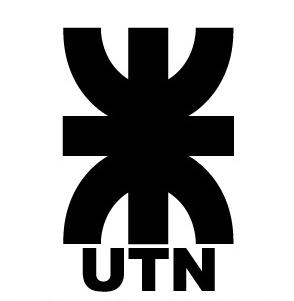
******

***Universidad Tecnológica Nacional - Facultad Regional Buenos Aires***

**Sintaxis y semántica de   
los lenguajes**

**2023**

**Trabajo Práctico Grupal N° 1**

**Tema: Trabajo práctico Teórico**

**Curso: K2055**

**Profesor: Roxana Leituz**

**Fecha Estipulada de Entrega: 25/08/2023**

**Grupo N°34**

| **Alumnos** | **Legajo** | **GitHub** |
| --- | --- | --- |
| Julian Franco Sola | 172.877-5 | https://github.com/Julifsola/SSL |
| Fernando Aquino | 176.149-3 | https://github.com/Fernando-23/SSL |

## **GitHub grupal:** [**https://github.com/SSL-Grupo34/SSL-Grupo34**](https://github.com/SSL-Grupo34/SSL-Grupo34)

## **Índice**

## Historia de Go…………………………………………….. 2

## Historia de Java………………………………………….. 2

## BNF de Go………………………………………………… 4

## BNF de Java………………………………………………. 4

## Comparación de rendimiento de Go…………………… 6

## Comparación de rendimiento de Java………………… 7

## 

## **Lenguajes elegidos:**

Go y Java

## **Historia de Go:**

Go, a menudo referido como "Golang", es un lenguaje de programación moderno desarrollado por Google. Fue creado por un equipo de ingenieros de Google (en su creación participó Ken Thompson, creador del algoritmo de Thompson) y fue anunciado públicamente en 2009. El objetivo principal detrás de la creación de Go era abordar algunas de las deficiencias y complejidades percibidas en otros lenguajes de programación, especialmente en lo que respecta a la eficiencia, la concurrencia y la legibilidad del código. Go ha ganado una gran popularidad en la comunidad de programadores debido a su enfoque pragmático y sus características útiles. Se ha utilizado en una variedad de aplicaciones, desde desarrollo web hasta sistemas distribuidos y desarrollo de herramientas. Los principales objetivos de diseño de Go incluyen:

1. **Simplicidad y Legibilidad**: Go se esfuerza por ser simple y fácil de entender, con una sintaxis clara y concisa.
2. **Eficiencia**: Go está diseñado para ser eficiente tanto en términos de tiempo de ejecución como en consumo de recursos. Sus características de compilación estática y recolección de basura eficiente contribuyen a su desempeño.
3. **Concurrencia**: Una de las características más destacadas de Go es su soporte incorporado para la concurrencia y la paralelización. El modelo de goroutines y canales facilita la escritura de programas concurrentes y escalables.
4. **Compilación Rápida**: Go se compila rápidamente, lo que facilita un ciclo de desarrollo más rápido y menos tiempos de espera.
5. **Seguridad**: Go se esfuerza por evitar errores comunes de programación, como punteros nulos y buffer overflow, mediante el diseño de su sistema de tipos y otras características.

Go ha sido adoptado por muchas empresas y proyectos notables, incluidos Docker, Kubernetes (Hacen mucho uso de la concurrencia nativa para las operaciones que utilizan) y la infraestructura interna de Google.

## **Historia de Java:**

Java se creó en Sun Microsystems en 1991 con el nombre "OAK" por un grupo llamado *green team*, posteriormente se cambió el nombre por problemas legales, y finalmente con la denominación actual JAVA.

El objetivo de java era crear un lenguaje de programación parecido a C++ en estructura y sintaxis, fuertemente orientado a objetos, pero con una máquina virtual propia. Esto se hizo bajo el principio de poder ser usado bajo cualquier arquitectura (“Write Once, Run Anywhere”).

En 1992 se presenta el proyecto verde, con los prototipos a bajo nivel. Entre 1993 y 1994 se trabaja para poder presentar un prototipo funcional (hotJava) donde se ve todo el potencial que JAVA puede ofrecer.

En 1995, es presentada la versión alpha de java, y un año después en 1996 Sun fundó el grupo empresarial JavaSoft para que se encargará del desarrollo tecnológico. Dos semanas más tarde la primera versión de Java fue publicada.

## 

## **BNF de Go:**

| BNF | Código |
| --- | --- |
| <Program> ::= <PackageClause> <ImportDecl> <TopLevelDecl>\*  <PackageClause> ::= "package" <PackageName>  <ImportDecl> ::= "import" ( <ImportSpec> | "(" { <ImportSpec> } ")" )  <ImportSpec> ::= [ "." | <PackageName> ] <ImportPath>  <TopLevelDecl> ::= <Declaration> | <FunctionDecl> | <MethodDecl>  <Declaration> ::= <ConstDecl> | <TypeDecl> | <VarDecl>  <ConstDecl> ::= "const" ( <ConstSpec> | "(" { <ConstSpec> } ")" )  <TypeDecl> ::= "type" ( <TypeSpec> | "(" { <TypeSpec> } ")" )  <VarDecl> ::= "var" ( <VarSpec> | "(" { <VarSpec> } ")" )  <FunctionDecl> ::= "func" <FunctionName> <Signature> <FunctionBody>  <MethodDecl> ::= "func" <Receiver> <MethodName> <Signature> <FunctionBody>  <Signature> ::= <Parameters> [ <Result> ]  <Parameters> ::= "(" [ <ParameterList> [ "..." ] ] ")"  <Result> ::= <Parameters> | <Type>  <ParameterList> ::= <ParameterDecl> | <ParameterList> "," <ParameterDecl>  <ParameterDecl> ::= [ <IdentifierList> ] [ "..." ] <Type> | package main  import (  "fmt"  "strings"  )  const pi = 3.14159  type Circle struct {  radius float64  }  var num int = 42  func add(x, y int) int {  return x + y  }  func (c Circle) area() float64 {  return pi \* c.radius \* c.radius  } |

## **BNF de Java:**

| BNF | Código |
| --- | --- |
| <compilationUnit> ::= <packageDeclaration>? <importDeclaration>\* <typeDeclaration>\*  <packageDeclaration> ::= "package" <packageName> ";"  <importDeclaration> ::= "import" <qualifiedName> ";"  <typeDeclaration> ::= <classDeclaration> | <interfaceDeclaration>  <classDeclaration> ::= ( "public" | "abstract" | "final" )? "class" <className>  ( "extends" <superClassName> )? ( "implements" <interfaceName> ( "," <interfaceName> )\* )?  "{" <classBodyDeclaration>\* "}"  <classBodyDeclaration> ::= <fieldDeclaration> | <methodDeclaration> | <constructorDeclaration>  <fieldDeclaration> ::= ( "public" | "protected" | "private" | "static" )? <type> <variableDeclarators> ";"  <methodDeclaration> ::= ( "public" | "protected" | "private" | "static" | "abstract" | "final" )?  ( <type> | "void" ) <methodName> <parameters> <methodBody>  <constructorDeclaration> ::= <modifiers>? <constructorName> <parameters> <constructorBody>  <modifiers> ::= ( "public" | "protected" | "private" | "static" | "abstract" | "final" )+ | package com.example;  import java.util.ArrayList;  import java.util.List;  public class MyClass {  private int number;  public static final double PI = 3.14159;  public MyClass(int number) {  this.number = number;  }  public int getNumber() {  return number;  }  public void setNumber(int number) {  this.number = number;  }  public static void main(String[] args) {  List<String> fruits = new ArrayList<>();  fruits.add("Apple");  fruits.add("Orange");  System.out.println(fruits);  }  } |

# 

# **Análisis de rendimiento de concurrencia**

Fuente:  
<https://arthuracosta.medium.com/benchmarking-golang-and-java-concurrency-performance-analysis-for-file-consumption-563105ae028b>

Algoritmo a ejecutar:

Procesamiento de un CSV de aproximadamente 100.000 líneas, realizando la lectura por bloques e imprimiendo las líneas en el proceso.

Implementación de Go:

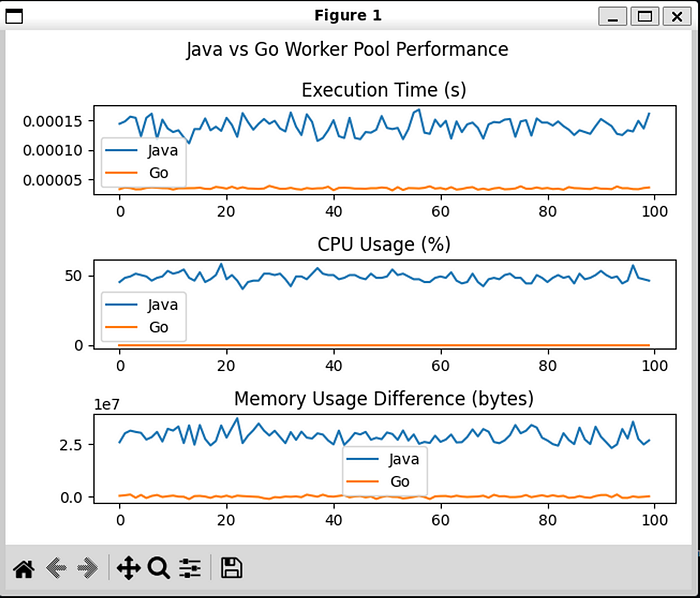
Se especifican la cantidad de hilos (Workers) que va a tener el proceso a ejecutar, una parte del proceso se va a encargar de leer los bloques de información del CSV y junto con una administrador de hilos llamado Worker Pool Manager se va a dividir y asignar el número de tareas en los hilos especificados previamente. Finalmente se recolecta la información procesada por los hilos y se almacena en una lista común.

Implementación de Java:

En Java se utiliza la estrategia de ThreadPool que ayuda a alcanzar el mismo objetivo planteado en Go, en este caso es necesario utilizar algunas estrategias para evadir condiciones de carrera a la hora de consumir bloques de la lista de información.

Resultado:

Al correr los programas correspondientes se crean dos archivos, cada uno ligado a su programa que contiene los resultados de tiempo de ejecución, uso de CPU y manejo de memoria dentro de 100 iteraciones de la misma implementación y ejecución.



Conclusión:

La ventaja de Java es que cuenta con JVM, esto interpretará el código y mejorará la performance a la hora de ejecutar.

Las ventajas de Go son: Implementación nativa de hilos liviano (Java usa hilos pesados, es decir que consumen más tiempo ejecutarlos) y un Recolector de Basura también nativo que permite un uso más eficiente de la memoria heap

Las observaciones que se pueden concluir son:

1. Tanto Java como Go demuestran un nivel de eficiencia muy alto pero Go tiene una ventaja demostrando un mayor manejo de tareas en un corto periodo de tiempo.
2. El uso de CPU indica un manejo de recursos computacionales muy eficiente con mínimo overhead debido a la manera en que Go está programado.
3. El uso de memoria no varía entre iteraciones de la ejecución del proceso lo que habla de la estabilidad.

# **Análisis de programa de factorial**

Fuente: <https://medium.com/@radadiyasunny970/a-simple-performance-test-and-difference-go-v-s-java-e6f29ad65293>

Algoritmo a ejecutar: Un programa de factoriales de números a fin de comparar el tiempo entre los 2 lenguajes.

Se calcularán los factoriales de 10000, 50000, 100000, 500000 y 1000000.

Go   
<https://gist.github.com/sunnyRK/ab646e1d1bc47969b0aa85707bb01058#file-forloop-go>

Java

<https://gist.github.com/sunnyRK/aa26097567dac5e8df24566cd409a180#file-factorial-java>

Resultados:

| **Factorial** | **Tiempo Go** | **Tiempo Java** |
| --- | --- | --- |
| 10000 | 0.03 seg. | 0.112 seg. |
| 50000 | 0.41 seg. | 1.185 seg. |
| 100000 | 2.252 seg. | 2.252 seg. |
| 500000 | 68.961 seg. | 89.500 seg. |
| 1000000 | 224.135 seg. | 385.868 seg. |

Resultado:

Se puede apreciar que Go es más rápido que Java al realizar estas entradas e incluso esta diferencia aumenta más al ser el número más alto.

Esta diferencia se debe porque Go es compilado a código máquina y ejecutado directamente, lo que lo hace mucho más rápido que java ya que, esta misma, utiliza una máquina virtual para correr el código (Es decir, lleva un proceso mucho más pesado en términos de cómputo que Golang).

Golang también es bueno en el manejo de memoria, lo cual es crucial en los lenguajes de programación. Un detalle es que no usa referencias sino punteros.

Conclusión:

Golang no es un reemplazo de Java, pero si cuenta con ciertas ventajas que pueden hacer que elijamos uno sobre otro. Golang, al ser uno de los más rápidos en tiempos de ejecución, es entendible porque es adoptado cada vez más.