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

**Nombre: Michel Sneyder Morales Camacho**

**Fecha: 27 de enero de 2025**

**Sección 1: Preguntas Teóricas**

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

En realidad el tipo var lo que permite es la inferencia de tipos, mientras que los explicitos no: por ejemplo

var lista = new Arraylist<String>(); // De esta forma se infiere como un Arraylist, lo cual permite que var tome el valor sin necesidad de hacerlo declarativo o explicito.

List <String> Lista = new Arraylist<>(); // Aqui se define de manera explicita.

1. **Explique la diferencia entre un Record y una Class tradicional en Java.**
   * ¿Qué ventajas ofrece un Record en comparación con una clase convencional? ¿En qué escenarios es preferible usar un Record?
   * Como tal Record se define como un tipo especial que permite crear clases inmutables(o que no pueden cambiar) por ejemplo:

Record Cliente(String id, String nombre, String apellido, String mail){}

De esta manera se genera un objeto con sus atributos y no hay necesidad de crear los getters y setters ya que automáticamente los genera

1. **¿Qué es un sealed class en Java 17? ¿Cómo se usa y qué ventajas ofrece?**
   * Explica cómo funcionan las clases selladas en Java 17 y proporciona un ejemplo de cómo se pueden utilizar para modelar jerarquías de clases.
   * Permite controlar las clases que se van a extender y tambien de donde extienden, por ejemplo:

public sealed class Figura permits Circulo, Cuadrado{}

public final class Circulo extends Figura{}

public final class Cuadrado extends Figura{}

1. **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?**

Para el uso de HttpClient podriamos decir que es muy versatil, ya que puede recibir varios tipo de cuerpo en las solicitudes, desde JSON hasta Streams, tambien se puede configurar de manera sincrona y asincrona, maneja tambien autenticacion en dado caso que se quira implementar entre otras cosas, como ejemplo de un GET seria asi:

HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder().

Uri(URI.create(“http://localhost”)).

build();

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

1. **¿Qué mejoras importantes introdujo Java 17 respecto a Java 11 en términos de rendimiento y nuevas características?**
   * Menciona al menos tres mejoras clave que afecten el rendimiento o la experiencia de desarrollo.
   * Para la actualización de Java 17 lo que cambio fue mas que todo el performance optimizado, dandonos mas rendimiento, tambien introdujo nuevas caracteristicas como records, las clases sealed y el pattern Matching
2. **En Java, ¿cuál es la diferencia entre HashMap y ConcurrentHashMap?**
   * ¿En qué situaciones utilizarías cada uno de ellos? Explica cómo manejan la concurrencia.
   * El Hashmap por lo general no se usa en hilos ya uqe no es segura su persistencia de datos, y el ConcurrentHashMap divide la información en segmentos lo cual permite concurrencia parcial y da mucho mejor performance
3. **¿Cómo se implementan y gestionan los Streams en Java 11 y 17?**
   * 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.
   * Un Stream es un conjunto de elementos que permite hacer operaciones sobre datos más que todo, las diferencias mas notables de la version 11 a 17 es que se intridujeron nuevas funciones como toArray() que se encarga de convertirlo el Stream en un Array de manera sencilla, tambien takeWhile o dropWhile que toma o descarta un elemento si cumple o no una condición entre otros.
4. **Explica cómo funciona el sistema de módulos en Java 9 y su evolución hasta Java 17.**
   * ¿Cómo se maneja el acceso entre módulos? ¿Cómo afecta esto a la estructura de aplicaciones grandes y monolíticas?
5. **¿Qué es un Pattern Matching en Java 17 y cómo mejora la legibilidad del código?**
   * Describe el concepto y la sintaxis de Pattern Matching en Java 17 y cómo ayuda a reducir el código repetitivo.

**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.

1. **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).
2. **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.
3. **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.

1. **Crea una clase Prestamo** con los siguientes atributos:
   * id (String), monto (double), clienteId (String), fecha (LocalDate), estado (enum con valores PENDIENTE, PAGADO).
2. **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.
3. **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.