Lab 11

Part a.

Given is the simple unit testing framework code in the project SimpleTestFramework. This simple test framework will call all methods annotated with @Test.

Expand the framework, so that it also support the @Before annotation. The framework should call the method annotated with @Before before every single test method.
So the test code:

```java
@TestClass
public class MyTest {

	@Before
	public void init() {
		System.out.println("perform initialization");
	}

	@Test
	public void testMethod1() {
		System.out.println("perform test method 1");
	}

	@Test
	public void testMethod2() {
		System.out.println("perform test method 2");
	}
}
```

Should give the following output:

```
perform initialization
perform test method 1
perform initialization
perform test method 2
```

You do not need to write code that checks if there is only 1 @before method.

Part b.

In the framework of part a, add the following class with a simple assertEquals method.

```
package framework;

public class Asserts {

  public static void assertEquals(int x, int y) {
    if (x != y)
      System.out.println("Fail: result = "+x+" but expected "+y);
  }
}
```

In the application package, add the following simple Calculator:

```
public interface Calculator {
      public void reset() ;
      public int add(int newValue);
      public int subtract(int newValue);
}


public class CalculatorImpl implements Calculator {
      private int calcValue=0;

      public void reset() {
            calcValue=0;
      }

      public int add(int newValue) {
            calcValue=calcValue+newValue;
            return calcValue;
      }

      public int subtract(int newValue) {
            calcValue=calcValue-newValue;
            return calcValue;
      }
}
```

This should allow you to write simple test:

```java
package application;

import framework.Before;
import framework.Test;
import framework.TestClass;
import static framework.Asserts.*;

@TestClass
public class MyTest {
	Calculator calculator;

	@Before
	public void init() {
		calculator = new CalculatorImpl();
	}

	@Test
	public void testMethod1() {
		assertEquals(calculator.add(3),3);
		assertEquals(calculator.add(6),9);
	}

	@Test
	public void testMethod2() {
		assertEquals(calculator.add(3),3);
		assertEquals(calculator.subtract(6),-1);
	}

}
```

Part c.

Now modify the code of part b, so that the framework also allow us to **inject** classes that are annotated with @Service:

```
@Service
public class CalculatorImpl implements Calculator {
      private int calcValue=0;

      public void reset() {
            calcValue=0;
      }

      public int add(int newValue) {
            calcValue=calcValue+newValue;
            return calcValue;
      }

      public int subtract(int newValue) {
            calcValue=calcValue-newValue;
            return calcValue;
      }
}



@TestClass
public class MyTest {
      @Inject
      Calculator calculator;

      @Before
      public void init() {
            calculator.reset();
      }

      @Test
      public void testMethod1() {
            assertEquals(calculator.add(3),3);
            assertEquals(calculator.add(4),7);
      }

      @Test
      public void testMethod2() {
            assertEquals(calculator.add(3),3);
            assertEquals(calculator.subtract(6),-1);
      }

}
```