2º Teste de "Introdução à Programação" (2013/14 – 1º Semestre)

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**.
- **Pode** usar caneta ou lápis.
- Não é permitido consultar quaisquer elementos para além deste enunciado.
- Qualquer tentativa de fraude comprovada acarretará a reprovação na disciplina.
- Não é permitido sair da sala antes que o teste termine.

Número do Aluno: _____ Nome do Aluno: _____

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

Objectivo:

Gerir os empréstimos de livros de uma biblioteca.

Descrição:

A biblioteca tem um conjunto de no máximo 500 livros. Cada livro pode ter até 5 exemplares. Nesta biblioteca é possível pedir emprestado um exemplar de um dado livro.

Um empréstimo é realizado numa dada data sobre um dado exemplar de um livro por um dado leitor da biblioteca. Cada empréstimo poderá ter uma duração máxima de 15 dias. Em caso de atraso na devolução do exemplar, o leitor deve pagar uma multa de 1.5 Euros por dia em falta.

Um livro na biblioteca está identificado pelo seu ISBN (valor inteiro positivo). Nos casos em que há vários exemplares do mesmo livro, todos esses exemplares possuem o mesmo ISBN. No entanto cada exemplar tem um identificador único que é a cota (valor inteiro positivo).

Cada leitor é identificado pelo seu número de leitor (valor inteiro positivo).

Interação com o utilizador:

Interpretador de comandos, onde é permitido registar a requisição de um empréstimo ("RE"), a devolução do exemplar do livro requisitado ("DE"), listar todos os empréstimos ("LE") e gravar no ficheiro "emprestimos.txt" todos os empréstimos em atraso na biblioteca ("LEA") e sair da aplicação ("S").

Para a resolução dos grupos I, II e III, pode usar a classe *Data*, que já se encontra implementada, e apresenta os seguintes métodos públicos:

```
/* Valida se ano, mês e dia corresponde a uma data válida
 * @param ano – ano
 * @param mes – mês
 * @param dia – dia
 * @pre ano>0 && mes>0 && mes<=12 && dia>0 && dia>=31)
 */
public static boolean dataValida(int ano, int mes, int dia){…}

/* Cria uma instância da classe Data
 * @param ano – ano da data
 * @param mes – mes da data
 * @param dia – dia da data
 * @pre Data.dataValida(ano,mes,dia)
 */
public Data(int ano, int mes, int dia){…}

/** Consulta o ano da data
 * @return – o valor do ano
 */
public int ano(){…}

/** Consulta o mês da data
 * @return – o valor do mês
 */
public int mes(){…}

/** Consulta o dia da data
 * @return – o valor do dia
 */
public int dia(){…}

/* Compara duas datas (this e outra)
 * @return 0 se this e outra são iguais;
 *         valor menor que 0, se this é uma data anterior a outra;
 *         valor maior que 0 se this é uma data posterior a outra.
 * @pre outra != null
 */
public int compareTo(Data outra){…}
/**
 * Consulta o número de dias que existe entre a data this e a data outra
 * @param outra – uma data
 * @pre outra != null && this.compareTo(outra)<=0
 */
public int quantosDias(Data outra){…}

/**
 * Acrescenta à data um dado número de dias
 * @param dias – dias a acrescentar
 * @pre dias>=0
 */
public void acrescentaDias(int dias){…}
```

Número do Aluno: _____ Nome do Aluno: _____

Grupo I

Este exercício visa a construção de uma classe *Emprestimo* que representa um empréstimo de um exemplar de um livro realizado na biblioteca. Cada objecto desta classe (um empréstimo) caracteriza-se pelo identificador do livro (ISBN) cujo exemplar está a ser requisitado, a cota do exemplar, o número de leitor que está a requisitar e a data em que foi realizada a requisição.
Complete a seguinte implementação da classe *Emprestimo* **(5 valores)**:

```
public class Emprestimo{
    /** Constantes  e Variáveis de instância **/
```

```
     /**Constructor
      * @param isbn - ISBN do exemplar do livro
      * @param cota - cota do exemplar do livro emprestado
      * @param leitor - número de leitor
      * @param dataReq - data de realização do empréstimo
      * @param dias - número de dias permitidos para o empréstimo
      * @pre isbn>0 && leitor>0 && dias>0 && dataReq != null
      */
     public Emprestimo(int isbn, int cota, int leitor, Data dataReq, int dias){
```

```
     }
     /* Consulta o ISBN do livro cujo exemplar foi emprestado
      * @return – ISBN do livro
      */
     public int isbn(){
```

```
     }
     /* Consulta a cota do exemplar que foi emprestado
      * @return – cota do exemplar do livro
      */
     public int cota(){
```

```
     }
     /* Consulta o número do leitor que realizou o empréstimo
      * @return – número de leitor
      */
     public int leitor(){
```

```
     }
     /* Indica o número de dias em atraso do empréstimo
      * @param hoje - data actual
      * @return – número de dias em atraso
      * @pre hoje != null
      */
     public int diasEmAtraso(Data hoje){
```

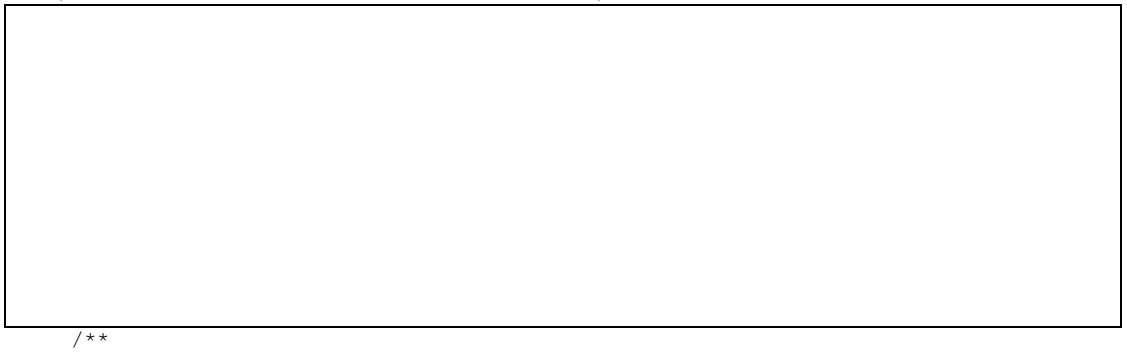
```
     }
}
```

Número do Aluno: _____ Nome do Aluno: _____

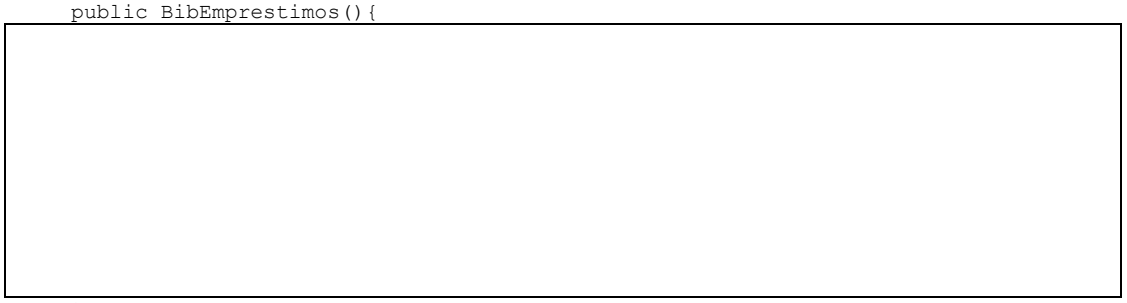
Grupo II

Este exercício visa a construção de uma classe *BibEmprestimos* que gere os empréstimos pendentes de uma biblioteca. Cada objecto desta classe caracteriza-se por um conjunto de empréstimos e o valor em caixa resultante da cobrança de empréstimos em atraso. Note que no conjunto de empréstimos pendentes, só pode existir um empréstimo por cada exemplar de livro. Complete a seguinte implementação da classe *BibEmprestimos* **(8 valores)**:

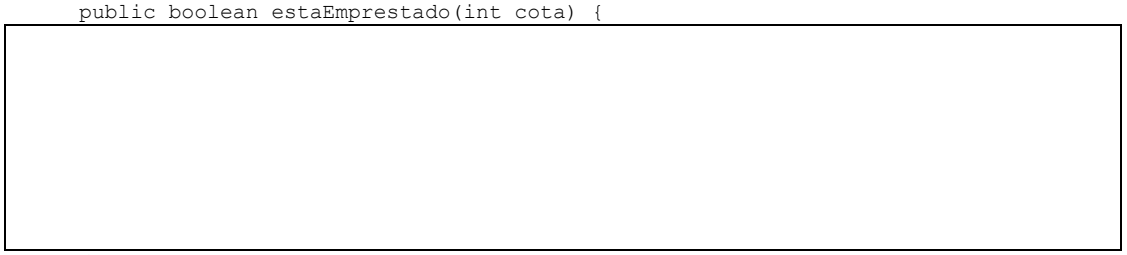
```
public class BibEmprestimos{
    /** Constantes e Variáveis de instância **/
```



```
    /**
     * Constructor – Conjunto de empréstimos vazios e valor em caixa 0 Euros
     * Não esqueça que a biblioteca tem um conjunto de no máximo 500 livros
     * e, que cada livro pode ter até 5 exemplares.
     */
    public BibEmprestimos(){
```



```
    }
    /* Indica se existe um empréstimo para um dado exemplar.
     * @param cota - cota do exemplar a emprestar
     * @return - true, caso exista; false, caso contrário
     */
     public boolean estaEmprestado(int cota) {
```

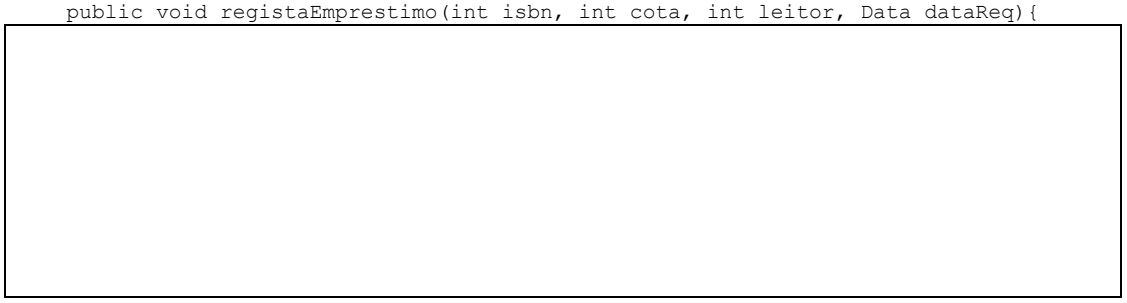


```
     }
    /* Consulta o número de empréstimos existentes para um dado livro
     * @param isbn - ISBN do livro
     * @return – número de empréstimos existentes
     */
    public int quantosExemplares(int isbn){
```

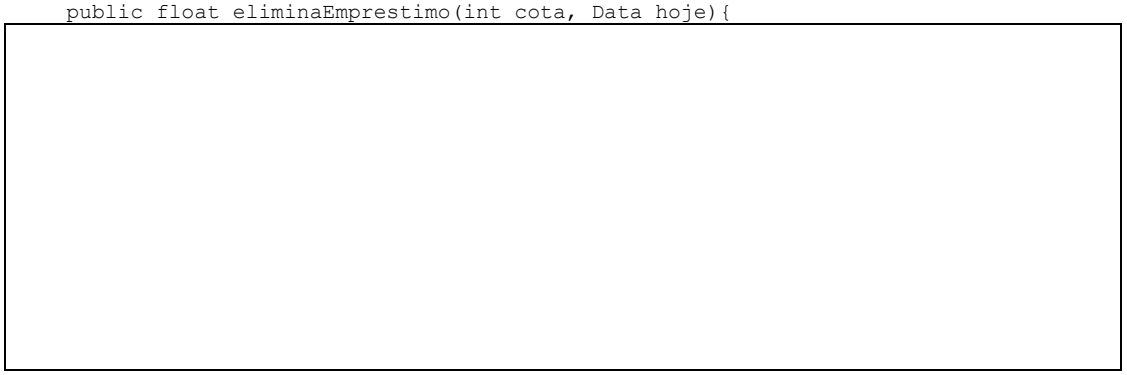
```
     }
```

Número do Aluno: _____ Nome do Aluno: _____

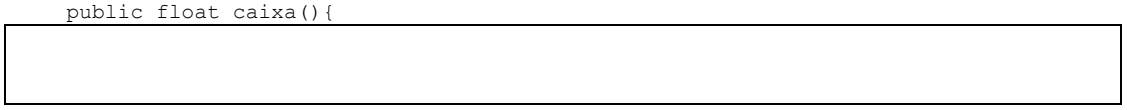
```
/* Regista um novo empréstimo na biblioteca. Note que o registo do empréstimo tem
 * sempre sucesso e a duração do empréstimo é sempre de 15 dias.
 * @param isbn - ISBN do exemplar do livro
 * @param cota - cota do exemplar a emprestar
 * @param leitor - número de leitor
 * @param dataReq - data de realização do empréstimo
 * @pre !this.estaEmprestado(cota) && this.quantosExemplares(isbn) < 5
         cota>0 && isbn>0 && leitor>0 && dataReq != null
 */
public void registaEmprestimo(int isbn, int cota, int leitor, Data dataReq){
```



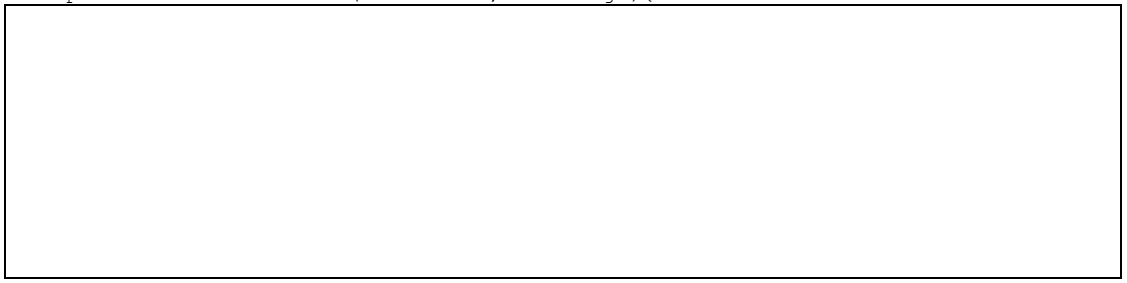
```
}
/* Devolve o valor a pagar; atualiza valor em caixa; e elimina o empréstimo
 * do exemplar. Note que só se paga se o empréstimo estiver em atraso.
 * O valor a pagar por cada dia em atraso é de 1,5 Euros.
 * @param cota - cota do exemplar do livro
 * @param hoje – data da entrega do exemplar
 * @return - valor a pagar em Euros
 * @pre this.estaEmprestado(cota) && hoje != null
 */
public float eliminaEmprestimo(int cota, Data hoje){
```



```
}
/* Consulta o valor em caixa
 * @return – valor em Euros na caixa
 */
public float caixa(){
```



```
}
/* Indica se um dado leitor está em falta, ou seja,
 * tem pelo menos um empréstimo em atraso.
 * @param leitor - número do leitor
 * @param hoje – data de hoje
 * @return – true, se tem pelo menos um empréstimo em atraso;
             false, caso contrário
 * @pre hoje != null
 */
public boolean emFalta(int leitor, Data hoje){
```



```
    }
```

Número do Aluno: _____ Nome do Aluno: _____

```
        /* Métodos para o iterador –
         * Assuma que estão implementados e funcionam como explicado nas aulas
         */

        /* Inicializa o iterador de empréstimos
         */
        public void iniciaIterador(){…}
        /* Verifica se existem mais empréstimos para iterar
         * @return – true, se existir pelo menos mais um empréstimo;
                    false, caso contrário
         */
        public boolean temSeguinte(){…}
        /* Retorna o seguinte empréstimo na iteração
         * @return um empréstimo
         * @pre this.temSeguinte()
         */
        public Emprestimo seguinte(){…}

        /* Métodos auxiliares */
```

```
}
```

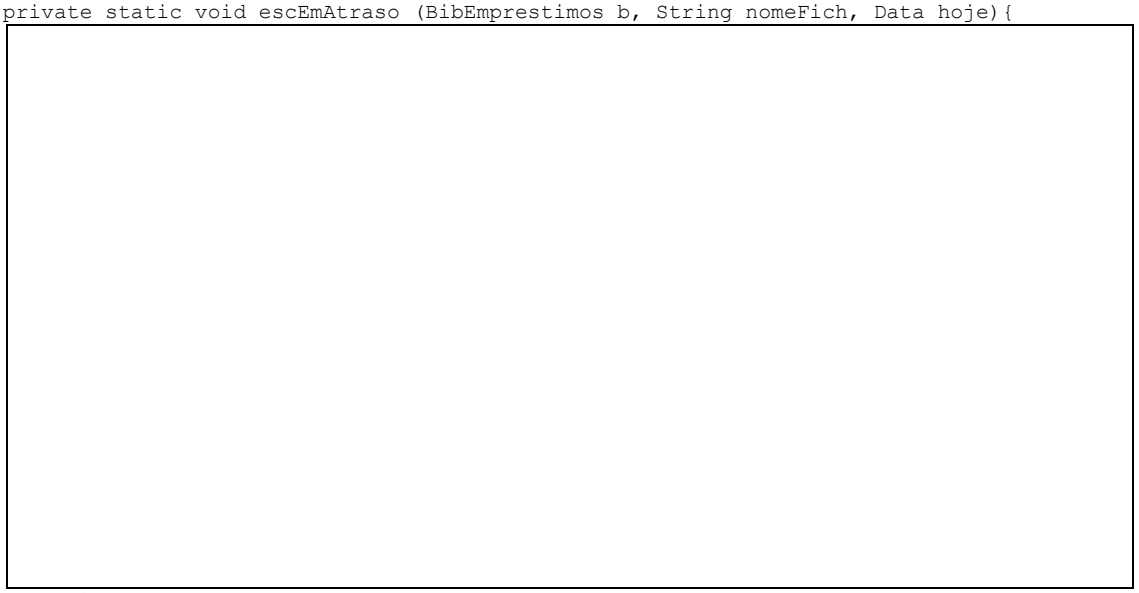
Número do Aluno: _____ Nome do Aluno: _____

Grupo III

Implemente o seguinte método auxiliar que existe na classe Main, onde é testada a classe *BibEmprestimos* (4 valores):

```
import java.io.PrintWriter;
…
public class Main {
  …
   /* Escreve no ficheiro dado, os empréstimos em atraso existentes na biblioteca.
    * Em cada linha do ficheiro deve ser escrito um empréstimo da seguinte forma:
    * isbn;cota;número leitor;dias em atraso
    * Caso exista erro na abertura do ficheiro, deve escrever na consola "ERRO"
    * @param b biblioteca onde estão todos os empréstimos pendentes
    * @param nomeFich nome do ficheiro onde se deve escrever
    * @param hoje data actual
    * @pre b != null && nomeFich!=null && hoje != null
    */

  private static void escEmAtraso (BibEmprestimos b, String nomeFich, Data hoje){
```



```
        }
        …
         /* Métodos auxiliares */
```

```
}
```

Número do Aluno: _____ Nome do Aluno: _____

Grupo IV

Este exercício visa a implementação do método *tamanhoSequencia* que se encontra na classe *SemNome*.

```
public class SemNome {
  ...
  /* Vector de valores inteiros, que se encontra preenchido
   * desde a posição 0 até contador-1
   */
  private int [] numeros; // vector de valores inteiros
  private int contador;
  ...
   /* Devolve o tamanho da maior sequência de números pares que ocorre
    * no vector numeros. Exemplos:
    * numeros = [1,7,3,9,...], contador = 4 → devolve 0 (não existe sequência)
    * numeros = [1,2,3,4,...], contador = 4 → devolve 1 (a sequência maior é "2")
    * numeros = [1,2,3,4,6,...], contador = 5 → devolve 2 (a sequência maior é "4,6")
    * numeros = [2,2,4,3,4,6,...], contador = 6 → devolve 3 (a sequência maior é "2,2,4")
    * @return tamanho da maior sequência de números pares
    */
  private int tamanhoSequencia (){...}
  ...
}
```

A. Apresente num diagrama a análise do problema, explicitando (em português) qual será a inicialização, a condição, a acção e as actualizações. (2 valores)

B. Com base no diagrama que desenvolveu, implemente o método. (1 valor)

```
private int tamanhoSequencia (){
```

```
}
```