Vladimir de Oliveira Henrique vladimirohenrique@yahoo.com.br

The approach used is based on a subtractive method which takes a valid integer and maps it to the corresponding Roman numerals using a foreach loop to step through the array and, if the value is greater than the parameter passed, the numeral is added to a string and removed from the parameter to carried on through the loop. Reverse conversion is achieved by iteratively converting chunks of the Roman numeral string to integers and subtracting, using the flipped mapping array of integers to characters. Error handling includes checking for integer and Roman character validity and also the specified integer range.

Assumptions:

- There is only one Roman numeral representation for any number.

- Only integers are supported (strings will result in null).

Caveats:

- Integers must be between 1-3999.

/\* This is the original interface provided for converting between integers and Roman numerals.

\*/

interface RomanNumeralGenerator

{

public function generate($integer); // convert from int -> roman

public function parse($string); // convert from roman -> int

}

// The class to implement this interface is as follows.

class NumericConversion implements RomanNumeralGenerator

{

const INT\_TYPE = "integer";

const ROM\_TYPE = "Roman numeral";

const CHARS\_PERMITTED = "/[MDCLXVI]/"; *// defines permissible characters for Roman numerals*

const MIN = 1; *// minimum integer value*

const MAX = 3999; *// maximum integer value*

*// Conversion mapping array between integers and Roman numerals.*

public $intToRomMap = array(

"1" => "I"

"4" => "IV",

"5" => "V",

"9" => "IX",

"10" => "X",

"40" => "XL",

"50" => "L",

"90" => "XC",

"100" => "C",

"400" => "CD",

"500" => "D",

"900" => "CM",

"1000" => "M",);

public $RomToIntMap = array\_flip($this->intToRomMap); *// reverse conversion definition with array flip*

// *Error messages to be displayed upon validation check*

public $errorhandling = array(

0 => "Valid integer = false",

1 => "Valid Roman num. = false"

2 => "Between 1-3999 = false",);

public function generate($integer) // *integer🡪Roman function*

{

$result = '';

if($this->intCheck($integer)) // *valid integer check*

{

foreach($this->intToRomMap as $units => $map)

{

$result .= $this->getRoman($map, floor($integer/$units));

$integer %= $units;

} // *maps each integer to string of Roman numerals*

}

else

{

$result = $this->handleInputError(self::INT\_TYPE, $integer);

}

return $result; // *returns Roman numerals*

}

public function parse($string) // *Roman 🡪 integer function*

{

$result = 0;

$Roman = $string;

$match = preg\_grep(self::CHARS\_PERMITTED, (str\_split($Roman)), PREG\_GREP\_INVERT);

if(count($match) > 0) // *valid numeral check*

{

$result = $this->handleInputError(self::ROM\_TYPE);

return $result;

}

// *Uses an iterative subtractive method to convert string*

foreach ($RomToIntMap as $units => $map) {

while (strpos($Roman, $units) === 0) {

$result += $map;

$Roman = substr($Roman, strlen($units));

}

}

// *Validation check for integer range compliance*

if(!$this->intCheck($result))

{

$result = $this->handleInputError(self::INT\_TYPE, $result);

}

return $result; // *returns integer*

}

// *Protected functions*

*/\* Determines repetitive conversion iterations, @param string $map mapped numeral / character; @param int $integer number of times to repeat character; @return string resultant string of characters*

*\*/*

protected function getRoman($map, $integer)

{

$result = '';

while($integer--)

{

$result .= $map;

}

return $result;

}

*\\ Boolean integer check*

protected function intCheck($integer)

{

if(!$integer || $integer < self::MIN || $integer > self::MAX)

{

return false;

}

else

{

return true;

}

}

*\\ Error handling to call specific error message*

protected function handleInputError($inputType, $integer = null)

{

$msg = 'unknown error';

switch($inputType)

{

case(self::INT\_TYPE):

if(!$integer)

{

$msg = $this->errorMessages[0];

}

break;

case(self::ROM\_TYPE):

$msg = $this->errorMessages[1];

if($integer < self::MIN || $integer > self::MAX)

{

$msg = $this->errorMessages[2];

}

break;

}

//throw new Exception($msg);

return $msg;

}

}