NOTAS INTRODUTÓRIAS SOBRE PROGRAMAÇÃO EM JAVA (PARTE I)

João M. P. Cardoso

Faculdade de Ciências e Tecnologia

Universidade do Algarve

Campus de Gambelas

8000 – 117 Faro, Portugal

Email: jmcardo@ualg.pt

1 Introdução

Este documento serve de introdução a alguns aspectos relacionados com o desenvolvimento de programas em Java. Leitores interessados em aprender mais sobre a linguagem e tecnologia Java podem consultar o tutor on-line [1] e sugere-se que não dispensem a leitura de um bom livro sobre Java [2]. Os leitores já familiarizados com a linguagem, podem utilizar o documento disponibilizado gratuitamente sobre a especificação da linguagem Java [3].

A linguagem de programação Java é uma linguagem orientada por objectos e cuja sintaxe tem muitas semelhanças com a sintaxe das linguagens C/C++. É uma linguagem que permite ciclos de desenvolvimento normalmente mais rápidos do que aqueles obtidos com outras linguagens de programação. A linguagem Java é em parte responsável pela proliferação do conceito de *write once, run everywhere*, *i.e.*, uma aplicação Java pode ser desenvolvida independentemente da plataforma computacional alvo para depois poder ser executada em qualquer plataforma computacional. Além de outras facilidades, a tecnologia Java inclui um conjunto de APIs[1], de importância primordial para o desenvolvimento de aplicações complexas.

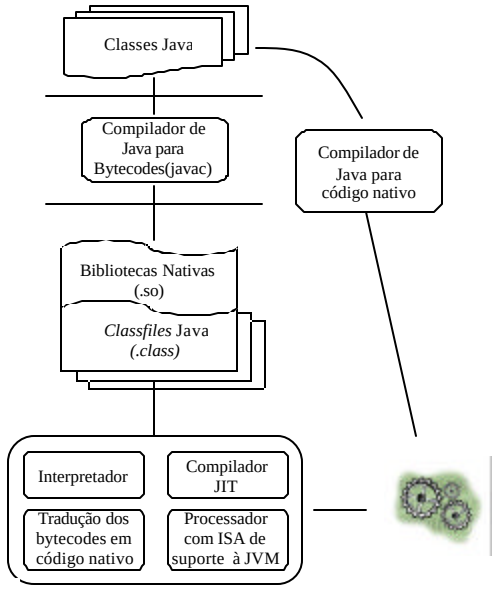

Figura 1. De um programa em Java até à sua execução.

Um programa em linguagem Java é forçosamente constituído por um conjunto de uma ou mais classes. Cada classe é compilada para uma máquina virtual designada por JVM (*Java Virtual Machine*). Esta compilação fornece para cada classe um ficheiro binário chamado de *classfile* e que inclui os *bytecodes* Java (representam instruções da JVM e outra informação). Estes bytecodes podem depois

[1] *Aplication Programming Interface*: São pacotes (*packages*) software que podem ser utilizados pelo programador e que podem ser vistos como bibliotecas de classes.

ser executados por um interpretador, por um compilador JIT [2] (*Just-In Time*), ou por um processador que implemente o conjunto de instruções da JVM[3]. A Figura 1 apresenta o fluxo de processamento desde o programa fonte em Java até à sua execução.

A SUN disponibiliza, entre outras ferramentas, um compilador, interpretador e compilador JIT que integram o Java 2 SDK (*standard development kit*) [5].

Tem havido alguns esforços para compilar programas Java para código nativo. Estas abordagens têm de forçosamente incluir o ambiente de *runtime* no código final. Um exemplo é o GCJ disponibilizado pela GNU [4]. Alerta-se para a possibilidade destes compiladores não suportarem todas as potencialidades do Java.

2 Primeiro Exemplo

Uma aplicação Java (exceptua-se aqui o caso das *applets*) tem de ter um método `main` definido numa classe (neste caso designamos a classe por `myClass`):

```
class myClass {
     public static void main(String args[]) {
              ...
       }
}
```

No caso de se pretender escrever "Hello World" no ecrã tem de se utilizar uma chamada a uma função que escreve no ecrã[4].

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

Supondo que esta classe é guardada no ficheiro `myClass.java` para se compilar deve fazer-se:

```
$ javac myClass.java ?
```

e obtém-se o ficheiro `myClass.class` (ficheiro com os *bytecodes*). Para executar o programa deve fazer-se:

```
$ java myClass ?
```

O interpretador ou compilador JIT procura pelo método `main` na referida classe e executa-o[5].

3 Ler de um ficheiro de texto

O método `main` da classe seguinte abre um ficheiro para leitura e lê caracter a caracter até ao fim do ficheiro. Cada caracter lido é escrito no ecrã.

```
import java.io.*;
```

[2] É um processo de compilação em que a geração do código máquina é realizada durante a execução do próprio programa.

[3] Existem alguns processadores que implementam um subconjunto das instruções JVM e que suportam as restantes por software.

[4] Este enunciado de código Java indica a chamada do método `println` existente na classe `out`, que é um dos atributos da classe `System` do pacote `java.lang`.

[5] Com o compilador GCJ: gcj -c -g -O myClassJava.java; gcj --main=myClassJava -o myClassJava myClassJava.o. gcj –C myClassJava.java (compila para *bytecodes*, tal como o javac).

```
// import java.io.FileReader;

public class readFile {

  public static void main(String args[]) {

    try {
      // create a new instance of the class FileReader
      // which opens the file with the name "input.txt"
      FileReader inFile = new FileReader("input.txt");

      // Read each char of the file
      int c = inFile.read();
      while (c != -1 ) {
         System.out.println((char) c);
         c = inFile.read();
      }

      // close the file
      inFile.close();
    } catch(FileNotFoundException e) {
      // if the named file does not exist, is a directory rather than a regular file,
      // or for some other reason cannot be opened for reading.
      // if so then exit and report an error message
      System.err.println("File not found! "+e);
    } catch(IOException e) {
      System.err.println("Exception reading/writing/closing the file! "+e);
    }
  }
}
```

No caso de querermos que o utilizador escreva o nome do ficheiro de texto a ser lido da linha de comandos, aquando da invocação do programa, poderíamos utilizar a versão seguinte:

```
import java.io.*;
// import java.io.FileReader;

public class readFileFromName {

    public static void main(String args[]) throws IOException  {

        if(args.length != 1) {
            System.out.println("No name for the input file!");
            return;
        }

        String fileName = args[0];

        try {
            // create a new instance of the class FileReader
            // which opens the file with the name given by the variable FileName
            FileReader inFile = new FileReader(fileName);

            // Read each char from the file
            int c = inFile.read();
            while (c != -1) {
```

```
                System.out.println((char) c);
                c = inFile.read();
            }

            // close the file
            inFile.close();

        } catch(FileNotFoundException e) {
            System.err.println("File not found! "+e);
        } catch(IOException e) {
            System.err.println("Exception closing file! "+e);
        }
    }
}
```

Poder-se-ia também ler um bloco de caracteres de cada vez. Para tal o ciclo de leitura do ficheiro poderá ser:

```
// Read each time a block of N chars
char[] cbuf = new char[N];
int c = 0 ;
while (c != -1) {
    c = inFile.read(cbuf);
    System.out.println(cbuf);
}
```

4 Escrever num ficheiro de texto

O exemplo a seguir lê, de um ficheiro origem, caracter a caracter e à medida que vai lendo cada caracter escreve-o no ficheiro destino. Ambos os ficheiros são especificados na linha de comandos aquando da invocação do programa.

```
import java.io.*;
// import java.io.FileReader;
// import java.io.FileWriter;

public class writeFile {

    public static void main(String args[]) throws IOException  {

        if(args.length != 2) {
            System.out.println("usage: java writeFile <input file> <output file>");
            return;
        }

        String inputName = args[0];
        String outputName = args[1];

        try {
            FileReader inFile = new FileReader(inputName);

            FileWriter outFile = new FileWriter(outputName);

            // Read each char of the file
            int c = inFile.read();
            while(c != -1) {
                outFile.write(c);
                c = inFile.read();
```

```
            }

            // close each file
            inFile.close();
            outFile.close();

        } catch(FileNotFoundException e) {
            System.err.println("File not found! "+e);
        } catch(IOException e) {
            System.err.println("Exception with file! "+e);
        }
    }
}
```

5 Como migrar do C para o Java

A sintaxe da linguagem Java, como foi anteriormente referido, tem muitas similitudes com a linguagem C. Este facto facilita a migração da linguagem C para Java. Por exemplo, as construções de ciclos (`do while`, `while`, e `for`) são idênticas. A partir daqui introduziremos algumas das diferenças.

Em Java as declarações de variáveis podem ficar situadas em pontos do programa onde são necessárias e não apenas em determinadas zonas de declaração de variáveis. Em Java não é permitido que uma variável possa ser utilizada (numa expressão, por exemplo) sem antes lhe ter sido atribuído um valor (existem muitos *bugs* de código que acontecem, precisamente, quando programamos com linguagens que não impõem esta regra).

5.1 Noção de Classe e Objecto

De uma forma simplista, uma classe pode ser visto como uma estrutura do tipo registo (`struct` em C) que para além dos campos pode também incluir procedimentos que podem ou não manipular esses campos. Na terminologia de classes, os campos são designados de atributos e esses procedimentos de métodos da classe. Quando se declara uma variável como sendo do tipo da classe especificada estamos a criar um objecto (instância ou concretização de uma classe especifica). Para criar um objecto é utilizada a palavra `new` e chamado o construtor da classe (o construtor é um método da classe que pode ser utilizado para, por exemplo, atribuir um valor inicial aos atributos). Esse objecto, sendo uma instância ou concretização da classe, retém os atributos e os métodos desta. Os atributos e os métodos podem ser acedidos externamente (quando na declaração dos mesmos é utilizada a palavra reservada **`public`**) ou simplesmente internamente. Para aceder a atributos ou métodos é utilizado o nome do objecto seguido de '.' e do nome do atributo ou do método.

Os exemplos seguintes ilustram o que foi referido. Na primeira coluna á apresentado um exemplo, em que é definida uma classe com dois atributos que podem ser acedidos externamente de modo directo. Na coluna do meio é apresentada uma classe com os mesmos atributos, mas agora declarados como não podendo ser acedidos externamente de modo directo. São incluídos dois métodos, declarados para que possam ser acedidos externamente, que atribuem valores a cada um dos dois atributos. Na última coluna é apresentado o mesmo exemplo, desta vez incluindo a atribuição inicial aos atributos da classe na construção do objecto (o método com o mesmo nome da classe é designado por construtor e pode ser invocado aquando da criação do objecto). Por omissão, existe sempre um construtor sem argumentos. Outras das propriedades do Java é o facto de permitir que na mesma classe existam métodos com o mesmo nome, mas com declarações de argumentos de tipos ou em número distintos (esta propriedade designa-se por polimorfismo). Nos exemplos apresentados

na segunda e na terceira coluna não existe forma de utilizar o valor dos atributos (por exemplo, numa determinada expressão) em código externo à classe. Quais os métodos que teríamos de acrescentar à classe para que esses atributos pudessem ser utilizados externamente?

```
Class Reg {
    Public int line;
    Public int errorID;
    …
}

// por ex., no main
Reg ex = new Reg();
ex.line = 0;
ex.errorID = 0;
```

```
Class Reg {
   int line;
    int errorID;
   public void setLine(int a) {
      This.line = a;
   }
   public void setError(int a) {
      This.errorID = a;
   }
}

// por ex., no main
Reg ex = new Reg();
ex.setLine(0);
ex.setError(0);
```

```
Class Reg {
   int line;
    int errorID;
   public void setLine(int a) {
      This.line = a;
   }
   public void setError(int a) {
      This.errorID = a;
   }
   // constructor
   Public Reg(int a, int b) {
      This.line = a;
      This.errorID = a;
   }
}

// por ex., no main
Reg ex = new Reg(0, 0);
```

O Java utiliza um colector de objectos obsoletos (*garbage collector*), que é um programa, incluído na plataforma que executa os *bytecodes*, que vai libertando posições de memória relativas a objectos que não são mais utilizados durante a execução de uma determinada aplicação. Como a libertação dinâmica de memória é realizada pelo colector de objectos obsoletos, a linguagem não necessita de uma instrução como a instrução `delete` do C++.

5.2 Noção de referência

Em Java não existem ponteiros (também designados por apontadores). Em Java todas as variáveis, à excepção das variáveis de tipo primitivo (`long`, `int`, `short`, `byte`, e `boolean`) identificam a referência para um determinado objecto. Quando se declara uma variável como sendo do tipo da classe especificada e se lhe atribui um objecto, estamos a dizer semanticamente que esta variável é uma referência a um objecto armazenado algures na memória.

Uma referência pode por exemplo não referenciar qualquer objecto (por exemplo na altura da declaração: `FileReader inFile;`), ou pode ser-lhe atribuída a referência `null` (`inFile = null`);

5.3 Noção de package

Em Java não existem os ficheiros separados ".h" e ".c" e a importação de bibliotecas não é realizada com a directiva `#include`. Java utiliza *packages* que são referenciadas com a palavra `import`.

O termo *package* refere-se a um pacote de software pré-desenvolvido, vulgarmente designado por biblioteca, e que se encontra à disposição do programador. Um programador pode inclusive desenvolver os seus próprios pacotes de software. O código abaixo apresenta a definição da package `myFirstPackage`. Todas as classes desenvolvidas para este pacote de software devem estar localizadas debaixo do directório designado por `myFirstPackage`.

```
Package myFirstPackage;        // myFirstPackage é a designação do pacote software
```

```
// a palavra chave public especifica que esta class e pode ser acedida
// externamente ao pacote myFirstPackage
Public class myClass {
    ...
}
```

Para se utilizarem classes do pacote de software myFirstPackage noutros pacotes de software deve-se incluir o enunciado import seguido da classe a importar. As duas linhas de código seguintes apresentam dois métodos para importar classes de pacotes de software (neste caso é dado como exemplo a package myFirstPackage):

```
Import myFirstPackage.*; // dizemos para procurar no pacote myFirstPackage pelas classes utilizadas
Import myFirstPackage.myClass; // identificamos explicitamente a classe utilizada no pacote myFirstPackage
```

5.4 Excepções

A linguagem Java inclui um esquema para geração e tratamento de excepções. No Java é utilizado a construção try{…} catch(){..} para canalizar as excepções de forma a que quando aconteçam o programa possa provocar determinada reação. Quando se utilizam métodos que geram excepções, essas mesmas excepções têm de ser canalizadas por estruturas try-catch ou então temos de transmitir (propagar) as mesmas para o invocador do método que as utiliza (utiliza-se a palavra reservada throws a seguir ao cabeçalho do método, e seguida do nome da excepção a propagar). O código seguinte apresenta uma versão da classe readFile na qual no método main se transmitem as excepções que possam ocorrer durante a execução deste método.

```
import java.io.*;
//import java.io.FileReader;

public class readFile {

    public static void main(String args[])
    throws FileNotFoundException, IOException {

        // create a new instance of the class FileReader
        // which opens the file with the name "input.txt"
        FileReader inFile = new FileReader("input.txt");

        // Read each char of the file
        int c = inFile.read();
        while(c != -1) {
            System.out.println((char) c);
            c = inFile.read();
        }

        // close the file
        inFile.close();
    }
}
```

5.5 Comparação entre linguagens

A tabela abaixo ilustra algumas das diferenças entre a linguagem C/C++ e Java.

	C/C++	Java
Boolean	Bool	boolean
Chars	Char	Char (representação de 16 bits)
Tipos inteiros	Int, unsigned int	Int (representação de 32 bits sempre com sinal)
	Short, unsigned short	Short (representação de 16 bits sempre com sinal)
	Long, unsigned long	Long (representação de 64 bits sempre com sinal)
Tipos reais	float double	Float (representação em vírgula flutuante de 32 bits, sempre com sinal) Double (representação em vírgula flutuante de 64 bits, sempre com sinal)
Arrays	Int a[3];	Int[] A = new int[3]; (A é uma referência a um array de 3 elementos inteiros)
	-	A.length (identifica o tamanho do array A)

Strings	char name[10];	char[] name = new char[10]; // como array
		String name; // Java incluiStrings

Importar de bibliotecas	#include ...	Import ...;

6 Criação de documentação

Um ponto importante no desenvolvimento de software é a documentação do código. Parte desta documentação esclarece determinadas decisões ou descreve o que é feito por cada método. Para documentar enunciados são normalmente utilizadas as duas barras para a direita (`// isto é uma linha de comentário em Java`). Para comentar cabeçalhos de métodos, atributos de classes, a própria classe, ou o ficheiro em questão é utilizado o mesmo estilo de documentação da linguagem C (`/* Isto é um comentário que pode ter mais do que uma linha de texto */`).

A tecnologia Java integra um gerador de documentação que se baseia em determinados comentários no programa para produzir, por exemplo, ficheiros `html` que representam também a relação de classes do programa[6]. O comando `javadoc` pode ser utilizado para este fim (exemplo: `javadoc myClass.java`). Para tal teremos que indicar à ferramenta quais os comentários inseridos no código que desejamos ver nos documentos `html` relativos à documentação. Esses comentários são identificados por `/** Comentários para serem utilizados pelo javadoc */`. Para além desta identificação existem também directivas que, por exemplo, relacionam certos comentários a argumentos dos métodos.

[6] Podem ser encontradas ferramentas do mesmo tipo para outras linguagens de programação. Uma dessas ferramentas é o Doxygen (http://www.stack.nl/~dimitri/doxygen/).

7 Sumário

Este documento apresenta algumas das propriedades da tecnologia e linguagem de programação Java. Outras propriedades serão apresentadas em documentos posteriores. Essas propriedades incluirão mais conceitos de classes, interfaces, tipos, asserções, etc.

8 Bibliografia

[1] **[online]** The Java Tutorial: http://java.sun.com/docs/books/tutorial/index.html

São aconselhados os seguintes tópicos: *Trails Covering the Basics: Your First Cup of Java: Detailed instructions to help you run your first program; Getting Started; Learning the Java Language; Essential Java Classes*.

[2] Ken Arnold, James Gosling, David Holmes: *The Java Programming Language*, Third Edition. Addison-Wesley, 2000

[3] **[online]** THE JAVA LANGUAGE SPECIFICATION (livro disponível de borla! **CUIDADO: NÂO É UM LIVRO POR ONDE SE DEVA COMEÇAR A APRENDER JAVA OU PROGRAMAÇÃO**): http://java.sun.com/docs/books/jls/

9 Referências complementares

[4] *The GNU Compiler for the Java ™ Programming Language*, http://gcc.gnu.org/java/

[5] Java 2 SDK (*standard development kit*), v.1.4.1, http://java.sun.com/j2se/1.4.1/download.html

1 Introdução 1
2 Primeiro Exemplo 2
3 Ler de um ficheiro de texto 2
4 Escrever num ficheiro de texto 4
5 Como migrar do C para o Java 5
5.1 Noção de Classe e Objecto 5
5.2 Noção de referência 6
5.3 Noção de package 6
5.4 Excepções 7
5.5 Comparação entre linguagens 8
6 Criação de documentação 8
7 Sumário 9
8 Bibliografia 9
9 Referências complementares 9