Learning aims

The aim of this practice sheet is to get to know and acquire basic experience in test-driven development (TDD) and pair programming. It serves as preparation for the next practice sheet for which you will implement test cases in a real open source project.

Task 2-1: Familiarize with TDD

The aim of this task is to familiarize with the process of test-driven development.

1. Study the following sources and make sure to gain a good understanding of TDD for answering the questions following below. (If necessary, look for and study additional sources.)
   - Scott W. Ambler: Introduction to Test Driven Development
     http://www.agiledata.org/essays/tdd.html
     (You may skip sections 5, 9, and 10)
   - Frank Westphal: Testgetriebene Entwicklung
     http://www.frankwestphal.de/TestgetriebeneEntwicklung.html

You might split the work with your exercise partner, so each of you reads only one text. The first source is fairly general, while the second (in German) is more specific and provides examples and is therefore longer.

2. Answer the following questions:
   a) What is the ideal TDD cycle? Which time frame should be aimed for?
   b) Test cases should ideally be designed according to which criterion?
   c) How many test cases should ideally be rewritten at the same time?
   d) Why is it not recommended to write more code than the test case demands?
   e) Why is it possible to write tests that run (rather than fail) at first go? Which question should then be asked?
   f) What can you assume when your test cases contain a lot of program logic of their own?
   g) What is refactoring?
   h) When and how does refactoring interact ideally with TDD?
   i) What is the relation between test cases produced via TDD and a specification?
   j) Ward Cunningham argues that test driven development is no test technique. What does he mean?
Task 2-2: First use of TDD

The aim of this task is to gain practical experience with TDD and to reflect on it.

1. Develop a Java program to solve the problem “Why Johnny Can’t Count” (see next page). Adhere to the following two rules:
   - *Pair Programming:* Program the entire code as a pair on one computer. Take turns with your partner in using mouse and keyboard.
   - *TDD:* Develop the entire code in a test-driven manner.

Bring the executable program, source code, and all test cases to the tutorial. Be prepared to present both your solution and solution process.

In case you do not possess a laptop, contact the tutor before 2015-10-23 12:00 (zieris@inf.fu-berlin.de).

2. Now, evaluate your programming session in preparation for the tutorial. In particular, think about the following questions.
   a) How did the development start (analysis / requirement elicitation, planning, decision for first test case, etc.)?
   b) When did you find it difficult to define new tests? Why?
   c) In which situations did you and your partner have different ideas concerning the next tests? Why?
   d) When did you change all tests again? Why?
   e) Were there situations in which (implicit) requirements were recognized too late? Why?
   f) When did you refactor? Why?

Why Johnny Can’t Count*

Description
Poor Johnny: He can hardly count. Johnny needs a program to “spell out” numbers into their equivalent English text. For example, the number 109210 is read in English as: “one hundred and nine thousand, two hundred and ten”. To make the program easier, Johnny is willing to accept the following compromises:

1. The program will be given a single positive integer less than a million.
2. No need to print any punctuation marks.
3. Use singular words, not plural. For example: “thousand” rather than “thousands”.
4. Don’t use the word “and” in the phrase. For example, instead of converting the number 102 into “one hundred and two”, all you need to do is convert it to “one hundred two”. Similarly, 109210 would be spelled out as: “one hundred nine thousand two hundred ten”.

On the other hand, Johnny hates spelling mistakes. The output should be spelled correctly and using only small letters. The list of allowable words in the output is:

zero one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen twenty thirty forty fifty sixty seventy eighty ninety hundred thousand

Input Format
The input consists of one positive integer.

Output Format
The given input number followed by a colon ‘:’, a single space, and the spelled-out number.

Examples

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>14: fourteen</td>
</tr>
<tr>
<td>112</td>
<td>112: one hundred twelve</td>
</tr>
<tr>
<td>199</td>
<td>199: one hundred ninety nine</td>
</tr>
<tr>
<td>123456</td>
<td>123456: one hundred twenty three thousand four hundred fifty six</td>
</tr>
<tr>
<td>421000</td>
<td>421000: four hundred twenty one thousand</td>
</tr>
<tr>
<td>999999</td>
<td>999999: nine hundred ninety nine thousand nine hundred ninety nine</td>
</tr>
</tbody>
</table>

*Simplified version of https://icpcarchive.ecs.baylor.edu/external/25/2553.pdf