Instruções para a entrega: fazer os exercícios e mostrar para o professor na aula do dia **29/mar**. Os testes deverão ser executados no computador. A entrega pode ser em dupla. Alunos ausentes não terão a nota considerada.

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 os testes precisaremos da biblioteca mockito disponível em http://mvnrepository.com/artifact/org.mockito/mockito-all/1.10.19. Copie o arquivo mockito-all-1.10.19.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 mockito-all-1.10.19.jar na pasta lib;
- A biblioteca mockito-all-1.10.19 precisa vir antes da JUnit no build path, então acesse a aba Order and Export, na Figura 2, para alterar a ordem.

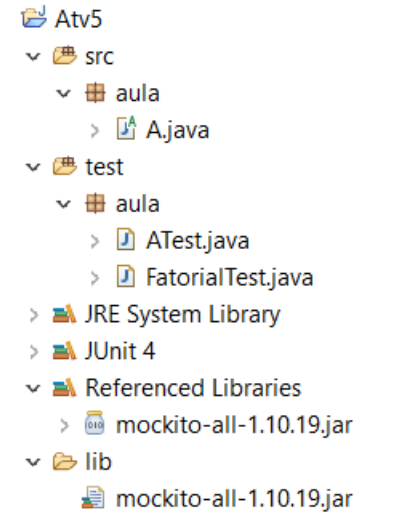

Figura 1 – Estrutura do projeto

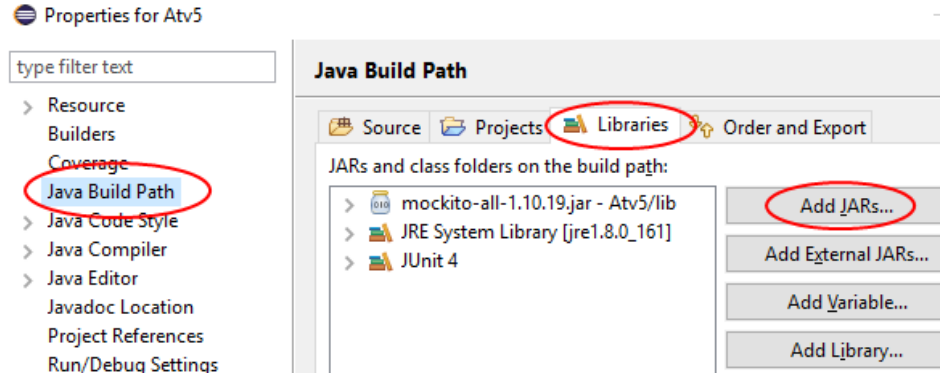

Figura 2 – Adicionar biblioteca ao projeto.

Considere a classe ***A*** nos exercícios.

```
public abstract class A {
   public long fatorial(long n) {
      if( n <= 1 ) {
         return 1;
      }
      return n * fatorial(n-1);
   }

   public abstract Object calc(Object x, Object y) throws NullPointerException, Exception;

   public void msg(String txt) {}

   public double area(double r) {
      return 2* pi() * r;
```

```
    }

    public double pi() {
        return Math.PI;
    }

    public double pow() {
        return pi() * pi();
    }

    public abstract int inc();
}
```

Exercício 1 – O Desenvolvimento Orientado a Testes (TDD) diz que os testes devem ser codificados antes dos artefatos serem implementados. Programar uma classe de testes de nome ATest para testar o método ***calc***, da classe ***A***, nas seguintes situações:

{a: 2, b: 2, esperado: 4}

{a: "x", b: "y", esperado: "xy"}

{a: null, b: "y", esperado: NullPointerException}

{a: "2", b: 2, esperado: Exception}

Observe que será necessário utilizar um framework de Mock para fazer os stubs (simulações) do comportamento do método, já que uma classe abstrata não pode ser instanciada.

Exercício 2 – O método `doThrow` pode ser utilizado para testar métodos com retorno void. Adicionar na classe ATest um teste para o método ***msg***, da classe ***A***.

Exercício 3 – O método `verify`, do pacote `org.mockito.Mockito.`*`verify`*, é usado para checar a quantidade de vezes que um método é invocado. Adicionar na classe ATest um teste para checar se ao invocar o método `area(2)` o método `pi()` é invocado exatamente 1 vez.

Exercício 4 – Adicionar na classe ATest um teste para checar se ao invocar o método `pow()` o método `pi()` é invocado exatamente 2 vez.

Exercício 5 – Adicionar na classe ATest o método a seguir e fazer as devidas alterações para que o resultado do teste seja "verde", isto é, aprovado no teste.

```
@Test
public void incTest() {
    when(a.inc()).thenReturn(1,2,3,4).thenThrow(new NullPointerException("Além do limite"));
    while( true ) {
        System.out.println( a.inc() );
    }
}
```

Observação: não é permitido alterar o bloco `while`.

Exercício 6 – Criar uma classe parametrizada de nome FatorialTest para testar o método fatorial nas seguintes situações:

{entrada: 0, esperado: 0}

{entrada: 1, esperado: 1}

{entrada: 2, esperado: 2}

{entrada: 5, esperado: 120}

Observação: não é possível instanciar uma classe abstrata.