Advanced Mockito

Throwing exceptions with Mockito:

- Checks how method behaves when exceptions are thrown to them.
- Use Junit 5 assertions and Mockito to generate exceptions

```
@Test
void testDoThrow(){
    //using mockito to throw exception when delete is called
    doThrow(new
RuntimeException("boom")).when(specialtyRepository).delete(any());
    //making an assertion that exception is thrown
    assertThrows(RuntimeException.class, ()->{
        specialtyRepository.delete(new Speciality());
    });
    verify(specialtyRepository).delete(any());
}

@Test
void testFindByIdThrows(){
    //given
    given(specialtyRepository.findById(1l)).willThrow(new
RuntimeException("boom"));
    //then
    assertThrows(RuntimeException.class,
            ()-> specialtyRepository.findById(1l) // when
    );
    then(specialtyRepository).should().findById(1l);
}

@Test
void testDeleteBDD(){
    // here we cannot use given directly as delete returns void
    willThrow(new
RuntimeException("boom")).given(specialtyRepository).delete(any());
    assertThrows(RuntimeException.class,()->specialtyRepository.delete(new
Speciality()));
    then(specialtyRepository).should().delete(any());
}
```

Java 8 lambda Argument Matchers:

- We gonna return a match on a specific property.

```
@Test
void testSaveLambda(){
    //given
    final String MATCH_ME = "MATCH_ME";
    Speciality speciality = new Speciality();
    speciality.setDescription(MATCH_ME);

    Speciality savedSpeciality = new Speciality();
    savedSpeciality.setId(1l);

    //need mock to only return on MATCH_ME string
    given(specialtyRepository.save(argThat(argument->
argument.getDescription().equals(MATCH_ME)))).willReturn(savedSpeciality);
    //when
    Speciality returnedSpeciality = specialtyRepository.save(speciality);
    //then
    assertThat(returnedSpeciality.getId()).isEqualTo(1l);
}
```

- argThat is ArgumentMatcher of mockito.
- ArgumentMatcher uses lambda are strict matcher so if we do not get that true then mockito will not let other things on its way to execute

E.g.

```
@Test
void testSaveLambdaNoMatch(){
    //given
    final String MATCH_ME = "MATCH_ME";
    Speciality speciality = new Speciality();
    speciality.setDescription("I am not gonna match");

    Speciality savedSpeciality = new Speciality();
    savedSpeciality.setId(1l);

    //need mock to only return on MATCH_ME string
    given(specialtyRepository.save(argThat(argument->
argument.getDescription().equals(MATCH_ME)))).willReturn(savedSpeciality);
    //when
    Speciality returnedSpeciality = specialtyRepository.save(speciality);
    //then
    assertNull(returnedSpeciality);
}
```

To let assertNull work ->
@Mock(lenient = true)

SpecialtyRepository specialtyRepository;

Assignment:

Write test for processCreationForm method using mockito:

```
package guru.springframework.sfgpetclinic.controllers;

import guru.springframework.sfgpetclinic.fauxspring.BindingResult;
import guru.springframework.sfgpetclinic.fauxspring.Model;
import guru.springframework.sfgpetclinic.fauxspring.ModelAndView;
import guru.springframework.sfgpetclinic.fauxspring.WebDataBinder;
import guru.springframework.sfgpetclinic.model.Owner;
import guru.springframework.sfgpetclinic.services.OwnerService;

import javax.validation.Valid;
import java.util.List;


public class OwnerController {
    private static final String VIEWS_OWNER_CREATE_OR_UPDATE_FORM =
"owners/createOrUpdateOwnerForm";

    private final OwnerService ownerService;

    public OwnerController(OwnerService ownerService) {
        this.ownerService = ownerService;
    }
    public String processCreationForm(@Valid Owner owner, BindingResult
result) {
        if (result.hasErrors()) {
            return VIEWS_OWNER_CREATE_OR_UPDATE_FORM;
        } else {
            Owner savedOwner =  ownerService.save(owner);
            return "redirect:/owners/" + savedOwner.getId();
        }
    }

}
```

Sol:

```
package guru.springframework.sfgpetclinic.controllers;

import guru.springframework.sfgpetclinic.fauxspring.BindingResult;
import guru.springframework.sfgpetclinic.model.Owner;
import guru.springframework.sfgpetclinic.repositories.OwnerRepository;
```

```
import guru.springframework.sfgpetclinic.services.OwnerService;
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 static org.assertj.core.api.Java6Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.BDDMockito.given;

@ExtendWith(MockitoExtension.class)
class OwnerControllerTest {

    @Mock(lenient = true)
    OwnerService service;
    @InjectMocks
    OwnerController controller;

    @Mock
    BindingResult bindingResult;
    @Test
    void processCreationFormHasErrors() {
        //given
        Owner owner = new Owner(1l,"John","Thomson");
        given(bindingResult.hasErrors()).willReturn(true);

        //when
        String viewName = controller.processCreationForm(owner,bindingResult);
        //then
        assertThat(viewName).isEqualToIgnoringCase("owners/
createOrUpdateOwnerForm");

    }

    @Test
    void processCreationFormNoErrors() {
        //given
        Owner owner = new Owner(5l,"John","Thomson");
        given(bindingResult.hasErrors()).willReturn(false);
        given(service.save(any())).willReturn(owner);

        //when
        String viewName = controller.processCreationForm(owner,bindingResult);
        //then
        assertThat(viewName).isEqualToIgnoringCase("redirect:/owners/5");
```

```
    }
}
```

Mockito ArgumentCapture

2 ways:
1. Writing in line of code 2. Using annotation

ArgumentCaptor is a feature provided by Mockito, a Java mocking framework, used for capturing arguments passed to mock objects during method invocations. It allows developers to inspect and verify the values of method parameters passed to mocked objects within their unit tests.
Here's how ArgumentCaptor is typically used:

- **Creating ArgumentCaptor**:
  - First, you create an instance of ArgumentCaptor for the type of argument you want to capture. For example:
  - Code:
    ```
    ArgumentCaptor<String> captor =
    ArgumentCaptor.forClass(String.class);
    ```

- **Interacting with Mock Object**:
  - Next, you perform method invocations on your mock object that you're interested in capturing arguments from.
- **Capturing Arguments**:
  - When the method is invoked on the mock object, you use the ArgumentCaptor to capture the arguments passed to it. For example:
  - Code:
    ```
    verify(mockObject).someMethod(captor.capture());
    ```

- **Inspecting Captured Values**:
  - After capturing the arguments, you can retrieve the captured values from the ArgumentCaptor and perform assertions or verifications on them. For example:
  - Code:
    ```
    assertEquals("expectedValue", captor.getValue());
    ```

e.g. 1st way:
```
@Test
```

```
void processFindFormWildcardString(){
    //given
    Owner owner = new Owner(5l,"John","Thomson");
    List<Owner> ownerList = new ArrayList<>();
    final ArgumentCaptor<String> captor =
ArgumentCaptor.forClass(String.class);

given(service.findAllByLastNameLike(captor.capture())).willReturn(ownerList);

    //when
    String viewName = controller.processFindForm(owner,bindingResult,null);

    assertThat("%Thomson%").isEqualToIgnoringCase(captor.getValue());

}
```

Written for this method.

```
public String processFindForm(Owner owner, BindingResult result, Model
model){
    // allow parameterless GET request for /owners to return all records
    if (owner.getLastName() == null) {
        owner.setLastName(""); // empty string signifies broadest possible search
    }

    // find owners by last name
    List<Owner> results = ownerService.findAllByLastNameLike("%"+
owner.getLastName() + "%");

    if (results.isEmpty()) {
        // no owners found
        result.rejectValue("lastName", "notFound", "not found");
        return "owners/findOwners";
    } else if (results.size() == 1) {
        // 1 owner found
        owner = results.get(0);
        return "redirect:/owners/" + owner.getId();
    } else {
        // multiple owners found
        model.addAttribute("selections", results);
        return "owners/ownersList";
    }
}
```

2nd way:

```
@Captor
ArgumentCaptor<String> stringArgumentCaptor;
```

```
@Test
void processFindFormWildcardStringAnnotation(){
    //given
    Owner owner = new Owner(5l,"John","Thomson");
    List<Owner> ownerList = new ArrayList<>();

given(service.findAllByLastNameLike(stringArgumentCaptor.capture())).willRet
urn(ownerList);

    //when
    String viewName = controller.processFindForm(owner,bindingResult,null);


assertThat("%Thomson%").isEqualToIgnoringCase(stringArgumentCaptor.getV
alue());

}
```

Using Mockito Answers:

- Helps in holding actual invocation of the mock and make some decisions about it.
- It compares very closely to argument captor but its gonna combine the where clause of what we gonna return form the mock.

```
@BeforeEach
void setup() {

given(service.findAllByLastNameLike(stringArgumentCaptor.capture())).
willAnswer(
        invocation -> {
            List<Owner> list = new ArrayList<>();
            String name = invocation.getArgument(0);
            if(name.equals("%Thomson%")){
                list.add(new Owner(1l,"John","Thomson"));
                return list;
            }
            throw new RuntimeException("Invalid Input");
        }
    );
}
```

This code is a JUnit test setup method annotated with @BeforeEach, which means it will be executed before each test method in the test class. The purpose of this setup method is to configure behavior for the findAllByLastNameLike method of a mock service object using Mockito's

given() and willAnswer() methods.
Let's break down the code:

- @BeforeEach **Annotation**:
  - This annotation is used to mark a method as a setup method that should run before each test method in the test class.
- setup() **Method**:
  - This method is the setup method that configures the behavior for the mock service object.
- given(service.findAllByLastNameLike(stringArgumentCaptor.capture())):
  - This part of the code sets up a behavior for the findAllByLastNameLike method of the mock service object.
  - service is a mock object of some service class.
  - findAllByLastNameLike(stringArgumentCaptor.capture()) is the method invocation that is being configured. It captures the argument passed to findAllByLastNameLike.
- willAnswer(...):
  - This method is used to define the behavior of the mock service method.
  - It takes a lambda expression as an argument, where the behavior of the method is implemented.
- **Lambda Expression**:
  - The lambda expression captures the invocation of the findAllByLastNameLike method and defines its behavior.
  - It extracts the argument passed to the method using invocation.getArgument(0) and stores it in the variable name.
  - If the name equals "%Thomson%", it creates a new Owner object with some predefined values and adds it to a list.
  - If the name does not equal "%Thomson%", it throws a RuntimeException with the message "Invalid Input".

In summary, this setup method configures the behavior of the findAllByLastNameLike method of the mock service object. When invoked with a specific argument ("%Thomson%"), it will return a list containing a predefined Owner object. Otherwise, it will throw an exception. This setup ensures consistent behavior for the mocked service method during tests.

Now there is no need of given in test:

```
@Test
   void processFindFormWildcardStringAnnotation(){
      //given
      Owner owner = new Owner(5l,"John","Thomson");
      List<Owner> ownerList = new ArrayList<>();
//
```

```
given(service.findAllByLastNameLike(stringArgumentCaptor.capture())).willRetu
rn(ownerList);

        //when
        String viewName = controller.processFindForm(owner,bindingResult,null);


assertThat("%Thomson%").isEqualToIgnoringCase(stringArgumentCaptor.getV
alue());
        assertThat("redirect:/owners/1").isEqualToIgnoringCase(viewName);

    }
```

Verify Order of Interactions:

- Testing service is called before the model is called

```
 @Test
    void processFindFormWildcardStringFound(){
        //given
        Owner owner = new Owner(5l,"John","FindMe");
        InOrder inOrder = inOrder(model,service);
//
given(service.findAllByLastNameLike(stringArgumentCaptor.capture())).willRetu
rn(ownerList);

        //when
//        String viewName = controller.processFindForm(owner,bindingResult,
Mockito.mock(Model.class));
        String viewName = controller.processFindForm(owner,bindingResult,
model);

assertThat("%FindMe%").isEqualToIgnoringCase(stringArgumentCaptor.getVal
ue());
        assertThat("owners/ownersList").isEqualToIgnoringCase(viewName);
        inOrder.verify(service).findAllByLastNameLike(anyString());
        inOrder.verify(model).addAttribute(anyString(),anyList());
    }
```

Here matters is order in which we verify... and not order in which we declare.

Verify interactions with specified time:

```
@Test
    void deleteById() {
```

```
        //given - none

        //when
        service.deleteById(1l);
        service.deleteById(1l);
        //then
//        verify(specialtyRepository,times(2)).deleteById(1l);
        then(specialtyRepository).should(timeout(100).times(2)).deleteById(1l);
    }
```

It is given in milliseconds...
It does not work with atMost matcher.
It works with atLeast, atLeastOnce, times only...

Verify zero or no more interactions with mock:

- We can do with model mock

```
@Test
    void processFindFormWildcardStringAnnotation(){
        //given
        Owner owner = new Owner(5l,"John","Thomson");
//        List<Owner> ownerList = new ArrayList<>();
//
given(service.findAllByLastNameLike(stringArgumentCaptor.capture())).willRetu
rn(ownerList);

        //when
        String viewName = controller.processFindForm(owner,bindingResult,null);


assertThat("%Thomson%").isEqualToIgnoringCase(stringArgumentCaptor.getV
alue());
        assertThat("redirect:/owners/1").isEqualToIgnoringCase(viewName);
        verifyZeroInteractions(model);
    }
    @Test
    void processFindFormWildcardStringNotFound(){
        //given
        Owner owner = new Owner(5l,"John","DontFindMe");
//
given(service.findAllByLastNameLike(stringArgumentCaptor.capture())).willRetu
rn(ownerList);

        //when
        String viewName = controller.processFindForm(owner,bindingResult,null);
```

```
assertThat("%DontFindMe%").isEqualToIgnoringCase(stringArgumentCaptor.getValue());
        assertThat("owners/findOwners").isEqualToIgnoringCase(viewName);
        verifyZeroInteractions(model);
    }
    @Test
    void processFindFormWildcardStringFound(){
        //given
        Owner owner = new Owner(5l,"John","FindMe");
        InOrder inOrder = inOrder(model,service);
//
given(service.findAllByLastNameLike(stringArgumentCaptor.capture())).willReturn(ownerList);

        //when
//        String viewName = controller.processFindForm(owner,bindingResult, Mockito.mock(Model.class));
        String viewName = controller.processFindForm(owner,bindingResult, model);
        //then

assertThat("%FindMe%").isEqualToIgnoringCase(stringArgumentCaptor.getValue());
        assertThat("owners/ownersList").isEqualToIgnoringCase(viewName);
        //inOrder asserts
        inOrder.verify(service).findAllByLastNameLike(anyString());
        inOrder.verify(model,times(1)).addAttribute(anyString(),anyList());
        verifyNoMoreInteractions(model); // not gonna invoke after this point
    }
```

Using Mockito Spies:

- Act as wrapper around the real implementation
- It is different from mock as it allows to access the underlying object and also allow it to treat like a normal mock where we can verify interaction with the spy.

```
package guru.springframework.sfgpetclinic.controllers;

import guru.springframework.sfgpetclinic.model.Pet;
import guru.springframework.sfgpetclinic.model.Visit;
import guru.springframework.sfgpetclinic.services.PetService;
import guru.springframework.sfgpetclinic.services.VisitService;
```

```
import guru.springframework.sfgpetclinic.services.map.PetMapService;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Spy;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.HashMap;
import java.util.Map;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.anyLong;
import static org.mockito.BDDMockito.given;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;

@ExtendWith(MockitoExtension.class)
class VisitControllerTest {

    @Mock
    VisitService visitService;

//    @Mock
//    PetService petService;
    //We will utilize the underlying object of PetService -> PetMapService
    @Spy
    PetMapService petService;

    @InjectMocks
    VisitController visitController;
    @Test
    void loadPetWithVisit() {
        //given
        Map<String, Object> model = new HashMap<>();
        Pet pet = new Pet(12L);
        Pet pet3 = new Pet(3L);

        petService.save(pet);
        petService.save(pet3);

        //explicitly calling the real method

given(petService.findById(anyLong())).willCallRealMethod(); //.willReturn(pet);

        //when
```

```
        Visit visit = visitController.loadPetWithVisit(12L, model);

        //then
        assertThat(visit).isNotNull();
        assertThat(visit.getPet()).isNotNull();
        assertThat(visit.getPet().getId()).isEqualTo(12L);
        verify(petService, times(1)).findById(anyLong());
    }


    //we can tell that spy to return back a value just like a mock
    //actually checking willcallRealMethod worked by explicitly throwing a wrong
answer by willReturn
    @Test
    void loadPetWithVisitWithStubbing() {
        //given
        Map<String, Object> model = new HashMap<>();
        Pet pet = new Pet(12L);
        Pet pet3 = new Pet(3L);

        petService.save(pet);
        petService.save(pet3);

        given(petService.findById(anyLong())).willReturn(pet3);

        //when
        Visit visit = visitController.loadPetWithVisit(12L, model);

        //then
        assertThat(visit).isNotNull();
        assertThat(visit.getPet()).isNotNull();
        assertThat(visit.getPet().getId()).isEqualTo(3L);
        verify(petService, times(1)).findById(anyLong());
    }
}
```