Behaviour Driven Mockito

```
@Test
void findByIdBddTest(){
    Speciality speciality = new Speciality();
    given(specialtyRepository.findById(1l)).willReturn(Optional.of(speciality));
    Speciality foundspeciality = service.findById(1l);
    assertThat(foundspeciality).isNotNull();
    verify(specialtyRepository).findById(anyLong());
}
```

```
@Test
    void findByIdBddTest(){
        //given -> responsible for setting up speciality object and configuring the
mock
        Speciality speciality = new Speciality();
        given(specialtyRepository.findById(1l)).willReturn(Optional.of(speciality));

        //when -> doing the test action/service is being called
        Speciality foundspeciality = service.findById(1l);

        //then -> using assertion and mockito to check
        assertThat(foundspeciality).isNotNull();
//        verify(specialtyRepository).findById(anyLong());
        //checking specialtyRepository deleteById was called once
        then(specialtyRepository).should().deleteById(1l);
        // and further there are no more interactions with specialityRepository
        then(specialtyRepository).shouldHaveNoMoreInteractions();
    }
```

```
package guru.springframework.sfgpetclinic.services.springdatajpa;

import guru.springframework.sfgpetclinic.model.Speciality;
import guru.springframework.sfgpetclinic.repositories.SpecialtyRepository;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Optional;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.BDDMockito.given;
```

```
import static org.mockito.BDDMockito.then;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
class SpecialitySDJpaServiceTest {

    @Mock
    SpecialtyRepository specialtyRepository;

    @InjectMocks
    SpecialitySDJpaService service;
    @Test
    void deleteById() {
        //given - none

        //when
        service.deleteById(1l);
        service.deleteById(1l);
        //then
//        verify(specialtyRepository,times(2)).deleteById(1l);
        then(specialtyRepository).should(times(2)).deleteById(1l);
    }

    @Test
    void deleteByIdAtLeast() {
        //given

        //when
        service.deleteById(1l);
        service.deleteById(1l);
        //then
        then(specialtyRepository).should(atLeast(1)).deleteById(1l);
//        verify(specialtyRepository,atLeast(1)).deleteById(1l);
    }

    @Test
    void deleteByIdAtMost() {
        //given

        //when
        service.deleteById(1l);
        service.deleteById(1l);
        //then
        then(specialtyRepository).should(atMost(5)).deleteById(1l);
//        verify(specialtyRepository,atMost(5)).deleteById(1l);
    }
    @Test
```

```
    void deleteByIdNever() {
        //given

        //when
        service.deleteById(1l);
        service.deleteById(1l);

        //then
//        verify(specialtyRepository,atLeast(1)).deleteById(1l);
//        verify(specialtyRepository,never()).deleteById(5l);
        then(specialtyRepository).should(atLeast(1)).deleteById(1l);
        then(specialtyRepository).should(never()).deleteById(5l);
    }

    @Test
    void testDelete() {
        //when
        service.delete(new Speciality());
        //then
        then(specialtyRepository).should().delete(any());
    }

    @Test
    void findByIdTest(){
        Speciality speciality = new Speciality();
        when(specialtyRepository.findById(1l)).thenReturn(Optional.of(speciality));
        Speciality foundspeciality = service.findById(1l);
        assertThat(foundspeciality).isNotNull();
        verify(specialtyRepository).findById(anyLong());
    }

    @Test
    void findByIdBddTest(){
        //given -> responsible for setting up speciality object and configuring the
mock
        Speciality speciality = new Speciality();
        given(specialtyRepository.findById(1l)).willReturn(Optional.of(speciality));

        //when -> doing the test action/service is being called
        Speciality foundspeciality = service.findById(1l);

        //then -> using assertion and mockito to check
        assertThat(foundspeciality).isNotNull();
//        verify(specialtyRepository).findById(anyLong());
        //checking specialtyRepository deleteById was called once
        then(specialtyRepository).should().findById(1l);
        // and further there are no more interactions with specialityRepository
```

```
//        then(specialtyRepository).should().deleteById(anyLong());
        then(specialtyRepository).shouldHaveNoMoreInteractions();
    }

    @Test
    void testDeleteByObject(){
        //given
        Speciality speciality = new Speciality();
        //when
        service.delete(speciality);
        //then
//        verify(specialtyRepository).delete(any(Speciality.class));
        then(specialtyRepository).should().delete(any(Speciality.class));
    }
}
```