Instruções para a entrega: fazer os exercícios e mostrar para o professor na aula do dia **22/mar**. Os testes deverão ser executados no computador. A entrega pode ser em dupla. Alunos ausentes não terão a nota considerada.

Descrição:

O método `assertThat` da JUnit permite fazer comparações mais genéricas quando comparado com o método `assertEquals`, que é capaz de comparar apenas a igualdade. Para isso ele faz uso de implementações da interface `Matcher` (https://junit.org/junit4/javadoc/4.12/org/hamcrest/Matcher.html) aqui usaremos a implementação Hamcrest (https://code.google.com/archive/p/hamcrest/wikis/Tutorial.wiki). O JUnit possui uma implementação da biblioteca Hamcrest, porém ela possui um conjunto menor de métodos implementados, por este motivo, usaremos uma implementação de terceiros.

O método `assertThat` possui a assinatura `assertThat(resultado, expressão matcher).`

Instruções para criar o projeto na IDE Eclipse:

- Crie um projeto e adicione a biblioteca JUnit 4;
- No projeto crie um folder (não é source folder) de nome lib, assim como na Figura 1;
- Para fazer alguns testes precisaremos da biblioteca Hamcrest disponível em https://code.google.com/archive/p/hamcrest/downloads. Copie o arquivo hamcrest-all-1.3.jar para a pasta lib do seu projeto assim como na Figura 1;
- Para incluir a biblioteca no classpath do projeto, clique com botão direito sobre o nome do projeto e acesse Properties > Java Build Path > Libraries e na sequência clique no botão Add JARs... (Figura 2). Localize o arquivo hamcrest-all-1.3.jar na pasta lib;
- A biblioteca hamcrest precisa vir antes da JUnit no build path, então acesse a aba Order and Export, na Figura 2, para alterar a ordem.

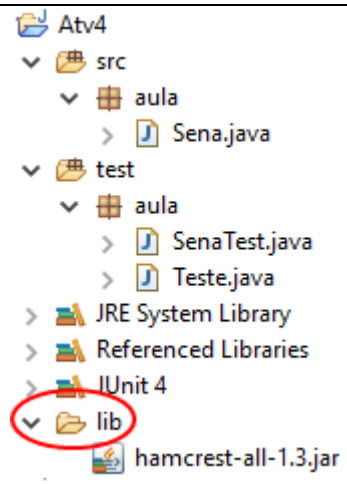

Figura 1 – Estrutura do projeto

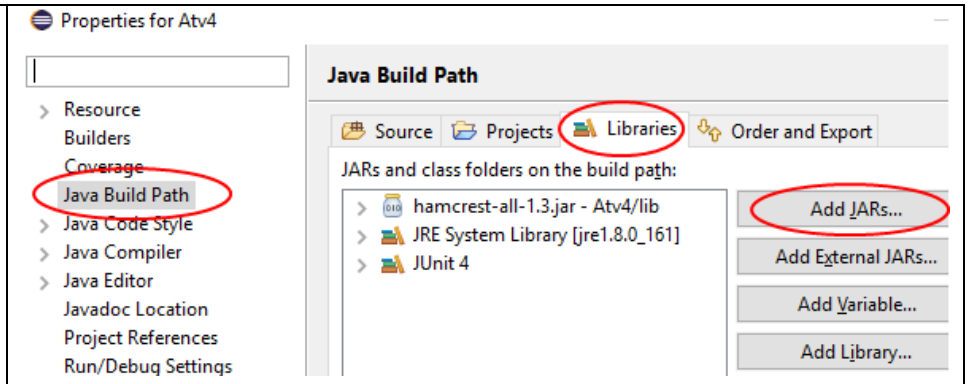

Figura 2 – Adicionar biblioteca ao projeto.

A classe de teste a seguir faz uso de alguns métodos da classe `Matchers` da biblioteca Hamcrest, para mais opções http://hamcrest.org/JavaHamcrest/javadoc/1.3/org/hamcrest/Matchers.html. Copie as classes assim como exemplificado no projeto da Figura 1.

```
package aula;

import org.junit.Test;

import static org.hamcrest.CoreMatchers.*;
import static org.hamcrest.MatcherAssert.assertThat;

import static org.hamcrest.Matchers.greaterThan;
import static org.hamcrest.Matchers.either;
import static org.hamcrest.Matchers.isIn;
import static org.hamcrest.collection.IsCollectionWithSize.hasSize;

import org.hamcrest.Description;
import org.hamcrest.TypeSafeMatcher;

public class Teste {

   @Test
   public void isTest() {
      assertThat(0, is(0));
   }

   @Test
   public void notTest() {
      assertThat(2, not(1) );
   }

   @Test
   public void greaterThanTest() {
      assertThat(5, greaterThan(2) );
   }

   @Test
   public void eitherTest() {
      assertThat( (int)(Math.random()*3), either(is(0)).or(is(1)).or(is(2)) );
   }

   @Test
   public void containsStringTest() {
      assertThat("A casa é bela", containsString("é") );
   }

   @Test
   public void anyOfTest1() {
      assertThat("A casa é bela", anyOf(containsString("é"), containsString("ã")) );
   }

   @Test
   public void anyOfTest2() {
      assertThat( (int)(Math.random()*3), anyOf(equalTo(0),equalTo(1),equalTo(2)) );
   }

   @Test
   public void allOfTest() {
      assertThat("A casa é bela", not( allOf(containsString("é"), containsString("ã")) ) );
   }

   @Test
   public void instanceOfTest() {
      assertThat(12.5, instanceOf(java.lang.Number.class) );
```

```
    }

    @Test
    public void hasSizeTest() {
       java.util.List<Integer> lista = java.util.Arrays.asList(5, 2, 4);
       assertThat(lista, hasSize(3));
    }

    @Test
    public void isInTest() {
       java.util.List<Integer> lista = java.util.Arrays.asList(5, 2, 4);
       assertThat(2, isIn(lista));
    }

    @Test
    public void everyItemTest() {
       java.util.List<Integer> lista = java.util.Arrays.asList(5, 2, 4);
       assertThat(lista, everyItem(greaterThan(1)));
    }

    @Test
    public void segundoCaracterTest() {
       assertThat("abcde", segundoCaracter('b'));
    }

    @Test
    public void anyCaracterTest() {
       assertThat("abcde", anyCaracter('e',4));
    }

    /* matcher para testar o 2o caracter da string */
    private TypeSafeMatcher<String> segundoCaracter(char comparacao){
       return new TypeSafeMatcher<String>(){
          protected boolean matchesSafely (String str){
             if( str.length() < 2 ) return false;
             if( str.charAt(1) != comparacao ) return false;
             return true;
          }

          @Override
          public void describeTo(Description description) {
             description.appendText("describe the error has you like more");
          }
       };
    }

    /* matcher para testar o nth caracter da string */
    private TypeSafeMatcher<String> anyCaracter(char comparacao, int index){
       return new TypeSafeMatcher<String>(){
          protected boolean matchesSafely (String str){
             if( str.length() < index + 1 ) return false;
             if( str.charAt(index) != comparacao ) return false;
             return true;
          }

          @Override
          public void describeTo(Description description) {
             description.appendText("describe the error has you like more");
          }
       };
    }
}
```

```
package aula;

import java.util.*;
```

```
public class Sena {

   public List<Integer> getSena(Integer n) throws Exception {
      if( n < 6 ) throw new Exception("Mínimo 6");
      if( n > 12 ) throw new Exception("Máximo 12");

      Integer[] lista = new Integer[n];
      for( int i = 0; i < lista.length; i++ ) {
         lista[i] = (int) (Math.random() * 60 + 1);
      }
      Arrays.sort(lista);
      return Arrays.asList(lista);
   }
}
```

Exercício – Programar na classe de testes `SenaTest` (Figura 1) os seguintes testes para o método ***getSena***:

a) getSena(6) retorna um array com 6 elementos. Use o método `hasSize` da classe `IsCollectionWithSize`;

b) getSena(10) retorna um array onde cada elemento possui valor no intervalo [1,60]. Para fazer esse teste será necessário combinar os métodos `everyItem` (classe `CoreMatchers`), `allOf`, `greaterThanOrEqualTo` e `lessThanOrEqualTo` da classe `Matchers`;

c) getSena(10) retorna um array onde todos os elementos estão ordenados. Será necessário criar um método para comparar os elementos usando uma implementação da classe abstrata `TypeSafeMatcher` (https://junit.org/junit4/javadoc/4.12/org/hamcrest/TypeSafeMatcher.html), que por usa vez herda a classe abstrata `BaseMatcher` (https://junit.org/junit4/javadoc/4.12/org/hamcrest/BaseMatcher.html) e esta implementa a interface `Matcher`;

d) getSena(10) retorna um array sem elementos duplicados. Será necessário criar um método para comparar os elementos usando uma implementação da classe abstrata `TypeSafeMatcher`;

e) getSena(null) neste caso o indicado seria usar o atributo `expected=Exception.class` na anotação `@Test`. Porém aqui será obrigatório usar uma `@Rule` para `ExpectedException`;

f) getSena(5) use uma `@Rule` para testar se a mensagem da exceção possui o texto `"Mínimo 6"`;

g) getSena(12) use uma `@Rule` para testar se a mensagem da exceção possui o texto `"Máximo 12"`;

Observação: uma dica sobre testes de exceções usando `@Rule` pode ser encontrada em https://github.com/junit-team/junit4/wiki/exception-testing