bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Test-Driven Development Lab

Description of exercises for the training

bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Task 1 - Calculator

Goal: Familiarize yourself with the TDD process.

Write a calculator class using TDD

1. Write a method adding and a method subtracting two integers, for example:

```
[C#]   int result = calculator.Add(4, 5);
[Java] int result = calculator.add(4, 5);
```

2. How should the calculator react when the range is exceeded (the result exceeds `Integer`)?

   Let the method throw an exception in this case. How will the test look?

3. Write a method dividing two integers that has the result `double`.

```
[C#]   double result = calculator.Divide(10, 5);
[Java] double result = calculator.divide(10, 5);
```

4. In the case of dividing by zero let the method throw an exception.

Note:

- Remember about the following conventions:
  - Test classes have `Behaviour` or `Test` suffixes;
  - Test methods start from `[C#] Should` or `[Java] should`.
- Given mathematical operations should be tested for different cases – use parametrized tests (`[C#] [TestCase]` or `[Java] @RunWith(Parametrized.class)`).

bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Task 2 – Multilingual string

Goal: Familiarize yourself with the TDD process as a source of API-related decisions.

Use TDD to create `StringI18n` class which will store multilingual strings. Remember to start from the test!

The class has to fulfill the following requirements:

1. For a given character string you can download any existing language version by specifying the name of this version.
2. If the specified language version does not exist, an exception occurs.
3. If the language version is not specified, the string is provided in the default language version.
4. The character string must have at least one language version.

Example of an object from this class:

```
[C#]
StringI18n hello = new StringI18n("pl", "Cześć");
hello.Add("en", "Hello");
hello.Add("de", "Hallo");
String polishHello = hello.Get("pl");

[Java]
StringI18n hello = new StringI18n("pl", "Cześć");
hello.add("en", "Hello");
hello.add("de", "Hallo");
String polishHello = hello.get("pl");
```

bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Task 3 – Class reading the contents of a directory

Goal: Create complex integration testing and use refactoring in TDD.

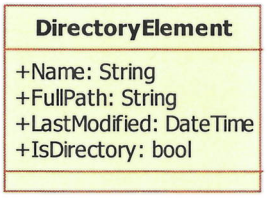
Use TDD to create `DirectoryReader` class which will provide information about a specific directory. Information about a single element of the directory should be represented by an object of the `DirectoryReader` class.

When the solution is ready, refactor the existing code:

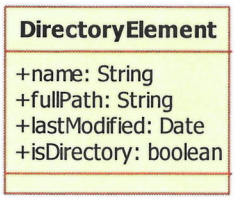
- eliminate repetitions;
- extract methods which increase readibility;
- introduce custom assertions;
- introduce factory methods to create objects for testing.

Example of calling the method of this class:

```
[C#]   List<DirectoryElement> elements
                = directoryReader.Read(@"C:\Program Files");
```




```
[Java] List<DirectoryElement> elements
                = directoryReader.read("C:\\Program Files");
```



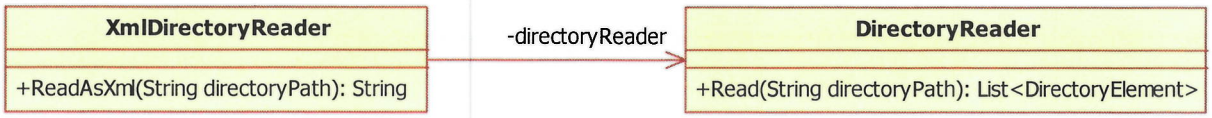
bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Task 4 – Class presenting data in the form of a directory in XML

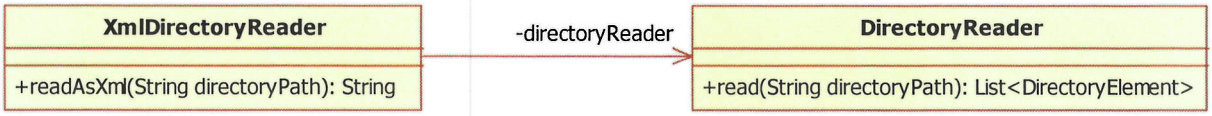
Goal: Simplify tests by testing the class in isolation from related classes.

In the previous task we created the `DirectoryReader` class, now we want to create one that is quite similar, i.e. `XmlDirectoryReader`. The main task of this class will be to present the contents of the directory as XML. Since we already have the functionality of reading the directory information, we only have to implement converting objects to XML. For this purpose, we will use composition.

[C#]

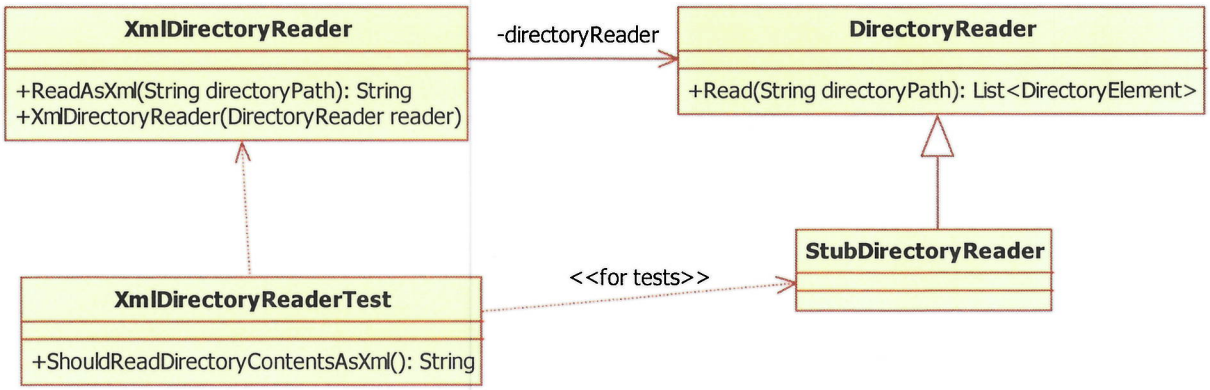


[Java]



Reading the contents of the directory has already been tested, so the test of the `XmlDirectoryReader` class should apply only to converting objects to XML. For this purpose, substitute the real `DirectoryReader` object with **Stub** object.

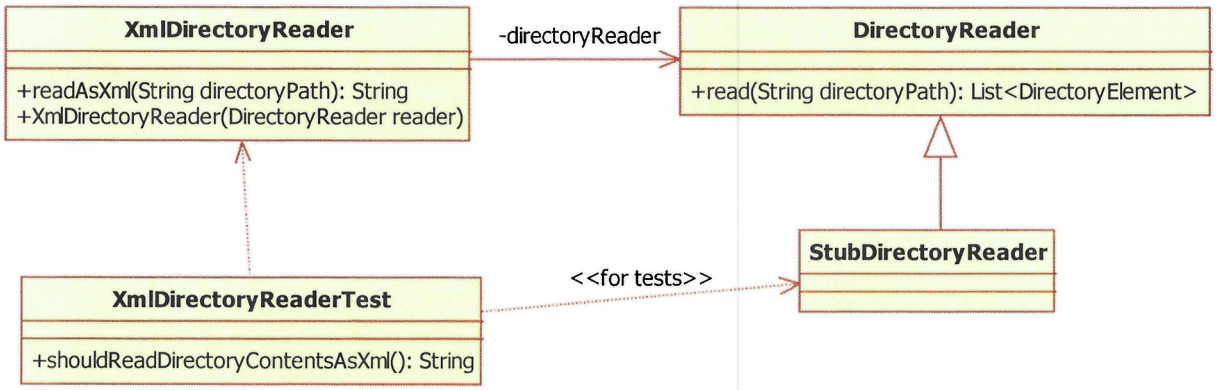
[C#]



bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Test-Driven Development Lab

[Java]



bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Task 5 – Experimenting with Mock Objects library.

Goal: Familiarize yourself with capabilities of the Mock Objects libraries.

Go through all examples included in the training materials. Experiment with different parameters and different calls of the mechanisms presented there.

bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Task 6 – Class presenting information about a directory in XML - version using Mock Objects.

Goal: Use Mock Objects to isolate dependencies.

In the previous task we tested the `XmlDirectoryReader` class using **Stub**. In the long run, creating this type of classes is uncomfortable. Modify the existing test in order to use **Mock** objects.

bns it
Łódź, ul. Piotrkowska 60
tel. 42 630 24 42
fax 42 209 38 63
www.bnsit.pl

Task 7 – Text obfuscator.

Goal: Experience how tests might give hints about the design. Introduce best practices for improving code testability.

Your task is to create a class the aim of which is to obfuscate the input text. Here is an example.

Input text:

Lehman's second law describes the phenomenon known in many areas of physics: increasing disorder (entropy) over time.

Obfuscated version:

law!Second!LEHMAns..dEsCribesthepHERnomenon!KNowN.in,areAS,mAnY!!ofPHysiCs!increaSIngDiSoRDer!.ENtropY. ,hOwevER! SOMEhoW!oVEr..tImE SoMEhOw..timeHowEVEr

Use TDD to write code that obscures the text in a way presented above. Obfuscator should make the following transformations:

- deleting the spaces;
- replacement of Polish diacritical marks with non-diacritics;
- random replacement of uppercase/lowercase letters (rarely);
- removal of punctuation;
- inserting random punctuation;
- swapping the order of the letters in a word (rarely);
- deleting a character from a word (rarely);
- adding insignificant conjunctions to sentences (rarely);
- swapping the order of words in a sentence (rarely).

Note:

- How to test the entire mechanism?
- How to deal with the randomness of the process?
- What hints do the tests bring in relation to the code?
- Will Mock Objects be useful here?

[Option] Implementation of a simple state machine using TDD.