Introdução à Linguagem Java

Luís Lopes

DCC-FCUP

Estruturas de Dados

A Linguagem Java

- desenvolvida pela Sun Microsystems
- início em 1991 (com o nome de Oak)
- disponibilizada em 1995 com o nome Java
- fundamental no desenvolvimento da Web
- passou para Oracle em 2009
- orientada a objectos (Object Oriented/OO)

Vantagens

- portabilidade
  - compilada para bytecode
  - executado por uma máquina virtual (**JVM**)
  - basta ter a JVM instalada para executar qualquer programa Java
- segurança e robustez
  - verificação de tipos estática
  - gestão de memória automática
  - excepções para tratar erros de execução
- ferramentas de desenvolvimento (**JDK**)
  - compilador: `javac`
  - interpretador de bytecode: `java`
  - ferramentas (`jar`, `javadoc`, ...)
  - Aplication Programming Interfaces (APIs)

Modelo de Programação

- baseado nos conceitos de classe e objecto
- classes são estáticas (criadas pelo compilador) e correspondem a especificações de objectos
- definem a informação mantida pelos objectos (atributos) e como esta pode ser manipulada (métodos)
- objectos são criados dinamicamente (durante a execução de um programa) e são instâncias de classes
- a criação de objectos é feita usando a palavra chave `new`, seguida do nome da classe correspondente e eventuais argumentos
- são descartados automaticamente por garbage collection

Modelo de Programação

- programa é composto por um conjunto classes
- cada classe `X` deve estar num ficheiro `X.java`
- (apenas) uma das classes deve ter o método `main()`

```
public class HelloWorld {
   public static void main(String[] args) {
     System.out.println("Hello World!");
   }
}
```

- para compilar e executar este programa na linha de comando:

```
$ javac HelloWorld.java
$ java HelloWorld
```

Exemplo Vector: `Vector.java`

```
// a 3d vector
public class Vector {
  // coordinates
  double x;
  double y;
  double z;

  // constructor
  public Vector(double x0, double y0, double z0 ) {
    x = x0;
    y = y0;
    z = z0;
  }
}
```

Exemplo Vector: LibVector.java

```
public class LibVector {
  public static Vector minus(Vector u) {
    return new Vector( -u.x, -u.y, -u.z );
  }
  public static Vector add(Vector u, Vector v) {
    return new Vector( u.x + v.x, u.y + v.y, u.z + v.z );
  }
  public static Vector sub(Vector u, Vector v) {
    return add( u, minus(v) );
  }
  public static Vector scale(double s, Vector u) {
    return new Vector( s * u.x, s * u.y, s * u.z );
  }
  public static double scalarProduct(Vector u, Vector v) {
    return u.x * v.x + u.y * v.y + u.z * v.z;
  }
  public static double modulus(Vector u) {
    return java.lang.Math.sqrt(scalarProd(u,u));
  }
  ...
}
```

Exemplo Vector: TestVector.java

```
public class TestVector {
  public static void main(String[] args) {
    Vector u1 = new Vector (  -2.1,  5.6, 3.3 );
    Vector u2 = new Vector (   9.7,  1.1, 2.7 );
    Vector u3 = new Vector ( -12.1, -7.6, 5.1 );
    Vector u4 = LibVector.add( u1, u2 );
    Vector u5 = LibVector.sub( u1, u2 );
    Vector u6 = LibVector.minus( u3 );
    u6 = LibVector.scale( 2.6, u6 );

    System.out.println( LibVector.toString(u4) + "\n"
                        LibVector.toString(u5) + "\n"
                        LibVector.toString(u6) + "\n" );
}
```

Modelo de Programação

- para compilar e executar este programa na linha de comando:

```
$ javac Vector.java LibVector.java TestVector.java
$ java  TestVector
```

- ou simplesmente

```
$ javac TestVector.java
$ java  TestVector
```

Algumas regras de etiqueta

- nomes de classes são substantivos e começam com maiúscula (e.g., `Vector`)
- se o nome tiver várias palavras, todas começam com maiúscula (e.g., `VectorField`)
- nomes de atributos são substantivos e começam com minúscula (e.g., `radius`)
- se o nome tiver várias palavras, apenas a primeira começa com minúscula (e.g., `positionVector`)
- nomes de métodos são verbos começam com minúscula (e.g., `scale`)
- se o nome tiver várias palavras, apenas a primeira começa com minúscula (e.g., `scalarProduct`)
- sempre que possível devem ser utilizados nomes por extenso, excepto em casos onde haja convenção (e.g., `x`, `y`, `z` para as coordenadas, `lib` para biblioteca)

Algumas regras de etiqueta

- há excepções às regras de etiqueta, e.g.:
  - `sqrt`
  - `minus`,
  - `scalarProduct`
  - `modulus`
- usar o bom senso
- o fundamental é que o nome/verbo transmita com clareza a semântica pretendida

Tipos de Dados

- tipos básicos, manipulados directamente na máquina virtual
  - inteiros (int, long)
  - vírgula-flutuante (float, double)
  - caracteres (char)
  - bytes (byte)

```
int     i = 2;
float   x = 3.192;
boolean b = true;
```

- os restantes tipos são classes em bibliotecas da linguagem ou definidas pelos programadores
  - a partir das classes são criados e inicializados objectos (**new**) que residem na memória
  - quando deixam de ser utilizados a libertação do espaço que ocupam na memória é automática (**garbage collection**)

```
Vector v = new Vector( 0.0, 0.0, -1.0 );
```

Tipos de dados básicos

- **Inteiros**

`short`	16 bits	$[-2^{15} : 2^{15} - 1]$
`int`	32 bits	$[-2^{31} : 2^{31} - 1]$
`long`	64 bits	$[-2^{63} : 2^{63} - 1]$

- **Vírgula flutuante (IEEE 754)**

`float`	32 bits	$[-3.4029E + 38 : +3.4029E + 38]$
`double`	64 bits	$[-1.79769E + 308 : +1.79769E + 308]$

- **Outros**

`boolean`	8 bits	valor booleano (`true` ou `false`)
`char`	16 bits	ISO Unicode char set
`byte`	8 bits	sequência de bits

Exemplo

```
public class BasicTypes {
   public static void main(String[] args) {
      boolean flag = true;
      char       ch = 'A';
      byte        b = 12;
      short       s = 24;
      int         i = 257;
      long        l = 890L;
      float       f = 3.1415f;
      double      d = 2.1828d;
      System.out.println("flag = " + flag);
      System.out.println("ch = "   + ch);
      System.out.println("b = "    + b);
      System.out.println("s = "    + s);
      System.out.println("i = "    + i);
      System.out.println("l = "    + l);
      System.out.println("f = "    + f);
      System.out.println("d = "    + d);
   }
}
```

Operadores

Aritméticos:

+	adição	-	subtracção
*	multiplicação	/	divisão
%	módulo	++	incremento
--	decremento		

Lógicos:

!	NOT lógico
&&	AND lógico
||	OR lógico

Bits:

~	NOT binário	&	AND binário
|	OR binário	^	XOR binário
<<	shift binário esquerda	>>	shift binário direita

Relacionais ou de comparação:

==	igualdade	!=	diferente
<	menor que	<=	menor ou igual que
>	maior que	>=	maior ou igual que

Divisão-inteira e Resto-da-divisão

- Quando dividimos inteiros, o quociente é inteiro.
  - `18/4` dá `4` e não `4.5`
  - `y = x/2;` – a divisão será inteira se `x` for um `int`.
  - se `x` for `float` e `y` for `int`, a divisão será em virgula flutuante, mas depois o resultado é truncado para inteiro.
  - dividir por 0 provoca um erro durante a execução
- O operador % calcula o resto da divisão inteira.
  - `18 % 4` dá `2`
  - `59 % 5` dá `4`
- Aplicações do operador %:
  - obter o último dígito de um número: `12345 % 10` é `5`
  - obter os últimos três dígitos: `734528 % 1000` é `528`
  - verificar se um número é par: `9 % 2` é `1` e `16 % 2` é `0`

Regras de precedência entre operadores

Precedência: ordem de avaliação dos operadores.

- a regra geral é avaliação da esquerda para a direita.
  - `5-2-7` é igual a `(5-2)-7` que é `-4`
- mas os operadores `* / %` têm maior precedência que `+ -`
  - `5+2*3` é `11`
  - `5+10/2*3` é `5+5*3` que dá `20`
- os parentesis forçam a ordem de avaliação

Operadores de incremento/decremento

- operadores incremento/decremento:
  - `variavel++;` $\Longleftrightarrow$ `variavel = variavel + 1;`
  - `variavel--;` $\Longleftrightarrow$ `variavel = variavel - 1;`
  - `++variavel;` $\Longleftrightarrow$ `variavel = variavel + 1;`
  - `--variavel;` $\Longleftrightarrow$ `variavel = variavel - 1;`
- Cuidado com o uso destes operadores sobre variáveis no meio de expressões. Consideremos o seguinte:
  - `int x= 5, y= 3, z;`
  - `x++; // incrementa x, i.e. x= x+1`
  - `--y; // decrementa y, i.e. y= y-1`
  - `z= x-- * ++y; // com que valores ficam x, y e z ?`
- `x--` diz-nos para usar primeiro o valor de `x` e depois decrementar em uma unidade;
- `++y` diz-nos para primeiro incrementar `y` uma unidade e depois usar o seu valor.

Operadores de imodificação-e-atribuição

- operadores modifica-e-atribui:
  - `variavel += valor;` $\Longleftrightarrow$ `variavel= variavel + valor;`
  - `variavel -= valor;` $\Longleftrightarrow$ `variavel= variavel - valor;`
  - `variavel *= valor;` $\Longleftrightarrow$ `variavel= variavel * valor;`
  - `variavel /= valor;` $\Longleftrightarrow$ `variavel= variavel / valor;`
- Exemplos:
  - `x -= 1;` $\Longleftrightarrow$ `x--;` $\Longleftrightarrow$ `x= x-1;`
  - `y /= 2;` $\Longleftrightarrow$ `y = y/2;`
  - `x *= y+2;` $\Longleftrightarrow$ `x= x*(y+2);`

if-else

```
class IfElseTest {
  public static void main(String[] args) {
    int testscore = 76;
    char grade;

    if (testscore >= 90) {
      grade = 'A';
    } else if (testscore >= 80) {
      grade = 'B';
    } else if (testscore >= 70) {
      grade = 'C';
    } else if (testscore >= 60) {
      grade = 'D';
    } else {
      grade = 'F';
    }
    System.out.println("Grade = " + grade);
  }
}
```

switch-case

```
public class SwitchTest {
  public static void main(String[] args) {
    int month = 2;
    String monthString;
    switch (month) {
    case 1:  monthString = "January";
             break;
    case 2:  monthString = "February";
             break;
    case 3:  monthString = "March";
             break;
    ...
    case 12: monthString = "December";
             break;
    default: monthString = "Invalid month";
             break;
    }
    System.out.println(monthString);
  }
}
```

while

```
class TestWhile {
  public static void main(String[] args) {
    String what = new String(isPrime(19) ? " " : "not ");
    System.out.println("19 is" + what + "prime");
  }


  public static boolean isPrime(int n) {
    int divisor = 2;
    while (divisor < n/2) {
      if ( (n % divisor) == 0 )
        return false;
        divisor++;
    }
    return true;
  }
}
```

for

```
class TestFor {
  public static void main(String[] args) {
    String what = new String(isPrime(19) ? " " : "not ");
    System.out.println("19 is" + what + "prime");
  }

  public static boolean isPrime(int n) {
    for(int divisor = 2; divisor < n/2; divisor++)
      if ( (n % divisor) == 0 )
        return false;
    return true;
  }
}
```

do-while

```
class TestDoWhile {
  public static void main(String[] args) {
    String what = new String(isPrime(19) ? " " : "not ");
    System.out.println("19 is" + what + "prime");
  }

  public static boolean isPrime(int n) {
      int divisor = 2;
      do {
         if ( (n % divisor) == 0 )
            return false;
         divisor++;
      } while (divisor < n/2);
      return true;
   }
```

Strings

- uma string é uma sequência de caracteres
- não é um tipo básico
- são instâncias da classe `String`
- não se comportam como arrays, são imutáveis
- criação/inicialização invocando métodos construtores:
  - `String c1 = new String("Trimeresurus");`
- criação simplificada:
  - `String c2 = "albolabris";`
- operador de concatenação:
  - `String c3 = c1 + " " + c2 + " insularis";`
- o valor que fica em c3 é:
  - `"Trimeresurus albolabris insularis";`

Strings

- e.g., `String s = "jararaca";`

índice	0	1	2	3	4	5	6	7
caracter	'j'	'a'	'r'	'a'	'r'	'a'	'c'	'a'

- podemos ler caracteres individuais mas não alterá-los

```
String s = "jararaca";
System.out.println(s.charAt(6)); // 'c'
```

```
for(int i = 0; i < s.length(); i++)
  System.out.print(s.charAt(i));
```

- podemos converter strings em arrays de caracteres e vice-versa

```
String s1 = "jararaca";
char[] cs = s1.toCharArray();
String s2 = Arrays.toString(cs);
```

Strings

```
class UseStrings {
   public static void main (String[] args) {
      char[] cs = {'a','e','i','o','u'};
      String s0 = new String("Ola");
      String s1 = new String();
      String s2 = new String(s0);
      String s3 = new String(cs);
      String s4 = new String(cs,2,2); // "io"
      System.out.println(
            "s1 = " + s1 + "\n" + "s2 = " + s2 + "\n" +
            "s3 = " + s3 + "\n" + "s4 = " + s4);
   }
}
```

Strings

- comparação de duas strings
  - usar o método `equals()` e não o habitual operador "==".
- dadas duas strings `s1` e `s2`
  - `s1 == s2` apenas compara as referências dos dois objectos
  - `s1.equals(s2)` compara as strings, caracter a caracter
- O método `s1.compareTo(s2)` compara as strings `s1` e `s2` por ordem lexicográfica
  - `< 0`, se `s1` precede `s2`
  - `== 0`, se forem iguais
  - `> 0`, se `s1` vem a seguir a `s2`

Strings

```
String s1= "Hello";
String s2= "Hello";

// true
System.out.print(s1 + " equals " + s2 + ":" + s1.equals(s2));
// false
System.out.print(s1 + "   ==   " + s2 + ":" + (s1 == s2));

String s3= "Good-bye";
String s4= "HELLO";

s1.equals(s3)            // false
s1.equals(s4)            // false
s1.equalsIgnoreCase(s4) // true
```

Arrays

Os **arrays** são objectos que guardam, em posições contíguas de memória, um conjunto de valores de um mesmo tipo (primitivo ou não). Os valores são localizados por um índice inteiro ≥ 0.

- criar variável para guardar a referência para um array de um dado tipo (não reserva espaço em memória!)
  - `int[] grades;`
  - `Vector[] velocities;`
- o operador `new` cria o array com a capacidade indicada em memória
  - `int[] grades = new int[20];`
  - `Vector[] velocities = new Vector[1024];`
  - os elementos do array são acessíveis pelo nome da variável e um índice, e.g., `grades[17]` ou `velocities[221]`

Arrays

- Ao trabalharmos com arrays é necessário ter atenção a possíveis excepções que sejam geradas e indiciadoras de situações de erro, e.g:
- `NullPointerException` – tentar usar o array sem o criar

```
int[] v;
v[0] = 2; // NullPointerException
```

- `ArrayIndexOutOfBounds` – aceder ao array fora dos limites

```
int[] v = new int[4];
v[0] = 2;
v[4] = 5; // ArrayIndexOutOfBoundsException
```

- quando declara um array deve sempre inicializá-lo de imediato
- de resto, este conselho é válido para todas as variáveis
- o atributo `length` de um arrays dá-nos o seu número máximo de elementos

Arrays

- Como imprimimos os elementos de um array?

```
int[] primes = {2,3,5,7,11,13};
System.out.println(primes);
System.out.println(Arrays.toString(primes));
```

output

```
[I@fee4
[2,3,5,7,11,13]
```

- Como comparámos dois arrays?
  - usar um ciclo que compara os elementos, um a um
  - usar o método `Arrays.equals()`

```
int[] a = {1,2,3,4,5};
int[] b = {1,2,3,4,5};
if( Arrays.equals(a,b) )
  System.out.println("same contents");
```

Arrays

- Um método pode ter parâmetros na chamada que são arrays e pode dar como resultado um array.
- e.g., um método que recebe 2 arrays (de inteiros) de igual tamanho e retorna um array com a soma dos dois

```
int[] addArrays (int[] u, int[] v) {
  int res[] = new int[u.length];
  for ( int i = 0 ; i < u.length ; i++ )
    res[i] = u[i] + v[i];
  return res;
}
```

Arrays Multimensionais

- os arrays podem ser multi-dimensionais

```
int[][]    v = new int[4][4];     // bi-dimensional 4x4

int[][][]  u = new int[5][3][7]; // tri-dimensional 5x3x7
```

- e.g., multiplicação de uma matriz $a[N][M]$ por um vector $u[M]$ para dar $v[N]$:

  i.e.

$$v_i = \sum_{j=0}^{M-1} a_{ij} \times u_j \quad (0 \leq i < N)$$

Multiplicação de matriz por array

```
class MatrixVectorProductTest {
  public static void main(String[] args) {
    int[][]  a = {{1,2,3},{4,5,6}};
    int[]    u = {1,2,3};
    int[]    v = matrixVectorMult(a,u);

    for( int i = 0 ; i < v.length ; i++ )
      System.out.print(v[i] + " ");
    System.out.println();
  }
  static int[] matrixVectorMult(int[][] a, int[] u) {
    int[] v = new int[a.length];
    for( int i = 0 ; i < v.length ; i++ ) {
      v[i] = 0;
      for( int j = 0 ; j < u.length ; j++ )
        v[i] += a[i][j] * u[j];
    }
  }
}
```

Input/Output

as classes mais importantes que lidam com I/O no Java estão definidas nas "packages" `java.io` e `java.lang`

- a leitura e escrita faz-se através de canais (*streams*) que podem representar um qualquer periférico físico.
- a classe **System**, definida em `java.lang`, inclui muitas definições de sistema, nomeadamente 3 canais: **in**, **out**, e **err**.
  - `InputStream System.in` – objecto que representa o standard input stream (por defeito o teclado);
  - `PrintStream System.out` – objecto que representa o standard output stream (por defeito a consola);
  - `PrintStream System.err` – objecto que representa o standard error stream (consola).

A classe Scanner

- simplifica muito o processamento do input vindo de:
  - teclado (tipo `InputStream`)

```
Scanner stdIn= new Scanner(System.in);
```

  - através de uma `String`

```
String line = new String("Hello World!");
Scanner strIn= new Scanner(line);
```

  - ou de um ficheiro

```
File file= new File(fileName);
Scanner fileIn= new Scanner(file);
```

- divide input em strings separadas por *delimitadores*.
- `useDelimiter(expr)` permite especificar delimitadores, e.g.,

```
scanner.useDelimiter("\r\n")
```

A Classe Scanner

Para se poder usar a classe Scanner é necessário declarar no programa:

- **import** `java.util.Scanner;`

Alguns métodos relevantes desta classe:

`hasNext()`	**true** se e só se existir mais uma palavra no input
`next()`	retorna a próxima palavra (String) do input
`hasNextLine()`	**true** se e só se o input tiver mais uma linha de texto
`nextLine()`	retorna a próxima linha de texto do input
`hasNextType()`	**true** se e só se a próxima palavra for do tipo `Type` onde `Type` pode ser qualquer tipo básico: **int**, **float**, ...
`nextType()`	retorna a próxima palavra convertida para o tipo básico definido por `Type`.

Scanner: leitura a partir de uma String

```
import java.util.Scanner;

class TestScannerFromString {
  public static void main (String[] args) {
    Scanner strIn = new Scanner("1 - 2 - 3 - 4 - 5");
    strIn.useDelimiter(" - ");
    while ( strIn.hasNextInt() ) {
      int n = strIn.nextInt();
      System.out.println(n);
    }
  }
}
```

Scanner: leitura a partir do teclado

```
import java.util.Scanner;

class TestScannerFromKeyboard {
  public static void main (String[] args) {
    Scanner stdIn = new Scanner(System.in);
    System.out.println("Number of persons: ");

    int n = stdIn.nextInt();
    String[] names = new String[n];
    int[]    ages  = new int[n];

    for( int i = 0; i < n ; i++ ) {
      System.out.println("input name[space]age: ");
      names[i] = stdIn.next();
      ages[i]  = stdIn.nextInt();
    }

    for( int i = 0; i < n ; i++ )
      System.out.println("name:"+names[i]+" age: "+ages[i]);
  }
}
```

Scanner: leitura a partir de um ficheiro

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

class TestScannerFromFile {
  public static void main (String args[]) {
    try {
      File file = new File("./example.txt");
      Scanner fileIn = new Scanner(file);
      while( fileIn.hasNextLine() )
       System.out.println(fileIn.nextLine());
    }
    catch (IOException e) {
      System.out.println("File not found");
    }
  }
}
```