**CodeAfterCoverageLaunch.docx**

**Source code and unit test case suite with annotation after coverage launch of unit test suite**

**NumberNamer.java (With source code annotation)**

**package** exercises.numbernamer;

**import** java.util.Scanner;

/\*\*

\* Reads in sentences, and expands

\* any numbers (if surrounded by whitespace) into English words.

\*

\* **@author** David Matuszek

\* **@author** KLM

\*/

**public** **class** NumberNamer {

/\*\*

\* Asks the user to enter sentences, then scans each sentence

\* entered looking for numbers. Any number in the range 0 to 9999

\* is expanded into English words. However, to simplify the

\* problem, only numbers surrounded by whitespace, or the beginning

\* or end of the line, are expanded; this is because the Scanner

\* method hasNextInt() will return true only for such numbers.

\* <br /><br />Special cases:

\* <ul><li>For any number larger than 9999, only the last four digits

\* will be considered.</li>

\* <li>Negative numbers are translated as "???".</li>

\* <li>Numbers larger than Integer.MAX\_VALUE are left as is.</li>

\* </ul>

\*

\* **@param** args Unused.

\*/

**public** **static** **void** main(String[] args) {

**new** NumberNamer().run();

}

/\*\*

\* Reads input, up to the five-character sequence "\*end\*",

\* converts numbers in the range 0 to 9999 to English words,

\* and prints the converted input.

\*/

**void** run() {

System.*out*.println("Enter sentences:");

Scanner scanner = **new** Scanner(System.*in*);

String token = "";

**int** count = 0;

**while** (!token.equals("\*end\*")) {

**if** (++count > 20) **break**;

**if** (scanner.hasNextInt()) {

**int** number = scanner.nextInt();

token = nameOf(number);

} **else** **if** (scanner.hasNextLine()) {

token = scanner.next();

}

**if** (token.equals("\n")) System.*out*.println(" EOL ");

System.*out*.print(token + " ");

}

}

/\*\*

\* Converts the given integer, if in the range 0 to 9999, to

\* English words. The method works properly only for numbers

\* in this range.

\*

\* **@param** number A number in the range 0 to 9999, inclusive.

\* **@return** English words representing the number.

\*/

String nameOf(**int** number) {

**int** ones, tens, hundreds, thousands;

String result = "";

**boolean** started = **false**;

**int** lastTwoDigits = number % 100;

ones = number % 10;

number = number / 10;

tens = number % 10;

number = number / 10;

hundreds = number % 10;

number = number / 10;

thousands = number % 10;

// handle the thousands digit, if present

**if** (thousands > 0) {

result = nameOfDigit(thousands) + " thousand";

started = **true**;

}

// handle the hundreds digit, if present

**if** (hundreds > 0) {

**if** (started) {

result = result + " ";

}

result = result + nameOfDigit(hundreds) + " hundred";

started = **true**;

}

// handle the last two digits, if between 10 and 19, and return

**if** (lastTwoDigits > 9 && lastTwoDigits < 20) {

**if** (started) {

result = result + " ";

}

**return** result + nameOfTeens(lastTwoDigits);

}

**assert** lastTwoDigits <= 9 || lastTwoDigits >= 20; // not a teen

// handle the tens digit, if present

**if** (tens > 0) {

**if** (started) {

result = result + " ";

}

result = result + nameOfTensDigit(tens);

started = **true**;

}

// handle the ones digit, if present

**if** (ones > 0 || !started) {

**if** (started) {

result = result + " ";

}

result = result + nameOfDigit(ones);

}

**return** result;

}

/\*\*

\* Returns the name of the given number, which must be

\* in the range 0 to 9, inclusive.

\*

\* **@param** digit The digit to be named.

\* **@return** The name of the digit.

\*/

String nameOfDigit(**int** digit) {

**switch** (digit) {

**case** 0:

**return** "zero";

**case** 1:

**return** "one";

**case** 2:

**return** "two";

**case** 3:

**return** "three";

**case** 4:

**return** "four";

**case** 5:

**return** "five";

**case** 6:

**return** "six";

**case** 7:

**return** "seven";

**case** 8:

**return** "eight";

**case** 9:

**return** "nine";

}

**return** "???";

}

/\*\*

\* Returns the English word for the given number, which

\* must be in the range 10 to 19, inclusive.

\*

\* **@param** number A number between 10 and 19, inclusive.

\* **@return** The English word that represents the given number.

\*/

String nameOfTeens(**int** number) {

**switch** (number) {

**case** 10:

**return** "ten";

**case** 11:

**return** "eleven";

**case** 12:

**return** "twelve";

**case** 13:

**return** "thirteen";

**case** 14:

**return** "fourteen";

**case** 15:

**return** "fifteen";

**case** 16:

**return** "sixteen";

**case** 17:

**return** "seventeen";

**case** 18:

**return** "eighteen";

**case** 19:

**return** "nineteen";

}

**return** "???";

}

/\*\*

\* Returns the English word for the given digit when in

\* the tens position.

\*

\* **@param** digit The tens digit to be named.

\* **@return** the English word for (10 \* the digit).

\*/

String nameOfTensDigit(**int** digit) {

**switch** (digit) {

**case** 0:

**return** "";

**case** 1:

**return** "ten";

**case** 2:

**return** "twenty";

**case** 3:

**return** "thirty";

**case** 4:

**return** "forty";

**case** 5:

**return** "fifty";

**case** 6:

**return** "sixty";

**case** 7:

**return** "seventy";

**case** 8:

**return** "eighty";

**case** 9:

**return** "ninety";

}

**return** "???";

}

**NumberNamerTest.java (With source code annotation)**

**package** exercises.numbernamer;

**import** org.junit.Before;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** NumberNamerTest {

**private** NumberNamer namer;

@Before

**public** **void** setUp() **throws** Exception {

namer = **new** NumberNamer();

}

@Test

**public** **final** **void** testNameOfDigitZero() {

*assertEquals*("zero", namer.nameOf(0));

}

@Test

**public** **final** **void** testNameOfDigit() {

*assertEquals*("one", namer.nameOf(1));

*assertEquals*("two", namer.nameOf(2));

*assertEquals*("three", namer.nameOf(3));

*assertEquals*("four", namer.nameOf(4));

*assertEquals*("five", namer.nameOf(5));

*assertEquals*("six", namer.nameOf(6));

*assertEquals*("seven", namer.nameOf(7));

*assertEquals*("eight", namer.nameOf(8));

*assertEquals*("nine", namer.nameOf(9));

}

@Test

**public** **final** **void** testNameOfTeens() {

*assertEquals*("eleven", namer.nameOf(11));

*assertEquals*("twelve", namer.nameOf(12));

*assertEquals*("thirteen", namer.nameOf(13));

*assertEquals*("fourteen", namer.nameOf(14));

*assertEquals*("fifteen", namer.nameOf(15));

*assertEquals*("sixteen", namer.nameOf(16));

*assertEquals*("seventeen", namer.nameOf(17));

*assertEquals*("eighteen", namer.nameOf(18));

*assertEquals*("nineteen", namer.nameOf(19));

}

@Test

**public** **final** **void** testNameOfTensDigit() {

*assertEquals*("ten", namer.nameOf(10));

*assertEquals*("twenty", namer.nameOf(20));

*assertEquals*("thirty", namer.nameOf(30));

*assertEquals*("forty", namer.nameOf(40));

*assertEquals*("fifty", namer.nameOf(50));

*assertEquals*("sixty", namer.nameOf(60));

*assertEquals*("seventy", namer.nameOf(70));

*assertEquals*("eighty", namer.nameOf(80));

*assertEquals*("ninety", namer.nameOf(90));

}

@Test

**public** **final** **void** testNameOfHundreds() {

*assertEquals*("one hundred", namer.nameOf(100));

*assertEquals*("two hundred", namer.nameOf(200));

*assertEquals*("three hundred", namer.nameOf(300));

*assertEquals*("four hundred", namer.nameOf(400));

*assertEquals*("five hundred", namer.nameOf(500));

*assertEquals*("six hundred", namer.nameOf(600));

*assertEquals*("seven hundred", namer.nameOf(700));

*assertEquals*("eight hundred", namer.nameOf(800));

*assertEquals*("nine hundred", namer.nameOf(900));

}

@Test

**public** **final** **void** testNameOfThousands() {

*assertEquals*("one thousand", namer.nameOf(1000));

*assertEquals*("two thousand", namer.nameOf(2000));

*assertEquals*("three thousand", namer.nameOf(3000));

*assertEquals*("four thousand", namer.nameOf(4000));

*assertEquals*("five thousand", namer.nameOf(5000));

*assertEquals*("six thousand", namer.nameOf(6000));

*assertEquals*("seven thousand", namer.nameOf(7000));

*assertEquals*("eight thousand", namer.nameOf(8000));

*assertEquals*("nine thousand", namer.nameOf(9000));

}

@Test

**public** **final** **void** testNameOfAssortedNumbers() {

*assertEquals*("twenty three", namer.nameOf(23));

*assertEquals*("ninety nine", namer.nameOf(99));

*assertEquals*("two hundred ten", namer.nameOf(210));

*assertEquals*("four hundred one", namer.nameOf(401));

*assertEquals*("six hundred seventeen", namer.nameOf(617));

*assertEquals*("seven hundred forty three", namer.nameOf(743));

*assertEquals*("eight hundred sixty one", namer.nameOf(861));

*assertEquals*("nine hundred six", namer.nameOf(906));

*assertEquals*("one thousand", namer.nameOf(1000));

*assertEquals*("two thousand one", namer.nameOf(2001));

*assertEquals*("three thousand ten", namer.nameOf(3010));

*assertEquals*("four thousand nineteen", namer.nameOf(4019));

*assertEquals*("five thousand one hundred", namer.nameOf(5100));

*assertEquals*("six thousand one hundred six", namer.nameOf(6106));

*assertEquals*("seven thousand one hundred twenty", namer.nameOf(7120));

*assertEquals*("eight thousand eleven", namer.nameOf(8011));

*assertEquals*("nine thousand nine hundred ninety nine", namer.nameOf(9999));

}

@Test

**public** **final** **void** testNameOfAssortedTrimmedNumbers() {

assertTrimmedEquals("twenty three", namer.nameOf(23));

assertTrimmedEquals("ninety nine", namer.nameOf(99));

assertTrimmedEquals("two hundred ten", namer.nameOf(210));

assertTrimmedEquals("four hundred one", namer.nameOf(401));

assertTrimmedEquals("six hundred seventeen", namer.nameOf(617));

assertTrimmedEquals("seven hundred forty three", namer.nameOf(743));

assertTrimmedEquals("eight hundred sixty one", namer.nameOf(861));

assertTrimmedEquals("nine hundred six", namer.nameOf(906));

assertTrimmedEquals("one thousand", namer.nameOf(1000));

assertTrimmedEquals("two thousand one", namer.nameOf(2001));

assertTrimmedEquals("three thousand ten", namer.nameOf(3010));

assertTrimmedEquals("four thousand nineteen", namer.nameOf(4019));

assertTrimmedEquals("five thousand one hundred", namer.nameOf(5100));

assertTrimmedEquals("six thousand one hundred six", namer.nameOf(6106));

assertTrimmedEquals("seven thousand one hundred twenty", namer.nameOf(7120));

assertTrimmedEquals("eight thousand eleven", namer.nameOf(8011));

assertTrimmedEquals("nine thousand nine hundred ninety nine", namer.nameOf(9999));

}

**private** **void** assertTrimmedEquals(String expected, String actual) {

*assertEquals*(expected, actual.trim());

}

}