**Examen Teórico-Práctico de Java para Programadores Senior (Java 11 / Java 17)**

**Nombre:Ruben Caraveo Garcia**

**Fecha: 27/02/25 3:00 pm**

**Sección 1: Preguntas Teóricas**

* **¿Cuál es la diferencia entre var y un tipo explícito en Java 11 y Java 17?**

la principal es la inferencia y la legibilidad de codigo el var es mas corto pero menos claro y en casos como new ArrayList<> se infiere como arreglo de objetos y complica un poco las cosas por que no tiene seguridad de tipos y la ultima seria que var se debe inicializar de manera obligada y los tipos explicitos no siempre.

* Explica cómo se manejan las inferencias de tipo en Java 11 y Java 17 al utilizar var. ¿Hay diferencias significativas entre ambas versiones? Da ejemplos.

funciona igual, ya que var esta desde Java 10 y no ha cambiado en versiones posteriores lo unico es que java 11 lo metio en lamdas fuera de eso es igual.

* **Explique la diferencia entre un Record y una Class tradicional en Java.**

record modela datos inmutables y class cualquier tipo de modelado

record genera constructores getters equals to string de manera automatica

en record todos los campos son finals por defecto y class puede ser mutable o inmutable según se implemente

record es public por defecto y class es cualquier nivel de acceso

* ¿Qué ventajas ofrece un Record en comparación con una clase convencional? ¿En qué escenarios es preferible usar un Record?

que se escribe menos codigo y mayor seguridad en datos.

y los casos serian cuando necesitas un objeto solo para datos, la inmutabilidad es deseable, no necesitas herencia, solo necesitas getters y métodos básicos.

* **¿Qué es un sealed class en Java 17? ¿Cómo se usa y qué ventajas ofrece?**

restringe qué clases pueden heredar de ella, permite mas control sobre la jerarquía de clases y evita extensiones

* Explica cómo funcionan las clases selladas en Java 17 y proporciona un ejemplo de cómo se pueden utilizar para modelar jerarquías de clases.

public sealed class Persona permits Fisica, Moral {

public abstract double impuesto();

}

se usa la keyword sealed, la keyword permits para especificar la herencia y las subclases solo pueden ser final, sealed o non-sealed.

Clase base solo permite herencia a ciertas subclases --> Subclase final: No se puede extender más allá de esta clase

* **En Java 11, se introdujo la API HttpClient. Explica sus principales características. ¿Cómo se realiza una solicitud GET en un servidor usando la API HttpClient en Java 11?**

soporte nativo para HTTP/1.1 y HTTP/2, soporte para autenticación y cookies, manejo automático de redirecciones, soporte para WebSockets

HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder().uri(URI.create("URL del servidor")).get.build();

HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

String resp = response.statusCode();

* **¿Qué mejoras importantes introdujo Java 17 respecto a Java 11 en términos de rendimiento y nuevas características?**

optimiza el Garbage Collector, menor consumo de memoria y optimización en la compilación

mete las nuevas funcionalidades sealed class, pattern matching, record patterns, foreign function API

mejoras para compartir información y datos entre sí con código nativo y más compatibilidad con JVM

* Menciona al menos tres mejoras clave que afecten el rendimiento o la experiencia de desarrollo.

más expresividad en código, menos uso de memoria y mayor compatibilidad en Docker y Kubernetes

* **En Java, ¿cuál es la diferencia entre HashMap y ConcurrentHashMap?**

hash no es seguro threads y concurrent si, hash no sincroniza internament y concurrent sincroniza por segmento, concurrent bloquea por segmento hash no, hash la lectura no es segura y concurrent si.

* ¿En qué situaciones utilizarías cada uno de ellos? Explica cómo manejan la concurrencia.

hash: para aplicaciones de un solo hilo o donde la sincronización es manejada externamente, es muy rápido en un solo hilo pero no es seguro para hilos.

concurrent: para entornos multihilo donde se realizan muchas lecturas y escrituras simultáneas, con alta concurrencia, y tiene los métodos atómicos como putIfAbsent() y compute()

* **¿Cómo se implementan y gestionan los Streams en Java 11 y 17?**

se crean a partir de colecciones (como List, Set), arrays, o usando el método Stream.of()

ejemplo:

Stream<String> stream = lista.stream();

stream.forEach(System.out::println);

ejemplo mas pro:

lista.stream().filter(s -> s.startsWith("R")).map(String::toUpperCase).forEach(System.out::println);

* Explica el concepto de Streams en Java, destacando sus diferencias de uso en Java 11 y 17. Proporciona un ejemplo donde se utilicen operaciones de filtrado, transformación y agrupación de datos.

En java 17 se mejoro en el manejo de Collectors, el manejo de particiones y resultados paralelos, y los Streams paralelos

int suma = lista.parallelStream().mapToInt(Integer::intValue).sum();

* **Explica cómo funciona el sistema de módulos en Java 9 y su evolución hasta Java 17.**

para modularizar el código en Java que ayuda mucho para el mantenimiento y la organización de proyectos grandes.

ejemplo:

module com.ejemplo.modulo {

requires java.base; // Módulo requerido

exports com.ejemplo.paquete; // Paquete exportado

}

java 10: se introdujeron algunas optimizaciones internas y mejoras en el rendimiento general del JDK

java 11: tiene mejoras en la gestión de dependencias y la resolución de módulos

java 12: mejoras de estabilidad y rendimiento

java 17: mejoras en la modularización, optimización en la ejecución de aplicaciones modulares y mejoras en el manejo de dependencias

* ¿Cómo se maneja el acceso entre módulos? ¿Cómo afecta esto a la estructura de aplicaciones grandes y monolíticas?

con el archivo module-info.java, en una aplicación monolítica, todo el código suele estar en un único proyecto o conjunto de paquetes, lo que dificulta la comprensión y mantenimiento del sistema. Con el sistema de módulos, puedes dividir la aplicación en módulos más pequeños y bien definidos.

* **¿Qué es un Pattern Matching en Java 17 y cómo mejora la legibilidad del código?**

es una característica que permite una manera más legible de trabajar con tipos de datos, condiciones, y estructuras de control mejora la sentencia instanceof para comprobar el tipo de un objeto y luego hacer un casting explícito.

* Describe el concepto y la sintaxis de Pattern Matching en Java 17 y cómo ayuda a reducir el código repetitivo.

if (obj instanceof String) {

String str = (String) obj; // Cast explícito

System.out.println(str);

}

el mayor beneficio es que el tipo del objeto se verifica y se asigna a una variable en una sola línea.

**Sección 2: Ejercicios Prácticos**

**Ejercicio 1: CRUD de Cliente con Record y Pattern Matching**

**Objetivo:** Implementar un CRUD de clientes utilizando un Record para representar a los clientes y aplicar Pattern Matching para trabajar con sus tipos.

* **Crea un Record en Java 17** que represente a un **Cliente** con los siguientes atributos: id (String), nombre (String), email (String), edad (int), y tipoCliente (enum que puede ser REGULAR, VIP).
* **Implementa un CRUD** (Crear, Leer, Actualizar, Eliminar) para gestionar los clientes en memoria:
* Utiliza una List<Cliente> para almacenar los clientes.
* Crea métodos para:
* Crear un nuevo cliente.
* Obtener todos los clientes.
* Actualizar los datos de un cliente por su ID (por ejemplo, cambiar su email o su tipo de cliente).
* Eliminar un cliente por su ID.
* **Usa Pattern Matching** para realizar una acción específica si el cliente es de tipo VIP o REGULAR. Por ejemplo, un cliente VIP tiene un descuento en los préstamos, mientras que un cliente REGULAR no.

**Ejercicio 2: CRUD de Préstamos y Manejo de Intereses**

**Objetivo:** Implementar un CRUD para gestionar préstamos de clientes y aplicar diferentes tasas de interés según el tipo de cliente.

* **Crea una clase Prestamo** con los siguientes atributos:
* id (String), monto (double), clienteId (String), fecha (LocalDate), estado (enum con valores PENDIENTE, PAGADO).
* **Implementa un CRUD** para gestionar los préstamos:
* Crear un préstamo.
* Obtener todos los préstamos activos.
* Actualizar el estado de un préstamo (por ejemplo, cambiar de PENDIENTE a PAGADO).
* Eliminar un préstamo.
* **Calcula el interés**:
* Los clientes VIP tendrán una tasa de interés del 5% y los REGULAR tendrán una tasa del 10%.
* Implementa un método que calcule el monto total a pagar (monto + interés) en función del tipo de cliente.

Tomando en cuenta el planteamiento anterior, considerar los siguientes puntos.

* Agregar contrato de interfaz yaml con estándar OpenAPI 2.0.
* Pruebas Unitarias (mínimo un método/caso).
* Legibilidad del código.