de int para double
a = (int)b; //variável b é convertida forçosamente para int
```

Conversão de tipo de variável (cont.)

O resultado de conversão dum tipo primitivo de maior capacidade de armazenamento para o tipo com menor capacidade é o valor trancado.

Caso pretenda que o valor seja arredondado, deve usar o método **round()** disponível na classe **java.lang.Math**.

Exemplo:

```
float b = 9.9f; //f foi inserido para guiar o compilador
int c = (int)b; //o valor de c é 9
int d = Math.round(b); //o valor de d é 10
```

Classe Math

A classe **java.lang.Math** contém métodos que permitem realizar operações numéricas básicas, tais como logaritmos, raíz quadrada, funções trigonométricas, etc.

Exemplo:

```
Scanner sc = new Scanner(System.in);
double valor = sc.nextDouble();
System.out.println(Math.sqrt(valor));
System.out.println(Math.pow(valor, 3));
sc.close();
```

Literais

Normalmente, quando se digita um literal, o compilador sabe determinar o seu tipo e interpretá-lo.

Entretanto há situações ambíguas quando o compilador precisa de ser guiado para descobrir o tipo certo.

Para tal inserem-se carateres especiais: l ou L – **long**, f/F – **float**, d/D – **double**, 0x/0Xvalor – valor hexadecimal, 0valor – valor octal.

Exemplo:

```
long a = 23L;
double d = 0.12d;
float f = 0.12f; //obrigatório
```

Operadores

Os operadores levam um, dois ou três argumentos e produzem um valor novo.

Java inclui operadores seguintes:

- aritméticos: *, /, +, -, %, ++, --
- atribuição: =, ...
- relacionais: <, <=, >, >=, ==, !=
- lógicos: !, ||, &&
- manipulação de bits: &, ~, |, ^, >>, <<
- operador de decisão ternário ?:

Operadores aritméticos unários

- Os operadores unários de incremento (++) e decremento (--) só podem ser utilizados com variáveis e atualizam o seu valor de uma unidade.

- Quando colocados antes do operando (`++a` ou `--a`) são pré-incremento e pré-decremento. Neste caso a variável é primeiro alterada antes de ser usada.

- Quando colocados depois do operando (`a++` ou `a--`) são pós-incremento e pós-decremento e neste caso a variável é primeiro usada na expressão onde está inserida e depois atualizada.

Exemplo:

```
int a = 1;
int b = ++a; // a = 2, b = 2
int c = b++; // b = 3, c = 2
```

Operadores de atribuição

Atribuição de primitivas é direta: o conteúdo da variável do lado direito do sinal = é copiado para a variável esquerda.

Exemplo:

```
int a = 1;
int b = a; //ambas a e b têm valor 1
a = 2; // a tem valor 2, b tem valor 1
```

Representação de números em computadores

Em sistemas computacionais quandidades numéricas são codificadas com dígitos binários, ou bits. Um bit pode tomar um de dois valores possíveis: 0 ou 1. O sistema de numeração resultante chama-se sistema binário e a representação particular - complemento para 2.

Nesta representação um número inteiro B pode ser representado de maneira seguinte (onde p é o número de bits e b_i é o valor de bit que esteja na posição i; as posições são numeradas da direita para a esquerda, começando com 0):

$$B = -b_{p-1} * 2^{p-1} + \sum_{i=0}^{p-2} b_i * 2^i$$

Exemplo:

0 1 1 1

bit mais significativo
(posição 3)

bit menos significativo
(posição 0)

Complemento para 2

Note que o peso do bit mais significativo é negativo!

Se o bit mais significativo = 0 => temos um número positivo.

Se o bit mais significativo = 1 => temos um número negativo.

$$B = -b_{p-1} * 2^{p-1} + \sum_{i=0}^{p-2} b_i * 2^i$$

Exemplos:

$110110_2 = -2^5 + 2^4 + 2^2 + 2^1 = -32 + 16 + 4 + 2 = -10_{10}$

$010110_2 = 2^4 + 2^2 + 2^1 = 16 + 4 + 2 = +22_{10}$

$1111_2 = -2^3 + 2^2 + 2^1 + 2^0 = -8 + 4 + 2 + 1 = -1_{10}$

Outros sistemas de numeração

Para além do sistema binário, são usados frequentemente sistemas base 8 (octal) e base 16 (hexadecimal) porque permitem representar bits duma maneira mais compata.

base	alfabeto
2	0,1
8	0, 1,…, 7
10	0, 1,…, 9
16	0, 1,…, 9, A, B, C, D, E, F

Correspondência entre sistemas base 2 e 8

Cada dígito octal corresponde diretamente a 3 bits:

binário	octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Exemplo:

$111\ 010\ 110_2 = 7\ 2\ 6_8$

Correspondência entre sistemas base 2 e 16

Cada dígito hexadecimal corresponde diretamente a 4 bits:

binário	hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Exemplo:

$1000\ 1111\ 1010\ 0101_2 = 8\ F\ A\ 5_{16}$

Revisão de literais

Carateres especiais permitem especificar valores numéricos em bases 8 ou 16: 0x/0Xvalor – valor hexadecimal, 0valor – valor octal.

Exemplo:

```
int i = 0x2f; // qual é o valor de i?
byte j = 033; // qual é o valor de j?
byte k = (byte)0xff; // qual é o valor de k?
```

Valores min/max para tipos primitivos

Cada uma das classes **Character**, **Byte**, **Short**, **Integer**, **Long**, **Float**, **Double** tem membros MAX_VALUE e MIN_VALUE que guardam valores máximo e mínimo para o tipo respetivo.

byte	8 bits
short	16 bits
int	32 bits
long	64 bits

Explique

Exemplo:

```
System.out.println(Byte.MAX_VALUE); // 127
System.out.println(Byte.MIN_VALUE); //-128

System.out.println(Integer.MAX_VALUE); // 2147483647
System.out.println(Integer.MIN_VALUE); //-2147483648

System.out.println(Long.MAX_VALUE); // 9223372036854775807
System.out.println(Long.MIN_VALUE); //-9223372036854775808

System.out.println(Double.MAX_VALUE);//1.7976931348623157E308
System.out.println(Double.MIN_VALUE);//4.9E-324 = 4.9×10^-324
```

Operadores de manipulação de bits

Java inclui operadores & (*AND*), ~ (*NOT*), | (*OR*), ^ *(XOR)* que permitem manipular bits individuais em valores inteiros (operadores bitwise).

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

x	~x
0	1
1	0

Exemplo:

```
int i = -1; int j = 2;
System.out.print("AND: ");
System.out.println(i & j);

System.out.print("OR: ");
System.out.println(i | j);

System.out.print("XOR: ");
System.out.println(i ^ j);

System.out.print("NOT: ");
System.out.println(~i);
```

```
AND: 2
OR: -1
XOR: -3
NOT: 0
```

Operadores de manipulação de bits (cont.)

Java inclui também operadores de deslocamento:

- `n << x` (deslocamento para a esquerda) – número `n` é deslocado `x` posições para a esquerda, `x` bits menos significativos são preenchidos com 0.

- `n >> x` (deslocamento para a direita com sinal) – número `n` é deslocado `x` posições para a direita, `x` bits mais significativos são preenchidos com o valor do bit mais significativo original de `n`.

- `n >>> x` (deslocamento para a direita sem sinal) – número `n` é deslocado `x` posições para a direita, `x` bits mais significativos são preenchidos com 0.

Exemplo:

```
int i = -1; //inteiros ocupam 32 bits
System.out.print("<< 2: ");
System.out.println(i << 2);

System.out.print(">> 2: ");
System.out.println(i >> 2);

System.out.print(">>> 30: ");
System.out.println(i >>> 30);
```

```
<< 2: -4
>> 2: -1
>>> 30: 3
```