Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2º Teste**

**Introdução à Programação**

**12/12/2017**

Duração 2H

**Instruções importantes**:

Responda a cada pergunta no espaço atribuído para o efeito.

Verifique que **todas** as folhas estão identificadas com o seu nome e número. Antes de começar a resolver, leia o enunciado do **princípio até ao fim**. Não é permitido consultar quaisquer elementos para além deste enunciado. Qualquer tentativa de fraude comprovada acarretará a reprovação na disciplina.

Nome:

Número de aluno:

Número de páginas entregues:

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Considere o seguinte enunciado do problema, o qual será usado nos grupos I, II e III: **Objectivo**:

Gerir os colaboradores nos projetos científicos partilhados por cientistas no laboratório de investigação X.

**Descrição**:

O laboratório X tem vários projetos propostos por cientistas, nos quais necessitam de colaboradores. Cada cientista tem um *email* único que o identifica e pode propor projetos no laboratório. Cada colaborador tem igualmente um *email* único que o identifica.

Cada projeto é identificado por um número único, e caracterizado por uma descrição, pelo *email* do seu proponente, a data de início e fim, e pela quantidade de colaboradores necessários no projeto, assim como os seus *emails*.

Note que um cientista pode ser colaborador, e vice-versa. Mas não pode ser simultaneamente proponente e colaborador no mesmo projeto.

É sempre possível adicionar projetos ao laboratório. Além disso também se pode atribuir/associar colaboradores a projetos, sempre e quando ainda estejam em falta.

É possível também retirar todos os projetos cuja data de fim é menor a uma dada data. Para além disso, é sempre possível consultar os projetos de um dado cientista.

**Interacção com o utilizador**:

Interpretador de comandos, onde é permitido adicionar/registar um projeto de um cientista já existente ou de um novo ("RP"), associar um colaborador a um dado projeto ("ACP"), consultar um dado projeto ("CP"), listar todos os projetos dum dado cientista ("LP"), retirar e listar todos os projetos que finalizaram antes de uma dada data ("RPD") e sair da aplicação ("S").

Para a resolução do problema, foram definidas 7 classes: Main, Aplicacao, ColProjetos, IteradorProjetos, IteradorString, Projeto e Data.

As classes Aplicacao, IteradorString e Data estão definidas neste enunciado, não sendo necessário implementá-las (assuma que já estão implementadas). As classes ColProjetos e Projeto serão implementadas neste teste, enquanto que a classe IteradorProjetos já está parcialmente implementada, tendo apenas que a complementar.

---------------------------------------------------Classe Aplicacao---------------------------

**public class** Aplicacao {

/\* Representa o sistema a implementar \*/

…

/\* Regista o projeto proposto pelo cientista dado,

\* caso não exista um projeto com esse número.

\*@**param** numero número do projeto

\*@**param** descricao descricao do projeto

\*@**param** email email do proponente (cientísta)

\*@**param** dataI data início do projeto

\*@**param** dataF data fim do projeto

\*@**param** colaboradores número de colaboradores necessários no projeto

\* @**pre** desc != null && email!=null && dataI!= null && dataF != null &&

\* dataI.eValida() && dataF.eValida() && dataI.compareTo(dataF) < 0 && colaboradores > 0 \*@**return** true, se sucedeu em registar o projeto; false, caso contrário

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*/

**public boolean** registaProjeto(int numero,String descricao, String email, Data dataI, Data dataF, int colaboradores) { … }

/\* Consulta o projeto dado um número.

\*@**param** numero número do projeto

\*@**return** o projeto, caso exista um projeto com o número dado; null, caso contrário \*/

**public** Projeto consultaProjeto(int numero) { … }

/\* Consulta o email do proponente do projeto com o número dado.

\*@**param** numero número do projeto

\* @**pre** consultaProjeto(numero)!=null

\*@**return** o email do proponente

\*/

**public** String proponenteProjeto(int numero) { … }

/\* Associa o colaborador dado ao projeto com o número dado,

\* caso exista ainda vaga no projeto. Caso contrário não faz nada e retorna false. \*@**param** numero número do projeto

\*@**param** email email do colaborador

\* @**pre** consultaProjeto(numero) != null && !proponenteProjeto(numero).equals(email) \*@**return** true, caso se tenha feito a associação; false, caso contrário

\*/

**public** boolean associaProjeto(int numero, String email) { … }

/\* Remove os projetos cuja data fim seja menor que a data dada.

\* E retorna um iterador com os projetos removidos.

\*@**param** data data

\* @**pre** data != null && data.eValida()

\*@**return** iterador dos projetos, caso tenha removido algum; null, caso contrário \*/

**public** I**teradorProjetos** removeProjetos(Data data) { … }

/\* Retorna um iterador com os projetos do cientista com o email dado.

\*@**param** email email do cientista

\* @**pre** email!= null

\*@**return** iterador dos projetos, caso exista algum; null, caso contrário

\*/

**public** I**teradorProjetos** iteradorProjeto(String email) { … }

/\* Retorna um iterador com os projetos que ainda tem vaga para colaboradores. \*@**return** iterador dos projetos, caso exista algum; null, caso contrário

\*/

**public** I**teradorProjetos** iteradorProjeto() { … }

}

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

---------------------------------------------------Classe Data---------------------------

**public class** Data {

/\* Representa uma data com dia, mes e ano \*/

…

/\* Cria uma data apartir da string (“dd-mm-aaaa”) dada.

\*@**param** data string com uma data no formato “dd-mm-aaaa”

\*@**pre** data!= null

\*/

**public** Data(String data) { … }

/\* Indica se a data é válida.

\* @**return** true, se data válida; false, caso contrário

\*/

**public** boolean eValida() { … }

/\* Compara a data com a data dada..

\*@**param** outra uma data válida

\*@**pre** outra!= null && outra.eValida()

\* @**return** 1, se a data é mais recente que a outra; 0 se são iguais; -1, no outro caso \*/

**public int** compareTo(Data outra) { … }

….

}

---------------------------------------------------Classe IteradorString--------------------------- **public class** IteradorString {

/\* Representa um iterador para uma coleção de Strings \*/

/\*\*Cria um iterador para todos osstrings no vector dado.

\*@**param** vector vector de String

\*@**param** numElem número de elementos no vector

\* @**pre** numElem >0

\*@**return** iterador de todos os strings

\*/

**public** IteradorString(String [] vector, int numElem) {...}

/\*\* Indica se há mais strings para iterar

\* **@return** – true, se houver, false caso contrário

\*/

**public** boolean temSeguinte() { … }

/\*\*Consulta a String seguinte

\* **@pre** temSeguinte()

\* **@return** – a seguinte String

\*/

**public** String seguinte() { … }

}

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Grupo I**

Este exercício visa a construção de uma classe *Projeto* que representa um projeto, com o seu número, *email* de proponente, breve descrição, data de início e fim, e os colaboradores necessários e já associados, incluindo os seus *emails*. Considerando que esta classe tem como variáveis de instância:

**private int** numero;

**private** String emailProponente;

**private** String descricao;

**private int** numColaboradoresNecessarios;

**private** Data dataI;

**private** Data dataF;

**private int** numColaboradoresAssociados;

**private** String [] emailsColaboradores;

e o constructor tem o seguinte assinatura:

**public** Projeto(**int** numero, String descricao, String emailProponente, **int** numColabNec, Data dI, Data dF){ ... }

e as seguintes funcionalidades:

1. Consulta o *email* do proponente;

2. Indica se existem vagas para colaboradores;

3. Indica se o projeto já finalizou, dando uma data (ou seja a data fim do projeto é menor, mais antiga, que a data dada);

4. Acrescenta um colaborador com o *email* dado ao projeto, caso esse *email* não seja igual ao proponente. Assuma que o projeto ainda tem vaga. Este método deve retornar um booleano indicando se foi acrescentado ou não.

5. Indica se há colaboradores já associados ao projeto;

6. Retorna o iterador dos *emails* dos colaboradores. Assuma que já há colaboradores associados;

7. Consulta o número do projeto.

**Implemente as funcionalidades e o constructor** da classe Projeto*.* Não esqueça colocar o comentário, assinatura e corpo dos métodos públicos pedidos **(7 valores)**.

| //retorna o email do Proponente.  public String getPropEmail(){return emailProponente;}  //indica se existem vagas para colaboradores.  public boolean temVagas() {return numColaboradoresNecessarios == numColaboradoresAssociados;}  //indica se o projeto já acabou.  public boolean projAcabou(int data){return data > dataF; }  //acrescenta um colaborador ao projeto  //@pre: temVagas() ==true;  public boolean addColab(String email){boolean added = false;  if (email !=emailProponente ){  emailsColaboradores[numColaboradoresAssociados++]=email;  added = true;}  return added;}  //indica se ha colaboradores ja associados  public boolean temColab(){return numColaboradoesAssociados != 0;}  // retorna iterador dos emails dos colaboradores  //@pre: temColab()==true;  public IteradorProjetos iteradorEmail(){return new IteratorString(vector, numElem, email);}  //consulta o numero do projeto.  public int projNum(){return numero;} |
| --- |

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Grupo II**

Este exercício pretende que seja implementado os 2 constructores da classe *IteradorProjetos* que representa um iterador para a colecção de projetos. Implemente os constructores **(2,5 valores)**:

**public class** IteradorProjetos {

/\* Representa um iterador para colecção de projetos \*/

/\* Constantes e Variáveis de instância \*/

**private** Projeto[] elementos;

**private int** numElementos;

**private int** current;

/\*\*Cria um iterador para todos os projetos no vector dado.

\*@**param** vector vector de projetos

\*@**param** numElem número de elementos no vector

\* @**pre** numElem >0

\*@**return** iterador de todos os projetos

\*/

**public** IteradorProjetos(Projeto [] vector, int numElem) {

| this.elementos=vetor;  this.numElementos = numElem;  current = 0; |
| --- |

}

/\*\*Cria um iterador para todos os projetos no vector dado, cujo proponente é o email dado. \*@**param** vector vector de projetos

\*@**param** numElem número de elementos no vector

\*@**param** email email do proponente

\* @**pre **\*@**return** iterador de todos os projetos

\*/

**public** IteradorProjetos(Projeto [] vector, int numElem, String email) {

| this.elementos = vector;  this.numElementos=numElem;  this.email = email  current = 0; |
| --- |
|  |

}

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\*\* Indica se há mais projetos para iterar

\* **@return** – true, se houver, false caso contrário

\*/

**public** boolean temSeguinte() {

**return** current<numElementos;

}

/\*\*Consulta o Projeto seguinte

\* **@pre **\* **@return** – o seguinte projeto

\*/

**public** Projeto seguinte() {

**return** elementos[current++];

**}**

**}**

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Grupo III**

Este exercício visa a construção de uma classe *ColProjetos* que representa uma colecção - à partida não limitada - de projetos propostos no laboratório.

Implemente a classe **(8 valores)**:

**public class** ColProjetos {

/\*\* Representa uma colecção de projetos \*/

/\* Constantes e Variáveis de instância \*/

| private Projeto[] projs;  private int count; |
| --- |

/\*\* Cria uma colecção de zero projetos, onde a lotação prevista é a indicada.

\* @**param** previstaLotacao número previsto de projetos, à partida

\* @**pre** lotacaoPrevista >0

\*/

**public** ColProjetos(**int** lotacaoPrevista) {

| projs = new Projeto[lotação Prevista];  count = 0; |
| --- |

}

/\*\* Consulta do número de projetos na colecção

\* **@return** – número de projetos

\*/

**public int** numeroProjetos() {

| return count; |
| --- |

}

/\*\* Indica se existe um projeto com o número dado

\* **@param** – id – número do projeto

\* **@return** – true, se existe e false caso contrário

\*/

**public boolean** existe(**int** id) {

| boolean ex = false  for (int i = 0; i < projs.length; i++)  if (projs[i].getId == id)  ex = true;  return ex; |
| --- |

}

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\*\* Adiciona o projeto dado à colecção, caso não exista um projeto com o mesmo número na colecção. Se necessário, a colecção deve aumentar a sua capacidade.

\* **@param** – elem projeto a adicionar

\* **@pre** elem != null

\* **@return** – true, se adicionar, false caso contrário.

\*/

**public boolean** adiciona(Projeto elem) {

| boolean add = true;  if( isFull())  reSize();  for(int i = 0; i<projs.length; i++)  if (projs[i].getId== elem.getId)  add = false;  else{projs[count++] = elem;}  return add; |
| --- |

}

/\*\* Devolve o projeto com o número dado, caso exista. Se não existir devolve null

\* **@param** – id – número do projeto

\* **@return** – o projeto, se existir; null caso contrário

\*/

**public** Projeto projeto(**int** id) {

| int i = 0;  boolean found = false;  while(i <count && !found ){  if (projs[i].getId == id)  found = true;  i++;}  if (found == true)  return projs[i-1];  else  return projs[count+1]; |
| --- |

}

/\*\*Indica se um dado cientista tem pelo menos um projeto

\* **@param** – email – email do cientista proponente

\* **@return** – true, se existir; false caso contrário

\*/

**public** boolean temProjeto(**String** email) {...}

/\*\* Retorna todos os projetos.

\* @**pre** numeroProjetos()>0

\*@**return** iterador dos projetos

\*/

**public** IteradorProjetos iterador() {

**return** new IteradorProjetos( ); }

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\*\* Retorna um iterador com os projetos de um dado cientista proponente.

\*@**param** email email do proponente

\* @**pre** email!= null && temProjetos(email)

\*@**return** iterador dos projetos

\*/

**public** IteradorProjetos iterador(String email) {

**return** new IteradorProjetos( ); }

/\*\* Remove os projetos cuja data fim seja menor que a data dada.

\* Retorna um iterador com os projetos removidos.

\* @**param** data data

\* @**pre** data != null && data.isValid()

\*@**return** iterador dos projetos, caso tenha removido algum; null, caso contrário

\*/

**public** IteradorProjetos remove(Data data) {

|  |
| --- |

}

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/\* Métodos privados \*/

|  |
| --- |

**}**

Número: \_\_\_\_\_\_\_\_\_\_\_ Nome: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Grupo IV.**

Considere uma classe com uma variável de instância vec que contém um vector de inteiros de comprimento pelo menos 1, totalmente preenchido.

Implemente o método temRepetidos que indica se existem elementos repetidos no vector.

Exemplos:

- Se vec = [4,6,7,3,4,1,8] então temRepetidos() é true

- Se vec = [5,6,7,3,1,4,8] então temRepetidos() é false

- Se vec = [5,3,-4,-3,-3,-8,2,1] então temRepetidos() é true

- Se vec = [1] então temRepetidos() é false

Implemente o método (2,5 **valores**):

**public boolean** temRepetidos() {

| boolean find = false;  for(int i = 0 ; i < vec.length-1 && !find ; i++){  for(int j = i + 1 ; j < vec.length && !find ; j++){  if(vec[i] == vec[j])  find = true;  }  }  return find; |
| --- |
|  |
|  |

}