

## Aplicação Final GameLand

A submeter  
no moodle

Objectivos Gerais:

**Analisar e Resolver Problemas Computacionalmente** usando o paradigma procedimental estruturando o programa em módulos.

Codificar programas utilizando a linguagem Java na perspectiva essencialmente procedimental.

Assumir atitudes de aprendizagem activa, colaborativa e responsável, de trabalho persistente e de **aplicação de espírito crítico na análise e resolução de problemas.**

## Objectivos Específicos

Mediante a especificação do problema o aluno deverá ser capaz de:

- Definir estruturas de dados adequadas (neste caso devem incluir *arrays unidimensionais e bidimensionais*);
- Utilizar ficheiros de texto como fontes de entrada de dados (leitura) e como destinos de informação (escrita);
- Analisar, conceber e descrever algoritmos estruturando-os em módulos;
- Codificar a aplicação em Java (na perspectiva procedimental) tendo em conta a utilização das normas adequadas de codificação e documentação;
- Elaborar um plano de testes adequado.

Maria da Conceição Neves

## Especificação do problema

Ler enunciado

## Exemplos do tipo de ficheiros disponíveis

20181020.txt - Bloco de notas

```
Eicheiro  Editar  Formatar  Ver  Ajuda

110-Estratégia-150
120-Estratégia-200
210-Aventura-150
220-Aventura-180
310-Ação-100
|
330-Ação-120
```

Estes ficheiros são meramente  
exemplificativos

thebest.txt - Bloco de notas

```
Eicheiro  Editar  Formatar  Ver  Ajuda

ac@abc.pt;      Ana Costa; 31/1/2005; The Best
rg@xyz.pt;Rui Gomes;  11/2/1999  ; The Best
ls@xyz.pt;Luís Sá Silva;  1/12/2000  ; The Best
```

20181020\_120.txt - Bloco de notas

```
Eicheiro  Editar  Formatar  Ver  Ajuda
ac@abc.pt;150

ls@xyz.,pt;180
```

Maria da Conceição Neves

# Processo de Desenvolvimento

## PARTE 1: Análise dos Requisitos

- Requisitos :
  - A aplicação deve ser orientada por um menu de funcionalidades a ser apresentado ao utilizador e que em função da escolha do utilizador disponibilizará a funcionalidade seleccionada. A aplicação só terminada quando a opção seleccionada for TERMINAR
  - Análise das funcionalidades de 1 a 12

Maria da Conceição Neves

## PARTE 2: Conceção da solução

1. Estrutura de Dados principal (ED do main)
2. Algoritmo do módulo principal (main)
3. Para cada funcionalidade
  - ED local
  - Algoritmo

Maria da Conceição Neves

## PARTE 3: Implementação e Teste

1. Após concebida a solução algorítmica de cada módulo passamos à fase de implementação e teste

- Usar as boas práticas de codificação:
  - Identificadores adequados
  - Evitar “valores literais” definindo constantes com esses valores e usar essas constantes
  - Evitar métodos com muitas linhas de código desdobrando-os
  - Desdobrar os métodos por várias classes em função dos seus objetivos
  - Usar comentários Javadoc
- À medida que forem sendo implementadas as funcionalidades deverão ser testadas

Maria da Conceição Neves

### Esboço da Estrutura de Dados a utilizar

**int nParticipantes**

3

**Informação dos participantes**  
String [MAX\_PARTIC ][ 4 ] participantes

ac@abc.pt	Ana Costa	31/1/2005	The Best
rg@xyz.pt	Rui Gomes	11/2/1999	The Best
ls@xyz.pt	Luís Sá Silva	9/01/2000	The Best

**Pontos nos jogos**  
int [MAX\_PARTIC ][N\_JOGOS] pontos

jogo1	jogo2	...
100	100	...
161	0	...
0	10	0...

**Premios nos jogos**  
double [MAX\_PARTIC ][N\_JOGOS] premios

jogo1	jogo2	...
220	2110	...
200	0	...
0	200	...

**OBS: Sempre que haja a informação disponível as estruturas de dados devem estar atualizadas**

## Criar / Utilizar outras classes como:

- Classe Utilitarios – disponibiliza métodos vários como:
  - Cálculo de idade
  - Reduzir nome – apelido + 1ª letra do nome
- Classe LogErros – disponibiliza métodos associados à criação de um ficheiro de registo de situações de erro.

Maria da Conceição Neves

Segue-se uma proposta de uma solução parcial do problema que deve analisar com espírito crítico.

### Módulo menu

**Definir menu() : int**

int op;

Escrever(“

Ler Informação do ficheiro de texto ... 1

... 2

... 3

...

Terminar 0

Qual a sua opção? “)

Ler (op)

**Retornar op**

**Fim**

Maria da Conceição Neves

```
public class GameLand{
```

```
// Configuração da aplicação
private final static int MAX_PARTICIPANTES = 30;
private final static int N_CAMPOS_INFO= 4;
private final static int N_JOGOS = 6;
private final static int MAX_LINHAS_PAGINA = 3;
//.....
```

Pode-se colocar numa classe Configurações.java que só tem estas **constantes de classe** que nesse caso deverão ser **públicas**

```
private final static Scanner in= new Scanner(System.in);
```

```
private static int menu() {
    String texto = "\nMENU:"
        + "\n Ler      ...    1"
        + "\n          ...    2"
        //...
        + "\n FIM          ...    0"
        + "\nQual a sua opção?";
    System.out.printf("%n%s%n", texto);
    int op = in.nextInt(); in.nextLine();
    return op;
}
```

```
// Continua o código de outros métodos
```

Maria da Conceição Neves

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

```
String[] jogos= new String[N_JOGOS];
String[] participantes= new String[MAX_PARTICIPANTES][N_CAMPOS_INFO];
int nParticipantes= 0;
int[][] pontos= ...
double[][] prémios= new double[MAX_PARTICIPANTES][N_JOGOS];
int op;
do {
    op = menu();

    switch (op) {
        case 1:
            System.out.println("Qual a data do evento (AAAAMMDD?");
            String nomeFich = in.nextLine();
            if(carregarJogosDoEvento(nomeFich+".txt", jogos)){
                System.out.println("Jogos carregados com sucesso");
            }else{
                System.out.println("Erro no carregamento dos jogos. Verifique ficheiro");
            }
        break;
    }
}
```

Maria da Conceição Neves

```

    case 2:
        // ...
        break;
    case 3:
        // ...
        break;
    // ...
    case 0:
        System.out.println(
            "Já fez todas as gravações necessárias? Confirma terminar(s/n)?");
        char resp = (in.next()).charAt(0);
        if (resp != 's' && resp != 'S') {
            op = 1;
        }
        break;

    default:
        System.out.println("Opção incorreta. Repita");
        break;
}
} while (op != 0);
}

```

Maria da Conceição Neves

1

```

private static boolean carregarJogosDoEvento(String nomeFichJogos, String[] jogos)
throws FileNotFoundException {

    Scanner flInput = new Scanner(new File(nomeFichJogos));
    int i = 0;
    while (flInput.hasNextLine() && i < N_JOGOS) {
        String linha = flInput.nextLine();
        // Verifica se linha não está em branco
        if ((linha.trim()).length() > 0) {
            jogos[i] = linha;
            i++;
        }
    }
    flInput.close();
    if (i == N_JOGOS) {
        return true;
    }
    return false;
}

```

Maria da Conceição Neves

2

```

private static void visualizarInfoJogos(String[] jogos) {
    System.out.println("Jogos do evento");
    System.out.printf("%15s%15s%15s%n",
        "ID do jogo", "Tipo de jogo", "Max. de pontos");
    for(int i=0;i<jogos.length;i++){
        String[] temp=jogos[i].split("-");
        System.out.printf("%15s%15s%15s%n",
            temp[0], temp[1], temp[2]);
    }
}

```

Maria da Conceição Neves

3

**DEFINIR lerInfoFicheiro(String nomeFich,  
String[][] info, int nElems) : int**

```

fIn=AbrirFichTextoParaLeitura( nomeFich )
ENQUANTO( não fim fIn AND
            nElems< linhas matriz)
    aux=lerLinha(fIn);
    nElems=guardarDados(aux,
                        info, nElems)
FIM ENQUANTO
FecharFich(fIn)

retorna nElems

```

**FIMDEFINIR**

Maria da Conceição Neves



3

```

/**
 * Carrega informação do ficheiro de texto para memória
 *
 * @param nomeFich - nome do ficheiro que contem a informação a guardar
 * @param info - matriz de strings para guardar a info do ficheiro
 * @param nElems - número de elementos já existentes na matriz
 * @return o número final de elementos na matriz
 * @throws FileNotFoundException
 */
public static int lerInfoFicheiro (String nomeFich, String[][] info, int nElems) throws
FileNotFoundException {
    Scanner flnput = new Scanner(new File(nomeFich));
    int nElemsInic=nElems;
    while (flnput.hasNext() && nElems< MAX_PARTICIPANTES) {
        String linha = flnput.nextLine();
        // Verifica se linha não está em branco
        if (linha.trim().length() > 0) {
            nElems = guardarDados(linha, info, nElems);
        }
    }
    flnput.close();
    if(nElems-nElemsInic!=3){
        nElems= nElemsInic;
    }
    return nElems;
}

```

Maria da Conceição Neves

3

```

* Accede à informação de uma linha do ficheiro e guarda-a na estrutura dados se ainda
* não existe linha com aquele valor no 1º elemento
* @param linha - String com o conteúdo de uma linha do ficheiro
* @param info - matriz de strings com a informação lida do ficheiro
* @param nElems - número de elementos existentes na matriz
* @return - o novo número de elementos da matriz
*/

```

```

private static int guardarDados(String linha, String[][] info, int nElems) {

```

```

    String[] temp = linha.split(SEPARADOR_DADOS_FICH);

```

```

    if (temp.length == N_CAMPOS_INFO) {

```

```

        String num = temp[0].trim();

```

```

        int pos = pesquisarElemento(num, nElems, info);

```

```

        if (pos == -1) {

```

```

            info[nElems][0] = num;
            info[nElems][1] = temp[1].trim();
            info[nElems][2] = temp[2].trim();
            info[nElems][3] = temp[3].trim();

```

```

            nElems++;

```

```

        }

```

```

    }

```

```

    return nElems; }

```

Este bloco deve substituir o do lado cinzento

```

for(int i=0; i<N_CAMPOS_INFO;i++){
    info[nElems][i] = temp[i].trim();
}

```

Maria da Conceição Neves

```

/**
 * Pesquisar linha de matriz por primeiro elemento da linha
 *
 * @param valor - elemento a pesquisar
 * @param nEl - nº de elementos da matriz
 * @param matriz - matriz com a informação
 * @return -1 se não existe linha com esse valor ou o
 *         nº da linha cujo primeiro elemento é esse valor
 */
public static int pesquisarElemento(String valor, int nEl, String[][] mat) {

    for (int i = 0; i < nEl; i++) {
        if (mat[i][0].equals(valor)) {
            return i;
        }
    }
    return -1;
}

```

3

Maria da Conceição Neves

```

/**
 * Visualizar toda a informação ( paginada)dos participantes existente em memória
 * @param matriz- matriz com a informação a listar
 * @param nEl – nº de linhas com informação
 */
private static void listagemPaginada(String[][] matriz, int nEl) {
    int contPaginas = 0;
    for (int i = 0; i < nEl; i++) {
        if (i % MAX_LINHAS_PAGINA == 0) {
            if (contPaginas > 0) { pausa(); }
            contPaginas++;
            System.out.println("\nPÁGINA: " + contPaginas);
            cabecalho(); }
        mostrarParticipante(out , matriz[i])
        for (int j = 0; j < N_CAMPOS_INFO; j++) {
            if (j == 1) {
                System.out.printf("%30s", matriz [i][j]);
            } else {
                System.out.printf("%10s", matriz[i][j]);
            }
        }
        System.out.println("");
    } pausa();
}

```

4

Maria da Conceição Neves

4

Esta é uma solução para a **listagem paginada**,

- Informação é organizada em páginas
- Cada página será constituída, no máximo, por MAX\_LINHAS\_PAGINA ( constante definida na classe).
- Sempre que muda de página o programa pára até o utilizador carregar na tecla ENTER
- A nova página inicia sempre com o cabeçalho.

```
private static void cabecalho() {
    System.out.printf("%50s%n", "PARTICIPANTES");
    System.out.printf("%75s%n",
        "=====");
}

private static void pausa() {
    System.out.println("\n\nPara continuar digite ENTER\n");
    in.nextLine();
}
```

Maria da Conceição Neves

5

```
/**
 * Atualiza informação alterável de um participante
 *
 * @param eMail – mail do participante
 * @param - matriz com toda a informação dos participantes
 * @param nElems - número de elementos
 * @return false se o eMail não foi encontrado ou true se foi encontrado e
 * atualizado
 */
public static boolean atualizarInfoParticipante(String eMail,
    String[][] matriz, int nElems) {

    int pos;
    if ((pos = pesquisarElemento(eMail, nElems, matriz)) > -1) {
        int op;
        do {
            Formatter out = new Formatter(System.out);
            mostrarParticipante(out, matriz[pos]);
            op = menuInfoParticipante();
            switch (op) {
                case 1:
                    System.out.println("Novo EMail:");
                    matriz[pos][0] = in.nextLine();
                    break;
            }
        } while (op != 0);
    }
}
```

O algoritmo desta funcionalidade foi previamente esboçado, mas não está aqui apresentado

Maria da Conceição Neves

5

```

case 2:
    System.out.println("Novo nome:");
    matriz[pos][2] = in.nextLine();
    break;
case 3:
    System.out.println("Nova data nascimento:");
    matriz[pos][3] = in.nextLine();
    break;
case 0:
    System.out.println("FIM");
    break;
default:
    System.out.println("Opção incorreta");
    break;
}
} while (op != 0);
return true;
}
System.out.printf("O participante %s não foi encontrado!", eMail);
return false;
}

```

Maria da Conceição Neves

5

```

/**
 * Mostrar a informação de um participante separada por ;
 *
 * @param out - instancia de formatter associado ao stream pretendido
 * @param participante - array de Strings com a informação do participante
 */
private static void mostrarParticipante(Formatter out,
                                         String[] participante) {
    for (int j = 0; j < N_CAMPOS_INFO; j++) {
        if (j == 1) {
            out.format("%30s;", participante[j]);
        } else {
            out.format("%12s;", participante[j]);
        }
    }
}
}

```

Maria da Conceição Neves

```

/**
 * Apresenta o menu de opções de atualização de dados
 * atualizáveis
 * do participante
 *
 * @return um inteiro que é a opção escolhida
 */
private static int menuInfoParticipante() {
    String texto = "ATUALIZAR INFORMAÇÃO DE PARTICIPANTE"
        + "\n EMail      ... 1"
        + "\n NOME      ... 2"
        + "\n DATA NASCIMENTO ... 3"
        + "\n TERMINAR    ... 0"
        + "\n\nQUAL A SUA OPÇÃO?";
    System.out.printf("%n%s%n", texto);
    int op = in.nextInt();
    in.nextLine();
    return op; }

```

5

Maria da Conceição Neves

## Classe Utilitários e Classe TesteUtilitários

Testar o software desenvolvido é essencial. Uma abordagem será associar a cada classe que implementa um conjunto de funcionalidades uma classe de testes que invoca esses métodos em situações determinadas permitindo em qualquer momento confirmar a validade das funcionalidades implementadas.

Esta foi a abordagem utilizada na implementação das funcionalidades da classe Utilitários.

Recomenda-se

- Analisar os métodos implementados em **classe Utilitários**
- Analisar a classe de testes designada por **classe TesteUtilitários**
  - Método main executa um plano de testes

Maria da Conceição Neves

- Por questão de organização do código colocou-se a classe **TestesUtilitarios** num novo package cujo nome escolhido foi **testespackage**
  - Criar o testespackage
    - No diretório **Source Packages** com o botão direito do rato seleccionar **New > Java package > nome testespackage**.
  - Neste package criou-se a classe **TestesUtilitarios**
    - Após a conceção de um plano de testes que consiste em definir os métodos da classe Utilitários a invocar com que parâmetros de entrada e quais as respectivas saídas esperadas passa-se à invocação desse plano no método **main()**.

Maria da Conceição Neves

#### UMA PROPOSTA DE IMPLEMENTAÇÃO DA CLASSE **TesteUtilitarios**

Nesta implementação optou-se por criar para cada método a testar um outro método de teste que tem mais um parâmetro que é a resposta esperada. É este método que invoca o método a testar e compara o seu retorno com a resposta esperada retornando o String **SUCESSO** OU **FALHA** conforme respondeu ou não como o esperado.

POR EXEMPLO PARA TESTAR O MÉTODO  
**verificaSeTemApelido** (String nome, String apelido)  
 CRIOU-SE O MÉTODO

```
public static String testeVerificaSeTemApelido(String nome, String apelido, boolean resp){
    String res="FALHA";
    if(resp== verificaSeTemApelido( nome,apelido) {
        res= "SUCESSO";
    }
    return res;
```

Maria da Conceição Neves

- No main() invocam-se os métodos de teste conforme planeados, isto é com os parâmetros de entrada definidos e a saída esperada
- A classe TestesUtilitarios deve importar a classe Utilitários

**import gameland.Utilitarios**

- Executar o plano de testes consiste em invocar o main() da classe TestesUtilitarios (botão direito do rato sobre a classe selecione Run File).
- O objetivo é obter SUCESSO para todos os métodos de teste invocados

Maria da Conceição Neves

```
public class Utilitarios {
    /**
     * Verifica se um determinado nome contém apelido
     * @param nome – nome completo
     * @param apelido – apelido a procurar
     * @return true ou false conforme o nome contém ou não o apelido
     */
    public static boolean verificaSeContemApelido(String nome, String apelido){
        // String implements CharSequence
        return nome.contains(apelido);
    }
    /**
     * Dado o nome completo retorna o apelido e a primeira letra .
     * @param str nome completo
     * @return o String último nome seguido da 1ª letra.
     */
    public static String apelidoELetraInicial(String str) {
        str=str.trim();
        int last = str.lastIndexOf(' '); // posição do último espaço
        // substring da posição do último espaço +1 até ao fim.
        String resultado =str.substring(last+1)+" "+str.substring(0, 1)+".";
        return resultado;
    }
}
```

Maria da Conceição Neves

```

/**
 * Converter data no formato dd/mm/ano em aaaammdd
 * @param data data no formato dd/mm/ano
 * @return data no formato aaaammdd
 */
public static String converterddmmaaaaParaaaammdd(String data){
    String[]aux=data.trim().split("/");
    String dia=aux[0].length()<2?"0"+aux[0]:aux[0];
    String mes=aux[1].length()<2?"0"+aux[1]:aux[1];
    return aux[2]+mes+dia;
}

```

Maria da Conceição Neves

```

/**
 * Dada a data de nascimento retorna a idade
 * @param anoMesDia data de nascimento no formato aaaammdd
 * @return idade
 */
public static int idade(String anoMesDia){
    int ano=Integer.parseInt(anoMesDia.substring(0, 4));
    int mes=Integer.parseInt(anoMesDia.substring(4,6));
    int dia=Integer.parseInt(anoMesDia.substring(6,8));
    Calendar hoje= Calendar.getInstance();
    int diaH=hoje.get(Calendar.DAY_OF_MONTH);
    int mesH=hoje.get(Calendar.MONTH)+1;
    int anoH=hoje.get(Calendar.YEAR);
    if(mesH>mes || mesH==mes && diaH>=dia){
        return anoH-ano;
    }
    return anoH-ano-1;
}

```

Maria da Conceição Neves



```

package testespackage;
import gameland.Utilitarios.*;
public class TestesUtilitarios {
    /**
     *Teste do método verificaSeTemApelido
     * @param nome – nome completo
     * @param apelido – apelido a procurar
     * @param resp – resposta esperada
     * @return – SUCESSO ou FALHA consoante passou no teste ou falhou
     */

    public static String testeVerificaSeTemApelido(String nome,
                                                    String apelido, boolean resp){
        String res="FALHA";
        if(resp== verificaSeTemApelido( nome,apelido) {
            res= "SUCESSO";
        }
        return res;
    }
}

```

Maria da Conceição Neves

```

/**
 *Teste método apelidoELetralInicial
 * @param nome completo
 * @param resp – resposta esperada
 * @return – SUCESSO ou FALHA consoante passou no teste ou falhou
 */

public static String testeApelidoELetralInicial(String nome, String resposta) {
    String res="FALHA"
    if (resposta.equals(apelidoELetralInicial(nome))) {
        res= "SUCESSO";
    }
    return res;
}

```

Métodos de teste para  
cada um dos métodos  
Deverão ser implementados

Maria da Conceição Neves

```

public class TesteUtilitarios {

    public static void main(String[] args) {

        System.out.println("\n\nTESTES ASSOCIADOS AO MÉTODO verificaSeTemApelido() ");

        System.out.print("\nTESTAR 'José Costa e Silva ' contém 'Costa' ");
        System.out.println("..." + testeVerificaSeContemApelido ("José Costa e Silva ", " Costa",
            true));

        System.out.print("\nTESTAR 'José Costa e Silva ' não contém 'Gomes' ");
        System.out.println("..." + testeVerificaSeContemApelido ("José Costa e Silva ",
            " Gomes", false));

        System.out.println("\n\nTESTES ASSOCIADOS AO MÉTODO apelidoELeتراInicial() ");

        System.out.print("\nTESTAR 'José Costa e Silva retorna 'Silva J.' ");
        System.out.println("..." + testeApelidoELeتراInicial("José Costa e Silva", "Silva J." ));
    }
}

```

Pretende-se que de todos  
resulte SUCESSO

Maria da Conceição Neves

#### OUTPUT

```

TESTES ASSOCIADOS AO MÉTODO verificaSeTemApelido()

TESTAR 'José Costa e Silva ' contém 'Costa' ...SUCESSO

TESTAR 'José Costa e Silva ' não contém 'Gomes' ...SUCESSO

TESTES ASSOCIADOS AO MÉTODO apelidoELeتراInicial()

TESTAR 'José Costa e Silva retorna 'Silva J.' ...SUCESSO

```

Pretende-se que de todos  
resulte SUCESSO

Maria da Conceição Neves

## Registo de Erros - Log de Erros

- Pretende-se que as leituras de informação de ficheiros de texto sejam validadas e em caso de informação errónea esta não deve ser considerada, **mas deve ser registada num ficheiro de registo de erros para que o utilizador possa analisar.**
- Foi criada uma classe LogErros que inclui funcionalidades associadas à:
  - Criar ficheiro de log de erros
  - Registar erro
  - Fechar ficheiro de log de erros

Maria da Conceição Neves

- Foram adicionadas as seguintes constantes à classe GameLand:

```
private final static String FILE_LOG_ERROS_INSCRICOES =
                                "ErrosInscricoes.txt";
private final static String FILE_LOG_ERROS_INSCRICOES= "ErrosInscricoes.txt";
```

- Na invocação de métodos de leitura de ficheiros deve-se informar o utilizador para consultar o log erros (Exemplo)

```
nParticipantes = lerInfoFicheiroComRegisterros(nomeFich,
                                                participantes, nParticipantes);
System.out.println(
    " Confirme a existência de possíveis erros na leitura da informação."
    + "\nConsulte o ficheiro de registo de erros "
    + FILE_LOG_ERROS_INSCRICOES);
```

Maria da Conceição Neves

```

/**
 * Carrega informação dos participantes para memória a partir de ficheiro de
 * texto
 * @param nomeFich - nome do ficheiro que contem info dos participantes
 * @param participantes - matriz de strings para guardar a info de participantes
 * @return o número de participantes inseridos na matriz
 * @throws FileNotFoundException
 */
private static int lerInfoFicheiroComRegistoErros(String nomeFich, String[][] participantes,
    int nPartic) throws FileNotFoundException {
    // Referência para o File Log de Erros Deputados
    Formatter logErros= LogErros.criarEAbriFileLogErros(FILE_LOG_ERROS_INSCRICOES);

    Scanner flnput = new Scanner(new File(nomeFich));
    int nElsInic= nPartic;
    while (flnput.hasNext() && nPartic < MAX_PARTICIPANTES{
        String linha = flnput.nextLine();
        // Se linha não está em branco trata-a
        if (linha.length() > 0) {
            nPartic = guardarDadosComRegistoErros(linha, participantes, nPartic ,logErros);
        }
    }
    flnput.close();
    if( nPartic-nElsInic !=3){
        nPartic= nElsInic;
        LogErros.registarErro(logErros, " ", "O número de elementos da equipa está incorreto");
    }
    LogErros.fecharFileLogErros(logErros);
    return nPartic; }

```

**Melhorar 1**

Maria da Conceição Neves

```

/**
 * Accede à informação de uma linha do ficheiro e guarda na estrutura dados participantes
 * @param linha - String com o conteúdo de uma linha do ficheiro com info de um participante
 * @param participantes - matriz de strings com a informação dos participantes
 * @param nParticip- número de participantes existentes na matriz
 * @param logErros - objeto Formatter associado ao ficheiro de registo de erros
 * @return o novo número de participantes
 */
private static int guardarDadosComRegistoErros(String linha, String[][] participantes, int nParticip, Formatter logErros) {
    String[] temp = linha.split(";");
    if (temp.length != N_CAMPOS_INFO) {
        LogErros.registarErro(logErros, linha, "Linha incorreta, nº campos incorretos");
    } else {
        String num = temp[0].trim();
        int pos= pesquisarElemento(num, nParticip, participantes);
        if ( pos!= -1) {
            LogErros.registarErro(logErros, linha, "Linha incorreta porque email já existe");
        } else {
            for(int i=0;i< N_CAMPOS_INFO;i++){
                participantes[nParticip][i] = temp[i].trim();
            }
        }
    }
    return nParticip;
}

```

**Melhorar 1**

Maria da Conceição Neves

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Formatter;
import java.util.Scanner;
/**
 * Métodos utilitários associados à criação e escrita de informação num
 * ficheiro de registo de erros
 * @author mcn
 */
public class LogErros {
    /**
     * Cria e abre um ficheiro de texto para escrita
     * @param nomeFile - o nome do ficheiro a criar
     * @return uma objeto de Formatter que referencia o ficheiro criado
     * @throws FileNotFoundException
     */
    public static Formatter criarEAbriFileLogErros(String nomeFile) throws
    FileNotFoundException {
        Formatter outputLogErros = new Formatter(new File(nomeFile));
        return outputLogErros;
    }
}

```

Maria da Conceição Neves

```

/**
 * Registrar um erro numa linha do ficheiro
 * @param outputLogErros - referência para o ficheiro de erros
 * @param linha - linha onde foi identificado erro
 * @param tipoErro - tipo de erro identificado nessa linha
 */
public static void registarErro(Formatter outputLogErros, String linha, String tipoErro) {
    outputLogErros.format("%s---%s%n", linha, tipoErro);
}

/**
 * Fechar o ficheiro de registo de erros
 * @param outputLogErros - referência para o ficheiro de erros
 */
public static void fecharFileLogErros(Formatter outputLogErros) {
    outputLogErros.close();
}
}

```

Maria da Conceição Neves