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

**Nombre: Eric Eduardo Hernandez Camargo**

**Fecha: 29/05/2025**

**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.

R: con var el compilador infiere automáticamente el tipo de variable según el valor que se le asigne, mientras que con el tipo explicito uno debe indicarlo manualmente y var solo se puede usar en variables locales, no en atributos de clase ni en parámetros de metodos

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?

R: con Record el compilador genera automáticamente setters, getters, hashcode,etc y reduce el código, por defecto es inmutable y una class requiere que todo se escriba manualmente y es mutable.

Y normalmente se usa cuando quiere objetos inmutables, buscar reducir código y cuando modelas respuestas de servicios.

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.
2. **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?**

R: Se crea una instancia de HttpClient, luego se construye un HttpRequest con su url y se envia su método send, obteniendo una respuesta HttpResponse

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.

R: se optimizo el uso de memoria y la eficiencia del garbagecollectors, mejora en Record, switch con patrones pattern matching mejora switch

1. **En Java, ¿cuál es la diferencia entre HashMap y ConcurrentHashMap?**
   * ¿En qué situaciones utilizarías cada uno de ellos? Explica cómo manejan la concurrencia.

R: HashMap funciona para un solo hilo, y concurrentHashMap para múltiples hilos

1. **¿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.

R: permiten procesar colecciones de datos de forma declarativa, facilita el filtrado, el orden, la reducción y todo de manera eficiente y se hace con collect, forEach.

1. **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?

R: Organiza el código en unidades, que agrupan paquetes relacionados, cada modulo define que paquetes exporta para que otros módulos puedan acceder a ellos y que otros módulos necesita para funcionar

1. **¿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.

R: Es una característica que permite comprobar y extraer información de objetos de forma mas sencilla y clara dentro de expresiones condicionales como el operador instanceof, en lugar de hacer un cheque seguido de un casteo explicito hace ambas cosas en una 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.

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.
* Implementación de logs