Tarea Independiente 28/07/2025

David Núñez Franco

July 30, 2025

Inventario de Conceptos Claves

- Paradigma Computacional
- Rol de la IA en Diagrama Realidad-Humano-Modelo
- Meta-Relaciones: is-A, ako, has-A
- Meta-Lógica de is-A, ako, has-A (símbolos : y ¡:)
- Relación dinámica versus estática en lenguajes de programación (PL)
- Lenguajes de Programación (PL): Sintaxis versus Semántica
- Arquitectura de Compilador para un PL
- Máquina Virtual vs Real
- Portabilidad
- Analizador sintáctico
- Analizador semántico (estático)
- Error de sintaxis versus error semántico estático versus error semántico dinámico
- shell de sistema operativo (cmd). Variable de entorno PATH.
- Tools: node y jshell

Trivia (extra-curricular)

> ¿Desde la llegada de Colón a "América": ¿cuántos años pasaron antes de que la corona española reconociera que los nativos de "América" eran seres humanos? Fue ese reconocimento efectivo en la aplicación?

Respuesta

Legalmente, los indígenas de América fueron reconocidos como seres humanos, libres, con alma y derechos desde el año 1500, por orden de la Reina Isabel La Católica, quien los declaró vasallos de la Corona y prohibió su esclavización. La Corona Española promulgó leyes —como las Leyes de Burgos (1512) y las Leyes Nuevas (1542)— para proteger a los pueblos indígenas de los abusos coloniales.

En el ámbito eclesiástico, el reconocimiento universal de la humanidad de los indígenas llegó en 1537 con la bula Sublimis "Deus", emitida por el papa Pablo III. Dado que Cristóbal Colón llegó a América en 1492, pasaron 45 años hasta que se emitió esta declaración papal. Sin embargo, el reconocimiento legal por parte de la Corona Española ocurrió mucho antes, apenas ocho años después del descubrimiento.

Ahora, aunque la Corona Española reconoció legalmente a los indígenas como seres humanos, libres y con derechos desde el año 1500, ese reconocimiento no fue efectivo en la práctica. A pesar de leyes como las de Burgos y las Nuevas, y de la bula Sublimis Deus (1537), muchos colonizadores ignoraron estas normas y continuaron explotando y maltratando a los indígenas. Factores como la distancia con España, el interés económico, el racismo y la falta de control efectivo hicieron que la realidad en América estuviera muy lejos de lo que dictaban las leyes. Así, aunque el reconocimiento fue adelantado para su tiempo, su aplicación fue limitada e insuficiente.

Referencia:

Martínez-Fornés, A. (2024, octubre 2). *Lo que hizo la Corona española para proteger a los indígenas de América. El Debate*. Recuperado de
`https://www.eldebate.com/espana/casa-real/20241002/hizo-corona-espanola-proteger-indiger`
`231994.html`

JIT COMPILATION (punto 0)

La compilación Just-In-Time (JIT) es una técnica utilizada en algunos entornos de ejecución, en la que el código fuente o intermedio se traduce a código máquina en tiempo de ejecución, es decir, mientras el programa se está ejecutando, en lugar de hacerlo de forma previa como en la compilación tradicional (Ahead-Of-Time).

Esta estrategia busca combinar la rapidez inicial que ofrece la interpretación con el rendimiento optimizado de la compilación nativa.

La compilación JIT es una técnica que busca el equilibrio entre rapidez de arranque y rendimiento a largo plazo. Al analizar el código en tiempo de ejecución y compilar solo las partes necesarias, ofrece una solución eficaz y flexible para aplicaciones que necesitan tanto agilidad como eficiencia.

errores.jsh (punto 1)

script a testear

```
class MedioTransporte{}
class Automovil extends MedioTransporte{}
List<MedioTransporte> la = null
var lb = new ArrayList<Automovil>()
lb.add(new Automovil())
la = lb
```

Listing 1: Script jshell errores.jsh

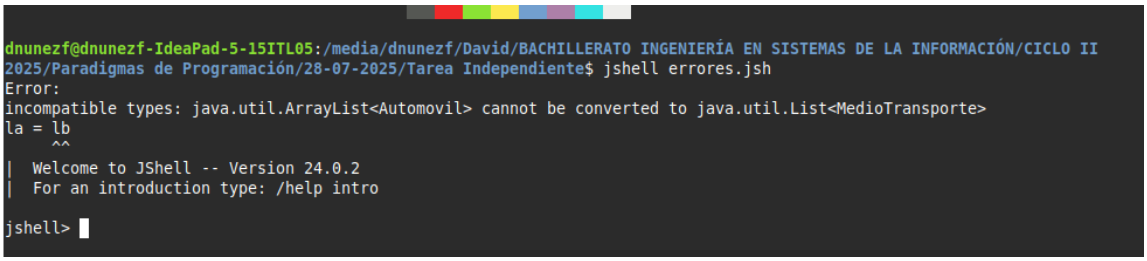

Figure 1: Mensaje error script jshell

Justificación del error

Cuando ejecutamos el script, dará un error de tipo semántico estático (detectado en tiempo de compilación). Es un error semántico estático porque el compilador de Java detecta un problema de tipos incompatibles en tiempo de compilación.

El error ocurre porque, aunque Automovil hereda de MedioTransporte, una lista de Automovil no es una lista de MedioTransporte.

Stack¡T¿ && Operators (punto 2)

Haga una clase Stack¡T¿ en un paquete paradigms.operations.

Stack permite size, top, pop, push, isEmpty definidas de la manera usual en cualquier pila. Hay Excepcions en donde corresponde. Además nuestro Stack¡T¿ tiene un método T operate(Operator oper) que se explica adelante, que es para ejecutar la operación oper usando el contenido de la pila. Se va a pedir un diseño que Ud. debe seguir. Primero obligatoriamente. Luego cuestionarlo y mejorarlo.

a. Cree Operator como un enum que tiene ciertas operaciones. Por ejemplo ADD, MINUS, MULT y DIV.

b. Cree una clase Operators que tiene un método int arityOf(Operator oper) que devuelve el número de operandos del Operator oper. Con esto: cuando se llama a void Stack::operate(oper) sucede lo siguiente: Se le pide a Operators el arity de oper, digamos que es n. Se sacan (pop) de la pila n objetos v_n, v_n-1, ..., v1 y dependiendo de oper se la hace la operación respectiva y el resultado se pone de vuelta en pila. Por ejemplo, si oper es ADD y n es su arity entonces se calcula v1

+ v2 + ... + v_n y ese resultado se pone (push) en la pila de vuelta. Se manejan excepciones

c. Cree una clase de prueba paradigms.utils.test.TestStack que usando lo anterior crea y prueba casos de prueba. Su proyecto se puede compilar y probar desde una consola (fuera de cualquier editor o IDE). No se revisa de otra forma.

Solución al problema planteado en Java

```java
package paradigms.operations;

// Este enum define las operaciones basicas
public enum Operator {
    ADD,
    MINUS,
    MULT,
    DIV
}
```

Listing 2: Clase Operator

```java
package paradigms.operations;

/*Nos va a permitir consultar cuantos operandos requiere
   cada operacion*/
public class Operators {

    public static int arityOf(Operator oper) {
        switch (oper) {
            case ADD:
            case MINUS:
            case MULT:
            case DIV:
            return 2;
            default:
            throw new IllegalArgumentException("Operator not
                implemented: " + oper);
        }
    }
}
```

Listing 3: Clase Operators

```java
package paradigms.operations;

import java.util.LinkedList;
import java.util.NoSuchElementException;

```

```java
/*Para efectos de este ejercicio, debemos crear nuestra propia clase Stack<T> generica,
* "Haga una clase Stack<T> en un paquete paradigms.operations". Por lo tanto, no podemos
* utilizar java.util.Stack; asi, recurrimos a LinkedList<T>, ya que proporciona todos los
* metodos necesarios para una pila.*/
public class Stack<T> {
    private LinkedList<T> elements;

    public Stack() {
        elements = new LinkedList<>();
    }

    /*Cantidad de elementos*/
    public int size() {
        return elements.size();
    }

    /*Indica si esta vacia*/
    public boolean isEmpty() {
        return elements.isEmpty();
    }

    /*Consulta el tope*/
    public T top() {
        if (isEmpty()) {
            throw new NoSuchElementException("Stack is empty - cannot top");
        }
        return elements.getFirst();
    }

    /*Inserta un elemento*/
    public void push(T item) {
        elements.addFirst(item);
    }

    /*Saca y retorna el tope*/
    public T pop() {
        if (isEmpty()) {
            throw new NoSuchElementException("Stack is empty - cannot pop");
        }
        return elements.removeFirst();
    }

```

```
48          /*Aplica una operacion aritmetica, definida en Operator,
                sobre los elementos de la pila.
49          * 1. Consulta cuantos operandos se necesitan, usando
                Operators.arityOf()
50          * 2. Hace pop a los operandos de la pila.
51          * 3. Aplica la operacion aritmetica (solo soportando
                Integer).
52          * 4. Inserta el resultado de vuelta en la pila y retorna
                .*/
53          public T operate(Operator oper) {
54              // Consultamos la aridad (cantidad de operandos que
                    requiere la operacion)
55              int arity = Operators.arityOf(oper);
56
57              // Validamos que hayan sificientes elementos en la
                    pila
58              if (size() < arity) {
59                  throw new IllegalStateException("No hay
                        suficientes elementos en la pila para esta
                        operacion: " + oper);
60              }
61
62              // Extraemos los operandos desde la pila (Last In,
                    First Out)
63              T[] operands = (T[]) new Object[arity];
64              for(int i = arity - 1; i >= 0; i--) {
65                  operands[i] = pop(); // Se guardan en orden
                        inverso, para operar correctamente
66              }
67
68              /*Asegurar que el tipo T es Integer*/
69              if(!(operands[0] instanceof Integer)) {
70                  throw new UnsupportedOperationException("Only
                        Integer operations are supported.");
71              }
72
73              /*Realizar la operacion, segun el tipo*/
74              int result = (Integer) operands[0];
75              for(int i = 1; i < arity; i++) {
76                  int value = (Integer) operands[i];
77
78                  switch (oper) {
79                      case ADD:
80                      result += value;
81                      break;
82                      case MINUS:
83                      result -= value;
```

```
                    break;
                    case MULT:
                    result *= value;
                    break;

                    case DIV:
                    if (value == 0) {
                        throw new ArithmeticException("Division
                            by zero");
                    }
                    result /= value;
                    break;

                    default:
                    throw new UnsupportedOperationException("
                        Unsupported operation: " + oper);
                }
            }

            // Hacemos push al resultado devuelta en la pila
            T resultWrapped = (T) Integer.valueOf(result);
            push(resultWrapped);
            return resultWrapped;
        }
    }
```

Listing 4: Clase Stack

```
    package paradigms.utils.test;

    import paradigms.operations.Operator;
    import paradigms.operations.Stack;

    public class TestStack {
        public static void main(String[] args) {
            Stack<Integer> stack = new Stack<>();

            System.out.println("=== Test de pila con operaciones
                ===");

            //Push de algunos elementos
            stack.push(10);
            stack.push(5);
            System.out.println("Pila despues de push(10), push
                (5): tamano = " + stack.size());

            //Operacion de suma (10 + 5)
            Integer resultado = stack.operate(Operator.ADD);
```

```
19                  System.out.println("Resultado de ADD: " + resultado)
                        ;
20
21                  // Push de mas elementos
22                  stack.push(3);
23                  stack.push(2);
24                  System.out.println("Pila despues de push(3), push(2)
                        : tamano = " + stack.size());
25
26                  // Multiplicacion (3 * 2)
27                  resultado = stack.operate(Operator.MULT);
28                  System.out.println("Resultado de MULT: " + resultado
                        );
29
30                  // Resta (15 - 6)
31                  resultado = stack.operate(Operator.MINUS);
32                  System.out.println("Resultado de MINUS: " +
                        resultado);
33
34                  System.out.println("Top final de la pila: " + stack.
                        top());
35
36                  // Probamos division entre cero
37                  try {
38                      stack.push(0);
39                      stack.operate(Operator.DIV);
40                  }
41                  catch (Exception e) {
42                      System.out.println("Division entre cero: " + e.
                            getMessage());
43                  }
44
45                  // Probamos operar sin suficientes elementos
46                  try{
47                      stack = new Stack<>();
48                      stack.push(1);
49                      stack.operate(Operator.ADD);
50                  } catch (Exception e) {
51                      System.out.println("Falta de operandos: " + e.
                            getMessage());
52                  }
53              }
54          }
```

Listing 5: Clase tipo test TestStack