FISA – Prueba Técnica

Realizada por: Hernan Ludena

Fecha: 28 de enero de 2024

Contenido

1. Problemática

Decidiste abandonar la Tierra después del último colapso financiero que dejó al 99.99% de la población de la Tierra con el 0.01% de la riqueza. Afortunadamente, con la escasa suma de dinero que queda en tu cuenta, puedes permitirte alquilar una nave espacial, dejar la Tierra y volar por toda la galaxia para vender metales comunes y tierra (que aparentemente vale mucho).

Comprar y vender en toda la galaxia requiere que conviertas números y unidades, y decidiste escribir un programa para ayudarte.

Los números utilizados para transacciones intergalácticas siguen una convención similar a los números romanos, y has recopilado cuidadosamente la traducción adecuada entre ellos.

Los números romanos se basan en siete símbolos:

Símbolo Valor
I 1
V 5
X 10
L50
C 100
D 500
M 1,000

Los números se forman combinando símbolos y sumando los valores. Por ejemplo, MMVI es 1000 + 1000 + 5 + 1 = 2006. Generalmente, los símbolos se colocan en orden de valor, comenzando con los valores más grandes. Cuando los valores más pequeños preceden a los valores más grandes, los valores más pequeños se restan de los valores más grandes, y el resultado se suma al total. Por ejemplo, MCMXLIV = 1000 + (1000 − 100) + (50 − 10) + (5 − 1) = 1944.

Los símbolos "I", "X", "C", y "M" pueden repetirse tres veces seguidas, pero no más. (Pueden aparecer cuatro veces si el tercero y el cuarto están separados por un valor más pequeño, como XXXIX). "D", "L", y "V" nunca pueden repetirse.

"I" solo se puede restar de "V" y "X". "X" solo se puede restar de "L" y "C" solo. "C" solo se puede restar de "D" y "M". "V", "L" y "D" nunca se pueden restar.
Solo un símbolo de valor pequeño se puede restar de cualquier símbolo de valor grande.
Un número escrito en números arábigos se puede descomponer en dígitos. Por ejemplo, 1903 se compone de 1, 9, 0 y 3.
Para escribir el número romano, cada uno de los dígitos diferentes de cero debe tratarse por separado. En el ejemplo anterior, 1,000 = M, 900 = CM y 3 = III. Por lo tanto, 1903 = MCMIII.
-- Fuente: Wikipedia (http://en.wikipedia.org/wiki/Roman_numerals)

La entrada a tu programa consta de líneas de texto que detallan tus notas sobre la conversión entre unidades intergalácticas y números romanos.

Se espera que manejes de manera apropiada las consultas inválidas.

Test input:
glob is I
prok is V
pish is X
tegj is L
glob glob Silver is 34 Credits
glob prok Gold is 57800 Credits
pish pish Iron is 3910 Credits
how much is pish tegj glob glob ?
how many Credits is glob prok Silver ?
how many Credits is glob prok Gold ?
how many Credits is glob prok Iron ?
how much wood could a woodchuck chuck if a woodchuck could chuck wood ?

Test Output:
pish tegj glob glob is 42
glob prok Silver is 68 Credits
glob prok Gold is 57800 Credits
glob prok Iron is 782 Credits
I have no idea what you are talking about

2. Desarrollo

2.1 Análisis

Para resolver el problema de convertir números intergalácticos a números romanos en Java se plantea crear un programa que permita lo siguiente:

1. Interpretar las palabras alienígenas y convertirlas a números romanos.

- glob is I → glob = 1
- prok is V → prok = 5
- pish is X → pish = 10
- tegj is L → tegj = 50

2. Convertir los números romanos a números arábigos.

- I: 1
- V: 5
- X: 10
- L: 50
- C: 100
- D: 500
- M: 1,000

3. Calcula el valor de los commodities/mercancías/materiaPrima(como Oro, Plata, Hierro) en Créditos.

- glob glob Silver is 34 Credits indica que 2 globs (o sea, 2) de Plata valen 34 Créditos. Por lo tanto, 1 unidad de Plata = 34 / 2 = 17 Créditos.
- glob prok Gold is 57800 Credits indica que 1 glob + 1 prok (o sea, 1 + 5 = 6) de Oro valen 57800 Créditos. Así, 1 unidad de Oro = 57800 / 6 = 9633.33 Créditos.
- pish pish Iron is 3910 Credits indica que 2 pish (o sea, 20) de Hierro valen 3910 Créditos. Entonces, 1 unidad de Hierro = 3910 / 20 = 195.5 Créditos.

4. Responder a las preguntas específicas proporcionadas en el archivo de entrada.

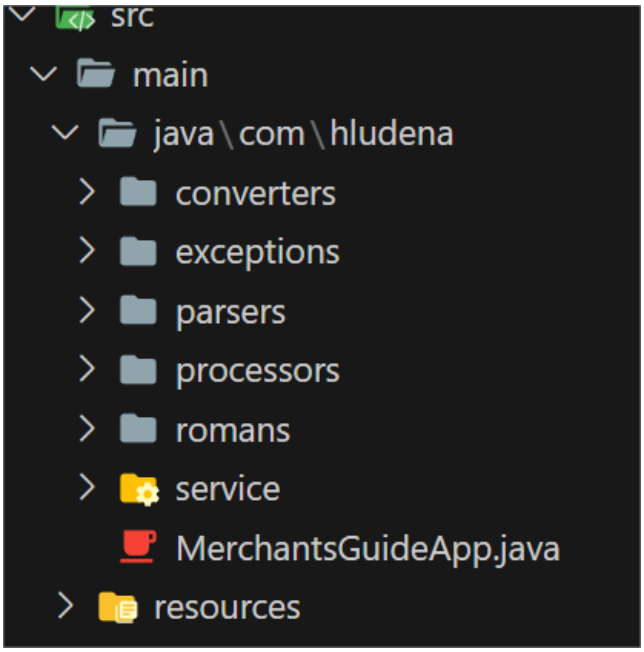
- how much is pish tegj glob glob? → 10 + 50 + 1 + 1 = 62.
- how many Credits is glob prok Silver? → 1 + 5 = 6 de Plata = 6 * 17 = 102 Créditos.
- how many Credits is glob prok Gold? → 1 + 5 = 6 de Oro = 6 * 9633.33 = 57799.98, que se redondea a 57800 Créditos.
- how many Credits is glob prok Iron? → 1 + 5 = 6 de Hierro = 6 * 195.5 = 1173 Créditos.

- how much wood could a woodchuck chuck if a woodchuck could chuck wood ? have no idea what you are talking about

2.2 Solución

Se plantea crear un programa que contenga utilitarios para el manejo de los números romanos, así como diccionarios de datos, reglas de conversión de números romanos, conversores de lenguaje y procesadores de preguntas.

Estructura de paquetes y clases:

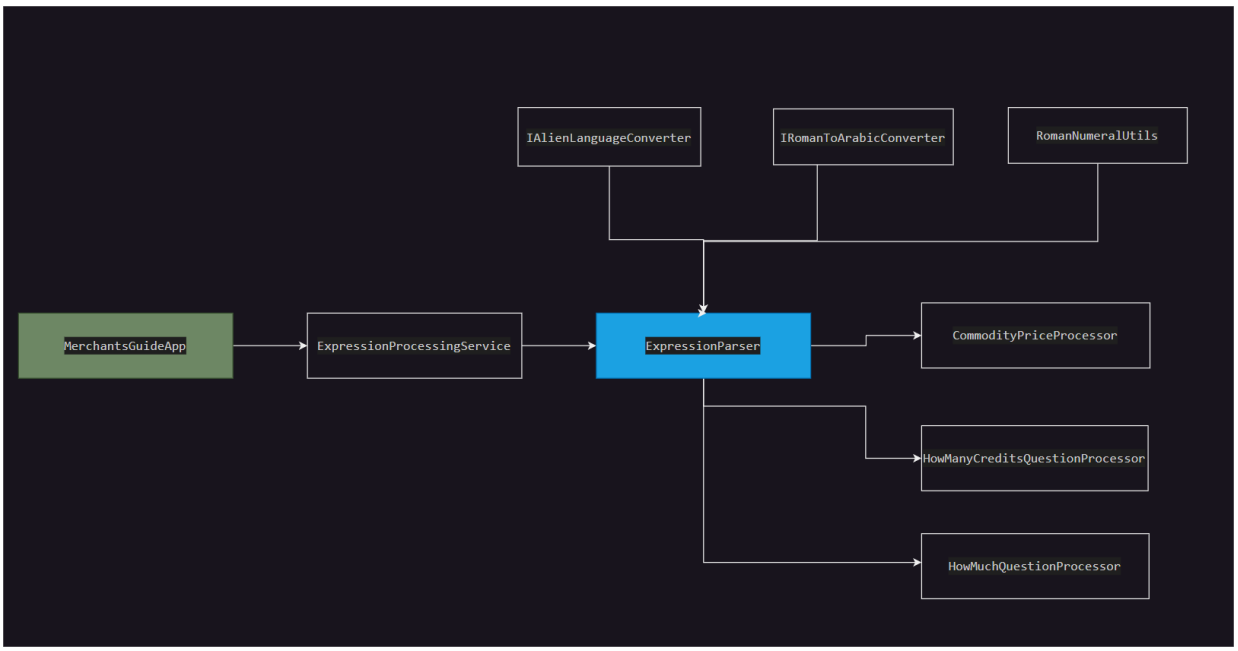


MerchantsGuideApp	App Principal	
converters	IAlienLanguageConverter AlienLanguageConverter IRomanToArabicConverter RomanToArabicConverter	Interfaces y clases para convertir números alienígenas a números romanos y luego en números arábigos.
exceptions	IAlienLanguageConverter	Es una excepción personalizada que se lanza cuando se detecta una entrada inválida en la aplicación.

parsers	ExpressionParser	Se encarga de analizar y procesar diferentes tipos de expresiones o consultas. Dependiendo del formato de la línea de entrada, delega la tarea a diferentes procesadores o conversores.
processors	CommodityPriceProcessor HowManyCreditsQuestionProcessor HowMuchQuestionProcessor	Clases que se encargan de procesar líneas de texto que contienen información sobre precios de mercancías en un lenguaje alienígena. Y se encargan de procesar y responder preguntas sobre el valor total en créditos de una cantidad de mercancía, expresada en un idioma alienígena.
romans	RomanNumeral RomanNumeralRules RomanNumeralUtils	Proporciona utilidades para trabajar con números romanos. Incluye una representación de los numeros romanos y funciones para validar cadenas como numeros romanos.
service	ExpressionProcessingService	Inicializa las dependencias necesarias para el procesamiento de expresiones, incluyendo convertidores de lenguaje y numerales romanos, y el analizador de expresiones.
resources	InpuText.txt	Contiene el archivo con los datos de entrada

2.3 Diseño arquitectónico Ato Nivel

En el diagrama adjunto, se evidencia, las diferentes dependencias con las que cuenta el programa, el uso de interfaces para los conversores, un utilitario para manejo de números romanos, y los procesadores de mercancías y preguntas



2.4 Estilo Programación

Para solventar el problema se realizo las siguientes buenas prácticas:

- Se utilizo Interfaces para establecer contratos sobre los conversores de lenguaje alienígena a romano y luego a número arábigo. Diferentes implementaciones de estas interfaces pueden ser intercambiadas sin cambiar el código que depende de ellas haciendo de esta forma uso del Polimorfismo.
- Se creo un Utilitario de números romanos utilizando el **Patron Singleton** y obtener una instancia.
- Se utilizo el Patrón de **Inyección** de **dependencias** por constructor, delegando la creación de las clases a un Clase Servicio, para tener un bajo acoplamiento de clases.

- Se realizo una abstracción de clases según el requerimiento planteado, pero se puede realizar abstracciones más profundas donde se haga uso de Objetos y más interfaces.
- Uso de principios SOLID como la Responsabilidad Única (Single Responsibility Principle) y la Inversión de Dependencias (Dependency Inversion Principle), ambos centrales en la OOP moderna

2.5 Requerimientos de Instalación

java	openjdk version "17.0.9" 2023-10-17
maven	Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Visual Studio Code	V 1.85.2
Windows 11	Sistema Operativo

2.5.1 Ejecución

Al ser un **código listo para producción** debe ejecutarse directo desde la terminal usando los comandos:

- **Para ejecución directa:** - Descargar proyecto de Gitbuh: https://github.com/hernanludena/Merchants-Guide - Ubicarse en el path principal del proyecto, carpeta dist donde el artefacto generado: Ejemplo C:\develop\Merchants-Guide\dist - Levantar la aplicación indicando el path del archivo de entrada	java -jar .\target\MerchantsGuideToTheGalaxyM-1.0-SNAPSHOT.jar ./src/main/resources/InputText.txt Ejecucion desde Windows, si se desea ejecutar el jar desde otro Sistema Operativo tomar en consideración el ajuste de los slash / para el mapeo del archivo de entrada
- **Para validar las pruebas unitarias se** debe **construir el proyecto** - Abrir con un IDE y desde la terminal construir artefacto usando Maven.	mvn clean package

Salida Esperada:

```
PS C:\develop\java\IDEA\fisa\MerchantsGuideToTheGalaxyM\dist> java -jar .\target\MerchantsGuideToTheGalaxyM-1.0-SNAPSH
OT.jar ./src/main/resources/InputText.txt
Error: Unable to access jarfile .\target\MerchantsGuideToTheGalaxyM-1.0-SNAPSHOT.jar
PS C:\develop\java\IDEA\fisa\MerchantsGuideToTheGalaxyM\dist>
```

1.1 Pruebas Unitarias

1.1.1 Uso de Junit 5 y Mockito

Se realizan pruebas unitarias de los Conversores y Procesadores.

Cada prueba tiene su documentación correspondiente sobre el código.