TestNG

Введение

- Оболочки модульного тестирования — это программные средства для разработки тестов, включающие: построение, выполнение тестов и создание отчетов.
- Первая оболочка модульного тестирования SUnit была создана Кентом Беком в 1999 году для языка Smalltalk.
- Позднее Эрик Гамма вместе с Кентом Беком создали JUnit.
- TestNG — имеет много общего с JUnit. Это достаточно широко используемая и расширенная версия оболочек модульного тестирования.
- TestNG создан на языке Java и используется для тестирования Java кода

Модульное тестирование

- Модульное тестирование, или unit testing, — процесс проверки на корректность функционирования отдельных частей исходного кода программы путем запуска тестов в искусственной среде.
- Под частью кода в Java следует понимать исполняемый компонент.
- С помощью модульного тестирования обычно тестируют такие низкоуровневые элементы кода, как методы.
- Кроме основного положительного сценария, может выполняться проверка работоспособности системы в альтернативных сценариях, например, при генерации методом исключения как реакция на ошибочные исходные данные.

Подключение с использованием Maven

```
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.6.1</version>
</dependency>
```

Подключение с использованием Maven

```
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.6.1</version>
    <scope>test</scope>
</dependency>
```

Для удобства, можно подключить дополнительные библиотеки

```
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.23.1</version>
    <scope>test</scope>
</dependency>
```

Основные моменты

- тесты разрабатываются для нетривиальных методов системы;
- ошибки выявляются в процессе проектирования метода или класса;
- в первую очередь разрабатываются тесты на основной положительный сценарий;
- разработчику приходится больше уделять внимания альтернативным сценариям поведения, так как они являются источником ошибок, выявляемых на поздних стадиях разработки;
- разработчику приходится создавать более сфокусированные на своих обязанностях методы и классы, так как сложный код тестировать значительно труднее;
- снижается число новых ошибок при добавлении новой функциональности;
- устаревшие тесты можно игнорировать;
- тест отражает элементы технического задания, то есть некорректное завершение теста сообщает о нарушении технических требований заказчика;
- каждому техническому требованию соответствует тест;
- и как результат, получение работоспособного кода с наименьшими затратами.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Аннотация @Test

- Аннотация **@Test** из пакета **org.testng.annotations** помечает метод как тестовый, что позволяет использовать возможности класса **org.testng.Assert** и запускать его в режиме тестирования.
- Тестовый метод должен всегда объявляться как **public void**.
- Аннотация может использовать параметры

Параметры аннотации @Test

- **description** — содержит описание тестового метода;
- **enabled** — определяет, включена аннотация либо игнорируется;
- **expectedExceptions** — определяет список ожидаемых классов исключений;
- **timeOut** — определяет время, превышение которого делает тест ошибочным;
- **alwaysRun** — при значении true метод будет запускаться всегда вне зависимости от принадлежности к группе;
- **priority** — задает приоритет теста;
- **groups** — включение тестового метода в группу тестов;
- **dataProvider** — имя для доступа к данным для теста;
- **invocationCount** — число вызовов метода;
- **dependsOnMethods**, **dependsOnGroups** — определяет зависимость теста от выполнения предыдущего метода или группы.

Пример разработки теста (1 / 2)

```
package main;

public class Converter {
    private static final int ABSOLUTE_ZERO = -273;

    public double convertCelsiusToFahrenheit(double celsius) {
        if (celsius < ABSOLUTE_ZERO) {
            throw new IllegalArgumentException("error data");
        }
        double fahrenheit = celsius * 1.8 + 32;
        return fahrenheit;
    }

    public boolean checkCelsius(double celsius) {
        return celsius >= ABSOLUTE_ZERO;
    }

}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Пример разработки теста (2 / 2)

```
package main;

import org.testng.annotations.Test;

import static org.testng.Assert.*;

public class ConverterTest {

   Converter converter = new Converter();

   @Test()
   public void testConvertCelsiusToFahrenheit() {
      double actual = converter.convertCelsiusToFahrenheit(10.0);
      double expected = 50.;
      assertEquals(actual, expected, 0.001, "Test failed as...");
   }
}
```

assertEquals()

- Статический метод **assertEquals()** проверяет на равенство значений **expected** и **actual** с возможной погрешностью **delta**.
- При выполнении заданных условий сообщает об успешном завершении, в противном случае — об аварийном завершении теста.
- При аварийном завершении генерируется ошибка **java.lang.AssertionError**.

Методы класса **Assert**

- Все методы класса **Assert** в качестве возвращаемого значения имеют тип **void**.
- Среди них можно выделить такие:

- **assertTrue(boolean condition) / assertFalse(boolean condition)** — проверяет на истину/ложь значение **condition**;
- **assertSame(Object expected, Object actual)** — проверяет, ссылаются ли ссылки на один и тот же объект;
- **assertNotSame(Object unexpected, Object actual)** — проверяет, ссылаются ли ссылки на различные объекты;
- **assertNull(Object object)/assertNotNull(Object object)** — проверяет, имеет или не имеет ссылка значение **null**;
- **fail()** — вызывает ошибку, используется для проверки, достигнута ли определенная часть кода или для заглушки, сообщающей, что тестовый метод пока не реализован

Примеры использования

```
@Test
public void checkSameTest() {
    String actual = "java" + 17;
    String expected = "ja" + "va" + 17;
    assertSame(actual, expected);
}

@Test
public void testCheckCelsiusFalse() {
    boolean condition = converter.checkCelsius(-300);
    assertFalse(condition, "test false failed as...");
}
```

В тестовом методе не следует размещать более одного assert-метода

Фикстуры

- Фикстура (*Fixture*) — состояние среды тестирования, которое требуется для успешного выполнения тестового метода.
- Может быть представлено набором каких-либо объектов, состоянием базы данных, наличием определенных файлов, соединений и проч.
- В TestNG аннотации позволяют исполнять одну и ту же фикстуру для каждого теста или всего один раз для всего класса, или не исполнять ее совсем.
- Предусмотрено восемь аннотаций фикстур — по две для фикстур уровня теста, группы, класса и метода

Фикстуры

- **@BeforeSuite** и **@AfterSuite** — запускается только один раз перед и после запуска всех тестов. Например, конфигурирование пула соединений, создание и заполнение тестовой БД.
- **@BeforeGroups** и **@AfterGroups** — запускается до вызова первого и после вызова последнего метода каждой группы.
- **@BeforeClass** и **@AfterClass** — запускается только один раз до и после запуска всех тестов класса.
- **@BeforeMethod** и **@AfterMethod** — запускается до и после каждого тестового метода.

Фикстуры

- Фикстурой **@BeforeClass** задан момент создания объектов класса перед каждым тестом, что позволит всем тестам класса пользоваться одними и теми же объектами. По окончании работы всех методов объект может быть уничтожен.

```
public class ConverterTest {
    Converter converter;
    @BeforeClass
    public void setUp() {
        converter = new Converter();
    }
    @Test
    public void testConvertCelsiusToFahrenheit() {
        double actual = converter.convertCelsiusToFahrenheit(10.0);
        double expected = 50.;
        assertEquals(actual, expected, 0.001, " Test failed as...");
    }
    @Test()
    public void testCheckCelsius() {
        boolean condition = converter.checkCelsius(-45);
        assertTrue(condition);
    }
    @AfterClass
    public void tierDown(){
        converter = null;
    }
}
```

Тестирование исключительных ситуаций

- При тестировании альтернативных сценариев работы метода часто требуется точно определить тип генерируемого методом исключения на основе переданных некорректных параметров.
- Если тест выдает исключение, то инфраструктура тестирования сообщает о корректном результате его исполнения.
- Аннотацию **@Test** при необходимости тестирования генерации конкретного исключения следует использовать с параметром **expectedExceptions**.
- Параметр предназначен для задания типа исключения, которое данный тест должен генерировать в процессе своего выполнения.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Тестирование исключительных ситуаций

```
@Test(enabled = true, expectedExceptions = IllegalArgumentException.class,
        expectedExceptionsMessageRegExp = "error data")
public void testConvertCelsiusToFahrenheitException() {
    converter.convertCelsiusToFahrenheit(-300);
}
```

- Метод **convertCelsiusToFahrenheit(double celsius)** генерирует исключение **IllegalArgumentException**, если число, переданное в метод, выходит за границы минимального отрицательного значения.
- Тест завершится успешно лишь в том случае, если будет сгенерирована исключительная ситуация.
- Исключение **IllegalArgumentException** не нужно указывать в секции **throws** тестового метода, так как оно относится к непроверяемым.
- Если же метод может выбрасывать проверяемое исключение, то, по правилам работы с проверяемыми исключениями, его следует указывать в секции **throws** тестового метода

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Тестирование исключительных ситуаций

- Может появиться необходимость проверить не только возникновение исключительной ситуации, но и текст сообщения в экземпляре исключения.
- В этом случае лучше прибегнуть к обычному подходу без параметра **expected**.

```
@Test(expectedExceptionsMessageRegExp = "error data")
public void testConvertCelsiusToFahrenheitExceptionMessage() {
    double celsius = -300;
    try {
        converter.convertCelsiusToFahrenheit(celsius);
        fail("Test for celsius " + celsius
                + " should have thrown a IllegalArgumentException");
    } catch (IllegalArgumentException e) {
        assertEquals("error data", e.getMessage());
    }
}
```

- В блоке **try**, если исключение не возникло, вызывается метод **fail()**, сигнализирующий о провале теста.
- В блоке **catch**, если исключительная ситуация произошла, проверяется на эквивалентность текстов сообщения об ошибке.

Ограничение по времени

- В качестве параметра тестового сценария аннотации **@Test** может быть использовано значение лимита времени **timeOut**.
- Параметр **timeOut** определяет максимальный временной промежуток в миллисекундах, отводимый на исполнение теста.
- Если выделенное время истекло, а тест продолжает выполняться, то тест завершается неудачей

```
@Test(timeOut = 1_000)
public void testTime() {
    IntStream.range(-273, 100_000_000)
            .boxed()
            .forEach(t -> converter.convertCelsiusToFahrenheit(t));
}
```

Примерно через 500 миллисекунд после запуска тест провалится, так как на выполнение метода уходит несколько больше времени. Если увеличить время до одной секунды, то тест пройдет успешно. Если заменить реализацию тела теста со Stream API на обычный цикл, то времени на выполнение теста понадобится намного меньше:

Ограничение по времени

- Примерно через 500 миллисекунд после запуска тест провалится, так как на выполнение метода уходит несколько больше времени.
- Если увеличить время до одной секунды, то тест пройдет успешно.
- Если заменить реализацию тела теста со Stream API на обычный цикл, то времени на выполнение теста понадобится намного меньше:

```
@Test(timeOut = 250)
public void testTime() {
    for (int t = -273; t < 100_000_000; t++) {
        converter.convertCelsiusToFahrenheit(t);
    }
}
```

Игнорирование тестов

- При контроле корректности функционирования бизнес-логики приложений игнорирование нереализованных, незавершенных или устаревших тестов может представлять определенную проблему.
- Параметр **enabled=false** заставляет инфраструктуру тестирования проигнорировать данный тестовый метод.
- Параметр **description** в этом случае предусматривает наличие комментария о причине игнорирования теста, полезного при следующем к нему обращении, например, в виде:

```
@Test(description = "test is ignored because ...", enabled = false)
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Параметризованные тесты

- TestNG позволяет создавать тест, который может работать с различными наборами значений параметров, что дает возможность разработать единый тестовый сценарий и запускать его несколько раз — по числу наборов параметров.
- Создание параметризованного теста с набором данных требует создания метода для формирования и поставки данных, помеченного аннотацией **@DataProvider**.
- Аннотации следует задать имя, по которому к нему будут обращаться методы-тесты.
- Тестовый метод для использования данных, предоставляемых **DataProvider**, должен в аннотации теста указать имя провайдера в параметре **dataProvider**

```
@DataProvider(name = "celsius_3")
public Object[][] createData() {
    return new Object[][]{{10, 50}, {0, 32}, {40, 104}};
}
@Test(dataProvider = "celsius_3")
public void testParamsConvert(double celsius, double expectedFahrenheit) {
    double actual = converter.convertCelsiusToFahrenheit(celsius);
    assertEquals(actual, expectedFahrenheit, 0.001);
}
```

Параметризованные тесты

- Тест будет запущен по числу наборов данных, в данном случае — три раза, что и будет отражено в консоли.
- Совершенно необязательно хранить данные в коде тестового класса.
- Параметры можно передавать через файл конфигурации **testing.xml**.
- К методу добавляется аннотация **@Parameters** со списком имен параметров.

```
@Test
@Parameters({"celsius", "expectedFahrenheit"})
public void testConvertWithParam(double celsius, double expectedFahrenheit) {
    double actual = converter.convertCelsiusToFahrenheit(celsius);
    assertEquals(actual, expectedFahrenheit, 0.001);
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Параметризованные тесты

- Эти же параметры становятся параметрами метода и получат свои значения из файла **testng.xml**.

```
<suite name="SuiteParam" verbose="1">
    <parameter name="celsius" value="10"/>
    <parameter name="expectedFahrenheit" value="50"/>
    <test name="TestParam">
        <classes>
            <class name="main.ConverterTest">
                <methods>
                    <include name="testConvertWithParam"/>
                </methods>
            </class>
        </classes>
    </test>
</suite>
```

- Параметры будут передаваться всем классам, объявленным после тега **parameter**.
- Методы могут исключаться из теста класса тегом **<exclude>**.
- Еще два способа хранения параметров для последующей передачи в метод представлены в виде класса со статическими методами, возвращающими массив массивов и итератор.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Параметризованные тесты

```
public class StaticDataProvider {
    @DataProvider(name = "dataProviderArray")
    public static Object[][] createData() {
        return new Object[][]{{0.0, 32.0}, {10, 50.0}, {20, 68}};
    }

    @DataProvider(name = "dataProviderIterator")
    public static Iterator<Object[]> createIterator() {
        Object[][] ob = new Object[][]{{5, 41}, {15, 59}, {30, 86}};
        List<Object[]> list = List.of(ob);
        return list.iterator();
    }
}
```

Методы получают данные, указывая имя необходимого ему **dataProvider** и класс, где он расположен.

```
@Test(dataProvider = "dataProviderArray", dataProviderClass = StaticDataProvider.class)
public void testConvertCelsiusArray(double in, double expected){
    double actual = converter.convertCelsiusToFahrenheit(in);
    assertEquals(actual, expected);
}
@Test(dataProvider="dataProviderIterator",dataProviderClass = StaticDataProvider.class)
public void testConvertCelsiusIterator(double in, double expected){
    double actual = converter.convertCelsiusToFahrenheit(in);
    assertEquals(actual, expected);
}
```

Группы тестов

- Тесты можно объединять в группы и отправлять на выполнение группами.
- В группе могут находиться часть методов одного и часть методов нескольких классов.
- Один метод может находиться в нескольких группах.
- Группы тестов могут тестировать какую-либо часть приложения, либо относиться к какому-либо отдельному методу.
- Группы, в свою очередь, также могут быть частью других групп.
- Группы можно запускать одну за другой либо параллельно.
- Гибкость организации процесса тестирования повышается

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Группы тестов

```
public class ConverterTestGroup {
    Converter converter;
    @BeforeGroups(groups = {"calc", "check"})
    public void setUp() {
        converter = new Converter();
    }
    @Test(groups = {"calc"})
    @Parameters({"celsius", "expectedFahrenheit"})
    public void testConvertWithParam(double celsius, double expectedFahrenheit) {
        double actual = converter.convertCelsiusToFahrenheit(celsius);
        assertEquals(actual, expectedFahrenheit, 0.001);
    }
    @Test(groups = {"check", "calc"}, timeOut = 1000)
    public void testTime() {
        for (int t = -273; t < 100_000_000; t++) {
            converter.convertCelsiusToFahrenheit(t);
        }
    }
    @Test(groups = {"calc"})
    public void testConvertCelsiusToFahrenheit() {
        double actual = converter.convertCelsiusToFahrenheit(10.0);
        double expected = 551.;// failed value
        assertEquals(actual, expected, 0.001, " Test failed as...");
    }
    @Test(groups = {"check"}, expectedExceptions = IllegalArgumentException.class)
    public void testConvertCelsiusToFahrenheitException() {
        converter.convertCelsiusToFahrenheit(-300);
    }
    @Test(groups = {"check"})
    public void testCheckCelsius() {
        boolean condition = converter.checkCelsius(-45);
        assertTrue(condition);
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Группы тестов

Тег **<groups>**, определяющий группу, может располагаться в тегах **<suite>** или **<test>**. Группа объявляется со своим именем, на которые и ссылается метод, включенный в эту группу

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="All Test Suite">
    <test name="check_arg1">
        <groups>
            <run>
                <include name="arg2"/>
                <include name="converter"/>
            </run>
        </groups>
        <classes>
            <class name="main.ConverterTest"/>
            <class name="main.MainTest"/>
        </classes>
    </test>
    <parameter name="celsius" value="10"/>
    <parameter name="expectedFahrenheit" value="50"/>
    <test name="TestParam">
        <parameter name="celsius" value="10"/>
        <parameter name="expectedFahrenheit" value="50"/>
    </test>
</suite>
```

Приоритеты и зависимости

- Когда тестируется некоторый последовательный рабочий процесс приложения: чтение данных, парсинг строк, создание объектов, выполнение бизнес логики и пр., то необходимо организовать работу теста, чтобы методы вызывались в определенном порядке.
- Также можно сделать запуск методов зависимым
  одного от другого, то есть если не сработал некоторый метод в тесте, то последующий уже вызывать нет смысла по логике тестирования.
- Пусть разработано приложение для конвертации данных из шкалы по Цельсию в шкалу по Фаренгейту, которое включает чтение данных из файла в виде строки:

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Приоритеты и зависимости

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
public class DataReader {
    public String read(String filename) {
        String data = null;
        Path path = Paths.get(filename);
        if(Files.exists(path) && !Files.isDirectory(path) && Files.isReadable(path)) {
            try {
                StringBuilder builder = new StringBuilder();
                data = Files.lines(path).reduce((s1, s2) -> s1 + " " +
                                                            s2).orElse("empty");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return data;
    }
}
```

где файл celsius.txt представлен в виде:

```
0 10.1 20 a7b 36.6
45 -25 -300
7 5n.9 -888.1
-1 777
```

Приоритеты и зависимости

- Разбор данных из строки, преобразование их к числовому виду и фильтрация некорректных данных, не являющихся числом:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class DataParser {
    public List<Double> parseData(String data) {
        List<Double> doubles = new ArrayList<>();
        Arrays.stream(data.split("\\s+"))
                .filter(s -> s.matches("-?\\d{1,6}\\.?\\d{0,2}?"))
                .forEach(d -> doubles.add(Double.valueOf(d)));
        return doubles;
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Приоритеты и зависимости

- Разработанные тесты на основной положительный сценарий приложения объединены в группу **base_flow**, для которой файл **testng.xml** содержит:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="SuiteDependenPrior" verbose="1">
    <test name="Test Group_Main">
        <groups>
            <run>
                <include name="base_flow"/>
            </run>
        </groups>
        <classes>
            <class name="main.action.ConverterPriorTest"/>
            <class name="main.action.DataParserTest"/>
            <class name="main.action.DataReaderTest"/>
        </classes>
    </test>
</suite>
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Классы тестов: DataReaderTest

```
public class DataReaderTest {
    static final String TEST_FILE = "data\\celsius.txt";

    @Test(groups = "base_flow")
    public void testRead() {
        DataReader reader = new DataReader();
        String actual = reader.read(TEST_FILE);
        String expected =
           "0 10.1 20 a7b 36.6 45 -25 -300 7 5n.9 -888.1 -1 777";
        assertEquals(actual, expected);
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Классы тестов: DataParserTest

```
public class DataParserTest {
    @Test(groups = "base_flow")
    public void testParseData() {
        DataParser parser = new DataParser();
        List<Double> actual = parser.parseData(
          "0 10.1 20 a7b 36.6 45 -25 -300 7 5n.9 -888.1 -1 777");
        List<Double> expected = List.of(
           0d, 10.1d, 20d, 36.6d, 45d, -25d,
           -300d, 7d, -888.1, -1d, 777d);
        Assert.assertEquals(actual, expected);
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Классы тестов: ConverterPriorTest (1/2)

```
public class ConverterPriorTest {
    Converter converter;
    @BeforeGroups(groups = "base_flow")
    public void setUp() {
        converter = new Converter();
    }
    @Test(dataProvider = "celsius_9", groups = "base_flow")
    public void testParamsConvert(double celsius, double
expectedFahrenheit) {
        double actual = converter.convertCelsiusToFahrenheit(celsius);
        assertEquals(actual, expectedFahrenheit, 0.001);
    }
    @Test(timeOut = 250, groups = "base_flow")
    public void testTime() {
        for (int t = -273; t < 100_000_000; t++) {
            converter.convertCelsiusToFahrenheit(t);
        }
    }
    @Test(dataProvider = "celsius_check_10", groups = "base_flow")
    public void testCheckCelsius(double celsius, boolean flag) {
        boolean condition = converter.checkCelsius(celsius);
        assertTrue(condition == flag);
    }
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Классы тестов: ConverterPriorTest (2/2)

```
    @DataProvider(name = "celsius_9")
    public Object[][] createData() {
        return new Object[][]{
                {0.0, 32.0}, {10.1, 50.18}, {20.0, 68.0},
                {36.6, 97.88}, {45.0, 113.0}, {-25.0, -13.0},
                {7.0, 44.6}, {-1.0, 30.2}, {777.0, 1430.6}
        };
    }

    @DataProvider(name = "celsius_check_10")
    public Object[][] createDataCheck() {
        return new Object[][]{
                {0.0, true}, {10.1, true}, {20.0, true},
                {36.6, true}, {45.0, true}, {-25.0, true},
                {7.0, true}, {-888.1, false}, {-1.0, true},
                {777.0, true}
        };
    }
}
```