```java
package org.springframework.samples.petclinic.util;


import static org.assertj.core.api.Assertions.assertThat;
import static org.assertj.core.api.Assertions.failBecauseExceptionWasNotThrown;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Spy;
import org.mockito.runners.MockitoJUnitRunner;
import org.springframework.samples.petclinic.model.Owner;
import org.springframework.samples.petclinic.repository.OwnerRepository;
import org.springframework.samples.petclinic.service.ClinicServiceImpl;
import org.springframework.samples.petclinic.service.OwnerUtils;


@RunWith(MockitoJUnitRunner.class)
public class MockitoExampleTests {

    @Mock
    private List names;

    @Mock
    private OwnerRepository ownerRepository;

    @Spy
    private OwnerUtils ownerUtils;

    @InjectMocks
    private ClinicServiceImpl clinicService;


    @Test
    public void stupidTest() {

        //List names = mock(List.class);

        when(names.size()).thenReturn(10);

        int size = names.size();

        verify(names).size();
        assertThat(size).isEqualTo(10);
    }

    @Test
    public void anotherStupidTest() {

        //List names = mock(List.class);

        when(names.get(0)).thenReturn("Alan");
        when(names.get(1)).thenThrow(new NullPointerException());

        assertThat(names.get(0)).isEqualTo("Alan");

        try {
            names.get(1);
            failBecauseExceptionWasNotThrown(NullPointerException.class);
        } catch(NullPointerException npe) {
            assertThat(npe).isNotNull();
        }
    }

    @Test
    public void testOwnerName() {

        Owner o = new Owner();
        o.setFirstName("Alan");

```

```
78          when(ownerRepository.findById(1)).thenReturn(o);
79
80          assertThat(clinicService.findOwnerById(1).getFirstName()).isEqualTo("Alan");
81      }
82
83      @Test
84      public void testLastNameReversion() {
85
86          Owner o1 = new Owner();
87          o1.setLastName("azerty");
88
89          Owner o2 = new Owner();
90          o2.setLastName("abcdef");
91
92          Collection<Owner> c = new ArrayList<Owner>();
93          c.add(o1);
94          c.add(o2);
95
96          when(ownerRepository.findByLastName("a")).thenReturn(c);
97
98          List<Owner> owners = (List<Owner>)clinicService.findOwnerByLastName("a");
99          verify(ownerUtils).reverseLastname(c);
100
101
102         assertThat(owners.get(0).getLastName()).isEqualTo("ytreza");
103         assertThat(owners.get(1).getLastName()).isEqualTo("fedcba");
104     }
105
106
107     /**
108      * Exercise 1 : Reverse last name of owners
109      *
110      * - You need to write a Service, call it OwnerUtils for example.
111      * - Write a reverse method inside this service that reverse a string (use TDD !).
112      * - Write a reverseLastname method that loop on a Collection of Owner and reverse their lastnames.
113      * - Inject this service inside ClinicServiceImpl
114      * - use the sort method in the findOwnerByLastName method
115      * - write a test that mock the repository call and test the good reversion of your owners' lastnames
116      *       - If you declare OwnerUtils as @Mock in your test, it doesn't work, why ?
117      *       - use @Spy instead of @Mock and relaunch your test
118      *
119      * Remember that this is an exercise, in real life, we would wouldn't test ClinicService for real because it
    has no business logic.
120      * All is done in the reverse methods.
121      */
122
123 }
124
```