Advanced junit testing

Tags and Filtering:

Tags are way to identify test and put them in a group. Can be applied at class or method level. Click on java and select on run all tests.
In intellij, the tests will not run directly so go to edit confirmation and choose junit..

Give the name and in test kind put tags..
And in tag expression type the value same as given in tag annotation.

Gives control over which tests are being run over suite of tests.

Junit Nested classes:

```
@DisplayName("Owner Map Service Test - ")
class OwnerMapServiceTest {

    OwnerMapService ownerMapService;
    PetTypeService petTypeService;
    PetService petService;

    @BeforeEach
    void setUp() {
        petTypeService = new PetTypeMapService();
        petService = new PetMapService();
        ownerMapService = new OwnerMapService(petTypeService, petService);
    }

    @DisplayName("Verify Zero Owners")
    @Test
    void ownersAreZero() {
        int ownerCount = ownerMapService.findAll().size();

        assertThat(ownerCount).isZero();
    }

    @DisplayName("Pet Type - ")
    @Nested
    class TestCreatePetTypes {

        @BeforeEach
        void setUp() {
            PetType petType = new PetType(1L, "Dog");
            PetType petType2 = new PetType(2L, "Cat");
            petTypeService.save(petType);
```

```
        petTypeService.save(petType2);
    }

    @DisplayName("Test Pet Count")
    @Test
    void testPetCount() {
        int petTypeCount = petTypeService.findAll().size();

        assertThat(petTypeCount).isNotZero().isEqualTo(2);
    }

    @DisplayName("Save Owners Tests - ")
    @Nested
    class SaveOwnersTests {

        @BeforeEach
        void setUp() {
            ownerMapService.save(new Owner(1L, "Before", "Each"));
        }

        @DisplayName("Save Owner")
        @Test
        void saveOwner() {
            Owner owner = new Owner(2L, "Joe", "Buck");

            Owner savedOwner = ownerMapService.save(owner);

            assertThat(savedOwner).isNotNull();
        }

        @DisplayName("Save Owners Tests - ")
        @Nested
        class FindOwnersTests {

            @DisplayName("Find Owner")
            @Test
            void findOwner() {

                Owner foundOwner = ownerMapService.findById(1L);

                assertThat(foundOwner).isNotNull();
            }

            @DisplayName("Find Owner Not Found")
            @Test
            void findOwnerNotFound() {
```

```
                Owner foundOwner = ownerMapService.findById(2L);

                assertThat(foundOwner).isNull();
            }
        }
    }
}

    @DisplayName("Verify Still Zero Owners")
    @Test
    void ownersAreStillZero() {
        int ownerCount = ownerMapService.findAll().size();

        assertThat(ownerCount).isZero();
    }
}
```

Points:

- Running nested tests also runs before each test of parent and nested class respectively.
- We can see the hierarchy in the terminal also...
- Every time before each of most parent class resets the type of object so every time we get new object for them.
- What happening is beforeEach runs for every test so it resets the object for every class... and so at last tests give 0 count....

```
    @DisplayName("Verify Still Zero Owners")
    @DisplayName("Verify Still Zero Owners")
    @Test
    @DisplayName("Verify Still Zero Owners")
    @Test
    void ownersAreStillZero() {
        int ownerCount = ownerMapService.findAll().size();

        assertThat(ownerCount).isZero();
    }
```

Junit Test Interfaces:

- Helps in extends test classes using interfaces.

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

```
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;

@Tag("controllers")
public interface ControllerTests {
}
```

We can implements this for concrete classes and remove the tag from them....

Junit 4 could not pick up functionality like this...

Using Junit Default Test Methods:

We can use to reduce code duplication...

Directly introducing static beforeAll can lead to error

We have to add the annotation:

@TestInstance(TestInstance.Lifecycle.*PER_CLASS*)

```
package guru.springframework.sfgpetclinic;


import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;

@Tag("controllers")
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public interface ControllerTests {
    @BeforeAll
    static void beforeAll(){
        System.out.println("Lets do something here");
    }
}
```

Junit Repeated Tests:

```
@RepeatedTest(10)
void repeatTests(){

}
```

- We have special placeholders for it

```
@RepeatedTest(value = 10,name ="{displayName} : {currentRepetition} -
{totalRepetitions}")
void repeatTests(){}
```

- We can change the name also:

```
@RepeatedTest(value = 10,name ="{displayName} : {currentRepetition} -
{totalRepetitions}")
@DisplayName("MyRepeatedTest")
void repeatTests(){

}
```

Remember name of every test is (<display Name> <current iteration> of <number of iterations>).
Given by annotation repeated tag with name parameter.
Default name is repetition <current iteration > of <total iterations>
Overall name for this is given by @DisplayName.

Junit Dependency Injection:

Parameter Resolver:
Helps in injecting properties and objects into test methods.

TestReporter:
Publish runtime info back to metadata about the test that can be used for reporting later.

```
@RepeatedTest(5)
void myrepeatedTests(TestInfo testInfo, RepetitionInfo repetitionInfo){
    System.out.println(testInfo.getDisplayName() + ": " +
repetitionInfo.getCurrentRepetition());
}
```

Junit inject the objects at runtime....

Junit Parameterized Tests - Value Source:

Helps in providing data into the tests to use directly....

Add this dependency:

```
<dependency>
```

```
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <version>${junit-platform.version}</version>
    <scope>test</scope>
</dependency>

@ParameterizedTest
@ValueSource(strings = {"Spring","Framework","Pratik"})
void testValueSource(String val){
    System.out.println(val);
}
```

Output:

```
Running Test - [1] Spring
Spring
Running Test - [2] Framework
Framework
Running Test - [3] Pratik
Pratik
```

By default name of every test is [index] <current value>

```
@DisplayName("My value test")
@ParameterizedTest(name = "{displayName} : [{index}] {arguments}")
@ValueSource(strings = {"Spring","Framework","Pratik"})
void testValueSource(String val){
    System.out.println(val);
}
```

We have changed by using placeholder...

Junit Parameterized Tests - ENUM Source

```
package guru.springframework.sfgpetclinic.model;

public enum OwnerType {
    INDIVIDUAL, COMPANY
}

@DisplayName("My Enum test")
@ParameterizedTest(name = "{displayName} : [{index}] {arguments}")
@EnumSource(OwnerType.class)
void testValueSource(OwnerType type){
```

```
    System.out.println(type);
}
```

Output:
Running Test - My Enum test : [1] INDIVIDUAL
INDIVIDUAL
Running Test - My Enum test : [2] COMPANY
COMPANY

Junit Parameterized Tests - CSV Source

There are 2 ways of doing -> 1) through list 2) through file

1)

```
@DisplayName("My Enum test")
@ParameterizedTest(name = "{displayName} : [{index}] {arguments}")
@CsvSource({
        "FL, 1, 1",
        "OH, 2, 2",
        "MI, 1, 1"
})
void csvInputTest(String stateName, int val1, int val2){
    System.out.println(stateName + " = " + val1 + ":" + val2 );
}
```

Output:

Running Test - My Enum test : [1] FL, 1, 1
FL = 1:1
Running Test - My Enum test : [2] OH, 2, 2
OH = 2:2
Running Test - My Enum test : [3] MI, 1, 1
MI = 1:1

2)

Make an input.csv file under test/resources

```
state, val1, val2
FL, 1, 1
OH, 2, 2
MI, 1, 1
```

```
@DisplayName("CSV From File Test")
@ParameterizedTest(name = "{displayName} : [{index}] {arguments}")
@CsvFileSource(resources = "/input.csv",numLinesToSkip = 1)
```

```
void csvFromFileTest(String stateName, int val1, int val2){
    System.out.println(stateName + " = " + val1 + ":" + val2 );
}
```

numLinesToSkip -> to skip header.

Junit Parameterized Tests - Method Provider

Using provider method gives us lot of capabilities ->

1. Talking to database
2. Talking to message queue
3. Reading xml file

```
static Stream<Arguments> getArgs(){
    return Stream.of(Arguments.of("FL",1,1),
            Arguments.of("OH",2,2),
            Arguments.of("MI",3,3));
}

@DisplayName("Method Provider Test")
@ParameterizedTest(name = "{displayName} : [{index}] {arguments}")
@MethodSource("getArgs")
void fromMethodTest(String stateName, int val1, int val2){
    System.out.println(stateName + " = " + val1 + ":" + val2 );
}
```

output:

```
Running Test - Method Provider Test : [1] FL, 1, 1
FL = 1:1
Running Test - Method Provider Test : [2] OH, 2, 2
OH = 2:2
Running Test - Method Provider Test : [3] MI, 3, 3
MI = 3:3
```

Junit Parameterized Tests - Custom Provider

```
package guru.springframework.sfgpetclinic;

import org.junit.jupiter.api.extension.ExtensionContext;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.ArgumentsProvider;

import java.util.stream.Stream;

public class CustomArgsProvider implements ArgumentsProvider {
    @Override
```

```
    public Stream<? extends Arguments> provideArguments(ExtensionContext
extensionContext) throws Exception {
        return Stream.of(Arguments.of("FL",1,1),
                Arguments.of("OH",2,2),
                Arguments.of("MI",3,3));
    }
}

@DisplayName("Custom Provider Test")
@ParameterizedTest(name = "{displayName} : [{index}] {arguments}")
@ArgumentsSource(CustomArgsProvider.class)
void fromCustomProvider(String stateName, int val1, int val2){
    System.out.println(stateName + " = " + val1 + ":" + val2 );
}
```

Output:

```
Running Test - Custom Provider Test : [1] FL, 7, 10
FL = 7:10
Running Test - Custom Provider Test : [2] OH, 2, 5
OH = 2:5
Running Test - Custom Provider Test : [3] MI, 3, 3
MI = 3:3
```

Junit Extensions:

Allow extend Junit API...
Junit has no extension we have to give custom extensions...

e.g

```
package guru.springframework.sfgpetclinic.junitextensions;

import org.junit.jupiter.api.extension.AfterTestExecutionCallback;
import org.junit.jupiter.api.extension.BeforeTestExecutionCallback;
import org.junit.jupiter.api.extension.ExtensionContext;

import java.lang.reflect.Method;
import java.util.logging.Logger;

/**
 * Original source - https://junit.org/junit5/docs/current/user-guide/#extensions-
lifecycle-callbacks-timing-extension
 *
 * Created by jt on 2018-10-28.
 */
```

```
public class TimingExtension implements BeforeTestExecutionCallback,
AfterTestExecutionCallback {

    private static final Logger logger =
Logger.getLogger(TimingExtension.class.getName());

    private static final String START_TIME = "start time";

    @Override
    public void beforeTestExecution(ExtensionContext context) throws Exception
{
        getStore(context).put(START_TIME, System.currentTimeMillis());
    }

    @Override
    public void afterTestExecution(ExtensionContext context) throws Exception {
        Method testMethod = context.getRequiredTestMethod();
        long startTime = getStore(context).remove(START_TIME, long.class);
        long duration = System.currentTimeMillis() - startTime;

        logger.info(() -> String.format("Method [%s] took %s ms.",
testMethod.getName(), duration));
    }

    private ExtensionContext.Store getStore(ExtensionContext context) {
        return context.getStore(ExtensionContext.Namespace.create(getClass(),
context.getRequiredTestMethod()));
    }
}
```

This shows how long each method runs

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

import guru.springframework.sfgpetclinic.junitextensions.TimingExtension;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;

import static org.junit.jupiter.api.Assertions.*;

@ExtendWith(TimingExtension.class)
class PetTypeSDJpaServiceIT {

    @BeforeEach
    void setUp() {
    }
```

```
    @Test
    void findAll() {
    }

    @Test
    void findById() {
    }

    @Test
    void save() {
    }

    @Test
    void delete() {
    }

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

@Extend helps to extend Junit Test and hook into Junit API. Third parties to extend junit tests.