Programação Orientada por Objectos

Entrada e Saída de dados

Prof. Rui César das Neves, Prof. José Cordeiro

Departamento de Sistemas e Informática

Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal

2014/2015

Sumário

- Entrada de dados a partir do teclado
  - A Classe **`Scanner`** para leitura do teclado.
  - Tokens
- Ficheiros
  - A Classe **`File`** do Java.
  - Escrita em ficheiros de texto com **`FileWriter`**
  - Leitura de ficheiros de texto com **`FileReader`**
  - Leitura de ficheiros de texto com **`Scanner`**
- Serialização
  - Conceito de Serialização
  - Escrita de ficheiros binários: **`ObjectOutputStream`**
  - Leitura de ficheiros binários: **`ObjectInputStream`**

Classe Scanner

- A classe **`Scanner`** permite converter texto para tipos primitivos.
  - Um objeto da classe **`Scanner`** pode obter o texto das mais diversas fontes tais como objectos do tipo **`String`**, **`File`** ou do teclado.
  - A proveniência do texto é indicada como argumento do construtor.
  - A classe **`Scanner`** tem diversos métodos que permitem separar o texto em tokens que são lidos individualmente
- Um **Token** é uma sequência de caracteres separados por delimitadores.
  - Por omissão, os delimitadores são espaços em branco, tabs e mudanças de linha.

```
import java.util.Scanner;

class TesteDoScanner{
    public static void main(String[] args) {

        String string = "Vamos experimentar o scanner";

        Scanner scannerString = new Scanner(string);

        while (scannerString.hasNext())
               System.out.println ("Token:" + scannerString.next());
    }
}
```

Leitura do Teclado

- Para ler texto do canal de entrada padrão - **`System.in`** - normalmente associado ao teclado, é preciso criar primeiro um **`Scanner`** sobre o canal **`System.in`**:
  - **`Scanner sc = new Scanner(System.in);`**

- Para cada tipo primitivo existente na classe **`Scanner`** existe um método correspondente - com a assinatura **`next`*`Xxx`*`()`** - que retorna um valor desse tipo.
  - Exemplos de algumas leituras de tokens:
    - **`String umToken = sc.next() ;  //Lê um token`**
    - **`int num1 = sc.nextInt() ;  //Lê um token inteiro`**
    - **`double num2 = sc.nextDouble();  //Lê um token real`**
    - **`String linha = sc.nextLine() ;  //Lê uma linha`**

- Se o token não puder ser interpretada como sendo do tipo que se pretende ler, é lançada uma excepção do tipo **`InputMismatchException`**.

Leitura do Teclado

```
import java.util.InputMismatchException;
import java.util.Scanner;

class TesteDoScannerInteiros{
  public static void main(String[] args) {

     // 1. Cria o objeto da classe Scanner
    Scanner scanTeclado = new Scanner(System.in);

     // 2. pede um inteiro ao utilizador
    System.out.print("Introduza um inteiro: ");

    try{
       // 3. Lê o token com o inteiro do teclado
      int numero = scanTeclado.nextInt();
      System.out.println ("Numero introduzido: " + numero);
    }

    catch (InputMismatchException e){
       // caso o token não seja passível de converter em inteiro
      System.out.println ("Erro na leitura do valor: " + e );
    }

  }
}
```

Ficheiros

- Um ficheiro é uma entidade de armazenamento permanente de informação.
  - A manipulação de ficheiros é feita utilizando canais.
  - Existem canais diferentes
    - para a leitura de informação e que derivam das classes: **`InputStream`** ou **`Reader`**
    - Para a escrita de informação e que derivam das classes: **`OutputStream`** ou **`Writer`**
- De acordo com o tipo de informação que podem armazenar, os ficheiros podem ser classificados como:
  - Ficheiros de Texto (Orientado ao caracter)
    - Derivam das classes : **`Reader`** e **`Writer`**
    - De fácil leitura por humanos com ferramentas simples (**`type`**, **`more`**, **`edit`**, ...).
    - Linhas com comprimento variável.
    - Cada linha termina com uma marca.
    - Exemplo: ficheiro com código-fonte em Java.
  - Ficheiros Binários (Orientado ao byte)
    - Derivam das classes : **`InputStream`** e **`OutputStream`**
    - Apropriado para o processamento por ferramentas automáticas.
    - Armazenamento de tipos primitivos, serialização de objectos, etc.
    - Armazenamento eficiente ocupando menos memória.

Ficheiros – Classe File

- **Classe File**: Representa os ficheiros e directorias de um sistema de ficheiros.

  - **CONSTRUTORES**

`File(String caminho)`	construtor de directórios/ficheiros
`File(String caminho&filename)`	construtor com caminho e nome do ficheiro

  - **MÉTODOS**

`boolean canRead()`	ficheiro/directório pode ser lido
`boolean canWrite()`	pode-se gravar no ficheiro/directório
`boolean delete()`	apaga ficheiro/directório
`boolean exists()`	verifica se ficheiro/directório existem
`boolean isAbsolute()`	verifica se caminho é absoluto
`boolean isDirectory()`	verifica se objecto é directório
`boolean isFile()`	verifica se objecto é ficheiro
`boolean mkdir()`	cria directório do objecto
`boolean mkdirs()`	cria directórios do caminho
`boolean renameTo(String novo)`	muda nome do ficheiro/directório para novo

Ficheiros – Classe File

```
import java.io.File;

class TesteDeFile {

   public static void main(String[] args){

   String nomeDoFicheiro = "dados.txt";

   // cria o ficheiro
   File meuFicheiro = new File(nomeDoFicheiro);

   // verifica se o ficheiro foi criado
   if (meuFicheiro.exists())

      System.out.println(meuFicheiro.getName() + " existente");

   else

      System.out.println(meuFicheiro.getName() + " não existente");

  }
}
```

Escrita em Ficheiros de Texto

☐ Passos para escrita de um ficheiro de Texto com `FileWriter`

```
public class TesteFileWriter{
  public static void main (String arg[]) {
      // 1. Criar um objecto do tipo File. Permite manipular atributos de um ficheiro.
    File ficheiro = new File ("textOutput.txt");
    try {
      // 2. Criar um canal  FileWriter ligado ao objecto File.
      // Associa um ficheiro de texto a um canal de escrita.
    FileWriter fileWriter = new FileWriter (ficheiro);
      // 3. Criar um canal BufferedWriter ligado ao objecto FileWriter.
      // Escrita mais eficiente
    BufferedWriter bufferedWriter = new BufferedWriter (fileWriter);
      // 4. Criar um canal PrintWriter ligado ao objecto BufferedWriter.
      // Permite a utilização dos métodos print e println.
    PrintWriter printWriter = new PrintWriter (bufferedWriter);
      // 5. Escrever no ficheiro
    printWriter.println("Saida c/ PrintWriter. Tipos primitivos conv. em strings ");
    boolean aBoolean =  false;                    int anInt =  1234567;
    printWriter.println (aBoolean);               printWriter.println (anInt);
      // 6. forçar a escrita em disco com o método .flush()
    printWriter.flush();
    }
    catch (IOException e){  System.out.println(e.getMessage()); }
  }
}
```

Leitura de Ficheiros de Texto

- Passos para leitura de um ficheiro de Texto com `FileReader`

```
public class TesteFileReader {
  public static void main (String arg[]) {

     // 1. Criar um objecto do tipo File com o nome do ficheiro de onde vamos ler
    File readingFile = new File ("textOutput.txt");

    try {
       // 2. Criar um canal  FileReader ligado ao objecto File.
      FileReader fileReader = new FileReader (readingFile);
       // 3. Criar um canal  tampão BufferedReader ligado ao objecto FileReader.
      BufferedReader bufferedReader = new BufferedReader (fileReader);

       // 4. Ler as linhas do ficheiro
      String line = "";
      while (line != null){
        line = bufferedReader.readLine();
        System.out.println(line);
      }
    }
    catch (IOException e){
      System.out.println(e.getMessage());
    }
  }
}
```

Leitura de Ficheiros de Texto

- Passos para leitura de um ficheiro de Texto com Scanner

```
public class TesteLeituraComScanner {
    public static void main (String arg[]) {
            // 1. Criar o objeto file com o nome do ficheiro a ler
        File file = new File ("textOutput.txt");
            // 2. Declarar o objeto scanner para ler do ficheiro
        Scanner scanner;
        try {
                // 3. Criar o objeto Scanner com o objeto File a ser lido
            scanner = new Scanner (file);
                // 4. Ler as linhas do ficheiro
            String primeiraLinha = scanner.nextLine();
            System.out.println ("String lida: " + primeiraLinha);
            boolean segundaLinha = scanner.nextBoolean();
            System.out.println ("Boolean lido: " +segundaLinha);
            int terceiraLinha = scanner.nextInt();
            System.out.println ("Inteiro lido: " +terceiraLinha);
        }
        catch (InputMismatchException e) {
            System.out.println ("Mismatch exception:" + e );
        }
        catch (FileNotFoundException e) {
            System.out.println ("Ficheiro não encontrado!");
            System.exit (0);
        }
    }
}
```

Serialização

- Serialização (serialization) – processo de escrita em sequência de um ou mais objetos.
  - Escrever, para um canal (stream), um objecto de uma determinada classe incluindo os objetos que poderá referenciar nos atributos.
- Deserialização (deserialization)
  - Restaurar, a partir de um canal (stream) o(s) objeto(s) anteriormente serializados pela mesma ordem em que foram guardados.
- Em Java o algoritmo de serialização de dados garante:
  - Que quando os dados venham a ser lidos de um repositório serializado, todos os objetos com os respetivos atributos serão reconstruidos no estado em que estavam aquando da sua gravação.

Serialização – Implements Serializable

- A serialização é aplicável apenas a instâncias de classes que implementem a interface Serializable:

```
public class Data implements Serializable{
    private int ano;
    private int mes;
    private int dia;
}
```

- Ao declarar que um classe implementa a interface **`Serializable`**, o compilador gera dois métodos privados para essa classe:
  - `void writeObject ( ObjectOutputStream out ) throws IOException`
  - `Object readOject ( ObjectInputStream in ) throws IOException, ClassNotFoundException`
    - Um objeto "**`ObjectOutputStream`**" representa um canal especial que trabalha directamente sobre um ficheiro e que armazena objetos e valores simples, usando o método **`writeObject()`** o qual implementa um algoritmo de serialização (serialize).
    - Um objeto "**`ObjectInputStream`**" representa um canal especial que trabalha directamente sobre um ficheiro e lê objectos e valores simples, usando o método **`readObject()`** o qual implementa um algoritmo de deserialização (deserialize).

Serialização – Gravação em Ficheiro

- Considerando que
  - A classe **Pessoas** inclui um array de objetos da classe **Pessoa**
  - E que a classe **Pessoa** possui um atributo **dataNascimento** da classe **Data**
  - O método abaixo vai gravar num ficheiro binário
    - um objeto da classe **Pessoas**,
    - com todos os elementos do array de objetos (com todos os objetos da classe **Pessoa**)
    - e para cada objeto da classe **Pessoa**, a respectiva data de nascimento
    - Num ficheiro binário serializado a partir do qual será possível reconstituir completamente o objeto da classe **Pessoas**

```
public static void gravaFicheiro(Pessoas listaPessoas, String nomeFicheiro){
  try {
    ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeFicheiro));
    oos.writeObject(listaPessoas);
    oos.flush();
    oos.close();
  }
  catch (IOException e) {
      System.out.println(e.getMessage());
  }
}
```

Serialização – Leitura de Ficheiro

- Considerando que
  - Foi gravado através de uma **`ObjectOutputStream`** num qualquer ficheiro binário, um objeto da classe **`Pessoas`**
  - É possível lê-lo do ficheiro reconstituindo completamente o seu estado no momento da gravação através do método abaixo:

```
public static Pessoas leFicheiroSerializado(String nomeFicheiro){
  Pessoas listaPessoas;
  try{
    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeFicheiro));
    listaPessoas = (Pessoas)ois.readObject();
    ois.close();
  }
  catch (IOException e) {
    System.out.println(e.getMessage());
    listaPessoas = new Pessoas(10);
  }
  catch (ClassNotFoundException e) {
    System.out.println(e.getMessage());
    listaPessoas = new Pessoas(10);
  }
  return listaPessoas;
}
```

Serialização – modificador transient

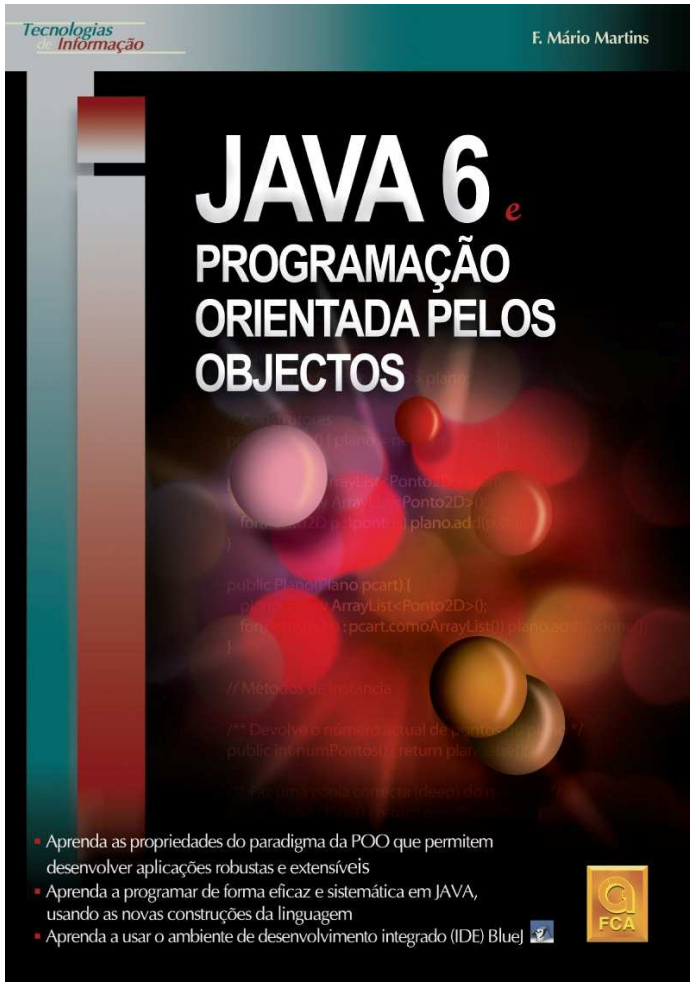
- Na serialização o Java escreve no ficheiro todos os atributos, não **`static`**, da classe que implementa a interface **`java.io.Serializable`**.
- Podemos indicar que não pretendemos que um atributo seja escrito (eventualmente porque o seu tipo é de uma classe que não implementa a interface **`Serializable`**) desde que utilizemos o modificador **`transient`**:
  - **`private transient Color cor; //Color não é Serializable`**
- Se for importante a informação do atributo teremos que implementar as nossas versões dos métodos que fazem a escrita e leitura da informação:
  - **`private void writeObject(java.io.ObjectOutputStream oos) throws IOException`**
  - **`private void readObject(ObjectInputStream ois) throws ClassNotFoundException, IOException`**
  - Estes métodos devem chamar, normalmente no seu início, o comportamento por omissão:
    - **`oos.defaultWriteObject();`**
    - **`ois.defaultReadObject();`**

Resumindo

- Entrada de dados a partir do teclado
  - **Criamos um objeto da classe Scanner:** `Scanner sc = new Scanner();`
  - **Verificamos se o próximo token é do tipo que pretendemos ler:** `sc.hasNextInt();`
  - **Caso seja, lemos o token com:** `sc.nextInt();`
- Ficheiros
  - **A escrita em ficheiros de texto permite ao utilizador abrir o ficheiro com qualquer editor de texto e comprender o seu conteúdo.**
  - **Mas não é apropriada para guardar informação sobre o estado de um conjunto de objetos.**
- Serialização
  - **O recurso à serialização revela-se o intrumento por excelência para gravar em ficheiros binários a informação sobre o estado de um conjunto de objetos em dado momento.**
  - **Exige que todas as classes cujos objetos possam ser gravados implementem a interface Serializable:** `public class Pessoas implements Serializable`
  - **A gravação inicia-se com a criação de um objeto da classe** `ObjectOutputStream`.
  - **Grava-se em ficheiro com o método** `writeObject(objetoAGravar)`
  - **A leitura inicia-se com a criação de um objeto da classe** `ObjectInputStream`
  - **Lê-se do ficheiro com o método** `readObject()`

Leitura Complementar

- Capítulo 10
  - Páginas 401 a 439
- Tutoriais:
  - http://docs.oracle.com/javase/tutorial/essential/io/index.html



Exercicio Prático

Criar uma classe **`ValorInvalidoException`** do tipo **`Exception`**:

- O construtor deve:
  - receber como argumentos o valor inválido e o nome da variável para a qual o valor foi mal inserido (Mês ou Dia). Ex.: **`ValorInvalidoException(mes, "mês")`**
  - Enviar uma mensagem no formato: **`O valor`** *`valor`* **`não é válido para o`** *`mês`***`.`**

```
public class ValorInvalidoException extends Exception {

   public ValorInvalidoException(int valor, String texto) {
      super("O valor "+ valor + " não é válido para o " + texto + ".");
   }
}
```

Exercicio Prático

Criar uma classe **Data** semelhante à obtida na aula anterior.

```
package Serializacao;
import java.io.Serializable;
public class Data implements Serializable{
  private int ano;
  private int mes;
  private int dia;
  public Data(int ano, int mes, int dia)
            throws ValorInvalidoException{
    this.ano = ano;
    this.mes = verificarMes(mes);
    this.dia = verificarDia(dia);}

  public int getAno() {
    return ano;}

  public int getMes() {
    return mes;}

  public int getDia() {
    return dia;}

  public void setAno(int ano){
    this.ano = ano;}

  public void setDia (int dia) throws
         ValorInvalidoException {
    verificarDia(dia);}
```

```
  public void setMes(int mes)throws
           ValorInvalidoException{
    verificarMes(mes);}

  private void verificarDia(int dia)throws
           ValorInvalidoException{
      if ((dia>=1) && (dia <=31))
        this.dia = dia;
      else
        throw new
          ValorInvalidoException(dia,"dia");}

  private void verificarMes(int mes)throws
           ValorInvalidoException{
      if ((mes>=1) && (mes <=12))
        this.mes = mes;
      else
        throw new
          ValorInvalidoException(mes,"mes");}

  @Override
  public String toString (){
      return ano + "/" + mes + "/" + dia;
  }
}
```

Exercicio Prático

Criar uma classe **Pessoa** com os atributos **nome** e **dataNascimento**

```
package Serializacao;

import java.io.Serializable;

public class Pessoa implements Serializable{
  private String nome;
  private Data dataDeNas;

  public Pessoa(String nome, Data dataNas) {
    this.nome = nome;
    this.dataDeNas = dataDeNas;
  }

  public Data getDataDeNascimento() {
    return dataDeNas;
  }

  public void setDataDeNascimento(Data dataDeNas){
        this.dataDeNascimento = dataDeNascimento;
  }

  @Override
  public String toString(){
    return "Sr(a)" + nome + " nasceu em: " +
                    dataDeNascimento.toString();
  }
```

```
  public String getNome() {
    return nome;
  }

  public void setNome(String nome) {
    this.nome = nome;
  }
}
```

Exercicio Prático

Criar uma classe **Pessoas** com um array de objetos da classe **Pessoa** como atributo

```
package Serializacao;
import java.io.Serializable;
public class Pessoas implements Serializable{
  Pessoa[] listaPessoas;

  public Pessoas(int comprimentoListaPessoas) {
      listaPessoas = new Pessoa[comprimentoListaPessoas];
    }

  public void inserePessoa(Pessoa pessoa){
      if (listaPessoas == null)
        throw new NullPointerException("Lista de Pessoas não foi criada");
      int i=-1;
      do  i++;
      while ((listaPessoas[i] != null) && (i<listaPessoas.length));
      if (i<listaPessoas.length)
        listaPessoas[i] = pessoa;
  }

  public Pessoa[] getListaPessoas(){
      return listaPessoas;
  }

  public Pessoa getPessoaAt(int i){
      return listaPessoas[i];
  }
}
```

Exercicio Prático

No método **`main`** gravar e ler um objeto da classe **`Pessoas`**

```
package Serializacao;
import java.io.*;
public class ProgramaSerializacaEscritaLeitura {
  public static void main(String[] args){
     try {
       Data data1 = new Data (2010, 1, 2);
       Data data2 = new Data (2010, 3, 4);
       Pessoa pessoa1 = new Pessoa ("Jorge", data1);
       Pessoa pessoa2 = new Pessoa ("Ana", data2);
       Pessoas lista = new Pessoas(3);
       lista.inserePessoa(pessoa1);
       lista.inserePessoa(pessoa2);
       gravarFicheiroSerializado(lista, "FicheiroDePessoas.dat");
       Pessoas novaLista = lerFicheiroSerializado("FicheiroDePessoas.dat");
       Pessoa[] pessoas = novaLista.getListaPessoas();
       for (int i=0; i < pessoas.length; i++)
         System.out.println(pessoas[i]);
       }
     } catch (ValorInvalidoException e) {
       System.out.println("Data impossível!");
     }
  }

  // os métodos estão no slide seguinte
```

Exercicio Prático

No método **`main`** gravar e ler um objeto da classe **`Pessoas`**

```
  public static void gravarFicheiroSerializado (Pessoas lista, String fileName) {
    try {
       ObjectOutputStream oos = new ObjectOutputStream (new FileOutputStream(fileName));
       oos.writeObject(lista);
       oos.flush();
       oos.close();
    } catch (IOException e) { System.out.println(e.getMessage()); }
  }

  public static Pessoas lerFicheiroSerializado (String fileName) {
    Pessoas listaPessoas;
    try{
      ObjectInputStream oin = new ObjectInputStream(new FileInputStream(nomeFicheiro));
      listaPessoas = (Pessoas) oin.readObject();
      oin.close();
    }catch (IOException e) {
      System.out.println(e.getMessage());
      listaPessoas = new Pessoas(10);
    }catch (ClassNotFoundException e) {
      System.out.println(e.getMessage());
      listaPessoas = new Pessoas(10);
    }
    return listaPessoas;
  }
}
```