Erwin Harder Garzón

Camilo Ruiz López

Programación 3 Universidad del Quindío

PREGUNTAS

1. Elixir se ejecuta sobre Erlang, ¿qué es Erlang y que características tiene?
2. ¿Qué ventajas tiene elixir en usar elixir en lugar de Erlang?

(distribuido, concurrente, resiliente, velocidad, fácil de usar, actualización de código en vivo, metaprogramación/DSD)

1. ¿Porque deberías aprender a usar elixir?
2. ¿En qué tipo de proyectos es ideal usar elixir?

RESPUESTAS

1R. Erlang es un lenguaje de programación creado en 1980 el cual fue desarrollado principalmente para sistemas de telecomunicaciones, hoy en día, se usa principalmente para aplicaciones que requieran una alta disponibilidad y resistencia a posibles fallos en el software, esto para permitir que el sistema siga funcionando incluso si parte de ellos llegan a tener fallas o errores como tal.

Características: Una característica muy llamativa e importante de erlang la cual es por la que más llama la atención es porque nos brinda un modelo muy simple y sencillo, pero a su vez potente para la contención de errores y soporte para los fallos. Tambien que es un lenguaje concurrente y distribuido, esto que quiere decir? Al ser concurrente nos permite realizar múltiples tareas o procesos a la vez y la distribución nos servirá para algo muy parecido a la concurrencia, nos permitirá realizar múltiples tareas al mismo tiempo pero en diferentes dispositivos.

2R. Las ventajas consisten en varias, primeramente, acerca de la sintaxis, Elixir permite una mucho más moderna y expresiva que Erlang, contiene funciones como coincidencia de patrones, metaprogramación y polimorfismo, lo que concede más legibilidad y flexibilidad para los desarrolladores. En caso contrario, Erlang tiene una sintaxis más tradicional que podría ser difícil para los principiantes adaptarse a el. Posteriormente, esta la programación concurrente se refleja en Elixir permitiendo herramientas fáciles de usar para los desarrolladores, aprovechando subprocesos livianos para el paralelismo.

3R. Un desarrollador eficiente hoy en dia debería aprender a usar Elixir porque permite realizar programación funcional y en bajo nivel, lo que es muy útil en cierto tipo de situaciones. Además, cuenta con la disponibilidad de conjuntos de código fáciles de usar, es sencillo de aprender y usar. Cuenta con reglas sintácticas claras y el código final es muy legible y orientado a procesos. Otro beneficio es entrenar al cerebro a confrontar y realizar resolución de desafíos cuyos beneficios son muy valiosos y van más allá de las necesidades de tu puesto de trabajo. Para agregar, también proporciona información para ser un mejor programador en otros lenguajes, nos sirve para crear aplicaciones escalables y fáciles de mantener, esto permite un mayor tráfico de datos y además una buena optimización en el almacenamiento de estos mismos. Para finalizar los procesos fallan de manera aislada lo que hace que aplicaciones que requieran de alta disponibilidad y dinero no se vean tan afectadas.

4R. Elixir, al ser un lenguaje de programación con concurrencia y distribuido, lo hace perfecto para poder manejar fallos mientras el sistema sigue funcionando de forma perfecta, por ende, es perfecto para proyectos con alta concurrencia y muchísima escalabilidad ya que, a pesar de que el sistema pueda llegar a recibir muchas entradas, este de igual forma lograria funcionar en perfecto estado. proyectos como aplicaciones en tiempo real, sistemas embebidos, desarrollo web, sistemas de telecomunicaciones, aplicaciones de sistemas de datos, entre otros.

Observe los siguientes dos programas y conteste:

¿Qué hacen?, ¿Son equivalentes? y ¿Cuál es más legible?

1. Repita el proceso con un archivo llamado "*termina\_en-01.exs*" y el código

IO.puts(String.ends\_with?(String.upcase("Hola Mundo"), "UNDO"))

1. Repita el proceso con un archivo llamado "*termina\_en-02.exs*" y el código

"Hola Mundo"

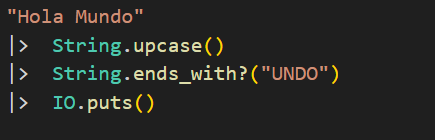
|> String.upcase()

|> String.ends\_with?("UNDO")

|> IO.puts()

Preguntas elixir

Código 1:



¿Qué hace?

Lo que hace es devolver “true” en la consola por medio de un sistema de paréntesis, el de más adentro, que está más anidado propone poner todo lo que esta en la cadena a mayúsculas en este caso “Hola Mundo” que devolvería “HOLA MUNDO”, luego el paréntesis que le sigue pregunta si lo que esta adentro de la cadena de texto termina en “UNDO” lo cual es verdad, por esa razón devuelve true

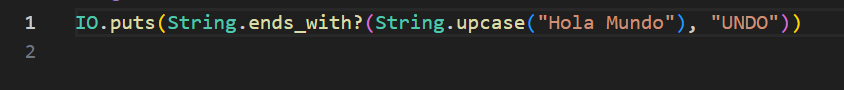
¿Son equivalentes?

Con respecto al otro código, si, son equivalentes primeramente porque ambos llevan a lo mismo el devolver “true” solo que el segundo código propone una estructura más separada por medio de pipe (tuberías) las cuales permiten concatenar o encapsular partes de código reemplazando los paréntesis del código anterior

¿Cuál es más legible?

El segundo código porque se ven la estructura del código separada y por así decirlo cada acción/método del código más fácil de visualizar

Código 2:



¿Qué hace?

Igual que el anterior código, devuelve “true” en la consola, este funciona poniendo en el string la palabra “Hola Mundo” y lo pasa a la siguiente línea de código por medio del Upcase, esto para que la palabra quede en mayúsculas y asi en la siguiente línea que verifica si el string termina en UNDO sea verdadero ya que al pasar la palabra a mayúscula queda como “HOLA MUNDO”, haciendo que este devuelva “true”.

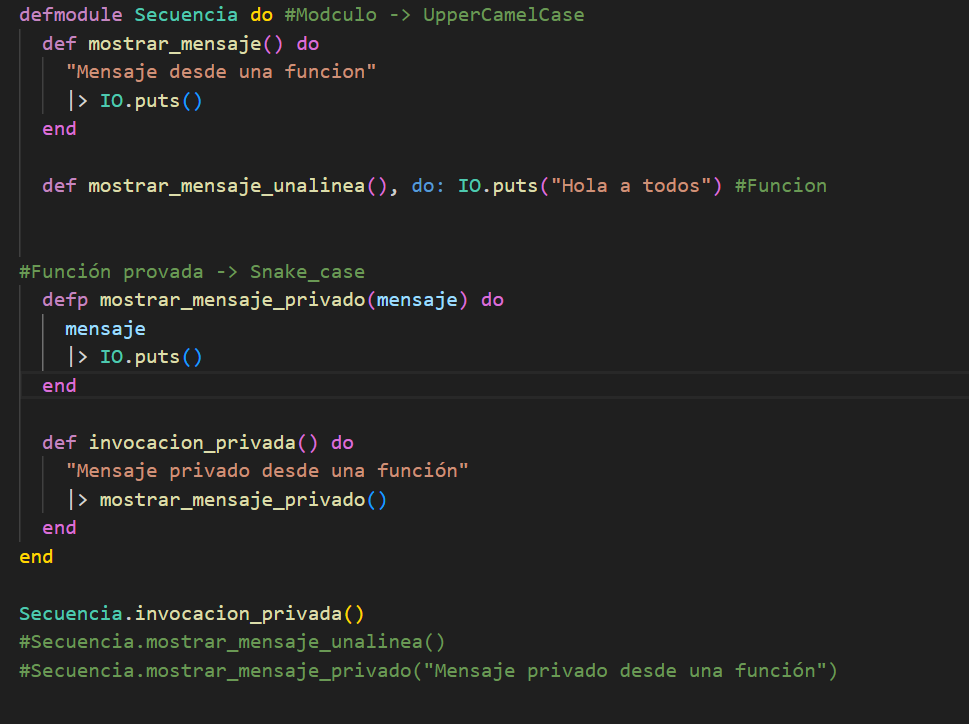
¿Son equivalentes?

Si, los dos códigos son equivalentes ya que ambos devuelven “true”, solo que el otro código hace todo en una misma línea mientras que este prefiere hacerlo en una estructura mas separada lo cual a mi parecer, lo hace más legible y fácil de entender.

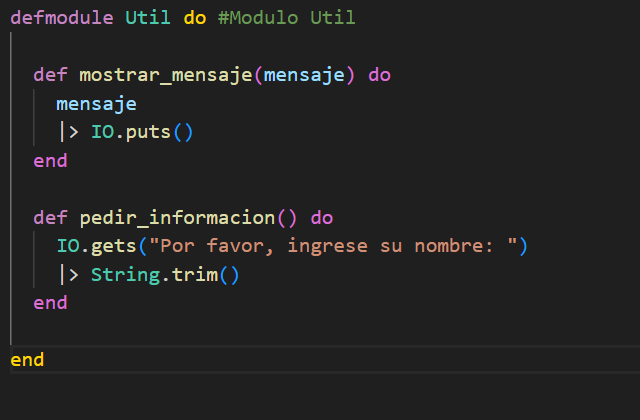
¿Cuál es más legible?

El segundo código es muchísimo más legible porque tiene una estructura mas separada y fácil de entender con solo leer las líneas de código y entender un poco el lenguaje, haciendo que este sea mucho más sencillo a la hora de entenderlo.

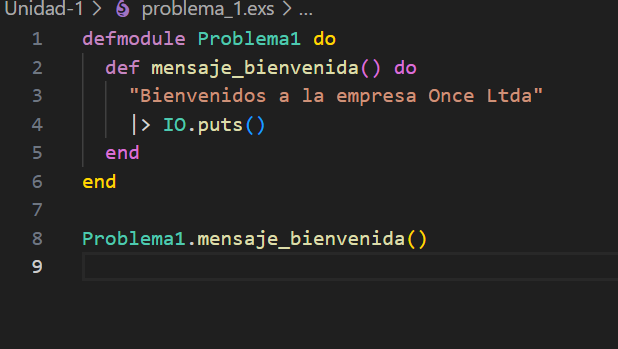
Código secuencial:



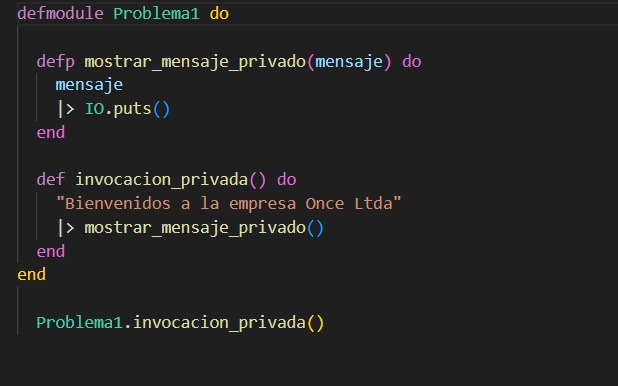
UTIL, compilado:



Problema 1 Forma de codigo publica empresa “ Once Ltda”



Forma de código privada



¿Cómo ejecutar Java desde Elixir con JOption?

Para ejecutar código Java desde Elixir, usando JOptionPane para la entrada y salida, puedes utilizar la librería java\_embed de Elixir. Esta librería permite invocar código Java desde Elixir y pasar datos entre ambos lenguajes.

Pasos generales:

Instalar java\_embed: Agrega java\_embed a tu lista de dependencias en mix.exs.

Compilar código Java: Compila tu clase Java que contiene la lógica de JOptionPane.

Crear un puente Java: En Elixir, crea un puente a tu clase Java usando Java.import/1.

Llamar a métodos Java: Invoca los métodos de tu clase Java desde Elixir usando Java.call/3.

Manejar datos: Convierte los datos entre Elixir y Java según sea necesario.

Ejemplo detallado:

1. Código Java (MiClase.java):

Java

import javax.swing.JOptionPane;

public class MiClase {

public static String mostrarDialogo(String mensaje) {

return JOptionPane.showInputDialog(null, mensaje);

}

public static void mostrarMensaje(String mensaje) {

JOptionPane.showMessageDialog(null, mensaje);

}

}

2. Código Elixir (app.ex):

Código

defmodule App do

use Application

def start(\_type, \_args) do

java\_embed\_path = Path.join(:code.priv\_dir(:app), "java")

Java.start(java\_embed\_path)

import Java

# Importar la clase Java

Java.import("MiClase")

# Llamar a mostrarDialogo

nombre = Java.call("MiClase", "mostrarDialogo", "Ingresa tu nombre:")

Java.call("MiClase", "mostrarMensaje", "Hola, #{nombre}!")

{:ok, self()}

end

def stop(\_state) do

Java.stop()

:ok

end

end

3. Archivos y estructura del proyecto:

Código

app/

├── mix.exs

├── lib

│ └── app.ex

└── priv

└── java

└── MiClase.class # Compilado

4. Compilación del código Java:

Código

javac MiClase.java -d priv/java

5. Ejecución del código Elixir:

Código

iex -S mix

Explicación:

Java.start/1: Inicia el puente Java, especificando el directorio donde se encuentran los archivos .class.

Java.import/1: Importa la clase Java MiClase para que pueda ser usada desde Elixir.

Java.call/3: Llama a un método de la clase Java. El primer argumento es el nombre de la clase, el segundo el nombre del método, y los siguientes son los argumentos del método.

Path.join(:code.priv\_dir(:app), "java"): Obtiene la ruta al directorio priv/java, donde se guardarán los archivos .class compilados.

Java.stop(): Detiene el puente Java.

Tenemos otras opciones como las siguientes:

* + - 1. Usando System.cmd/3

# Compilar y ejecutar un archivo Java

{output, exit\_code} = System.cmd("java", ["-cp", ".", "MiClase"], cd: "/ruta/al/directorio")

# Si ya tienes un JAR compilado

{output, exit\_code} = System.cmd("java", ["-jar", "mi\_aplicacion.jar", "argumentos"])

* + - 1. Usando Ports

defmodule JavaRunner do

def ejecutar\_java(clase, args \\ []) do

port = Port.open({:spawn\_executable, System.find\_executable("java")}, [

{:args, ["-cp", ".", clase] ++ args},

{:cd, "/ruta/al/directorio"},

:binary,

:exit\_status,

{:line, 1024}

])

recibir\_salida(port, "")

end

defp recibir\_salida(port, acc) do

receive do

{^port, {:data, {:eol, line}}} ->

nuevo\_acc = acc <> line <> "\n"

recibir\_salida(port, nuevo\_acc)

{^port, {:exit\_status, status}} ->

{acc, status}

after

5000 -> {:error, :timeout}

end

end

end

# Uso

{salida, codigo\_salida} = JavaRunner.ejecutar\_java("MiClase", ["arg1", "arg2"])

* + - 1. Usando GenServer

defmodule JavaProcess do

use GenServer

def start\_link(clase\_java) do

GenServer.start\_link(\_\_MODULE\_\_, clase\_java, name: \_\_MODULE\_\_)

end

def enviar\_comando(comando) do

GenServer.call(\_\_MODULE\_\_, {:comando, comando})

end

def init(clase\_java) do

port = Port.open({:spawn\_executable, System.find\_executable("java")}, [

{:args, ["-cp", ".", clase\_java]},

:binary,

{:packet, 2}

])

{:ok, %{port: port}}

end

def handle\_call({:comando, comando}, \_from, %{port: port} = state) do

Port.command(port, comando)

receive do

{^port, {:data, respuesta}} ->

{:reply, respuesta, state}

after

5000 -> {:reply, {:error, :timeout}, state}

end

end

end

Cómo usar JOptionPane desde Elixir (vía JNI):

La forma de usar JOptionPane desde Elixir implica el uso de Java Native Interface (JNI). JNI permite a código nativo (como Elixir en este caso, que se ejecuta en la máquina virtual Erlang) llamar a funciones y clases de Java.

Pasos generales:

1.Crear una clase Java: Se crea una clase Java que contiene la lógica para interactuar con JOptionPane. Por ejemplo, una clase que muestre un mensaje o solicite un nombre.

2.Compilar la clase Java: Se compila la clase Java a un archivo .class y luego a un archivo .jar.

3.Crear un módulo Elixir: Se crea un módulo Elixir que actúa como puente entre Elixir y la clase Java.

4.Cargar la librería Java: Se usa el módulo Java.NativeInterface para cargar el archivo .jar.

5. Llamar a los métodos Java: Se utilizan las funciones del módulo Java.NativeInterface para invocar los métodos de la clase Java, incluyendo los de JOptionPane.

Ejemplo (esquemático):

Código

# Java (MyJavaClass.java)

public class MyJavaClass {

public static void showMessage(String message) {

javax.swing.JOptionPane.showMessageDialog(null, message);

}

}

Código

# Elixir (my\_elixir\_module.ex)

defmodule MyElixirModule do

use Java.NativeInterface

java\_import "MyJavaClass"

def show\_message\_from\_java(message) do

Java.call(MyJavaClass, :showMessage, [message])

end

end

Código

# Uso

MyElixirModule.show\_message\_from\_java("Hola desde Elixir con JOptionPane!")

Referencias.

*The Elixir programming language*, https://elixir-lang.org. Accessed 14 August 2025.

“CONCURRENT, PARALLEL AND DISTRIBUTED SYSTEMS – COMPUTER SCIENCE.” *computer science*, https://computingstudy.wordpress.com/concurrent-parallel-and-distributed-systems/. Accessed 14 August 2025.

Gillis, Alexander S. “What is an Embedded System? | Definition from TechTarget.” *TechTarget*, 13 August 2024, https://www.techtarget.com/iotagenda/definition/embedded-system. Accessed 14 August 2025.

“1 What is Erlang.” *Erlang*, https://www.erlang.org/faq/introduction.html. Accessed 14 August 2025.

“What is Elixir?” *BairesDev*, https://www.bairesdev.com/blog/what-is-elixir. Accessed 14 August 2025.