# README

## Assumptions

A few assumptions have been made:

1. If the generate method is given a value of less than 1 or greater than 3,999 it should throw an exception as numbers outside of that range are not valid Roman numerals.
2. If the parse method is given a null or blank string it should throw an exception.

## Approach

The following list details the approach I took to generating the service, all code was written with TDD in mind (no production code written before the tests):

* Generated the webapp project using: mvn archetype:generate -DgroupId=bbc -DartifactId=numerals -DarchetypeArtifactId=maven-archetype-webapp
* Added JUnit, Hamcrest and Mockito dependencies (for unit testing, useful assets and mock objects (plus verification) respectively).
* Added test class for the Roman numeral generator implementation.
  + Created the given interface and added an empty implementation class.
  + Inside generator implementation class, added unit tests for common cases, e.g.: 1=I, 5=V, 10=X, 4=IV, 9=IX, etc. for both the parse and generate methods.
  + Added additional error cases, for example: parse has to be given a string so if given a null or blank string it will throw an exception. Generate doesn’t accept values less than 1 and greater than 3,999 so throw exceptions.
* Filled out generate method implementation running the unit tests after each change to ensure tests were passing. Used an array to hold the Roman Numerals and their associated values. Once all tests were passing for the generate method:
  + Refactored method to make it less complex.
  + Reran tests to ensure they passed.
* Created implementation of parse method:
  + Using separate arrays of Roman Numerals and their values (i.e., special cases like CM, CD, XC were listed first).
  + Method works by iterating through array and:
    - Counting number of times the Roman numeral exists in the parse string then times that by the numeral’s value and add that value to a total.
  + Reran unit tests each time I made a change to ensure I hadn’t broken existing functionality.
* Once all unit tests were passing:
  + Refactored need for two separate lists of Roman Numerals:
    - Added enumeration that listed the Roman numeral and its value.
    - Added methods to return a list of numerals in either parse or generate format.
    - Reran unit tests to ensure they still passed.
* Added dependency on Spring and Spring test library (used Spring annotations over XML configuration).
* Added two unit test classes for Roman numeral web-service controller: simple unit test to ensure routing methods were working and an integration unit test class (to ensure added routes were working and they returned the correct value (using XPath)).
* Created a RomanNumeralController class and filled out implementation ensuring new unit tests were passing. Added two routes: GET: numerals/parse/:roman\_numeral and GET: numerals/generate/:arabic\_numeral
  + Created a domain model to return either a Roman or Arabic numeral (encapsulated in XML).
* As the parse/generate methods can generate exceptions if given bad data, decided to handle this using Spring exception resolvers and return a useable HTTP status code:
  + Added new integration test class to handle giving the web-service bad data (giving the generate method less than 1 or greater than 3,999). Added test cases for this to ensure service returned a code of 400 (bad data).
  + Created an exception resolver test class that verified that the response status code was set to 400 (bad data).
  + Provided exception resolved class implementation.
  + Reran all unit tests to ensure they passed.
  + Added log4j as a dependency and logger the exception in the exception resolver class.
* Ran unit tests with code coverage enabled: 100% class, 90% method and 96% line covered (missing coverage is for the default constructors that are needed for JAXB to work but are never used, given protected status).
* Finally added Jackson-xc library to pom for JSON support (set Content-type header to application/json for JSON output).

## Caveats

Behavioural tests haven’t been written for the service.

Using the Wikipedia page linked I could have created some acceptance criteria and using Ruby and Cucumber written this service using BDD followed by TDD.