Forma de entrega: mostrar para o professor na aula do dia **31/mar**. Os testes deverão estar rodando no Eclipse e os membros do grupo serão indagados sobre os testes implementados.

A entrega pode ser em dupla.

Observações:

- Todas as classe de programação deverão estar no pacote aula do folder src e as classes de teste deverão estar no pacote aula do folder test;
- Todas as classes de teste deverão ser finalizadas com o sufixo Test, por exemplo, SenaTest.

Exercício 1 – Programar os seguintes testes para a classe Sena da Figura 1.

a. Testar o construtor nas seguintes situações:

   new Sena(5), new Sena(6), new Sena(11), new Sena(12) e new Sena(13)

b. Testar se todos os números gerados no objeto Sena estão no intervalo [1,60];

c. Testar se os números do objeto Sena estão ordenados;

d. Testar se existem números repetidos no objeto Sena.

Observação: a classe Sena **não** pode ser alterada.

```
package aula;
import java.util.Arrays;
public class Sena {
  private int[] nros;

  public Sena(int quant) throws Exception{
    /* uma aposta por ter [6,12] números*/
    if( quant >= 6 && quant <= 12 ){
      nros = new int[quant];
      /* gera os números aleatórios */
      for( int i = 0; i < quant; i++ ){
        nros[i] = (int) (Math.random() * 60 + 1);
      }
      Arrays.sort(nros); /* ordena o array */
    }
    else{
      throw new Exception("Uma aposta precisa ter [6,12] números");
    }
  }

  /* verifica se o nro existe em nros */
  public boolean hasNro(int nro){
    for( int i = 0; i < nros.length; i++ ){
      if( nros[i] == nro ){
        return true;
      }
    }
    return false;
  }

  public String toString(){
    String r = ""+ nros[0];
    for( int i = 1; i < nros.length; i++ ){
      r += ";" +nros[i];
```

```
    }
    return r;
  }
}
```

Figura 1 – Código da classe Sena.

Exercício 2 – Programar os seguintes testes para a classe Cadastro da Figura 2.

a. Testar o construtor nas seguintes situações:

new Cadastro("teste.txt"), new Cadastro("") e new Cadastro(null)

b. Testar o método insert para os seguintes parâmetros:

1 e "De volta para o futuro I";

0 e "Missão impossível I";

2 e null;

3 e "".

Observação: a classe Cadastro **não** pode ser alterada.

```
public class Cadastro {
  private File file;
  private FileWriter fw;

  public Cadastro(String filename) throws IOException{
    file = new File(filename);
    fw = new FileWriter(file);
    fw.close();
  }

  public boolean insert(int idFilme, String nome) throws IOException{
    if( idFilme > 0 && nome != null && !nome.isEmpty() ){
      fw = new FileWriter(file,true);
      fw.write( idFilme +";"+ nome +"\r\n");
      fw.close();
      return true;
    }
    else{
      return false;
    }
  }

  public void imprimir() throws IOException{
    FileReader fr = new FileReader(file);
    BufferedReader br = new BufferedReader(fr);
    String linha = null;
    do{
      linha = br.readLine();
      if( linha != null ){
        System.out.println( linha );
      }
    }while( linha != null );
    br.close();
    fr.close();
  }
```

```
}
```

Figura 2 – Código da classe Cadastro.

Exercício 3 – As classes Celula (Figura 3) e Fila (Figura 4) podem ser usadas para manter um cadastro de nomes. O cadastro deve formar uma fila de nomes, de tal forma que, todo nome adicionado será colocado na final da fila e todo nome retirado estará no início da fila.

Programar os seguintes testes para as classes Celula e Fila:

a. Testar o construtor da classe célula nas seguintes situações:

new Celula("Ana"), new Celula ("") e new Celula(null)

b. Testar o método add da classe Fila;

c. Testar o método remover da classe Fila;

d. Testar o método getFinal da classe Fila.

Observação: as classes Celula e Fila **não** podem ser alteradas.

```
package aula;
public class Celula {
  private String nome;
  private Celula anterior;
  private Celula proxima;

  Celula( String nome ) throws Exception{
    if( nome != null && !nome.isEmpty() ){
      this.nome = nome;
    }
    else{
      throw new Exception("Parâmetro inválido");
    }
  }

  public void setAnterior( Celula anterior ){
    this.anterior = anterior;
  }

  public void setProxima(Celula proxima) {
    this.proxima = proxima;
  }

  public Celula getAnterior() {
    return anterior;
  }

  public Celula getProxima() {
    return proxima;
  }

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

Figura 3 – Código da classe Celula.

```
package aula;
public class Fila {
  private Celula primeira, ultima;

  public void add( String nome ) throws Exception {
    Celula celula = new Celula(nome);
    /* quando a fila está vazia */
    if( primeira == null ){
      primeira = celula;
      ultima = celula;
    }
    /* quando existe somente um ou mais elementos na fila */
    else{
      celula.setAnterior(ultima);
      ultima.setProxima(celula);
      ultima = celula;
    }
  }

  /* remove o primeiro elemento da fila */
  public String remover(){
    if( primeira != null ){
      Celula celula = primeira;
      /* fila só tem um elemento */
      if( primeira.getProxima() == null ){
        primeira = null;
        ultima = null;
      }
      else{
        primeira = primeira.getProxima();
        primeira.setAnterior(null);
      }
      return celula.getNome();
    }
    return null;
  }
```

```
  public String getFinal(){
    return ultima.getNome();
  }

  /* imprime na ordem crescente e decrescente */
  public void imprimir( boolean ascendente ){
    Celula celula;
    if( ascendente ){
      celula = primeira;
      while( celula != null ){
        System.out.println( celula.getNome() );
        celula = celula.getProxima();
      }
    }
    else{
      celula = ultima;
      while( celula != null ){
        System.out.println( celula.getNome() );
        celula = celula.getAnterior();
      }
    }
  }
}
```

Figura 4 – Código da classe Fila.

Exercício 4 – As classes No (Figura 5) e Lista (Figura 8) podem ser usadas para manter um cadastro de números inteiros no formato de uma lista duplamente encadeada. A Figura 6 mostra um exemplo de uso das classes No e Lista para manter um conjunto de números, a Figura 7 mostra o resultado.

Identificar e programar os testes das classes No e Lista.

Observação: as classes No e Lista **não** podem ser alteradas. A classe Principal não precisa ser testada ela existe apenas para você saber o significado das demais classes.

```
package aula;
public class No {
  public int valor;
  public No anterior, proximo;

  public No( int valor ){
    this.valor = valor;
  }
}
```

Figura 5 – Código da classe No.

```
package aula;
public class Principal {
  public static void main(String[] args) {
    Lista lista = new Lista();
    lista.add( 10 );
    lista.add( 17 );
    lista.add( 8 );
    lista.add( 20 );
    lista.add( 18 );
```

```
package aula;
public class Lista {
  private No inicio;

  public void add( int valor ){
    /* lista está vazia */
    if( inicio == null ){
      inicio = new No(valor);
    }
    /* lista não está vazia */
    else{
      add( inicio, valor );
    }
  }

  private void add( No atual, int valor ){
    /* checa se é para ir para a direita */
    if( atual.valor < valor ){
      /* verificar se existe nó a direita */
      if( atual.proximo == null ){
        No no = new No(valor);
```

```
    lista.add( 8 );
    lista.add( 9 );
    lista.add( 17 );
    lista.add( 20 );
    lista.add( 11 );
    System.out.println("Do menor para o maior valor:");
    lista.printLeft2Right();
    System.out.println("Do maior para o menor valor:");
    lista.printRight2Left();
  }
}
```

Figura 6 – Código da classe Principal.

```
Do menor para o maior valor:
8 < 8 < 9 < 10 < 11 < 17 < 17 < 18 < 20 < 20 <
Do maior para o menor valor:
20 > 20 > 18 > 17 > 17 > 11 > 10 > 9 > 8 > 8 >
```

Figura 7 – Resultado do código da Figura 6.

```
      /* coloca o novo nó no final da lista */
      atual.proximo = no;
      no.anterior = atual;
    }
    /* verifica se o próximo nó possui valor MENOR
     do que o que estamos querendo adicionar, desta forma,
     precisamos continuar percorrendo a lista para a direita */
    else if( atual.proximo.valor < valor ){
      add( atual.proximo, valor );
    }
    else{
      No no = new No(valor);
      /* colocar entre dois nós */
      no.proximo = atual.proximo;
      atual.proximo.anterior = no;
      atual.proximo = no;
      no.anterior = atual;
    }
  }
  /* ir para esquerda */
  else{
    /* verificar se existe nó a esquerda */
    if( atual.anterior == null ){
      No no = new No(valor);
      /* coloca o novo nó no início da lista */
      atual.anterior = no;
      no.proximo = atual;
    }
    else{
      /* continua procurando onde colocar o novo nó */
      add( atual.anterior, valor );
    }
  }
}

public void printLeft2Right(){
  /* posiciona no nó mais à esquerda */
  No atual = getMostLeft();
  /* percorre a lista enquanto existir nó a direita */
  while( atual != null ){
    System.out.print( atual.valor + " < ");
    atual = atual.proximo;
  }
  System.out.println(); /* quebra de linha */
}

public void printRight2Left(){
  /* posiciona no nó mais a direita */
  No atual = getMostRight();
  /* percorre a lista enquanto existir nó a esquerda */
  while( atual != null ){
    System.out.print( atual.valor + " > " );
    atual = atual.anterior;
  }
  System.out.println(); /* quebra de linha */
}

/* retorna o nó mais à esquerda */
private No getMostLeft(){
```

```
    No no = inicio;
    /* verifica se a lista está vazia */
    if( no != null ){
      /* repete enquanto existir nó a esquerda */
      while( no.anterior != null ){
        no = no.anterior;
      }
    }
    return no;
  }

  /* retorna o nó mais a direita */
  private No getMostRight(){
    No no = inicio;
    /* verifica se a lista está vazia */
    if( no != null ){
      /* repete enquanto existir nó a direita */
      while( no.proximo != null ){
        no = no.proximo;
      }
    }
    return no;
  }
}
```

Figura 8 – Código da classe Lista.

Exercício 5 – Programar uma classe de testes para executar os testes dos Exercícios 1 a 4. Assim como foi feito no Exemplo A da Atividade 1.