TÉCNICO LISBOA (http://tecnico.ulisboa.pt)

Programação Avançada (https://fenix.tecnico.ulisboa.pt/disciplinas/PAva26/2016-2017/2-semestre)

Task: A tester program based on annotations

Before performing the exercise of this laboratory you should be familiar with Java annotations. Consider the example for a simple testing tool given in Oracle's technical guide on Java annotations (http://docs.oracle.com/javase/6/docs/technotes/guides/language/annotations.html). The goal is to start with the example and extend it with the following additional functionality:

The @Setup annotation will complement the @Test annotation. The @Setup contains a single attribute, that defines its name, e.g. @Setup("a"). The @Test annotation is enhanced to support a list of "setup methods" that must be executed before the test method is executed. For example, @Test({"a", "b"}) indicates that the setup methods annotated with the name "a" and "b" need to be executed, in the given order, before the test method is executed. The default value for the list of setup methods in the @Test annotation is "*", which means that all setup methods will be executed, in any order.

Consider only static methods that return void and take zero arguments for both @Setup and @Test annotations. The annotated methods can be private, protected or public.

When testing a hierarchy of annotated classes, the tests in the superclass should run first.

The following test classes provide some examples of the expected behaviour.

```
public class TestSimple {
    @Test public static void m1() { System.out.println("m1"); }
    @Test private static void m2() { System.out.println("m2"); }
    @Test private static void m3() { throw new RuntimeException("problem"); }
    public static void m4() { System.out.println("m4"); }
}

$ java RunTests TestSimple
m1
Test public static void TestSimple.m1() OK!
m2
Test private static void TestSimple.m2() OK!
Test private static void TestSimple.m3() failed
Passed: 2, Failed 1

public class TestWithSetup {
    @Setup("s1") private static void s1() { System.out.println("s1"); }
    @Setup("s2") protected static void s2() { System.out.println("s2"); }
    @Setup("s3") private static void s3() { throw new RuntimeException("fail"); }

    @Test public static void m5() { System.out.println("m5"); }
    @Test({"s2", "s1"}) private static void m6() { System.out.println("m6"); }
    @Test({"s1", "s2"}) private static void m7() { throw new RuntimeException("problem"); }
    @Test("s1") private static void m8() { System.out.println("m8"); }
    public static void m9() { System.out.println("m9"); }
}

$ java RunTests TestWithSetup
s2
s1
Test public static void TestWithSetup.m5() failed
s2
s1
m6
Test private static void TestWithSetup.m6() OK!
s1
s2
Test private static void TestWithSetup.m7() failed
s1
m8
Test private static void TestWithSetup.m8() OK!
Passed: 2, Failed 2

public class TestInheritance extends TestWithSetup {
    @Setup("s4") protected static void s5() { System.out.println("s4"); }

    @Test("*") public static void m10() { System.out.println("m10"); }
    @Test("s2") public void m11() { System.out.println("m11"); }
    @Test("s1") public static void m12() { System.out.println("m12"); }
    public static void m13() { System.out.println("m13"); }
    @Test({"s1", "s4"}) public static void m14() { System.out.println("m14"); }
}

$ java RunTests TestInheritance
s2
s1
Test public static void TestWithSetup.m5() failed
s2
s1
m6
Test private static void TestWithSetup.m6() OK!
s1
s2
Test private static void TestWithSetup.m7() failed
s1
m8
Test private static void TestWithSetup.m8() OK!
s2
s1
Test public static void TestInheritance.m10() failed
s1
```

```
m12
Test public static void TestInheritance.m12() OK!
s1
s4
m14
Test public static void TestInheritance.m14() OK!
Passed: 4, Failed 3
```